# Q 101.

Table: UserActivity

|  |  |
| --- | --- |
| Column Name | Type |
| username | varchar |
| activity | varchar |
| startDate | Date |
| endDate | Date |

There is no primary key for this table. It may contain duplicates.

This table contains information about the activity performed by each user in a period of time. A person with a username performed an activity from startDate to endDate.

Write an SQL query to show the second most recent activity of each user.

If the user only has one activity, return that one. A user cannot perform more than one activity at the same time.

Return the result table in any order.

The query result format is in the following example.

Input:

UserActivity table:

|  |  |  |  |
| --- | --- | --- | --- |
| username | activity | startDate | endDate |
| Alice | Travel | 2020-02-12 | 2020-02-20 |
| Alice | Dancing | 2020-02-21 | 2020-02-23 |
| Alice | Travel | 2020-02-24 | 2020-02-28 |
| Bob | Travel | 2020-02-11 | 2020-02-18 |

Output:

|  |  |  |  |
| --- | --- | --- | --- |
| username | activity | startDate | endDate |
| Alice | Dancing | 2020-02-21 | 2020-02-23 |
| Bob | Travel | 2020-02-11 | 2020-02-18 |

Explanation:

The most recent activity of Alice is Travel from 2020-02-24 to 2020-02-28, before that she was dancing from 2020-02-21 to 2020-02-23.

Bob only has one record, we just take that one.

create table UserActivity

(username varchar(20),activity varchar(20),startDate date,endDate date);

insert into UserActivity values

('Alice','Travel','2020-02-12','2020-02-20'),

('Alice','Dancing','2020-02-21','2020-02-23'),

('Alice','Travel','2020-02-24','2020-02-28'),

('Bob','Travel','2020-02-11','2020-02-18');

with cte as (

select username,activity,startDate,endDate,

row\_number() over(partition by username order by enddate desc) rw,

count(\*)over(partition by username)cnt from UserActivity

)

-- select \* from cte;

select username,activity,startDate,endDate from cte where (  rw = 2 and cnt <> 1 ) or (rw = 1 and cnt = 1);

# Q102.

**Table: UserActivity**

|  |  |
| --- | --- |
| Column Name | Type |
| username | varchar |
| activity | varchar |
| startDate | Date |
| endDate | Date |

There is no primary key for this table. It may contain duplicates.

This table contains information about the activity performed by each user in a period of time. A person with a username performed an activity from startDate to endDate.

Write an SQL query to show the second most recent activity of each user.

If the user only has one activity, return that one. A user cannot perform more than one activity at the same time.

Return the result table in any order.

The query result format is in the following example.

Input:

**UserActivity table:**

|  |  |  |  |
| --- | --- | --- | --- |
| username | activity | startDate | endDate |
| Alice | Travel | 2020-02-12 | 2020-02-20 |
| Alice | Dancing | 2020-02-21 | 2020-02-23 |
| Alice | Travel | 2020-02-24 | 2020-02-28 |
| Bob | Travel | 2020-02-11 | 2020-02-18 |

Output:

|  |  |  |  |
| --- | --- | --- | --- |
| username | activity | startDate | endDate |
| Alice | Dancing | 2020-02-21 | 2020-02-23 |
| Bob | Travel | 2020-02-11 | 2020-02-18 |

**Explanation:**

The most recent activity of Alice is Travel from 2020-02-24 to 2020-02-28, before that she was dancing from 2020-02-21 to 2020-02-23.

Bob only has one record, we just take that one.

create table UserActivity

(username varchar(20),activity varchar(20),startDate date,endDate date);

insert into UserActivity values

('Alice','Travel','2020-02-12','2020-02-20'),

('Alice','Dancing','2020-02-21','2020-02-23'),

('Alice','Travel','2020-02-24','2020-02-28'),

('Bob','Travel','2020-02-11','2020-02-18');

with cte as (

select username,activity,startDate,endDate,

row\_number() over(partition by username order by enddate desc) rw,

count(\*)over(partition by username)cnt from UserActivity

)

-- select \* from cte;

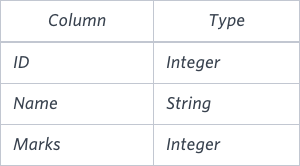
select username,activity,startDate,endDate from cte where (  rw = 2 and cnt <> 1 ) or (rw = 1 and cnt = 1);

# Q103.

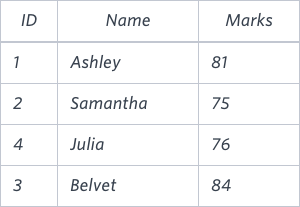
Query the Name of any student in STUDENTS who scored higher than 75 Marks. Order your output by the last three characters of each name. If two or more students both have names ending in the same last three characters (i.e.: Bobby, Robby, etc.), secondary sort them by ascending ID.

Input Format

The STUDENTS table is described as follows:



The Name column only contains uppercase (A-Z) and lowercase (a-z) letters. Sample Input



Sample Output Ashley

Julia Belvet

Explanation

Only Ashley, Julia, and Belvet have Marks > 75 . If you look at the last three characters of each of their names, there are no duplicates and 'ley' < 'lia' < 'vet'.

CREATE TABLE students ( id int, name varchar(10), marks int);

truncate table students;

insert into students values

 (1, 'Ashley',     81 ),

(2, 'Samantha',  75 ),

  (3, 'Julia',     76 ),

 ( 4, 'Belvet',     84 );

SELECT

    name

FROM

     students

WHERE

     marks > 75

ORDER BY

     RIGHT(name, 3)

   , id ;

# Q104.

Write a query that prints a list of employee names (i.e.: the name attribute) for employees in Employee having a salary greater than $2000 per month who have been employees for less than 10 months. Sort your result by ascending employee\_id.

Input Format

The Employee table containing employee data for a company is described as follows:

where employee\_id is an employee's ID number, name is their name, months is the total number of months they've been working for the company, and salary is the their monthly salary.

Sample Input



Sample Output Angela Michael

Todd Joe

Explanation

Angela has been an employee for 1 month and earns $3443 per month. Michael has been an employee for 6 months and earns $2017 per month. Todd has been an employee for 5 months and earns $3396 per month.

Joe has been an employee for 9 months and earns $3573 per month. We order our output by ascending employee\_id.

create table Employee

(employee\_id int,

name varchar(20),

months int,

salary int);

insert into Employee values

(12228,'Rose',15,1968),

(33645,'Angela',1,3443),

(45692,'Frank',17,1608),

(56118,'Patrick',7,1345),

(59725,'Lisa',11,2330),

(74197,'Kimberly',16,4372),

(78454,'Bonnie',8,1771),

(83565,'Michael',6,2017),

(98607,'Todd',5,3396),

(99989,'Joe',9,3573);

select name from Employee where salary > 2000 and months < 10 order by name;

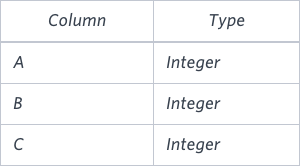
# Q105

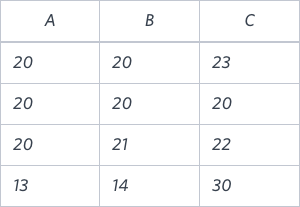
Write a query identifying the type of each record in the TRIANGLES table using its three side lengths. Output one of the following statements for each record in the table:

* Equilateral: It's a triangle with sides of equal length.
* Isosceles: It's a triangle with sides of equal length.
* Scalene: It's a triangle with sides of differing lengths.
* Not A Triangle: The given values of A, B, and C don't form a triangle.

Input Format

The TRIANGLES table is described as follows:



Each row in the table denotes the lengths of each of a triangle's three sides. Sample Input

Sample Output Isosceles Equilateral Scalene

Not A Triangle

Explanation

Values in the tuple(20,20,23) form an Isosceles triangle, because A ≡ B.

Values in the tuple(20,20,20) form an Equilateral triangle, because A ≡ B ≡ C . Values in the tuple(20,21,22) form a Scalene triangle, because A ≠ B ≠C .Values in the tuple (13,14,30) cannot form a triangle because the combined value of sides A and B is not larger than that of side C .

create table Triangle(A int,B int,C int);

insert  into Triangle VALUES

(20,20,23),

(20,20,20),

(20,21,22),

(13,14,30);

with cte as

(select A,B,C,

case

when A=B and B=C  then 'Equilateral'

when A=B or B=C or A=C then ' Isosceles'

when A+b <= C or A+C <= B or B+C <= A then 'Not A Triangle'

when A<>B and B<>C and C<>A then 'Scalene'

end triangle

from Triangle)

select triangle from cte;

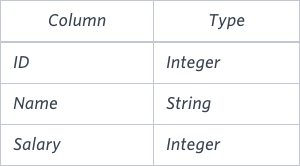
# Q106.

Samantha was tasked with calculating the average monthly salaries for all employees in the EMPLOYEES table, but did not realise her keyboard's 0 key was broken until after completing the calculation. She wants your help ﬁnding the difference between her miscalculation (using salaries with any zeros removed), and the actual average salary.

Write a query calculating the amount of error (i.e.: actual - miscalculated average monthly salaries), and round it up to the next integer.

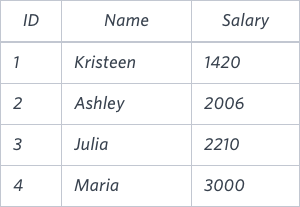
Input Format

The EMPLOYEES table is described as follows:



Note: Salary is per month. Constraints

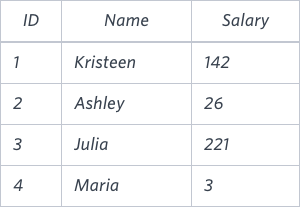
1000<salary < 10^5 Sample Input



Sample Output 2061

Explanation

The table below shows the salaries without zeros as they were entered by Samantha:



Samantha computes an average salary of 98.00 . The actual average salary is 2159.00.

The resulting error between the two calculations is 2159.00-98.00 = 2061.00. Since it is equal to the integer 2061, it does not get rounded up.

create table  EMPLOYEES

(id int,name varchar(20),salary int);

insert into  EMPLOYEES values

(1,'Kristeen',1420),

(2,'Ashley',2006),

(3,'Julia',2210),

(4,'Maria',3000);

SELECT (avg(salary)) - (avg(replace(salary, '0', '')))

AS avg\_salary\_diff FROM EMPLOYEES;

# Q107.

We deﬁne an employee's total earnings to be their monthly salary \* months worked, and the maximum total earnings to be the maximum total earnings for any employee in the Employee table. Write a query to ﬁnd the maximum total earnings for all employees as well as the total number of employees who have maximum total earnings. Then print these values as 2 space-separated integers.

Level - Easy

Hint - Use Aggregation functions Input Format

The Employee table containing employee data for a company is described as follows:

where employee\_id is an employee's ID number, name is their name, months is the total number of months they've been working for the company, and salary is the their monthly salary.

Sample Input



Sample Output 69952 1

Explanation:

The table and earnings data is depicted in the following diagram:



The maximum earnings value is 69952. The only employee with earnings= 69952 is Kimberly, so we print the maximum earnings value (69952) and a count of the number of employees who have earned

$69952 (which is 1) as two space-separated values.

create table Employee

(empployee\_id int,name varchar(20),months int,salary bigint);

insert into Employee values

(12228,'Rose'   ,15,1968),

(33645,'Angela' ,1,3443),

(45692,'Frank'  ,17,1608),

(56118,'Patrick',7,1345),

(59725,'Lisa'   ,11,2330),

(74197,'Kimberly',16,4372),

(78454,'Bonnie' ,8,1771),

(83565,'Michael',6,2017),

(98607,'Todd',5,3396),

(99989,'Joe',9,3573);

select earnings,count(earnings) count from (

select empployee\_id ,name,months ,salary,(months\*salary) earnings,

dense\_rank() over (order by months\*salary DESC) rnk from Employee)a

where a.rnk  =1 group by earnings;

# Q108.

Generate the following two result sets:

1. Query an alphabetically ordered list of all names in OCCUPATIONS, immediately followed by the ﬁrst letter of each profession as a parenthetical (i.e.: enclosed in parentheses). For example: AnActorName(A), ADoctorName(D), AProfessorName(P), and ASingerName(S).

Query the number of occurrences of each occupation in OCCUPATIONS. Sort the occurrences in ascending order, and output them in the following format:

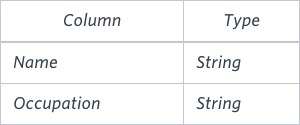
Level - Medium

There are a total of [occupation\_count] [occupation]s.

1. where [occupation\_count] is the number of occurrences of an occupation in OCCUPATIONS and [occupation] is the lowercase occupation name. If more than one Occupation has the same [occupation\_count], they should be ordered alphabetically.

Note: There will be at least two entries in the table for each type of occupation. Input Format

The OCCUPATIONS table is described as follows:



Occupation will only contain one of the following values: Doctor, Professor, Singer or Actor. Sample Input

An OCCUPATIONS table that contains the following records:

Sample Output Ashely(P) Christeen(P) Jane(A) Jenny(D) Julia(A) Ketty(P) Maria(A) Meera(S) Priya(S) Samantha(D)

There are a total of 2 doctors. There are a total of 2 singers. There are a total of 3 actors. There are a total of 3 professors.

Hint -

The results of the ﬁrst query are formatted to the problem description's speciﬁcations.

The results of the second query are ascendingly ordered ﬁrst by number of names corresponding to each profession (2<= 2<=3<=3), and then alphabetically by profession (doctor <= singer , and actor <= professor ).

create table Occupations

(Name varchar(10),Occupation varchar(10));

insert into Occupations values

('Samantha','Doctor'),

('Julia','Actor'),

('Maria','Actor'),

('Meera','Singer'),

('Ashley','Professor'),

('Ketty','Professor'),

('Christeen','Professor'),

('Jane','Actor'),

('Jenny','Doctor'),

('Priya','Singer');

select concat(name,' (',left(Occupation,1),')') from Occupations order by name;

select

concat('There are a total of ',

count(Name),' ', Occupation,'s.')

 from Occupations

 group by Occupation order by count(Name);

(select concat(name,' (',left(Occupation,1),')') output from Occupations order by Name)

union all

(select

concat('There are a total of ',

count(Name),' ', Occupation,'s.') output

 from Occupations

 group by Occupation order by count(Name));

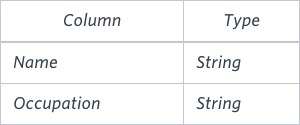
# Q109 .

Pivot the Occupation column in OCCUPATIONS so that each Name is sorted alphabetically and displayed underneath its corresponding Occupation. The output column headers should be Doctor, Professor, Singer, and Actor, respectively.

Note: Print NULL when there are no more names corresponding to an occupation.

Input Format

The OCCUPATIONS table is described as follows:



Occupation will only contain one of the following values: Doctor, Professor, Singer or Actor. Sample Input



Sample Output

Jenny Ashley Meera Jane Samantha Christeen Priya Julia NULL Ketty NULL Maria

Hint -

The ﬁrst column is an alphabetically ordered list of Doctor names.

The second column is an alphabetically ordered list of Professor names. The third column is an alphabetically ordered list of Singer names.

The fourth column is an alphabetically ordered list of Actor names.

The empty cell data for columns with less than the maximum number of names per occupation (in this case, the Professor and Actor columns) are ﬁlled with NULL values.

create table Occupations

(Name varchar(10),Occupation varchar(10));

insert into Occupations values

('Samantha','Doctor'),

('Julia','Actor'),

('Maria','Actor'),

('Meera','Singer'),

('Ashley','Professor'),

('Ketty','Professor'),

('Christeen','Professor'),

('Jane','Actor'),

('Jenny','Doctor'),

('Priya','Singer');

 with cte as (select row\_number() over(partition by Occupation order by Name ASC) as rnk,Occupation,Name

from Occupations),

cte2 as (select rnk,

max(case  when Occupation =  'Doctor' then Name end) as Doctor,

max(case  when Occupation = 'Professor' then Name end) as Professor,

max(case  when Occupation = 'Singer' then Name end) as Singer,

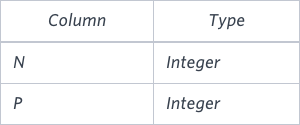
max(case  when Occupation = 'Actor' then Name end) as Actor

from cte group  by rnk)

select Doctor,Professor,Singer,Actor from cte2;

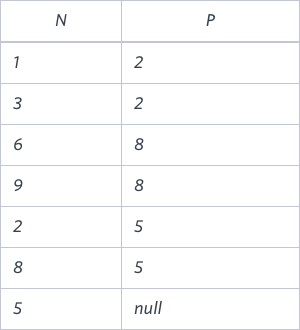
# Q110.

You are given a table, BST, containing two columns: N and P, where N represents the value of a node in Binary Tree, and P is the parent of N.



