目录

[175. Combine Two Tables 4](#_Toc57721060)

[176. Second Highest Salary 6](#_Toc57721061)

[177. Nth Highest Salary 8](#_Toc57721062)

[178. Rank Scores 10](#_Toc57721063)

[180. Consecutive Numbers 12](#_Toc57721064)

[181. Employees Earning More Than Their Managers 14](#_Toc57721065)

[182. Duplicate Emails 16](#_Toc57721066)

[183. Customers Who Never Order 18](#_Toc57721067)

[184. Department Highest Salary 20](#_Toc57721068)

[185. Department Top Three Salaries 22](#_Toc57721069)

[196. Delete Duplicate Emails 25](#_Toc57721070)

[197. Rising Temperature 27](#_Toc57721071)

[262. Trips and Users 29](#_Toc57721072)

[511. Game Play Analysis I 32](#_Toc57721073)

[512. Game Play Analysis II 34](#_Toc57721074)

[534. Game Play Analysis III 36](#_Toc57721075)

[550. Game Play Analysis IV 39](#_Toc57721076)

[569. Median Employee Salary★★ 42](#_Toc57721077)

[570. Managers with at Least 5 Direct Reports 44](#_Toc57721078)

[571. Find Median Given Frequency of Numbers★★★ 46](#_Toc57721079)

[574. Winning Candidate 48](#_Toc57721080)

[577. Employee Bonus 51](#_Toc57721081)

[578. Get Highest Answer Rate Question 53](#_Toc57721082)

[579. Find Cumulative Salary of an Employee★ 55](#_Toc57721083)

[580. Count Student Number in Departments 58](#_Toc57721084)

[584. Find Customer Referee 61](#_Toc57721085)

[585. Investments in 2016 63](#_Toc57721086)

[586. Customer Placing the Largest Number of Orders 66](#_Toc57721087)

[595. Big Countries 68](#_Toc57721088)

[596. Classes More Than 5 Students 70](#_Toc57721089)

[597. Friend Requests I: Overall Acceptance Rate 72](#_Toc57721090)

[601. Human Traffic of Stadium★★ 74](#_Toc57721091)

[602. Friend Requests II: Who Has the Most Friends 76](#_Toc57721092)

[603. Consecutive Available Seats 78](#_Toc57721093)

[607. Sales Person 80](#_Toc57721094)

[608. Tree Node 83](#_Toc57721095)

[610. Triangle Judgement 86](#_Toc57721096)

[612. Shortest Distance in a Plane 87](#_Toc57721097)

[613. Shortest Distance in a Line 89](#_Toc57721098)

[614. Second Degree Follower 90](#_Toc57721099)

[615. Average Salary: Departments VS Company 92](#_Toc57721100)

[618. Students Report By Geography★★★★ 95](#_Toc57721101)

[619. Biggest Single Number 97](#_Toc57721102)

[620. Not Boring Movies 99](#_Toc57721103)

[626. Exchange Seats 101](#_Toc57721104)

[627. Swap Salary 104](#_Toc57721105)

[1045. Customers Who Bought All Products 106](#_Toc57721106)

[1050. Actors and Directors Who Cooperated At Least Three Times 109](#_Toc57721107)

[1068. Product Sales Analysis I 111](#_Toc57721108)

[1069. Product Sales Analysis II 114](#_Toc57721109)

[1070. Product Sales Analysis III 117](#_Toc57721110)

[1075. Project Employees I 120](#_Toc57721111)

[1076. Project Employees II 123](#_Toc57721112)

[1077. Project Employees III 126](#_Toc57721113)

[1082. Sales Analysis I 129](#_Toc57721114)

[1083. Sales Analysis II 132](#_Toc57721115)

[1084. Sales Analysis III 135](#_Toc57721116)

[1097. Game Play Analysis V 138](#_Toc57721117)

[1098. Unpopular Books 141](#_Toc57721118)

[1107. New Users Daily Count 144](#_Toc57721119)

[1112. Highest Grade For Each Student 147](#_Toc57721120)

[1113. Reported Posts 150](#_Toc57721121)

[1126. Active Businesses 153](#_Toc57721122)

[1127. User Purchase Platform★ 156](#_Toc57721123)

[1132. Reported Posts II 159](#_Toc57721124)

[1141. User Activity for the Past 30 Days I 163](#_Toc57721125)

[1142. User Activity for the Past 30 Days II 165](#_Toc57721126)

[1148. Article Views I 168](#_Toc57721127)

[1149. Article Views II 170](#_Toc57721128)

[1158. Market Analysis I 172](#_Toc57721129)

[1159. Market Analysis II★ 176](#_Toc57721130)

[1164. Product Price at a Given Date 180](#_Toc57721131)

[1173. Immediate Food Delivery I 183](#_Toc57721132)

[1174. Immediate Food Delivery II 185](#_Toc57721133)

[1179. Reformat Department Table★★★ 188](#_Toc57721134)

[1193. Monthly Transactions I★ 191](#_Toc57721135)

[1194. Tournament Winners 194](#_Toc57721136)

[1204. Last Person to Fit in the Elevator 198](#_Toc57721137)

[1205. Monthly Transactions II 201](#_Toc57721138)

[1211. Queries Quality and Percentage 204](#_Toc57721139)

[1212. Team Scores in Football Tournament 207](#_Toc57721140)

[1225. Report Contiguous Dates★★★ 211](#_Toc57721141)

[1241. Number of Comments per Post 214](#_Toc57721142)

[1251. Average Selling Price 217](#_Toc57721143)

[1264. Page Recommendations 220](#_Toc57721144)

[1270. All People Report to the Given Manager 224](#_Toc57721145)

[1280. Students and Examinations 227](#_Toc57721146)

[1285. Find the Start and End Number of Continuous Ranges★ 232](#_Toc57721147)

[1294. Weather Type in Each Country 235](#_Toc57721148)

[1303. Find the Team Size 239](#_Toc57721149)

[1308. Running Total for Different Genders 241](#_Toc57721150)

[1321. Restaurant Growth 244](#_Toc57721151)

[1322. Ads Performance 248](#_Toc57721152)

[1327. List the Products Ordered in a Period 251](#_Toc57721153)

[1336. Number of Transactions per Visit●● 255](#_Toc57721154)

[1341. Movie Rating 261](#_Toc57721155)

[1350. Students With Invalid Departments 265](#_Toc57721156)

[1355. Activity Participants 268](#_Toc57721157)

[1364. Number of Trusted Contacts of a Customer 271](#_Toc57721158)

[1369. Get the Second Most Recent Activity 276](#_Toc57721159)

[1378. Replace Employee ID With The Unique Identifier 278](#_Toc57721160)

[1384. Total Sales Amount by Year●● 281](#_Toc57721161)

[1393. Capital Gain/Loss 286](#_Toc57721162)

[1398. Customers Who Bought Products A and B but Not C 289](#_Toc57721163)

[1407. Top Travellers 293](#_Toc57721164)

[1412. Find the Quiet Students in All Exams★★ 297](#_Toc57721165)

[1421. NPV Queries 301](#_Toc57721166)

[1435. Create a Session Bar Chart 304](#_Toc57721167)

[1440. Evaluate Boolean Expression 307](#_Toc57721168)

[1445. Apples & Oranges 311](#_Toc57721169)

[1454. Active Users★ 314](#_Toc57721170)

[1459. Rectangles Area 318](#_Toc57721171)

[1468. Calculate Salaries★ 321](#_Toc57721172)

[1479. Sales by Day of the Week★★ 324](#_Toc57721173)

[1485. Group Sold Products By The Date★ 329](#_Toc57721174)

[1495. Friendly Movies Streamed Last Month 332](#_Toc57721175)

[1501. Countries You Can Safely Invest In 335](#_Toc57721176)

[1511. Customer Order Frequency 339](#_Toc57721177)

[1517. Find Users With Valid E-Mails★ 343](#_Toc57721178)

[1527. Patients With a Condition 345](#_Toc57721179)

[1532. The Most Recent Three Orders 347](#_Toc57721180)

[1543. Fix Product Name Format 351](#_Toc57721181)

[1549. The Most Recent Orders for Each Product 354](#_Toc57721182)

[1555. Bank Account Summary 359](#_Toc57721183)

[1565. Unique Orders and Customers Per Month 363](#_Toc57721184)

[1571. Warehouse Manager 366](#_Toc57721185)

### [175. Combine Two Tables](https://leetcode-cn.com/problems/combine-two-tables/)

Table: Person

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

| Column Name | Type |

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

| PersonId | int |

| FirstName | varchar |

| LastName | varchar |

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

PersonId is the primary key column for this table.

Table: Address

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

| Column Name | Type |

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

| AddressId | int |

| PersonId | int |

| City | varchar |

| State | varchar |

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

AddressId is the primary key column for this table.

Write a SQL query for a report that provides the following information for each person in the Person table, regardless if there is an address for each of those people:

FirstName, LastName, City, State

SELECT FirstName, LastName, City, State

FROM Person p

LEFT JOIN Address a

ON p.PersonId = a.PersonId;

### [176. Second Highest Salary](https://leetcode-cn.com/problems/second-highest-salary/)

Write a SQL query to get the second highest salary from the Employee table.

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

| Id | Salary |

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

| 1 | 100 |

| 2 | 200 |

| 3 | 300 |

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

For example, given the above Employee table, the query should return 200 as the second highest salary. If there is no second highest salary, then the query should return null.

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

| SecondHighestSalary |

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

| 200 |

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

SELECT

IFNULL(

    (SELECT DISTINCT Salary

     FROM Employee

     ORDER BY Salary DESC

     LIMIT 1 OFFSET 1), NULL)

AS SecondHighestSalary

SELECT

    (SELECT DISTINCT Salary

     FROM Employee

    ORDER BY Salary DESC

    LiMIT 1, 1)

AS SecondHighestSalary

### [177. Nth Highest Salary](https://leetcode-cn.com/problems/nth-highest-salary/)

Write a SQL query to get the nth highest salary from the Employee table.

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

| Id | Salary |

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

| 1 | 100 |

| 2 | 200 |

| 3 | 300 |

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

For example, given the above Employee table, the nth highest salary where n = 2 is 200. If there is no nth highest salary, then the query should return null.

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

| getNthHighestSalary(2) |

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

| 200 |

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

CREATE FUNCTION getNthHighestSalary(N INT)RETURNS INT

BEGIN

  DECLARE M INT;

  SET M = N-1;

  RETURN (

      # Write your MySQL query statement below.

      SELECT DISTINCT Salary

      FROM Employee

      ORDER BY Salary DESC

      LIMIT M, 1

  );

END

### [178. Rank Scores](https://leetcode-cn.com/problems/rank-scores/)

Write a SQL query to rank scores. If there is a tie between two scores, both should have the same ranking. Note that after a tie, the next ranking number should be the next consecutive integer value. In other words, there should be no "holes" between ranks.

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

| Id | Score |

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

| 1 | 3.50 |

| 2 | 3.65 |

| 3 | 4.00 |

| 4 | 3.85 |

| 5 | 4.00 |

| 6 | 3.65 |

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

For example, given the above Scores table, your query should generate the following report (order by highest score):

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

| score | Rank |

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

| 4.00 | 1 |

| 4.00 | 1 |

| 3.85 | 2 |

| 3.65 | 3 |

| 3.65 | 3 |

| 3.50 | 4 |

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

**Important Note:** For MySQL solutions, to escape reserved words used as column names, you can use an apostrophe before and after the keyword. For example**`Rank`**.

 select S.Score, count(\*) 'Rank'

 from Scores S

 join (select distinct Score FROM Scores) S2

on S.Score <= S2.Score

 group by S.Id

 order by S.Score desc;

select Score,(select count(distinct Score) from Scores where Score >= S.Score) 'Rank'

from Scores S

order by Score desc;

### [180. Consecutive Numbers](https://leetcode-cn.com/problems/consecutive-numbers/)

Write a SQL query to find all numbers that appear at least three times consecutively.

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

| Id | Num |

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

| 1 | 1 |

| 2 | 1 |

| 3 | 1 |

| 4 | 2 |

| 5 | 1 |

| 6 | 2 |

| 7 | 2 |

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

For example, given the above Logs table, 1 is the only number that appears consecutively for at least three times.

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

| ConsecutiveNums |

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

| 1 |

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

select distinct l1.num as 'ConsecutiveNums'

from Logs l1

join Logs l2 on l1.Id = l2.Id - 1 and l1.Num = l2.Num

join Logs l3 on l2.Id = l3.Id - 1 and l2.Num = l3.Num;

### [181. Employees Earning More Than Their Managers](https://leetcode-cn.com/problems/employees-earning-more-than-their-managers/)

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

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

| Id | Name | Salary | ManagerId |

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

| 1 | Joe | 70000 | 3 |

| 2 | Henry | 80000 | 4 |

| 3 | Sam | 60000 | NULL |

| 4 | Max | 90000 | NULL |

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

Given the Employee table, write a SQL query that finds out employees who earn more than their managers. For the above table, Joe is the only employee who earns more than his manager.

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

| Employee |

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

| Joe |

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

select a.Name as 'Employee'

from Employee a

join  Employee b on a.ManagerId = b.Id and a.Salary > b.Salary;

### [182. Duplicate Emails](https://leetcode-cn.com/problems/duplicate-emails/)

Write a SQL query to find all duplicate emails in a table named Person.

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

| Id | Email |

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

| 1 | a@b.com |

| 2 | c@d.com |

| 3 | a@b.com |

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

For example, your query should return the following for the above table:

+---------+

| Email |

+---------+

| a@b.com |

+---------+

**Note**: All emails are in lowercase.

select Email

from Person

group by Email

having count(Email) > 1;

### [183. Customers Who Never Order](https://leetcode-cn.com/problems/customers-who-never-order/)

Suppose that a website contains two tables, the Customers table and the Orders table. Write a SQL query to find all customers who never order anything.

Table: Customers.

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

| Id | Name |

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

| 1 | Joe |

| 2 | Henry |

| 3 | Sam |

| 4 | Max |

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

Table: Orders.

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

| Id | CustomerId |

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

| 1 | 3 |

| 2 | 1 |

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

Using the above tables as example, return the following:

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

| Customers |

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

| Henry |

| Max |

select Name as 'Customers'

from Customers

left join Orders on Customers.Id = Orders.CustomerId

where CustomerId is null

### [184. Department Highest Salary](https://leetcode-cn.com/problems/department-highest-salary/)

The Employee table holds all employees. Every employee has an Id, a salary, and there is also a column for the department Id.

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

| Id | Name | Salary | DepartmentId |

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

| 1 | Joe | 70000 | 1 |

| 2  | Jim   | 90000  | 1            |

| 3 | Henry | 80000 | 2 |

| 4 | Sam | 60000 | 2 |

| 5 | Max | 90000 | 1 |

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

The Department table holds all departments of the company.

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

| Id | Name |

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

| 1 | IT |

| 2 | Sales |

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

Write a SQL query to find employees who have the highest salary in each of the departments. For the above tables, your SQL query should return the following rows (order of rows does not matter).

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

| Department | Employee | Salary |

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

| IT | Max | 90000 |

| IT         | Jim      | 90000  |

| Sales | Henry | 80000 |

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

**Explanation:**

Max and Jim both have the highest salary in the IT department and Henry has the highest salary in the Sales department.

select Department.Name as 'Department', Employee.Name Employee, Salary

from Employee

Join Department on Employee.DepartmentId = Department.Id

where (Employee.DepartmentId, Salary) IN

    (

        SELECT DepartmentId, MAX(Salary)

        FROM Employee

        GROUP BY DepartmentId

    )

### [185. Department Top Three Salaries](https://leetcode-cn.com/problems/department-top-three-salaries/)

The Employee table holds all employees. Every employee has an Id, and there is also a column for the department Id.

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

| Id | Name | Salary | DepartmentId |

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

| 1 | Joe | 85000 | 1 |

| 2 | Henry | 80000 | 2 |

| 3 | Sam | 60000 | 2 |

| 4 | Max | 90000 | 1 |

| 5 | Janet | 69000 | 1 |

| 6 | Randy | 85000 | 1 |

| 7 | Will | 70000 | 1 |

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

The Department table holds all departments of the company.

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

| Id | Name |

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

| 1 | IT |

| 2 | Sales |

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

Write a SQL query to find employees who earn the top three salaries in each of the department. For the above tables, your SQL query should return the following rows (order of rows does not matter).

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

| Department | Employee | Salary |

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

| IT | Max | 90000 |

| IT | Randy | 85000 |

| IT | Joe | 85000 |

| IT | Will | 70000 |

| Sales | Henry | 80000 |

| Sales | Sam | 60000 |

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

**Explanation:**

In IT department, Max earns the highest salary, both Randy and Joe earn the second highest salary, and Will earns the third highest salary. There are only two employees in the Sales department, Henry earns the highest salary while Sam earns the second highest salary.

select d.Name Department, e1.Name Employee, e1.Salary

from Employee e1

join Department d on e1.DepartmentId = d.Id

where 3 > (

select count(distinct(e2.Salary))

           from Employee e2

           where e2.Salary > e1.Salary

                 and e1.DepartmentId = e2.DepartmentId

)

select d.Name Department, e1.Name Employee, e1.Salary

from Employee e1

join Department d on e1.DepartmentId = d.Id

join Employee e2 on e2.Salary >= e1.Salary and e1.DepartmentId = e2.DepartmentId

group by e1.Id

having count(distinct e2.Salary) <= 3

### [196. Delete Duplicate Emails](https://leetcode-cn.com/problems/delete-duplicate-emails/)

Write a SQL query to **delete** all duplicate email entries in a table named Person, keeping only unique emails based on its *smallest* **Id**.

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

| Id | Email |

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

| 1 | john@example.com |

| 2 | bob@example.com |

| 3 | john@example.com |

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

Id is the primary key column for this table.

For example, after running your query, the above Person table should have the following rows:

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

| Id | Email |

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

| 1 | john@example.com |

| 2 | bob@example.com |

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

**Note:**

Your output is the whole Person table after executing your sql. Use delete statement.

/\*

delete p1 from Person p1, Person p2

where p1.Email = p2.Email and p1.Id > p2.Id

\*/

delete from Person where Id not in (

    select id from (

        select min(Id) as id

from Person

        group by Email

    ) as \_      # it needs an alias for set

)

### [197. Rising Temperature](https://leetcode-cn.com/problems/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 in a certain day.

Write an SQL query to find all dates' id with higher temperature compared to its previous dates (yesterday).

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

The query result format is in the following example:

Weather

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

| id | recordDate | Temperature |

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

| 1 | 2015-01-01 | 10 |

| 2 | 2015-01-02 | 25 |

| 3 | 2015-01-03 | 20 |

| 4 | 2015-01-04 | 30 |

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

Result table:

+----+

| id |

+----+

| 2 |

| 4 |

+----+

In 2015-01-02, temperature was higher than the previous day (10 -> 25).

In 2015-01-04, temperature was higher than the previous day (30 -> 20).

select a.Id

from Weather a

join Weather b

on a.Temperature > b.Temperature

and datediff(a.RecordDate, b.RecordDate) = 1

### [262. Trips and Users](https://leetcode-cn.com/problems/trips-and-users/)

The Trips table holds all taxi trips. Each trip has a unique Id, while Client\_Id and Driver\_Id are both foreign keys to the Users\_Id at the Users table. Status is an ENUM type of (‘completed’, ‘cancelled\_by\_driver’, ‘cancelled\_by\_client’).

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

| Id | Client\_Id | Driver\_Id | City\_Id | Status |Request\_at|

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

| 1 | 1 | 10 | 1 | completed |2013-10-01|

| 2 | 2 | 11 | 1 | cancelled\_by\_driver|2013-10-01|

| 3 | 3 | 12 | 6 | completed |2013-10-01|

| 4 | 4 | 13 | 6 | cancelled\_by\_client|2013-10-01|

| 5 | 1 | 10 | 1 | completed |2013-10-02|

| 6 | 2 | 11 | 6 | completed |2013-10-02|

| 7 | 3 | 12 | 6 | completed |2013-10-02|

| 8 | 2 | 12 | 12 | completed |2013-10-03|

| 9 | 3 | 10 | 12 | completed |2013-10-03|

| 10 | 4 | 13 | 12 | cancelled\_by\_driver|2013-10-03|

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

The Users table holds all users. Each user has an unique Users\_Id, and Role is an ENUM type of (‘client’, ‘driver’, ‘partner’).

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

| Users\_Id | Banned | Role |

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

| 1 | No | client |

| 2 | Yes | client |

| 3 | No | client |

| 4 | No | client |

| 10 | No | driver |

| 11 | No | driver |

| 12 | No | driver |

| 13 | No | driver |

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

Write a SQL query to find the cancellation rate of requests made by unbanned users (both client and driver must be unbanned) between **Oct 1, 2013** and **Oct 3, 2013**. The cancellation rate is computed by dividing the number of canceled (by client or driver) requests made by unbanned users by the total number of requests made by unbanned users.

For the above tables, your SQL query should return the following rows with the cancellation rate being rounded to two decimal places.

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

| Day | Cancellation Rate |

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

| 2013-10-01 | 0.33 |

| 2013-10-02 | 0.00 |

| 2013-10-03 | 0.50 |

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

select Request\_at Day, Round(sum(Status like 'can%') / count(Client\_Id)

                             , 2) as 'Cancellation Rate'

from Trips

where Request\_at between '2013-10-01' and '2013-10-03'

      and Client\_Id not in (

                                select Users\_id

                                from Users

                                where Role = 'client' and Banned ='Yes'

                           )

group by Request\_at

### [511. Game Play Analysis I](https://leetcode-cn.com/problems/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 game.

Each row is a record of a player who logged in and played a number of games (possibly 0) before logging out on some day using some device.

Write an SQL query that reports the **first login date** for each player.

The query result format is in the following example:

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 |

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

Result table:

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

| player\_id | first\_login |

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

| 1 | 2016-03-01 |

| 2 | 2017-06-25 |

| 3 | 2016-03-02 |

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

select player\_id, min(event\_date) as 'first\_login'

from Activity

group by player\_id

### [512. Game Play Analysis II](https://leetcode-cn.com/problems/game-play-analysis-ii/)

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 game.

Each row is a record of a player who logged in and played a number of games (possibly 0) before logging out on some day using some device.

Write a SQL query that reports the **device** that is first logged in for each player.

The query result format is in the following example:

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 |

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

Result table:

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

| player\_id | device\_id |

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

| 1 | 2 |

| 2 | 3 |

| 3 | 1 |

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

select player\_id, device\_id

from Activity

where (player\_id, event\_date) in (

select player\_id, min(event\_date)

                                 from Activity

                                 group by player\_id

)

### [534. Game Play Analysis III](https://leetcode-cn.com/problems/game-play-analysis-iii/)

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 game.

Each row is a record of a player who logged in and played a number of games (possibly 0) before logging out on some day using some device.

Write an SQL query that reports for each player and date, how many games played **so far** by the player. That is, the total number of games played by the player until that date. Check the example for clarity.

The query result format is in the following example:

Activity table:

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

| player\_id | device\_id | event\_date | games\_played |

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

| 1 | 2 | 2016-03-01 | 5 |

| 1 | 2 | 2016-05-02 | 6 |

| 1 | 3 | 2017-06-25 | 1 |

| 3 | 1 | 2016-03-02 | 0 |

| 3 | 4 | 2018-07-03 | 5 |

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

Result table:

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

| player\_id | event\_date | games\_played\_so\_far |

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

| 1 | 2016-03-01 | 5 |

| 1 | 2016-05-02 | 11 |

| 1 | 2017-06-25 | 12 |

| 3 | 2016-03-02 | 0 |

| 3 | 2018-07-03 | 5 |

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

For the player with id 1, 5 + 6 = 11 games played by 2016-05-02, and 5 + 6 + 1 = 12 games played by 2017-06-25.

For the player with id 3, 0 + 5 = 5 games played by 2018-07-03.

Note that for each player we only care about the days when the player logged in.

select a.player\_id, a.event\_date, sum(b.games\_played) as 'games\_played\_so\_far'

from Activity a

join Activity b on a.player\_id = b.player\_id

and a.event\_date >= b.event\_date

group by a.player\_id, a.event\_date;

select player\_id, event\_date, sum(games\_played) over(partition by player\_id order by event\_date) as 'games\_played\_so\_far'

from  activity;

### [550. Game Play Analysis IV](https://leetcode-cn.com/problems/game-play-analysis-iv/)

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 game.

Each row is a record of a player who logged in and played a number of games (possibly 0) before logging out on some day using some device.

Write an SQL query that reports the **fraction** of players that logged in again on the day after the day they first logged in, **rounded to 2 decimal places**. In other words, you need to count the number of players that logged in for at least two consecutive days starting from their first login date, then divide that number by the total number of players.

The query result format is in the following example:

Activity table:

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

| player\_id | device\_id | event\_date | games\_played |

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

| 1 | 2 | 2016-03-01 | 5 |

| 1 | 2 | 2016-03-02 | 6 |

| 2 | 3 | 2017-06-25 | 1 |

| 3 | 1 | 2016-03-02 | 0 |

| 3 | 4 | 2018-07-03 | 5 |

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

Result table:

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

| fraction |

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

| 0.33 |

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

Only the player with id 1 logged back in after the first day he had logged in so the answer is 1/3 = 0.33

select

    ROUND((select count(\*)

        from Activity

        where (player\_id, event\_date) in

(

select player\_id, Date(min(event\_date)+1)

                          from Activity

                          group by player\_id

)

    ) / count(distinct player\_id), 2) as fraction

from Activity;

### [569. Median Employee Salary](https://leetcode-cn.com/problems/median-employee-salary/)★★

The Employee table holds all employees. The employee table has three columns: Employee Id, Company Name, and Salary.

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

|Id | Company | Salary |

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

|1 | A | 2341 |

|2 | A | 341 |

|3 | A | 15 |

|4 | A | 15314 |

|5 | A | 451 |

|6 | A | 513 |

|7 | B | 15 |

|8 | B | 13 |

|9 | B | 1154 |

|10 | B | 1345 |

|11 | B | 1221 |

|12 | B | 234 |

|13 | C | 2345 |

|14 | C | 2645 |

|15 | C | 2645 |

|16 | C | 2652 |

|17 | C | 65 |

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

Write a SQL query to find the median salary of each company. Bonus points if you can solve it without using any built-in SQL functions.

