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.

```sql
select * from (
select *,row_number() over(partition by username order by start_date) as row_num,count(*) over (partition by username) as instances
from user_activity
)b
where (row_num=2 and instances>1) or (row_num=1 and instances=1)
```

Q102.

Table: UserActivity

Column Name	Type
username	varchar
activity	varchar
startDate	Date
endDate	Date

There is no primary key for this table. It may contain duplicates.
This table contains information about the activity performed by each user in a period of time.
A person with a username performed an activity from startDate to endDate.

Write an SQL query to show the second most recent activity of each user.
If the user only has one activity, return that one. A user cannot perform more than one activity at the same time.
Return the result table in any order.
The query result format is in the following example.

Input:
UserActivity table:

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

Output:

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

Explanation:
The most recent activity of Alice is Travel from 2020-02-24 to 2020-02-28, before that she was dancing from 2020-02-21 to 2020-02-23.
Bob only has one record, we just take that one.

Answer Same As Question 101

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:

Column	Type
ID	Integer
Name	String
Marks	Integer

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

Sample Input

ID	Name	Marks
1	Ashley	81
2	Samantha	75
4	Julia	76
3	Belvet	84

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'.

```
select *,SUBSTRING(name, LENGTH(name) - 2, LENGTH(name)) as last_3 from students
where marks>75
order by 4 asc
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

id	name	marks	last_3
1	ASHLEY	81	LEY
3	JULIA	76	LIA
4	BELVET	84	VET

Result 224

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:

Column	Type
employee_id	Integer
name	String
months	Integer
salary	Integer

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

employee_id	name	months	salary
12228	Rose	15	1968
33645	Angela	1	3443
45692	Frank	17	1608
56118	Patrick	7	1345
59725	Lisa	11	2330
74197	Kimberly	16	4372
78454	Bonnie	8	1771
83565	Michael	6	2017
98607	Todd	5	3396
99989	Joe	9	3573

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.

```
select name from employee
where salary>2000 and month<10
order by 1
```

Result Grid | Filter Rows: | Export: | Wrap Cell

name
ANGELA
JOE
MICHAEL
TODD

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:

Column	Type
A	Integer
B	Integer
C	Integer

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

A	B	C
20	20	23
20	20	20
20	21	22
13	14	30

Sample Output
Isosceles
Equilateral
Scalene
Not A Triangle

Explanation
Values in the tuple(20,20,23) form an Isosceles triangle, because $A \equiv B$.
Values in the tuple(20,20,20) form an Equilateral triangle, because $A \equiv B \equiv C$. Values in the tuple(20,21,22) form a Scalene triangle, because $A \neq B \neq C$.

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 .

```sql
select *,case
when a=b and b=c then 'Equilateral'
WHEN a = b OR b = c OR c = a THEN 'ISSOCELES'
WHEN a <> b AND b <> c THEN 'SCALEAN'
WHEN a + b <= c OR b + c <= a OR a + c <= b THEN 'NOT A TRIANGLE'
end as remark
from triangles
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

a	b	c	remark
20	20	23	ISSOCELES
20	20	20	Equilateral
20	21	22	SCALEAN
13	14	30	SCALEAN

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:

Column	Type
ID	Integer
Name	String
Salary	Integer

Note: Salary is per month.
Constraints
1000<salary < 10^5
Sample Input

ID	Name	Salary
1	Kristeen	1420
2	Ashley	2006
3	Julia	2210
4	Maria	3000

Sample Output
2061

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

ID	Name	Salary
1	Kristeen	142
2	Ashley	26
3	Julia	221
4	Maria	3

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.

```
select round(avg(salary)-avg(wrong_salary)) as round_error
 from (
select *,REPLACE(salary, 0, '') as wrong_salary
 from employees
)b
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

	round_error
▶	2061

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:

Column	Type
employee_id	Integer
name	String
months	Integer
salary	Integer

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

employee_id	name	months	salary
12228	Rose	15	1968
33645	Angela	1	3443
45692	Frank	17	1608
56118	Patrick	7	1345
59725	Lisa	11	2330
74197	Kimberly	16	4372
78454	Bonnie	8	1771
83565	Michael	6	2017
98607	Todd	5	3396
99989	Joe	9	3573

Sample Output
69952 1

Explanation:

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

employee_id	name	months	salary	earnings
12228	Rose	15	1968	29520
33645	Angela	1	3443	3443
45692	Frank	17	1608	27336
56118	Patrick	7	1345	9415
59725	Lisa	11	2330	25630
74197	Kimberly	16	4372	69952
78454	Bonnie	8	1771	14168
83565	Michael	6	2017	12102
98607	Todd	5	3396	16980
99989	Joe	9	3573	32157

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.

