**/\*---------------------------------------------------------------------------\*/**

**/\* Leetcode SQL Problems: Medium and Hard Complexity**   **\*/**

**/\*---------------------------------------------------------------------------\*/**

/\* 1978. Employees Whose Manager Left the Company

Table: Employees

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

| Column Name | Type |

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

| employee\_id | int |

| name | varchar |

| manager\_id | int |

| salary | int |

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

In SQL, employee\_id is the primary key for this table.

This table contains information about the employees, their salary, and the ID of their manager. Some employees do not have a manager (manager\_id is null).

Find the IDs of the employees whose salary is strictly less than $30000 and whose manager left the company. When a manager leaves the company, their information is deleted from the Employees table, but the reports still have their manager\_id set to the manager that left.

Return the result table ordered by employee\_id.

The result format is in the following example.

Example 1:

Input:

Employees table:

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

| employee\_id | name | manager\_id | salary |

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

| 3 | Mila | 9 | 60301 |

| 12 | Antonella | null | 31000 |

| 13 | Emery | null | 67084 |

| 1 | Kalel | 11 | 21241 |

| 9 | Mikaela | null | 50937 |

| 11 | Joziah | 6 | 28485 |

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

Output:

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

| employee\_id |

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

| 11 |

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

Explanation:

The employees with a salary less than $30000 are 1 (Kalel) and 11 (Joziah).

Kalel's manager is employee 11, who is still in the company (Joziah).

Joziah's manager is employee 6, who left the company because there is no row for employee 6 as it was deleted. \*/

create table ltcd\_employee (employee\_id int primary key, name varchar(20), manager\_id int, salary int);

insert into ltcd\_employee values (3 ,'Mila',9,60301),

(12,'Antonella',null,31000 ),

(13,'Emery ',null,67084 ),

(1 ,'Kalel ',11 ,21241 ),

(9 ,'Mikaela ',null,50937 ),

(11,'Joziah ',6 ,28485 );

select employee\_id from ltcd\_employee where salary < 30000 and manager\_id

not in (select employee\_id from ltcd\_employee ) order by employee\_id;

A screenshot of a computer

Description automatically generated

/\*--------------------------------------------------------------------------\*/

/\*1965. Employees With Missing Information

Table: Employees

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

| Column Name | Type |

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

| employee\_id | int |

| name | varchar |

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

employee\_id is the column with unique values for this table.

Each row of this table indicates the name of the employee whose ID is employee\_id.

Table: Salaries

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

| Column Name | Type |

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

| employee\_id | int |

| salary | int |

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

employee\_id is the column with unique values for this table.

Each row of this table indicates the salary of the employee whose ID is employee\_id.

Write a solution to report the IDs of all the employees with missing information. The information of an employee is missing if:

The employee's name is missing, or

The employee's salary is missing.

Return the result table ordered by employee\_id in ascending order.

The result format is in the following example.

Example 1:

Input:

Employees table:

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

| employee\_id | name |

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

| 2 | Crew |

| 4 | Haven |

| 5 | Kristian |

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

Salaries table:

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

| employee\_id | salary |

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

| 5 | 76071 |

| 1 | 22517 |

| 4 | 63539 |

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

Output:

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

| employee\_id |

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

| 1 |

| 2 |

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

Explanation:

Employees 1, 2, 4, and 5 are working at this company.

The name of employee 1 is missing.

The salary of employee 2 is missing.

\*/

create table ltcd\_emp1 (employee\_id int, name varchar(20));

insert into ltcd\_emp1 values (2,'Crew'),(4,'Haven'), (5,'Kristian');

create table ltcd\_salaries (employee\_id int, salary int);

insert into ltcd\_salaries values (5,76071), (1,22517), (4,63539);

select employee\_id from

(select employee\_id from ltcd\_emp1 where employee\_id not in (select employee\_id from ltcd\_salaries)

union

select employee\_id from ltcd\_salaries where employee\_id not in (select employee\_id from ltcd\_emp1) )a

order by a.employee\_id

/\*--------------------------------------------------------------------------\*/

/\*--------------------------------------------------------------------------\*/

/\* 1141. User Activity for the Past 30 Days I

Table: Activity

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

| Column Name | Type |

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

| user\_id | int |

| session\_id | int |

| activity\_date | date |

| activity\_type | enum |

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

This table may have duplicate rows.

The activity\_type column is an ENUM (category) 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 a solution to find the daily active user count for a period of 30 days ending 2019-07-27 inclusively. A user was active on

someday if they made at least one activity on that day.

Return the result table in any order.

The result format is in the following example.

Example 1:

Input:

Activity table:

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

| user\_id | session\_id | activity\_date | activity\_type |

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

| 1 | 1 | 2019-07-20 | open\_session |

| 1 | 1 | 2019-07-20 | scroll\_down |

| 1 | 1 | 2019-07-20 | end\_session |

| 2 | 4 | 2019-07-20 | open\_session |

| 2 | 4 | 2019-07-21 | send\_message |

| 2 | 4 | 2019-07-21 | end\_session |

| 3 | 2 | 2019-07-21 | open\_session |

| 3 | 2 | 2019-07-21 | send\_message |

| 3 | 2 | 2019-07-21 | end\_session |

| 4 | 3 | 2019-06-25 | open\_session |

| 4 | 3 | 2019-06-25 | end\_session |

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

Output:

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

| day | active\_users |

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

| 2019-07-20 | 2 |

| 2019-07-21 | 2 |

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

Explanation: Note that we do not care about days with zero active users.

\*/

create table ltcd\_activity( user\_id int, session\_id int, activity\_date date,

activity\_type varchar(30) NOT NULL CHECK (activity\_type in ('open\_session', 'end\_session', 'scroll\_down', 'send\_message')));

insert into ltcd\_activity values (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 ');

select \* from ltcd\_activity;

select

activity\_date as day,

count(distinct user\_id) as active\_users

from

ltcd\_activity

where

activity\_date between '2019-06-28' and '2019-07-27'

group by

activity\_date;

A screenshot of a computer

Description automatically generated

/\*--------------------------------------------------------------------------\*/

/\*--------------------------------------------------------------------------\*/

/\*1148. Article Views

Table: Views

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

| Column Name | Type |

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

| article\_id | int |

| author\_id | int |

| viewer\_id | int |

| view\_date | date |

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

There is no primary key (column with unique values) for this table, the table 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 a solution to find all the authors that viewed at least one of their own articles.

Return the result table sorted by id in ascending order.

The result format is in the following example.

Example 1:

Input:

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 |

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

Output:

+------+

| id |

+------+

| 4 |

| 7 |]

+------+

\*/

create table ltcd\_view (article\_id int, author\_id int, viewer\_id int, view\_date date);

insert into ltcd\_view values

(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');

select distinct author\_id from ltcd\_view

where author\_id = viewer\_id

order by author\_id

/\*--------------------------------------------------------------------------\*/

/\*1179. Reformat Department Table

Table: Department

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

| Column Name | Type |

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

| id | int |

| revenue | int |

| month | varchar |

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

In SQL,(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"].

Reformat the table such that there is a department id column and a revenue column for each month.

Return the result table in any order.

The result format is in the following example.

Example 1:

Input:

Department table:

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

| id | revenue | month |

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

| 1 | 8000 | Jan |

| 2 | 9000 | Jan |

| 3 | 10000 | Feb |

| 1 | 7000 | Feb |

| 1 | 6000 | Mar |

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

Output:

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

| id | Jan\_Revenue | Feb\_Revenue | Mar\_Revenue | ... | Dec\_Revenue |

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

| 1 | 8000 | 7000 | 6000 | ... | null |

| 2 | 9000 | null | null | ... | null |

| 3 | null | 10000 | null | ... | null |

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

Explanation: The revenue from Apr to Dec is null.

Note that the result table has 13 columns (1 for the department id + 12 for the months).

\*/

create table ltcd\_Department (id int, revenue int, month varchar(10));

insert into ltcd\_Department values

(1,8000,'Jan'),

(2,9000,'Jan'),

(3,10000,'Feb'),

(1,7000,'Feb'),

(1,6000,'Mar');

select \* from ltcd\_Department;

with CTE as

(select \* from (select id, revenue, month from ltcd\_Department ) as dept\_pivot\_base

pivot

( sum(revenue)

for month in ([Jan],[Feb],[Mar],[Apr],[May],[Jun],

[Jul],[Aug],[Sep],[Oct],[Nov],[Dec])

) as pivot\_table )

select id,Jan as 'Jan\_Revenue', Feb as 'Feb\_Revenue',Mar as 'Mar\_Revenue',Apr as 'Apr\_Revenue',May as 'May\_Revenue',Jun as 'Jun\_Revenue',

Jul as 'Jul\_Revenue',Aug as 'Aug\_Revenue',Sep as 'Sep\_Revenue',Oct as 'Oct\_Revenue',Nov as 'Nov\_Revenue',Dec as 'Dec\_Revenue'

from cte ;

select id, sum(case when month = 'Jan' then revenue end) Jan\_Revenue,

sum(case when month = 'Feb' then revenue end) Feb\_Revenue ,

sum(case when month = 'Mar' then revenue end) Mar\_Revenue ,

sum(case when month = 'Apr' then revenue end) Apr\_Revenue ,

sum(case when month = 'May' then revenue end) May\_Revenue ,

sum(case when month = 'Jun' then revenue end) Jun\_Revenue ,

sum(case when month = 'Jul' then revenue end) Jul\_Revenue ,

sum(case when month = 'Aug' then revenue end) Aug\_Revenue ,

sum(case when month = 'Sep' then revenue end) Sep\_Revenue ,

sum(case when month = 'Oct' then revenue end) Oct\_Revenue ,

sum(case when month = 'Nov' then revenue end) Nov\_Revenue ,

sum(case when month = 'Dec' then revenue end) Dec\_Revenue

from ltcd\_Department group by id

A screenshot of a computer

Description automatically generated

/\*--------------------------------------------------------------------------\*/

/\*1890. The Latest Login in 2020

Table: Logins

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

| Column Name | Type |

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

| user\_id | int |

| time\_stamp | datetime |

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

(user\_id, time\_stamp) is the primary key (combination of columns with unique values) for this table.

