**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.

Ans**:with new as** (**select t.username,t.activity,t.startDate,t.endDate from** (**select username,activity,startDate,endDate,dense\_rank() over(partition by username order by endDate desc) as r from UserActivity) t where r=2)**

**Select \* from new UNION select n.username,n.activity,n.startDate,n.endDate from** (**select username,activity,startDate,endDate,dense\_rank() over(partition by username order by endDate desc) as r from UserActivity) n where r=1 and username not in(select username from new);**

**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.

**Ans: with new as** (**select t.username,t.activity,t.startDate,t.endDate from** (**select username,activity,startDate,endDate,dense\_rank() over(partition by username order by endDate desc) as r from UserActivity) t where r=2)**

**Select \* from new UNION select n.username,n.activity,n.startDate,n.endDate from** (**select username,activity,startDate,endDate,dense\_rank() over(partition by username order by endDate desc) as r from UserActivity) n where r=1 and username not in(select username from new);**

**Q103.**

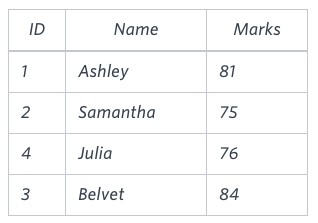
Query the Name of any student in STUDENTS who scored higher than 75 Marks. Order your output by the last three characters of each name. If two or more students both have names ending in the same last three characters (i.e.: Bobby, Robby, etc.), secondary sort them by ascending ID.

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'.

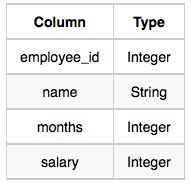
**Ans:select Name from Students where marks>75 order by right(name,3),id;**

**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.

Ans:**select name from Employee where salary >2000 and months<10 order by employee\_id ;**

**Q105**

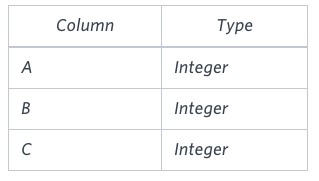
Write a query identifying the type of each record in the TRIANGLES table using its three side lengths.

Output one of the following statements for each record in the table:

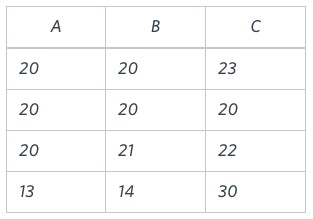
* Equilateral: It's a triangle with sides of equal length.
* Isosceles: It's a triangle with sides of equal length.
* Scalene: It's a triangle with sides of differing lengths.
* Not A Triangle: The given values of A, B, and C don't form a triangle.

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 .

Ans:**select case when A+B>C and C+A>B then (**

**Case when A!=B and B!=C then ‘Scalane’**

**When A=B and B=C then ‘Equilateral’**

**Else ‘Isosceles’ end ) else ‘Not a Traiangle’ end as Result from Triangles;**

**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:

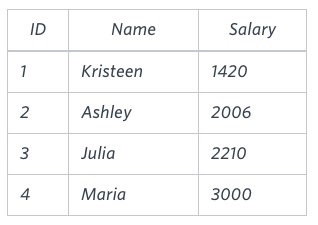


Note: Salary is per month.

Constraints

1000<salary < 10^5

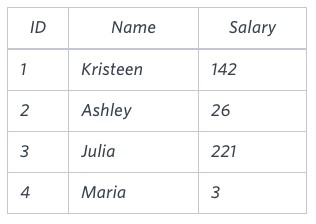
Sample Input



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.

Ans**: select round(ceil(avg(salary)-avg(replace(salary,0,’’))),2) as calculation\_difference from Employees;**

**Q107.**

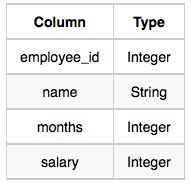
We define an employee's total earnings to be their monthly salary \* months worked, and the maximum total earnings to be the maximum total earnings for any employee in the Employee table. Write a query to find the maximum total earnings for all employees as well as the total number of employees who have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

Ans:**select concat(max(t.earnings),’’,sum(case when earniings=max\_salary then 1 else 0 end )) as Output from**

**(select max(salary\*months) over() as max\_salary,salary\*month as earnings from Employee) t;**

**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 ).

**Ans:select concat(name,’(‘,left(occupation,1),’)’) as occupation\_name from occupations order by name**

**Select concat(‘There are a total of ‘,’ ’,count(occupation),’’,lower(occupation),’s.’) as occupation\_count from occupations group by occupation order by count(occupation),occupation;**

**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.

**Ans:select max(case occupation when ‘Doctor’ then Name end) as Doctors, max(case occupation when ‘Proffessor’ then Name end) as Proffesors, max(case occupation when ‘Singer’ then Name end) as Singers,**

**max(case occupation when ‘Actor’ then Name end) as Actors from**

**(select occupation,name,row\_number() over(partition by occupation order by name) as r ) t group by r;**

**Q110.**

You are given a table, BST, containing two columns: N and P, where N represents the value of a node

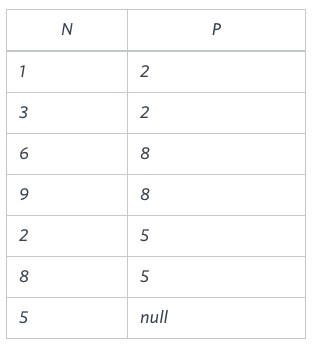
in Binary Tree, and P is the parent of N.