Write a query to ﬁnd the node type of Binary Tree ordered by the value of the node. Output one of the following for each node:

* Root: If node is root node.
* Leaf: If node is leaf node.
* Inner: If node is neither root nor leaf node. Sample Input

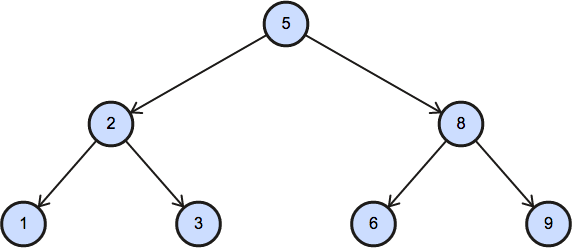


Sample Output 1 Leaf

1. Inner
2. Leaf
3. Root
4. Leaf
5. Inner
6. Leaf

Explanation

The Binary Tree below illustrates the sample:



create table BST(N int,P int);

insert  into BST VALUES

(1,2),(3,2),(6,8),(9,8),(2,5),(8,5),(5,NULL);

select concat(N,' ',

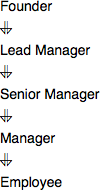
case when P is NULL then 'Root'

when N in (select P from BST) then 'Inner'

else 'Leaf'  end ) as output

from BST order by N;

# Q111 .

Amber's conglomerate corporation just acquired some new companies. Each of the companies

follows this hierarchy:

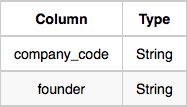
Given the table schemas below, write a query to print the company\_code, founder name, total number of lead managers, total number of senior managers, total number of managers, and total number of employees. Order your output by ascending company\_code.

Level - Medium Note:

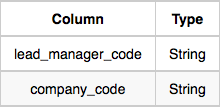
* The tables may contain duplicate records.
* The company\_code is string, so the sorting should not be numeric. For example, if the company\_codes are C\_1, C\_2, and C\_10, then the ascending company\_codes will be C\_1, C\_10, and C\_2.

Input Format

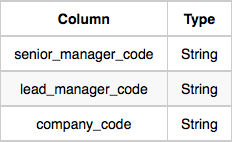
The following tables contain company data:

* Company: The company\_code is the code of the company and founder is the founder of the

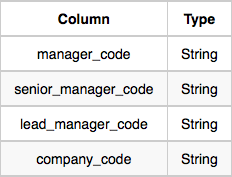
company.

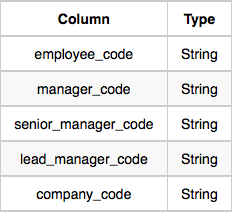
* Lead\_Manager: The lead\_manager\_code is the code of the lead manager, and the

company\_code is the code of the working company.

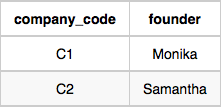
* Senior\_Manager: The senior\_manager\_code is the code of the senior manager, the lead\_manager\_code is the code of its lead manager, and the company\_code is the code of the

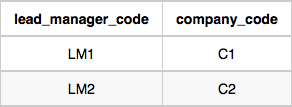
working company.

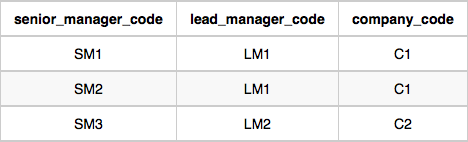
* Manager: The manager\_code is the code of the manager, the senior\_manager\_code is the code of its senior manager, the lead\_manager\_code is the code of its lead manager, and the company\_code is the code of the working company.
* Employee: The employee\_code is the code of the employee, the manager\_code is the code of its manager, the senior\_manager\_code is the code of its senior manager, the

lead\_manager\_code is the code of its lead manager, and the company\_code is the code of the

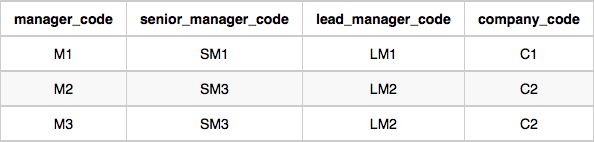
working company.

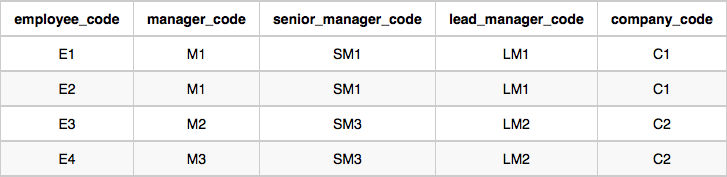
Sample Input

Company Table:

Lead\_Manager Table: Senior\_Manager Table:

Manager Table:



Employee Table:

Sample Output C1 Monika 1 2 1 2

C2 Samantha 1 1 2 2

Hint -

In company C1, the only lead manager is LM1. There are two senior managers, SM1 and SM2, under LM1. There is one manager, M1, under senior manager SM1. There are two employees, E1 and E2, under manager M1.

In company C2, the only lead manager is LM2. There is one senior manager, SM3, under LM2. There are two managers, M2 and M3, under senior manager SM3. There is one employee, E3, under manager M2, and another employee, E4, under manager, M3.

create table Company(company\_code varchar(10),founder varchar(10));

create table Lead\_Manager

(lead\_manager\_code varchar(10), company\_code varchar(10));

create table Senior\_Manager

(senior\_manager\_code varchar(10),lead\_manager\_code varchar(10),

 company\_code varchar(10));

create table Manager

(manager\_code varchar(10),senior\_manager\_code varchar(10),lead\_manager\_code varchar(10),

 company\_code varchar(10));

create table Employee

(employee\_code varchar(10),manager\_code varchar(10),senior\_manager\_code varchar(10),lead\_manager\_code varchar(10),

 company\_code varchar(10));

insert into Company values

('C1','Monika'),('C2','Samantha');

insert into Lead\_Manager values

('LM1','C1'),('LM2','C2');

insert into Senior\_Manager values

('SM1','LM1','C1'),('SM2','LM1','C1'),('SM3','LM2','C2');

insert into Manager values

('M1','SM1','LM1','C1'),('M2','SM3','LM2','C2'),('M3','SM3','LM2','C2');

insert into Employee values

('E1','M1','SM1','LM1','C1'),

('E2','M1','SM1','LM1','C1'),

('E3','M2','SM3','LM2','C2'),

('E4','M3','SM3','LM2','C2');

select

c.company\_code,c.founder,count(l.lead\_manager\_code) cnt

from

Company c left join  Lead\_Manager l

on c.company\_code = l.company\_code

group by c.company\_code,c.founder;

select

a.company\_code,a.founder,a.cnt,

count(s.senior\_manager\_code) from

(select

c.company\_code,c.founder,count(l.lead\_manager\_code) cnt

from

Company c left join  Lead\_Manager l

on c.company\_code = l.company\_code

group by c.company\_code,c.founder)a

left join Senior\_Manager s on

a.company\_code = s.company\_code

group by a.company\_code,a.founder,a.cnt;

select

b.company\_code,b.founder,b.cnt, b.cnt2,count(m.manager\_code) cnt3

from

(select

a.company\_code,a.founder,a.cnt,

count(s.senior\_manager\_code) cnt2 from

(select

c.company\_code,c.founder,count(l.lead\_manager\_code) cnt

from

Company c left join  Lead\_Manager l

on c.company\_code = l.company\_code

group by c.company\_code,c.founder)a

left join Senior\_Manager s on

a.company\_code = s.company\_code

group by a.company\_code,a.founder,a.cnt)b

left join Manager m on

b.company\_code = m.company\_code

group by b.company\_code,b.founder,b.cnt, b.cnt2;

-- final

select x.company\_code,x.founder,x.cnt, x.cnt2, x.cnt3,

count(employee\_code) cnt4

from

(select

b.company\_code,b.founder,b.cnt, b.cnt2,count(m.manager\_code) cnt3

from

(select

a.company\_code,a.founder,a.cnt,

count(s.senior\_manager\_code) cnt2 from

(select

c.company\_code,c.founder,count(l.lead\_manager\_code) cnt

from

Company c left join  Lead\_Manager l

on c.company\_code = l.company\_code

group by c.company\_code,c.founder)a

left join Senior\_Manager s on

a.company\_code = s.company\_code

group by a.company\_code,a.founder,a.cnt)b

left join Manager m on

b.company\_code = m.company\_code

group by b.company\_code,b.founder,b.cnt, b.cnt2)x

left join Employee e on

x.company\_code = e.company\_code

group by x.company\_code,x.founder,x.cnt, x.cnt2, x.cnt3;

# Q112.

Write a query to print all prime numbers less than or equal to 1000. Print your result on a single line, and use the ampersand () character as your separator (instead of a space).

For example, the output for all prime numbers <=10 would be: 2&3&5&7

Hint - Firstly, select L Prime\_Number from (select Level L from Dual connect Level ≤ 1000) and then do the same thing to create Level M, and then ﬁlter by M ≤ L and then group by L having count(case when L/M = truc(L/M) then ‘Y’ end) = 2 order by L

WITH RECURSIVE cte AS

(

    SELECT 1 as n

    UNION ALL

    SELECT n+1 FROM cte WHERE n < 1000

)

SELECT replace(group\_concat(n,'%'),',','')

FROM

(

  SELECT  x.n

  FROM cte x

  LEFT JOIN cte y ON y.n BETWEEN 2 AND x.n - 1

  WHERE x.n > 1

  GROUP BY x.n

  HAVING COUNT (CASE x.n % y.n WHEN 0 THEN 1 END) = 0

  ORDER BY x.n

) z

# Q113.

P(R) represents a pattern drawn by Julia in R rows. The following pattern represents P(5):

\*

\* \*

\* \* \*

\* \* \* \*

\* \* \* \* \*

Write a query to print the pattern P(20). Level - Easy

Source - Hackerrank

Hint - Use SYS\_CONNECT\_BY\_PATH(NULL, '\* ') FROM DUAL

WITH RECURSIVE cte AS

(

    SELECT 1 as n

    UNION ALL

    SELECT n+1 FROM cte WHERE n <20

)

select repeat('\*',n) from cte;

# Q114.

P(R) represents a pattern drawn by Julia in R rows. The following pattern represents P(5):

\* \* \* \* \*

\* \* \* \*

\* \* \*

\* \*

\*

Write a query to print the pattern P(20). Level - Easy

Hint - Use SYS\_CONNECT\_BY\_PATH(NULL, '\* ') FROM DUAL

WITH RECURSIVE cte AS

(

    SELECT 1 as n

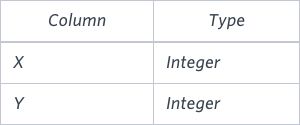
    UNION ALL

    SELECT n+1 FROM cte WHERE n <20

)

select repeat('\*',n) from cte order by n desc;

Q116. You are given a table, Functions, containing two columns: X and Y.

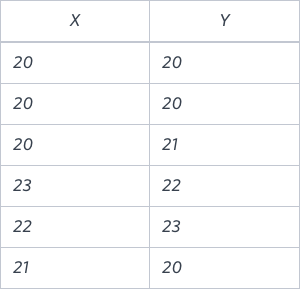


Two pairs (X1, Y1) and (X2, Y2) are said to be symmetric pairs if X1 = Y2 and X2 = Y1.

Write a query to output all such symmetric pairs in ascending order by the value of X. List the rows such that X1 ≤ Y1.

.

Sample Input



Sample Output 20 20

20 21

22 23

create table test(x int,y int);

insert into test VALUES

(20,20),(20,20),(20,21),(23,22),(22,23),(21,20);

select distinct t1.x,  t1.y from test t1 cross join test t2

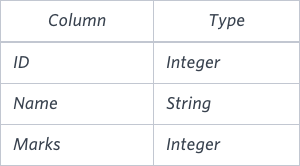
on t1.x = t2.x and t1.y = t2.y

and t1.x <= t1.y;

# Q115.

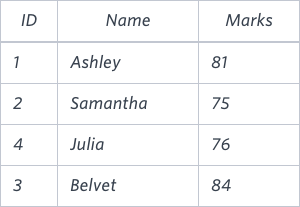
Query the Name of any student in STUDENTS who scored higher than 75 Marks. Order your output by the last three characters of each name. If two or more students both have names ending in the same last three characters (i.e.: Bobby, Robby, etc.), secondary sort them by ascending ID.

Level - Easy Hint - Use Like Input Format



The STUDENTS table is described as follows:

The Name column only contains uppercase (A-Z) and lowercase (a-z) letters. Sample Input



Sample Output Ashley

Julia Belvet

Explanation

Only Ashley, Julia, and Belvet have Marks > 75 . If you look at the last three characters of each of their names, there are no duplicates and 'ley' < 'lia' < 'vet'.

create table students(id int,name varchar(20),marks int);

insert into students values

(1,'Ashley',81),

(2,'Samantha',75),

(4,'Julia',76),

(3,'Belvet',84);

select name from students where marks > 75

order by right(name,3);

# Q116.

Write a query that prints a list of employee names (i.e.: the name attribute) from the Employee table in alphabetical order.

Level - Easy

Hint - Use ORDER BY Input Format

The Employee table containing employee data for a company is described as follows:



where employee\_id is an employee's ID number, name is their name, months is the total number of months they've been working for the company, and salary is their monthly salary.

Sample Input



Sample Output Angela

Bonnie Frank Joe Kimberly Lisa Michael Patrick Rose Todd

create table Employee

(empployee\_id int,name varchar(20),months int,salary bigint);

insert into Employee values

(12228,'Rose'   ,15,1968),

(33645,'Angela' ,1,3443),

(45692,'Frank'  ,17,1608),

(56118,'Patrick',7,1345),

(59725,'Lisa'   ,11,2330),

(74197,'Kimberly',16,4372),

(78454,'Bonnie' ,8,1771),

(83565,'Michael',6,2017),

(98607,'Todd',5,3396),

(99989,'Joe',9,3573);

select name from Employee  order by name;

# Q117.

Write a query that prints a list of employee names (i.e.: the name attribute) for employees in Employee having a salary greater than $2000 per month who have been employees for less than 10 months. Sort your result by ascending employee\_id.

Level - Easy

Hint - Use Ascending Input Format

The Employee table containing employee data for a company is described as follows:

where employee\_id is an employee's ID number, name is their name, months is the total number of months they've been working for the company, and salary is the their monthly salary.

Sample Input



Sample Output Angela Michael

Todd Joe

Explanation

Angela has been an employee for 1 month and earns $3443 per month. Michael has been an employee for 6 months and earns $2017 per month.

Todd has been an employee for 5 months and earns $3396 per month. Joe has been an employee for 9 months and earns $3573 per month. We order our output by ascending employee\_id.

create table Employee

(employee\_id int,

name varchar(20),

months int,

salary int);

insert into Employee values

(12228,'Rose',15,1968),

(33645,'Angela',1,3443),

(45692,'Frank',17,1608),

(56118,'Patrick',7,1345),

(59725,'Lisa',11,2330),

(74197,'Kimberly',16,4372),

(78454,'Bonnie',8,1771),

(83565,'Michael',6,2017),

(98607,'Todd',5,3396),

(99989,'Joe',9,3573);

select name from Employee where salary > 2000 and months < 10 order by name;

# Q118.

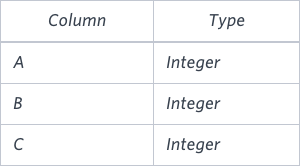
Write a query identifying the type of each record in the TRIANGLES table using its three side lengths. Output one of the following statements for each record in the table:

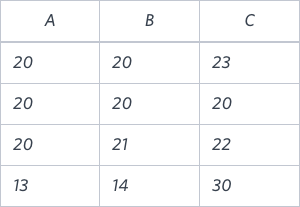
* Equilateral: It's a triangle with sides of equal length.
* Isosceles: It's a triangle with sides of equal length.
* Scalene: It's a triangle with sides of differing lengths.
* Not A Triangle: The given values of A, B, and C don't form a triangle. Level - Easy

Hint - Use predeﬁned functions for calculation.

Input Format

The TRIANGLES table is described as follows:



Each row in the table denotes the lengths of each of a triangle's three sides. Sample Input

Sample Output Isosceles Equilateral Scalene

Not A Triangle Explanation

Values in the tuple(20,20,23) form an Isosceles triangle, because A ≡ B.

Values in the tuple(20,20,20) form an Equilateral triangle, because A ≡ B ≡ C . Values in the tuple(20,21,22) form a Scalene triangle, because A ≠ B ≠C .

Values in the tuple (13,14,30) cannot form a triangle because the combined value of sides A and B is not larger than that of side C .

create table Triangle(A int,B int,C int);

insert  into Triangle VALUES

(20,20,23),

(20,20,20),

(20,21,22),

(13,14,30);

with cte as

(select A,B,C,

case

when A=B and B=C  then 'Equilateral'

when A=B or B=C or A=C then ' Isosceles'

when A+b <= C or A+C <= B or B+C <= A then 'Not A Triangle'

when A<>B and B<>C and C<>A then 'Scalene'

end triangle

from Triangle)

select triangle from cte;

# Q119.

Assume you are given the table below containing information on user transactions for particular products. Write a query to obtain the year-on-year growth rate for the total spend of each product for each year.

Output the year (in ascending order) partitioned by product id, current year's spend, previous year's spend and year-on-year growth rate (percentage rounded to 2 decimal places).