Each row contains information about the login time for the user with ID user\_id.

Write a solution to report the latest login for all users in the year 2020. Do not include the users who did not login in 2020.

Return the result table in any order.

The result format is in the following example.

Example 1:

Input:

Logins table:

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

| user\_id | time\_stamp |

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

| 6 | 2020-06-30 15:06:07 |

| 6 | 2021-04-21 14:06:06 |

| 6 | 2019-03-07 00:18:15 |

| 8 | 2020-02-01 05:10:53 |

| 8 | 2020-12-30 00:46:50 |

| 2 | 2020-01-16 02:49:50 |

| 2 | 2019-08-25 07:59:08 |

| 14 | 2019-07-14 09:00:00 |

| 14 | 2021-01-06 11:59:59 |

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

Output:

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

| user\_id | last\_stamp |

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

| 6 | 2020-06-30 15:06:07 |

| 8 | 2020-12-30 00:46:50 |

| 2 | 2020-01-16 02:49:50 |

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

Explanation:

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

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

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

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

create table ltcd\_logins (user\_id int, time\_stamp datetime);

insert into ltcd\_logins values

(6 ,'2020-06-30 15:06:07'),

(6 ,'2021-04-21 14:06:06'),

(6 ,'2019-03-07 00:18:15'),

(8 ,'2020-02-01 05:10:53'),

(8 ,'2020-12-30 00:46:50'),

(2 ,'2020-01-16 02:49:50'),

(2 ,'2019-08-25 07:59:08'),

(14,'2019-07-14 09:00:00'),

(14,'2021-01-06 11:59:59');

select distinct user\_id, max(time\_stamp) over (partition by user\_id) last\_stamp from ltcd\_logins where YEAR(time\_stamp) = '2020';

A screenshot of a graph

Description automatically generated

/\*--------------------------------------------------------------------------\*/

/\*2356. Number of Unique Subjects Taught by Each Teacher

Table: Teacher

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

| Column Name | Type |

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

| teacher\_id | int |

| subject\_id | int |

| dept\_id | int |

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

(subject\_id, dept\_id) is the primary key (combinations of columns with unique values) of this table.

Each row in this table indicates that the teacher with teacher\_id teaches the subject subject\_id in the department dept\_id.

Write a solution to calculate the number of unique subjects each teacher teaches in the university.

Return the result table in any order.

The result format is shown in the following example.

Example 1:

Input:

Teacher table:

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

| teacher\_id | subject\_id | dept\_id |

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

| 1 | 2 | 3 |

| 1 | 2 | 4 |

| 1 | 3 | 3 |

| 2 | 1 | 1 |

| 2 | 2 | 1 |

| 2 | 3 | 1 |

| 2 | 4 | 1 |

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

Output:

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

| teacher\_id | cnt |

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

| 1 | 2 |

| 2 | 4 |

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

Explanation:

Teacher 1:

- They teach subject 2 in departments 3 and 4.

- They teach subject 3 in department 3.

Teacher 2:

- They teach subject 1 in department 1.

- They teach subject 2 in department 1.

- They teach subject 3 in department 1.

- They teach subject 4 in department 1.

\*/

create table ltcd\_teachers(teacher\_id int, subject\_id int, dept\_id int, primary key (subject\_id, dept\_id));

insert into ltcd\_teachers values

(1,2,3),

(1,2,4),

(1,3,3),

(2,1,1),

(2,2,1),

(2,3,1),

(2,4,1);

--number of unique subjects each teacher teaches in the university.

select teacher\_id, count(distinct subject\_id) cnt from ltcd\_teachers group by teacher\_id;

select X.teacher\_id, count(X.subject\_id) cnt from (select \*,row\_number() over (partition by teacher\_id, subject\_id order by teacher\_id) rnk from ltcd\_teachers) X

where X.rnk=1

group by teacher\_id;

/\*--------------------------------------------------------------------------\*/

/\* 3220. Odd and Even Transactions

Medium

Topics

Companies

SQL Schema

Pandas Schema

Table: transactions

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

| Column Name | Type |

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

| transaction\_id | int |

| amount | int |

| transaction\_date | date |

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

The transactions\_id column uniquely identifies each row in this table.

Each row of this table contains the transaction id, amount and transaction date.

Write a solution to find the sum of amounts for odd and even transactions for each day. If there are no odd or even transactions for a specific date, display as 0.

Return the result table ordered by transaction\_date in ascending order.

The result format is in the following example.

Example:

Input:

transactions table:

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

| transaction\_id | amount | transaction\_date |

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

| 1 | 150 | 2024-07-01 |

| 2 | 200 | 2024-07-01 |

| 3 | 75 | 2024-07-01 |

| 4 | 300 | 2024-07-02 |

| 5 | 50 | 2024-07-02 |

| 6 | 120 | 2024-07-03 |

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

Output:

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

| transaction\_date | odd\_sum | even\_sum |

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

| 2024-07-01 | 75 | 350 |

| 2024-07-02 | 0 | 350 |

| 2024-07-03 | 0 | 120 |

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

Explanation:

For transaction dates:

2024-07-01:

Sum of amounts for odd transactions: 75

Sum of amounts for even transactions: 150 + 200 = 350

2024-07-02:

Sum of amounts for odd transactions: 0

Sum of amounts for even transactions: 300 + 50 = 350

2024-07-03:

Sum of amounts for odd transactions: 0

Sum of amounts for even transactions: 120

Note: The output table is ordered by transaction\_date in ascending order.\*/

create table ltcd\_transactions(transaction\_id int, amount int, transaction\_date date);

insert into ltcd\_transactions values

(1,150,'2024-07-01'),

(2,200,'2024-07-01'),

(3,75 ,'2024-07-01'),

(4,300,'2024-07-02'),

(5,50 ,'2024-07-02'),

(6,120,'2024-07-03');

select transaction\_date,

COALESCE(sum(case when amount%2=1 then amount end),0) odd\_sum,

COALESCE(sum(case when amount%2=0 then amount end),0) even\_sum

from ltcd\_transactions

group by transaction\_date

order by transaction\_date;

select transaction\_date,

sum(case when amount % 2 = 1 then amount else 0 end) as odd\_sum,

sum(case when amount % 2 = 0 then amount else 0 end) as even\_sum

from transactions

group by transaction\_date

order by transaction\_date

/\*--------------------------------------------------------------------------\*/

/\* 1873. Calculate Special Bonus

Table: Employees

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

| Column Name | Type |

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

| employee\_id | int |

| name | varchar |

| salary | int |

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

employee\_id is the primary key (column with unique values) for this table.

Each row of this table indicates the employee ID, employee name, and salary.

Write a solution to calculate the bonus of each employee. The bonus of an employee is 100% of their salary if the ID of the

employee is an odd number and the employee's name does not start with the character 'M'.

The bonus of an employee is 0 otherwise.

Return the result table ordered by employee\_id.

The result format is in the following example.

Example 1:

Input:

Employees table:

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

| employee\_id | name | salary |

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

| 2 | Meir | 3000 |

| 3 | Michael | 3800 |

| 7 | Addilyn | 7400 |

| 8 | Juan | 6100 |

| 9 | Kannon | 7700 |

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

Output:

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

| employee\_id | bonus |

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

| 2 | 0 |

| 3 | 0 |

| 7 | 7400 |

| 8 | 0 |

| 9 | 7700 |

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

Explanation:

The employees with IDs 2 and 8 get 0 bonus because they have an even employee\_id.

The employee with ID 3 gets 0 bonus because their name starts with 'M'.

The rest of the employees get a 100% bonus.

\*/

create table ltcd\_employees(employee\_id int, name varchar(20), salary int);

insert into ltcd\_employees values

(2,'Meir',3000),

(3,'Michael',3800),

(7,'Addilyn',7400),

(8,'Juan',6100),

(9,'Kannon',7700);

SELECT employee\_id,

CASE

WHEN employee\_id%2 = 0 THEN 0

WHEN name LIKE 'M%' THEN 0

ELSE salary

END AS bonus

FROM ltcd\_employees ORDER BY employee\_id;

/\*--------------------------------------------------------------------------\*/

/\*1393. Capital Gain/Loss

Table: Stocks

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

| Column Name | Type |

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

| stock\_name | varchar |

| operation | enum |

| operation\_day | int |

| price | int |

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

(stock\_name, operation\_day) is the primary key (combination of columns with unique values) for this table.

The operation column is an ENUM (category) 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.

It is also guaranteed that each 'Buy' operation for a stock has a corresponding 'Sell' operation in an upcoming day.

Write a solution to report the Capital gain/loss for each stock.

The Capital gain/loss of a stock is the total gain or loss after buying and selling the stock one or many times.

Return the result table in any order.

The result format is in the following example.

Example 1:

Input:

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 |

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

Output:

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

| stock\_name | capital\_gain\_loss |

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

| Corona Masks | 9500 |

| Leetcode | 8000 |

| Handbags | -23000 |

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

Explanation:

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$. Capital gain/loss is the sum of capital gains/losses for each ('Buy' --> 'Sell') operation = (1010 - 10) + (500 - 1000) + (10000 - 1000) = 1000 - 500 + 9000 = 9500$.