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

|Id | Company | Salary |

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

|5 | A | 451 |

|6 | A | 513 |

|12 | B | 234 |

|9 | B | 1154 |

|14 | C | 2645 |

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

select Id, Company, Salary

from (

    select Id, Company, Salary,

        row\_number() over(partition by Company order by Salary) as rnk,

        count(\*) over(partition by Company) as cnt

    from Employee ) t

where rnk in (cnt/2, cnt/2+1, cnt/2+0.5)

### [570. Managers with at Least 5 Direct Reports](https://leetcode-cn.com/problems/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.

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

|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:

+-------+

| Name |

+-------+

| John |

+-------+

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

select a.Name as 'Name'

from Employee a

join Employee b on b.ManagerId = a.Id

group by a.Id

having count(\*) > 4

### [571. Find Median Given Frequency of Numbers](https://leetcode-cn.com/problems/find-median-given-frequency-of-numbers/)★★★

The Numbers table keeps the value of number and its frequency.

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

| Number | Frequency |

+----------+-------------|

| 0 | 7 |

| 1 | 1 |

| 2 | 3 |

| 3 | 1 |

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

In this table, the numbers are 0, 0, 0, 0, 0, 0, 0, 1, 2, 2, 2, 3, so the median is (0 + 0) / 2 = 0.

+--------+

| median |

+--------|

| 0.0000 |

+--------+

Write a query to find the median of all numbers and name the result as median.

select avg(Number) as median from

(

    select Number, @c1 + 1 as 'c1',(@c1 := @c1 + Frequency) 'c2', t2.s

    from Numbers

    join (select @c1 := 0) t1

    join (select sum(Frequency) as s from Numbers) t2

    order by Number

) tmp

where c1 <= s/2 + 1 and c2 >= s/2;

### [574. Winning Candidate](https://leetcode-cn.com/problems/winning-candidate/)

Table: Candidate

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

| id | Name |

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

| 1 | A |

| 2 | B |

| 3 | C |

| 4 | D |

| 5 | E |

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

Table: Vote

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

| 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.

+------+

| Name |

+------+

| B |

+------+

**Notes:**

1. You may assume **there is no tie**, in other words there will be **only one** winning candidate.

select a.Name

from Candidate a

join Vote b on a.id = b.CandidateId

group by a.id

order by count(\*) desc

limit 0, 1

select Name

from Candidate

where id = (

                select CandidateId

                from Vote

                group by CandidateId

                order by count(\*) desc

                limit 1

            )

### [577. Employee Bonus](https://leetcode-cn.com/problems/employee-bonus/)

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

Table:Employee

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

| 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

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

| empId | bonus |

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

| 2 | 500 |

| 4 | 2000 |

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

empId is the primary key column for this table.

Example ouput:

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

| name | bonus |

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

| John | null |

| Dan | 500 |

| Brad | null |

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

select name, bonus

from Employee a

left join bonus b on a.empId = b.empId

where ifnull(bonus, 0) < 1000;

### [578. Get Highest Answer Rate Question](https://leetcode-cn.com/problems/get-highest-answer-rate-question/)

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

id 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:**

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

| id | 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:**

| 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.

#count 不计null行

select question\_id as 'survey\_log'

from survey\_log

where action <> 'skip'

group by question\_id

order by count(answer\_id) / (count(\*) - count(answer\_id)) desc

limit 1

#sum里面可以加函数

select question\_id as survey\_log

from survey\_log

group by question\_id

order by sum(if(action = 'answer', 1, 0)) / sum(if(action = 'show', 1, 0)) desc

limit 1

### [579. Find Cumulative Salary of an Employee](https://leetcode-cn.com/problems/find-cumulative-salary-of-an-employee/)★

The **Employee** table holds the salary information in a year.

Write a SQL to get the cumulative sum of an employee's salary over a period of 3 months but exclude the most recent month.

The result should be displayed by 'Id' ascending, and then by 'Month' descending.

**Example**  
**Input**

| Id | Month | Salary |

|----|-------|--------|

| 1 | 1 | 20 |

| 2 | 1 | 20 |

| 1 | 2 | 30 |

| 2 | 2 | 30 |

| 3 | 2 | 40 |

| 1 | 3 | 40 |

| 3 | 3 | 60 |

| 1 | 4 | 60 |

| 3 | 4 | 70 |

**Output**

| Id | Month | Salary |

|----|-------|--------|

| 1 | 3 | 90 |

| 1 | 2 | 50 |

| 1 | 1 | 20 |

| 2 | 1 | 20 |

| 3 | 3 | 100 |

| 3 | 2 | 40 |

**Explanation**

Employee '1' has 3 salary records for the following 3 months except the most recent month '4': salary 40 for month '3', 30 for month '2' and 20 for month '1'  
So the cumulative sum of salary of this employee over 3 months is 90(40+30+20), 50(30+20) and 20 respectively.

| Id | Month | Salary |

|----|-------|--------|

| 1 | 3 | 90 |

| 1 | 2 | 50 |

| 1 | 1 | 20 |

Employee '2' only has one salary record (month '1') except its most recent month '2'.

| Id | Month | Salary |

|----|-------|--------|

| 2 | 1 | 20 |

Employ '3' has two salary records except its most recent pay month '4': month '3' with 60 and month '2' with 40. So the cumulative salary is as following.

| Id | Month | Salary |

|----|-------|--------|

| 3 | 3 | 100 |

| 3 | 2 | 40 |

select Id, Month,

sum(Salary) over (partition by Id order by Month rows 2 preceding) Salary

from Employee

where (Id, Month) not in (

                            select Id, max(Month)

                            from Employee

                            group by Id

                         )

order by Id, Month desc

### [580. Count Student Number in Departments](https://leetcode-cn.com/problems/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 dept\_name, count(student\_id) `student\_number`

from department a

left join student b on a.dept\_id = b.dept\_id

group by a.dept\_id

order by student\_number desc, dept\_name

### [584. Find Customer Referee](https://leetcode-cn.com/problems/find-customer-referee/)

Given a table customer holding customers information and the referee.

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

| 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:

+------+

| name |

+------+

| Will |

| Jane |

| Bill |

| Zack |

+------+

#对于null,referee\_id <> 2 和 !(referee\_id = 2)都不成立

select name

from customer

where ifnull(referee\_id, 0) <> 2

### [585. Investments in 2016](https://leetcode-cn.com/problems/investments-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 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.

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 concat(lat, lon) in (

select concat(lat,lon)

from insurance

group by concat(lat,lon)

having count(\*) = 1

)

SELECT SUM(TIV\_2016) as TIV\_2016

FROM (

        SELECT

            \*,

            count(\*) over(partition by TIV\_2015) as cnt\_1,

            count(\*) over(partition by LAT, LON) as cnt\_2

        FROM insurance

    ) a

WHERE a.cnt\_1 > 1 AND a.cnt\_2 < 2

### [586. Customer Placing the Largest Number of Orders](https://leetcode-cn.com/problems/customer-placing-the-largest-number-of-orders/)

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'.

***Follow up:****What if more than one customer have the largest number of orders, can you find all the customer\_number in this case?*

select customer\_number

from orders

group by customer\_number

order by count(\*) desc

limit 0, 1

### [595. Big Countries](https://leetcode-cn.com/problems/big-countries/)

There is a table World

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

| name | continent | area | population | gdp |

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

| Afghanistan | Asia | 652230 | 25500100 | 20343000 |

| Albania | Europe | 28748 | 2831741 | 12960000 |

| Algeria | Africa | 2381741 | 37100000 | 188681000 |

| Andorra | Europe | 468 | 78115 | 3712000 |

| Angola | Africa | 1246700 | 20609294 | 100990000 |

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

A country is big if it has an area of bigger than 3 million square km or a population of more than 25 million.

Write a SQL solution to output big countries' name, population and area.

For example, according to the above table, we should output:

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

| name | population | area |

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

| Afghanistan | 25500100 | 652230 |

| Algeria | 37100000 | 2381741 |

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

select name, population, area

from World

where area > 3000000 or population > 25000000

### [596. Classes More Than 5 Students](https://leetcode-cn.com/problems/classes-more-than-5-students/)

There is a table courses with columns: **student** and **class**

Please list out all classes which have more than or equal to 5 students.

For example, the table:

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

| student | class |

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

| A | Math |

| B | English |

| C | Math |

| D | Biology |

| E | Math |

| F | Computer |

| G | Math |

| H | Math |

| I | Math |

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

Should output:

+---------+

| class |

+---------+

| Math |

+---------+

**Note:**  
The students should not be counted duplicate in each course.

select class

from courses

group by class

having count(distinct student) >= 5;

### [597. Friend Requests I: Overall Acceptance Rate](https://leetcode-cn.com/problems/friend-requests-i-overall-acceptance-rate/)

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.

**Follow-up:**

* Can you write a query to return the accept rate but for every month?
* How about the cumulative accept rate for every day?

select

    round(

        ifnull(

            (select count(\*) from (select distinct requester\_id, accepter\_id from request\_accepted) as A)

            / (select count(\*) from (select distinct sender\_id, send\_to\_id from friend\_request) as B)

        , 0)

    , 2) as accept\_rate;

### [601. Human Traffic of Stadium](https://leetcode-cn.com/problems/human-traffic-of-stadium/)★★

X city built a new stadium, each day many people visit it and the stats are saved as these columns: **id**, **visit\_date**, **people**

Please write a query to display the records which have 3 or more consecutive rows and the amount of people more than 100(inclusive).

For example, the table stadium:

| id | visit\_date | people |

| 1 | 2017-01-01 | 10 |

| 2 | 2017-01-02 | 109 |

| 3 | 2017-01-03 | 150 |

| 4 | 2017-01-04 | 99 |

| 5 | 2017-01-05 | 145 |

| 6 | 2017-01-06 | 1455 |

| 7 | 2017-01-07 | 199 |

| 8 | 2017-01-08 | 188 |

For the sample data above, the output is:

| id | visit\_date | people |

| 5 | 2017-01-05 | 145 |

| 6 | 2017-01-06 | 1455 |

| 7 | 2017-01-07 | 199 |

| 8 | 2017-01-08 | 188 |

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

**Note:**  
Each day only have one row record, and the dates are increasing with id increasing.

select t.\*

from stadium t

left join stadium p1 on t.id - 1 = p1.id

left join stadium p2 on t.id - 2 = p2.id

left join stadium n1 on t.id + 1 = n1.id

left join stadium n2 on t.id + 2 = n2.id

where  (t.people >= 100 and p1.people >= 100 and p2.people >= 100)

    or (t.people >= 100 and n1.people >= 100 and n2.people >= 100)

    or (t.people >= 100 and n1.people >= 100 and p1.people >= 100)

order by id;

select s1.\*

from stadium as s1

join stadium as s2 on s2.people >= 100

join stadium as s3 on s3.people >= 100

where s1.people >= 100

      and ((s1.id + 1 = s2.id and s1.id + 2 = s3.id)

        or (s1.id - 1 = s2.id and s1.id + 1 = s3.id)

        or (s1.id - 2 = s2.id and s1.id - 1 = s3.id))

group by s1.id

order by s1.id

with t1 as (

                select id, visit\_date, people,

                       id-rank() over(order by id) rk

                from stadium

                where people >= 100

            )

select id, visit\_date, people

from t1

where rk in (

                select rk

                from t1

                group by rk

                having count(\*) >= 3

            );

### [602. Friend Requests II: Who Has the Most Friends](https://leetcode-cn.com/problems/friend-requests-ii-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

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

| 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 |

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

This table holds the data of friend acceptance, while **requester\_id** and **accepter\_id** both are the id of a person.

Write a query to find the the people who has most friends and the most friends number under the following rules:

* 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.

For the sample data above, the result is:

Result table:

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

| id | num |

|------|------|

| 3 | 3 |

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

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.

**Follow-up:**  
In the real world, multiple people could have the same most number of friends, can you find all these people in this case?

select id, count(\*) num

from (

        (

            select requester\_id id

            from request\_accepted

        )

        union all

        (

            select accepter\_id id

            from request\_accepted

        )

    ) t3

group by id

order by num desc

limit 1

### [603. Consecutive Available Seats](https://leetcode-cn.com/problems/consecutive-available-seats/)

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.

select distinct(c1.seat\_id)

from cinema c1

join cinema c2 on abs(c2.seat\_id-c1.seat\_id) = 1

where c1.free = 1 and c2.free = 1

order by c1.seat\_id

### [607. Sales Person](https://leetcode-cn.com/problems/sales-person/)

**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

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

| 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

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

| 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

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

| order\_id | order\_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**

+------+

| name |

+------+

| Amy |

| Mark |

| Alex |

+------+

**Explanation**

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

select name

from salesperson

where sales\_id not in (

                        select sales\_id

                        from orders a

                        join company b on a.com\_id = b.com\_id

                        where b.name = 'RED'

                    )

### [608. Tree Node](https://leetcode-cn.com/problems/tree-node/)

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

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

| 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:

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

| 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

**Note**

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

select id, case

               when p\_id is null then 'Root'

               when id in (select p\_id from tree)then 'Inner'

               else 'Leaf'

           end as Type

from tree

### [610. Triangle Judgement](https://leetcode-cn.com/problems/triangle-judgement/)

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 |

select \*, IF(x+y > z and x+z > y and y+z > x, 'Yes', 'No') 'triangle'

from triangle

### [612. Shortest Distance in a Plane](https://leetcode-cn.com/problems/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.

select round(min(sqrt(pow(a.x-b.x, 2)

+ pow(a.y-b.y, 2))), 2) as 'shortest'

from point\_2d a

join point\_2d b on !(a.x = b.x && a.y = b.y)

### [613. Shortest Distance in a Line](https://leetcode-cn.com/problems/shortest-distance-in-a-line/)

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.

**Follow-up:** What if all these points have an id and are arranged from the left most to the right most of x axis?

select min(abs(a.x-b.x)) as 'shortest'

from point a

join point b on a.x != b.x

### [614. Second Degree Follower](https://leetcode-cn.com/problems/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:

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

| followee | follower |

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

| A | B |

| B | C |

| B | D |

| D | E |

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

should output:

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

| 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.

select followee as 'follower', count(distinct follower) 'num'

from follow

where followee in (

                        select distinct follower

                        from follow

                  )

group by followee

order by follower

### [615. Average Salary: Departments VS Company](https://leetcode-cn.com/problems/average-salary-departments-vs-company/)

Given two tables as below, write a query to display the comparison result (higher/lower/same) of the average salary of employees in a department to the company's average salary.

Table: salary

| id | employee\_id | amount | pay\_date |

|----|-------------|--------|------------|

| 1 | 1 | 9000 | 2017-03-31 |

| 2 | 2 | 6000 | 2017-03-31 |

| 3 | 3 | 10000 | 2017-03-31 |

| 4 | 1 | 7000 | 2017-02-28 |

| 5 | 2 | 6000 | 2017-02-28 |

| 6 | 3 | 8000 | 2017-02-28 |

The **employee\_id** column refers to the **employee\_id** in the following table employee.

| employee\_id | department\_id |

|-------------|---------------|

| 1 | 1 |

| 2 | 2 |

| 3 | 2 |

So for the sample data above, the result is:

| pay\_month | department\_id | comparison |

|-----------|---------------|-------------|

| 2017-03 | 1 | higher |

| 2017-03 | 2 | lower |

| 2017-02 | 1 | same |

| 2017-02 | 2 | same |

**Explanation**

In March, the company's average salary is (9000+6000+10000)/3 = 8333.33...

The average salary for department '1' is 9000, which is the salary of **employee\_id** '1' since there is only one employee in this department. So the comparison result is 'higher' since 9000 > 8333.33 obviously.

The average salary of department '2' is (6000 + 10000)/2 = 8000, which is the average of **employee\_id** '2' and '3'. So the comparison result is 'lower' since 8000 < 8333.33.

With he same formula for the average salary comparison in February, the result is 'same' since both the department '1' and '2' have the same average salary with the company, which is 7000.

select distinct pay\_month, department\_id,  case

                                                when d\_avg > c\_avg then "higher"

                                                when d\_avg = c\_avg then "same"

                                                else "lower"

                                            end as comparison

from (

        select date\_format(pay\_date, "%Y-%m") as pay\_month, department\_id,

                avg(amount)over(partition by date\_format(pay\_date, "%Y-%m"),department\_id) as d\_avg,

                avg(amount)over(partition by date\_format(pay\_date, "%Y-%m")) as c\_avg

        from salary

        join employee on salary.employee\_id = employee.employee\_id

    ) t

select date\_format(pay\_date, "%Y-%m") pay\_month, department\_id, case

                                                    when abs(avg(amount) - avg\_m) < 1e-3 then 'same'

                                                    when avg(amount) < avg\_m then 'lower'

                                                    else 'higher'

                                                 end as comparison

from salary a

join employee b on a.employee\_id = b.employee\_id

join (

        select date\_format(pay\_date, "%Y-%m") pay\_month, avg(amount) avg\_m

        from salary

        group by date\_format(pay\_date, "%Y-%m")

    ) t on t.pay\_month = date\_format(pay\_date, "%Y-%m")

group by date\_format(pay\_date, "%Y-%m"), department\_id

### [618. Students Report By Geography](https://leetcode-cn.com/problems/students-report-by-geography/)★★★★

A U.S graduate school has students from Asia, Europe and America. The students' location information are stored in table student as below.

| name | continent |

|--------|-----------|

| Jack | America |

| Pascal | Europe |

| Xi | Asia |

| Jane | America |

[Pivot](https://en.wikipedia.org/wiki/Pivot_table) the continent column in this table so that each name is sorted alphabetically and displayed underneath its corresponding continent. The output headers should be America, Asia and Europe respectively. It is guaranteed that the student number from America is no less than either Asia or Europe.

For the sample input, the output is:

| America | Asia | Europe |

|---------|------|--------|

| Jack | Xi | Pascal |

| Jane | | |

**Follow-up:** If it is unknown which continent has the most students, can you write a query to generate the student report?

#聚合函数 遍历

select

    max(case when continent = 'America' then name else null end) America,

    min(case when continent = 'Asia' then name else null end) Asia,

    max(case when continent = 'Europe' then name else null end) Europe

from (

        select name, continent,

            row\_number()over(partition by continent order by name) cur\_rank

        from student

     ) t

group by cur\_rank

select America, Asia, Europe

from (

        select row\_number() over(order by name) id, name as America

        from student

        where continent = 'America'

     ) a

left join (

            select row\_number() over(order by name) id, name as Asia

            from student

            where continent = 'Asia'

          ) b on a.id = b.id

left join (

            select row\_number() over(order by name) id, name as Europe

            from student

            where continent = 'Europe'

          ) c on a.id = c.id

### [619. Biggest Single Number](https://leetcode-cn.com/problems/biggest-single-number/)

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

+---+

|num|

+---+

| 8 |

| 8 |

| 3 |

| 3 |

| 1 |

| 4 |

| 5 |

| 6 |

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

+---+

|num|

+---+

| 6 |

**Note:**  
If there is no such number, just output **null**.

select (

            select num

            from my\_numbers

            group by num

            having count(num) = 1

            order by num desc

            limit 1

        )  as num

### [620. Not Boring Movies](https://leetcode-cn.com/problems/not-boring-movies/)

X city opened a new cinema, many people would like to go to this cinema. The cinema also gives out a poster indicating the movies’ ratings and descriptions.

Please write a SQL query to output movies with an odd numbered ID and a description that is not 'boring'. Order the result by rating.

For example, table cinema:

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

| id | movie | description | rating |

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

| 1 | War | great 3D | 8.9 |

| 2 | Science | fiction | 8.5 |

| 3 | irish | boring | 6.2 |

| 4 | Ice song | Fantacy | 8.6 |

| 5 | House card| Interesting| 9.1 |

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

For the example above, the output should be:

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

| id | movie | description | rating |

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

| 5 | House card| Interesting| 9.1 |

| 1 | War | great 3D | 8.9 |

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

select \*

from cinema

where mod(id ,2) = 1 and description <> 'boring'

order by rating desc

### [626. Exchange Seats](https://leetcode-cn.com/problems/exchange-seats/)

Mary is a teacher in a middle school and she has a table seat storing students' names and their corresponding seat ids.

The column **id** is continuous increment.

Mary wants to change seats for the adjacent students.

Can you write a SQL query to output the result for Mary?

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

| id | student |

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

| 1 | Abbot |

| 2 | Doris |

| 3 | Emerson |

| 4 | Green |

| 5 | Jeames |

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

For the sample input, the output is:

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

| id | student |

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

| 1 | Doris |

| 2 | Abbot |

| 3 | Green |

| 4 | Emerson |

| 5 | Jeames |

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

**Note:**  
If the number of students is odd, there is no need to change the last one's seat.

select

    if(id < (select count(\*) from seat), if(mod(id, 2) = 0, id-1, id+1)

           , if(mod(id, 2) = 0, id-1, id)) as id, student

from seat

order by id asc;

select

    (case

        when mod(id, 2) != 0 and counts != id then id + 1

        when mod(id, 2) != 0 and counts = id then id

        else id - 1

     end) AS id, student

from seat, (select count(\*) as counts from seat) as \_

order by id asc;

### [627. Swap Salary](https://leetcode-cn.com/problems/swap-salary/)

Given a table salary, such as the one below, that has m=male and f=female values. 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 temp table.

Note that you must write a single update statement, **DO NOT** write any select statement for this problem.

**Example:**

| id | name | sex | salary |

|----|------|-----|--------|

| 1 | A | m | 2500 |

| 2 | B | f | 1500 |

| 3 | C | m | 5500 |

| 4 | D | f | 500 |

After running your **update** statement, the above salary table should have the following rows:

| id | name | sex | salary |

|----|------|-----|--------|

| 1 | A | f | 2500 |

| 2 | B | m | 1500 |

| 3 | C | f | 5500 |

| 4 | D | m | 500 |

update salary

set sex = if(sex = 'f', 'm', 'f');

update salary

set sex = case sex

         when 'm' THEN 'f'

         else 'm'

     end

### [1045. Customers Who Bought All Products](https://leetcode-cn.com/problems/customers-who-bought-all-products/)

Table: Customer

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

| Column Name | Type |

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

| customer\_id | int |

| product\_key | int |

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

product\_key is a foreign key to Product table.

Table: Product

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

| Column Name | Type |

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

| product\_key | int |

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

product\_key is the primary key column for this table.

Write an SQL query for a report that provides the customer ids from the Customer table that bought all the products in the Product table.

For example:

Customer table:

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

| customer\_id | product\_key |

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

| 1 | 5 |

| 2 | 6 |

| 3 | 5 |

| 3 | 6 |

| 1 | 6 |

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

Product table:

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

| product\_key |

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

| 5 |

| 6 |

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

Result table:

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

| customer\_id |

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

| 1 |

| 3 |

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

The customers who bought all the products (5 and 6) are customers with id 1 and 3.

select customer\_id

from Customer

group by customer\_id

having count(distinct product\_key) = (select count(\*) from Product)

### [1050. Actors and Directors Who Cooperated At Least Three Times](https://leetcode-cn.com/problems/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 have cooperated with the director at least 3 times.

**Example:**

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 |

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

Result table:

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

| actor\_id | director\_id |

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

| 1 | 1 |

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

The only pair is (1, 1) where they cooperated exactly 3 times.

select actor\_id, director\_id

from ActorDirector

group by actor\_id, director\_id

having count(\*) >= 3

### [1068. Product Sales Analysis I](https://leetcode-cn.com/problems/product-sales-analysis-i/)

Table: Sales

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

| Column Name | Type |

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

| sale\_id | int |

| product\_id | int |

| year | int |

| quantity | int |

| price | int |

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

(sale\_id, year) is the primary key of this table.

product\_id is a foreign key to Product table.

Note that the price is per unit.

Table: Product

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

| Column Name | Type |

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

| product\_id | int |

| product\_name | varchar |

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

product\_id is the primary key of this table.

Write an SQL query that reports all **product names** of the products in the Sales table along with their selling **year** and **price**.

For example:

Sales table:

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

| sale\_id | product\_id | year | quantity | price |

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

| 1 | 100 | 2008 | 10 | 5000 |

| 2 | 100 | 2009 | 12 | 5000 |

| 7 | 200 | 2011 | 15 | 9000 |

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

Product table:

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

| product\_id | product\_name |

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

| 100 | Nokia |

| 200 | Apple |

| 300 | Samsung |

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

Result table:

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

| product\_name | year | price |

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

| Nokia | 2008 | 5000 |

| Nokia | 2009 | 5000 |

| Apple | 2011 | 9000 |

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

select product\_name, year, price

from Sales a

join Product b on a.product\_id = b.product\_id

### [1069. Product Sales Analysis II](https://leetcode-cn.com/problems/product-sales-analysis-ii/)

Table: Sales

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

| Column Name | Type |

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

| sale\_id | int |

| product\_id | int |

| year | int |

| quantity | int |

| price | int |

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

sale\_id is the primary key of this table.

product\_id is a foreign key to Product table.

Note that the price is per unit.

Table: Product

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

| Column Name | Type |

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

| product\_id | int |

| product\_name | varchar |

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

product\_id is the primary key of this table.

Write an SQL query that reports the total quantity sold for every product id.

The query result format is in the following example:

Sales table:

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

| sale\_id | product\_id | year | quantity | price |

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

| 1 | 100 | 2008 | 10 | 5000 |

| 2 | 100 | 2009 | 12 | 5000 |

| 7 | 200 | 2011 | 15 | 9000 |

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

Product table:

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

| product\_id | product\_name |

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

| 100 | Nokia |

| 200 | Apple |

| 300 | Samsung |

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

Result table:

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

| product\_id | total\_quantity |

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

| 100 | 22 |

| 200 | 15 |

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

select product\_id, sum(quantity) as total\_quantity

from Sales

group by product\_id

### [1070. Product Sales Analysis III](https://leetcode-cn.com/problems/product-sales-analysis-iii/)

Table: Sales

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

| Column Name | Type |

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

| sale\_id | int |

| product\_id | int |

| year | int |

| quantity | int |

| price | int |

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

sale\_id is the primary key of this table.

product\_id is a foreign key to Product table.