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

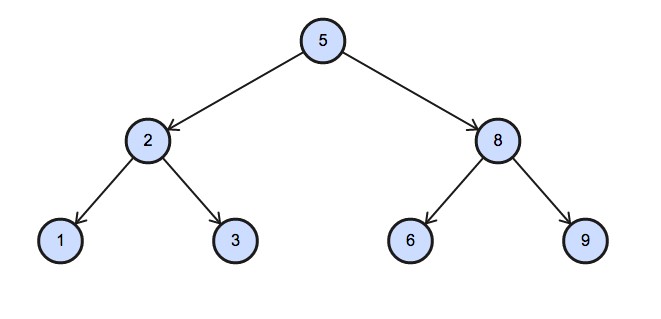


Sample Output

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

Explanation

The Binary Tree below illustrates the sample:



**Ans:select (**

**Case when P is Null then ‘Root’**

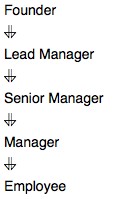
**When N not in (select distinct P from BST where P is not null) then ‘Leaf’**

**Else ‘Inner’**

**End) as Node\_Type from BST ordder by N;**

**Q111 .**

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

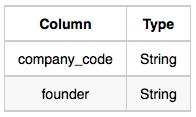
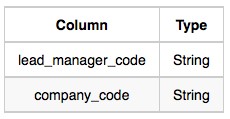
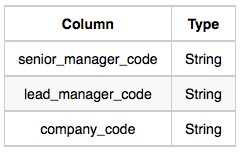
follows this hierarchy: 

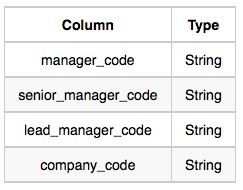
Given the table schemas below, write a query to print the company\_code, founder name, total number of lead managers, total number of senior managers, total number of managers, and total number of employees. Order your output by ascending company\_code. Level - Medium Note:

* + 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.

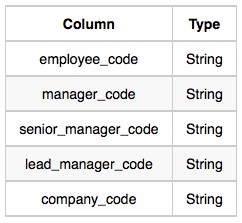
Input Format

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.



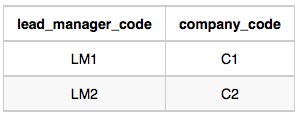
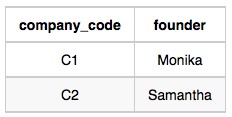
* + Employee: The employee\_code is the code of the employee, the manager\_code is the code of its manager, the senior\_manager\_code is the code of its senior manager, the

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

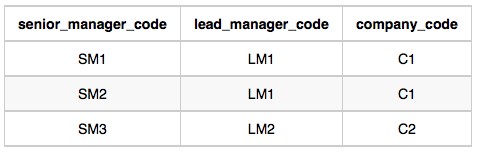
Sample Input

Company Table:

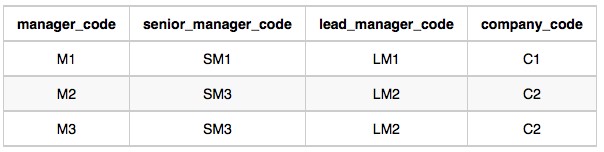
Lead\_Manager Table:



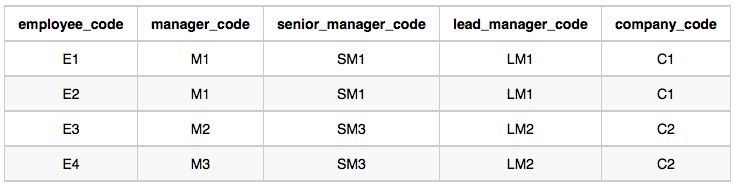
Senior\_Manager Table:



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.

**Ans:select concat(c.company\_code,’’,c.founder,’’,**

**count(distinct l.lead\_manager\_code),’’,**

**count(distinct s.senior\_manager\_code,’’,**

**count(distinct m.manager\_code),’’,**

**count(distinct e.employee\_code)) as output**

**from company c**

**left outer join Lead\_manager l on c.company\_code=l.company\_code**

**left join Senior\_manager s on l.lead\_manager\_code = s.lead\_manager\_code**

**left join Manager m on s.senior\_manager\_code =m.senoor\_manager\_code**

**left join Emplyee e**

**on m.manager\_code = e.manager\_code**

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

**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

**Ans:** **with recursive cte as ( select 2 as num  
union select num+1 from cte where num+1 <= 1000 )select GROUP\_CONCAT(num SEPARATOR "&") as prime from ( select 2 as num union select c1.num from cte c1 inner join cte c2 on c2.num <= round(c1.num/2) group by num having min(c1.num % c2.num) > 0 order by num )t;**

**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

**Ans:with recursive num(n) as (**

**Select 1**

**Union**

**Select n+1 from num where n+1 <=20)**

**Select lpad(‘’,num.n,’\*’) as ‘P(20)’ from num;**

**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 DUAL20 20](#_Toc103236)

[20 21](#_Toc103237)

[22 23](#_Toc103238)

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

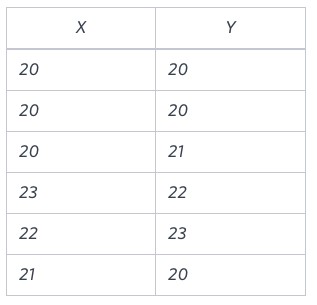


Two pairs (X1, Y1) and (X2, Y2) are said to be symmetric pairs if X1 = Y2 and X2 = Y1.