\*/

create table ltcd\_stocks(stock\_name varchar(20), operation varchar(10) NOT NULL check(operation in ('Sell', 'Buy')), operation\_day int, price int)

insert into ltcd\_stocks values

('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);

--The Capital gain/loss of a stock is the total gain or loss after buying and selling the stock one or many times.

with cte as

(select \*, lead(price,1,0) over (partition by stock\_name order by operation\_day) next\_price from ltcd\_stocks)

select stock\_name, sum(case when operation= 'Buy' then next\_price-price end) capital\_gain\_loss from cte group by stock\_name order by 2 desc

A screenshot of a chat

Description automatically generated

/\*--------------------------------------------------------------------------\*/

/\*1907. Count Salary Categories

Table: Accounts

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

| Column Name | Type |

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

| account\_id | int |

| income | int |

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

account\_id is the primary key (column with unique values) for this table.

Each row contains information about the monthly income for one bank account.

Write a solution to calculate the number of bank accounts for each salary category. The salary categories are:

"Low Salary": All the salaries strictly less than $20000.

"Average Salary": All the salaries in the inclusive range [$20000, $50000].

"High Salary": All the salaries strictly greater than $50000.

The result table must contain all three categories. If there are no accounts in a category, return 0.

Return the result table in any order.

The result format is in the following example.

Example 1:

Input:

Accounts table:

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

| account\_id | income |

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

| 3 | 108939 |

| 2 | 12747 |

| 8 | 87709 |

| 6 | 91796 |

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

Output:

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

| category | accounts\_count |

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

| Low Salary | 1 |

| Average Salary | 0 |

| High Salary | 3 |

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

Explanation:

Low Salary: Account 2.

Average Salary: No accounts.

High Salary: Accounts 3, 6, and 8.

\*/

create table ltcd\_accounts (account\_id int, income int);

insert into ltcd\_accounts values

(3,108939),

(2,12747 ),

(8,87709 ),

(6,91796 );

Select 'Low Salary' as Category, Count(\*) as accounts\_count

From ltcd\_accounts

where income<20000

union

Select 'High Salary' as Category, Count(\*) as accounts\_count

From ltcd\_accounts

where income>50000

union

Select 'Average Salary' as Category, Count(\*) as accounts\_count

From ltcd\_accounts

where income between 20000 and 50000

with base\_query as

(select account\_id , Category =

case when income <20000 then 'Low Salary'

when income between 20000 and 50000 then 'Average Salary'

when income > 50000 then 'High Salary'

end

from ltcd\_accounts),

level1 as

(select count(case when Category = 'High Salary' then account\_id

when Category = 'Average Salary' then account\_id

when Category = 'Low Salary' then account\_id

end ) Accounts\_Count,

Category

from base\_query group by category

union select 0,'High Salary'

union select 0,'Average Salary'

union select 0,'Low Salary')

select Category, max(Accounts\_Count) Accounts\_Count from level1 group by Category;

A screenshot of a computer

Description automatically generated

/\*--------------------------------------------------------------------------\*/

/\*1934. Confirmation Rate

Table: Signups

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

| Column Name | Type |

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

| user\_id | int |

| time\_stamp | datetime |

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

user\_id is the column of unique values for this table.

Each row contains information about the signup time for the user with ID user\_id.

Table: Confirmations

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

| Column Name | Type |

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

| user\_id | int |

| time\_stamp | datetime |

| action | ENUM |

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

(user\_id, time\_stamp) is the primary key (combination of columns with unique values) for this table.

user\_id is a foreign key (reference column) to the Signups table.

action is an ENUM (category) of the type ('confirmed', 'timeout')

Each row of this table indicates that the user with ID user\_id requested a confirmation message at time\_stamp and

that confirmation message was either confirmed ('confirmed') or expired without confirming ('timeout').

The confirmation rate of a user is the number of 'confirmed' messages divided by the total number of requested confirmation

messages. The confirmation rate of a user that did not request any confirmation messages is 0. Round the confirmation rate

to two decimal places.

Write a solution to find the confirmation rate of each user.

Return the result table in any order.

The result format is in the following example.

Example 1:

Input:

Signups table:

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

| user\_id | time\_stamp |

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

| 3 | 2020-03-21 10:16:13 |

| 7 | 2020-01-04 13:57:59 |

| 2 | 2020-07-29 23:09:44 |

| 6 | 2020-12-09 10:39:37 |

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

Confirmations table:

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

| user\_id | time\_stamp | action |

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

| 3 | 2021-01-06 03:30:46 | timeout |

| 3 | 2021-07-14 14:00:00 | timeout |

| 7 | 2021-06-12 11:57:29 | confirmed |

| 7 | 2021-06-13 12:58:28 | confirmed |

| 7 | 2021-06-14 13:59:27 | confirmed |

| 2 | 2021-01-22 00:00:00 | confirmed |

| 2 | 2021-02-28 23:59:59 | timeout |

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

Output:

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

| user\_id | confirmation\_rate |

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

| 6 | 0.00 |

| 3 | 0.00 |

| 7 | 1.00 |

| 2 | 0.50 |

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

Explanation:

User 6 did not request any confirmation messages. The confirmation rate is 0.

User 3 made 2 requests and both timed out. The confirmation rate is 0.

User 7 made 3 requests and all were confirmed. The confirmation rate is 1.

User 2 made 2 requests where one was confirmed and the other timed out. The confirmation rate is 1 / 2 = 0.5.

\*/

create table ltcd\_Signups(user\_id int primary key, time\_stamp datetime);

create table ltcd\_Confirmations(user\_id int, time\_stamp datetime,

action varchar(15) not null check (action in ('confirmed', 'timeout')),

constraint pk\_signups primary key (user\_id, time\_stamp),

CONSTRAINT FK\_User\_Id FOREIGN KEY (user\_id) REFERENCES ltcd\_Signups(user\_id));

insert into ltcd\_Signups values

(3,'2020-03-21 10:16:13'),

(7,'2020-01-04 13:57:59'),

(2,'2020-07-29 23:09:44'),

(6,'2020-12-09 10:39:37');

insert into ltcd\_Confirmations values

(3,'2021-01-06 03:30:46','timeout'),

(3,'2021-07-14 14:00:00','timeout'),

(7,'2021-06-12 11:57:29','confirmed'),

(7,'2021-06-13 12:58:28','confirmed'),

(7,'2021-06-14 13:59:27','confirmed'),

(2,'2021-01-22 00:00:00','confirmed'),

(2,'2021-02-28 23:59:59','timeout');

/\*The confirmation rate of a user is the number of 'confirmed' messages divided by the total number of requested confirmation

messages. The confirmation rate of a user that did not request any confirmation messages is 0. Round the confirmation rate

to two decimal places.

Write a solution to find the confirmation rate of each user.

\*/

/\* Write your T-SQL query statement below \*/

with confirm\_base\_query as

(select s.user\_id,

Confirmation\_value = (case when c.action = 'confirmed' then 1.0 else 0.0 end ),

ROW\_NUMBER () over (partition by c.user\_id order by c.time\_stamp) rn

from ltcd\_Confirmations c right outer join ltcd\_Signups s

on c.user\_id = s.user\_id )

select

user\_id,

convert(decimal(3,2), (sum(Confirmation\_value))/(max(rn))) confirmation\_rate

from confirm\_base\_query

group by user\_id;

A screenshot of a computer

Description automatically generated

/\*--------------------------------------------------------------------------\*/

/\*176. Second Highest Salary

Table: Employee

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

| Column Name | Type |

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

| id | int |

| salary | int |

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

id is the primary key (column with unique values) for this table.

Each row of this table contains information about the salary of an employee.

Write a solution to find the second highest salary from the Employee table. If there is no second highest salary, return null (return None in Pandas).

The result format is in the following example.

Example 1:

Input:

Employee table:

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

| id | salary |

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

| 1 | 100 |

| 2 | 200 |

| 3 | 300 |

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

Output:

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

| SecondHighestSalary |

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

| 200 |

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

Example 2:

Input:

Employee table:

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

| id | salary |

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

| 1 | 100 |

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

Output:

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

| SecondHighestSalary |

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

| null |

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

\*/

select max(salary) SecondHighestSalary from Employee where salary <> (select max(salary) from employee);

select max(salary) as "SecondHighestSalary" from Employee where salary < (select max(salary) from Employee);

select isnull((select distinct salary from employee order by salary desc offset 1 rows fetch next 1 rows only),null) as SecondHighestSalary;

select top 1 lead(salary) over (order by salary desc) as SecondHighestSalary from (select distinct salary from employee) a order by salary desc;

select distinct salary as SecondHighestSalary from (select salary, dense\_rank() over (order by salary desc) as rank from employee) a right join (select 1 as rank union select 2 as rank) b on a.rank = b.rank where b.rank = 2;

select max(a.salary) as SecondHighestSalary from employee a right join employee b on a.salary < b.salary;

select max(a.salary) as SecondHighestSalary from employee a, employee b where a.salary < b.salary;

CREATE FUNCTION getNthHighestSalary (@N INT)

RETURNS INT

AS

BEGIN

  DECLARE @result INT;

  SELECT @result = Salary

  FROM (

    SELECT Salary, DENSE\_RANK() OVER (ORDER BY Salary DESC) AS ranking

    FROM Employee

  ) AS ranked\_salaries

  WHERE ranking = @N;

  RETURN @result;

END;

/\*--------------------------------------------------------------------------\*/

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

Table: Scores

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

| Column Name | Type |

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

| id | int |

| score | decimal |

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

id is the primary key (column with unique values) for this table.

Each row of this table contains the score of a game. Score is a floating point value with two decimal places.