Note that the price is per unit.

Table: Product

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

| Column Name | Type |

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

| product\_id | int |

| product\_name | varchar |

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

product\_id is the primary key of this table.

Write an SQL query that selects the **product id**, **year**, **quantity**, and **price** for the **first year** of every product sold.

The query result format is in the following example:

Sales table:

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

| sale\_id | product\_id | year | quantity | price |

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

| 1 | 100 | 2008 | 10 | 5000 |

| 2 | 100 | 2009 | 12 | 5000 |

| 7 | 200 | 2011 | 15 | 9000 |

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

Product table:

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

| product\_id | product\_name |

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

| 100 | Nokia |

| 200 | Apple |

| 300 | Samsung |

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

Result table:

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

| product\_id | first\_year | quantity | price |

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

| 100 | 2008 | 10 | 5000 |

| 200 | 2011 | 15 | 9000 |

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

select  product\_id, year as first\_year, quantity, price

from Sales

where (product\_id, year) in (

                                select product\_id, min(year)

                                from Sales

                                group by product\_id

                            )

### [1075. Project Employees I](https://leetcode-cn.com/problems/project-employees-i/)

Table: Project

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

| Column Name | Type |

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

| project\_id | int |

| employee\_id | int |

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

(project\_id, employee\_id) is the primary key of this table.

employee\_id is a foreign key to Employee table.

Table: Employee

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

| Column Name | Type |

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

| employee\_id | int |

| name | varchar |

| experience\_years | int |

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

employee\_id is the primary key of this table.

Write an SQL query that reports the **average** experience years of all the employees for each project, **rounded to 2 digits**.

The query result format is in the following example:

Project table:

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

| project\_id | employee\_id |

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

| 1 | 1 |

| 1 | 2 |

| 1 | 3 |

| 2 | 1 |

| 2 | 4 |

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

Employee table:

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

| employee\_id | name | experience\_years |

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

| 1 | Khaled | 3 |

| 2 | Ali | 2 |

| 3 | John | 1 |

| 4 | Doe | 2 |

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

Result table:

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

| project\_id | average\_years |

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

| 1 | 2.00 |

| 2 | 2.50 |

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

The average experience years for the first project is (3 + 2 + 1) / 3 = 2.00 and for the second project is (3 + 2) / 2 = 2.50

select project\_id, round(avg(experience\_years), 2) as average\_years

from Project a

join Employee b on a.employee\_id = b.employee\_id

group by a.project\_id

### [1076. Project Employees II](https://leetcode-cn.com/problems/project-employees-ii/)

Table: Project

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

| Column Name | Type |

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

| project\_id | int |

| employee\_id | int |

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

(project\_id, employee\_id) is the primary key of this table.

employee\_id is a foreign key to Employee table.

Table: Employee

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

| Column Name | Type |

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

| employee\_id | int |

| name | varchar |

| experience\_years | int |

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

employee\_id is the primary key of this table.

Write an SQL query that reports all the **projects** that have the most employees.

The query result format is in the following example:

Project table:

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

| project\_id | employee\_id |

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

| 1 | 1 |

| 1 | 2 |

| 1 | 3 |

| 2 | 1 |

| 2 | 4 |

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

Employee table:

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

| employee\_id | name | experience\_years |

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

| 1 | Khaled | 3 |

| 2 | Ali | 2 |

| 3 | John | 1 |

| 4 | Doe | 2 |

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

Result table:

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

| project\_id |

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

| 1 |

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

The first project has 3 employees while the second one has 2.

select project\_id

from project

group by project\_id

having count(\*) >= all(select count(\*) over(partition by project\_id) from project);

### [1077. Project Employees III](https://leetcode-cn.com/problems/project-employees-iii/)

Table: Project

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

| Column Name | Type |

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

| project\_id | int |

| employee\_id | int |

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

(project\_id, employee\_id) is the primary key of this table.

employee\_id is a foreign key to Employee table.

Table: Employee

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

| Column Name | Type |

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

| employee\_id | int |

| name | varchar |

| experience\_years | int |

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

employee\_id is the primary key of this table.

Write an SQL query that reports the **most experienced** employees in each project. In case of a tie, report all employees with the maximum number of experience years.

The query result format is in the following example:

Project table:

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

| project\_id | employee\_id |

| 1 | 1 |

| 1 | 2 |

| 1 | 3 |

| 2 | 1 |

| 2 | 4 |

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

Employee table:

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

| employee\_id | name | experience\_years |

| 1 | Khaled | 3 |

| 2 | Ali | 2 |

| 3 | John | 3 |

| 4 | Doe | 2 |

Result table:

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

| project\_id | employee\_id |

| 1 | 1 |

| 1 | 3 |

| 2 | 1 |

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

Both employees with id 1 and 3 have the most experience among the employees of the first project. For the second project, the employee with id 1 has the most experience.

select a.project\_id, a.employee\_id

from Project a

join Employee b on a.employee\_id = b.employee\_id

where (a.project\_id, b.experience\_years) in (

                                                select a.project\_id, max(experience\_years)

                                                from Project a

                                                join Employee b on a.employee\_id = b.employee\_id

                                                group by a.project\_id

                                            )

SELECT project\_id, employee\_id

FROM (

        SELECT p.project\_id, p.employee\_id,

                RANK() OVER(PARTITION BY p.project\_id ORDER BY e.experience\_years DESC) as 'num'

        FROM Project p

        JOIN Employee e ON p.employee\_id = e.employee\_id

    ) t

WHERE num = 1

### [1082. Sales Analysis I](https://leetcode-cn.com/problems/sales-analysis-i/)

Table: Product

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

| Column Name | Type |

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

| product\_id | int |

| product\_name | varchar |

| unit\_price | int |

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

product\_id is the primary key of this table.

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 Product table.

Write an SQL query that reports the best **seller** by total sales price, If there is a tie, report them all.

The query result format is in the following example:

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 |

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

Result table:

| seller\_id |

| 1 |

| 3 |

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

Both sellers with id 1 and 3 sold products with the most total price of 2800.

select seller\_id

from sales

group by seller\_id

having sum(price) >= all(select sum(price) over(partition by seller\_id) from sales)

### [1083. Sales Analysis II](https://leetcode-cn.com/problems/sales-analysis-ii/)

Table: Product

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

| Column Name | Type |

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

| product\_id | int |

| product\_name | varchar |

| unit\_price | int |

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

product\_id is the primary key of this table.

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 Product table.

Write an SQL query that reports the **buyers** who have bought S8 but not iPhone. Note that S8 and iPhone are products present in the Product table.

The query result format is in the following example:

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 | 1 | 3 | 2019-06-02 | 1 | 800 |

| 3 | 3 | 3 | 2019-05-13 | 2 | 2800 |

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

Result table:

| buyer\_id |

| 1 |

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

The buyer with id 1 bought an S8 but didn't buy an iPhone. The buyer with id 3 bought both.

select s.buyer\_id

from product p

join sales s

where p.product\_id = s.product\_id

group by s.buyer\_id

having sum(p.product\_name = 'S8') > 0 and sum(p.product\_name = 'iphone') < 1

select distinct buyer\_id

from Sales

where buyer\_id in

                (

                    select buyer\_id

                    from Sales a

                    join Product b

on a.product\_id = b.product\_id and product\_name = "S8"

                )

     and buyer\_id not in

                (

                    select buyer\_id

                    from Sales a

                    join Product b

on a.product\_id = b.product\_id and product\_name = "iPhone"

                )

### [1084. Sales Analysis III](https://leetcode-cn.com/problems/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.

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 Product table.

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

The query result format is in the following example:

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 |

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

Result table:

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

| product\_id | product\_name |

| 1 | S8 |

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

The product with id 1 was only sold in spring 2019 while the other two were sold after.

select a.product\_id, product\_name

from Sales a

join Product b on a.product\_id = b.product\_id

group by product\_id

having max(sale\_date) <= '2019-03-31' and min(sale\_date) >= '2019-01-01'

### [1097. Game Play Analysis V](https://leetcode-cn.com/problems/game-play-analysis-v/)

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 game.

Each row is a record of a player who logged in and played a number of games (possibly 0) before logging out on some day using some device.

We define the install date of a player to be the first login day of that player.

We also define day 1 retention of some date X to be the number of players whose install date is X and they logged back in on the day right after X, divided by the number of players whose install date is X, **rounded to 2 decimal places**.

Write an SQL query that reports for each **install date**, the **number of players** that installed the game on that day and the **day 1 retention**.

The query result format is in the following example:

Activity table:

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

| player\_id | device\_id | event\_date | games\_played |

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

| 1 | 2 | 2016-03-01 | 5 |

| 1 | 2 | 2016-03-02 | 6 |

| 2 | 3 | 2017-06-25 | 1 |

| 3 | 1 | 2016-03-01 | 0 |

| 3 | 4 | 2016-07-03 | 5 |

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

Result table:

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

| install\_dt | installs | Day1\_retention |

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

| 2016-03-01 | 2 | 0.50 |

| 2017-06-25 | 1 | 0.00 |

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

Player 1 and 3 installed the game on 2016-03-01 but only player 1 logged back in on 2016-03-02 so the day 1 retention of 2016-03-01 is 1 / 2 = 0.50

Player 2 installed the game on 2017-06-25 but didn't log back in on 2017-06-26 so the day 1 retention of 2017-06-25 is 0 / 1 = 0.00

select a1.install\_dt,

       count(\*) installs,

       round(count(a2.event\_date)/count(\*),2) Day1\_retention

from (

        select player\_id, min(event\_date) install\_dt

        from Activity

        group by player\_id

     ) a1

left join Activity a2 on a1.player\_id = a2.player\_id

                and datediff(a2.event\_date, a1.install\_dt) = 1

group by a1.install\_dt

### [1098. Unpopular Books](https://leetcode-cn.com/problems/unpopular-books/)

Table: Books

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

| Column Name | Type |

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

| book\_id | int |

| name | varchar |

| available\_from | date |

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

book\_id is the primary key of this table.

Table: Orders

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

| Column Name | Type |

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

| order\_id | int |

| book\_id | int |

| quantity | int |

| dispatch\_date | date |

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

order\_id is the primary key of this table.

book\_id is a foreign key to the Books table.

Write an SQL query that reports the **books** that have sold **less than 10** copies in the last year, excluding books that have been available for less than 1 month from today. **Assume today is 2019-06-23**.

The query result format is in the following example:

Books table:

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

| book\_id | name | available\_from |

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

| 1 | "Kalila And Demna" | 2010-01-01 |

| 2 | "28 Letters" | 2012-05-12 |

| 3 | "The Hobbit" | 2019-06-10 |

| 4 | "13 Reasons Why" | 2019-06-01 |

| 5 | "The Hunger Games" | 2008-09-21 |

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

Orders table:

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

| order\_id | book\_id | quantity | dispatch\_date |

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

| 1 | 1 | 2 | 2018-07-26 |

| 2 | 1 | 1 | 2018-11-05 |

| 3 | 3 | 8 | 2019-06-11 |

| 4 | 4 | 6 | 2019-06-05 |

| 5 | 4 | 5 | 2019-06-20 |

| 6 | 5 | 9 | 2009-02-02 |

| 7 | 5 | 8 | 2010-04-13 |

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

Result table:

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

| book\_id | name |

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

| 1 | "Kalila And Demna" |

| 2 | "28 Letters" |

| 5 | "The Hunger Games" |

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

select a.book\_id, a.name

from books a

left join orders b on a.book\_id = b.book\_id

where available\_from < '2019-05-23'

group by a.book\_id

having ifnull(sum(if(dispatch\_date < '2018-06-23', 0, quantity)), 0) < 10

order by a.book\_id

### [1107. New Users Daily Count](https://leetcode-cn.com/problems/new-users-daily-count/)

Table: Traffic

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

| Column Name | Type |

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

| user\_id | int |

| activity | enum |

| activity\_date | date |

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

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

The activity column is an ENUM type of ('login', 'logout', 'jobs', 'groups', 'homepage').

Write an SQL query that reports for every date within at most **90 days** from today, the number of users that logged in for the first time on that date. Assume today is **2019-06-30**.

The query result format is in the following example:

Traffic table:

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

| user\_id | activity | activity\_date |

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

| 1 | login | 2019-05-01 |

| 1 | homepage | 2019-05-01 |

| 1 | logout | 2019-05-01 |

| 2 | login | 2019-06-21 |

| 2 | logout | 2019-06-21 |

| 3 | login | 2019-01-01 |

| 3 | jobs | 2019-01-01 |

| 3 | logout | 2019-01-01 |

| 4 | login | 2019-06-21 |

| 4 | groups | 2019-06-21 |

| 4 | logout | 2019-06-21 |

| 5 | login | 2019-03-01 |

| 5 | logout | 2019-03-01 |

| 5 | login | 2019-06-21 |

| 5 | logout | 2019-06-21 |

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

Result table:

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

| login\_date | user\_count |

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

| 2019-05-01 | 1 |

| 2019-06-21 | 2 |

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

Note that we only care about dates with non zero user count.

The user with id 5 first logged in on 2019-03-01 so he's not counted on 2019-06-21.

select t.d login\_date, count(t.user\_id) user\_count

from (

    select user\_id, min(activity\_date) as d

    from Traffic

    where activity = "login"

    group by user\_id

    having datediff('2019-06-30', d) <= 90

) t

group by t.d

### [1112. Highest Grade For Each Student](https://leetcode-cn.com/problems/highest-grade-for-each-student/)

Table: Enrollments

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

| Column Name | Type |

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

| student\_id | int |

| course\_id | int |

| grade | int |

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

(student\_id, course\_id) is the primary key of this table.

Write a SQL query to find the highest grade with its corresponding course for each student. In case of a tie, you should find the course with the smallest course\_id. The output must be sorted by increasing student\_id.

The query result format is in the following example:

Enrollments table:

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

| student\_id | course\_id | grade |

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

| 2 | 2 | 95 |

| 2 | 3 | 95 |

| 1 | 1 | 90 |

| 1 | 2 | 99 |

| 3 | 1 | 80 |

| 3 | 2 | 75 |

| 3 | 3 | 82 |

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

Result table:

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

| student\_id | course\_id | grade |

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

| 1 | 2 | 99 |

| 2 | 2 | 95 |

| 3 | 3 | 82 |

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

select student\_id, course\_id, grade

from (

        select \*, row\_number() over(partition by student\_id order by grade desc, course\_id) as n

        from Enrollments

    ) as a

where n = 1;

select student\_id, min(course\_id) course\_id, grade

from Enrollments

where (student\_id, grade) in (

                                select student\_id, max(grade)

                                from Enrollments

                                group by student\_id

                            )

group by student\_id, grade

order by student\_id

### [1113. Reported Posts](https://leetcode-cn.com/problems/reported-posts/)

Table: Actions

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

| Column Name | Type |

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

| user\_id | int |

| post\_id | int |

| action\_date | date |

| action | enum |

| extra | varchar |

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

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

The action column is an ENUM type of ('view', 'like', 'reaction', 'comment', 'report', 'share').

The extra column has optional information about the action such as a reason for report or a type of reaction.

Write an SQL query that reports the number of posts reported yesterday for each report reason. Assume today is **2019-07-05**.

The query result format is in the following example:

Actions table:

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

| user\_id | post\_id | action\_date | action | extra |

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

| 1 | 1 | 2019-07-01 | view | null |

| 1 | 1 | 2019-07-01 | like | null |

| 1 | 1 | 2019-07-01 | share | null |

| 2 | 4 | 2019-07-04 | view | null |

| 2 | 4 | 2019-07-04 | report | spam |

| 3 | 4 | 2019-07-04 | view | null |

| 3 | 4 | 2019-07-04 | report | spam |

| 4 | 3 | 2019-07-02 | view | null |

| 4 | 3 | 2019-07-02 | report | spam |

| 5 | 2 | 2019-07-04 | view | null |

| 5 | 2 | 2019-07-04 | report | racism |

| 5 | 5 | 2019-07-04 | view | null |

| 5 | 5 | 2019-07-04 | report | racism |

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

Result table:

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

| report\_reason | report\_count |

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

| spam | 1 |

| racism | 2 |

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

Note that we only care about report reasons with non zero number of reports.

select extra report\_reason, count(distinct post\_id) report\_count

from Actions

where action\_date = '2019-07-04'

and extra is not null and action = "report"

group by extra

### [1126. Active Businesses](https://leetcode-cn.com/problems/active-businesses/)

Table: Events

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

| Column Name | Type |

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

| business\_id | int |

| event\_type | varchar |

| occurences | int |

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

(business\_id, event\_type) is the primary key of this table.

Each row in the table logs the info that an event of some type occured at some business for a number of times.

Write an SQL query to find all active businesses.

An active business is a business that has more than one event type with occurences greater than the average occurences of that event type among all businesses.

The query result format is in the following example:

Events table:

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

| business\_id | event\_type | occurences |

| 1 | reviews | 7 |

| 3 | reviews | 3 |

| 1 | ads | 11 |

| 2 | ads | 7 |

| 3 | ads | 6 |

| 1 | page views | 3 |

| 2 | page views | 12 |

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

Result table:

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

| business\_id |

| 1 |

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

Average for 'reviews', 'ads' and 'page views' are (7+3)/2=5, (11+7+6)/3=8, (3+12)/2=7.5 respectively.

Business with id 1 has 7 'reviews' events (more than 5) and 11 'ads' events (more than 8) so it is an active business.

select business\_id

from Events a

join (

        select event\_type, avg(occurences) as avg\_o

        from Events

        group by event\_type

    ) b on a.event\_type = b.event\_type and a.occurences > b.avg\_o

group by business\_id

having count(\*) > 1

### [1127. User Purchase Platform](https://leetcode-cn.com/problems/user-purchase-platform/)★

Table: Spending

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

| Column Name | Type |

| user\_id | int |

| spend\_date | date |

| platform | enum |

| amount | int |

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

The table logs the spendings history of users that make purchases from an online shopping website which has a desktop and a mobile application.

(user\_id, spend\_date, platform) is the primary key of this table.

The platform column is an ENUM type of ('desktop', 'mobile').

Write an SQL query to find the total number of users and the total amount spent using mobile **only**, desktop **only** and **both** mobile and desktop together for each date.

The query result format is in the following example:

Spending table:

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

| user\_id | spend\_date | platform | amount |

| 1 | 2019-07-01 | mobile | 100 |

| 1 | 2019-07-01 | desktop | 100 |

| 2 | 2019-07-01 | mobile | 100 |

| 2 | 2019-07-02 | mobile | 100 |

| 3 | 2019-07-01 | desktop | 100 |

| 3 | 2019-07-02 | desktop | 100 |

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

Result table:

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

| spend\_date | platform | total\_amount | total\_users |

| 2019-07-01 | desktop | 100 | 1 |

| 2019-07-01 | mobile | 100 | 1 |

| 2019-07-01 | both | 200 | 1 |

| 2019-07-02 | desktop | 100 | 1 |

| 2019-07-02 | mobile | 100 | 1 |

| 2019-07-02 | both | 0 | 0 |

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

On 2019-07-01, user 1 purchased using **both** desktop and mobile, user 2 purchased using mobile **only** and user 3 purchased using desktop **only**.

On 2019-07-02, user 2 purchased using mobile **only**, user 3 purchased using desktop **only** and no one purchased using **both** platforms.

select t2.\*, ifnull(sum(amount), 0) total\_amount, ifnull(count(user\_id), 0) total\_users

from (

        select distinct spend\_date, "desktop" as platform from Spending

        union

        select distinct spend\_date, "mobile" as platform from Spending

        union

        select distinct spend\_date, "both" as platform from Spending

     ) t2

left join (

             select spend\_date, sum(amount) amount, user\_id,

                   case when count(\*) = 1 then platform else "both" end as platform

             from Spending

             group by spend\_date, user\_id

          ) t1 on t1.spend\_date = t2.spend\_date and t1.platform = t2.platform

group by t2.spend\_date, t2.platform

### [1132. Reported Posts II](https://leetcode-cn.com/problems/reported-posts-ii/)

Table: Actions

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

| Column Name | Type |

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

| user\_id | int |

| post\_id | int |

| action\_date | date |

| action | enum |

| extra | varchar |

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

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

The action column is an ENUM type of ('view', 'like', 'reaction', 'comment', 'report', 'share').

The extra column has optional information about the action such as a reason for report or a type of reaction.

Table: Removals

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

| Column Name | Type |

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

| post\_id | int |

| remove\_date | date |

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

post\_id is the primary key of this table.

Each row in this table indicates that some post was removed as a result of being reported or as a result of an admin review.

Write an SQL query to find the average for daily percentage of posts that got removed after being reported as spam, **rounded to 2 decimal places**.

The query result format is in the following example:

Actions table:

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

| user\_id | post\_id | action\_date | action | extra |

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

| 1 | 1 | 2019-07-01 | view | null |

| 1 | 1 | 2019-07-01 | like | null |

| 1 | 1 | 2019-07-01 | share | null |

| 2 | 2 | 2019-07-04 | view | null |

| 2 | 2 | 2019-07-04 | report | spam |

| 3 | 4 | 2019-07-04 | view | null |

| 3 | 4 | 2019-07-04 | report | spam |

| 4 | 3 | 2019-07-02 | view | null |

| 4 | 3 | 2019-07-02 | report | spam |

| 5 | 2 | 2019-07-03 | view | null |

| 5 | 2 | 2019-07-03 | report | racism |

| 5 | 5 | 2019-07-03 | view | null |

| 5 | 5 | 2019-07-03 | report | racism |

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

Removals table:

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

| post\_id | remove\_date |

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

| 2 | 2019-07-20 |

| 3 | 2019-07-18 |

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

Result table:

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

| average\_daily\_percent |

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

| 75.00 |

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

The percentage for 2019-07-04 is 50% because only one post of two spam reported posts was removed.

The percentage for 2019-07-02 is 100% because one post was reported as spam and it was removed.

The other days had no spam reports so the average is (50 + 100) / 2 = 75%

Note that the output is only one number and that we do not care about the remove dates.

select round(avg(a)\*100, 2)  average\_daily\_percent

from (

        select (count(distinct b.post\_id)/count(distinct a.post\_id)) a

        from actions a

        left join removals b on a.post\_id = b.post\_id

        where extra = 'spam'

        group by action\_date

    ) v

### [1141. User Activity for the Past 30 Days I](https://leetcode-cn.com/problems/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 some day if he/she made at least one activity on that day.

The query result format is in the following example:

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 |

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

Result table:

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

| day | active\_users |

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

| 2019-07-20 | 2 |

| 2019-07-21 | 2 |

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

Note that we do not care about days with zero active users.

select activity\_date day, count(distinct user\_id) active\_users

from Activity

where datediff('2019-07-27', activity\_date) < 30

group by activity\_date

### [1142. User Activity for the Past 30 Days II](https://leetcode-cn.com/problems/user-activity-for-the-past-30-days-ii/)

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 average number of sessions per user for a period of 30 days ending **2019-07-27** inclusively, **rounded to 2 decimal places**. The sessions we want to count for a user are those with at least one activity in that time period.

The query result format is in the following example:

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 |

| 3 | 5 | 2019-07-21 | open\_session |

| 3 | 5 | 2019-07-21 | scroll\_down |

| 3 | 5 | 2019-07-21 | end\_session |

| 4 | 3 | 2019-06-25 | open\_session |

| 4 | 3 | 2019-06-25 | end\_session |

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

Result table:

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

| average\_sessions\_per\_user |

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

| 1.33 |

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

User 1 and 2 each had 1 session in the past 30 days while user 3 had 2 sessions so the average is (1 + 1 + 2) / 3 = 1.33.

select round(ifnull(count(distinct user\_id, session\_id)

/ count(distinct user\_id), 0), 2) average\_sessions\_per\_user

from Activity

where datediff('2019-07-27', activity\_date) < 30

### [1148. Article Views I](https://leetcode-cn.com/problems/article-views-i/)

Table: Views

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

| Column Name | Type |

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

| article\_id | int |

| author\_id | int |

| viewer\_id | int |

| view\_date | date |

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

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

Each row of this table indicates that some viewer viewed an article (written by some author) on some date.

Note that equal author\_id and viewer\_id indicate the same person.

Write an SQL query to find all the authors that viewed at least one of their own articles, sorted in ascending order by their id.

The query result format is in the following example:

Views table:

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

| article\_id | author\_id | viewer\_id | view\_date |

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

| 1 | 3 | 5 | 2019-08-01 |

| 1 | 3 | 6 | 2019-08-02 |

| 2 | 7 | 7 | 2019-08-01 |

| 2 | 7 | 6 | 2019-08-02 |

| 4 | 7 | 1 | 2019-07-22 |

| 3 | 4 | 4 | 2019-07-21 |

| 3 | 4 | 4 | 2019-07-21 |

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

Result table:

+------+

| id |

+------+

| 4 |

| 7 |

+------+

select distinct author\_id as id

from Views

where author\_id = viewer\_id

order by author\_id

### [1149. Article Views II](https://leetcode-cn.com/problems/article-views-ii/)

Table: Views

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

| Column Name | Type |

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

| article\_id | int |

| author\_id | int |

| viewer\_id | int |

| view\_date | date |

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

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

Each row of this table indicates that some viewer viewed an article (written by some author) on some date.

Note that equal author\_id and viewer\_id indicate the same person.

Write an SQL query to find all the people who viewed more than one article on the same date, sorted in ascending order by their id.