```
select months*salary as max_total_salary,count(name) as no_of_emps
 from employee where months*salary=(
select max(total_earning)
from (
select *,months*salary as total_earning from employee
)b )
group by 1
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

	max_total_salary	no_of_emps
▶	69952	1

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.

2. 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:

Column	*Type*
Name	*String*
Occupation	*String*

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

Sample Input

An OCCUPATIONS table that contains the following records:

Name	Occupation
Samantha	Doctor
Julia	Actor
Maria	Actor
Meera	Singer
Ashely	Professor
Ketty	Professor
Christeen	Professor
Jane	Actor
Jenny	Doctor
Priya	Singer

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).

Part 1

```
select concat(name,'(',substring(occupation,1,1),')') as name_profession
from occupations
order by name
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

name_profession
ASHLEY(P)
CHRISTEEN(P)
JANE(A)
JENNY(D)
JULIA(A)

Result 255

Output

Part 2

```
select concat('(',substring(occupation,1,1),')') as profession,count(occupation) as instances
from occupations
group by 1
order by 1
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

profession	instances
(A)	3
(D)	2
(P)	3
(S)	2

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:

Column	*Type*
Name	*String*
Occupation	*String*

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

Name	*Occupation*
Samantha	*Doctor*
Julia	*Actor*
Maria	*Actor*
Meera	*Singer*
Ashely	*Professor*
Ketty	*Professor*
Christeen	*Professor*
Jane	*Actor*
Jenny	*Doctor*
Priya	*Singer*

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.

```sql
select row_num,
max(case when occupation ='DOCTOR' then name end) as DOCTOR,
max(case when occupation ='ACTOR' then name end) as ACTOR,
max(case when occupation ='PROFESSOR' then name end) as PROGFESSOR,
max(case when occupation ='SINGER' then name end) as SINGER
from (
select name,occupation,row_number() over( partition by occupation ) as row_num
from occupations
)b
group by 1
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

row_num	DOCTOR	ACTOR	PROGFESSOR	SINGER
1	SAMNATHA	JULIA	ASHLEY	MEERA
2	JENNY	MARIA	KETTY	PRIYA
3	NULL	JANE	CHRISTEEN	NULL

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.

Column	*Type*
N	*Integer*
P	*Integer*

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:

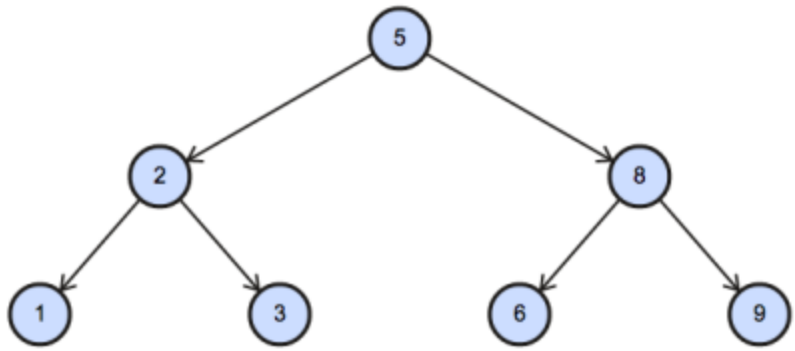
- Root: If node is root node.
- Leaf: If node is leaf node.
- Inner: If node is neither root nor leaf node.

Sample Input

N	*P*
1	*2*
3	*2*
6	*8*
9	*8*
2	*5*
8	*5*
5	*null*

Sample Output
1 Leaf
2 Inner
3 Leaf
5 Root
6 Leaf
8 Inner
9 Leaf

Explanation
The Binary Tree below illustrates the sample:




```
select *,
case
when n not in  (select distinct p from bst where p is not null) then 'leaf'
when p is null then 'root'
else 'inner' end as remark
 from bst
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

n	p	remark
6	8	leaf
9	8	leaf
2	5	inner
8	5	inner
5	NULL	root

Result 269

Output

Q111 .