Level - Hard

Hint - Use extract function

**user\_transactions Table:**

|  |  |
| --- | --- |
| Column Name | Type |
| transaction\_id | integer |
| product\_id | integer |
| spend | decimal |
| transaction\_date | datetime |

**user\_transactions Example Input:**

|  |  |  |  |
| --- | --- | --- | --- |
| transaction\_i  d | product\_i d | spend | transaction\_date |
| 1341 | 123424 | 1500.60 | 12/31/2019 12:00:00 |
| 1423 | 123424 | 1000.20 | 12/31/2020 12:00:00 |
| 1623 | 123424 | 1246.44 | 12/31/2021 12:00:00 |
| 1322 | 123424 | 2145.32 | 12/31/2022 12:00:00 |

Example Output:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| y | product\_i d | curr\_year\_spend | prev\_year\_spend | yoy\_rate |
| 2 | 123424 | 1500.60 |  |  |
| 2 | 123424 | 1000.20 | 1500.60 | -33.35 |
| 2 | 123424 | 1246.44 | 1000.20 | 24.62 |
| 2 | 123424 | 2145.32 | 1246.44 | 72.12 |

create table user\_transactions

(transaction\_id int,

product\_id int,

spend float,

transaction\_date timestamp);

insert into user\_transactions values

(1341,123424,1500.60,'2019-12-31 12:00:00'),

(1423,123424,1000.20,'2020-12-31 12:00:00'),

(1623,123424,1246.44,'2021-12-31 12:00:00'),

(1322,123424,2145.32,'2022-12-31 12:00:00');

select product\_id,

spend curr\_year\_spend,

lag(spend,1) over(order by transaction\_date) prev\_year\_spend,

round((spend - lag(spend,1) over(order by transaction\_date))/

(lag(spend,1) over(order by transaction\_date))\*100,2) yoy\_rate

from user\_transactions;

# Q120.

Amazon wants to maximise the number of items it can stock in a 500,000 square feet warehouse. It wants to stock as many prime items as possible, and afterwards use the remaining square footage to stock the most number of non-prime items.

Write a SQL query to ﬁnd the number of prime and non-prime items that can be stored in the 500,000 square feet warehouse. Output the item type and number of items to be stocked.

Hint - create a table containing a summary of the necessary ﬁelds such as item type ('prime\_eligible', 'not\_prime'), SUM of square footage, and COUNT of items grouped by the item type.

**inventory table:**

|  |  |
| --- | --- |
| Column Name | Type |
| item\_id | integer |
| item\_type | string |
| item\_category | string |
| square\_footage | decimal |

**inventory Example Input:**

|  |  |  |  |
| --- | --- | --- | --- |
| item\_id | item\_type | item\_category | square\_footage |
| 1374 | prime\_eligible | mini refrigerator | 68.00 |
| 4245 | not\_prime | standing lamp | 26.40 |
| 2452 | prime\_eligible | television | 85.00 |
| 3255 | not\_prime | side table | 22.60 |
| 1672 | prime\_eligible | laptop | 8.50 |

**Example Output:**

|  |  |
| --- | --- |
| item\_type | item\_count |
| prime\_eligible | 9285 |
| not\_prime | 6 |

create table inventory

(item\_id integer,

item\_type varchar(20),

item\_category varchar(20),

square\_footage FLOAT);

insert into inventory values

(1374,'prime\_eligible','mini refrigerator', 68.00),

(4245,'not\_prime', 'standing lamp', 26.40),

(2452,'prime\_eligible', 'television', 85.00),

(3255,'not\_prime', 'side table', 22.60),

(1672,'prime\_eligible', 'laptop',8.50);

with cte as

(select item\_type,sum(square\_footage) sum\_square\_foot,

floor(500000/sum(square\_footage)) total\_space\_for\_itemtype,

COUNT(item\_category),floor(500000/sum(square\_footage))\*

COUNT(item\_category) item\_count from  inventory where item\_type = 'prime\_eligible'

group by item\_type),

cte2 as (select 500000-(select sum\_square\_foot\*total\_space\_for\_itemtype from cte) remaining\_square\_feet),

cte3 as

(select item\_type,sum(square\_footage) sum\_square\_foot,

floor((select remaining\_square\_feet from cte2)/sum(square\_footage)) total\_space\_for\_itemtype,

COUNT(item\_category),floor((select remaining\_square\_feet from cte2)/sum(square\_footage))\*

COUNT(item\_category) item\_count from  inventory where item\_type = 'not\_prime'

group by item\_type)

select item\_type,item\_count from cte

UNION

select item\_type,item\_count from cte3;

# Q121.

Assume you have the table below containing information on Facebook user actions. Write a query to obtain the active user retention in July 2022. Output the month (in numerical format 1, 2, 3) and the number of monthly active users (MAUs).

Hint: An active user is a user who has user action ("sign-in", "like", or "comment") in the current month and last month.

**Hint- Use generic correlated subquery user\_actions Table:**

|  |  |
| --- | --- |
| Column Name | Type |
| user\_id | integer |
| event\_id | integer |
| event\_type | string ("sign-in, "like", "comment") |
| event\_date | datetime |

**user\_actionsExample Input:**

|  |  |  |  |
| --- | --- | --- | --- |
| user\_id | event\_id | event\_type | event\_date |
| 445 | 7765 | sign-in | 05/31/2022 12:00:00 |
| 742 | 6458 | sign-in | 06/03/2022 12:00:00 |
| 445 | 3634 | like | 06/05/2022 12:00:00 |
| 742 | 1374 | comment | 06/05/2022 12:00:00 |
| 648 | 3124 | like | 06/18/2022 12:00:00 |

**Example Output for June 2022:**

|  |  |
| --- | --- |
| month | monthly\_active\_users |
| 6 | 1 |

create Table user\_actions

(user\_id int,

event\_id int,

event\_type enum ('sign-in', 'like', 'comment'),

event\_date timestamp);

insert into user\_actions values

(445,7765,'sign-in','2022-5-31 12:00:00'),

(742,6458,'sign-in', '2022-6-3 12:00:00'),

(445,3634,'like','2022-6-5 12:00:00'),

(742,1374,'comment','2022-6-5 12:00:00'),

(648,3124,'like','2022-6-18 12:00:00');

with cte as (

select user\_id,event\_type,event\_date,

row\_number() over(partition by user\_id order by user\_id) rw from user\_actions

where month(event\_date) in (6)

order by user\_id,event\_date)

select month(event\_date) month,count(user\_id) monthly\_active\_users

from cte where rw>1

group by month(event\_date);

# Q122.

Google's marketing team is making a Superbowl commercial and needs a simple statistic to put on their TV ad: the median number of searches a person made last year.

However, at Google scale, querying the 2 trillion searches is too costly. Luckily, you have access to the summary table which tells you the number of searches made last year and how many Google users fall into that bucket.

Write a query to report the median of searches made by a user. Round the median to one decimal point.

Hint- Write a subquery or common table expression (CTE) to generate a series of data (that's keyword for column) starting at the ﬁrst search and ending at some point with an optional incremental value.

**search\_frequency Table:**

|  |  |
| --- | --- |
| Column Name | Type |
| searches | integer |
| num\_users | integer |

**search\_frequency Example Input:**

|  |  |
| --- | --- |
| searches | num\_users |
| 1 | 2 |
| 2 | 2 |
| 3 | 3 |
| 4 | 1 |

**Example Output:**

2.5

median

create table search\_frequency(

searches int, num\_users int);

insert into search\_frequency values

(1,2),

(2,2),

(3,3),

(4,1);

with recursive generate\_numbers as

(

  select 1 as n

  union

  select n+1 from generate\_numbers where n<(select max(num\_users) from search\_frequency)

)

SELECT searches

FROM search\_frequency

    JOIN generate\_numbers

        ON search\_frequency.num\_users >= generate\_numbers.n

        order by searches;

with cte as(

with recursive generate\_numbers as

(

  select 1 as n

  union

  select n+1 from generate\_numbers where n<(select max(num\_users) from search\_frequency)

)

SELECT searches,row\_number() over(order by searches) rw

FROM search\_frequency

    JOIN generate\_numbers

        ON search\_frequency.num\_users >= generate\_numbers.n

        order by searches

)

select case

when if(count(\*)%2 = 0,'even','odd')='odd'

then (select searches from cte where rw = (select count(\*)/2 from cte))

when if(count(\*)%2 = 0,'even','odd')='even'

then (select round(avg(searches),1) from cte where rw in (select floor(avg(rw))  from cte union select  ceil(avg(rw)) from cte))

end as test from cte;

# Q123.

Write a query to update the Facebook advertiser's status using the daily\_pay table. Advertiser is a two-column table containing the user id and their payment status based on the last payment and daily\_pay table has current information about their payment. Only advertisers who paid will show up in this table.

Output the user id and current payment status sorted by the user id.

Hint- Query the daily\_pay table and check through the advertisers in this table. . **advertiser Table:**

|  |  |
| --- | --- |
| Column Name | Type |
| user\_id | string |
| status | string |

**advertiser Example Input:**

|  |  |
| --- | --- |
| user\_id | status |
| bing | NEW |
| yahoo | NEW |
| alibaba | EXISTING |

**daily\_pay Table:**

|  |  |
| --- | --- |
| Column Name | Type |
| user\_id | string |
| paid | decimal |

**daily\_pay Example Input:**

|  |  |
| --- | --- |
| user\_id | paid |
| yahoo | 45.00 |
| alibaba | 100.00 |
| target | 13.00 |

**Deﬁnition of advertiser status:**

* New: users registered and made their ﬁrst payment.
* Existing: users who paid previously and recently made a current payment.
* Churn: users who paid previously, but have yet to make any recent payment.
* Resurrect: users who did not pay recently but may have made a previous payment and have made payment again recently.

**Example Output:**

|  |  |
| --- | --- |
| user\_id | new\_status |
| bing | CHURN |
| yahoo | EXISTING |
| alibaba | EXISTING |

Bing's updated status is CHURN because no payment was made in the daily\_pay table whereas Yahoo which made a payment is updated as EXISTING.

The dataset you are querying against may have different input & output - this is just an example! Read this before proceeding to solve the question

For better understanding of the advertiser's status, we're sharing with you a table of possible transitions based on the payment status.

|  |  |  |  |
| --- | --- | --- | --- |
| # | Start | End | Condition |
| 1 | NEW | EXISTING | Paid on day T |
| 2 | NEW | CHURN | No pay on day T |
| 3 | EXISTING | EXISTING | Paid on day T |
| 4 | EXISTING | CHURN | No pay on day T |
| 5 | CHURN | RESURRECT | Paid on day T |
| 6 | CHURN | CHURN | No pay on day T |
| 7 | RESURRECT | EXISTING | Paid on day T |
| 8 | RESURRECT | CHURN | No pay on day T |

* 1. Row 2, 4, 6, 8: As long as the user has not paid on day T, the end status is updated to CHURN regardless of the previous status.
  2. Row 1, 3, 5, 7: When the user paid on day T, the end status is updated to either EXISTING or RESURRECT, depending on their previous state. RESURRECT is only possible when the previous state is CHURN. When the previous state is anything else, the status is updated to EXISTING.

create Table advertiser

(user\_id varchar(20),

status varchar(20));

insert into advertiser values

('bing','NEW'),

('yahoo','NEW'),

('alibaba','EXISTING');

create Table daily\_pay

(user\_id varchar(20),

paid float);

insert into daily\_pay values

('yahoo', 45.00),

('alibaba', 100.00),

('target', 13.00);

with cte as

(select a.user\_id,status,paid from advertiser a left join daily\_pay d on

a.user\_id = d.user\_id)

-- select \* from cte ;

select user\_id,

case when paid is NULL then 'CHURN'

when status != 'CHURN' and paid is not NULL then 'EXISTING'

    when status = 'CHURN' and paid is not NULL then 'RESURRECT'

    when status is NULL then 'NEW'

  end as new\_status

  from cte;

# Q124.

Amazon Web Services (AWS) is powered by ﬂeets of servers. Senior management has requested data-driven solutions to optimise server usage.

Write a query that calculates the total time that the ﬂeet of servers was running. The output should be in units of full days.

Level - Hard Hint-

1. Calculate individual uptimes
2. Sum those up to obtain the uptime of the whole ﬂeet, keeping in mind that the result must be output in units of full days

**Assumptions:**

* + Each server might start and stop several times.
  + The total time in which the server ﬂeet is running can be calculated as the sum of each server's uptime.

**server\_utilization Table:**

|  |  |
| --- | --- |
| Column Name | Type |
| server\_id | integer |
| status\_time | timestamp |
| session\_status | string |

**server\_utilization Example Input:**

|  |  |  |
| --- | --- | --- |
| server\_id | status\_time | session\_status |
| 1 | 08/02/2022 10:00:00 | start |
| 1 | 08/04/2022 10:00:00 | stop |
| 2 | 08/17/2022 10:00:00 | start |
| 2 | 08/24/2022 10:00:00 | stop |

**Example Output:**

21

total\_uptime\_days

create Table server\_utilization

(server\_id  int,

status\_time timestamp,

session\_status varchar(10));

insert into server\_utilization values

(1,'2022-08-02 10:00:00','start'),

(1,'2022-08-04 10:00:00','stop'),

(2,'2022-08-17 10:00:00','start'),

(2,'2022-08-24 10:00:00','stop');

with cte as(

select server\_id,status\_time start\_time,lead(status\_time,1)

over(partition by server\_id ) end\_time

from server\_utilization)

select sum(datediff(end\_time,start\_time)+1) total\_uptime\_days from cte;

# Q125.

Sometimes, payment transactions are repeated by accident; it could be due to user error, API failure or a retry error that causes a credit card to be charged twice.

Using the transactions table, identify any payments made at the same merchant with the same credit card for the same amount within 10 minutes of each other. Count such repeated payments.

Level - Hard

Hint- Use Partition and order by

**Assumptions:**

* + The ﬁrst transaction of such payments should not be counted as a repeated payment. This means, if there are two transactions performed by a merchant with the same credit card and for the same amount within 10 minutes, there will only be 1 repeated payment.

**transactions Table:**

|  |  |
| --- | --- |
| Column Name | Type |
| transaction\_id | integer |
| merchant\_id | integer |
| credit\_card\_id | integer |
| amount | integer |
| transaction\_timestamp | datetime |

**transactions Example Input:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| transaction\_id | merchant\_id | credit\_card\_id | amount | transaction\_timestamp |
| 1 | 101 | 1 | 100 | 09/25/2022 12:00:00 |
| 2 | 101 | 1 | 100 | 09/25/2022 12:08:00 |
| 3 | 101 | 1 | 100 | 09/25/2022 12:28:00 |
| 4 | 102 | 2 | 300 | 09/25/2022 12:00:00 |
| 6 | 102 | 2 | 400 | 09/25/2022 14:00:00 |

**Example Output:**

1

payment\_count

create Table transactions

(transaction\_id int,

merchant\_id int,

credit\_card\_id int,

amount int,

transaction\_timestamp timestamp);

insert into transactions values

(1,101,1,100,'2022-09-25 12:00:00'),

(2,101,1,100,'2022-09-25 12:08:00'),

(3,101,1,100,'2022-09-25 12:28:00'),

(4,102,2,300,'2022-09-25 12:00:00'),

(6,102,2,400,'2022-09-25 14:00:00');

with cte as

(select transaction\_id,merchant\_id,credit\_card\_id,amount,

transaction\_timestamp,

 count(merchant\_id) over(partition by credit\_card\_id order by transaction\_timestamp range between current row and interval '10'

  minute following)cnt

 from transactions)

 select count(credit\_card\_id) payment\_count from cte  where cnt = 2 group by credit\_card\_id;

# Q126.

DoorDash's Growth Team is trying to make sure new users (those who are making orders in their ﬁrst 14 days) have a great experience on all their orders in their 2 weeks on the platform.

Unfortunately, many deliveries are being messed up because:

* + the orders are being completed incorrectly (missing items, wrong order, etc.)
  + the orders aren't being received (wrong address, wrong drop off spot)
  + the orders are being delivered late (the actual delivery time is 30 minutes later than when the order was placed). Note that the estimated\_delivery\_timestamp is automatically set to 30 minutes after the order\_timestamp.

Hint- Use Where Clause and joins

Write a query to ﬁnd the bad experience rate in the ﬁrst 14 days for new users who signed up in June 2022. Output the percentage of bad experience rounded to 2 decimal places.