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

**Ans: with recursive num(n) as (**

**Select 20**

**Union**

**Select n-1 from num where n-1>=1)**

**Select lpad(‘’.num.n,’\*’) as ‘P(20)’ from num;**

**Q115.**

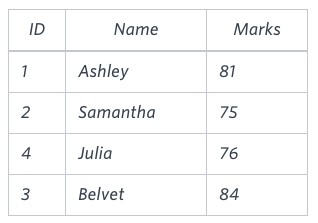
Query the Name of any student in STUDENTS who scored higher than 75 Marks. Order your output by the last three characters of each name. If two or more students both have names ending in the same last three characters (i.e.: Bobby, Robby, etc.), secondary sort them by ascending ID. Level - Easy

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'.

**Ans:select name from students where marks >75 order by right(name,3),id;**

**Q116.**

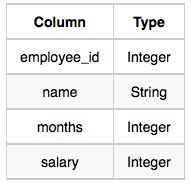
Write a query that prints a list of employee names (i.e.: the name attribute) from the Employee table

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

**Ans:Selectname from Employe**

**order by name;**

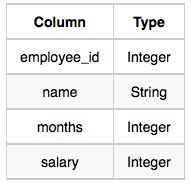
**Q117**. Write a query that prints a list of employee names (i.e.: the name attribute) for employees in Employee having a salary greater than $2000 per month who have been employees for less than 10 months. Sort your result by ascending employee\_id.

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.

**Ans:Select name from employees where salary >2000 and months >=10 order by employee\_id;**

**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:

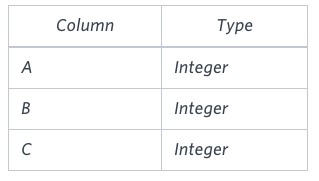
* Equilateral: It's a triangle with sides of equal length.
* Isosceles: It's a triangle with sides of equal length.
* Scalene: It's a triangle with sides of differing lengths.
* Not A Triangle: The given values of A, B, and C don't form a triangle.

Level - Easy

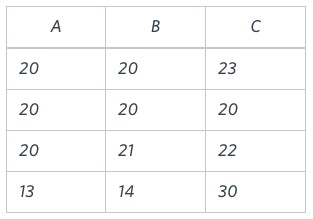
Hint - Use predefined functions for calculation.

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 .

**Ans:select case when A+B>C and B+C >A and C+A>B then (**

**Case when A!=B and B!=C then ‘Scalene’**

**When A=B and B=C then ‘Equilateral’**

**Else ‘Isosceles’**

**End)**

**Else ‘Not a Triangle’**

**End as Result from Triangles;**

**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 | e a  r | product\_i d | curr\_year\_spend | prev\_year\_spend | yoy\_rate |
| 2 | 0  9 | 123424 | 1500.60 |  |  |
| 2 | 0  2  0 | 123424 | 1000.20 | 1500.60 | -33.35 |
| 2 | 0  2 | 123424 | 1246.44 | 1000.20 | 24.62 |
| 2 | 0  2  2 | 123424 | 2145.32 | 1246.44 | 72.12 |

**Ans:select year,product\_id,curr\_year\_spend,coalesce(t.prev\_year\_spend,’’) as prev\_year\_spend,**

**Coalesce(round(curr\_year\_spend -prev\_year\_spend)/prev\_year\_spend \*100,2),’’) as yoy**

**select year(transaction\_date) as year,product\_id,spend as curr\_year\_spend,**

**round(lag(spend,1) over(partition by product\_id order by transaction\_date),2) as prev\_year\_spend from user\_transactions)t;**

**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 |

Ans:

**select item\_type, (case when item\_type = 'prime\_eligible' then floor(500000/sum(square\_footage)) \* count(item\_type) when item\_type = 'not\_prime' then floor((500000 -(select floor(500000/sum(square\_footage)) \* sum(square\_footage) from inventory where item\_type = 'prime\_eligible'))/sum(square\_footage)) \* count(item\_type) end) as item\_count from inventory group by item\_type order by count(item\_type) desc;**

**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 |

**Ans:**

**July:**

**select month(a.event\_date) as month, count(distinct a.user\_id) as monthly\_active\_users from**

**user\_actions a inner join user\_actions b**

**on concat(month(a.event\_date),year(a.event\_date)) = concat(1+month(b.event\_date),year(b.event\_date)) and a.user\_id = b.user\_id**