The query result format is in the following example:

Views table:

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

| article\_id | author\_id | viewer\_id | view\_date |

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

| 1 | 3 | 5 | 2019-08-01 |

| 3 | 4 | 5 | 2019-08-01 |

| 1 | 3 | 6 | 2019-08-02 |

| 2 | 7 | 7 | 2019-08-01 |

| 2 | 7 | 6 | 2019-08-02 |

| 4 | 7 | 1 | 2019-07-22 |

| 3 | 4 | 4 | 2019-07-21 |

| 3 | 4 | 4 | 2019-07-21 |

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

Result table:

+------+

| id |

+------+

| 5 |

| 6 |

+------+

select distinct viewer\_id id

from Views

group by viewer\_id, view\_date

having count(distinct article\_id) > 1

order by viewer\_id asc

### [1158. Market Analysis I](https://leetcode-cn.com/problems/market-analysis-i/)

Table: Users

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

| Column Name | Type |

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

| user\_id | int |

| join\_date | date |

| favorite\_brand | varchar |

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

user\_id is the primary key of this table.

This table has the info of the users of an online shopping website where users can sell and buy items.

Table: Orders

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

| Column Name | Type |

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

| order\_id | int |

| order\_date | date |

| item\_id | int |

| buyer\_id | int |

| seller\_id | int |

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

order\_id is the primary key of this table.

item\_id is a foreign key to the Items table.

buyer\_id and seller\_id are foreign keys to the Users table.

Table: Items

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

| Column Name | Type |

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

| item\_id | int |

| item\_brand | varchar |

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

item\_id is the primary key of this table.

Write an SQL query to find for each user, the join date and the number of orders they made as a buyer in **2019**.

The query result format is in the following example:

Users table:

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

| user\_id | join\_date | favorite\_brand |

| 1 | 2018-01-01 | Lenovo |

| 2 | 2018-02-09 | Samsung |

| 3 | 2018-01-19 | LG |

| 4 | 2018-05-21 | HP |

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

Orders table:

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

| order\_id | order\_date | item\_id | buyer\_id | seller\_id |

| 1 | 2019-08-01 | 4 | 1 | 2 |

| 2 | 2018-08-02 | 2 | 1 | 3 |

| 3 | 2019-08-03 | 3 | 2 | 3 |

| 4 | 2018-08-04 | 1 | 4 | 2 |

| 5 | 2018-08-04 | 1 | 3 | 4 |

| 6 | 2019-08-05 | 2 | 2 | 4 |

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

Items table:

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

| item\_id | item\_brand |

| 1 | Samsung |

| 2 | Lenovo |

| 3 | LG |

| 4 | HP |

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

Result table:

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

| buyer\_id | join\_date | orders\_in\_2019 |

| 1 | 2018-01-01 | 1 |

| 2 | 2018-02-09 | 2 |

| 3 | 2018-01-19 | 0 |

| 4 | 2018-05-21 | 0 |

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

select a.user\_id buyer\_id, join\_date, count(seller\_id) as orders\_in\_2019

from Users a

left join Orders b

on a.user\_id = b.buyer\_id and b.order\_date between '2019-01-01' and '2019-12-31'

group by a.user\_id

### [1159. Market Analysis II](https://leetcode-cn.com/problems/market-analysis-ii/)★

Table: Users

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

| Column Name | Type |

| user\_id | int |

| join\_date | date |

| favorite\_brand | varchar |

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

user\_id is the primary key of this table.

This table has the info of the users of an online shopping website where users can sell and buy items.

Table: Orders

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

| Column Name | Type |

| order\_id | int |

| order\_date | date |

| item\_id | int |

| buyer\_id | int |

| seller\_id | int |

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

order\_id is the primary key of this table.

item\_id is a foreign key to the Items table.

buyer\_id and seller\_id are foreign keys to the Users table.

Table: Items

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

| Column Name | Type |

| item\_id | int |

| item\_brand | varchar |

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

item\_id is the primary key of this table.

Write an SQL query to find for each user, whether the brand of the second item (by date) they sold is their favorite brand. If a user sold less than two items, report the answer for that user as no.

It is guaranteed that no seller sold more than one item on a day.

The query result format is in the following example:

Users table:

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

| user\_id | join\_date | favorite\_brand |

| 1 | 2019-01-01 | Lenovo |

| 2 | 2019-02-09 | Samsung |

| 3 | 2019-01-19 | LG |

| 4 | 2019-05-21 | HP |

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

Orders table:

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

| order\_id | order\_date | item\_id | buyer\_id | seller\_id |

| 1 | 2019-08-01 | 4 | 1 | 2 |

| 2 | 2019-08-02 | 2 | 1 | 3 |

| 3 | 2019-08-03 | 3 | 2 | 3 |

| 4 | 2019-08-04 | 1 | 4 | 2 |

| 5 | 2019-08-04 | 1 | 3 | 4 |

| 6 | 2019-08-05 | 2 | 2 | 4 |

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

Items table:

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

| item\_id | item\_brand |

| 1 | Samsung |

| 2 | Lenovo |

| 3 | LG |

| 4 | HP |

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

Result table:

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

| seller\_id | 2nd\_item\_fav\_brand |

| 1 | no |

| 2 | yes |

| 3 | yes |

| 4 | no |

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

The answer for the user with id 1 is no because they sold nothing.

The answer for the users with id 2 and 3 is yes because the brands of their second sold items are their favorite brands.

The answer for the user with id 4 is no because the brand of their second sold item is not their favorite brand.

select user\_id seller\_id,

IF(item\_brand = favorite\_brand, 'yes', 'no') 2nd\_item\_fav\_brand

from Users a

left join (

              select \*, rank() over (partition by seller\_id order by order\_date) rk

              from Orders

          ) b on a.user\_id = b.seller\_id and rk = 2

left join Items c on b.item\_id = c.item\_id

group by user\_id, favorite\_brand, item\_brand

### [1164. Product Price at a Given Date](https://leetcode-cn.com/problems/product-price-at-a-given-date/)

Table: Products

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

| Column Name | Type |

| product\_id | int |

| new\_price | int |

| change\_date | date |

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

(product\_id, change\_date) is the primary key of this table.

Each row of this table indicates that the price of some product was changed to a new price at some date.

Write an SQL query to find the prices of all products on **2019-08-16**. Assume the price of all products before any change is **10**.

The query result format is in the following example:

Products table:

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

| product\_id | new\_price | change\_date |

| 1 | 20 | 2019-08-14 |

| 2 | 50 | 2019-08-14 |

| 1 | 30 | 2019-08-15 |

| 1 | 35 | 2019-08-16 |

| 2 | 65 | 2019-08-17 |

| 3 | 20 | 2019-08-18 |

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

Result table:

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

| product\_id | price |

| 2 | 50 |

| 1 | 35 |

| 3 | 10 |

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

select distinct a.product\_id, ifnull(b.new\_price,10) as price

from Products a

left join (

                select \*, row\_number() over(partition by product\_id order by change\_date desc)as rk

                from Products

                where change\_date <= '2019-08-16'

          ) b on a.product\_id = b.product\_id and rk = 1

### [1173. Immediate Food Delivery I](https://leetcode-cn.com/problems/immediate-food-delivery-i/)

Table: Delivery

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

| Column Name | Type |

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

| delivery\_id | int |

| customer\_id | int |

| order\_date | date |

| customer\_pref\_delivery\_date | date |

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

delivery\_id is the primary key of this table.

The table holds information about food delivery to customers that make orders at some date and specify a preferred delivery date (on the same order date or after it).

If the preferred delivery date of the customer is the same as the order date then the order is called immediate otherwise it's called scheduled.

Write an SQL query to find the percentage of immediate orders in the table, **rounded to 2 decimal places**.

The query result format is in the following example:

Delivery table:

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

| delivery\_id | customer\_id | order\_date | customer\_pref\_delivery\_date |

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

| 1 | 1 | 2019-08-01 | 2019-08-02 |

| 2 | 5 | 2019-08-02 | 2019-08-02 |

| 3 | 1 | 2019-08-11 | 2019-08-11 |

| 4 | 3 | 2019-08-24 | 2019-08-26 |

| 5 | 4 | 2019-08-21 | 2019-08-22 |

| 6 | 2 | 2019-08-11 | 2019-08-13 |

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

Result table:

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

| immediate\_percentage |

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

| 33.33 |

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

The orders with delivery id 2 and 3 are immediate while the others are scheduled.

select round(sum(order\_date = customer\_pref\_delivery\_date)

/ count(\*)\*100, 2) immediate\_percentage

from Delivery

### [1174. Immediate Food Delivery II](https://leetcode-cn.com/problems/immediate-food-delivery-ii/)

Table: Delivery

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

| Column Name | Type |

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

| delivery\_id | int |

| customer\_id | int |

| order\_date | date |

| customer\_pref\_delivery\_date | date |

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

delivery\_id is the primary key of this table.

The table holds information about food delivery to customers that make orders at some date and specify a preferred delivery date (on the same order date or after it).

If the preferred delivery date of the customer is the same as the order date then the order is called immediate otherwise it's called scheduled.

The first order of a customer is the order with the earliest order date that customer made. It is guaranteed that a customer has exactly one first order.

Write an SQL query to find the percentage of immediate orders in the first orders of all customers, **rounded to 2 decimal places**.

The query result format is in the following example:

Delivery table:

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

| delivery\_id | customer\_id | order\_date | customer\_pref\_delivery\_date |

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

| 1 | 1 | 2019-08-01 | 2019-08-02 |

| 2 | 2 | 2019-08-02 | 2019-08-02 |

| 3 | 1 | 2019-08-11 | 2019-08-12 |

| 4 | 3 | 2019-08-24 | 2019-08-24 |

| 5 | 3 | 2019-08-21 | 2019-08-22 |

| 6 | 2 | 2019-08-11 | 2019-08-13 |

| 7 | 4 | 2019-08-09 | 2019-08-09 |

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

Result table:

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

| immediate\_percentage |

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

| 50.00 |

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

The customer id 1 has a first order with delivery id 1 and it is scheduled.

The customer id 2 has a first order with delivery id 2 and it is immediate.

The customer id 3 has a first order with delivery id 5 and it is scheduled.

The customer id 4 has a first order with delivery id 7 and it is immediate.

Hence, half the customers have immediate first orders.

select round(sum(order\_date = customer\_pref\_delivery\_date)

/ count(distinct customer\_id)\*100, 2) immediate\_percentage

from Delivery

where (customer\_id, order\_date) in

(

                                    select customer\_id, min(order\_date)

                                    from Delivery

                                    group by customer\_id

                                )

### [1179. Reformat Department Table](https://leetcode-cn.com/problems/reformat-department-table/)★★★

Table: Department

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

| Column Name | Type |

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

| id | int |

| revenue | int |

| month | varchar |

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

(id, month) is the primary key of this table.

The table has information about the revenue of each department per month.

The month has values in ["Jan","Feb","Mar","Apr","May","Jun","Jul","Aug","Sep","Oct","Nov","Dec"].

Write an SQL query to reformat the table such that there is a department id column and a revenue column **for each month**.

The query result format is in the following example:

Department table:

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

| id | revenue | month |

| 1 | 8000 | Jan |

| 2 | 9000 | Jan |

| 3 | 10000 | Feb |

| 1 | 7000 | Feb |

| 1 | 6000 | Mar |

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

Result table:

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

| id | Jan\_Revenue | Feb\_Revenue | Mar\_Revenue | ... | Dec\_Revenue |

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

| 1 | 8000 | 7000 | 6000 | ... | null |

| 2 | 9000 | null | null | ... | null |

| 3 | null | 10000 | null | ... | null |

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

Note that the result table has 13 columns (1 for the department id + 12 for the months).

#不加sum/max/min, 只会去组的第一个，而加上聚合函数会遍历每一个

select  id,

        sum(case `month` when 'Jan' then revenue end) as Jan\_Revenue,

        sum(case `month` when 'Feb' then revenue end) as Feb\_Revenue,

        sum(case `month` when 'Mar' then revenue end) as Mar\_Revenue,

        sum(case `month` when 'Apr' then revenue end) as Apr\_Revenue,

        sum(case `month` when 'May' then revenue end) as May\_Revenue,

        sum(case `month` when 'Jun' then revenue end) as Jun\_Revenue,

        sum(case `month` when 'Jul' then revenue end) as Jul\_Revenue,

        sum(case `month` when 'Aug' then revenue end) as Aug\_Revenue,

        sum(case `month` when 'Sep' then revenue end) as Sep\_Revenue,

        sum(case `month` when 'Oct' then revenue end) as Oct\_Revenue,

        sum(case `month` when 'Nov' then revenue end) as Nov\_Revenue,

        sum(case `month` when 'Dec' then revenue end) as Dec\_Revenue

from Department

group by id

### [1193. Monthly Transactions I](https://leetcode-cn.com/problems/monthly-transactions-i/)★

Table: Transactions

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

| Column Name | Type |

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

| id | int |

| country | varchar |

| state | enum |

| amount | int |

| trans\_date | date |

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

id is the primary key of this table.

The table has information about incoming transactions.

The state column is an enum of type ["approved", "declined"].

Write an SQL query to find for each month and country, the number of transactions and their total amount, the number of approved transactions and their total amount.

The query result format is in the following example:

Transactions table:

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

| id | country | state | amount | trans\_date |

| 121 | US | approved | 1000 | 2018-12-18 |

| 122 | US | declined | 2000 | 2018-12-19 |

| 123 | US | approved | 2000 | 2019-01-01 |

| 124 | DE | approved | 2000 | 2019-01-07 |

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

Result table:

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

| month | country | trans\_count | approved\_count | trans\_total\_amount | approved\_total\_amount |

| 2018-12 | US | 2 | 1 | 3000 | 1000 |

| 2019-01 | US | 1 | 1 | 2000 | 2000 |

| 2019-01 | DE | 1 | 1 | 2000 | 2000 |

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

#sum的用法： 可以访问同一行的两个不同列的值 ！！

select date\_format(trans\_date, "%Y-%m") month, country, count(\*) trans\_count,

     sum(state = 'approved') approved\_count, sum(amount) trans\_total\_amount,

     sum(case when state = 'approved' then amount else 0 end) approved\_total\_amount

from Transactions

group by month, country

### [1194. Tournament Winners](https://leetcode-cn.com/problems/tournament-winners/)

Table: Players

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

| Column Name | Type |

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

| player\_id | int |

| group\_id | int |

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

player\_id is the primary key of this table.

Each row of this table indicates the group of each player.

Table: Matches

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

| Column Name | Type |

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

| match\_id | int |

| first\_player | int |

| second\_player | int |

| first\_score | int |

| second\_score | int |

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

match\_id is the primary key of this table.

Each row is a record of a match, first\_player and second\_player contain the player\_id of each match.

first\_score and second\_score contain the number of points of the first\_player and second\_player respectively.

You may assume that, in each match, players belongs to the same group.

The winner in each group is the player who scored the maximum total points within the group. In the case of a tie, the **lowest** player\_id wins.

Write an SQL query to find the winner in each group.

The query result format is in the following example:

Players table:

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

| player\_id | group\_id |

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

| 15 | 1 |

| 25 | 1 |

| 30 | 1 |

| 45 | 1 |

| 10 | 2 |

| 35 | 2 |

| 50 | 2 |

| 20 | 3 |

| 40 | 3 |

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

Matches table:

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

| match\_id | first\_player | second\_player | first\_score | second\_score |

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

| 1 | 15 | 45 | 3 | 0 |

| 2 | 30 | 25 | 1 | 2 |

| 3 | 30 | 15 | 2 | 0 |

| 4 | 40 | 20 | 5 | 2 |

| 5 | 35 | 50 | 1 | 1 |

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

Result table:

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

| group\_id | player\_id |

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

| 1 | 15 |

| 2 | 35 |

| 3 | 40 |

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

#在group前可以先排序好， group by之后不会影响原先的顺序， 是稳定分组？

select group\_id, player\_id

from (

    select players.\*, sum(if(player\_id = first\_player, first\_score, second\_score)) score

    from players

    join matches on player\_id = first\_player or player\_id = second\_player

    group by player\_id, group\_id

    order by score desc, player\_id

) tmp

group by group\_id

select group\_id, player\_id

from (

        select group\_id, player\_id,

rank() over (partition by group\_id order by score desc, player\_id asc) as rnk

        from Players a

        join (

                select player, sum(score) score

                from (

                    select first\_player player, first\_score score

                    from Matches

                    union  all

                    select second\_player player, second\_score score

                    from Matches

                ) t

                group by player

            ) a on player\_id = player

     ) b

where rnk = 1

### [1204. Last Person to Fit in the Elevator](https://leetcode-cn.com/problems/last-person-to-fit-in-the-elevator/)

Table: Queue

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

| Column Name | Type |

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

| person\_id | int |

| person\_name | varchar |

| weight | int |

| turn | int |

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

person\_id is the primary key column for this table.

This table has the information about all people waiting for an elevator.

The person\_id and turn columns will contain all numbers from 1 to n, where n is the number of rows in the table.

The maximum weight the elevator can hold is **1000**.

Write an SQL query to find the person\_name of the last person who will fit in the elevator without exceeding the weight limit. It is guaranteed that the person who is first in the queue can fit in the elevator.

The query result format is in the following example:

Queue table

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

| person\_id | person\_name | weight | turn |

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

| 5 | George Washington | 250 | 1 |

| 3 | John Adams | 350 | 2 |

| 6 | Thomas Jefferson | 400 | 3 |

| 2 | Will Johnliams | 200 | 4 |

| 4 | Thomas Jefferson | 175 | 5 |

| 1 | James Elephant | 500 | 6 |

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

Result table

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

| person\_name |

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

| Thomas Jefferson |

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

Queue table is ordered by turn in the example for simplicity.

In the example George Washington(id 5), John Adams(id 3) and Thomas Jefferson(id 6) will enter the elevator as their weight sum is 250 + 350 + 400 = 1000.

Thomas Jefferson(id 6) is the last person to fit in the elevator because he has the last turn in these three people.

select person\_name

from (

        select person\_name, turn,

sum(weight) over(order by turn asc) as sum\_w

        from Queue

     ) as t

where sum\_w <= 1000

order by turn desc

limit 0, 1

select a.person\_name

from Queue a

join Queue b on a.turn >= b.turn

group by a.person\_id

having sum(b.weight) <= 1000

order by sum(b.weight) desc

limit 0, 1

### [1205. Monthly Transactions II](https://leetcode-cn.com/problems/monthly-transactions-ii/)

Table: Transactions

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

| Column Name | Type |

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

| id | int |

| country | varchar |

| state | enum |

| amount | int |

| trans\_date | date |

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

id is the primary key of this table.

The table has information about incoming transactions.

The state column is an enum of type ["approved", "declined"].

Table: Chargebacks

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

| Column Name | Type |

| trans\_id | int |

| charge\_date | date |

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

Chargebacks contains basic information regarding incoming chargebacks from some transactions placed in Transactions table.

trans\_id is a foreign key to the id column of Transactions table.

Each chargeback corresponds to a transaction made previously even if they were not approved.

Write an SQL query to find for each month and country, the number of approved transactions and their total amount, the number of chargebacks and their total amount.

**Note**: In your query, given the month and country, ignore rows with all zeros.

The query result format is in the following example:

Transactions table:

| id | country | state | amount | trans\_date |

| 101 | US | approved | 1000 | 2019-05-18 |

| 102 | US | declined | 2000 | 2019-05-19 |

| 103 | US | approved | 3000 | 2019-06-10 |

| 104 | US | approved | 4000 | 2019-06-13 |

| 105 | US | approved | 5000 | 2019-06-15 |

Chargebacks table:

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

| trans\_id | trans\_date |

| 102 | 2019-05-29 |

| 101 | 2019-06-30 |

| 105 | 2019-09-18 |

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

Result table:

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

| month | country | approved\_count | approved\_amount | chargeback\_count | chargeback\_amount |

| 2019-05 | US | 1 | 1000 | 1 | 2000 |

| 2019-06 | US | 3 | 12000 | 1 | 1000 |

| 2019-09 | US | 0 | 0 | 1 | 5000 |

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

select month, country,

        sum(case when tag=0 then 1 else 0 end) as approved\_count,

        sum(case when tag=0 then amount else 0 end) as approved\_amount,

        sum(case when tag=1 then 1 else 0 end) as chargeback\_count,

        sum(case when tag=1 then amount else 0 end) as chargeback\_amount

from (

        select date\_format(trans\_date,'%Y-%m') month, country, amount, 0 tag

        from Transactions

        where state = 'approved'

        union all

        select date\_format(a.trans\_date,'%Y-%m') month, country, amount, 1 tag

        from Chargebacks a

        left join Transactions b on a.trans\_id = b.id

     ) as t

 group by month, country

### [1211. Queries Quality and Percentage](https://leetcode-cn.com/problems/queries-quality-and-percentage/)

Table: Queries

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

| Column Name | Type |

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

| query\_name | varchar |

| result | varchar |

| position | int |

| rating | int |

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

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

This table contains information collected from some queries on a database.

The position column has a value from **1** to **500**.

The rating column has a value from **1** to **5**. Query with rating less than 3 is a poor query.

We define query quality as:

The average of the ratio between query rating and its position.

We also define poor query percentage as:

The percentage of all queries with rating less than 3.

Write an SQL query to find each query\_name, the quality and poor\_query\_percentage.

Both quality and poor\_query\_percentage should be **rounded to 2 decimal places**.

The query result format is in the following example:

Queries table:

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

| query\_name | result | position | rating |

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

| Dog | Golden Retriever | 1 | 5 |

| Dog | German Shepherd | 2 | 5 |

| Dog | Mule | 200 | 1 |

| Cat | Shirazi | 5 | 2 |

| Cat | Siamese | 3 | 3 |

| Cat | Sphynx | 7 | 4 |

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

Result table:

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

| query\_name | quality | poor\_query\_percentage |

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

| Dog | 2.50 | 33.33 |

| Cat | 0.66 | 33.33 |

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

Dog queries quality is ((5 / 1) + (5 / 2) + (1 / 200)) / 3 = 2.50

Dog queries poor\_ query\_percentage is (1 / 3) \* 100 = 33.33

Cat queries quality equals ((2 / 5) + (3 / 3) + (4 / 7)) / 3 = 0.66

Cat queries poor\_ query\_percentage is (1 / 3) \* 100 = 33.33

select query\_name, round(avg(rating / position), 2) quality,

round(sum(rating < 3) / count(\*)\*100, 2) poor\_query\_percentage

from Queries

group by query\_name

### [1212. Team Scores in Football Tournament](https://leetcode-cn.com/problems/team-scores-in-football-tournament/)

Table: Teams

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

| Column Name | Type |

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

| team\_id | int |

| team\_name | varchar |

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

team\_id is the primary key of this table.

Each row of this table represents a single football team.

Table: Matches

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

| Column Name | Type |

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

| match\_id | int |

| host\_team | int |

| guest\_team | int |

| host\_goals | int |

| guest\_goals | int |

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

match\_id is the primary key of this table.

Each row is a record of a finished match between two different teams.

Teams host\_team and guest\_team are represented by their IDs in the teams table (team\_id) and they scored host\_goals and guest\_goals goals respectively.

You would like to compute the scores of all teams after all matches. Points are awarded as follows:

* A team receives three points if they win a match (Score strictly more goals than the opponent team).
* A team receives one point if they draw a match (Same number of goals as the opponent team).
* A team receives no points if they lose a match (Score less goals than the opponent team).

Write an SQL query that selects the **team\_id**, **team\_name** and **num\_points** of each team in the tournament after all described matches. Result table should be ordered by **num\_points** (decreasing order). In case of a tie, order the records by **team\_id** (increasing order).

The query result format is in the following example:

Teams table:

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

| team\_id | team\_name |

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

| 10 | Leetcode FC |

| 20 | NewYork FC |

| 30 | Atlanta FC |

| 40 | Chicago FC |

| 50 | Toronto FC |

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

Matches table:

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

| match\_id | host\_team | guest\_team | host\_goals | guest\_goals |

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

| 1 | 10 | 20 | 3 | 0 |

| 2 | 30 | 10 | 2 | 2 |

| 3 | 10 | 50 | 5 | 1 |

| 4 | 20 | 30 | 1 | 0 |

| 5 | 50 | 30 | 1 | 0 |

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

Result table:

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

| team\_id | team\_name | num\_points |

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

| 10 | Leetcode FC | 7 |

| 20 | NewYork FC | 3 |

| 50 | Toronto FC | 3 |

| 30 | Atlanta FC | 1 |

| 40 | Chicago FC | 0 |

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

select team\_id, team\_name, ifnull(sum(point), 0) num\_points

from Teams

left join (

                select host\_team id, case

                                            when host\_goals = guest\_goals then 1

                                            when host\_goals > guest\_goals then 3

                                            else 0

                                    end as 'point'

                from Matches

                union all

                select guest\_team id, case

                                            when host\_goals = guest\_goals then 1

                                            when host\_goals > guest\_goals then 0

                                            else 3

                                    end as 'point'

                from Matches

            ) b on id = team\_id

group by team\_id

order by num\_points desc, team\_id asc

### [1225. Report Contiguous Dates](https://leetcode-cn.com/problems/report-contiguous-dates/)★★★

Table: Failed

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

| Column Name | Type |

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

| fail\_date | date |

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

Primary key for this table is fail\_date.

Failed table contains the days of failed tasks.

Table: Succeeded

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

| Column Name | Type |

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

| success\_date | date |

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

Primary key for this table is success\_date.

Succeeded table contains the days of succeeded tasks.

A system is running one task **every day**. Every task is independent of the previous tasks. The tasks can fail or succeed.

Write an SQL query to generate a report of period\_state for each continuous interval of days in the period from **2019-01-01** to **2019-12-31**.

period\_state is 'failed' if tasks in this interval failed or 'succeeded' if tasks in this interval succeeded. Interval of days are retrieved as start\_date and end\_date.

Order result by start\_date.