Write a solution to find the rank of the scores. The ranking should be calculated according to the following rules:

* The scores should be ranked from the highest to the lowest.
* If there is a tie between two scores, both should have the same ranking.
* After a tie, the next ranking number should be the next consecutive integer value. In other words, there should be no holes between ranks.

Return the result table ordered by score in descending order.

The result format is in the following example.

**Example 1:**

**Input:**

Scores table:

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

| id | score |

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

| 1 | 3.50 |

| 2 | 3.65 |

| 3 | 4.00 |

| 4 | 3.85 |

| 5 | 4.00 |

| 6 | 3.65 |

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

**Output:**

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

| score | rank |

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

| 4.00 | 1 |

| 4.00 | 1 |

| 3.85 | 2 |

/\* Write your T-SQL query statement below \*/

select score, dense\_rank () over (order by score desc) rank from scores order by score x`desc;

| 3.65 | 3 |

| 3.65 | 3 |

| 3.50 | 4 |

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

Solution:

A screenshot of a computer

Description automatically generated

/\*--------------------------------------------------------------------------\*/

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

Table: Logs

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

| Column Name | Type |

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

| id | int |

| num | varchar |

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

In SQL, id is the primary key for this table.

id is an autoincrement column.

Find all numbers that appear at least three times consecutively.

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

The result format is in the following example.

**Example 1:**

**Input:**

Logs table:

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

| id | num |

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

| 1 | 1 |

| 2 | 1 |

| 3 | 1 |

| 4 | 2 |

| 5 | 1 |

| 6 | 2 |

| 7 | 2 |

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

**Output:**

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

| ConsecutiveNums |

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

| 1 |

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

**Explanation:** 1 is the only number that appears consecutively for at least three times.

Solution:

/\* Write your T-SQL query statement below \*/

with cte3 as

(select \*,

case when num = lead(num) over (order by id)

     and num = lead(num, 2) over (order by id)

     then num

     else null

end as CN,

case when lead(id) over (order by id) -id  = 1

     and lead(id,2) over (order by id) -id  = 2

     then id

     else null

end as seq

from logs )

select distinct num as ConsecutiveNums from cte3 where CN is not null and seq is not null;

A screenshot of a computer

Description automatically generated

/\*--------------------------------------------------------------------------\*/

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

Table: Employee

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

| Column Name | Type |

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

| id | int |

| name | varchar |

| salary | int |

| departmentId | int |

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

id is the primary key (column with unique values) for this table.

departmentId is a foreign key (reference columns) of the ID from the Department table.

Each row of this table indicates the ID, name, and salary of an employee. It also contains the ID of their department.

Table: Department

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

| Column Name | Type |

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

| id | int |

| name | varchar |

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

id is the primary key (column with unique values) for this table. It is guaranteed that department name is not NULL.

Each row of this table indicates the ID of a department and its name.

Write a solution to find employees who have the highest salary in each of the departments.

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

The result format is in the following example.

**Example 1:**

**Input:**

Employee table:

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

| id | name | salary | departmentId |

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

| 1 | Joe | 70000 | 1 |

| 2 | Jim | 90000 | 1 |

| 3 | Henry | 80000 | 2 |

| 4 | Sam | 60000 | 2 |

| 5 | Max | 90000 | 1 |

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

Department table:

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

| id | name |

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

| 1 | IT |

| 2 | Sales |

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

**Output:**

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

| Department | Employee | Salary |

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

| IT | Jim | 90000 |

| Sales | Henry | 80000 |

| IT | Max | 90000 |

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

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

**Solution:**

/\* Write your T-SQL query statement below \*/

with cte as

    (select e.name Employee,

            e.salary,

            d.name Department,

            rank () over(partition by e.departmentId order by e.salary desc) rnk

    from employee e

        join department d

        on e.departmentId = d.id)

select Department, Employee, Salary

from cte where rnk = 1;

A screenshot of a computer

Description automatically generated

/\*--------------------------------------------------------------------------\*/

/\*550. 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 (combination of columns with unique values) of this table.

This table shows the activity of players of some games.

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

Write a solution to report 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 result format is in the following example.

Example 1:

Input:

Activity table:

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

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

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

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

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

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

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

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

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

Output:

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

| fraction |

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

| 0.33 |

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

Explanation:

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

\*/

create table ltcd\_activity1 (player\_id int, device\_id int, event\_date date, games\_played int);

truncate table ltcd\_activity1;

insert into ltcd\_activity1 values

(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),

(1,2,'2016-03-03',5),

(1,2,'2016-03-04',6);

with cte as(select player\_id, min(event\_date) m\_date

from ltcd\_activity1

group by player\_id)

--select cast(count(distinct a.player\_id)\*1.0 / (select count(distinct player\_id) from cte) as decimal (10,2))

select count(distinct a.player\_id)

as fraction

from ltcd\_activity1 a

join cte c

on a.player\_id = c.player\_id

and a.event\_date = DATEADD(day, 1, c.m\_date);

A screenshot of a chat

Description automatically generated

/\*--------------------------------------------------------------------------\*/

[**570. Managers with at Least 5 Direct Reports**](https://leetcode.com/problems/managers-with-at-least-5-direct-reports/)

Solution:

/\* Write your T-SQL query statement below \*/

with cte as (

select managerid   from employee

group by managerid

having count(managerid) >= 5 )

select e1.name name from employee e1 join cte c

on e1.id = c.managerid;

sol2:

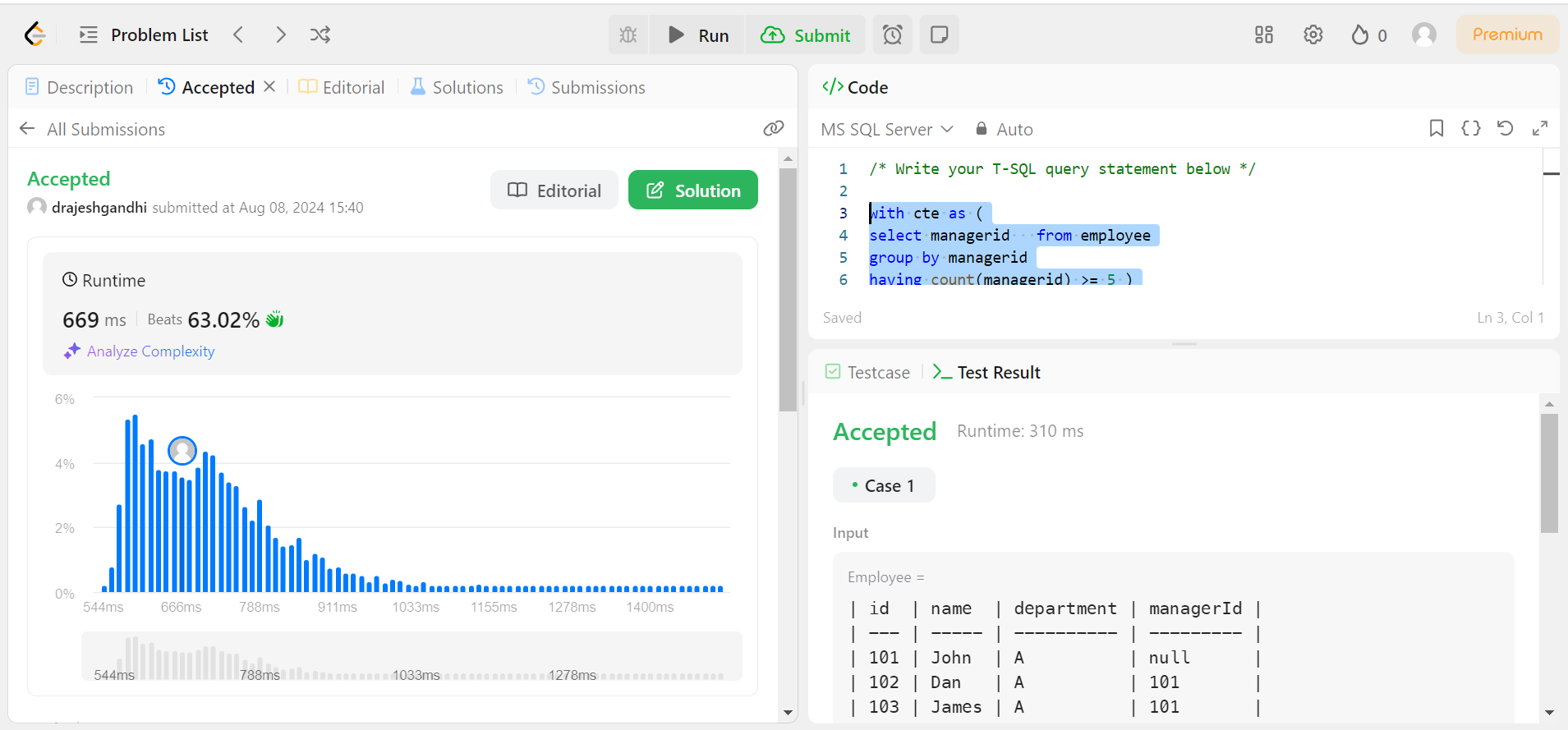
with cte as (

select managerid from employee

group by managerid

having count(managerid) >= 5 )

select e1.name from employee e1 where e1.id in ( select managerid from cte);



/\*--------------------------------------------------------------------------\*/

/\*585. Investments in 2016

Table: Insurance

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

| Column Name | Type |

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

| pid | int |

| tiv\_2015 | float |

| tiv\_2016 | float |

| lat | float |

| lon | float |

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

pid is the primary key (column with unique values) for this table.