**where a.event\_type in ('sign-in', 'like', 'comment') and**

**b.event\_type in ('sign-in', 'like', 'comment') and**

**concat(month(a.event\_date),'/',year(a.event\_date))='7/2022'and concat(1+month(b.event\_date),'/',year(b.event\_date)) = '7/2022' group by month(a.event\_date);**

**june:**

**select month(a.event\_date) as month, count(distinct a.user\_id) as monthly\_active\_users from user\_actions a inner join user\_actions b on concat(month(a.event\_date),year(a.event\_date)) = concat(1+month(b.event\_date),year(b.event\_date)) and a.user\_id = b.user\_id where a.event\_type in ('sign-in', 'like', 'comment') and b.event\_type in ('sign-in', 'like', 'comment') and concat(month(a.event\_date),'/',year(a.event\_date)) = '6/2022'  
and concat(1+month(b.event\_date),'/',year(b.event\_date)) = '6/2022' group by month(a.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:

|  |
| --- |
| median |
| 2.5 |

**Ans: with recursive seq as (**

**select searches, num\_users, 1 as c from search\_frequency**

**union**

**select searches, num\_users, c+1 from seq where c < num\_users )**

**select round(avg(t.searches),1) as median from**

**(select searches,row\_number() over(order by searches, c) as r1,**

**row\_number() over(order by searches desc, c desc) as r2**

**from seq order by searches) t where t.r1 in (t.r2, t.r2 - 1, t.r2 + 1);**

**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.

**Ans:** **select user\_id, case when status in ('NEW','EXISTING','CHURN','RESURRECT') and user\_id not in (select user\_id from daily\_pay) then 'CHURN' when status in ('NEW','EXISTING','RESURRECT') and user\_id in (select user\_id from daily\_pay) then 'EXISTING' when status = 'CHURN' and user\_id in (select user\_id from daily\_pay) then 'RESURRECT' end as new\_status from advertiser order by user\_id;**

**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 beoutput 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:

|  |
| --- |
| total\_uptime\_days |
| 21 |

**Ans:select sum(t.individual\_uptime) as total\_uptime\_days from**

**select case when status\_time=’Stop’ then timestampdiff(day,lag(status\_time) over(partition by server\_id order by status\_time),status\_time) end as individual\_uptime from server\_utilization)t;**

**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:

|  |
| --- |
| payment\_count |
| 1 |

**Ans:select sum(case when (unix\_timestamp(t.next\_transaction) - unix\_timestamp(t.transaction\_timestamp))/60 <= 10 then 1 else 0 end) as payment\_count from (select transaction\_timestamp, lead(transaction\_timestamp,1) over(partition by merchant\_id, credit\_card\_id, Amount order by transaction\_timestamp) as next\_transaction from transactions)t;**

**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 received') | successfully', | 'completed | incorrectly', | 'never |
| 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:

|  |
| --- |
| bad\_experience\_pct |
| 75.00 |

* **Ans:** select round(avg(t1.bad\_exp\_pct\_per\_cust),2) as bad\_exp\_pct from ( select t.customer\_id, 100\*sum(case when o.status <> 'completed successfully' then 1 else 0 end)/count(\*) as bad\_exp\_pct\_per\_cust from ( select customer\_id, signup\_timestamp from customers where month(signup\_timestamp) = 6 ) t inner join orders o on o.customer\_id = t.customer\_id where timestampdiff(day, t.signup\_timestamp, o.order\_timestamp) <= 13 group by t.customer\_id ) t1;

**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.

* **Ans:** select gender, day, sum(score\_points) over(partition by gender order by day) as total from Scores group by gender, day 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:

|  |
| --- |
| country |
| Peru |

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.

**Ans:** **select t3.Name from ( select t2.Name, avg(t1.duration) over(partition by t2.Name) as avg\_call\_duration, avg(t1.duration) over() as global\_average from ((select cl.caller\_id as id, cl.duration from Calls cl) union (select cl.callee\_id as id, cl.duration from Calls cl)) t1 left join (select p.id, c.Name from Person p left JOIN Country c ON cast(left(p.phone\_number,3) as int) = cast(c.country\_code as int)) t2 ON t1.id = t2.id) t3 where t3.avg\_call\_duration > global\_average group by t3.Name;**

**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:

|  |
| --- |
| median |
| 0 |

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.

* **Ans:** with recursive seq as ( select num, frequency, 1 as c from Numbers union select num, frequency, c+1 from seq where c < frequency ) select round(avg(t.num),1) as median from ( select num,row\_number() over(order by num, c) as r1, row\_number() over(order by num desc, c desc) as r2 from seq order by num ) t where t.r1 in (t.r2, t.r2 - 1,t.r2 + 1);

**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.

* **Ans:** select distinct concat(year(t.pay\_date),'-',month(t.pay\_date)) as pay\_month, t.department\_id, case when monthly\_department\_avg\_salary > monthly\_average\_salary then 'higher' when monthly\_department\_avg\_salary < monthly\_average\_salary then 'lower' else 'same' end as Comparison from (select s.pay\_date, e.department\_id, avg(s.amount) over(partition by month(s.pay\_date), e.department\_id) as monthly\_department\_avg\_salary,  
  avg(s.amount) over(partition by month(s.pay\_date)) as monthly\_average\_salary from Salary s left join Employee e on s.employee\_id = e.employee\_id) t order by t.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