**orders Table:**

|  |  |
| --- | --- |
| Column Name | Type |
| order\_id | integer |
| customer\_id | integer |
| trip\_id | integer |
| status | string ('completed successfully', 'completed incorrectly', 'never received') |
| order\_timestamp | timestamp |

**orders Example Input:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| order\_id | customer\_id | trip\_id | status | order\_timestamp |
| 727424 | 8472 | 100463 | completed successfully | 06/05/2022 09:12:00 |
| 242513 | 2341 | 100482 | completed incorrectly | 06/05/2022 14:40:00 |
| 141367 | 1314 | 100362 | completed incorrectly | 06/07/2022 15:03:00 |
| 582193 | 5421 | 100657 | never\_received | 07/07/2022 15:22:00 |
| 253613 | 1314 | 100213 | completed successfully | 06/12/2022 13:43:00 |

**trips Table:**

|  |  |
| --- | --- |
| Column Name | Type |
| dasher\_id | integer |
| trip\_id | integer |
| estimated\_delivery\_timestamp | timestamp |
| actual\_delivery\_timestamp | timestamp |

trips Example Input:

|  |  |  |  |
| --- | --- | --- | --- |
| dasher\_id | trip\_id | estimated\_delivery\_timestamp | actual\_delivery\_timestamp |
| 101 | 100463 | 06/05/2022 09:42:00 | 06/05/2022 09:38:00 |
| 102 | 100482 | 06/05/2022 15:10:00 | 06/05/2022 15:46:00 |
| 101 | 100362 | 06/07/2022 15:33:00 | 06/07/2022 16:45:00 |
| 102 | 100657 | 07/07/2022 15:52:00 | - |
| 103 | 100213 | 06/12/2022 14:13:00 | 06/12/2022 14:10:00 |

customers Table:

|  |  |
| --- | --- |
| Column Name | Type |
| customer\_id | integer |
| signup\_timestamp | timestamp |

customers Example Input:

|  |  |
| --- | --- |
| customer\_id | signup\_timestamp |
| 8472 | 05/30/2022 00:00:00 |
| 2341 | 06/01/2022 00:00:00 |
| 1314 | 06/03/2022 00:00:00 |
| 1435 | 06/05/2022 00:00:00 |
| 5421 | 06/07/2022 00:00:00 |

Example Output:

75.00

bad\_experience\_pct

create Table orders

(order\_id int,

customer\_id int,

trip\_id int,

status enum('completed successfully', 'completed incorrectly',

'never\_received'),

order\_timestamp timestamp);

insert into orders values

(727424,8472,100463,'completed successfully','2022-06-05 09:12:00'),

(242513,2341,100482,'completed incorrectly','2022-06-05 14:40:00'),

(141367,1314,100362,'completed incorrectly','2022-06-07 15:03:00'),

(582193,5421,100657,'never\_received','2022-07-07 15:22:00'),

(253613,1314,100213,'completed successfully','2022-06-12 13:43:00');

create Table trips

(dasher\_id int,

trip\_id int,

estimated\_delivery\_timestamp timestamp,

actual\_delivery\_timestamp timestamp);

insert into trips values

(101,100463,'2022-06-05 09:42:00','2022-06-05 09:38:00'),

(102,100482,'2022-06-05 15:10:00','2022-06-05 15:46:00'),

(101,100362,'2022-06-07 15:33:00','2022-06-07 16:45:00'),

(102,100657,'2022-07-07 15:52:00',NULL),

(103,100213,'2022-06-12 14:13:00','2022-06-12 14:10:00');

create  Table customers

(customer\_id int,

signup\_timestamp timestamp);

insert into customers values

(8472,'2022-05-06 00:00:00'),

(2341,'2022-06-01 00:00:00'),

(1314,'2022-06-03 00:00:00'),

(1435,'2022-06-05 00:00:00'),

(5421,'2022-06-07 00:00:00');

with cte as (

 select o.order\_id,o.trip\_id,o.status from customers c

  inner join orders o on c.customer\_id = o.customer\_id

where extract(year\_month from c.signup\_timestamp) = '202206'

and adddate(c.signup\_timestamp, INTERVAL 14 DAY) > o.order\_timestamp

)

SELECT ROUND(

  100.0 \*

    COUNT(c.order\_id)/

    (SELECT COUNT(order\_id) FROM cte),2) AS bad\_experience\_pct

FROM cte c

INNER JOIN trips t

  ON c.trip\_id = t.trip\_id

WHERE c.status IN ('completed incorrectly', 'never\_received')

  OR t.actual\_delivery\_timestamp IS NULL

  AND t.estimated\_delivery\_timestamp < t.actual\_delivery\_timestamp;

# Q127.

Table: Scores

|  |  |
| --- | --- |
| Column Name | Type |
| player\_name | varchar |
| gender | varchar |
| day | date |
| score\_points | int |

(gender, day) is the primary key for this table.

A competition is held between the female team and the male team.

Each row of this table indicates that a player\_name and with gender has scored score\_point in someday.

Gender is 'F' if the player is in the female team and 'M' if the player is in the male team.

Write an SQL query to ﬁnd the total score for each gender on each day. Return the result table ordered by gender and day in ascending order. The query result format is in the following example.

Input: Scores table:

|  |  |  |  |
| --- | --- | --- | --- |
| player\_name | gender | day | score\_points |
| Aron | F | 2020-01-01 | 17 |
| Alice | F | 2020-01-07 | 23 |
| Bajrang | M | 2020-01-07 | 7 |
| Khali | M | 2019-12-25 | 11 |
| Slaman | M | 2019-12-30 | 13 |
| Joe | M | 2019-12-31 | 3 |
| Jose | M | 2019-12-18 | 2 |
| Priya | F | 2019-12-31 | 23 |
| Priyanka | F | 2019-12-30 | 17 |

Output:

|  |  |  |
| --- | --- | --- |
| gender | day | total |
| F | 2019-12-30 | 17 |
| F | 2019-12-31 | 40 |
| F | 2020-01-01 | 57 |
| F | 2020-01-07 | 80 |
| M | 2019-12-18 | 2 |
| M | 2019-12-25 | 13 |
| M | 2019-12-30 | 26 |
| M | 2019-12-31 | 29 |
| M | 2020-01-07 | 36 |

Explanation:

For the female team:

The ﬁrst day is 2019-12-30, Priyanka scored 17 points and the total score for the team is 17. The second day is 2019-12-31, Priya scored 23 points and the total score for the team is 40. The third day is 2020-01-01, Aron scored 17 points and the total score for the team is 57.

The fourth day is 2020-01-07, Alice scored 23 points and the total score for the team is 80.

For the male team:

The ﬁrst day is 2019-12-18, Jose scored 2 points and the total score for the team is 2.

The second day is 2019-12-25, Khali scored 11 points and the total score for the team is 13. The third day is 2019-12-30, Slaman scored 13 points and the total score for the team is 26. The fourth day is 2019-12-31, Joe scored 3 points and the total score for the team is 29.

The ﬁfth day is 2020-01-07, Bajrang scored 7 points and the total score for the team is 36.

create table Scores

(player\_name varchar(20),gender varchar(2),day date,score\_points int);

insert into Scores  values

('Aron','F','20-01-01',17),

('Alice','F','20-01-07',23),

('Bajrang','M','20-01-07',7),

('Khali','M','19-12-25',11),

('Slaman','M','19-12-30',13),

('Joe','M','19-12-31',3),

('Jose','M','19-12-18',2),

('Priya','F','19-12-31',23),

('Priyanka','F','19-12-30',17);

select gender,day,score\_points,

sum(score\_points)

over(partition by gender order by day

rows between unbounded preceding and current row) total

from Scores

order by gender,day;

# Q128.

**Table Person:**

|  |  |
| --- | --- |
| Column Name | Type |
| id | int |
| name | varchar |
| phone\_number | varchar |

id is the primary key for this table.

Each row of this table contains the name of a person and their phone number.

Phone number will be in the form 'xxx-yyyyyyy' where xxx is the country code (3 characters) and yyyyyyy is the phone number (7 characters) where x and y are digits. Both can contain leading zeros.

**Table Country:**

|  |  |
| --- | --- |
| Column Name | Type |
| name | varchar |
| country\_code | varchar |

country\_code is the primary key for this table.

Each row of this table contains the country name and its code. country\_code will be in the form 'xxx' where x is digits.

**Table Calls:**

|  |  |
| --- | --- |
| Column Name | Type |
| caller\_id | int |
| callee\_id | int |
| duration | int |

There is no primary key for this table, it may contain duplicates.

Each row of this table contains the caller id, callee id and the duration of the call in minutes. caller\_id

!= callee\_id

A telecommunications company wants to invest in new countries. The company intends to invest in the countries where the average call duration of the calls in this country is strictly greater than the global average call duration.

Write an SQL query to ﬁnd the countries where this company can invest. Return the result table in any order.

The query result format is in the following example.

Input: Person table:

|  |  |  |
| --- | --- | --- |
| id | name | phone\_number |
| 3 | Jonathan | 051-1234567 |
| 12 | Elvis | 051-7654321 |
| 1 | Moncef | 212-1234567 |
| 2 | Maroua | 212-6523651 |
| 7 | Meir | 972-1234567 |
| 9 | Rachel | 972-0011100 |

Country table:

|  |  |
| --- | --- |
| name | country\_code |
| Peru | 51 |
| Israel | 972 |
| Morocco | 212 |
| Germany | 49 |
| Ethiopia | 251 |
| Ethiopia | 251 |

Calls table:

|  |  |  |
| --- | --- | --- |
| caller\_id | callee\_id | duration |
| 1 | 9 | 33 |
| 2 | 9 | 4 |
| 1 | 2 | 59 |
| 3 | 12 | 102 |
| 3 | 12 | 330 |
| 12 | 3 | 5 |
| 7 | 9 | 13 |
| 7 | 1 | 3 |
| 9 | 7 | 1 |
| 1 | 7 | 7 |

Output:

Peru

country

Explanation:

The average call duration for Peru is (102 + 102 + 330 + 330 + 5 + 5) / 6 = 145.666667

The average call duration for Israel is (33 + 4 + 13 + 13 + 3 + 1 + 1 + 7) / 8 = 9.37500

The average call duration for Morocco is (33 + 4 + 59 + 59 + 3 + 7) / 6 = 27.5000

Global call duration average = (2 \* (33 + 4 + 59 + 102 + 330 + 5 + 13 + 3 + 1 + 7)) / 20 = 55.70000 Since Peru is the only country where the average call duration is greater than the global average, it is the only recommended country.

create table Person(

id int,name varchar(20),phone\_number varchar(20));

insert into Person values

(3 ,'Jonathan','051-1234567'),

(12,' Elvis','051-7654321'),

(1 ,'Moncef','212-1234567'),

(2 ,'Maroua','212-6523651'),

(7 ,'Meir','972-1234567'),

(9 ,'Rachel','972-0011100');

create table Country(

name varchar(20),country\_code int);

insert into Country values

('Peru', 51),

('Israel', 972),

('Morocco', 212),

('Germany', 49),

('Ethiopia', 251);

create table Calls

(caller\_id int,

callee\_id int,duration int);

insert into Calls values

(1,9,33),

(2,9,4),

(1,2,59),

(3,12,102),

(3,12,330),

(12,3,5),

(7,9,13),

(7,1,3),

(9,7,1),

(1,7,7);

select \* from Person;

select name country from Country c join

(select left(phone\_number,3) code from Person p

 join

(select distinct caller\_id as id,call\_duration from(

select caller\_id,callee\_id,sum(duration) over(partition by caller\_id  order by caller\_id)

call\_duration from Calls)a order by call\_duration desc limit 1)a on

p.id = a.id)tmp where c.country\_code = tmp.code;

# Q129.

Table: Numbers

|  |  |
| --- | --- |
| Column Name | Type |
| num | int |
| frequency | int |

num is the primary key for this table.

Each row of this table shows the frequency of a number in the database.

The median is the value separating the higher half from the lower half of a data sample.

Write an SQL query to report the median of all the numbers in the database after decompressing the Numbers table. Round the median to one decimal point.

The query result format is in the following example.

Input: Numbers table:

|  |  |
| --- | --- |
| num | frequency |
| 0 | 7 |
| 1 | 1 |
| 2 | 3 |
| 3 | 1 |

Output:

0

median

Explanation:

If we decompose the Numbers table, we will get [0, 0, 0, 0, 0, 0, 0, 1, 2, 2, 2, 3], so the median is (0 + 0) /

2 = 0.

create table Numbers(

num int, frequency int);

insert into Numbers values

(0,7),

(1,1),

(2,3),

(3,1);

select max(frequency) from Numbers;

with recursive generate\_numbers as

(

  select 1 as n

  union

  select n+1 from generate\_numbers where n<(select max(frequency) from Numbers)

) select \* from generate\_numbers;

with cte as(

with recursive generate\_numbers as

(

  select 1 as n

  union

  select n+1 from generate\_numbers where n<(select max(frequency) from Numbers)

)

SELECT num,row\_number() over(order by num) rw

FROM Numbers

    JOIN generate\_numbers

        ON Numbers.frequency >= generate\_numbers.n

        order by num,rw

)

select case

when if(count(\*)%2 = 0,'even','odd')='odd'

then (select num from cte where rw = (select count(\*)/2 from cte))

when if(count(\*)%2 = 0,'even','odd')='even'

then (select round(avg(num),1) from cte where rw in (select floor(avg(rw))  from cte union select  ceil(avg(rw)) from cte))

end as median from cte;

# Q130.

Table: Salary

|  |  |
| --- | --- |
| Column Name | Type |
| id | int |
| employee\_id | int |
| amount | int |
| pay\_date | date |

id is the primary key column for this table.

Each row of this table indicates the salary of an employee in one month. employee\_id is a foreign key from the Employee table.

Table: Employee

|  |  |
| --- | --- |
| Column Name | Type |
| employee\_id | int |
| department\_id | int |

employee\_id is the primary key column for this table.

Each row of this table indicates the department of an employee.

Write an SQL query to report the comparison result (higher/lower/same) of the average salary of employees in a department to the company's average salary.

Return the result table in any order.

The query result format is in the following example.

Input:

Salary table:

|  |  |  |  |
| --- | --- | --- | --- |
| id | employee\_id | amount | pay\_date |
| 1 | 1 | 9000 | 2017/03/31 |
| 2 | 2 | 6000 | 2017/03/31 |
| 3 | 3 | 10000 | 2017/03/31 |
| 4 | 1 | 7000 | 2017/02/28 |
| 5 | 2 | 6000 | 2017/02/28 |
| 6 | 3 | 8000 | 2017/02/28 |

Employee table:

|  |  |
| --- | --- |
| employee\_id | department\_id |
| 1 | 1 |
| 2 | 2 |
| 3 | 2 |

Output:

|  |  |  |
| --- | --- | --- |
| pay\_month | department\_id | comparison |
| 2017-02 | 1 | same |
| 2017-03 | 1 | higher |
| 2017-02 | 2 | same |
| 2017-03 | 2 | lower |

Explanation:

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

The average salary for department '1' is 9000, which is the salary of employee\_id '1' since there is only one employee in this department. So the comparison result is 'higher' since 9000 > 8333.33 obviously. The average salary of department '2' is (6000 + 10000)/2 = 8000, which is the average of employee\_id '2' and '3'. So the comparison result is 'lower' since 8000 < 8333.33.

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

create table Salary

(id int,employee\_id int,amount int,pay\_date date);

insert into Salary values

(1,1,9000 ,'2017-03-31'),

(2,2,6000 ,'2017-03-31'),

(3,3,10000,'2017-03-31'),

(4,1,7000 ,'2017-02-28'),

(5,2,6000 ,'2017-02-28'),

(6,3,8000 ,'2017-02-28');

create table Employee

(employee\_id int,department\_id int);

insert into Employee values

(1,1),

(2,2),

(3,2);

with cte as (

select e.employee\_id,department\_id,pay\_date,avg(amount)

over (partition by pay\_date ) average,

avg(amount)

over (partition by department\_id,pay\_date )average2 from Salary s

join Employee e on s.employee\_id = e.employee\_id

order by pay\_date)

select pay\_date,department\_id,case

when average = average2 then 'same'

when average2 > average then 'higher'

when average2 < average then 'lower'

end as comparison

from cte

group by pay\_date,department\_id,comparison

order by department\_id;

# Q131.

Table: Activity

|  |  |
| --- | --- |
| Column Name | Type |
| player\_id | int |
| device\_id | int |
| event\_date | date |
| games\_played | int |

(player\_id, event\_date) is the primary key of this table. This table shows the activity of players of some 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.

The install date of a player is the ﬁrst login day of that player.

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

Write an SQL query to report for each install date, the number of players that installed the game on that day, and the day one retention.

Return the result table in any order.

The query result format is in the following example.

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 | 2016-03-01 | 0 |
| 3 | 4 | 2016-07-03 | 5 |

Output:

|  |  |  |
| --- | --- | --- |
| install\_dt | installs | Day1\_retention |
| 2016-03-01 | 2 | 0.5 |
| 2017-06-25 | 1 | 0 |

Explanation:

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

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

create table Activity

(player\_id int,device\_id int,event\_date date,games\_played int);

insert into Activity values

(1,2,'2016-03-01',5),

(1,2,'2016-03-02',6),

(2,3,'2017-06-25',1),

(3,1,'2016-03-01',0),

(3,4,'2016-07-03',5);

with cte as

(select player\_id,count(player\_id)

over (partition by event\_date order by event\_date ) ct,min(event\_date)

over (partition by player\_id order by player\_id ) dt,event\_date

,

lead(event\_date,1) over(

    partition by player\_id order by player\_id)

    next\_date from Activity

),

cte2 as(select

distinct dt install\_dt, max(ct) installs

from cte group by dt),

cte3 as (select count(\*) cnt,event\_date from cte where

next\_date - event\_date =1 group by event\_date)

select install\_dt,installs,ifnull(round(cnt/installs,1),0)

Day1\_retention from cte2 left join cte3 on cte2.install\_dt = cte3.event\_date;

# Q132.

Table: Players

|  |  |
| --- | --- |
| Column Name | Type |
| player\_id | int |
| group\_id | int |

player\_id is the primary key of this table.