The query result format is in the following example:

Failed table:

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

| fail\_date |

| 2018-12-28 |

| 2018-12-29 |

| 2019-01-04 |

| 2019-01-05 |

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

Succeeded table:

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

| success\_date |

| 2018-12-30 |

| 2018-12-31 |

| 2019-01-01 |

| 2019-01-02 |

| 2019-01-03 |

| 2019-01-06 |

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

Result table:

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

| period\_state | start\_date | end\_date |

| succeeded | 2019-01-01 | 2019-01-03 |

| failed | 2019-01-04 | 2019-01-05 |

| succeeded | 2019-01-06 | 2019-01-06 |

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

SELECT  CASE

            WHEN tag = 0 THEN 'failed'

            ELSE 'succeeded'

        END as period\_state,

        MIN(date) as start\_date,

        MAX(date) as end\_date

FROM (

        SELECT \*,

                @group := IF(@prev = tag, @group, @group+1) group\_id,

                @prev := tag

        FROM (

                SELECT fail\_date as date, 0 as tag

                FROM Failed

                UNION

                SELECT success\_date as date, 1 as tag

                FROM Succeeded

             ) a

        JOIN (SELECT @group := -1, @prev := -1) b

        WHERE date LIKE '2019%'

        ORDER BY date ASC

    ) t

GROUP BY group\_id

ORDER BY start\_date

### [1241. Number of Comments per Post](https://leetcode-cn.com/problems/number-of-comments-per-post/)

Table: Submissions

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

| Column Name | Type |

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

| sub\_id | int |

| parent\_id | int |

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

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

Each row can be a post or comment on the post.

parent\_id is null for posts.

parent\_id for comments is sub\_id for another post in the table.

Write an SQL query to find number of comments per each post.

Result table should contain post\_id and its corresponding number\_of\_comments, and must be sorted by post\_id in ascending order.

Submissions may contain duplicate comments. You should count the number of **unique comments** per post.

Submissions may contain duplicate posts. You should treat them as one post.

The query result format is in the following example:

Submissions table:

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

| sub\_id | parent\_id |

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

| 1 | Null |

| 2 | Null |

| 1 | Null |

| 12 | Null |

| 3 | 1 |

| 5 | 2 |

| 3 | 1 |

| 4 | 1 |

| 9 | 1 |

| 10 | 2 |

| 6 | 7 |

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

Result table:

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

| post\_id | number\_of\_comments |

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

| 1 | 3 |

| 2 | 2 |

| 12 | 0 |

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

The post with id 1 has three comments in the table with id 3, 4 and 9. The comment with id 3 is repeated in the table, we counted it **only once**.

The post with id 2 has two comments in the table with id 5 and 10.

The post with id 12 has no comments in the table.

The comment with id 6 is a comment on a deleted post with id 7 so we ignored it.

select a.sub\_id post\_id, count(distinct b.sub\_id) number\_of\_comments

from (

        select distinct sub\_id

        from Submissions

        where parent\_id is null

     ) a

left join Submissions b on a.sub\_id = b.parent\_id

group by a.sub\_id

### [1251. Average Selling Price](https://leetcode-cn.com/problems/average-selling-price/)

+Table: Prices

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

| Column Name | Type |

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

| product\_id | int |

| start\_date | date |

| end\_date | date |

| price | int |

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

(product\_id, start\_date, end\_date) is the primary key for this table.

Each row of this table indicates the price of the product\_id in the period from start\_date to end\_date.

For each product\_id there will be no two overlapping periods. That means there will be no two intersecting periods for the same product\_id.

Table: UnitsSold

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

| Column Name | Type |

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

| product\_id | int |

| purchase\_date | date |

| units | int |

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

There is no primary key for this table, it may contain duplicates.

Each row of this table indicates the date, units and product\_id of each product sold.

Write an SQL query to find the average selling price for each product.

average\_price should be **rounded to 2 decimal places**.

The query result format is in the following example:

Prices table:

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

| product\_id | start\_date | end\_date | price |

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

| 1 | 2019-02-17 | 2019-02-28 | 5 |

| 1 | 2019-03-01 | 2019-03-22 | 20 |

| 2 | 2019-02-01 | 2019-02-20 | 15 |

| 2 | 2019-02-21 | 2019-03-31 | 30 |

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

UnitsSold table:

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

| product\_id | purchase\_date | units |

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

| 1 | 2019-02-25 | 100 |

| 1 | 2019-03-01 | 15 |

| 2 | 2019-02-10 | 200 |

| 2 | 2019-03-22 | 30 |

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

Result table:

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

| product\_id | average\_price |

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

| 1 | 6.96 |

| 2 | 16.96 |

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

Average selling price = Total Price of Product / Number of products sold.

Average selling price for product 1 = ((100 \* 5) + (15 \* 20)) / 115 = 6.96

Average selling price for product 2 = ((200 \* 15) + (30 \* 30)) / 230 = 16.96

select a.product\_id, round(sum(units\*price)/sum(units), 2)average\_price

from UnitsSold a

join Prices b on a.product\_id = b.product\_id

and a.purchase\_date between start\_date and end\_date

group by a.product\_id

### [1264. Page Recommendations](https://leetcode-cn.com/problems/page-recommendations/)

Table: Friendship

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

| Column Name | Type |

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

| user1\_id | int |

| user2\_id | int |

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

(user1\_id, user2\_id) is the primary key for this table.

Each row of this table indicates that there is a friendship relation between user1\_id and user2\_id.

Table: Likes

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

| Column Name | Type |

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

| user\_id | int |

| page\_id | int |

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

(user\_id, page\_id) is the primary key for this table.

Each row of this table indicates that user\_id likes page\_id.

Write an SQL query to recommend pages to the user with user\_id = 1 using the pages that your friends liked. It should not recommend pages you already liked.

Return result table in any order without duplicates.

The query result format is in the following example:

Friendship table:

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

| user1\_id | user2\_id |

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

| 1 | 2 |

| 1 | 3 |

| 1 | 4 |

| 2 | 3 |

| 2 | 4 |

| 2 | 5 |

| 6 | 1 |

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

Likes table:

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

| user\_id | page\_id |

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

| 1 | 88 |

| 2 | 23 |

| 3 | 24 |

| 4 | 56 |

| 5 | 11 |

| 6 | 33 |

| 2 | 77 |

| 3 | 77 |

| 6 | 88 |

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

Result table:

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

| recommended\_page |

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

| 23 |

| 24 |

| 56 |

| 33 |

| 77 |

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

User one is friend with users 2, 3, 4 and 6.

Suggested pages are 23 from user 2, 24 from user 3, 56 from user 3 and 33 from user 6.

Page 77 is suggested from both user 2 and user 3.

Page 88 is not suggested because user 1 already likes it.

select distinct page\_id recommended\_page

from Likes

where user\_id in (

                    select user2\_id from Friendship where user1\_id = 1

                    union

                    select user1\_id from Friendship where user2\_id = 1

                 )

     and page\_id not in (

                            select page\_id

                            from Likes

                            where user\_id = 1

                        )

### [1270. All People Report to the Given Manager](https://leetcode-cn.com/problems/all-people-report-to-the-given-manager/)

Table: Employees

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

| Column Name | Type |

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

| employee\_id | int |

| employee\_name | varchar |

| manager\_id | int |

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

employee\_id is the primary key for this table.

Each row of this table indicates that the employee with ID employee\_id and name employee\_name reports his work to his/her direct manager with manager\_id

The head of the company is the employee with employee\_id = 1.

Write an SQL query to find employee\_id of all employees that directly or indirectly report their work to the head of the company.

The indirect relation between managers will not exceed 3 managers as the company is small.

Return result table in any order without duplicates.

The query result format is in the following example:

Employees table:

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

| employee\_id | employee\_name | manager\_id |

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

| 1 | Boss | 1 |

| 3 | Alice | 3 |

| 2 | Bob | 1 |

| 4 | Daniel | 2 |

| 7 | Luis | 4 |

| 8 | Jhon | 3 |

| 9 | Angela | 8 |

| 77 | Robert | 1 |

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

Result table:

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

| employee\_id |

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

| 2 |

| 77 |

| 4 |

| 7 |

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

The head of the company is the employee with employee\_id 1.

The employees with employee\_id 2 and 77 report their work directly to the head of the company.

The employee with employee\_id 4 report his work indirectly to the head of the company 4 --> 2 --> 1.

The employee with employee\_id 7 report his work indirectly to the head of the company 7 --> 4 --> 2 --> 1.

The employees with employee\_id 3, 8 and 9 don't report their work to head of company directly or indirectly.

select a.employee\_id

from Employees a

left join Employees b on a.manager\_id = b.employee\_id

left join Employees c on b.manager\_id = c.employee\_id

left join Employees d on c.manager\_id = d.employee\_id

where d.employee\_id = 1 and a.employee\_id <> 1

### [1280. Students and Examinations](https://leetcode-cn.com/problems/students-and-examinations/)

Table: Students

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

| Column Name | Type |

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

| student\_id | int |

| student\_name | varchar |

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

student\_id is the primary key for this table.

Each row of this table contains the ID and the name of one student in the school.

Table: Subjects

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

| Column Name | Type |

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

| subject\_name | varchar |

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

subject\_name is the primary key for this table.

Each row of this table contains the name of one subject in the school.

Table: Examinations

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

| Column Name | Type |

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

| student\_id | int |

| subject\_name | varchar |

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

There is no primary key for this table. It may contain duplicates.

Each student from the Students table takes every course from Subjects table.

Each row of this table indicates that a student with ID student\_id attended the exam of subject\_name.

Write an SQL query to find the number of times each student attended each exam.

Order the result table by student\_id and subject\_name.

The query result format is in the following example:

Students table:

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

| student\_id | student\_name |

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

| 1 | Alice |

| 2 | Bob |

| 13 | John |

| 6 | Alex |

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

Subjects table:

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

| subject\_name |

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

| Math |

| Physics |

| Programming |

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

Examinations table:

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

| student\_id | subject\_name |

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

| 1 | Math |

| 1 | Physics |

| 1 | Programming |

| 2 | Programming |

| 1 | Physics |

| 1 | Math |

| 13 | Math |

| 13 | Programming |

| 13 | Physics |

| 2 | Math |

| 1 | Math |

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

Result table:

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

| student\_id | student\_name | subject\_name | attended\_exams |

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

| 1 | Alice | Math | 3 |

| 1 | Alice | Physics | 2 |

| 1 | Alice | Programming | 1 |

| 2 | Bob | Math | 1 |

| 2 | Bob | Physics | 0 |

| 2 | Bob | Programming | 1 |

| 6 | Alex | Math | 0 |

| 6 | Alex | Physics | 0 |

| 6 | Alex | Programming | 0 |

| 13 | John | Math | 1 |

| 13 | John | Physics | 1 |

| 13 | John | Programming | 1 |

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

The result table should contain all students and all subjects.

Alice attended Math exam 3 times, Physics exam 2 times and Programming exam 1 time.

Bob attended Math exam 1 time, Programming exam 1 time and didn't attend the Physics exam.

Alex didn't attend any exam.

John attended Math exam 1 time, Physics exam 1 time and Programming exam 1 time.

select a.student\_id, student\_name, b.subject\_name,

count(c.student\_id) attended\_exams

from Students a

join Subjects b

left join Examinations c on a.student\_id = c.student\_id

and b.subject\_name = c.subject\_name

group by a.student\_id, a.student\_name, b.subject\_name

order by a.student\_id, student\_name desc

### [1285. Find the Start and End Number of Continuous Ranges](https://leetcode-cn.com/problems/find-the-start-and-end-number-of-continuous-ranges/)★

Table: Logs

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

| Column Name | Type |

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

| log\_id | int |

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

id is the primary key for this table.

Each row of this table contains the ID in a log Table.

Since some IDs have been removed from Logs. Write an SQL query to find the start and end number of continuous ranges in table Logs.

Order the result table by start\_id.

The query result format is in the following example:

Logs table:

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

| log\_id |

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

| 1 |

| 2 |

| 3 |

| 7 |

| 8 |

| 10 |

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

Result table:

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

| start\_id | end\_id |

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

| 1 | 3 |

| 7 | 8 |

| 10 | 10 |

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

The result table should contain all ranges in table Logs.

From 1 to 3 is contained in the table.

From 4 to 6 is missing in the table

From 7 to 8 is contained in the table.

Number 9 is missing in the table.

Number 10 is contained in the table.

SELECT min(log\_id) start\_id, max(log\_id) end\_id

FROM (

        SELECT log\_id, CASE

                            WHEN @id = log\_id - 1 THEN @num := @num

                            ELSE @num := @num + 1

                       END num

                       , @id := log\_id

        FROM LOGS

        JOIN (SELECT @num := 0, @id := NULL) a

     ) x

GROUP BY num

SELECT MIN(log\_id) start\_id, MAX(log\_id) end\_id

FROM (

         SELECT

            log\_id, log\_id - row\_number() OVER(ORDER BY log\_id) as num

        FROM Logs

     ) t

GROUP BY num

### [1294. Weather Type in Each Country](https://leetcode-cn.com/problems/weather-type-in-each-country/)

Table: Countries

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

| Column Name | Type |

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

| country\_id | int |

| country\_name | varchar |

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

country\_id is the primary key for this table.

Each row of this table contains the ID and the name of one country.

Table: Weather

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

| Column Name | Type |

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

| country\_id | int |

| weather\_state | varchar |

| day | date |

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

(country\_id, day) is the primary key for this table.

Each row of this table indicates the weather state in a country for one day.

Write an SQL query to find the type of weather in each country for November 2019.

The type of weather is **Cold** if the average weather\_state is less than or equal 15, **Hot** if the average weather\_state is greater than or equal 25 and **Warm** otherwise.

Return result table in any order.

The query result format is in the following example:

Countries table:

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

| country\_id | country\_name |

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

| 2 | USA |

| 3 | Australia |

| 7 | Peru |

| 5 | China |

| 8 | Morocco |

| 9 | Spain |

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

Weather table:

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

| country\_id | weather\_state | day |

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

| 2 | 15 | 2019-11-01 |

| 2 | 12 | 2019-10-28 |

| 2 | 12 | 2019-10-27 |

| 3 | -2 | 2019-11-10 |

| 3 | 0 | 2019-11-11 |

| 3 | 3 | 2019-11-12 |

| 5 | 16 | 2019-11-07 |

| 5 | 18 | 2019-11-09 |

| 5 | 21 | 2019-11-23 |

| 7 | 25 | 2019-11-28 |

| 7 | 22 | 2019-12-01 |

| 7 | 20 | 2019-12-02 |

| 8 | 25 | 2019-11-05 |

| 8 | 27 | 2019-11-15 |

| 8 | 31 | 2019-11-25 |

| 9 | 7 | 2019-10-23 |

| 9 | 3 | 2019-12-23 |

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

Result table:

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

| country\_name | weather\_type |

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

| USA | Cold |

| Austraila | Cold |

| Peru | Hot |

| China | Warm |

| Morocco | Hot |

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

Average weather\_state in USA in November is (15) / 1 = 15 so weather type is Cold.

Average weather\_state in Austraila in November is (-2 + 0 + 3) / 3 = 0.333 so weather type is Cold.

Average weather\_state in Peru in November is (25) / 1 = 25 so weather type is Hot.

Average weather\_state in China in November is (16 + 18 + 21) / 3 = 18.333 so weather type is Warm.

Average weather\_state in Morocco in November is (25 + 27 + 31) / 3 = 27.667 so weather type is Hot.

We know nothing about average weather\_state in Spain in November so we don't include it in the result table.

select a.country\_name, case

                          when avg(b.weather\_state) <= 15 then 'Cold'

                          when avg(b.weather\_state) >= 25 then 'Hot'

                          else 'Warm'

                       end as weather\_type

from Countries a

join Weather b on a.country\_id = b.country\_id and b.day like '2019-11%'

group by a.country\_name

### [1303. Find the Team Size](https://leetcode-cn.com/problems/find-the-team-size/)

Table: Employee

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

| Column Name | Type |

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

| employee\_id | int |

| team\_id | int |

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

employee\_id is the primary key for this table.

Each row of this table contains the ID of each employee and their respective team.

Write an SQL query to find the team size of each of the employees.

Return result table in any order.

The query result format is in the following example:

Employee Table:

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

| employee\_id | team\_id |

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

| 1 | 8 |

| 2 | 8 |

| 3 | 8 |

| 4 | 7 |

| 5 | 9 |

| 6 | 9 |

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

Result table:

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

| employee\_id | team\_size |

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

| 1 | 3 |

| 2 | 3 |

| 3 | 3 |

| 4 | 1 |

| 5 | 2 |

| 6 | 2 |

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

Employees with Id 1,2,3 are part of a team with team\_id = 8.

Employees with Id 4 is part of a team with team\_id = 7.

Employees with Id 5,6 are part of a team with team\_id = 9.

select employee\_id, count(\*) over(partition by team\_id) team\_size

from employee

### [1308. Running Total for Different Genders](https://leetcode-cn.com/problems/running-total-for-different-genders/)

Table: Scores

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

| Column Name | Type |

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

| player\_name | varchar |

| gender | varchar |

| day | date |

| score\_points | int |

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

(gender, day) is the primary key for this table.

A competition is held between females team and males team.

Each row of this table indicates that a player\_name and with gender has scored score\_point in someday.

Gender is 'F' if the player is in females team and 'M' if the player is in males team.

Write an SQL query to find the total score for each gender at each day.

Order the result table by gender and day

The query result format is in the following example:

Scores table:

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

| player\_name | gender | day | score\_points |

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

| Aron | F | 2020-01-01 | 17 |

| Alice | F | 2020-01-07 | 23 |

| Bajrang | M | 2020-01-07 | 7 |

| Khali | M | 2019-12-25 | 11 |

| Slaman | M | 2019-12-30 | 13 |

| Joe | M | 2019-12-31 | 3 |

| Jose | M | 2019-12-18 | 2 |

| Priya | F | 2019-12-31 | 23 |

| Priyanka | F | 2019-12-30 | 17 |

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

Result table:

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

| gender | day | total |

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

| F | 2019-12-30 | 17 |

| F | 2019-12-31 | 40 |

| F | 2020-01-01 | 57 |

| F | 2020-01-07 | 80 |

| M | 2019-12-18 | 2 |

| M | 2019-12-25 | 13 |

| M | 2019-12-30 | 26 |

| M | 2019-12-31 | 29 |

| M | 2020-01-07 | 36 |

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

For females team:

First day is 2019-12-30, Priyanka scored 17 points and the total score for the team is 17.

Second day is 2019-12-31, Priya scored 23 points and the total score for the team is 40.

Third day is 2020-01-01, Aron scored 17 points and the total score for the team is 57.

Fourth day is 2020-01-07, Alice scored 23 points and the total score for the team is 80.

For males team:

First day is 2019-12-18, Jose scored 2 points and the total score for the team is 2.

Second day is 2019-12-25, Khali scored 11 points and the total score for the team is 13.

Third day is 2019-12-30, Slaman scored 13 points and the total score for the team is 26.

Fourth day is 2019-12-31, Joe scored 3 points and the total score for the team is 29.

Fifth day is 2020-01-07, Bajrang scored 7 points and the total score for the team is 36.

select gender, day,

sum(score\_points) over(partition by gender order by day) as total

from Scores

order by gender, day

### [1321. Restaurant Growth](https://leetcode-cn.com/problems/restaurant-growth/)

Table: Customer

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

| Column Name | Type |

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

| customer\_id | int |

| name | varchar |

| visited\_on | date |

| amount | int |

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

(customer\_id, visited\_on) is the primary key for this table.

This table contains data about customer transactions in a restaurant.

visited\_on is the date on which the customer with ID (customer\_id) have visited the restaurant.

amount is the total paid by a customer.

You are the restaurant owner and you want to analyze a possible expansion (there will be at least one customer every day).

Write an SQL query to compute moving average of how much customer paid in a 7 days window (current day + 6 days before) .

The query result format is in the following example:

Return result table ordered by visited\_on.

average\_amount should be **rounded to 2 decimal places**, all dates are in the format ('YYYY-MM-DD').

Customer table:

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

| customer\_id | name | visited\_on | amount |

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

| 1 | Jhon | 2019-01-01 | 100 |

| 2 | Daniel | 2019-01-02 | 110 |

| 3 | Jade | 2019-01-03 | 120 |

| 4 | Khaled | 2019-01-04 | 130 |

| 5 | Winston | 2019-01-05 | 110 |

| 6 | Elvis | 2019-01-06 | 140 |

| 7 | Anna | 2019-01-07 | 150 |

| 8 | Maria | 2019-01-08 | 80 |

| 9 | Jaze | 2019-01-09 | 110 |

| 1 | Jhon | 2019-01-10 | 130 |

| 3 | Jade | 2019-01-10 | 150 |

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

Result table:

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

| visited\_on | amount | average\_amount |

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

| 2019-01-07 | 860 | 122.86 |

| 2019-01-08 | 840 | 120 |

| 2019-01-09 | 840 | 120 |

| 2019-01-10 | 1000 | 142.86 |

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

1st moving average from 2019-01-01 to 2019-01-07 has an average\_amount of (100 + 110 + 120 + 130 + 110 + 140 + 150)/7 = 122.86

2nd moving average from 2019-01-02 to 2019-01-08 has an average\_amount of (110 + 120 + 130 + 110 + 140 + 150 + 80)/7 = 120

3rd moving average from 2019-01-03 to 2019-01-09 has an average\_amount of (120 + 130 + 110 + 140 + 150 + 80 + 110)/7 = 120

4th moving average from 2019-01-04 to 2019-01-10 has an average\_amount of (130 + 110 + 140 + 150 + 80 + 110 + 130 + 150)/7 = 142.86

select a.visited\_on, sum(b.amount) amount,

round(sum(b.amount)/7, 2) average\_amount

from (

         select distinct visited\_on

         from Customer

         where visited\_on >= (select min(visited\_on) from Customer) + 6

     ) as a

join Customer b on datediff(a.visited\_on , b.visited\_on)between 0 and 6

group by a.visited\_on

select distinct visited\_on, sum\_amount as amount,

round(average\_amount, 2) as average\_amount

from (

        select visited\_on,

            sum(amount) over(order by visited\_on rows 6 preceding) as sum\_amount,

            avg(amount) over(order by visited\_on rows 6 preceding) as average\_amount

        from (

                select visited\_on, sum(amount) as amount

                from Customer

                group by visited\_on

             ) t1

    ) t2

where datediff(visited\_on, (select min(visited\_on) from Customer)) >= 6

### [1322. Ads Performance](https://leetcode-cn.com/problems/ads-performance/)

Table: Ads

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

| Column Name | Type |

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

| ad\_id | int |

| user\_id | int |

| action | enum |

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

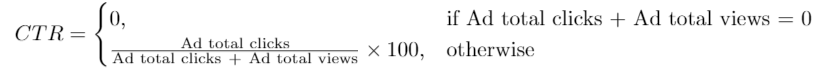
(ad\_id, user\_id) is the primary key for this table.

Each row of this table contains the ID of an Ad, the ID of a user and the action taken by this user regarding this Ad.

The action column is an ENUM type of ('Clicked', 'Viewed', 'Ignored').

A company is running Ads and wants to calculate the performance of each Ad.

Performance of the Ad is measured using Click-Through Rate (CTR) where:



Write an SQL query to find the ctr of each Ad.

**Round** ctr to 2 decimal points. **Order** the result table by ctr in descending order and by ad\_id in ascending order in case of a tie.

The query result format is in the following example:

Ads table:

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

| ad\_id | user\_id | action |

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

| 1 | 1 | Clicked |

| 2 | 2 | Clicked |

| 3 | 3 | Viewed |

| 5 | 5 | Ignored |

| 1 | 7 | Ignored |

| 2 | 7 | Viewed |

| 3 | 5 | Clicked |

| 1 | 4 | Viewed |

| 2 | 11 | Viewed |

| 1 | 2 | Clicked |

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

Result table:

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

| ad\_id | ctr |

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

| 1 | 66.67 |

| 3 | 50.00 |

| 2 | 33.33 |

| 5 | 0.00 |

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

for ad\_id = 1, ctr = (2/(2+1)) \* 100 = 66.67

for ad\_id = 2, ctr = (1/(1+2)) \* 100 = 33.33

for ad\_id = 3, ctr = (1/(1+1)) \* 100 = 50.00

for ad\_id = 5, ctr = 0.00, Note that ad\_id = 5 has no clicks or views.

Note that we don't care about Ignored Ads.

Result table is ordered by the ctr. in case of a tie we order them by ad\_id

select ad\_id, round(ifnull(sum(action = 'Clicked')

/ (sum(action = 'Clicked') + sum(action = 'Viewed'))\*100, 0), 2) ctr

from Ads

group by ad\_id

order by ctr desc, ad\_id

### [1327. List the Products Ordered in a Period](https://leetcode-cn.com/problems/list-the-products-ordered-in-a-period/)

Table: Products

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

| Column Name | Type |

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

| product\_id | int |

| product\_name | varchar |

| product\_category | varchar |

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

product\_id is the primary key for this table.

This table contains data about the company's products.

Table: Orders

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

| Column Name | Type |

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

| product\_id | int |

| order\_date | date |

| unit | int |

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

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

product\_id is a foreign key to Products table.

unit is the number of products ordered in order\_date.

Write an SQL query to get the names of products with greater than or equal to 100 units ordered in February 2020 and their amount.

Return result table in any order.

The query result format is in the following example:

Products table:

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

| product\_id | product\_name | product\_category |

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

| 1 | Leetcode Solutions | Book |

| 2 | Jewels of Stringology | Book |

| 3 | HP | Laptop |

| 4 | Lenovo | Laptop |

| 5 | Leetcode Kit | T-shirt |

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

Orders table:

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

| product\_id | order\_date | unit |

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

| 1 | 2020-02-05 | 60 |

| 1 | 2020-02-10 | 70 |

| 2 | 2020-01-18 | 30 |

| 2 | 2020-02-11 | 80 |

| 3 | 2020-02-17 | 2 |

| 3 | 2020-02-24 | 3 |

| 4 | 2020-03-01 | 20 |

| 4 | 2020-03-04 | 30 |

| 4 | 2020-03-04 | 60 |

| 5 | 2020-02-25 | 50 |

| 5 | 2020-02-27 | 50 |

| 5 | 2020-03-01 | 50 |

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

Result table:

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

| product\_name | unit |

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

| Leetcode Solutions | 130 |

| Leetcode Kit | 100 |

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

Products with product\_id = 1 is ordered in February a total of (60 + 70) = 130.

Products with product\_id = 2 is ordered in February a total of 80.

Products with product\_id = 3 is ordered in February a total of (2 + 3) = 5.

Products with product\_id = 4 was not ordered in February 2020.