**Ans:** **select t1.install\_dt, count(player\_id) as installs, round(count(t1.next\_install)/count(t1.player\_id),2) as Day1\_retention from ( select t.player\_id, t.install\_dt, a.event\_date as next\_install from ( select player\_id, min(event\_date) as install\_dt from Activity group by player\_id ) t left join Activity a on t. player\_id = a.player\_id and a.event\_date = t.install\_dt + 1 ) t1 group by install\_dt;**

**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 |

* **Ans:** select t2.group\_id, t2.player\_id from ( select t1.group\_id, t1.player\_id, dense\_rank() over(partition by group\_id order by score desc, player\_id) as r from ( select p.\*, case when p.player\_id = m.first\_player then m.first\_score when p.player\_id = m.second\_player then m.second\_score end as score from Players p, Matches m where player\_id in (first\_player, second\_player) ) t1 ) t2 where r = 1;

**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.

* **Ans:** select t.student\_id, t.student\_name from (select s.student\_name, s.student\_id, count(e.student\_id) over(partition by student\_name) as exams\_given, case when e.score > min(e.score) over(partition by e.exam\_id) and e.score < max(e.score) over(partition by e.exam\_id) then 1 else 0 end as quiet from Exam e left join Student s on e.student\_id = s.student\_id)t group by t.student\_name, t.student\_id, t.exams\_given having sum(t.quiet) = t.exams\_given

**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.

* **Ans:** select t.student\_id, t.student\_name from (select s.student\_name, s.student\_id, count(e.student\_id) over(partition by student\_name) as exams\_given, case when e.score > min(e.score) over(partition by e.exam\_id) and e.score < max(e.score) over(partition by e.exam\_id) then 1 else 0 end as quiet from Exam e left join Student s on e.student\_id = s.student\_id)t group by t.student\_name, t.student\_id, t.exams\_given having sum(t.quiet) = t.exams\_given

**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.

**Ans: with new as (select t.username, t.activity, t.startDate, t.endDate from( select username, activity, startDate, endDate, dense\_rank() over(partition by username order by endDate desc) as r from UserActivity)t where r = 2 ) select \* from new union select n.username, n.activity, n.startDate, n.endDate from( select username, activity, startDate, endDate, dense\_rank() over(partition by username order by endDate desc) as r from UserActivity)n where r = 1 and username not in (select username from new);**

**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.

Ans: **with new as (select t.username, t.activity, t.startDate, t.endDate from( select username, activity, startDate, endDate, dense\_rank() over(partition by username order by endDate desc) as r from UserActivity)t where r = 2 ) select \* from new union select n.username, n.activity, n.startDate, n.endDate from( select username, activity, startDate, endDate, dense\_rank() over(partition by username order by endDate desc) as r from UserActivity)n where r = 1 and username not in (select username from new);**

**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:

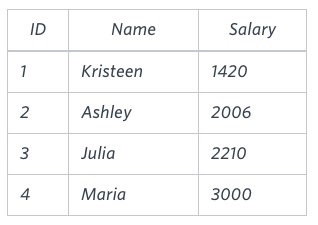


Note: Salary is per month.

Constraints

1000<salary < 10^5

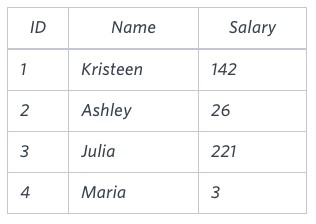
Sample Input



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.

Ans**: select ceil(avg(salary) - avg(replace(salary, 0, ''))) as calculation\_difference from Employees;**

**Q138.**

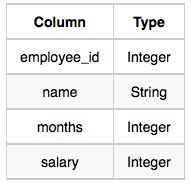
We define an employee's total earnings to be their monthly salary \* months worked, and the maximum total earnings to be the maximum total earnings for any employee in the Employee table. Write a query to find the maximum total earnings for all employees as well as the total number of employees who have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

Ans**: select concat(max(t.earnings), ' ', sum(case when earnings = max\_salary then 1 else 0 end)) as Output from ( select max(salary\*months) over() as max\_salary, salary\*months as earnings from Employee) t;**

**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 ).

**Ans : select concat(name, '(', left(occupation,1),')') as name\_occupation) from Occupations order by name; select concat('There are a total of', ' ', count(occupation), ' ', lower(occupation), 's.') as occupation\_count from Occupations group by occupation order by count(occupation), occupation;**

**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.**

**Ans:** **select max(case Occupation when 'Doctor' then Name end) as Doctors, max(case Occupation when 'Professor' then Name end) as Professors, max(case Occupation when 'Singer' then Name end) as Singers, max(case Occupation when 'Actor' then Name end) as Actors from ( select occupation, name, row\_number() over(partition by Occupation order by name) as r from Occupations ) t group by r;**

**Q141**.

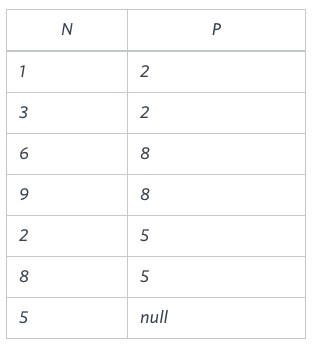
You are given a table, BST, containing two columns: N and P, where N represents the value of a node in Binary Tree, and P is the parent of N.