Each row of this table indicates the group of each player.

Table: Matches

|  |  |
| --- | --- |
| Column Name | Type |
| match\_id | int |
| ﬁrst\_player | int |
| second\_player | int |
| ﬁrst\_score | int |
| second\_score | int |

match\_id is the primary key of this table.

Each row is a record of a match, ﬁrst\_player and second\_player contain the player\_id of each match. ﬁrst\_score and second\_score contain the number of points of the ﬁrst\_player and second\_player respectively.

You may assume that, in each match, players belong to the same group.

The winner in each group is the player who scored the maximum total points within the group. In the case of a tie, the lowest player\_id wins.

Write an SQL query to ﬁnd the winner in each group. Return the result table in any order.

The query result format is in the following example.

Input: Players table:

|  |  |
| --- | --- |
| player\_id | group\_id |
| 15 | 1 |
| 25 | 1 |
| 30 | 1 |
| 45 | 1 |
| 10 | 2 |
| 35 | 2 |
| 50 | 2 |
| 20 | 3 |
| 40 | 3 |

Matches table:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| match\_id | ﬁrst\_player | second\_player | ﬁrst\_score | second\_score |
| 1 | 15 | 45 | 3 | 0 |
| 2 | 30 | 25 | 1 | 2 |
| 3 | 30 | 15 | 2 | 0 |
| 4 | 40 | 20 | 5 | 2 |
| 5 | 35 | 50 | 1 | 1 |

Output:

|  |  |
| --- | --- |
| group\_id | player\_id |
| 1 | 15 |
| 2 | 35 |
| 3 | 40 |

# Q133.

Table: Student

|  |  |
| --- | --- |
| Column Name | Type |
| student\_id | int |
| student\_name | varchar |

student\_id is the primary key for this table. student\_name is the name of the student.

Table: Exam

|  |  |
| --- | --- |
| Column Name | Type |
| exam\_id | int |
| student\_id | int |
| score | int |

(exam\_id, student\_id) is the primary key for this table.

Each row of this table indicates that the student with student\_id had a score points in the exam with id exam\_id.

A quiet student is the one who took at least one exam and did not score the high or the low score.

Write an SQL query to report the students (student\_id, student\_name) being quiet in all exams. Do not return the student who has never taken any exam.

Return the result table ordered by student\_id.

The query result format is in the following example.

Input:

Student table:

|  |  |
| --- | --- |
| student\_id | student\_name |
| 1 | Daniel |
| 2 | Jade |
| 3 | Stella |
| 4 | Jonathan |
| 5 | Will |

Exam table:

|  |  |  |
| --- | --- | --- |
| exam\_id | student\_id | score |
| 10 | 1 | 70 |
| 10 | 2 | 80 |
| 10 | 3 | 90 |
| 20 | 1 | 80 |
| 30 | 1 | 70 |
| 30 | 3 | 80 |
| 30 | 4 | 90 |
| 40 | 1 | 60 |
| 40 | 2 | 70 |
| 40 | 4 | 80 |

Output:

|  |  |
| --- | --- |
| student\_id | student\_name |
| 2 | Jade |

Explanation:

For exam 1: Student 1 and 3 hold the lowest and high scores respectively. For exam 2: Student 1 holds both the highest and lowest score.

For exam 3 and 4: Student 1 and 4 hold the lowest and high scores respectively. Students 2 and 5 have never got the highest or lowest in any of the exams.

Since student 5 is not taking any exam, he is excluded from the result. So, we only return the information of Student 2.

create  table Student

(student\_id int,student\_name varchar(10));

insert into Student values

(1,'Daniel'),

(2,'Jade'),

(3,'Stella'),

(4,'Jonathan'),

(5,'Will');

create table Exam

(exam\_id int,student\_id int,score int);

insert into Exam values

(10,1,70),

(10,2,80),

(10,3,90),

(20,1,80),

(30,1,70),

(30,3,80),

(30,4,90),

(40,1,60),

(40,2,70),

(40,4,80);

with cte as(

    select distinct ss,student\_name

from(

select s.student\_id ss,student\_name ,e.student\_id es,

exam\_id ,score,min(score) over(partition by

exam\_id) min\_score,max(score) over(partition by

exam\_id) max\_score

 from Student s join Exam e on s.student\_id = e.student\_id

)a

where  a.score <> a.min\_score and a.score <> a.max\_score),

-- select \* from cte;

cte2 as (select distinct b.ss

from(

select distinct s.student\_id ss,student\_name ,e.student\_id es,

exam\_id ,score,min(score) over(partition by

exam\_id) min\_score,max(score) over(partition by

exam\_id) max\_score

 from Student s join Exam e on s.student\_id = e.student\_id

)b

where  b.score =  b.min\_score or b.score = b.max\_score)

-- select \* from cte2;

select \* from cte where cte.ss not in (select \* from cte2);

# Q134.

Table: Student

|  |  |
| --- | --- |
| Column Name | Type |
| student\_id | int |
| student\_name | varchar |

student\_id is the primary key for this table. student\_name is the name of the student.

Table: Exam

|  |  |
| --- | --- |
| Column Name | Type |
| exam\_id | int |
| student\_id | int |
| score | int |

(exam\_id, student\_id) is the primary key for this table.

Each row of this table indicates that the student with student\_id had a score points in the exam with id exam\_id.

A quiet student is the one who took at least one exam and did not score the high or the low score. Write an SQL query to report the students (student\_id, student\_name) being quiet in all exams. Do not return the student who has never taken any exam.

Return the result table ordered by student\_id.

The query result format is in the following example.

Input: Student table:

|  |  |
| --- | --- |
| student\_id | student\_name |
| 1 | Daniel |
| 2 | Jade |
| 3 | Stella |
| 4 | Jonathan |
| 5 | Will |

Exam table:

|  |  |  |
| --- | --- | --- |
| exam\_id | student\_id | score |
| 10 | 1 | 70 |
| 10 | 2 | 80 |
| 10 | 3 | 90 |
| 20 | 1 | 80 |
| 30 | 1 | 70 |
| 30 | 3 | 80 |
| 30 | 4 | 90 |
| 40 | 1 | 60 |
| 40 | 2 | 70 |
| 40 | 4 | 80 |

Output:

|  |  |
| --- | --- |
| student\_id | student\_name |
| 2 | Jade |

Explanation:

For exam 1: Student 1 and 3 hold the lowest and high scores respectively. For exam 2: Student 1 holds both the highest and lowest score.

For exam 3 and 4: Student 1 and 4 hold the lowest and high scores respectively. Students 2 and 5 have never got the highest or lowest in any of the exams.

Since student 5 is not taking any exam, he is excluded from the result. So, we only return the information of Student 2.

create  table Student

(student\_id int,student\_name varchar(10));

insert into Student values

(1,'Daniel'),

(2,'Jade'),

(3,'Stella'),

(4,'Jonathan'),

(5,'Will');

create table Exam

(exam\_id int,student\_id int,score int);

insert into Exam values

(10,1,70),

(10,2,80),

(10,3,90),

(20,1,80),

(30,1,70),

(30,3,80),

(30,4,90),

(40,1,60),

(40,2,70),

(40,4,80);

with cte as(

    select distinct ss,student\_name

from(

select s.student\_id ss,student\_name ,e.student\_id es,

exam\_id ,score,min(score) over(partition by

exam\_id) min\_score,max(score) over(partition by

exam\_id) max\_score

 from Student s join Exam e on s.student\_id = e.student\_id

)a

where  a.score <> a.min\_score and a.score <> a.max\_score),

-- select \* from cte;

cte2 as (select distinct b.ss

from(

select distinct s.student\_id ss,student\_name ,e.student\_id es,

exam\_id ,score,min(score) over(partition by

exam\_id) min\_score,max(score) over(partition by

exam\_id) max\_score

 from Student s join Exam e on s.student\_id = e.student\_id

)b

where  b.score =  b.min\_score or b.score = b.max\_score)

-- select \* from cte2;

select \* from cte where cte.ss not in (select \* from cte2);

# Q135.

Table: UserActivity

|  |  |
| --- | --- |
| Column Name | Type |
| username | varchar |
| activity | varchar |
| startDate | Date |
| endDate | Date |

There is no primary key for this table. It may contain duplicates.

This table contains information about the activity performed by each user in a period of time. A person with a username performed an activity from startDate to endDate.

Write an SQL query to show the second most recent activity of each user.

If the user only has one activity, return that one. A user cannot perform more than one activity at the same time.

Return the result table in any order.

The query result format is in the following example.

Input: UserActivity table:

|  |  |  |  |
| --- | --- | --- | --- |
| username | activity | startDate | endDate |
| Alice | Travel | 2020-02-12 | 2020-02-20 |
| Alice | Dancing | 2020-02-21 | 2020-02-23 |
| Alice | Travel | 2020-02-24 | 2020-02-28 |
| Bob | Travel | 2020-02-11 | 2020-02-18 |

Output:

|  |  |  |  |
| --- | --- | --- | --- |
| username | activity | startDate | endDate |
| Alice | Dancing | 2020-02-21 | 2020-02-23 |
| Bob | Travel | 2020-02-11 | 2020-02-18 |

Explanation:

The most recent activity of Alice is Travel from 2020-02-24 to 2020-02-28, before that she was dancing from 2020-02-21 to 2020-02-23.

Bob only has one record, we just take that one.

create table UserActivity

(username varchar(20),activity varchar(20),startDate date,endDate date);

insert into UserActivity values

('Alice','Travel','2020-02-12','2020-02-20'),

('Alice','Dancing','2020-02-21','2020-02-23'),

('Alice','Travel','2020-02-24','2020-02-28'),

('Bob','Travel','2020-02-11','2020-02-18');

with cte as (

select username,activity,startDate,endDate,

row\_number() over(partition by username order by enddate desc) rw,

count(\*)over(partition by username)cnt from UserActivity

)

-- select \* from cte;

select username,activity,startDate,endDate from cte where (  rw = 2 and cnt <> 1 ) or (rw = 1 and cnt = 1);

# Q136.

Table: UserActivity

|  |  |
| --- | --- |
| Column Name | Type |
| username | varchar |
| activity | varchar |
| startDate | Date |
| endDate | Date |

There is no primary key for this table. It may contain duplicates.

This table contains information about the activity performed by each user in a period of time. A person with a username performed an activity from startDate to endDate.

Write an SQL query to show the second most recent activity of each user.

If the user only has one activity, return that one. A user cannot perform more than one activity at the same time.

Return the result table in any order.

The query result format is in the following example.

Input: UserActivity table:

|  |  |  |  |
| --- | --- | --- | --- |
| username | activity | startDate | endDate |
| Alice | Travel | 2020-02-12 | 2020-02-20 |
| Alice | Dancing | 2020-02-21 | 2020-02-23 |
| Alice | Travel | 2020-02-24 | 2020-02-28 |
| Bob | Travel | 2020-02-11 | 2020-02-18 |

Output:

|  |  |  |  |
| --- | --- | --- | --- |
| username | activity | startDate | endDate |
| Alice | Dancing | 2020-02-21 | 2020-02-23 |
| Bob | Travel | 2020-02-11 | 2020-02-18 |

Explanation:

The most recent activity of Alice is Travel from 2020-02-24 to 2020-02-28, before that she was dancing from 2020-02-21 to 2020-02-23.

Bob only has one record, we just take that one.

create table UserActivity

(username varchar(20),activity varchar(20),startDate date,endDate date);

insert into UserActivity values

('Alice','Travel','2020-02-12','2020-02-20'),

('Alice','Dancing','2020-02-21','2020-02-23'),

('Alice','Travel','2020-02-24','2020-02-28'),

('Bob','Travel','2020-02-11','2020-02-18');

with cte as (

select username,activity,startDate,endDate,

row\_number() over(partition by username order by enddate desc) rw,

count(\*)over(partition by username)cnt from UserActivity

)

-- select \* from cte;

select username,activity,startDate,endDate from cte where (  rw = 2 and cnt <> 1 ) or (rw = 1 and cnt = 1);

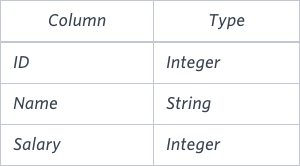
# Q137.

Samantha was tasked with calculating the average monthly salaries for all employees in the EMPLOYEES table, but did not realise her keyboard's 0 key was broken until after completing the calculation. She wants your help ﬁnding the difference between her miscalculation (using salaries with any zeros removed), and the actual average salary.

Write a query calculating the amount of error (i.e.: actual - miscalculated average monthly salaries), and round it up to the next integer.

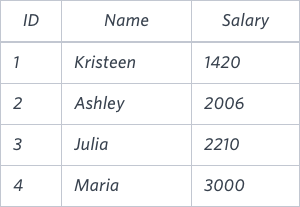
Input Format

The EMPLOYEES table is described as follows:



Note: Salary is per month. Constraints

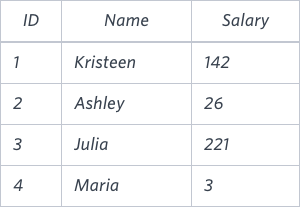
1000<salary < 10^5 Sample Input



Sample Output 2061

Explanation

The table below shows the salaries without zeros as they were entered by Samantha:



Samantha computes an average salary of 98.00 . The actual average salary is 2159.00.

The resulting error between the two calculations is 2159.00-98.00 = 2061.00. Since it is equal to the integer 2061, it does not get rounded up.

create table  EMPLOYEES

(id int,name varchar(20),salary int);

insert into  EMPLOYEES values

(1,'Kristeen',1420),

(2,'Ashley',2006),

(3,'Julia',2210),

(4,'Maria',3000);

SELECT (avg(salary)) - (avg(replace(salary, '0', '')))

AS avg\_salary\_diff FROM EMPLOYEES;

# Q138.

We deﬁne an employee's total earnings to be their monthly salary \* months worked, and the maximum total earnings to be the maximum total earnings for any employee in the Employee table. Write a query to ﬁnd the maximum total earnings for all employees as well as the total number of employees who have maximum total earnings. Then print these values as 2 space-separated integers.

Level - Easy

Hint - Use Aggregation functions Input Format

The Employee table containing employee data for a company is described as follows:

where employee\_id is an employee's ID number, name is their name, months is the total number of months they've been working for the company, and salary is the their monthly salary.

Sample Input



Sample Output 69952 1

Explanation:

The table and earnings data is depicted in the following diagram:



The maximum earnings value is 69952. The only employee with earnings= 69952 is Kimberly, so we print the maximum earnings value (69952) and a count of the number of employees who have earned

$69952 (which is 1) as two space-separated values.

create table Employee

(empployee\_id int,name varchar(20),months int,salary bigint);

insert into Employee values

(12228,'Rose'   ,15,1968),

(33645,'Angela' ,1,3443),

(45692,'Frank'  ,17,1608),

(56118,'Patrick',7,1345),

(59725,'Lisa'   ,11,2330),

(74197,'Kimberly',16,4372),

(78454,'Bonnie' ,8,1771),

(83565,'Michael',6,2017),

(98607,'Todd',5,3396),

(99989,'Joe',9,3573);

select earnings,count(earnings) count from (

select empployee\_id ,name,months ,salary,(months\*salary) earnings,

dense\_rank() over (order by months\*salary DESC) rnk from Employee)a

where a.rnk  =1 group by earnings;

# Q139.

Generate the following two result sets:

1. Query an alphabetically ordered list of all names in OCCUPATIONS, immediately followed by the ﬁrst letter of each profession as a parenthetical (i.e.: enclosed in parentheses). For example: AnActorName(A), ADoctorName(D), AProfessorName(P), and ASingerName(S).

Query the number of occurrences of each occupation in OCCUPATIONS. Sort the occurrences in ascending order, and output them in the following format:

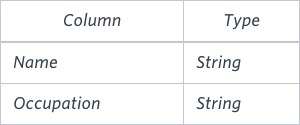
Level - Medium

There are a total of [occupation\_count] [occupation]s.

1. where [occupation\_count] is the number of occurrences of an occupation in OCCUPATIONS and [occupation] is the lowercase occupation name. If more than one Occupation has the same [occupation\_count], they should be ordered alphabetically.

Note: There will be at least two entries in the table for each type of occupation. Input Format

The OCCUPATIONS table is described as follows:



Occupation will only contain one of the following values: Doctor, Professor, Singer or Actor. Sample Input

An OCCUPATIONS table that contains the following records:



Sample Output Ashely(P) Christeen(P) Jane(A) Jenny(D) Julia(A) Ketty(P) Maria(A) Meera(S) Priya(S) Samantha(D)

There are a total of 2 doctors. There are a total of 2 singers. There are a total of 3 actors. There are a total of 3 professors.

Hint -

The results of the ﬁrst query are formatted to the problem description's speciﬁcations.