Products with product\_id = 5 is ordered in February a total of (50 + 50) = 100.

select product\_name, sum(unit) unit

from Products a

join Orders b

on a.product\_id = b.product\_id and order\_date like '2020-02%'

group by a.product\_id

having sum(unit) >= 100

### [1336. Number of Transactions per Visit](https://leetcode-cn.com/problems/number-of-transactions-per-visit/)●●

Table: Visits

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

| Column Name | Type |

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

| user\_id | int |

| visit\_date | date |

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

(user\_id, visit\_date) is the primary key for this table.

Each row of this table indicates that user\_id has visited the bank in visit\_date.

Table: Transactions

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

| Column Name | Type |

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

| user\_id | int |

| transaction\_date | date |

| amount | int |

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

There is no primary key for this table, it may contain duplicates.

Each row of this table indicates that user\_id has done a transaction of amount in transaction\_date.

It is guaranteed that the user has visited the bank in the transaction\_date.(i.e The Visits table contains (user\_id, transaction\_date) in one row)

A bank wants to draw a chart of the number of transactions bank visitors did in one visit to the bank and the corresponding number of visitors who have done this number of transaction in one visit.

Write an SQL query to find how many users visited the bank and didn't do any transactions, how many visited the bank and did one transaction and so on.

The result table will contain two columns:

* transactions\_count which is the number of transactions done in one visit.
* visits\_count which is the corresponding number of users who did transactions\_count in one visit to the bank.

transactions\_count should take all values from 0 to max(transactions\_count) done by one or more users.

Order the result table by transactions\_count.

The query result format is in the following example:

Visits table:

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

| user\_id | visit\_date |

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

| 1 | 2020-01-01 |

| 2 | 2020-01-02 |

| 12 | 2020-01-01 |

| 19 | 2020-01-03 |

| 1 | 2020-01-02 |

| 2 | 2020-01-03 |

| 1 | 2020-01-04 |

| 7 | 2020-01-11 |

| 9 | 2020-01-25 |

| 8 | 2020-01-28 |

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

Transactions table:

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

| user\_id | transaction\_date | amount |

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

| 1 | 2020-01-02 | 120 |

| 2 | 2020-01-03 | 22 |

| 7 | 2020-01-11 | 232 |

| 1 | 2020-01-04 | 7 |

| 9 | 2020-01-25 | 33 |

| 9 | 2020-01-25 | 66 |

| 8 | 2020-01-28 | 1 |

| 9 | 2020-01-25 | 99 |

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

Result table:

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

| transactions\_count | visits\_count |

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

| 0 | 4 |

| 1 | 5 |

| 2 | 0 |

| 3 | 1 |

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

\* For transactions\_count = 0, The visits (1, "2020-01-01"), (2, "2020-01-02"), (12, "2020-01-01") and (19, "2020-01-03") did no transactions so visits\_count = 4.

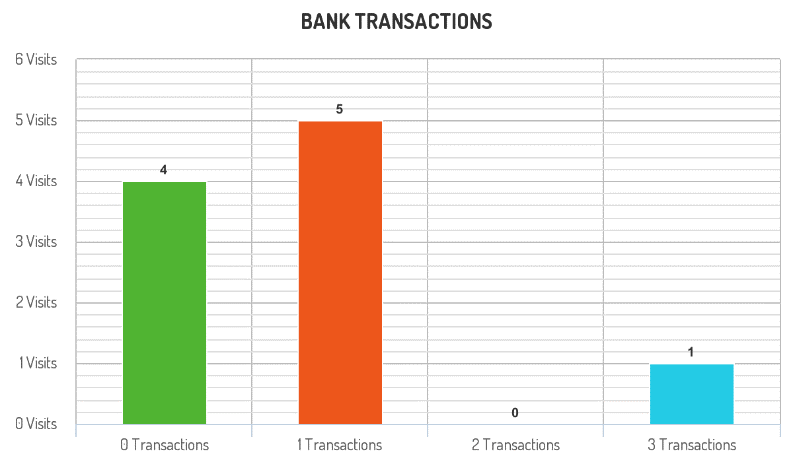
\* For transactions\_count = 1, The visits (2, "2020-01-03"), (7, "2020-01-11"), (8, "2020-01-28"), (1, "2020-01-02") and (1, "2020-01-04") did one transaction so visits\_count = 5.

\* For transactions\_count = 2, No customers visited the bank and did two transactions so visits\_count = 0.

\* For transactions\_count = 3, The visit (9, "2020-01-25") did three transactions so visits\_count = 1.

\* For transactions\_count >= 4, No customers visited the bank and did more than three transactions so we will stop at transactions\_count = 3

The chart drawn for this example is as follows:



select ceil(idx) transactions\_count, ifnull(visits\_count, 0) visits\_count

from (

        select 0 idx

        union all

        select @i := @i + 1

        from transactions

        join (select @i := 0) val

        where @i < (

            select count(\*) transactions\_count

            from transactions

            group by user\_id, transaction\_date

            order by transactions\_count desc

            limit 1

        )

    ) tmp1

left join (

             select transactions\_count, count(\*) visits\_count

             from (

                 select count(t.user\_id) transactions\_count

                 from visits v left join transactions t

                 on v.user\_id = t.user\_id and visit\_date = transaction\_date

                 group by v.user\_id, visit\_date

             ) tmp

             group by transactions\_count

          ) tmp2 on idx = transactions\_count

with recursive t(n) as (

                          select 0

                          union all

                          select n+1

                          from t where n < (

                                select max(transaction\_count)

                                from (

                                      select v.user\_id, v.visit\_date,

count(tr.amount) transaction\_count

                                      from visits v

                                      left join transactions tr

on v.user\_id=tr.user\_id

and v.visit\_date=tr.transaction\_date

                                      group by v.user\_id,v.visit\_date

                                   ) a

                           )

                        ),

tmp as(

        select v.user\_id, v.visit\_date, count(t.amount) transaction\_count

        from visits v

        left join transactions t on v.user\_id = t.user\_id

and v.visit\_date=t.transaction\_date

        group by v.user\_id,v.visit\_date

      )

select n transactions\_count, ifnull(visit\_count,0) visits\_count

from t

left join (

             select transaction\_count, count(\*) visit\_count from tmp

             group by transaction\_count

          ) b on t.n = b.transaction\_count

### [1341. Movie Rating](https://leetcode-cn.com/problems/movie-rating/)

Table: Movies

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

| Column Name | Type |

| movie\_id | int |

| title | varchar |

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

movie\_id is the primary key for this table.

title is the name of the movie.

Table: Users

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

| Column Name | Type |

| user\_id | int |

| name | varchar |

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

user\_id is the primary key for this table.

Table: Movie\_Rating

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

| Column Name | Type |

| movie\_id | int |

| user\_id | int |

| rating | int |

| created\_at | date |

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

(movie\_id, user\_id) is the primary key for this table.

This table contains the rating of a movie by a user in their review.

created\_at is the user's review date.

Write the following SQL query:

* Find the name of the user who has rated the greatest number of movies.

In case of a tie, return lexicographically smaller user name.

* Find the movie name with the ***highest average*** rating in **February 2020**.

In case of a tie, return lexicographically smaller movie name.

The query is returned in 2 rows, the query result format is in the following example:

Movies table:

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

| movie\_id | title |

| 1 | Avengers |

| 2 | Frozen 2 |

| 3 | Joker |

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

Users table:

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

| user\_id | name |

| 1 | Daniel |

| 2 | Monica |

| 3 | Maria |

| 4 | James |

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

Movie\_Rating table:

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

| movie\_id | user\_id | rating | created\_at |

| 1 | 1 | 3 | 2020-01-12 |

| 1 | 2 | 4 | 2020-02-11 |

| 1 | 3 | 2 | 2020-02-12 |

| 1 | 4 | 1 | 2020-01-01 |

| 2 | 1 | 5 | 2020-02-17 |

| 2 | 2 | 2 | 2020-02-01 |

| 2 | 3 | 2 | 2020-03-01 |

| 3 | 1 | 3 | 2020-02-22 |

| 3 | 2 | 4 | 2020-02-25 |

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

Result table:

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

| results |

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

| Daniel |

| Frozen 2 |

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

Daniel and Monica have rated 3 movies ("Avengers", "Frozen 2" and "Joker") but Daniel is smaller lexicographically.

Frozen 2 and Joker have a rating average of 3.5 in February but Frozen 2 is smaller lexicographically.

(

    select name results

    from Movie\_Rating a

    join Users b on a.user\_id = b.user\_id

    group by a.user\_id, name

    order by count(\*) desc, name

    limit 0, 1

)

union

(

    select title results

    from Movie\_Rating a

    join Movies b on a.movie\_id = b.movie\_id

    where created\_at like '2020-02%'

    group by a.movie\_id, title

    order by avg(rating) desc, title

    limit 0, 1

)

### [1350. Students With Invalid Departments](https://leetcode-cn.com/problems/students-with-invalid-departments/)

Table: Departments

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

| Column Name | Type |

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

| id | int |

| name | varchar |

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

id is the primary key of this table.

The table has information about the id of each department of a university.

Table: Students

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

| Column Name | Type |

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

| id | int |

| name | varchar |

| department\_id | int |

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

id is the primary key of this table.

The table has information about the id of each student at a university and the id of the department he/she studies at.

Write an SQL query to find the id and the name of all students who are enrolled in departments that no longer exists.

Return the result table in any order.

The query result format is in the following example:

Departments table:

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

| id | name |

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

| 1 | Electrical Engineering |

| 7 | Computer Engineering |

| 13 | Bussiness Administration |

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

Students table:

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

| id | name | department\_id |

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

| 23 | Alice | 1 |

| 1 | Bob | 7 |

| 5 | Jennifer | 13 |

| 2 | John | 14 |

| 4 | Jasmine | 77 |

| 3 | Steve | 74 |

| 6 | Luis | 1 |

| 8 | Jonathan | 7 |

| 7 | Daiana | 33 |

| 11 | Madelynn | 1 |

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

Result table:

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

| id | name |

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

| 2 | John |

| 7 | Daiana |

| 4 | Jasmine |

| 3 | Steve |

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

John, Daiana, Steve and Jasmine are enrolled in departments 14, 33, 74 and 77 respectively. department 14, 33, 74 and 77 doesn't exist in the Departments table.

select Students.id, Students.name

from Students

left join Departments on department\_id = Departments.id

where Departments.id is null

### [1355. Activity Participants](https://leetcode-cn.com/problems/activity-participants/)

Table: Friends

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

| Column Name | Type |

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

| id | int |

| name | varchar |

| activity | varchar |

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

id is the id of the friend and primary key for this table.

name is the name of the friend.

activity is the name of the activity which the friend takes part in.

Table: Activities

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

| Column Name | Type |

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

| id | int |

| name | varchar |

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

id is the primary key for this table.

name is the name of the activity.

Write an SQL query to find the names of all the activities with neither maximum, nor minimum number of participants.

Return the result table in any order. Each activity in table Activities is performed by any person in the table Friends.

The query result format is in the following example:

Friends table:

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

| id | name | activity |

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

| 1 | Jonathan D. | Eating |

| 2 | Jade W. | Singing |

| 3 | Victor J. | Singing |

| 4 | Elvis Q. | Eating |

| 5 | Daniel A. | Eating |

| 6 | Bob B. | Horse Riding |

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

Activities table:

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

| id | name |

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

| 1 | Eating |

| 2 | Singing |

| 3 | Horse Riding |

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

Result table:

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

| activity |

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

| Singing |

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

Eating activity is performed by 3 friends, maximum number of participants, (Jonathan D. , Elvis Q. and Daniel A.)

Horse Riding activity is performed by 1 friend, minimum number of participants, (Bob B.)

Singing is performed by 2 friends (Victor J. and Jade W.)

SELECT activity

FROM Friends

GROUP BY activity

HAVING COUNT(\*) > SOME(SELECT COUNT(\*) FROM Friends GROUP BY activity)

    and COUNT(\*) < SOME(SELECT COUNT(\*) FROM Friends GROUP BY activity)

### [1364. Number of Trusted Contacts of a Customer](https://leetcode-cn.com/problems/number-of-trusted-contacts-of-a-customer/)

Table: Customers

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

| Column Name | Type |

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

| customer\_id | int |

| customer\_name | varchar |

| email | varchar |

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

customer\_id is the primary key for this table.

Each row of this table contains the name and the email of a customer of an online shop.

Table: Contacts

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

| Column Name | Type |

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

| user\_id | id |

| contact\_name | varchar |

| contact\_email | varchar |

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

(user\_id, contact\_email) is the primary key for this table.

Each row of this table contains the name and email of one contact of customer with user\_id.

This table contains information about people each customer trust. The contact may or may not exist in the Customers table.

Table: Invoices

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

| Column Name | Type |

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

| invoice\_id | int |

| price | int |

| user\_id | int |

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

invoice\_id is the primary key for this table.

Each row of this table indicates that user\_id has an invoice with invoice\_id and a price.

Write an SQL query to find the following for each invoice\_id:

* customer\_name: The name of the customer the invoice is related to.
* price: The price of the invoice.
* contacts\_cnt: The number of contacts related to the customer.
* trusted\_contacts\_cnt: The number of contacts related to the customer and at the same time they are customers to the shop. (i.e His/Her email exists in the Customers table.)

Order the result table by invoice\_id.

The query result format is in the following example:

Customers table:

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

| customer\_id | customer\_name | email |

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

| 1 | Alice | alice@leetcode.com |

| 2 | Bob | bob@leetcode.com |

| 13 | John | john@leetcode.com |

| 6 | Alex | alex@leetcode.com |

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

Contacts table:

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

| user\_id | contact\_name | contact\_email |

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

| 1 | Bob | bob@leetcode.com |

| 1 | John | john@leetcode.com |

| 1 | Jal | jal@leetcode.com |

| 2 | Omar | omar@leetcode.com |

| 2 | Meir | meir@leetcode.com |

| 6 | Alice | alice@leetcode.com |

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

Invoices table:

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

| invoice\_id | price | user\_id |

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

| 77 | 100 | 1 |

| 88 | 200 | 1 |

| 99 | 300 | 2 |

| 66 | 400 | 2 |

| 55 | 500 | 13 |

| 44 | 60 | 6 |

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

Result table:

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

| invoice\_id | customer\_name | price | contacts\_cnt | trusted\_contacts\_cnt |

| 44 | Alex | 60 | 1 | 1 |

| 55 | John | 500 | 0 | 0 |

| 66 | Bob | 400 | 2 | 0 |

| 77 | Alice | 100 | 3 | 2 |

| 88 | Alice | 200 | 3 | 2 |

| 99 | Bob | 300 | 2 | 0 |

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

Alice has three contacts, two of them are trusted contacts (Bob and John).

Bob has two contacts, none of them is a trusted contact.

Alex has one contact and it is a trusted contact (Alice).

John doesn't have any contacts.

select invoice\_id, customer\_name, price,

count(c.contact\_name) contacts\_cnt,

       ifnull(sum(c.contact\_name in (select customer\_name from Customers))

, 0) as trusted\_contacts\_cnt

from Invoices a

join Customers b on a.user\_id = customer\_id

left join Contacts c on a.user\_id = c.user\_id

group by invoice\_id, customer\_name, price

order by invoice\_id

### [1369. Get the Second Most Recent Activity](https://leetcode-cn.com/problems/get-the-second-most-recent-activity/)

Table: UserActivity

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

| Column Name | Type |

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

| username | varchar |

| activity | varchar |

| startDate | Date |

| endDate | Date |

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

This table does not contain primary key.

This table contain information about the activity performed of each user in a period of time.

A person with username performed a activity from startDate to endDate.

Write an SQL query to show the **second most recent activity**of each user.

If the user only has one activity, return that one.

A user can't perform more than one activity at the same time. Return the result table in **any** order.

The query result format is in the following example:

UserActivity table:

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

| username | activity | startDate | endDate |

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

| Alice | Travel | 2020-02-12 | 2020-02-20 |

| Alice | Dancing | 2020-02-21 | 2020-02-23 |

| Alice | Travel | 2020-02-24 | 2020-02-28 |

| Bob | Travel | 2020-02-11 | 2020-02-18 |

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

Result table:

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

| username | activity | startDate | endDate |

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

| Alice | Dancing | 2020-02-21 | 2020-02-23 |

| Bob | Travel | 2020-02-11 | 2020-02-18 |

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

The most recent activity of Alice is Travel from 2020-02-24 to 2020-02-28, before that she was dancing from 2020-02-21 to 2020-02-23.

Bob only has one record, we just take that one.

select username, activity, startDate, endDate

from (

        select \*,rank() over (partition by username order by startDate desc) as rnk

                ,count(\*) over (partition by username) as cnt

        from UserActivity

    ) t

where cnt = 1 or rnk = 2

### [1378. Replace Employee ID With The Unique Identifier](https://leetcode-cn.com/problems/replace-employee-id-with-the-unique-identifier/)

Table: Employees

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

| Column Name | Type |

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

| id | int |

| name | varchar |

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

id is the primary key for this table.

Each row of this table contains the id and the name of an employee in a company.

Table: EmployeeUNI

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

| Column Name | Type |

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

| id | int |

| unique\_id | int |

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

(id, unique\_id) is the primary key for this table.

Each row of this table contains the id and the corresponding unique id of an employee in the company.

Write an SQL query to show the **unique ID**of each user, If a user doesn't have a unique ID replace just show null.

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

The query result format is in the following example:

Employees table:

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

| id | name |

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

| 1 | Alice |

| 7 | Bob |

| 11 | Meir |

| 90 | Winston |

| 3 | Jonathan |

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

EmployeeUNI table:

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

| id | unique\_id |

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

| 3 | 1 |

| 11 | 2 |

| 90 | 3 |

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

EmployeeUNI table:

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

| unique\_id | name |

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

| null | Alice |

| null | Bob |

| 2 | Meir |

| 3 | Winston |

| 1 | Jonathan |

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

Alice and Bob don't have a unique ID, We will show null instead.

The unique ID of Meir is 2.

The unique ID of Winston is 3.

The unique ID of Jonathan is 1.

select unique\_id, name

from Employees a

left join EmployeeUNI b on a.id = b.id

### [1384. Total Sales Amount by Year](https://leetcode-cn.com/problems/total-sales-amount-by-year/)●●

Table: Product

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

| Column Name | Type |

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

| product\_id | int |

| product\_name | varchar |

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

product\_id is the primary key for this table.

product\_name is the name of the product.

Table: Sales

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

| Column Name | Type |

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

| product\_id | int |

| period\_start | varchar |

| period\_end | date |

| average\_daily\_sales | int |

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

product\_id is the primary key for this table.

period\_start and period\_end indicates the start and end date for sales period, both dates are inclusive.

The average\_daily\_sales column holds the average daily sales amount of the items for the period.

Write an SQL query to report the Total sales amount of each item for each year, with corresponding product name, product\_id, product\_name and report\_year.

Dates of the sales years are between 2018 to 2020. Return the result table **ordered** by product\_id and report\_year.

The query result format is in the following example:

Product table:

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

| product\_id | product\_name |

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

| 1 | LC Phone |

| 2 | LC T-Shirt |

| 3 | LC Keychain |

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

Sales table:

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

| product\_id | period\_start | period\_end | average\_daily\_sales |

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

| 1 | 2019-01-25 | 2019-02-28 | 100 |

| 2 | 2018-12-01 | 2020-01-01 | 10 |

| 3 | 2019-12-01 | 2020-01-31 | 1 |

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

Result table:

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

| product\_id | product\_name | report\_year | total\_amount |

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

| 1 | LC Phone | 2019 | 3500 |

| 2 | LC T-Shirt | 2018 | 310 |

| 2 | LC T-Shirt | 2019 | 3650 |

| 2 | LC T-Shirt | 2020 | 10 |

| 3 | LC Keychain | 2019 | 31 |

| 3 | LC Keychain | 2020 | 31 |

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

LC Phone was sold for the period of 2019-01-25 to 2019-02-28, and there are 35 days for this period. Total amount 35\*100 = 3500.

LC T-shirt was sold for the period of 2018-12-01 to 2020-01-01, and there are 31, 365, 1 days for years 2018, 2019 and 2020 respectively.

LC Keychain was sold for the period of 2019-12-01 to 2020-01-31, and there are 31, 31 days for years 2019 and 2020 respectively.

select t.product\_id, product\_name, report\_year, sum(total\_amount) total\_amount

from (

        select product\_id, "2020" report\_year,

(datediff(if(period\_end < "2021-01-01", period\_end, date("2020-12-31")),

               if(period\_start > "2020-01-01", period\_start, date("2020-01-01"))) + 1)

                                          \* average\_daily\_sales total\_amount

        from Sales

        having total\_amount > 0

        union all

        select product\_id, "2019" report\_year,

(datediff(if(period\_end < "2020-01-01", period\_end, date("2019-12-31")),

               if(period\_start > "2019-01-01", period\_start, date("2019-01-01"))) + 1)

                                                \* average\_daily\_sales total\_amount

        from Sales

        having total\_amount > 0

        union all

        select product\_id, "2018" report\_year,

(datediff(if(period\_end<"2019-01-01",period\_end,date("2018-12-31")),

              if(period\_start > "2018-01-01", period\_start, date("2018-01-01"))) + 1)

                                                \* average\_daily\_sales total\_amount

        from Sales

        having total\_amount > 0

     ) t

left join product p on p.product\_id = t.product\_id

group by product\_id, report\_year

order by product\_id, report\_year

select s.PRODUCT\_ID, PRODUCT\_NAME, date\_format(bound, '%Y') REPORT\_YEAR,

        (datediff(

            if (bound < period\_end, bound, period\_end),

            if (makedate(year(bound), 1) > period\_start, makedate(year(bound), 1),

 period\_start)

         ) + 1) \* average\_daily\_sales TOTAL\_AMOUNT

from product p

join (

        select '2018-12-31' bound

        union all

        select '2019-12-31' bound

        union all

        select '2020-12-31' bound

     ) bounds

join sales s

on p.product\_id = s.product\_id and year(bound) between year(period\_start)

and year(period\_end)

order by s.product\_id, report\_year

### [1393. Capital Gain/Loss](https://leetcode-cn.com/problems/capital-gainloss/)

Table: Stocks

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

| Column Name | Type |

| stock\_name | varchar |

| operation | enum |

| operation\_day | int |

| price | int |

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

(stock\_name, day) is the primary key for this table.

The operation column is an ENUM of type ('Sell', 'Buy')

Each row of this table indicates that the stock which has stock\_name had an operation on the day operation\_day with the price.

It is guaranteed that each 'Sell' operation for a stock has a corresponding 'Buy' operation in a previous day.

Write an SQL query to report the Capital gain/loss for each stock.

The capital gain/loss of a stock is total gain or loss after buying and selling the stock one or many times.

Return the result table in any order.

The query result format is in the following example:

Stocks table:

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

| stock\_name | operation | operation\_day | price |

| Leetcode | Buy | 1 | 1000 |

| Corona Masks | Buy | 2 | 10 |

| Leetcode | Sell | 5 | 9000 |

| Handbags | Buy | 17 | 30000 |

| Corona Masks | Sell | 3 | 1010 |

| Corona Masks | Buy | 4 | 1000 |

| Corona Masks | Sell | 5 | 500 |

| Corona Masks | Buy | 6 | 1000 |

| Handbags | Sell | 29 | 7000 |

| Corona Masks | Sell | 10 | 10000 |

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

Result table:

+---------------+-------------------+

| stock\_name | capital\_gain\_loss |

| Corona Masks | 9500 |

| Leetcode | 8000 |

| Handbags | -23000 |

+---------------+-------------------+

Leetcode stock was bought at day 1 for 1000$ and was sold at day 5 for 9000$. Capital gain = 9000 - 1000 = 8000$.

Handbags stock was bought at day 17 for 30000$ and was sold at day 29 for 7000$. Capital loss = 7000 - 30000 = -23000$.

Corona Masks stock was bought at day 1 for 10$ and was sold at day 3 for 1010$. It was bought again at day 4 for 1000$ and was sold at day 5 for 500$. At last, it was bought at day 6 for 1000$ and was sold at day 10 for 10000$. Capital gain/loss is the sum of capital gains/losses for each ('Buy' --> 'Sell') operation = (1010 - 10) + (500 - 1000) + (10000 - 1000) = 1000 - 500 + 9000 = 9500$.

select stock\_name, sum(IF(operation = 'Buy',

-price, price)) capital\_gain\_loss

from Stocks

group by stock\_name

### [1398. Customers Who Bought Products A and B but Not C](https://leetcode-cn.com/problems/customers-who-bought-products-a-and-b-but-not-c/)

Table: Customers

+---------------------+---------+

| Column Name | Type |

+---------------------+---------+

| customer\_id | int |

| customer\_name | varchar |

+---------------------+---------+

customer\_id is the primary key for this table.

customer\_name is the name of the customer.

Table: Orders

+---------------+---------+

| Column Name | Type |

+---------------+---------+

| order\_id | int |

| customer\_id | int |

| product\_name | varchar |

+---------------+---------+

order\_id is the primary key for this table.

customer\_id is the id of the customer who bought the product "product\_name".

Write an SQL query to report the customer\_id and customer\_name of customers who bought products "A", "B" but did not buy the product "C" since we want to recommend them buy this product.

Return the result table **ordered** by customer\_id.

The query result format is in the following example.

Customers table:

+-------------+---------------+

| customer\_id | customer\_name |

+-------------+---------------+

| 1 | Daniel |

| 2 | Diana |

| 3 | Elizabeth |

| 4 | Jhon |

+-------------+---------------+

Orders table:

+------------+--------------+---------------+

| order\_id | customer\_id | product\_name |

+------------+--------------+---------------+

| 10 | 1 | A |

| 20 | 1 | B |

| 30 | 1 | D |

| 40 | 1 | C |

| 50 | 2 | A |

| 60 | 3 | A |

| 70 | 3 | B |

| 80 | 3 | D |

| 90 | 4 | C |

+------------+--------------+---------------+

Result table:

+-------------+---------------+

| customer\_id | customer\_name |

+-------------+---------------+

| 3 | Elizabeth |

+-------------+---------------+

Only the customer\_id with id 3 bought the product A and B but not the product C.

select a.customer\_id, customer\_name

from Orders a

join Customers b on a.customer\_id = b.customer\_id

group by a.customer\_id

having sum(product\_name = 'A') > 0

and sum(product\_name = 'B') > 0

and sum(product\_name = 'C') = 0

order by a.customer\_id

### [1407. Top Travellers](https://leetcode-cn.com/problems/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 travelled the distance "distance".