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

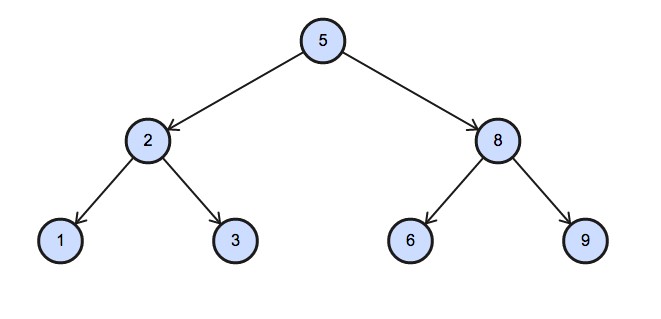


Sample Output

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

Explanation

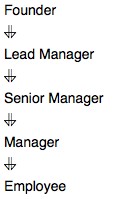
The Binary Tree below illustrates the sample:



Ans: **select ( case when P is NULL then 'Root' when N not in (select distinct P from BST where P is not null) then 'Leaf' else 'Inner' end ) as Node\_Type from BST order by N;**

**Q142 .**

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

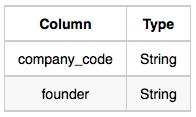
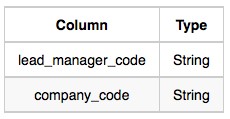
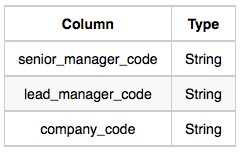
follows this hierarchy: 

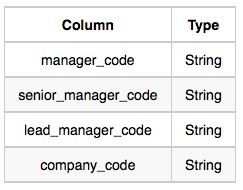
Given the table schemas below, write a query to print the company\_code, founder name, total number of lead managers, total number of senior managers, total number of managers, and total number of employees. Order your output by ascending company\_code. Level - Medium Note:

* + 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.

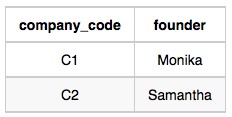
Input Format

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.



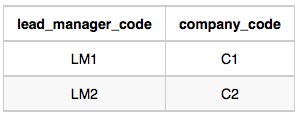
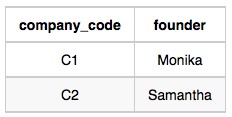
* + Employee: The employee\_code is the code of the employee, the manager\_code is the code of its manager, the senior\_manager\_code is the code of its senior manager, the

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

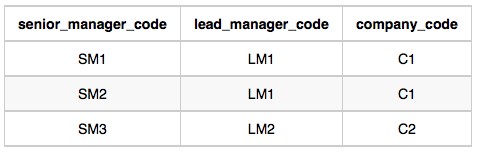
Sample Input

Company Table:

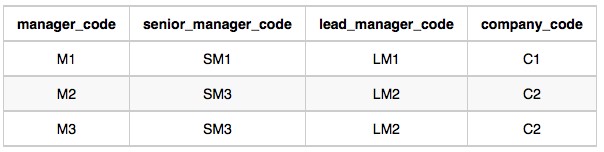
Lead\_Manager Table:



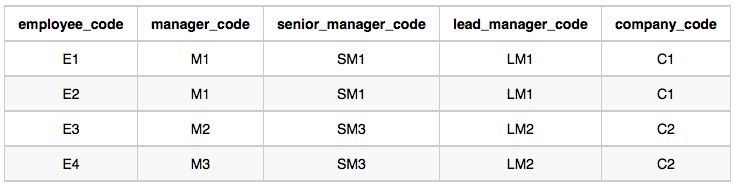
Senior\_Manager Table:



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.

* Ans: **select concat(c.company\_code, ' ', c.founder, ' ', count(distinct l.lead\_manager\_code), ' ', count(distinct s.senior\_manager\_code), ' ', count(distinct m.manager\_code), ' ', count(distinct e.employee\_code)) as Output from Company c left outer join Lead\_Manager l on c.company\_code = l.company\_code left join Senior\_Manager s on l.lead\_manager\_code = s.lead\_manager\_code left join Manager m on s.senior\_manager\_code = m.senior\_manager\_code left join Employee e on m.manager\_code = e.manager\_code group by c.company\_code, c.founder order by c.company\_code;**

**Q143 .**

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



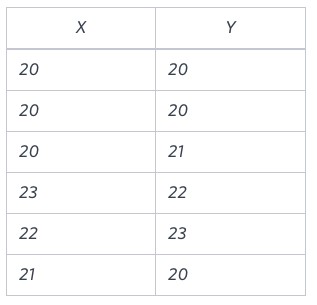
Two pairs (X1, Y1) and (X2, Y2) are said to be symmetric pairs if X1 = Y2 and X2 = Y1.