The results of the second query are ascendingly ordered ﬁrst by number of names corresponding to each profession (2<= 2<=3<=3), and then alphabetically by profession (doctor <= singer , and actor <= professor ).

create table Occupations

(Name varchar(10),Occupation varchar(10));

insert into Occupations values

('Samantha','Doctor'),

('Julia','Actor'),

('Maria','Actor'),

('Meera','Singer'),

('Ashley','Professor'),

('Ketty','Professor'),

('Christeen','Professor'),

('Jane','Actor'),

('Jenny','Doctor'),

('Priya','Singer');

select concat(name,' (',left(Occupation,1),')') from Occupations order by name;

select

concat('There are a total of ',

count(Name),' ', Occupation,'s.')

 from Occupations

 group by Occupation order by count(Name);

(select concat(name,' (',left(Occupation,1),')') output from Occupations order by Name)

union all

(select

concat('There are a total of ',

count(Name),' ', Occupation,'s.') output

 from Occupations

 group by Occupation order by count(Name));

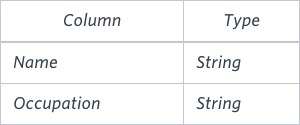
# Q140 .

Pivot the Occupation column in OCCUPATIONS so that each Name is sorted alphabetically and displayed underneath its corresponding Occupation. The output column headers should be Doctor, Professor, Singer, and Actor, respectively.

Note: Print NULL when there are no more names corresponding to an occupation.

Input Format

The OCCUPATIONS table is described as follows:



Occupation will only contain one of the following values: Doctor, Professor, Singer or Actor. Sample Input



Sample Output

Jenny Ashley Meera Jane Samantha Christeen Priya Julia NULL Ketty NULL Maria

Hint -

The ﬁrst column is an alphabetically ordered list of Doctor names.

The second column is an alphabetically ordered list of Professor names. The third column is an alphabetically ordered list of Singer names.

The fourth column is an alphabetically ordered list of Actor names.

The empty cell data for columns with less than the maximum number of names per occupation (in this case, the Professor and Actor columns) are ﬁlled with NULL values.

create table Occupations

(Name varchar(10),Occupation varchar(10));

insert into Occupations values

('Samantha','Doctor'),

('Julia','Actor'),

('Maria','Actor'),

('Meera','Singer'),

('Ashley','Professor'),

('Ketty','Professor'),

('Christeen','Professor'),

('Jane','Actor'),

('Jenny','Doctor'),

('Priya','Singer');

 with cte as (select row\_number() over(partition by Occupation order by Name ASC) as rnk,Occupation,Name

from Occupations),

cte2 as (select rnk,

max(case  when Occupation =  'Doctor' then Name end) as Doctor,

max(case  when Occupation = 'Professor' then Name end) as Professor,

max(case  when Occupation = 'Singer' then Name end) as Singer,

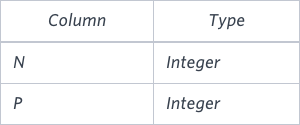
max(case  when Occupation = 'Actor' then Name end) as Actor

from cte group  by rnk)

select Doctor,Professor,Singer,Actor from cte2;

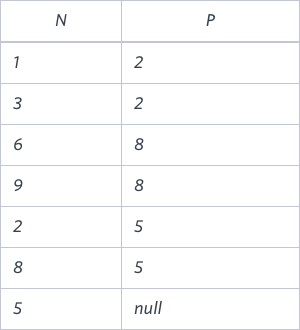
# Q141.

You are given a table, BST, containing two columns: N and P, where N represents the value of a node in Binary Tree, and P is the parent of N.



Write a query to ﬁnd the node type of Binary Tree ordered by the value of the node. Output one of the following for each node:

* + Root: If node is root node.
  + Leaf: If node is leaf node.
  + Inner: If node is neither root nor leaf node. Sample Input

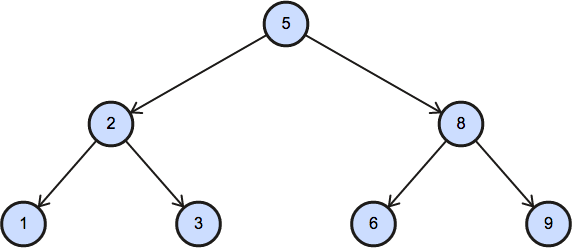


Sample Output 1 Leaf

1. Inner
2. Leaf
3. Root
4. Leaf
5. Inner
6. Leaf

Explanation

The Binary Tree below illustrates the sample:



create table BST(N int,P int);

insert  into BST VALUES

(1,2),(3,2),(6,8),(9,8),(2,5),(8,5),(5,NULL);

select concat(N,' ',

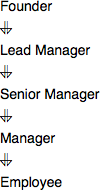
case when P is NULL then 'Root'

when N in (select P from BST) then 'Inner'

else 'Leaf'  end ) as output

from BST order by N;

# Q142 .

Amber's conglomerate corporation just acquired some new companies. Each of the companies

follows this hierarchy:

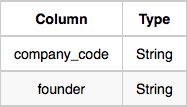
Given the table schemas below, write a query to print the company\_code, founder name, total number of lead managers, total number of senior managers, total number of managers, and total number of employees. Order your output by ascending company\_code.

Level - Medium Note:

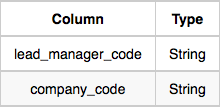
* The tables may contain duplicate records.
* The company\_code is string, so the sorting should not be numeric. For example, if the company\_codes are C\_1, C\_2, and C\_10, then the ascending company\_codes will be C\_1, C\_10, and C\_2.

Input Format

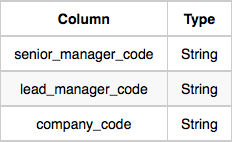
The following tables contain company data:

* Company: The company\_code is the code of the company and founder is the founder of the

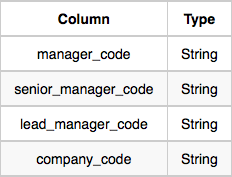
company.

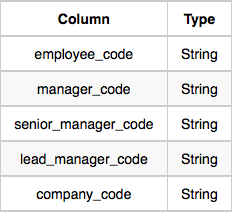
* Lead\_Manager: The lead\_manager\_code is the code of the lead manager, and the

company\_code is the code of the working company.

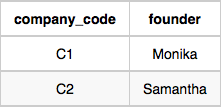
* Senior\_Manager: The senior\_manager\_code is the code of the senior manager, the lead\_manager\_code is the code of its lead manager, and the company\_code is the code of the

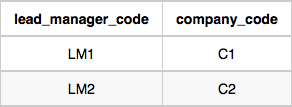
working company.

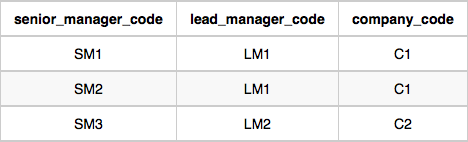
* Manager: The manager\_code is the code of the manager, the senior\_manager\_code is the code of its senior manager, the lead\_manager\_code is the code of its lead manager, and the company\_code is the code of the working company.
* Employee: The employee\_code is the code of the employee, the manager\_code is the code of its manager, the senior\_manager\_code is the code of its senior manager, the

lead\_manager\_code is the code of its lead manager, and the company\_code is the code of the

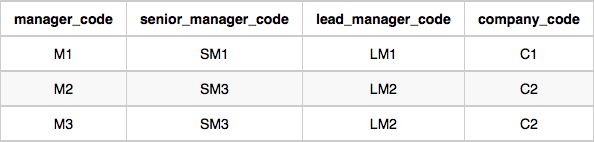
working company.

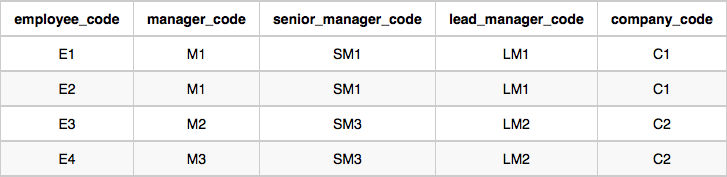
Sample Input

Company Table:

Lead\_Manager Table: Senior\_Manager Table:

Manager Table:



Employee Table:

Sample Output C1 Monika 1 2 1 2

C2 Samantha 1 1 2 2

Hint -

In company C1, the only lead manager is LM1. There are two senior managers, SM1 and SM2, under LM1. There is one manager, M1, under senior manager SM1. There are two employees, E1 and E2, under manager M1.

In company C2, the only lead manager is LM2. There is one senior manager, SM3, under LM2. There are two managers, M2 and M3, under senior manager SM3. There is one employee, E3, under manager M2, and another employee, E4, under manager, M3.

create table Company(company\_code varchar(10),founder varchar(10));

create table Lead\_Manager

(lead\_manager\_code varchar(10), company\_code varchar(10));

create table Senior\_Manager

(senior\_manager\_code varchar(10),lead\_manager\_code varchar(10),

 company\_code varchar(10));

create table Manager

(manager\_code varchar(10),senior\_manager\_code varchar(10),lead\_manager\_code varchar(10),

 company\_code varchar(10));

create table Employee

(employee\_code varchar(10),manager\_code varchar(10),senior\_manager\_code varchar(10),lead\_manager\_code varchar(10),

 company\_code varchar(10));

insert into Company values

('C1','Monika'),('C2','Samantha');

insert into Lead\_Manager values

('LM1','C1'),('LM2','C2');

insert into Senior\_Manager values

('SM1','LM1','C1'),('SM2','LM1','C1'),('SM3','LM2','C2');

insert into Manager values

('M1','SM1','LM1','C1'),('M2','SM3','LM2','C2'),('M3','SM3','LM2','C2');

insert into Employee values

('E1','M1','SM1','LM1','C1'),

('E2','M1','SM1','LM1','C1'),

('E3','M2','SM3','LM2','C2'),

('E4','M3','SM3','LM2','C2');

select

c.company\_code,c.founder,count(l.lead\_manager\_code) cnt

from

Company c left join  Lead\_Manager l

on c.company\_code = l.company\_code

group by c.company\_code,c.founder;

select

a.company\_code,a.founder,a.cnt,

count(s.senior\_manager\_code) from

(select

c.company\_code,c.founder,count(l.lead\_manager\_code) cnt

from

Company c left join  Lead\_Manager l

on c.company\_code = l.company\_code

group by c.company\_code,c.founder)a

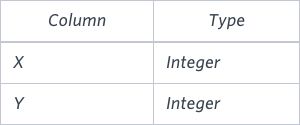
left join Senior\_Manager s on

a.company\_code = s.company\_code

group by a.company\_code,a.founder,a.cnt;

# Q143 .

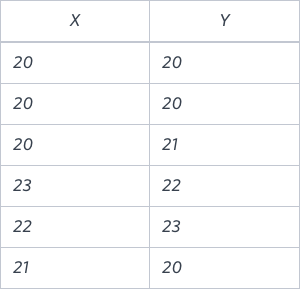
You are given a table, Functions, containing two columns: X and Y.



Two pairs (X1, Y1) and (X2, Y2) are said to be symmetric pairs if X1 = Y2 and X2 = Y1.

Write a query to output all such symmetric pairs in ascending order by the value of X. List the rows such that X1 ≤ Y1.

Level - Medium Source - Hackerrank

Hint - Use group by and having clause . Sample Input

Sample Output 20 20

20 21

22 23

create table test(x int,y int);

insert into test VALUES

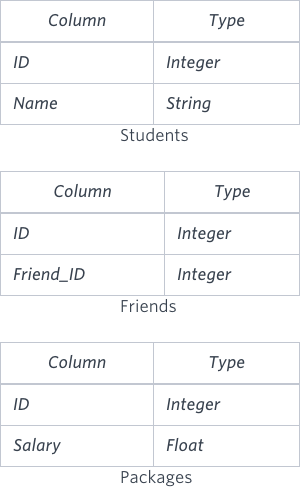
(20,20),(20,20),(20,21),(23,22),(22,23),(21,20);

select distinct t1.x,  t1.y from test t1 cross join test t2

on t1.x = t2.x and t1.y = t2.y

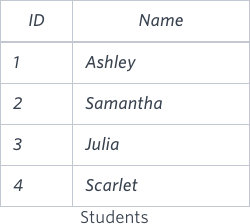
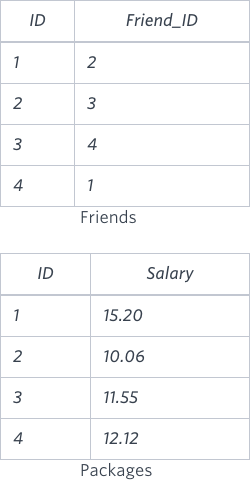
and t1.x <= t1.y;

# Q144 .

You are given three tables: Students, Friends and Packages. Students contains two columns: ID and Name. Friends contains two columns: ID and Friend\_ID (ID of the ONLY best friend). Packages contain two columns: ID and Salary (offered salary in $ thousands per month).

Write a query to output the names of those students whose best friends got offered a higher salary than them. Names must be ordered by the salary amount offered to the best friends. It is guaranteed that no two students get the same salary offer.

Sample Input

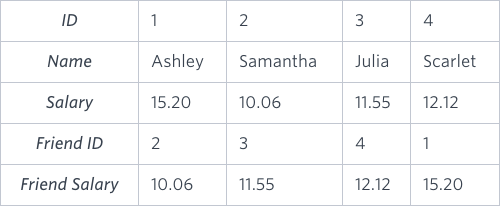
 

Sample Output Samantha Julia

Scarlet

Explanation

See the following table:



Now,

* Samantha's best friend got offered a higher salary than her at 11.55
* Julia's best friend got offered a higher salary than her at 12.12
* Scarlet's best friend got offered a higher salary than her at 15.2
* Ashley's best friend did NOT get offered a higher salary than her The name output, when ordered by the salary offered to their friends, will be:
* Samantha
* Julia
* Scarlet

create table Students

(id int,

name varchar(10));

insert into Students values

(1,'Ashley'),

(2,'Samantha'),

(3,'Julia'),

(4,'Scarlet');

create table Friends

(id int,

friend\_id int);

insert into Friends values

(1,2),

(2,3),

(3,4),

(4,1);

create table Packages

(id int,

salary int);

insert into Packages values

(1,15.20),

(2,10.06),

(3,11.55),

(4,12.12);

with cte as(

select a.id,a.name,a.friend\_id,b.name friend from (

select s.id,s.name,f.friend\_id from Students s

join Friends f

on s.id = f.id order by s.id)a

join

(select s.id,s.name,f.friend\_id from Students s

join Friends f

on s.id = f.id order by s.id)b

on a.friend\_id  = b.id order by a.id)

select cte.id,cte.name,p.salary,cte.friend\_id,

p2.salary from cte join Packages p

on cte.id = p.id left join Packages p2

on cte.friend\_id = p2.id;

with cte as(

select a.id,a.name,a.friend\_id,b.name friend from (

select s.id,s.name,f.friend\_id from Students s

join Friends f

on s.id = f.id order by s.id)a

join

(select s.id,s.name,f.friend\_id from Students s

join Friends f

on s.id = f.id order by s.id)b

on a.friend\_id  = b.id order by a.id),

cte2 as (

select cte.id,cte.name,round(p.salary,2) salary,cte.friend\_id,

round(p2.salary,2) `Friend Salary` from cte join Packages p

on cte.id = p.id left join Packages p2

on cte.friend\_id = p2.id)

SELECT colname,

MAX(CASE WHEN name = 'Ashley' THEN value ELSE 0 end) Ashley,

MAX(CASE WHEN name = 'Samantha' THEN value ELSE 0 end) Samantha,

MAX(CASE WHEN name = 'Julia' THEN value ELSE 0 end) Julia,

MAX(CASE WHEN name = 'Scarlet' THEN value ELSE 0 end) Scarlet

FROM

(

SELECT id,name,id value ,'id' colname  FROM cte2

UNION ALL

SELECT id,name,name value ,'name' colname  FROM cte2

UNION ALL

SELECT id,name,salary value ,'salary' colname  FROM cte2

UNION ALL

SELECT id,name,friend\_id value ,'friend\_id' colname  FROM cte2

UNION ALL

SELECT  id,name,`Friend Salary` value ,'Friend Salary' colname  FROM cte2

)temp\_cte2

GROUP BY colname;

# Q145.

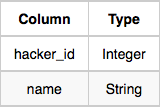
Julia just ﬁnished conducting a coding contest, and she needs your help assembling the leaderboard! Write a query to print the respective hacker\_id and name of hackers who achieved full scores for more than one challenge. Order your output in descending order by the total number of challenges in which the hacker earned a full score. If more than one hacker received full scores in the same number of challenges, then sort them by ascending hacker\_id.

Level - Medium

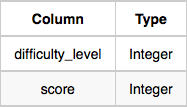
Hint - Use group by and having clause and order by . Input Format

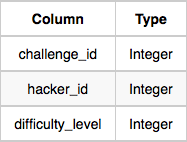
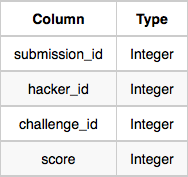
The following tables contain contest data:

* Hackers: The hacker\_id is the id of the hacker, and name is the name of the hacker.