Write an SQL query to report the distance travelled by each user.

Return the result table ordered by travelled\_distance in **descending order**, if two or more users travelled the same distance, order them by their name in **ascending order**.

The query result format is in the following example.

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 |

+------+----------+----------+

Result table:

+----------+--------------------+

| name | travelled\_distance |

+----------+--------------------+

| Elvis | 450 |

| Lee | 450 |

| Bob | 317 |

| Jonathan | 312 |

| Alex | 222 |

| Alice | 120 |

| Donald | 0 |

+----------+--------------------+

Elvis and Lee travelled 450 miles, Elvis is the top traveller 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 didn't have any rides, the distance travelled by him is 0.

select name, ifnull(sum(distance), 0) travelled\_distance

from Users a

left join Rides b on a.id = b.user\_id

group by a.id

order by travelled\_distance desc, name

### [1412. Find the Quiet Students in All Exams](https://leetcode-cn.com/problems/find-the-quiet-students-in-all-exams/)★★

Table: Student

+---------------------+---------+

| Column Name | Type |

+---------------------+---------+

| student\_id | int |

| student\_name | varchar |

+---------------------+---------+

student\_id is the primary key for this table.

student\_name is the name of the student.

Table: Exam

+---------------+---------+

| Column Name | Type |

+---------------+---------+

| exam\_id | int |

| student\_id | int |

| score | int |

+---------------+---------+

(exam\_id, student\_id) is the primary key for this table.

Student with student\_id got score points in exam with id exam\_id.

A "quite" student is the one who took at least one exam and didn't score neither the high score nor the low score.

Write an SQL query to report the students (student\_id, student\_name) being "quiet" in **ALL** exams.

Don't return the student who has never taken any exam. Return the result table **ordered** by student\_id.

The query result format is in the following example.

Student table:

+-------------+---------------+

| student\_id | student\_name |

+-------------+---------------+

| 1 | Daniel |

| 2 | Jade |

| 3 | Stella |

| 4 | Jonathan |

| 5 | Will |

+-------------+---------------+

Exam table:

+------------+--------------+-----------+

| exam\_id | student\_id | score |

+------------+--------------+-----------+

| 10 | 1 | 70 |

| 10 | 2 | 80 |

| 10 | 3 | 90 |

| 20 | 1 | 80 |

| 30 | 1 | 70 |

| 30 | 3 | 80 |

| 30 | 4 | 90 |

| 40 | 1 | 60 |

| 40 | 2 | 70 |

| 40 | 4 | 80 |

+------------+--------------+-----------+

Result table:

+-------------+---------------+

| student\_id | student\_name |

+-------------+---------------+

| 2 | Jade |

+-------------+---------------+

For exam 1: Student 1 and 3 hold the lowest and high score respectively.

For exam 2: Student 1 hold both highest and lowest score.

For exam 3 and 4: Studnet 1 and 4 hold the lowest and high score respectively.

Student 2 and 5 have never got the highest or lowest in any of the exam.

Since student 5 is not taking any exam, he is excluded from the result.

So, we only return the information of Student 2.

select a.student\_id, s.student\_name

from (

       select student\_id,

           rank() over (partition by exam\_id order by score) r1,

           rank() over (partition by exam\_id order by score desc) r2

       from Exam

    ) a

join Student s on a.student\_id = s.student\_id

group by a.student\_id

having min(r1) > 1 and min(r2) > 1

order by a.student\_id

### [1421. NPV Queries](https://leetcode-cn.com/problems/npv-queries/)

Table: NPV

+---------------+---------+

| Column Name | Type |

+---------------+---------+

| id | int |

| year | int |

| npv | int |

+---------------+---------+

(id, year) is the primary key of this table.

The table has information about the id and the year of each inventory and the corresponding net present value.

Table: Queries

+---------------+---------+

| Column Name | Type |

+---------------+---------+

| id | int |

| year | int |

+---------------+---------+

(id, year) is the primary key of this table.

The table has information about the id and the year of each inventory query.

Write an SQL query to find the npv of all each query of queries table.

Return the result table in any order.

The query result format is in the following example:

NPV table:

+------+--------+--------+

| id | year | npv |

| 1 | 2018 | 100 |

| 7 | 2020 | 30 |

| 13 | 2019 | 40 |

| 1 | 2019 | 113 |

| 2 | 2008 | 121 |

| 3 | 2009 | 12 |

| 11 | 2020 | 99 |

| 7 | 2019 | 0 |

+------+--------+--------+

Queries table:

+------+--------+

| id | year |

| 1 | 2019 |

| 2 | 2008 |

| 3 | 2009 |

| 7 | 2018 |

| 7 | 2019 |

| 7 | 2020 |

| 13 | 2019 |

+------+--------+

Result table:

+------+--------+--------+

| id | year | npv |

| 1 | 2019 | 113 |

| 2 | 2008 | 121 |

| 3 | 2009 | 12 |

| 7 | 2018 | 0 |

| 7 | 2019 | 0 |

| 7 | 2020 | 30 |

| 13 | 2019 | 40 |

+------+--------+--------+

The npv value of (7, 2018) is not present in the NPV table, we consider it 0.

The npv values of all other queries can be found in the NPV table.

select a.id, a.year, ifnull(npv, 0) as npv

from Queries a

left join NPV b on a.id = b.id and a.year = b.year

### [1435. Create a Session Bar Chart](https://leetcode-cn.com/problems/create-a-session-bar-chart/)

Table: Sessions

+---------------------+---------+

| Column Name | Type |

+---------------------+---------+

| session\_id | int |

| duration | int |

+---------------------+---------+

session\_id is the primary key for this table.

duration is the time in seconds that a user has visited the application.

You want to know how long a user visits your application. You decided to create bins of "[0-5>", "[5-10>", "[10-15>" and "15 minutes or more" and count the number of sessions on it.

Write an SQL query to report the (bin, total) in **any** order.

The query result format is in the following example.

Sessions table:

+-------------+---------------+

| session\_id | duration |

+-------------+---------------+

| 1 | 30 |

| 2 | 199 |

| 3 | 299 |

| 4 | 580 |

| 5 | 1000 |

+-------------+---------------+

Result table:

+--------------+--------------+

| bin | total |

+--------------+--------------+

| [0-5> | 3 |

| [5-10> | 1 |

| [10-15> | 0 |

| 15 or more | 1 |

+--------------+--------------+

For session\_id 1, 2 and 3 have a duration greater or equal than 0 minutes and less than 5 minutes.

For session\_id 4 has a duration greater or equal than 5 minutes and less than 10 minutes.

There are no session with a duration greater or equial than 10 minutes and less than 15 minutes.

For session\_id 5 has a duration greater or equal than 15 minutes.

select '[0-5>' BIN,

sum(if(duration < 300, 1, 0)) TOTAL from Sessions

union

select '[5-10>' bin,

sum(if(300 <= duration and duration<600, 1, 0)) total from Sessions

union

select '[10-15>' bin,

sum(if(600 <= duration and duration<900, 1, 0)) total from Session

union

select '15 or more' bin,

sum(if(900 <= duration, 1, 0)) total from Sessions

### [1440. Evaluate Boolean Expression](https://leetcode-cn.com/problems/evaluate-boolean-expression/)

Table Variables:

+---------------+---------+

| Column Name | Type |

+---------------+---------+

| name | varchar |

| value | int |

+---------------+---------+

name is the primary key for this table.

This table contains the stored variables and their values.

Table Expressions:

+---------------+---------+

| Column Name | Type |

+---------------+---------+

| left\_operand | varchar |

| operator | enum |

| right\_operand | varchar |

+---------------+---------+

(left\_operand, operator, right\_operand) is the primary key for this table.

This table contains a boolean expression that should be evaluated.

operator is an enum that takes one of the values ('<', '>', '=')

The values of left\_operand and right\_operand are guaranteed to be in the Variables table.

Write an SQL query to evaluate the boolean expressions in Expressions table.

Return the result table in any order.

The query result format is in the following example.

Variables table:

+------+-------+

| name | value |

+------+-------+

| x | 66 |

| y | 77 |

+------+-------+

Expressions table:

+--------------+----------+---------------+

| left\_operand | operator | right\_operand |

+--------------+----------+---------------+

| x | > | y |

| x | < | y |

| x | = | y |

| y | > | x |

| y | < | x |

| x | = | x |

+--------------+----------+---------------+

Result table:

+--------------+----------+---------------+-------+

| left\_operand | operator | right\_operand | value |

+--------------+----------+---------------+-------+

| x | > | y | false |

| x | < | y | true |

| x | = | y | false |

| y | > | x | true |

| y | < | x | false |

| x | = | x | true |

+--------------+----------+---------------+-------+

As shown, you need find the value of each boolean exprssion in the table using the variables table.

select Expressions.\*,

case operator

            when '>' then IF(a.value > b.value, 'true', 'false')

            when '<' then IF(a.value < b.value, 'true', 'false')

            else IF(a.value = b.value, 'true', 'false')

        end as value

from Expressions

join Variables a on left\_operand = a.name

join Variables b on right\_operand = b.name

### [1445. Apples & Oranges](https://leetcode-cn.com/problems/apples-oranges/)

Table: Sales

+---------------+---------+

| Column Name | Type |

+---------------+---------+

| sale\_date | date |

| fruit | enum |

| sold\_num | int |

+---------------+---------+

(sale\_date,fruit) is the primary key for this table.

This table contains the sales of "apples" and "oranges" sold each day.

Write an SQL query to report the difference between number of **apples** and **oranges** sold each day.

Return the result table **ordered** by sale\_date in format ('YYYY-MM-DD').

The query result format is in the following example:

Sales table:

+------------+------------+-------------+

| sale\_date | fruit | sold\_num |

| 2020-05-01 | apples | 10 |

| 2020-05-01 | oranges | 8 |

| 2020-05-02 | apples | 15 |

| 2020-05-02 | oranges | 15 |

| 2020-05-03 | apples | 20 |

| 2020-05-03 | oranges | 0 |

| 2020-05-04 | apples | 15 |

| 2020-05-04 | oranges | 16 |

+------------+------------+-------------+

Result table:

+------------+--------------+

| sale\_date | diff |

| 2020-05-01 | 2 |

| 2020-05-02 | 0 |

| 2020-05-03 | 20 |

| 2020-05-04 | -1 |

+------------+--------------+

Day 2020-05-01, 10 apples and 8 oranges were sold (Difference 10 - 8 = 2).

Day 2020-05-02, 15 apples and 15 oranges were sold (Difference 15 - 15 = 0).

Day 2020-05-03, 20 apples and 0 oranges were sold (Difference 20 - 0 = 20).

Day 2020-05-04, 15 apples and 16 oranges were sold (Difference 15 - 16 = -1).

select sale\_date,

sum(IF(fruit = 'apples', sold\_num, -sold\_num)) diff

from Sales

group by sale\_date

### [1454. Active Users](https://leetcode-cn.com/problems/active-users/)★

Table Accounts:

+---------------+---------+

| Column Name | Type |

+---------------+---------+

| id | int |

| name | varchar |

+---------------+---------+

the id is the primary key for this table.

This table contains the account id and the user name of each account.

Table Logins:

+---------------+---------+

| Column Name | Type |

+---------------+---------+

| id | int |

| login\_date | date |

+---------------+---------+

There is no primary key for this table, it may contain duplicates.

This table contains the account id of the user who logged in and the login date. A user may log in multiple times in the day.

Write an SQL query to find the id and the name of active users.

Active users are those who logged in to their accounts for 5 or more consecutive days.

Return the result table **ordered** by the id.

The query result format is in the following example:

Accounts table:

+----+----------+

| id | name |

+----+----------+

| 1 | Winston |

| 7 | Jonathan |

+----+----------+

Logins table:

+----+------------+

| id | login\_date |

+----+------------+

| 7 | 2020-05-30 |

| 1 | 2020-05-30 |

| 7 | 2020-05-31 |

| 7 | 2020-06-01 |

| 7 | 2020-06-02 |

| 7 | 2020-06-02 |

| 7 | 2020-06-03 |

| 1 | 2020-06-07 |

| 7 | 2020-06-10 |

+----+------------+

Result table:

+----+----------+

| id | name |

+----+----------+

| 7 | Jonathan |

+----+----------+

User Winston with id = 1 logged in 2 times only in 2 different days, so, Winston is not an active user.

User Jonathan with id = 7 logged in 7 times in 6 different days, five of them were consecutive days, so, Jonathan is an active user.

**Follow up question:**  
Can you write a general solution if the active users are those who logged in to their accounts for n or more consecutive days?

with tbl as

(

       SELECT id, login\_date,

dense\_rank() over(partition by id order by login\_date) rnk

       FROM Logins

    )

select distinct a.id, a.name

from tbl

JOIN Accounts a ON tbl.id = a.id

GROUP BY a.id, date\_add(login\_date, interval - rnk day)

HAVING count(distinct login\_date) >= 5

select distinct L1.id, name

from Logins as L1

JOIN Logins as L2 on L1.id = L2.id

          and Datediff(L1.login\_date, L2.login\_date) BETWEEN 0 and 4

join Accounts on Accounts.id = L1.id

group by L1.id, L1.login\_date

having count(distinct L2.login\_date) = 5

### [1459. Rectangles Area](https://leetcode-cn.com/problems/rectangles-area/)

Table: Points

+---------------+---------+

| Column Name | Type |

+---------------+---------+

| id | int |

| x\_value | int |

| y\_value | int |

+---------------+---------+

id is the primary key for this table.

Each point is represented as a 2D Dimensional (x\_value, y\_value).

Write an SQL query to report of all possible rectangles which can be formed by any two points of the table.

Each row in the result contains three columns (p1, p2, area) where:

* **p1** and **p2** are the id of two opposite corners of a rectangle and p1 < p2.
* Area of this rectangle is represented by the column **area**.

Report the query in descending order by area in case of tie in ascending order by p1 and p2.

Points table:

+----------+-------------+-------------+

| id | x\_value | y\_value |

+----------+-------------+-------------+

| 1 | 2 | 8 |

| 2 | 4 | 7 |

| 3 | 2 | 10 |

+----------+-------------+-------------+

Result table:

+----------+-------------+-------------+

| p1 | p2 | area |

+----------+-------------+-------------+

| 2 | 3 | 6 |

| 1 | 2 | 2 |

+----------+-------------+-------------+

p1 should be less than p2 and area greater than 0.

p1 = 1 and p2 = 2, has an area equal to |2-4| \* |8-7| = 2.

p1 = 2 and p2 = 3, has an area equal to |4-2| \* |7-10| = 6.

p1 = 1 and p2 = 3 It's not possible because the rectangle has an area equal to 0.

select a.id p1, b.id p2,

abs(a.x\_value-b.x\_value)\*abs(a.y\_value-b.y\_value) area

from Points a

join Points b on a.id < b.id

and a.x\_value <> b.x\_value

and a.y\_value <> b.y\_value

order by area desc, p1, p2

### [1468. Calculate Salaries](https://leetcode-cn.com/problems/calculate-salaries/)★

Table Salaries:

| Column Name | Type |

| company\_id | int |

| employee\_id | int |

| employee\_name | varchar |

| salary | int |

(company\_id, employee\_id) is the primary key for this table.

This table contains the company id, the id, the name and the salary for an employee.

Write an SQL query to find the salaries of the employees after applying taxes.

The tax rate is calculated for each company based on the following criteria:

* 0% If the max salary of any employee in the company is less than 1000$.
* 24% If the max salary of any employee in the company is in the range [1000, 10000] inclusive.
* 49% If the max salary of any employee in the company is greater than 10000$.

Return the result table **in any order**. Round the salary to the nearest integer.

The query result format is in the following example:

Salaries table:

+------------+-------------+---------------+--------+

| company\_id | employee\_id | employee\_name | salary |

| 1 | 1 | Tony | 2000 |

| 1 | 2 | Pronub | 21300 |

| 1 | 3 | Tyrrox | 10800 |

| 2 | 1 | Pam | 300 |

| 2 | 7 | Bassem | 450 |

| 2 | 9 | Hermione | 700 |

| 3 | 7 | Bocaben | 100 |

| 3 | 2 | Ognjen | 2200 |

| 3 | 13 | Nyancat | 3300 |

| 3 | 15 | Morninngcat | 1866 |

+------------+-------------+---------------+--------+

Result table:

| company\_id | employee\_id | employee\_name | salary |

| 1 | 1 | Tony | 1020 |

| 1 | 2 | Pronub | 10863 |

| 1 | 3 | Tyrrox | 5508 |

| 2 | 1 | Pam | 300 |

| 2 | 7 | Bassem | 450 |

| 2 | 9 | Hermione | 700 |

| 3 | 7 | Bocaben | 76 |

| 3 | 2 | Ognjen | 1672 |

| 3 | 13 | Nyancat | 2508 |

| 3 | 15 | Morninngcat | 5911 |

+------------+-------------+---------------+--------+

For company 1, Max salary is 21300. Employees in company 1 have taxes = 49%

For company 2, Max salary is 700. Employees in company 2 have taxes = 0%

For company 3, Max salary is 7777. Employees in company 3 have taxes = 24%

The salary after taxes = salary - (taxes percentage / 100) \* salary

For example, Salary for Morninngcat (3, 15) after taxes = 7777 - 7777 \* (24 / 100) = 7777 - 1866.48 = 5910.52, which is rounded to 5911.

select company\_id, employee\_id, employee\_name,

round((case

                 when max(salary) over (partition by company\_id) < 1000 then 1

                 when max(salary) over (partition by company\_id) <= 10000 then 0.76

                 else 0.51

              end) \* salary,0) as 'salary'

from Salaries

select company\_id, employee\_id, employee\_name,

case

            when max\_salary < 1000 then salary

            when max\_salary > 10000 then round(salary\*0.51,0)

            else round(salary\*0.76,0)

       end as salary

from (

        select \*, max(salary) over(partition by company\_id) max\_salary

        from Salaries

     ) t

### [1479. Sales by Day of the Week](https://leetcode-cn.com/problems/sales-by-day-of-the-week/)★★

Table: Orders

+---------------+---------+

| Column Name | Type |

+---------------+---------+

| order\_id | int |

| customer\_id | int |

| order\_date | date |

| item\_id | varchar |

| quantity | int |

+---------------+---------+

(ordered\_id, item\_id) is the primary key for this table.

This table contains information of the orders placed.

order\_date is the date when item\_id was ordered by the customer with id customer\_id.

Table: Items

+---------------------+---------+

| Column Name | Type |

+---------------------+---------+

| item\_id | varchar |

| item\_name | varchar |

| item\_category | varchar |

+---------------------+---------+

item\_id is the primary key for this table.

item\_name is the name of the item.

item\_category is the category of the item.

You are the business owner and would like to obtain a sales report for category items and day of the week.

Write an SQL query to report how many units in each category have been ordered on each **day of the week**.

Return the result table **ordered** by category.

The query result format is in the following example:

Orders table:

+------------+--------------+-------------+--------------+-------------+

| order\_id | customer\_id | order\_date | item\_id | quantity |

+------------+--------------+-------------+--------------+-------------+

| 1 | 1 | 2020-06-01 | 1 | 10 |

| 2 | 1 | 2020-06-08 | 2 | 10 |

| 3 | 2 | 2020-06-02 | 1 | 5 |

| 4 | 3 | 2020-06-03 | 3 | 5 |

| 5 | 4 | 2020-06-04 | 4 | 1 |

| 6 | 4 | 2020-06-05 | 5 | 5 |

| 7 | 5 | 2020-06-05 | 1 | 10 |

| 8 | 5 | 2020-06-14 | 4 | 5 |

| 9 | 5 | 2020-06-21 | 3 | 5 |

+------------+--------------+-------------+--------------+-------------+

Items table:

+------------+----------------+---------------+

| item\_id | item\_name | item\_category |

+------------+----------------+---------------+

| 1 | LC Alg. Book | Book |

| 2 | LC DB. Book | Book |

| 3 | LC SmarthPhone | Phone |

| 4 | LC Phone 2020 | Phone |

| 5 | LC SmartGlass | Glasses |

| 6 | LC T-Shirt XL | T-Shirt |

+------------+----------------+---------------+

Result table:

+------------+-----------+-----------+-----------+-----------+-----------+-----------+-----------+

| Category | Monday | Tuesday | Wednesday | Thursday | Friday | Saturday | Sunday |

+------------+-----------+-----------+-----------+-----------+-----------+-----------+-----------+

| Book | 20 | 5 | 0 | 0 | 10 | 0 | 0 |

| Glasses | 0 | 0 | 0 | 0 | 5 | 0 | 0 |

| Phone | 0 | 0 | 5 | 1 | 0 | 0 | 10 |

| T-Shirt | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

+------------+-----------+-----------+-----------+-----------+-----------+-----------+-----------+

On Monday (2020-06-01, 2020-06-08) were sold a total of 20 units (10 + 10) in the category Book (ids: 1, 2).

On Tuesday (2020-06-02) were sold a total of 5 units in the category Book (ids: 1, 2).

On Wednesday (2020-06-03) were sold a total of 5 units in the category Phone (ids: 3, 4).

On Thursday (2020-06-04) were sold a total of 1 unit in the category Phone (ids: 3, 4).

On Friday (2020-06-05) were sold 10 units in the category Book (ids: 1, 2) and 5 units in Glasses (ids: 5).

On Saturday there are no items sold.

On Sunday (2020-06-14, 2020-06-21) were sold a total of 10 units (5 +5) in the category Phone (ids: 3, 4).

There are no sales of T-Shirt.

select item\_category Category,

sum(if(date\_format(order\_date,'%W')='Monday',quantity,0)) as Monday,

sum(if(date\_format(order\_date,'%W')='Tuesday',quantity,0)) as Tuesday,

sum(if(date\_format(order\_date,'%W')='Wednesday',quantity,0)) as Wednesday,

sum(if(date\_format(order\_date,'%W')='Thursday',quantity,0)) as Thursday,

sum(if(date\_format(order\_date,'%W')='Friday',quantity,0)) as Friday,

sum(if(date\_format(order\_date,'%W')='Saturday',quantity,0)) as Saturday,

sum(if(date\_format(order\_date,'%W')='Sunday',quantity,0)) as Sunday

from Items a

left join Orders b on a.item\_id = b.item\_id

group by item\_category

order by item\_category

### [1485. Group Sold Products By The Date](https://leetcode-cn.com/problems/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 contains 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 distinct 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.

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 |

+------------+-------------+

Result table:

+------------+----------+------------------------------+

| sell\_date | num\_sold | products |

+------------+----------+------------------------------+

| 2020-05-30 | 3 | Basketball,Headphone,T-shirt |

| 2020-06-01 | 2 | Bible,Pencil |

| 2020-06-02 | 1 | Mask |

+------------+----------+------------------------------+

For 2020-05-30, Sold items were (Headphone, Basketball, T-shirt), we sort them lexicographically and separate them by comma.

For 2020-06-01, Sold items were (Pencil, Bible), we sort them lexicographically and separate them by comma.

For 2020-06-02, Sold item is (Masks), we just return it.

#group\_concat 连接多个字符串， 字符串连接时可以排序， 自定义分隔符

select sell\_date, count(distinct product) as num\_sold,

group\_concat(distinct product) as products

from Activities

group by sell\_date

### [1495. Friendly Movies Streamed Last Month](https://leetcode-cn.com/problems/friendly-movies-streamed-last-month/)

Table: TVProgram

+---------------+---------+

| Column Name | Type |

+---------------+---------+

| program\_date | date |

| content\_id | int |

| channel | varchar |

+---------------+---------+

(program\_date, content\_id) is the primary key for this table.

This table contains information of the programs on the TV.

content\_id is the id of the program in some channel on the TV.

Table: Content

+------------------+---------+

| Column Name | Type |

+------------------+---------+

| content\_id | varchar |

| title | varchar |

| Kids\_content | enum |

| content\_type | varchar |

+------------------+---------+

content\_id is the primary key for this table.

Kids\_content is an enum that takes one of the values ('Y', 'N') where:

'Y' means is content for kids otherwise 'N' is not content for kids.

content\_type is the category of the content as movies, series, etc.

Write an SQL query to report the distinct titles of the kid-friendly movies streamed in June 2020.

Return the result table in any order.

The query result format is in the following example.

TVProgram table:

+--------------------+--------------+-------------+

| program\_date | content\_id | channel |

+--------------------+--------------+-------------+

| 2020-06-10 08:00 | 1 | LC-Channel |

| 2020-05-11 12:00 | 2 | LC-Channel |

| 2020-05-12 12:00 | 3 | LC-Channel |

| 2020-05-13 14:00 | 4 | Disney Ch |

| 2020-06-18 14:00 | 4 | Disney Ch |

| 2020-07-15 16:00 | 5 | Disney Ch |

+--------------------+--------------+-------------+

Content table:

+------------+----------------+---------------+---------------+

| content\_id | title | Kids\_content | content\_type |

+------------+----------------+---------------+---------------+

| 1 | Leetcode Movie | N | Movies |

| 2 | Alg. for Kids | Y | Series |

| 3 | Database Sols | N | Series |

| 4 | Aladdin | Y | Movies |

| 5 | Cinderella | Y | Movies |

+------------+----------------+---------------+---------------+

Result table:

+--------------+

| title |

+--------------+

| Aladdin |

+--------------+

"Leetcode Movie" is not a content for kids.

"Alg. for Kids" is not a movie.

"Database Sols" is not a movie

"Alladin" is a movie, content for kids and was streamed in June 2020.