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

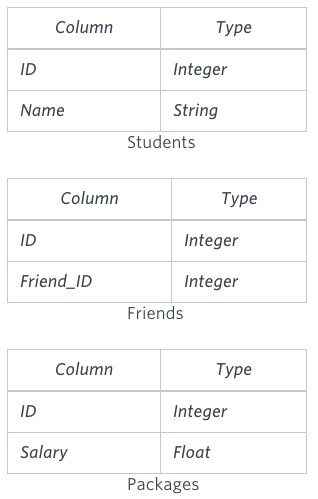
20 21

# 22 23

* **Ans:** **select distinct a.X, a.Y from (select \*, row\_number() over(order by X) as r1 from Functions) a inner join (select \*,row\_number() over(order by X) as r2 from Functions) b on a.X = b.Y and b.X = a.Y where a.X <= a.Y and a.r1 <> b.r2 order by a.X;**

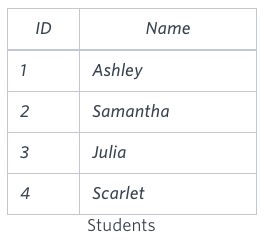
**Q144 .**

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

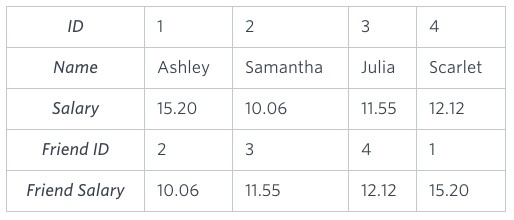
Samantha

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

**Ans:** **select s.name from Students s join Friends f on s.id = f.id join Packages sp on sp.id = s.id join Packages fp on fp.id = f.friend\_id where fp.salary > sp.salary order by fp.salary;**

**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.

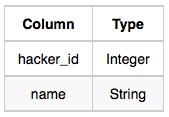
Level - Medium

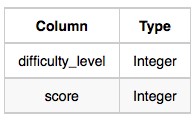
Hint - Use group by and having clause and order by .

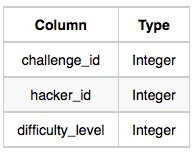
Input Format

The following tables contain contest data:

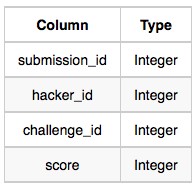
* Hackers: The hacker\_id is the id of the hacker, and name is the name of the hacker.



* 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

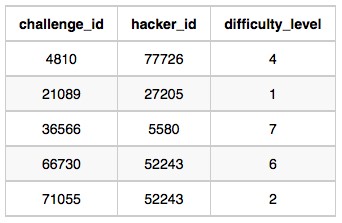
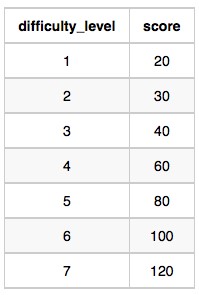
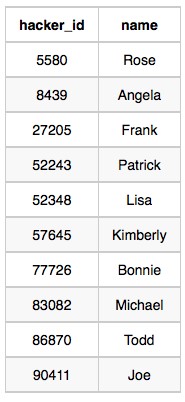
to, and score is the score of the submission. 

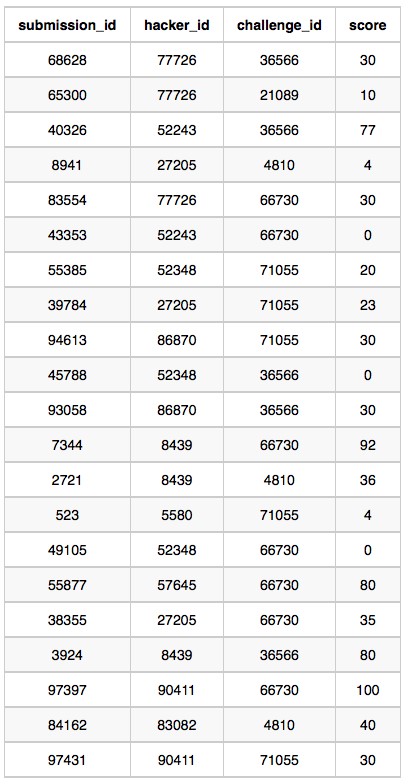
Sample Input

Hackers Table: Challenges Table:

Difficulty 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.

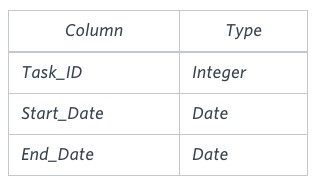
**Ans:select concat(t1.hacker\_id, ' ', t1.name) as Result from ( select t.hacker\_id, t.name, dense\_rank() over(order by full\_score\_challenge\_count desc) as r from ( select h.hacker\_id, h.name, count(h.hacker\_id) as full\_score\_challenge\_count from Submissions s join Hackers h on s.hacker\_id = h.hacker\_id join Challenges c on s.challenge\_id = c.challenge\_id join Difficulty d on d.difficulty\_level = c.difficulty\_level where s.score = d.score group by h.hacker\_id, h.name having full\_score\_challenge\_count > 1 ) t ) t1 where t1.r = 1 order by t1.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