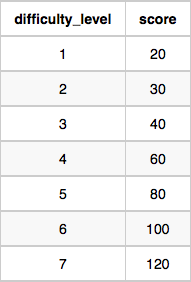
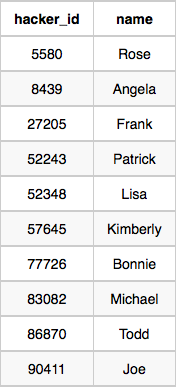
* Diﬃculty: The diﬃcult\_level is the level of diﬃculty of the challenge, and score is the

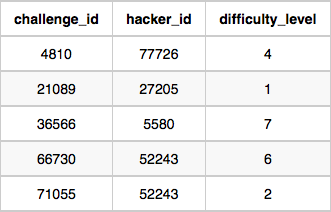
score of the challenge for the diﬃculty level.

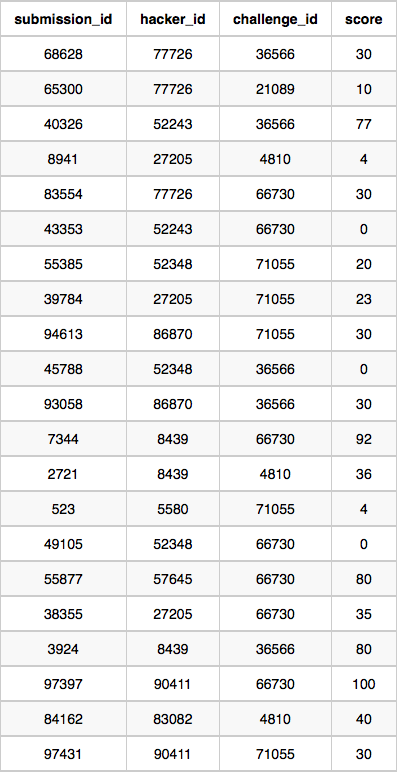
* Challenges: The challenge\_id is the id of the challenge, the hacker\_id is the id of the hacker who created the challenge, and diﬃculty\_level is the level of diﬃculty of the challenge.
* Submissions: The submission\_id is the id of the submission, hacker\_id is the id of the hacker who made the submission, challenge\_id is the id of the challenge that the submission belongs

to, and score is the score of the submission.

Sample Input

Hackers Table: Diﬃculty Table:

Challenges Table: :



Submissions Table

Sample Output

90411 Joe

Explanation

Hacker 86870 got a score of 30 for challenge 71055 with a diﬃculty level of 2, so 86870 earned a full score for this challenge.

Hacker 90411 got a score of 30 for challenge 71055 with a diﬃculty level of 2, so 90411 earned a full score for this challenge.

Hacker 90411 got a score of 100 for challenge 66730 with a diﬃculty level of 6, so 90411 earned a full score for this challenge.

Only hacker 90411 managed to earn a full score for more than one challenge, so we print their hacker\_id and name as 2 space-separated values.

create table Hackers(

hacker\_id int,name varchar(10));

create table Difficulty (difficulty\_level int,score int);

create table Challenges(challenge\_id int,hacker\_id int,difficulty int);

create table Submissions(submission\_id int,hacker\_id int,challenge\_id int,score int);

insert into Hackers values

(5580,'rose'),

(6439,'Angela'),

(27205,'Frank'),

(52243,'Patrick'),

(52348,'Lisa'),

(57645,'Kimberly'),

(77726,'Bonnie'),

(83083,'michael'),

(86870,'Todd'),

(90411,'Joe');

insert into Difficulty values

(1,20),

(2,30),

(3,40),

(4,60),

(5,80),

(6,100),

(7,120);

insert into Challenges values

(4810,77726,4),

(21089,27205,1),

(36566,5580,7),

(66730,52243,6),

(71055,52243,2);

insert into Submissions values

(68628,77726,36566,30),

(65300,77726,21089,10),

(40326,52243,36566,77),

(8941,27205,4810,4),

(83554,77726,66730,30),

(43353,52243,66730,0),

(55385,52348,71055,20),

(39784,27205,71055,23),

(94613,86870,71055,30),

(45788,52348,36566,0),

(93058,86870,36566,30),

(7344,8439,66730,92),

(2721,8439,4810,36),

(523,5580,71055,4),

(49105,52348,66730,0),

(55877,57645,66730,80),

(38355,27205,66730,35),

(3924,8439,36566,80),

(97397,90411,66730,100),

(84162,83082,4810,40),

(97431,90411,71055,30);

SELECT S.hacker\_id, name

FROM Submissions AS S

JOIN Hackers AS H ON S.hacker\_id = H.hacker\_id

JOIN Challenges AS C ON S.challenge\_id = C.challenge\_id

JOIN Difficulty AS D ON C.difficulty = D.difficulty\_level

WHERE S.score = D.score

GROUP BY S.hacker\_id,name

HAVING count(S.challenge\_id) > 1

ORDER BY count(S.challenge\_id) DESC, S.hacker\_id

# Q146.

You are given a table, Projects, containing three columns: Task\_ID, Start\_Date and End\_Date. It is guaranteed that the difference between the End\_Date and the Start\_Date is equal to 1 day for each row in the table.

Level - Medium

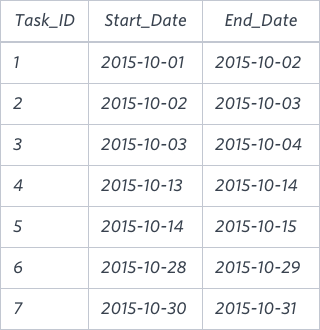
Hint - Use Advance join



If the End\_Date of the tasks are consecutive, then they are part of the same project. Samantha is interested in ﬁnding the total number of different projects completed.

Write a query to output the start and end dates of projects listed by the number of days it took to complete the project in ascending order. If there is more than one project that have the same number of completion days, then order by the start date of the project.

Sample Input



Sample Output

2015-10-28 2015-10-29

2015-10-30 2015-10-31

2015-10-13 2015-10-15

2015-10-01 2015-10-04

Explanation

The example describes following four projects:

* Project 1: Tasks 1, 2 and 3 are completed on consecutive days, so these are part of the project. Thus the start date of project is 2015-10-01 and end date is 2015-10-04, so it took 3 days to complete the project.
* Project 2: Tasks 4 and 5 are completed on consecutive days, so these are part of the project. Thus, the start date of project is 2015-10-13 and end date is 2015-10-15, so it took 2 days to complete the project.
* Project 3: Only task 6 is part of the project. Thus, the start date of project is 2015-10-28 and end date is 2015-10-29, so it took 1 day to complete the project.
* Project 4: Only task 7 is part of the project. Thus, the start date of project is 2015-10-30 and end date is 2015-10-31, so it took 1 day to complete the project.

create table Projects

(Task\_id int NOT NULL,

Start\_date DATE,

End\_date DATE

);

insert into Projects values

(1,'2015-10-01','2015-10-02'),

(2,'2015-10-02','2015-10-03'),

(3,'2015-10-03','2015-10-04'),

(4,'2015-10-13','2015-10-14'),

(5,'2015-10-14','2015-10-15'),

(6,'2015-10-28','2015-10-29'),

(7,'2015-10-30','2015-10-31');

select start\_date,min(end\_date) min\_date from

(select start\_date from (

select start\_date,lag(End\_date) over(order by start\_date) as next\_date

from Projects) a where start\_date  <> next\_date OR next\_date is NULL)x,

(select end\_date from (

select start\_date,end\_date,lead(start\_date) over(order by start\_date) as next\_date

from Projects) b where end\_date <> next\_date OR next\_date is NULL)y

where x.start\_date < y.end\_date

group by start\_date

order by datediff(min\_date,start\_date),start\_date;

# Q147.

In an effort to identify high-value customers, Amazon asked for your help to obtain data about users who go on shopping sprees. A shopping spree occurs when a user makes purchases on 3 or more consecutive days.

List the user IDs who have gone on at least 1 shopping spree in ascending order. transactions Table:

|  |  |
| --- | --- |
| Column Name | Type |
| user\_id | integer |
| amount | ﬂoat |
| transaction\_date | timestamp |

transactions Example Input:

|  |  |  |
| --- | --- | --- |
| user\_id | amount | transaction\_date |
| 1 | 9.99 | 08/01/2022 10:00:00 |
| 1 | 55 | 08/17/2022 10:00:00 |
| 2 | 149.5 | 08/05/2022 10:00:00 |
| 2 | 4.89 | 08/06/2022 10:00:00 |
| 2 | 34 | 08/07/2022 10:00:00 |

Example Output:

2

user\_id

create table transactions(

user\_id int, amount int,transaction\_date date);

insert into transactions values

(1,9.99,STR\_TO\_DATE('08/01/2022 10:00:00','%m/%d/%Y %h:%i:%s')),

(1,55,    STR\_TO\_DATE('08/17/2022 10:00:00','%m/%d/%Y %h:%i:%s')),

(2,149.5, STR\_TO\_DATE('08/05/2022 10:00:00','%m/%d/%Y %h:%i:%s')),

(2,4.89,  STR\_TO\_DATE('08/06/2022 10:00:00','%m/%d/%Y %h:%i:%s')),

(2,34,    STR\_TO\_DATE('08/07/2022 10:00:00','%m/%d/%Y %h:%i:%s'));

select user\_id from(

select user\_id,transaction\_date,transaction\_date -lag(transaction\_date) over

(partition by user\_id order by transaction\_date) date\_diff from

transactions order by user\_id)a where a.date\_diff = 1

group by user\_id having count(date\_diff) >1;

# Q148 .

You are given a table of PayPal payments showing the payer, the recipient, and the amount paid. A two-way unique relationship is established when two people send money back and forth. Write a query to ﬁnd the number of two-way unique relationships in this data.

Assumption:

* A payer can send money to the same recipient multiple times.

payments Table:

|  |  |
| --- | --- |
| Column Name | Type |
| payer\_id | integer |
| recipient\_id | integer |
| amount | integer |

payments Example Input:

|  |  |  |
| --- | --- | --- |
| payer\_id | recipient\_id | amount |
| 101 | 201 | 30 |
| 201 | 101 | 10 |
| 101 | 301 | 20 |
| 301 | 101 | 80 |
| 201 | 301 | 70 |

Example Output:

2

unique\_relationships

create table payments

(payer\_id int,recipient\_id int,amount int);

insert into payments values

(101,201,30),

(201,101,10),

(101,301,20),

(301,101,80),

(201,301,70);

select round(count(\*)/2) unique\_relationships from payments p1 join payments p2

on p1.payer\_id = p2.recipient\_id

and p2.payer\_id = p1.recipient\_id order by p1.payer\_id;

# Q149.

Assume you are given the table below on user transactions. Write a query to obtain the list of customers whose ﬁrst transaction was valued at $50 or more. Output the number of users.

Clariﬁcation:

* Use the transaction\_date ﬁeld to determine which transaction should be labeled as the ﬁrst for each user.
* Use a speciﬁc function (we can't give too much away!) to account for scenarios where a user had multiple transactions on the same day, and one of those was the ﬁrst.

user\_transactions Table:

|  |  |
| --- | --- |
| Column Name | Type |
| transaction\_id | integer |
| user\_id | integer |
| spend | decimal |
| transaction\_date | timestamp |

user\_transactions Example Input:

|  |  |  |  |
| --- | --- | --- | --- |
| transaction\_id | user\_id | spend | transaction\_date |
| 759274 | 111 | 49.50 | 02/03/2022 00:00:00 |
| 850371 | 111 | 51.00 | 03/15/2022 00:00:00 |
| 615348 | 145 | 36.30 | 03/22/2022 00:00:00 |
| 137424 | 156 | 151.00 | 04/04/2022 00:00:00 |
| 248475 | 156 | 87.00 | 04/16/2022 00:00:00 |

Example Output:

users

1

create table user\_transactions

(transaction\_id int, user\_id int,spend int,transaction\_date date);

insert into user\_transactions values

(759274,111,49.50  ,STR\_TO\_DATE('02/03/2022 00:00:00','%m/%d/%Y %H:%i:%s')),

(850371,111,51.00  ,STR\_TO\_DATE('03/15/2022 00:00:00','%m/%d/%Y %H:%i:%s')),

(615348,145,36.30  ,STR\_TO\_DATE('03/22/2022 00:00:00','%m/%d/%Y %H:%i:%s')),

(137424,156,151.00 ,STR\_TO\_DATE('04/04/2022 00:00:00','%m/%d/%Y %H:%i:%s')),

(248475,156,87.00  ,STR\_TO\_DATE('04/16/2022 00:00:00','%m/%d/%Y %H:%i:%s'));

select count(user\_id) users from (

select \*,row\_number()

over(partition by user\_id order by transaction\_date) rw

from user\_transactions )a

where rw = 1

and spend > 50;

# Q150.

Assume you are given the table below containing measurement values obtained from a sensor over several days. Measurements are taken several times within a given day.

Write a query to obtain the sum of the odd-numbered and even-numbered measurements on a particular day, in two different columns.

Note that the 1st, 3rd, 5th measurements within a day are considered odd-numbered measurements and the 2nd, 4th, 6th measurements are even-numbered measurements.

measurements Table:

|  |  |
| --- | --- |
| Column Name | Type |
| measurement\_id | integer |
| measurement\_value | decimal |
| measurement\_time | datetime |

measurements Example Input:

|  |  |  |
| --- | --- | --- |
| measurement\_id | measurement\_value | measurement\_time |
| 131233 | 1109.51 | 07/10/2022 09:00:00 |
| 135211 | 1662.74 | 07/10/2022 11:00:00 |
| 523542 | 1246.24 | 07/10/2022 13:15:00 |
| 143562 | 1124.50 | 07/11/2022 15:00:00 |
| 346462 | 1234.14 | 07/11/2022 16:45:00 |

Example Output:

|  |  |  |
| --- | --- | --- |
| measurement\_day | odd\_sum | even\_sum |
| 07/10/2022 00:00:00 | 2355.75 | 1662.74 |
| 07/11/2022 00:00:00 | 1124.50 | 1234.14 |

create table measurements

(measurement\_id int,measurement\_value float,measurement\_time date);

insert into measurements values

(131233 ,1109.51,STR\_TO\_DATE('07/10/2022 09:00:00','%m/%d/%Y %H:%i:%s')),

(135211 ,1662.74,STR\_TO\_DATE('07/10/2022 11:00:00','%m/%d/%Y %H:%i:%s')),

(523542 ,1246.24,STR\_TO\_DATE('07/10/2022 13:15:00','%m/%d/%Y %H:%i:%s')),

(143562 ,1124.50,STR\_TO\_DATE('07/11/2022 15:00:00','%m/%d/%Y %H:%i:%s')),

(346462 ,1234.14,STR\_TO\_DATE('07/11/2022 16:45:00','%m/%d/%Y %H:%i:%s'));

select distinct measurement\_time as measurement\_day,

round(sum(if(rw%2<>0,measurement\_value,0)) over

(partition by measurement\_time),2) odd\_sum,

round(sum(if(rw%2=0,measurement\_value,0)) over

(partition by measurement\_time),2) even\_sum

 from

(select \*,row\_number() over

(partition by measurement\_time order by measurement\_time) rw from measurements)a;

# Q151.

In an effort to identify high-value customers, Amazon asked for your help to obtain data about users who go on shopping sprees. A shopping spree occurs when a user makes purchases on 3 or more consecutive days.

List the user IDs who have gone on at least 1 shopping spree in ascending order.

Level - Medium Hint - Use self join

transactions Table:

|  |  |
| --- | --- |
| Column Name | Type |
| user\_id | integer |
| amount | ﬂoat |
| transaction\_date | timestamp |

transactions Example Input:

|  |  |  |
| --- | --- | --- |
| user\_id | amount | transaction\_date |
| 1 | 9.99 | 08/01/2022 10:00:00 |
| 1 | 55 | 08/17/2022 10:00:00 |
| 2 | 149.5 | 08/05/2022 10:00:00 |
| 2 | 4.89 | 08/06/2022 10:00:00 |
| 2 | 34 | 08/07/2022 10:00:00 |

Example Output:

2

user\_id

create table transactions(

user\_id int, amount int,transaction\_date date);

insert into transactions values

(1,9.99,STR\_TO\_DATE('08/01/2022 10:00:00','%m/%d/%Y %h:%i:%s')),

(1,55,    STR\_TO\_DATE('08/17/2022 10:00:00','%m/%d/%Y %h:%i:%s')),

(2,149.5, STR\_TO\_DATE('08/05/2022 10:00:00','%m/%d/%Y %h:%i:%s')),

(2,4.89,  STR\_TO\_DATE('08/06/2022 10:00:00','%m/%d/%Y %h:%i:%s')),

(2,34,    STR\_TO\_DATE('08/07/2022 10:00:00','%m/%d/%Y %h:%i:%s'));

select user\_id from(

select user\_id,transaction\_date,transaction\_date -lag(transaction\_date) over

(partition by user\_id order by transaction\_date) date\_diff from

transactions order by user\_id)a where a.date\_diff = 1

group by user\_id having count(date\_diff) >1;

# Q152.

The Airbnb Booking Recommendations team is trying to understand the "substitutability" of two rentals and whether one rental is a good substitute for another. They want you to write a query to ﬁnd the unique combination of two Airbnb rentals with the same exact amenities offered.

Output the count of the unique combination of Airbnb rentals.