"Cinderella" was not streamed in June 2020.

select distinct title

from TVProgram a

join Content b on a.content\_id = b.content\_id

and Kids\_content = 'Y' and content\_type = 'Movies'

where program\_date like "2020-06%"

### [1501. Countries You Can Safely Invest In](https://leetcode-cn.com/problems/countries-you-can-safely-invest-in/)

Table Person:

+----------------+---------+

| Column Name | Type |

+----------------+---------+

| id | int |

| name | varchar |

| phone\_number | varchar |

+----------------+---------+

id is the primary key for this table.

Each row of this table contains the name of a person and their phone number.

Phone number will be in the form 'xxx-yyyyyyy' where xxx is the country code (3 characters) and yyyyyyy is the phone number (7 characters) where x and y are digits. Both can contain leading zeros.

Table Country:

+----------------+---------+

| Column Name | Type |

+----------------+---------+

| name | varchar |

| country\_code | varchar |

+----------------+---------+

country\_code is the primary key for this table.

Each row of this table contains the country name and its code. country\_code will be in the form 'xxx' where x is digits.

Table Calls:

+-------------+------+

| Column Name | Type |

+-------------+------+

| caller\_id | int |

| callee\_id | int |

| duration | int |

+-------------+------+

There is no primary key for this table, it may contain duplicates.

Each row of this table contains the caller id, callee id and the duration of the call in minutes. caller\_id != callee\_id

A telecommunications company wants to invest in new countries. The company intends to invest in the countries where the average call duration of the calls in this country is strictly greater than the global average call duration.

Write an SQL query to find the countries where this company can invest.

Return the result table in any order.

The query result format is in the following example.

Person table:

+----+----------+--------------+

| id | name | phone\_number |

| 3 | Jonathan | 051-1234567 |

| 12 | Elvis | 051-7654321 |

| 1 | Moncef | 212-1234567 |

| 2 | Maroua | 212-6523651 |

| 7 | Meir | 972-1234567 |

| 9 | Rachel | 972-0011100 |

+----+----------+--------------+

Country table:

+----------+--------------+

| name | country\_code |

| Peru | 051 |

| Israel | 972 |

| Morocco | 212 |

| Germany | 049 |

| Ethiopia | 251 |

+----------+--------------+

Calls table:

+-----------+-----------+----------+

| caller\_id | callee\_id | duration |

| 1 | 9 | 33 |

| 2 | 9 | 4 |

| 1 | 2 | 59 |

| 3 | 12 | 102 |

| 3 | 12 | 330 |

| 12 | 3 | 5 |

| 7 | 9 | 13 |

| 7 | 1 | 3 |

| 9 | 7 | 1 |

| 1 | 7 | 7 |

+-----------+-----------+----------+

Result table:

+----------+

| country |

| Peru |

+----------+

The average call duration for Peru is (102 + 102 + 330 + 330 + 5 + 5) / 6 = 145.666667

The average call duration for Israel is (33 + 4 + 13 + 13 + 3 + 1 + 1 + 7) / 8 = 9.37500

The average call duration for Morocco is (33 + 4 + 59 + 59 + 3 + 7) / 6 = 27.5000

Global call duration average = (2 \* (33 + 3 + 59 + 102 + 330 + 5 + 13 + 3 + 1 + 7)) / 20 = 55.70000

Since Peru is the only country where average call duration is greater than the global average, it's the only recommended country.

select c2.name as country

from Calls c1

join Person p on id = caller\_id or id = callee\_id

join Country c2 on country\_code = left(phone\_number, 3)

group by c2.name

having avg(duration) > (select avg(duration) from Calls)

### [1511. Customer Order Frequency](https://leetcode-cn.com/problems/customer-order-frequency/)

Table: Customers

+---------------+---------+

| Column Name | Type |

+---------------+---------+

| customer\_id | int |

| name | varchar |

| country | varchar |

+---------------+---------+

customer\_id is the primary key for this table.

This table contains information of the customers in the company.

Table: Product

+---------------+---------+

| Column Name | Type |

+---------------+---------+

| product\_id | int |

| description | varchar |

| price | int |

+---------------+---------+

product\_id is the primary key for this table.

This table contains information of the products in the company.

price is the product cost.

Table: Orders

+---------------+---------+

| Column Name | Type |

+---------------+---------+

| order\_id | int |

| customer\_id | int |

| product\_id | int |

| order\_date | date |

| quantity | int |

+---------------+---------+

order\_id is the primary key for this table.

This table contains information on customer orders.

customer\_id is the id of the customer who bought "quantity" products with id "product\_id".

Order\_date is the date in format ('YYYY-MM-DD') when the order was shipped.

Write an SQL query to report the customer\_id and customer\_name of customers who have spent at least $100 in each month of June and July 2020.

Return the result table in any order.

The query result format is in the following example.

Customers

+--------------+-----------+-------------+

| customer\_id | name   | country     |

+--------------+-----------+-------------+

| 1   | Winston  | USA  |

| 2   | Jonathan | Peru  |

| 3   | Moustafa  | Egypt  |

+--------------+-----------+-------------+

Product

+--------------+-------------+-------------+

| product\_id | description | price     |

+--------------+-------------+-------------+

| 10   | LC Phone   | 300  |

| 20   | LC T-Shirt | 10  |

| 30   | LC Book  | 45  |

| 40 | LC Keychain | 2   |

+--------------+-------------+-------------+

Orders

+--------------+-------------+-------------+-------------+-----------+

| order\_id | customer\_id | product\_id | order\_date | quantity |

+--------------+-------------+-------------+-------------+-----------+

| 1   | 1   | 10  | 2020-06-10 | 1 |

| 2   | 1 | 20  | 2020-07-01 | 1 |

| 3   | 1 | 30  | 2020-07-08 | 2 |

| 4   | 2   | 10  | 2020-06-15 | 2 |

| 5   | 2 | 40  | 2020-07-01 | 10 |

| 6   | 3 | 20  | 2020-06-24 | 2 |

| 7   | 3   | 30  | 2020-06-25 | 2 |

| 9   | 3 | 30  | 2020-05-08 | 3 |

+--------------+-------------+-------------+-------------+-----------+

Result table:

+--------------+------------+

| customer\_id | name |

+--------------+------------+

| 1 | Winston |

+--------------+------------+

Winston spent $300 (300 \* 1) in June and $100 ( 10 \* 1 + 45 \* 2) in July 2020.

Jonathan spent $600 (300 \* 2) in June and $20 ( 2 \* 10) in July 2020.

Moustafa spent $110 (10 \* 2 + 45 \* 2) in June and $0 in July 2020.

select a.customer\_id, name

from Customers a

join Orders b on a.customer\_id = b.customer\_id

join Product c on b.product\_id = c.product\_id

and (order\_date like '2020-06%' or order\_date like '2020-07%')

group by a.customer\_id, name

having sum(IF(order\_date like '2020-06%', quantity\*price, 0)) >= 100

        and sum(IF(order\_date like '2020-07%',

quantity\*price, 0)) >= 100

### [1517. Find Users With Valid E-Mails](https://leetcode-cn.com/problems/find-users-with-valid-e-mails/)★

Table: Users

+---------------+---------+

| Column Name | Type |

+---------------+---------+

| user\_id | int |

| name | varchar |

| mail | varchar |

+---------------+---------+

user\_id is the primary key for this table.

This table contains information of the users signed up in a website. Some e-mails are invalid.

Write an SQL query to find the users who have **valid emails**.

A valid e-mail has a prefix name and a domain where:

* **The prefix name** is a string that may contain letters (upper or lower case), digits, underscore '\_', period '.' and/or dash '-'. The prefix name **must** start with a letter.
* **The domain** is '@leetcode.com'.

Return the result table in any order.

The query result format is in the following example.

Users

+---------+-----------+-------------------------+

| user\_id | name | mail |

+---------+-----------+-------------------------+

| 1 | Winston | winston@leetcode.com |

| 2 | Jonathan | jonathanisgreat |

| 3 | Annabelle | bella-@leetcode.com |

| 4 | Sally | sally.come@leetcode.com |

| 5 | Marwan | quarz#2020@leetcode.com |

| 6 | David | david69@gmail.com |

| 7 | Shapiro | .shapo@leetcode.com |

+---------+-----------+-------------------------+

Result table:

+---------+-----------+-------------------------+

| user\_id | name | mail |

+---------+-----------+-------------------------+

| 1 | Winston | winston@leetcode.com |

| 3 | Annabelle | bella-@leetcode.com |

| 4 | Sally | sally.come@leetcode.com |

+---------+-----------+-------------------------+

The mail of user 2 doesn't have a domain.

The mail of user 5 has # sign which is not allowed.

The mail of user 6 doesn't have leetcode domain.

The mail of user 7 starts with a period.

SELECT \*

FROM Users

WHERE mail REGEXP '^[A-Za-z][A-Za-z0-9\\_\.\-]\*@leetcode\.com$'

### [1527. Patients With a Condition](https://leetcode-cn.com/problems/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 all conditions of 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.

Patients

+------------+--------------+--------------+

| patient\_id | patient\_name | conditions |

+------------+--------------+--------------+

| 1 | Daniel  | YFEV COUGH |

| 2   | Alice |   |

| 3   | Bob  | DIAB100 MYOP |

| 4   | George  | ACNE DIAB100 |

| 5   | Alain  | DIAB201  |

+------------+--------------+--------------+

Result table:

+------------+--------------+--------------+

| patient\_id | patient\_name | conditions |

+------------+--------------+--------------+

| 3   | Bob  | DIAB100 MYOP |

| 4   | George   | ACNE DIAB100 |

+------------+--------------+--------------+

Bob and George both have a condition that starts with DIAB1.

select \*

from Patients

where conditions like '%DIAB1%'

select \*

from Patients

where conditions regexp 'DIAB1'

### [1532. The Most Recent Three Orders](https://leetcode-cn.com/problems/the-most-recent-three-orders/)

Table: Customers

+---------------+---------+

| Column Name | Type |

+---------------+---------+

| customer\_id | int |

| name | varchar |

+---------------+---------+

customer\_id is the primary key for this table.

This table contains information about customers.

Table: Orders

+---------------+---------+

| Column Name | Type |

+---------------+---------+

| order\_id | int |

| order\_date | date |

| customer\_id | int |

| cost | int |

+---------------+---------+

order\_id is the primary key for this table.

This table contains information about the orders made by customer\_id.

Each customer has **one order per day**.

Write an SQL query to find the most recent 3 orders of each user. If a user ordered less than 3 orders return all of their orders.

Return the result table sorted by customer\_name in **ascending** order and in case of a tie by the customer\_id in **ascending** order. If there still a tie, order them by the order\_date in **descending** order.

The query result format is in the following example:

Customers

+-------------+-----------+

| customer\_id | name |

+-------------+-----------+

| 1 | Winston |

| 2 | Jonathan |

| 3 | Annabelle |

| 4 | Marwan |

| 5 | Khaled |

+-------------+-----------+

Orders

+----------+------------+-------------+------+

| order\_id | order\_date | customer\_id | cost |

+----------+------------+-------------+------+

| 1 | 2020-07-31 | 1 | 30 |

| 2 | 2020-07-30 | 2 | 40 |

| 3 | 2020-07-31 | 3 | 70 |

| 4 | 2020-07-29 | 4 | 100 |

| 5 | 2020-06-10 | 1 | 1010 |

| 6 | 2020-08-01 | 2 | 102 |

| 7 | 2020-08-01 | 3 | 111 |

| 8 | 2020-08-03 | 1 | 99 |

| 9 | 2020-08-07 | 2 | 32 |

| 10 | 2020-07-15 | 1 | 2 |

+----------+------------+-------------+------+

Result table:

+---------------+-------------+----------+------------+

| customer\_name | customer\_id | order\_id | order\_date |

+---------------+-------------+----------+------------+

| Annabelle | 3 | 7 | 2020-08-01 |

| Annabelle | 3 | 3 | 2020-07-31 |

| Jonathan | 2 | 9 | 2020-08-07 |

| Jonathan | 2 | 6 | 2020-08-01 |

| Jonathan | 2 | 2 | 2020-07-30 |

| Marwan | 4 | 4 | 2020-07-29 |

| Winston | 1 | 8 | 2020-08-03 |

| Winston | 1 | 1 | 2020-07-31 |

| Winston | 1 | 10 | 2020-07-15 |

+---------------+-------------+----------+------------+

Winston has 4 orders, we discard the order of "2020-06-10" because it is the oldest order.

Annabelle has only 2 orders, we return them.

Jonathan has exactly 3 orders.

Marwan ordered only one time.

We sort the result table by customer\_name in ascending order, by customer\_id in ascending order and by order\_date in descending order in case of a tie.

**Follow-up:**  
Can you write a general solution for the most recent n orders?

select name customer\_name, a.customer\_id, order\_id, order\_date

from Customers a

join (

        select \*, dense\_rank() over (partition by customer\_id  order by order\_date desc) rnk

        from Orders

     ) b on a.customer\_id = b.customer\_id and rnk <= 3

order by name, a.customer\_id, order\_date desc

### [1543. Fix Product Name Format](https://leetcode-cn.com/problems/fix-product-name-format/)

Table: Sales

+--------------+---------+

| Column Name | Type |

+--------------+---------+

| sale\_id | int |

| product\_name | varchar |

| sale\_date | date |

+--------------+---------+

sale\_id is the primary key for this table.

Each row of this table contains the product name and the date it was sold.

Since table Sales was filled manually in the year 2000, product\_name may contain leading and/or trailing white spaces, also they are case-insensitive.

Write an SQL query to report

* product\_name in lowercase without leading or trailing white spaces.
* sale\_date in the format ('YYYY-MM')
* total the number of times the product was sold in this month.

Return the result table ordered by product\_name in **ascending order**, in case of a tie order it by sale\_date in **ascending order**.

The query result format is in the following example.

Sales

+------------+------------------+--------------+

| sale\_id | product\_name | sale\_date |

+------------+------------------+--------------+

| 1 | LCPHONE | 2000-01-16 |

| 2   | LCPhone | 2000-01-17 |

| 3   | LcPhOnE  | 2000-02-18 |

| 4   | LCKeyCHAiN | 2000-02-19 |

| 5   | LCKeyChain | 2000-02-28 |

| 6   | Matryoshka   | 2000-03-31 |

+------------+------------------+--------------+

Result table:

+--------------+--------------+----------+

| product\_name | sale\_date | total |

+--------------+--------------+----------+

| lcphone   | 2000-01  | 2  |

| lckeychain | 2000-02   | 2  |

| lcphone | 2000-02   | 1  |

| matryoshka | 2000-03   | 1  |

+--------------+--------------+----------+

In January, 2 LcPhones were sold, please note that the product names are not case sensitive and may contain spaces.

In Februery, 2 LCKeychains and 1 LCPhone were sold.

In March, 1 matryoshka was sold.

select lower(trim(product\_name)) product\_name,

DATE\_FORMAT(sale\_date, '%Y-%m') sale\_date, count(\*) total

from Sales

group by lower(trim(product\_name)), DATE\_FORMAT(sale\_date, '%Y-%m')

order by product\_name asc, sale\_date asc

### [1549. The Most Recent Orders for Each Product](https://leetcode-cn.com/problems/the-most-recent-orders-for-each-product/)

Table: Customers

+---------------+---------+

| Column Name | Type |

+---------------+---------+

| customer\_id | int |

| name | varchar |

+---------------+---------+

customer\_id is the primary key for this table.

This table contains information about the customers.

Table: Orders

+---------------+---------+

| Column Name | Type |

+---------------+---------+

| order\_id | int |

| order\_date | date |

| customer\_id | int |

| product\_id | int |

+---------------+---------+

order\_id is the primary key for this table.

This table contains information about the orders made by customer\_id.

There will be no product ordered by the same user **more than once** in one day.

Table: Products

+---------------+---------+

| Column Name | Type |

+---------------+---------+

| product\_id | int |

| product\_name | varchar |

| price | int |

+---------------+---------+

product\_id is the primary key for this table.

This table contains information about the Products.

Write an SQL query to find the most recent order(s) of each product.

Return the result table sorted by product\_name in **ascending** order and in case of a tie by the product\_id in **ascending** order. If there still a tie, order them by the order\_id in **ascending** order.

The query result format is in the following example:

Customers

+-------------+-----------+

| customer\_id | name |

| 1 | Winston |

| 2 | Jonathan |

| 3 | Annabelle |

| 4 | Marwan |

| 5 | Khaled |

+-------------+-----------+

Orders

+----------+------------+-------------+------------+

| order\_id | order\_date | customer\_id | product\_id |

| 1 | 2020-07-31 | 1 | 1 |

| 2 | 2020-07-30 | 2 | 2 |

| 3 | 2020-08-29 | 3 | 3 |

| 4 | 2020-07-29 | 4 | 1 |

| 5 | 2020-06-10 | 1 | 2 |

| 6 | 2020-08-01 | 2 | 1 |

| 7 | 2020-08-01 | 3 | 1 |

| 8 | 2020-08-03 | 1 | 2 |

| 9 | 2020-08-07 | 2 | 3 |

| 10 | 2020-07-15 | 1 | 2 |

+----------+------------+-------------+------------+

Products

+------------+--------------+-------+

| product\_id | product\_name | price |

| 1 | keyboard | 120 |

| 2 | mouse | 80 |

| 3 | screen | 600 |

| 4 | hard disk | 450 |

+------------+--------------+-------+

Result table:

+--------------+------------+----------+------------+

| product\_name | product\_id | order\_id | order\_date |

| keyboard | 1 | 6 | 2020-08-01 |

| keyboard | 1 | 7 | 2020-08-01 |

| mouse | 2 | 8 | 2020-08-03 |

| screen | 3 | 3 | 2020-08-29 |

+--------------+------------+----------+------------+

keyboard's most recent order is in 2020-08-01, it was ordered two times this day.

mouse's most recent order is in 2020-08-03, it was ordered only once this day.

screen's most recent order is in 2020-08-29, it was ordered only once this day.

The hard disk was never ordered and we don't include it in the result table.

select product\_name, a.product\_id, order\_id, order\_date

from Products a

join (

         select \*,

rank() over (partition by product\_id order by order\_date desc) rnk

         from Orders

     ) b on a.product\_id = b.product\_id and rnk = 1

order by product\_name, a.product\_id, order\_id

### [1555. Bank Account Summary](https://leetcode-cn.com/problems/bank-account-summary/)

Table: Users

+--------------+---------+

| Column Name | Type |

+--------------+---------+

| user\_id | int |

| user\_name | varchar |

| credit | int |

+--------------+---------+

user\_id is the primary key for this table.

Each row of this table contains the current credit information for each user.

Table: Transactions

+---------------+---------+

| Column Name | Type |

+---------------+---------+

| trans\_id | int |

| paid\_by | int |

| paid\_to | int |

| amount | int |

| transacted\_on | date |

+---------------+---------+

trans\_id is the primary key for this table.

Each row of this table contains the information about the transaction in the bank.

User with id (paid\_by) transfer money to user with id (paid\_to).

Leetcode Bank (LCB) helps its coders in making virtual payments. Our bank records all transactions in the table Transaction, we want to find out the current balance of all users and check wheter they have breached their credit limit (If their current credit is less than 0).

Write an SQL query to report.

* user\_id
* user\_name
* credit, current balance after performing transactions.
* credit\_limit\_breached, check credit\_limit ("Yes" or "No")

Return the result table in **any** order.

The query result format is in the following example.

Users table:

+------------+--------------+-------------+

| user\_id | user\_name | credit |

+------------+--------------+-------------+

| 1 | Moustafa | 100 |

| 2 | Jonathan | 200 |

| 3 | Winston | 10000 |

| 4 | Luis | 800 |

+------------+--------------+-------------+

Transactions table:

+------------+------------+------------+----------+---------------+

| trans\_id | paid\_by | paid\_to | amount | transacted\_on |

+------------+------------+------------+----------+---------------+

| 1 | 1 | 3 | 400 | 2020-08-01 |

| 2 | 3 | 2 | 500 | 2020-08-02 |

| 3 | 2 | 1 | 200 | 2020-08-03 |

+------------+------------+------------+----------+---------------+

Result table:

+------------+------------+------------+-----------------------+

| user\_id | user\_name | credit | credit\_limit\_breached |

+------------+------------+------------+-----------------------+

| 1 | Moustafa | -100 | Yes |

| 2 | Jonathan | 500 | No |

| 3 | Winston | 9900 | No |

| 4 | Luis | 800 | No |

+------------+------------+------------+-----------------------+

Moustafa paid $400 on "2020-08-01" and received $200 on "2020-08-03", credit (100 -400 +200) = -$100

Jonathan received $500 on "2020-08-02" and paid $200 on "2020-08-08", credit (200 +500 -200) = $500

Winston received $400 on "2020-08-01" and paid $500 on "2020-08-03", credit (10000 +400 -500) = $9990

Luis didn't received any transfer, credit = $800

select user\_id, user\_name,

credit + IFNULL(sum(IF(paid\_by=user\_id, -amount, amount)), 0) credit,

       IF(credit + IFNULL(sum(IF(paid\_by=user\_id, -amount, amount)),

0) < 0, 'Yes', 'No') credit\_limit\_breached

from Users

left join Transactions b on user\_id = paid\_by or user\_id = paid\_to

group by user\_id, user\_name

### [1565. Unique Orders and Customers Per Month](https://leetcode-cn.com/problems/unique-orders-and-customers-per-month/)

Table: Orders

+---------------+---------+

| Column Name | Type |

+---------------+---------+

| order\_id | int |

| order\_date | date |

| customer\_id | int |

| invoice | int |

+---------------+---------+

order\_id is the primary key for this table.

This table contains information about the orders made by customer\_id.

Write an SQL query to find the number of **unique orders** and the number of **unique customers** with invoices **> $20** for each **different month**.

Return the result table sorted in **any order**.

The query result format is in the following example:

Orders

+----------+------------+-------------+------------+

| order\_id | order\_date | customer\_id | invoice |

+----------+------------+-------------+------------+

| 1 | 2020-09-15 | 1 | 30 |

| 2 | 2020-09-17 | 2 | 90 |

| 3 | 2020-10-06 | 3 | 20 |

| 4 | 2020-10-20 | 3 | 21 |

| 5 | 2020-11-10 | 1 | 10 |

| 6 | 2020-11-21 | 2 | 15 |

| 7 | 2020-12-01 | 4 | 55 |

| 8 | 2020-12-03 | 4 | 77 |

| 9 | 2021-01-07 | 3 | 31 |

| 10 | 2021-01-15 | 2 | 20 |

+----------+------------+-------------+------------+

Result table:

+---------+-------------+----------------+

| month | order\_count | customer\_count |

+---------+-------------+----------------+

| 2020-09 | 2 | 2 |

| 2020-10 | 1 | 1 |

| 2020-12 | 2 | 1 |

| 2021-01 | 1 | 1 |

+---------+-------------+----------------+

In September 2020 we have two orders from 2 different customers with invoices > $20.

In October 2020 we have two orders from 1 customer, and only one of the two orders has invoice > $20.

In November 2020 we have two orders from 2 different customers but invoices < $20, so we don't include that month.

In December 2020 we have two orders from 1 customer both with invoices > $20.

In January 2021 we have two orders from 2 different customers, but only one of them with invoice > $20.

select DATE\_FORMAT(order\_date, "%Y-%m") month,

count(\*) order\_count,

count(distinct customer\_id) customer\_count

from Orders

where invoice > 20

group by month

### [1571. Warehouse Manager](https://leetcode-cn.com/problems/warehouse-manager/)

Table: Warehouse

+--------------+---------+

| Column Name | Type |

+--------------+---------+

| name | varchar |

| product\_id | int |

| units | int |

+--------------+---------+

(name, product\_id) is the primary key for this table.

Each row of this table contains the information of the products in each warehouse.

Table: Products

+---------------+---------+

| Column Name | Type |

+---------------+---------+

| product\_id | int |

| product\_name | varchar |

| Width | int |

| Length | int |

| Height | int |

+---------------+---------+

product\_id is the primary key for this table.

Each row of this table contains the information about the product dimensions (Width, Lenght and Height) in feets of each product.

Write an SQL query to report, How much cubic feet of **volume**does the inventory occupy in each warehouse.

* warehouse\_name
* volume

Return the result table in **any** order.

The query result format is in the following example.

Warehouse table:

+------------+--------------+-------------+

| name | product\_id | units |

+------------+--------------+-------------+

| LCHouse1 | 1 | 1 |

| LCHouse1 | 2 | 10 |

| LCHouse1 | 3 | 5 |

| LCHouse2 | 1 | 2 |

| LCHouse2 | 2 | 2 |

| LCHouse3 | 4 | 1 |

+------------+--------------+-------------+

Products table:

+------------+--------------+------------+----------+-----------+

| product\_id | product\_name | Width | Length | Height |

+------------+--------------+------------+----------+-----------+

| 1 | LC-TV | 5 | 50 | 40 |

| 2 | LC-KeyChain | 5 | 5 | 5 |

| 3 | LC-Phone | 2 | 10 | 10 |

| 4 | LC-T-Shirt | 4 | 10 | 20 |

+------------+--------------+------------+----------+-----------+

Result table:

+----------------+------------+

| warehouse\_name | volume |

+----------------+------------+

| LCHouse1 | 12250 |

| LCHouse2 | 20250 |

| LCHouse3 | 800 |

+----------------+------------+

Volume of product\_id = 1 (LC-TV), 5x50x40 = 10000

Volume of product\_id = 2 (LC-KeyChain), 5x5x5 = 125

Volume of product\_id = 3 (LC-Phone), 2x10x10 = 200

Volume of product\_id = 4 (LC-T-Shirt), 4x10x20 = 800

LCHouse1: 1 unit of LC-TV + 10 units of LC-KeyChain + 5 units of LC-Phone.

  Total volume: 1\*10000 + 10\*125 + 5\*200 = 12250 cubic feet

LCHouse2: 2 units of LC-TV + 2 units of LC-KeyChain.

  Total volume: 2\*10000 + 2\*125 = 20250 cubic feet

LCHouse3: 1 unit of LC-T-Shirt.

Total volume: 1\*800 = 800 cubic feet.

select name warehouse\_name, sum(units\*Width\*Length\*Height) as volume

from Warehouse a

join Products b on a.product\_id = b.product\_id

group by name