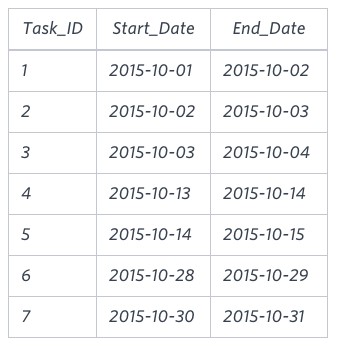
Hint - Use Advance join



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.
* Ans**: select s.start\_date, min(e.end\_date) as end\_date, (min(e.end\_date) - s.start\_date) as number\_of\_days from (select start\_date from Projects where start\_date - 1 not in (select start\_date from Projects)) s, (select end\_date from Projects where end\_date + 1 not in (select end\_date from Projects)) e where s.start\_date <= e.end\_date group by s.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:

|  |
| --- |
| user\_id |
| 2 |

**Ans:**

**select distinct t.user\_id from ( select user\_id, transaction\_date as first, lead(transaction\_date,1) over(partition by user\_id order by transaction\_date) as second, lead(transaction\_date,2) over(partition by user\_id order by transaction\_date) as third from transactions ) t where timestampdiff(day, first, second) = 1 and timestampdiff(day, second, third) = 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:

|  |
| --- |
| unique\_relationships |
| 2 |

* **Ans: select count(\*) as unique\_relationshis from (select count(\*) as relation\_count from ( select greatest(payer\_id, recipient\_id) as person1, least(payer\_id, recipient\_id) as person2 from (select distinct \* from payments) t ) t1 group by person1, person2 ) t2 where relation\_count = 2;**

**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

**Ans: select count(\*) as users from ( select transaction\_id, user\_id, spend, row\_number() over(partition by user\_id order by transaction\_date) as r from user\_transactions ) t where t.r =1 and t.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 |

**Ans: select measurement\_day, round(sum(case when r % 2 != 0 then measurement\_value else 0 end),2) as odd\_sum, round(sum(case when r % 2 = 0 then measurement\_value else 0 end),2) as even\_sum from ( select date\_format(measurement\_time, '%m/%d/%Y 00:00:00') as measurement\_day, measurement\_value, row\_number() over(partition by date(measurement\_time) order by measurement\_time) as r from measurements )t group by measurement\_day;**

**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:

|  |
| --- |
| user\_id |
| 2 |

**Ans: select distinct t.user\_id from ( select user\_id, transaction\_date as first, lead(transaction\_date,1) over(partition by user\_id order by transaction\_date) as second, lead(transaction\_date,2) over(partition by user\_id order by transaction\_date) as third from transactions ) t where timestampdiff(day, first, second) = 1 and timestampdiff(day, second, third) = 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:

|  |
| --- |
| matching\_airbnb |
| 1 |

* **Ans:** **select count(t1.amenity\_count) as matching\_airbnb from ( select t.amenities, count(\*) as amenity\_count from ( select rental\_id, group\_concat(amenity order by amenity) amenities from rental\_amenities group by rental\_id )t group by t.amenities )t1 where t1.amenity\_count>1;**

**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.

|  |
| --- |
| advertiser\_id |

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.

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 |

* **Ans:** **select advertiser\_id, sum(revenue)/sum(spend) as 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 |

Ans: **select t.employee\_id, t.salary, case when t.salary > t.base\_for\_overpaid then 'Overpaid' when t.salary < t.base\_for\_underpaid then 'Underpaid' end as status from (select employee\_id, salary, 2\*avg(salary) over(partition by title) as base\_for\_overpaid, 0.5\*avg(salary) over(partition by title) as base\_for\_underpaid from employee\_pay )t having status is not null order by t.employee\_id;**

**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:

|  |
| --- |
| unique\_relationships |
| 2 |

* **Ans: select count(\*) as unique\_relationshis from (select count(\*) as relation\_count from ( select greatest(payer\_id, recipient\_id) as person1, least(payer\_id, recipient\_id) as person2 from (select distinct \* from payments) t ) t1 group by person1, person2 ) t2 where relation\_count = 2;**

**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 official 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:

|  |
| --- |
| repeat\_purchasers |
| 1 |

**Ans:select count(distinct t.user\_id) as repeat\_purchasers from ( select user\_id, product\_id, count(\*) as c from purchases group by user\_id, product\_id having c > 1 ) t;**

**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 |

* **Ans: select distinct DATE\_FORMAT(transaction\_date, '%m/%d/%Y 12:00:00'), round(sum(amount) over(partition by month(transaction\_date) order by transaction\_date),2) as balance from ( select transaction\_date, case when type = 'deposit' then amount else -amount end as amount from transactions )t;**

**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 |

**Ans:select t.category, t.product, t.total\_spend from ( select category, product, round(sum(spend),2) as total\_spend, dense\_rank() over(partition by category order by sum(spend) desc) as r from product\_spend group by category, product ) t where r <= 2**

**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%.

Ans:

**select week(signup\_date), round(100\*sum(case when timestampdiff(day,signup\_date,last\_login) <= 28 then 1 else 0 end)/count(\*),2) as churn\_rate from users group by week(signup\_date);**