Each row of this table contains information about one policy where:

pid is the policyholder's policy ID.

tiv\_2015 is the total investment value in 2015 and tiv\_2016 is the total investment value in 2016.

lat is the latitude of the policy holder's city. It's guaranteed that lat is not NULL.

lon is the longitude of the policy holder's city. It's guaranteed that lon is not NULL.

Write a solution to report the sum of all total investment values in 2016 tiv\_2016, for all policyholders who:

have the same tiv\_2015 value as one or more other policyholders, and

are not located in the same city as any other policyholder (i.e., the (lat, lon) attribute pairs must be unique).

Round tiv\_2016 to two decimal places.

The result format is in the following example.

Example 1:

Input:

Insurance table:

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

| 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 |

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

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 the same as the third and fourth records, and its location is 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 as 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.

\*/

create table ltcd\_insurance (pid int, tiv\_2015 float, tiv\_2016 float, lat float, lon float);

insert into ltcd\_insurance values

(1,10,5 ,10,10),

(2,20,20,20,20),

(3,10,30,20,20),

(4,10,40,40,40);

with cte as (select \*, count(tiv\_2015) over(partition by tiv\_2015) as tiv\_cnt

from ltcd\_insurance)

select round(sum(tiv\_2016),2) as tiv\_2016 from cte

where tiv\_cnt > 1 and concat(lat,lon) in (select concat(lat,lon) from ltcd\_insurance

group by lat,lon

having count(\*) = 1)

A screenshot of a chat

Description automatically generated

/\*--------------------------------------------------------------------------\*/

/\*602. Friend Requests II: Who Has the Most Friends

Table: RequestAccepted

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

| Column Name | Type |

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

| requester\_id | int |

| accepter\_id | int |

| accept\_date | date |

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

(requester\_id, accepter\_id) is the primary key (combination of columns with unique values) for this table.

This table contains the ID of the user who sent the request, the ID of the user who received the request, and the date when the request was accepted.

Write a solution to find the people who have the most friends and the most friends number.

The test cases are generated so that only one person has the most friends.

The result format is in the following example.

Example 1:

Input:

RequestAccepted table:

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

| 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 |

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

Output:

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

| id | num |

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

| 3 | 3 |

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

Explanation:

The person with id 3 is a friend of people 1, 2, and 4, so he has three 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. Could you find all these people in this case?

\*/

create table ltcd\_RequestAccepted (requester\_id int, accepter\_id int, accept\_date date);

insert into ltcd\_RequestAccepted values

(1,2,'2016/06/03'),

(1,3,'2016/06/08'),

(2,3,'2016/06/08'),

(3,4,'2016/06/09');

select \* from ltcd\_RequestAccepted;

with cte1 as (

select requester\_id from ltcd\_RequestAccepted

union all

select accepter\_id from ltcd\_RequestAccepted),

cte2 as

(select requester\_id, count(requester\_id) over (partition by requester\_id) cnt from cte1),

cte3 as

(select max(cnt) mx from cte2)

select distinct cte2.requester\_id id, cte2.cnt num from cte2 , cte3 where cte2.cnt = cte3.mx ;

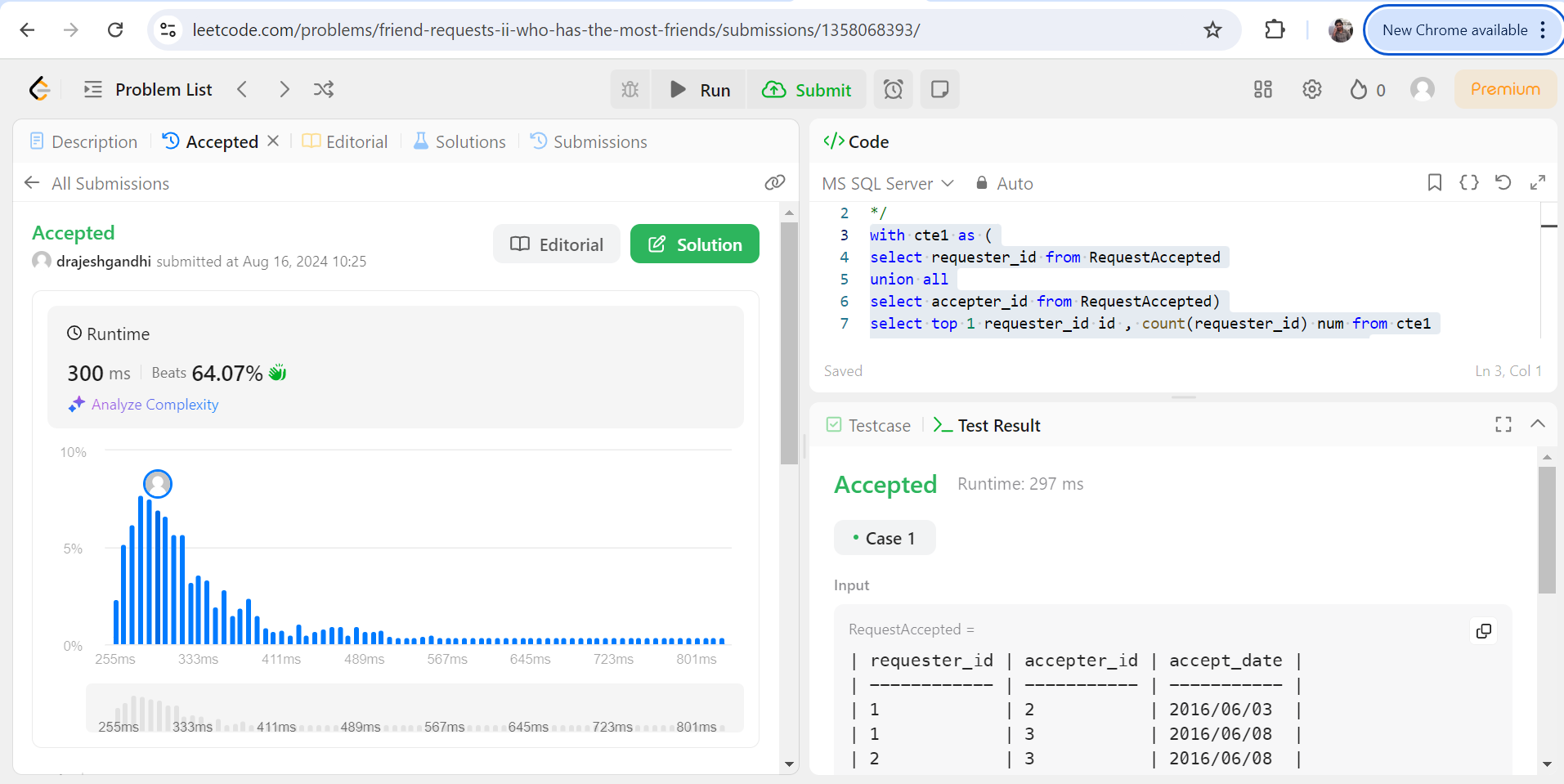
with cte1 as (

select requester\_id from ltcd\_RequestAccepted

union all

select accepter\_id from ltcd\_RequestAccepted)

select top 1 requester\_id id, count(requester\_id) num from cte1 group by requester\_id order by count(requester\_id) desc ;



/\*--------------------------------------------------------------------------\*/

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

Table: Tree

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

| Column Name | Type |

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

| id | int |

| p\_id | int |

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

id is the column with unique values for this table.

Each row of this table contains information about the id of a node and the id of its parent node in a tree.

The given structure is always a valid tree.

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 solution to report the type of each node in the tree.

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

The result format is in the following example.

**Example 1:**

A diagram of a network

Description automatically generated

**Input:**

Tree table:

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

| id | p\_id |

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

| 1 | null |

| 2 | 1 |

| 3 | 1 |

| 4 | 2 |

| 5 | 2 |

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

**Output:**

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

| id | type |

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

| 1 | Root |

| 2 | Inner |

| 3 | Leaf |

| 4 | Leaf |

| 5 | Leaf |

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

**Explanation:**

Node 1 is the root node because its parent node is null and it has child nodes 2 and 3.

Node 2 is an inner node because it has parent node 1 and child node 4 and 5.

Nodes 3, 4, and 5 are leaf nodes because they have parent nodes and they do not have child nodes.

**Example 2:**

A black number in a white circle

Description automatically generated

**Input:**

Tree table:

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

| id | p\_id |

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

| 1 | null |

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

**Output:**

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

| id | type |

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

| 1 | Root |

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

**Explanation:** 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 NOT IN

            (SELECT P\_ID FROM TREE WHERE P\_ID IS NOT NULL ) THEN 'Leaf'

        ELSE 'Inner'

        END TYPE

FROM TREE;

A screenshot of a computer

Description automatically generated

select distinct id,

case

when p\_id is null then 'Root'

when child\_id is null then 'Leaf'

else 'Inner'

end as type

from (

select t1.id, t1.p\_id, t2.id as child\_id

from ltcd\_tree t1

left join ltcd\_tree t2

on t1.id = t2.p\_id) a;

/\*--------------------------------------------------------------------------\*/

/\* 626. Exchange Seats

Table: Seat

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

| Column Name | Type |

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

| id | int |

| student | varchar |

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

id is the primary key (unique value) column for this table.

Each row of this table indicates the name and the ID of a student.

id is a continuous increment.

Write a solution to swap the seat id of every two consecutive students. If the number of students is odd, the id of the last student is not swapped.

Return the result table ordered by id in ascending order.

The result format is in the following example.