Level - Medium

Hint - Use unique statement Assumptions:

* If property 1 has a kitchen and pool, and property 2 has a kitchen and pool too, it is a good substitute and represents a unique matching rental.
* If property 3 has a kitchen, pool and ﬁreplace, and property 4 only has a pool and ﬁreplace, then it is not a good substitute.

rental\_amenities Table:

|  |  |
| --- | --- |
| Column Name | Type |
| rental\_id | integer |
| amenity | string |

rental\_amenities Example Input:

|  |  |
| --- | --- |
| rental\_id | amenity |
| 123 | pool |
| 123 | kitchen |
| 234 | hot tub |
| 234 | ﬁreplace |
| 345 | kitchen |
| 345 | pool |
| 456 | pool |

Example Output:

1

matching\_airbnb

create table rental\_amenities

(rental\_id int,amenity varchar(20));

insert into rental\_amenities values

(123,'pool'),

(123,'kitchen'),

(234,'hot tub'),

(234,'fireplace'),

(345,'kitchen'),

(345,'pool'),

(456,'pool');

with cte as(

select rental\_id,group\_concat(amenity order by amenity) amenity

 from rental\_amenities group by  rental\_id)

select count(distinct c1.rental\_id) matching\_airbnb from cte c1 join cte c2

on c1.amenity = c2.amenity

and c1.rental\_id <> c2.rental\_id ;

# Q153.

Google marketing managers are analysing the performance of various advertising accounts over the last month. They need your help to gather the relevant data.

Write a query to calculate the return on ad spend (ROAS) for each advertiser across all ad campaigns. Round your answer to 2 decimal places, and order your output by the advertiser\_id.

Level - Medium

Hint: ROAS = Ad Revenue / Ad Spend ad\_campaigns Table:

|  |  |
| --- | --- |
| Column Name | Type |
| campaign\_id | integer |
| spend | integer |
| revenue | ﬂoat |
| advertiser\_id | integer |

ad\_campaigns Example Input:

|  |  |  |  |
| --- | --- | --- | --- |
| campaign\_id | spend | revenue | advertiser\_id |
| 1 | 5000 | 7500 | 3 |
| 2 | 1000 | 900 | 1 |
| 3 | 3000 | 12000 | 2 |
| 4 | 500 | 2000 | 4 |
| 5 | 100 | 400 | 4 |

Example Output:

|  |  |
| --- | --- |
| advertiser\_id | ROAS |
| 1 | 0.9 |
| 2 | 4 |
| 3 | 1.5 |
| 4 | 4 |

create table ad\_campaigns

(campaign\_id int,spend int,revenue int,advertiser\_id int);

insert into ad\_campaigns values

(1,5000, 7500  ,3),

(2,1000, 900   ,1),

(3,3000, 12000 ,2),

(4,500 ,2000   ,4),

(5,100 ,400    ,4);

select advertiser\_id,round(sum(revenue)/sum(spend),2) ROAS

from ad\_campaigns

group by advertiser\_id

order by advertiser\_id;

# Q154.

Your team at Accenture is helping a Fortune 500 client revamp their compensation and beneﬁts program. The ﬁrst step in this analysis is to manually review employees who are potentially overpaid or underpaid.

An employee is considered to be potentially overpaid if they earn more than 2 times the average salary for people with the same title. Similarly, an employee might be underpaid if they earn less than half of the average for their title. We'll refer to employees who are both underpaid and overpaid as compensation outliers for the purposes of this problem.

Write a query that shows the following data for each compensation outlier: employee ID, salary, and whether they are potentially overpaid or potentially underpaid (refer to Example Output below).

Hint: ROAS = Ad Revenue / Ad Spend employee\_pay Table:

|  |  |
| --- | --- |
| Column Name | Type |
| employee\_id | integer |
| salary | integer |
| title | varchar |

employee\_pay Example Input:

|  |  |  |
| --- | --- | --- |
| employee\_id | salary | title |
| 101 | 80000 | Data Analyst |
| 102 | 90000 | Data Analyst |
| 103 | 100000 | Data Analyst |
| 104 | 30000 | Data Analyst |
| 105 | 120000 | Data Scientist |
| 106 | 100000 | Data Scientist |
| 107 | 80000 | Data Scientist |
| 108 | 310000 | Data Scientist |

Example Output:

|  |  |  |
| --- | --- | --- |
| employee\_id | salary | status |
| 104 | 30000 | Underpaid |
| 108 | 310000 | Overpaid |

create table employee\_pay

(employee\_id int,salary int,title varchar(20));

insert into employee\_pay values

(101,80000  ,'Data Analyst'),

(102,90000  ,'Data Analyst'),

(103,100000 ,'Data Analyst'),

(104,30000  ,'Data Analyst'),

(105,120000 ,'Data Scientist'),

(106,100000 ,'Data Scientist'),

(107,80000  ,'Data Scientist'),

(108,310000 ,'Data Scientist');

select employee\_id,salary,status from (

select employee\_id,salary,e.title,

case when salary > avgerage\*2 then 'over\_paid'

when salary < avgerage/2 then 'under\_paid'

end as status

 from employee\_pay e join (

select  round(avg(salary)) avgerage ,title from

employee\_pay group by title) a

where e.title  = a.title)x where x.status is  not null;

# Q155.

You are given a table of PayPal payments showing the payer, the recipient, and the amount paid. A two-way unique relationship is established when two people send money back and forth. Write a query to ﬁnd the number of two-way unique relationships in this data.

Assumption:

* A payer can send money to the same recipient multiple times.

Hint- Use the INTERSECT set operator. payments Table:

|  |  |
| --- | --- |
| Column Name | Type |
| payer\_id | integer |
| recipient\_id | integer |
| amount | integer |

payments Example Input:

|  |  |  |
| --- | --- | --- |
| payer\_id | recipient\_id | amount |
| 101 | 201 | 30 |
| 201 | 101 | 10 |
| 101 | 301 | 20 |
| 301 | 101 | 80 |
| 201 | 301 | 70 |

Example Output:

2

unique\_relationships

create table payments

(payer\_id int,recipient\_id int,amount int);

insert into payments values

(101,201,30),

(201,101,10),

(101,301,20),

(301,101,80),

(201,301,70);

select round(count(\*)/2) unique\_relationships from payments p1 join payments p2

on p1.payer\_id = p2.recipient\_id

and p2.payer\_id = p1.recipient\_id order by p1.payer\_id;

# Q156.

Assume you are given the table below containing information on user purchases. Write a query to obtain the number of users who purchased the same product on two or more different days. Output the number of unique users.

*PS. On 26 Oct 2022, we expanded the purchases data set, thus the oficial output may vary from before.*

Hint- Count the distinct number of dates formatted into the DATE format in the COUNT(DISTINCT ).

purchases Table:

|  |  |
| --- | --- |
| Column Name | Type |
| user\_id | integer |
| product\_id | integer |
| quantity | integer |
| purchase\_date | datetime |

purchasesExample Input:

|  |  |  |  |
| --- | --- | --- | --- |
| user\_id | product\_id | quantity | purchase\_date |
| 536 | 3223 | 6 | 01/11/2022 12:33:44 |
| 827 | 3585 | 35 | 02/20/2022 14:05:26 |
| 536 | 3223 | 5 | 03/02/2022 09:33:28 |
| 536 | 1435 | 10 | 03/02/2022 08:40:00 |
| 827 | 2452 | 45 | 04/09/2022 00:00:00 |

Example Output:

1

repeat\_purchasers

create table purchasesExample

(user\_id int,product\_id int,quantity int,purchase\_date date);

insert into purchasesExample values

(536,3223,6  ,STR\_TO\_DATE('01/11/2022 12:33:44','%m/%d/%Y %H:%i:%s')),

(827,3585,35 ,STR\_TO\_DATE('02/20/2022 14:05:26','%m/%d/%Y %H:%i:%s')),

(536,3223,5  ,STR\_TO\_DATE('03/02/2022 09:33:28','%m/%d/%Y %H:%i:%s')),

(536,1435,10 ,STR\_TO\_DATE('03/02/2022 08:40:00','%m/%d/%Y %H:%i:%s')),

(827,2452,45 ,STR\_TO\_DATE('04/09/2022 00:00:00','%m/%d/%Y %H:%i:%s'));

select count(\*)repeat\_purchasers

from

(select user\_id,product\_id,count(\*)same\_product

from purchasesExample group by user\_id,product\_id)a where a.same\_product >=2;

# Q157.

Say you have access to all the transactions for a given merchant account. Write a query to print the cumulative balance of the merchant account at the end of each day, with the total balance reset back to zero at the end of the month. Output the transaction date and cumulative balance.

Hint-You should use CASE. transactions Table:

|  |  |
| --- | --- |
| Column Name | Type |
| transaction\_id | integer |
| type | string ('deposit', 'withdrawal') |
| amount | decimal |
| transaction\_date | timestamp |

transactions Example Input:

|  |  |  |  |
| --- | --- | --- | --- |
| transaction\_id | type | amount | transaction\_date |
| 19153 | deposit | 65.90 | 07/10/2022 10:00:00 |
| 53151 | deposit | 178.55 | 07/08/2022 10:00:00 |
| 29776 | withdrawal | 25.90 | 07/08/2022 10:00:00 |
| 16461 | withdrawal | 45.99 | 07/08/2022 10:00:00 |
| 77134 | deposit | 32.60 | 07/10/2022 10:00:00 |

Example Output:

|  |  |
| --- | --- |
| transaction\_date | balance |
| 07/08/2022 12:00:00 | 106.66 |
| 07/10/2022 12:00:00 | 205.16 |

create table transactions

(transaction\_id int,type varchar(20),amount float,transaction\_date timestamp);

insert into transactions values

(19153,'deposit' ,65.90    ,STR\_TO\_DATE('07/10/2022 10:00:00','%m/%d/%Y %H:%i:%s')),

(53151,'deposit' ,178.55   ,STR\_TO\_DATE('07/08/2022 10:00:00','%m/%d/%Y %H:%i:%s')),

(29776,'withdrawal' ,25.90 ,STR\_TO\_DATE('07/08/2022 10:00:00','%m/%d/%Y %H:%i:%s')),

(16461,'withdrawal' ,45.99 ,STR\_TO\_DATE('07/08/2022 10:00:00','%m/%d/%Y %H:%i:%s')),

(77134,'deposit' ,32.60    ,STR\_TO\_DATE('07/10/2022 10:00:00','%m/%d/%Y %H:%i:%s'));

with cte as(

select transaction\_date,

round(sum(if(type = 'deposit',amount, -amount)),2) balance

from transactions

group by transaction\_date

order by transaction\_date)

select transaction\_date,

sum(balance) over(

    partition by extract(year\_month  from transaction\_date)

order by transaction\_date) balance

from cte;

# Q158.

Assume you are given the table below containing information on Amazon customers and their spend on products belonging to various categories. Identify the top two highest-grossing products within each category in 2022. Output the category, product, and total spend.

Hint- Use where ,and, group by . product\_spend Table:

|  |  |
| --- | --- |
| Column Name | Type |
| category | string |
| product | string |
| user\_id | integer |
| spend | decimal |
| transaction\_date | timestamp |

product\_spend Example Input:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| category | product | user\_id | spend | transaction\_date |
| appliance | refrigerator | 165 | 246.00 | 12/26/2021 12:00:00 |
| appliance | refrigerator | 123 | 299.99 | 03/02/2022 12:00:00 |
| appliance | washing machine | 123 | 219.80 | 03/02/2022 12:00:00 |
| electronics | vacuum | 178 | 152.00 | 04/05/2022 12:00:00 |
| electronics | wireless headset | 156 | 249.90 | 07/08/2022 12:00:00 |
| electronics | vacuum | 145 | 189.00 | 07/15/2022 12:00:00 |

Example Output:

|  |  |  |
| --- | --- | --- |
| category | product | total\_spend |
| appliance | refrigerator | 299.99 |
| appliance | washing machine | 219.80 |
| electronics | vacuum | 341.00 |
| electronics | wireless headset | 249.90 |

create table product\_spend

(category varchar(20),product varchar(20), user\_id int, spend float,transaction\_date timestamp);

insert into product\_spend values

('appliance','refrigerator',165,246.00 ,

STR\_TO\_DATE('12/26/2021 12:00:00','%m/%d/%Y %H:%i:%s')),

('appliance','refrigerator',123,299.99 ,

STR\_TO\_DATE('03/02/2022 12:00:00','%m/%d/%Y %H:%i:%s')),

('appliance','washing machine',123,219.80 ,

STR\_TO\_DATE('03/02/2022 12:00:00','%m/%d/%Y %H:%i:%s')),

('electronics','vacuum',178,152.00 ,

STR\_TO\_DATE('04/05/2022 12:00:00','%m/%d/%Y %H:%i:%s')),

('electronics','wireless headset',156,249.90 ,

STR\_TO\_DATE('07/08/2022 12:00:00','%m/%d/%Y %H:%i:%s')),

('electronics','vacuum',145,189.00 ,

STR\_TO\_DATE('07/15/2022 12:00:00','%m/%d/%Y %H:%i:%s'));

select category,product,spend total\_spend from(

select \*,dense\_rank() over

(partition by category,product order by spend desc) rnk

from product\_spend)a where a.rnk = 1;

# Q159.

Facebook is analysing its user signup data for June 2022. Write a query to generate the churn rate by week in June 2022. Output the week number (1, 2, 3, 4, ...) and the corresponding churn rate rounded to 2 decimal places.

For example, week number 1 represents the dates from 30 May to 5 Jun, and week 2 is from 6 Jun to 12 Jun.

Hint- Use Extract. Assumptions:

* If the last\_login date is within 28 days of the signup\_date, the user can be considered churned.
* If the last\_login is more than 28 days after the signup date, the user didn't churn.

users Table:

|  |  |
| --- | --- |
| Column Name | Type |
| user\_id | integer |
| signup\_date | datetime |
| last\_login | datetime |

users Example Input:

|  |  |  |
| --- | --- | --- |
| user\_id | signup\_date | last\_login |
| 1001 | 06/01/2022 12:00:00 | 07/05/2022 12:00:00 |
| 1002 | 06/03/2022 12:00:00 | 06/15/2022 12:00:00 |
| 1004 | 06/02/2022 12:00:00 | 06/15/2022 12:00:00 |
| 1006 | 06/15/2022 12:00:00 | 06/27/2022 12:00:00 |
| 1012 | 06/16/2022 12:00:00 | 07/22/2022 12:00:00 |

Example Output:

|  |  |
| --- | --- |
| signup\_week | churn\_rate |
| 1 | 66.67 |
| 3 | 50.00 |

User ids 1001, 1002, and 1004 signed up in the ﬁrst week of June 2022. Out of the 3 users, 1002 and 1004's last login is within 28 days from the signup date, hence they are churned users.

To calculate the churn rate, we take churned users divided by total users signup in the week. Hence 2 users / 3 users = 66.67%.

create table users

(user\_id int,signup\_date timestamp,last\_login timestamp);

insert into users values

(1001,STR\_TO\_DATE('06/01/2022 12:00:00','%m/%d/%Y %H:%i:%s'),STR\_TO\_DATE('07/05/2022 12:00:00','%m/%d/%Y %H:%i:%s')),

(1002,STR\_TO\_DATE('06/03/2022 12:00:00','%m/%d/%Y %H:%i:%s'),STR\_TO\_DATE('06/15/2022 12:00:00','%m/%d/%Y %H:%i:%s')),

(1004,STR\_TO\_DATE('06/02/2022 12:00:00','%m/%d/%Y %H:%i:%s'),STR\_TO\_DATE('06/15/2022 12:00:00','%m/%d/%Y %H:%i:%s')),

(1006,STR\_TO\_DATE('06/15/2022 12:00:00','%m/%d/%Y %H:%i:%s'),STR\_TO\_DATE('06/27/2022 12:00:00','%m/%d/%Y %H:%i:%s')),

(1012,STR\_TO\_DATE('06/16/2022 12:00:00','%m/%d/%Y %H:%i:%s'),STR\_TO\_DATE('07/22/2022 12:00:00','%m/%d/%Y %H:%i:%s'));

select \*,

case when extract(week from signup\_date)  = 22 then 1

 when extract(week from signup\_date)  = 23 then 2

 when extract(week from signup\_date)  = 24 then 3

end as wk,

case when DATE\_ADD(signup\_date, interval 28 day) < last\_login then 'churned'

when DATE\_ADD(signup\_date, interval 28 day) > last\_login then 'not\_churned'

end churned\_or\_not

from users;

with cte as(

select \*,

case when extract(week from signup\_date)  = 22 then 1

 when extract(week from signup\_date)  = 23 then 2

 when extract(week from signup\_date)  = 24 then 3

end as wk,

case when DATE\_ADD(signup\_date, interval 28 day) < last\_login then 'churned'

when DATE\_ADD(signup\_date, interval 28 day) > last\_login then 'not\_churned'

end churned\_or\_not

from users),

cte2 as (select wk,churned\_or\_not,count(\*) cnt

from cte group by wk,churned\_or\_not),

cte3 as (select wk,sum(cnt)cnt2 from cte2 group by wk)

select cte2.wk signup\_week,round(100\*max(cnt/cnt2),2) churn\_rate from cte2 join

cte3 on cte2.wk = cte3.wk group by cte2.wk;