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

/\* Leetcode Problems \*/

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

/\* 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 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 desc;

| 3.65 | 3 |

| 3.65 | 3 |

| 3.50 | 4 |

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

Solution:

A screenshot of a computer

Description automatically generated

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

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

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

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

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

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

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

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