Example 1:

Input:

Seat table:

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

| id | student |

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

| 1 | Abbot |

| 2 | Doris |

| 3 | Emerson |

| 4 | Green |

| 5 | Jeames |

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

Output:

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

| id | student |

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

| 1 | Doris |

| 2 | Abbot |

| 3 | Green |

| 4 | Emerson |

| 5 | Jeames6 |

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

Explanation:

Note that if the number of students is odd, there is no need to change the last one's seat.\*/

create table ltcd\_seats (id int, student varchar(20));

truncate table ltcd\_seats;

insert into ltcd\_seats values

(1,'Abbot'),

(2,'Doris'),

(3,'Emerson'),

(4,'Green'),

(5,'Jeames');

select \* from ltcd\_seats;

with cte as

(select id, student, lead(student) over (order by id) next\_student, lag(student) over (order by id) prev\_student from ltcd\_seats)

/\* Write your T-SQL query statement below \*/

WITH CTE AS

(SELECT ID,

STUDENT,

LEAD(STUDENT) OVER (ORDER BY ID) NEXT\_STUDENT,

LAG(STUDENT) OVER (ORDER BY ID) PREV\_STUDENT

FROM LTCD\_SEATS)

SELECT ID,

CASE WHEN ID = (SELECT MAX(ID) FROM SEAT) and ID%2=1 THEN STUDENT

ELSE

CASE WHEN ID%2 = 1 THEN NEXT\_STUDENT

ELSE PREV\_STUDENT

END

END STUDENT

FROM CTE ;

A screenshot of a computer

Description automatically generated

/\*--------------------------------------------------------------------------\*/

/\* 1045. Customers Who Bought All Products

Table: Customer

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

| Column Name | Type |

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

| customer\_id | int |

| product\_key | int |

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

This table may contain duplicates rows.

customer\_id is not NULL.

product\_key is a foreign key (reference column) to Product table.

Table: Product

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

| Column Name | Type |

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

| product\_key | int |

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

product\_key is the primary key (column with unique values) for this table.

Write a solution to report the customer ids from the Customer table that bought all the products in the Product table.

Return the result table in any order.

The result format is in the following example.

Example 1:

Input:

Customer table:

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

| customer\_id | product\_key |

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

| 1 | 5 |

| 2 | 6 |

| 3 | 5 |

| 3 | 6 |

| 1 | 6 |

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

Product table:

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

| product\_key |

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

| 5 |

| 6 |

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

Output:

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

| customer\_id |

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

| 1 |

| 3 |

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

Explanation:

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

\*/

create table ltcd\_product (product\_key int primary key);

create table ltcd\_customer (customer\_id int, product\_key int foreign key References ltcd\_product(product\_key));

insert into ltcd\_product values (5),(6);

insert into ltcd\_customer values

(1,5),

(2,6),

(3,5),

(3,6),

(1,6);

with cte as

(select count(product\_key) prod\_cnt from ltcd\_product),

cte2 as

(select customer\_id, product\_key, concat(customer\_id,'|',product\_key) c\_s

from ltcd\_customer

where product\_key in (

select distinct product\_key from ltcd\_Product) ),

cte3 as (select customer\_id, count(distinct(c\_s)) CS from cte2 group by customer\_id)

select customer\_id from cte3,cte where CS = prod\_cnt;

with cteProducts as (

select customer\_id , count(distinct product\_key) as total\_products from ltcd\_Customer group by customer\_id)

select customer\_id from cteProducts where total\_products = (select count(\*) from ltcd\_Product);

A screenshot of a computer

Description automatically generated

/\*--------------------------------------------------------------------------\*/

/\* 1070. Product Sales Analysis III

Table: Sales

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

| Column Name | Type |

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

| sale\_id | int |

| product\_id | int |

| year | int |

| quantity | int |

| price | int |

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

(sale\_id, year) is the primary key (combination of columns with unique values) of this table.

product\_id is a foreign key (reference column) to Product table.

Each row of this table shows a sale on the product product\_id in a certain year.

Note that the price is per unit.

Table: Product

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

| Column Name | Type |

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

| product\_id | int |

| product\_name | varchar |

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

product\_id is the primary key (column with unique values) of this table.

Each row of this table indicates the product name of each product.

Write a solution to select the product id, year, quantity, and price for the first year of every product sold.

Return the resulting table in any order.

The result format is in the following example.

Example 1:

Input:

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 |

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

Output:

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

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

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

| 100 | 2008 | 10 | 5000 |

| 200 | 2011 | 15 | 9000 |

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

\*/

create table ltcd\_sales (sale\_id int, product\_id int, year int, quantity int, price int);

insert into ltcd\_sales values

(1,100,2008,10,5000),

(2,100,2009,12,5000),

(7,200,2011,15,9000);

create table ltcd\_products(product\_id int, product\_name varchar(50));

insert into ltcd\_products values

(100,'Nokia'),

(200,'Apple'),

(300,'Samsung');

Write a solution to select the product id, year, quantity, and price for the first year of every product sold.

SELECT

PRODUCT\_ID,

YEAR FIRST\_YEAR,

QUANTITY,

PRICE

FROM (

SELECT

PRODUCT\_ID,

YEAR,

QUANTITY,

PRICE,

MIN(YEAR) OVER (PARTITION BY PRODUCT\_ID ORDER BY YEAR) MINI

FROM LTCD\_SALES ) A

WHERE A.YEAR = A.MINI;

A screenshot of a chat

Description automatically generated

/\*--------------------------------------------------------------------------\*/

1158. Market Analysis I

Table: Users

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

| Column Name | Type |

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

| user\_id | int |

| join\_date | date |

| favorite\_brand | varchar |

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

user\_id is the primary key (column with unique values) 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 (column with unique values) of this table.

item\_id is a foreign key (reference column) 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 (column with unique values) of this table.

Write a solution to find for each user, the join date and the number of orders they made as a buyer in 2019.

Return the result table in any order.

The result format is in the following example.

Example 1:

Input:

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 |

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

Output:

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

| 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 |

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

create table ltcd\_users1 (user\_id int, join\_date date, favorite\_brand varchar(20));

create table ltcd\_orders1 ( order\_id int, order\_date date, item\_id int, buyer\_id int, seller\_id int);

create table ltcd\_items(item\_id int, item\_brand varchar(20));

insert into ltcd\_users1 values

(1,'2018-01-01','Lenovo'),

(2,'2018-02-09','Samsung'),

(3,'2018-01-19','LG'),

(4,'2018-05-21','HP');

insert into ltcd\_orders1 values

(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);

insert into ltcs\_items values

(1,'Samsung'),

(2,'Lenovo'),

(3,'LG'),

(4,'HP');

select distinct buyer\_id, join\_date, count(order\_date) over (partition by buyer\_id order by join\_date) orders\_in\_2019 from

(select user\_id buyer\_id,join\_date,order\_date

from ltcd\_users1 left join ltcd\_orders1 on

user\_id = buyer\_id

and year(order\_date) = 2019) a

with cte as

(select buyer\_id, year(order\_date) y\_o\_r, count(order\_date) cnt

from ltcd\_orders1

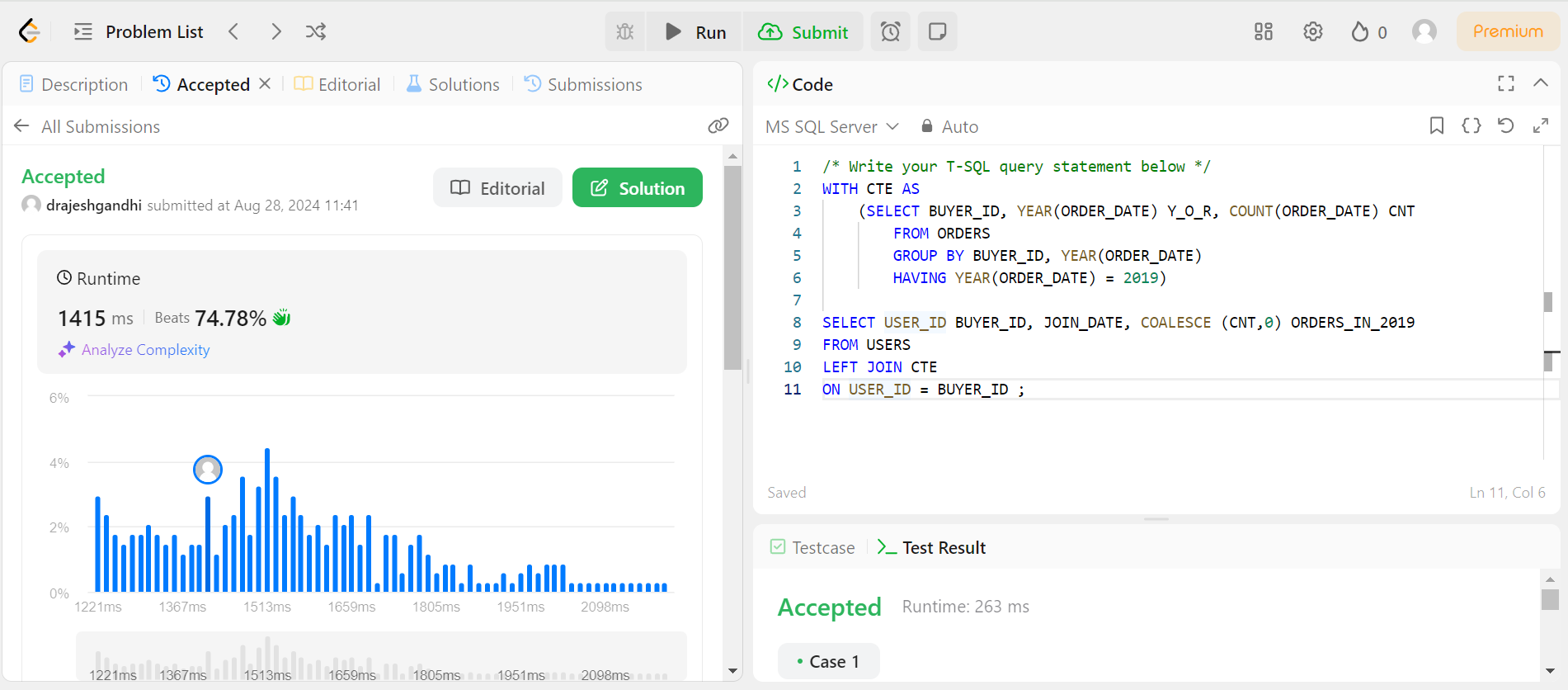
group by buyer\_id, year(order\_date)

having year(order\_date) = 2019)

select user\_id buyer\_id, join\_date, COALESCE (cnt,0) orders\_in\_2019

from ltcd\_users1

left join cte on user\_id = buyer\_id ;



/\*--------------------------------------------------------------------------\*/

1164. 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 (combination of columns with unique values) 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 a solution to find the prices of all products on 2019-08-16. Assume the price of all products before any change is 10.

Return the result table in any order.

The result format is in the following example.

Example 1:

Input:

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 |

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

Output:

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

| product\_id | price |

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

| 2 | 50 |

| 1 | 35 |

| 3 | 10 |

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

create table ltcd\_products1 (product\_id int, new\_price int, change\_date date);

insert into ltcd\_products1 values

(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');

with cte as

(select product\_id, new\_price price from

(select \*, rank() over(partition by product\_id order by change\_date desc) rnk from ltcd\_products1

where change\_date <= '2019-08-16') a

where rnk = 1)

select distinct l.product\_id, coalesce(c.price,10) price from ltcd\_products1 l left join cte c on l.product\_id = c.product\_id;

A screenshot of a computer

Description automatically generated

/\*--------------------------------------------------------------------------\*/

1321. Restaurant Growth

Table: Customer

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

| Column Name | Type |

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

| customer\_id | int |

| name | varchar |

| visited\_on | date |

| amount | int |

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

In SQL,(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) has 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).

Compute the moving average of how much the customer paid in a seven days window (i.e., current day + 6 days before). average\_amount should be rounded to two decimal places.

Return the result table ordered by visited\_on in ascending order.

The result format is in the following example.

Example 1:

Input:

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 |

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

Output:

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

| 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 |

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

Explanation:

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

create table ltcd\_customers (customer\_id int, name varchar(30), visited\_on date, amount int);

insert into ltcd\_customers values

(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);

with cte as

(select distinct visited\_on, DATEADD(DAY,-6,VISITED\_ON) six\_days\_old,

sum(amount) over (partition by visited\_on order by visited\_on) amount1 from ltcd\_customers

)

select visited\_on, sum(amount1) over (order by visited\_on ROWS between 6 preceding and current row) amount,

cast(sum(amount1\*1.0) over (order by visited\_on ROWS between 6 preceding and current row)/7 as decimal (10,2)) average\_amount

from cte

ORDER BY VISITED\_ON

OFFSET 6 ROWS

A screenshot of a chat

Description automatically generated

/\*--------------------------------------------------------------------------\*/ 1174. 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 column of unique values 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 customer's preferred delivery date is the same as the order date, then the order is called immediate; otherwise,

it is called scheduled.

The first order of a customer is the order with the earliest order date that the customer made. It is guaranteed that a

customer has precisely one first order.

Write a solution to find the percentage of immediate orders in the first orders of all customers, rounded to 2 decimal places.

The result format is in the following example.

Example 1:

Input:

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 |

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

Output:

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

| immediate\_percentage |

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

| 50.00 |

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

Explanation:

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.

create table ltcd\_delivery (delivery\_id int, customer\_id int, order\_date date, customer\_pref\_delivery\_date date);

insert into ltcd\_delivery values

(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');

with cte1 as

(select \*, rank() over (partition by customer\_id order by order\_date) rnk from ltcd\_delivery),

cte2 as

(select count(\*) first\_order\_cnt from cte1 where rnk = 1),

cte3 as

(select count(\*) immediate\_order\_cnt from cte1 where rnk = 1 and order\_date = customer\_pref\_delivery\_date)

select cast((immediate\_order\_cnt\*1.0/first\_order\_cnt)\*100 as decimal(10,2)) immediate\_percentage from cte2, cte3;

A screenshot of a computer

Description automatically generated

/\*--------------------------------------------------------------------------\*/

1193. 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.

Return the result table in any order.

The query result format is in the following example.

Example 1:

Input:

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 |

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

Output:

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

| 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 |

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

create table ltcd\_transactions1 (id int, country varchar(50),

state varchar(20) not null check (state in ('approved', 'declined')), amount int, trans\_date date);

insert into ltcd\_transactions1 values

(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');

select cast (MONTH(trans\_date) as varchar(2)) from ltcd\_transactions1

Select right(replicate('0',2) + LEFT(MONTH(trans\_date),2) ,2) month from ltcd\_transactions1

--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.

select

concat(YEAR(trans\_date),'-',right(replicate('0',2) + LEFT(MONTH(trans\_date),2) ,2)) month ,country,

count(\*) trans\_count,

count(case when state = 'approved' then 1 else NULL end) approved\_count,

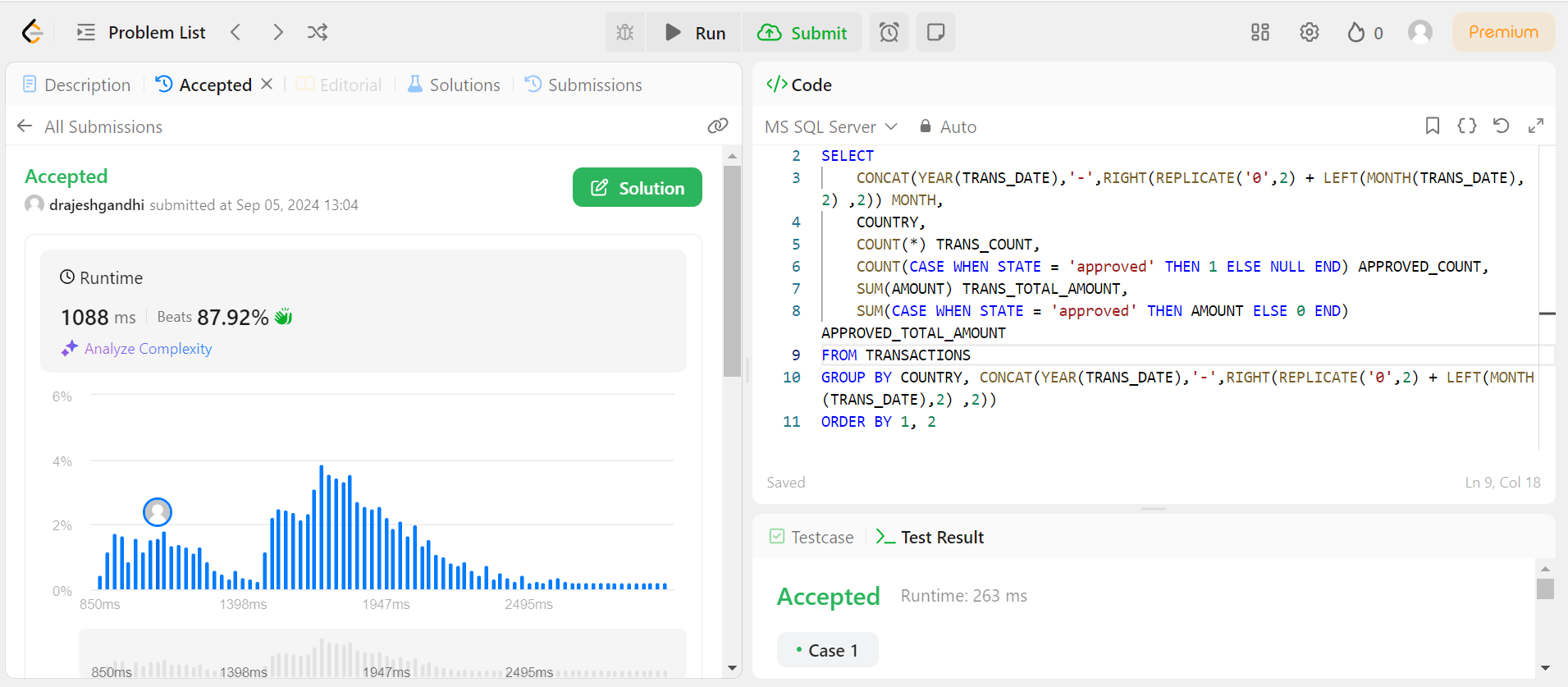
sum(amount) trans\_total\_amount,

sum(case when state = 'approved' then amount else 0 end) approved\_total\_amount

from ltcd\_transactions1

group by country, concat(YEAR(trans\_date),'-',right(replicate('0',2) + LEFT(MONTH(trans\_date),2) ,2))

order by 1,2



/\*--------------------------------------------------------------------------\*/

1204. Last Person to Fit in the Bus

Table: Queue

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

| Column Name | Type |

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

| person\_id | int |

| person\_name | varchar |

| weight | int |

| turn | int |

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

person\_id column contains unique values.

This table has the information about all people waiting for a bus.

The person\_id and turn columns will contain all numbers from 1 to n, where n is the number of rows in the table.

turn determines the order of which the people will board the bus, where turn=1 denotes the first person to board and turn=n denotes the last person to board.

weight is the weight of the person in kilograms.

There is a queue of people waiting to board a bus. However, the bus has a weight limit of 1000 kilograms, so there may be some people who cannot board.

Write a solution to find the person\_name of the last person that can fit on the bus without exceeding the weight limit. The test cases are generated such that the first person does not exceed the weight limit.

The result format is in the following example.

Example 1:

Input:

Queue table:

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

| person\_id | person\_name | weight | turn |

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

| 5 | Alice | 250 | 1 |

| 4 | Bob | 175 | 5 |

| 3 | Alex | 350 | 2 |

| 6 | John Cena | 400 | 3 |

| 1 | Winston | 500 | 6 |

| 2 | Marie | 200 | 4 |

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

Output:

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

| person\_name |

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

| John Cena |

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

Explanation: The folowing table is ordered by the turn for simplicity.

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

| Turn | ID | Name | Weight | Total Weight |

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

| 1 | 5 | Alice | 250 | 250 |

| 2 | 3 | Alex | 350 | 600 |

| 3 | 6 | John Cena | 400 | 1000 | (last person to board)

| 4 | 2 | Marie | 200 | 1200 | (cannot board)

| 5 | 4 | Bob | 175 | \_\_\_ |

| 6 | 1 | Winston | 500 | \_\_\_ |

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

create table ltcd\_queue (person\_id int, person\_name varchar(50), weight int, turn int);

insert into ltcd\_queue values

(5,'Alice',250,1),

(4,'Bob',175,5),

(3,'Alex',350,2),

(6,'John Cena',400,3),

(1,'Winston',500,6),

(2,'Marie',200,4);

with cte as

(select turn, person\_id, person\_name, weight, sum(weight) over (order by turn ROWS between unbounded preceding and current row) Total\_weight from ltcd\_queue)

select top 1 person\_name from cte where total\_weight <= 1000 order by total\_weight desc;

A screenshot of a chat

Description automatically generated

<https://www.geeksforgeeks.org/sql-rows-between/>

The purpose of the ROWS clause is **to specify the window frame in relation to the current row**. The syntax is:

ROWS BETWEEN lower\_bound AND upper\_bound

The bounds can be any of these five options:

* UNBOUNDED PRECEDING – All rows before the current row.
* n PRECEDING – n rows **before** the current row.
* CURRENT ROW – Just the current row.
* n FOLLOWING – n rows **after** the current row.
* UNBOUNDED FOLLOWING – All rows after the current row.

<https://learnsql.com/blog/sql-window-functions-rows-clause/>

/\*--------------------------------------------------------------------------\*/

185. Department Top Three Salaries

Table: Employee

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

| Column Name | Type |

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

| id | int |

| name | varchar |

| salary | int |

| departmentId | int |

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

id is the primary key (column with unique values) for this table.

departmentId is a foreign key (reference column) of the ID from the Department table.

Each row of this table indicates the ID, name, and salary of an employee. It also contains the ID of their department.

Table: Department

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

| Column Name | Type |

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

| id | int |

| name | varchar |

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

id is the primary key (column with unique values) for this table.

Each row of this table indicates the ID of a department and its name.

A company's executives are interested in seeing who earns the most money in each of the company's departments. A high earner in a department is an employee who has a salary in the top three unique salaries for that department.

Write a solution to find the employees who are high earners in each of the departments.

Return the result table in any order.

The result format is in the following example.

Example 1:

Input:

Employee table:

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

| 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 |

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

Department table:

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

| id | name |

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

| 1 | IT |

| 2 | Sales |

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

Output:

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

| Department | Employee | Salary |

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

| IT | Max | 90000 |

| IT | Joe | 85000 |

| IT | Randy | 85000 |

| IT | Will | 70000 |

| Sales | Henry | 80000 |

| Sales | Sam | 60000 |

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

Explanation:

In the IT department:

- Max earns the highest unique salary

- Both Randy and Joe earn the second-highest unique salary

- Will earns the third-highest unique salary

In the Sales department:

- Henry earns the highest salary

- Sam earns the second-highest salary

- There is no third-highest salary as there are only two employees

create table ltcd\_employee1 (id int,name varchar(50), salary int, departmentId int);

insert into ltcd\_employee1 values

(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);

create table ltcd\_department1(id int, name varchar(50));

insert into ltcd\_department1 values (1,'IT'), (2,'Sales');

select d.name, a.name, a.salary from

(select departmentId, salary, name, dense\_rank() over (partition by departmentId order by salary desc) rnk from ltcd\_employee1)

a join ltcd\_department1 d

on a.departmentId=d.id

and a.rnk <=3;

/\* Write your T-SQL query statement below \*/

SELECT

    D.NAME DEPARTMENT,  A.NAME EMPLOYEE, A.SALARY

FROM

    (SELECT DEPARTMENTID, SALARY, NAME,

            DENSE\_RANK() OVER (PARTITION BY DEPARTMENTID ORDER BY SALARY DESC) RNK

            FROM EMPLOYEE) A

JOIN DEPARTMENT D

ON A.DEPARTMENTID=D.ID

AND A.RNK <=3;

A screenshot of a computer

Description automatically generated

/\*--------------------------------------------------------------------------\*/

601. Human Traffic of Stadium

Table: Stadium

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

| Column Name | Type |

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

| id | int |

| visit\_date | date |

| people | int |

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

visit\_date is the column with unique values for this table.

Each row of this table contains the visit date and visit id to the stadium with the number of people during the visit.

As the id increases, the date increases as well.

Write a solution to display the records with three or more rows with consecutive id's, and the number of people is greater than or equal to 100 for each.

Return the result table ordered by visit\_date in ascending order.

The result format is in the following example.

Example 1:

Input:

Stadium table:

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

| 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 | 2017-01-09 | 188 |

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

Output:

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

| id | visit\_date | people |

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

| 5 | 2017-01-05 | 145 |

| 6 | 2017-01-06 | 1455 |

| 7 | 2017-01-07 | 199 |

| 8 | 2017-01-09 | 188 |

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

Explanation:

The four rows with ids 5, 6, 7, and 8 have consecutive ids and each of them has >= 100 people attended. Note that row 8 was included even though the visit\_date was not the next day after row 7.

The rows with ids 2 and 3 are not included because we need at least three consecutive ids.

create table ltcd\_stadium (id int, visit\_date date, people int);

insert into ltcd\_stadium values

(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,'2017-01-09',188 );

with cte as

(select \*,

id - row\_number () over (order by visit\_date) rnk

from ltcd\_stadium

where people >= 100),

cte1 as

(select \*, count(rnk) over (partition by rnk) cnt from cte )

select id, visit\_date, people from cte1 where cnt >=3

A screenshot of a computer

Description automatically generated

/\*--------------------------------------------------------------------------\*/

1917: Leetcodify Friends Suggestion

create table ltcd\_listens (user\_id int, song\_id int, day date);

--This table may contain duplicates. Each row indicates that a user has listened to a song on a particular day.

create table ltcd\_friendship (user1\_id int, user2\_id int);

--In SQL, (user1\_id, user2\_id) is the primary key. Each row of the table indicates that user1\_id and user2\_id are friends.

--Note: user1\_id < user2\_id

/\*Recommend friends to Leetcodify users. Recommend user x to user y if:

User X and Y are not in Friends table

User X and Y listened to same three or more different songs on the same day.

Note that friend recommendation should be unidirectional, meaning User X and Y should be recommended to each other.

The result table should have recommended both User X to User Y and User Y to User X.

Also note that result table should not contain duplicates (i.e., user Y should not be recommended to User X multiple times).

Return the results in any format.

truncate table ltcd\_listens;

\*/

insert into ltcd\_listens values

(1,10,'2021-03-15'),

(1,11,'2021-03-15'),

(1,12,'2021-03-15'),

(2,10,'2021-03-15'),

(2,11,'2021-03-15'),

(2,12,'2021-03-15'),

(3,10,'2021-03-15'),

(3,11,'2021-03-15'),

(3,12,'2021-03-15'),

(4,10,'2021-03-15'),

(4,11,'2021-03-15'),

(4,13,'2021-03-15'),

(5,10,'2021-03-16'),

(5,11,'2021-03-16'),

(5,12,'2021-03-16');

insert into ltcd\_friendship values

(1,2);

/\*Expected Output

user\_id | recommended\_id

1 | 3

2 | 3

3 | 1

3 | 2

Explanation:

User1 and User2 listened to songs 10, 11, 12 on same day but they are already friends.

User1 and User3 listened to same songs 10, 11, 12 on same day. Since they are not in friends table,

we recommend them to each other.

User1 and User4 did not listen to same three songs

User1 and User5 listened to same songs 10, 11, 12 but on different day

Similarly we can see that Users 2 and 3 also listened to same songs 10, 11, 12 on same day and not in friends,

so we recommend them each other.

\*/

with unique\_listens as

(select distinct \* from ltcd\_listens),

valid\_listens as

(select user\_id, day, count(1) cnt from

unique\_listens

group by user\_id, day

having count(1) >= 3 ),

friends as

(select l1.user\_id user\_id, l2.user\_id recommended , l1.day day, l2.day day1 , count(1) cnt

from unique\_listens l1 join unique\_listens l2

on l1.user\_id < l2.user\_id

and l1.day = l2.day and l1.song\_id = l2.song\_id

join valid\_listens v1 on l1.user\_id = v1.user\_id and l1.day = v1.day

group by l1.user\_id , l2.user\_id , l1.day, l2.day

having count(1) > 2 )

select user\_id, recommended from friends, ltcd\_friendship

where concat (user\_id,recommended) <> concat (user1\_id,user2\_id)

union

select recommended, user\_id from friends, ltcd\_friendship

where concat (user\_id,recommended) <> concat (user1\_id,user2\_id);

/\*--------------------------------------------------------------------------\*/

/\*--------------------------------------------------------------------------\*/