Amber's conglomerate corporation just acquired some new companies. Each of the companies follows this hierarchy:

Founder
⇓
Lead Manager
⇓
Senior Manager
⇓
Manager
⇓
Employee

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.

Column	Type
company_code	String
founder	String

- Lead_Manager: The lead_manager_code is the code of the lead manager, and the company_code is the code of the working company.

Column	Type
lead_manager_code	String
company_code	String

- 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.

Column	Type
senior_manager_code	String
lead_manager_code	String
company_code	String

- 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.

Column	Type
manager_code	String
senior_manager_code	String
lead_manager_code	String
company_code	String

- 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.

Column	Type
employee_code	String
manager_code	String
senior_manager_code	String
lead_manager_code	String
company_code	String

Sample Input

Company Table:

company_code	founder
C1	Monika
C2	Samantha

Lead_Manager Table:

lead_manager_code	company_code
LM1	C1
LM2	C2

Senior_Manager Table:

senior_manager_code	lead_manager_code	company_code
SM1	LM1	C1
SM2	LM1	C1
SM3	LM2	C2

Manager Table:

manager_code	senior_manager_code	lead_manager_code	company_code
M1	SM1	LM1	C1
M2	SM3	LM2	C2
M3	SM3	LM2	C2

Employee Table:

employee_code	manager_code	senior_manager_code	lead_manager_code	company_code
E1	M1	SM1	LM1	C1
E2	M1	SM1	LM1	C1
E3	M2	SM3	LM2	C2
E4	M3	SM3	LM2	C2

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.

```
select company_code,founder,count(distinct lead_manager_code),count(distinct senior_manager_code),count(distinct manager_code),count(
distinct employee_code) from (
select company.company_code,founder,lead_manager.lead_manager_code,senior_manager.senior_manager_code,manager.manager_code,employee_code
from company
left join lead_manager on lead_manager.company_code=company.company_code
left join senior_manager on senior_manager.company_code=company.company_code
left join manager on manager.company_code=company.company_code
left join employee on employee.company_code=company.company_code
)b
group by 1,2
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

company_code	founder	count(distinct lead_manager_code)	count(distinct senior_manager_code)	count(distinct manager_code)	count(distinct employee_code)
C1	MONIKA	1	2	1	2
C2	SAMANTHA	1	1	2	2

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

```sql
WITH RECURSIVE number_generation AS (
SELECT
1 num
union
select num+1 from number_generation
where num<1000 ),
number_generation2 AS (
SELECT n1.num AS numm
FROM
number_generation n1
JOIN number_generation n2 WHERE n1.num % n2.num = 0
GROUP BY n1.num
HAVING
COUNT(n1.num) = 2)
SELECT
        group_concat(numm ORDER BY numm SEPARATOR '&') AS prime_numbers
FROM
        number_generation2;
```

Q113.

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

```
*
**
***
****
*****
```

Write a query to print the pattern P(20).
Level - Easy
Source - Hackerrank
Hint - Use SYS_CONNECT_BY_PATH(NULL, '* ') FROM DUAL

```
WITH RECURSIVE generate_numbers AS
(
SELECT 1 AS n
UNION
SELECT n+1
FROM generate_numbers WHERE n<20
)
SELECT repeat('*',n) as pattern
FROM generate_numbers;
```

Q114.

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

```
*****
****
***
**
*
```

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

Level - Easy

Hint - Use SYS_CONNECT_BY_PATH(NULL, '* ') FROM DUAL

```
WITH RECURSIVE generate_numbers AS
(
SELECT 20 AS n
UNION
SELECT n-1
FROM generate_numbers WHERE n>1
)
SELECT repeat('*',n) as pattern
FROM generate_numbers;
```

Q.115 SAME AS Q.103

Q.116 SAME AS Q.79

Q.117 SAME AS Q.104

Q.118 SAME AS Q.105

Q.119 SAME AS Q.80

Q.120 SAME AS Q.81

Q.121 SAME AS Q.82

Q.122 SAME AS Q.83

Q.123 SAME AS Q.84

Q.124 SAME AS Q.85

Q.125 SAME AS Q.86

Q.126 SAME AS Q.87

Q.127 SAME AS Q.68

Q.128 SAME AS Q.55

Q.129 SAME AS Q.90

Q.130 SAME AS Q.91

Q.131 SAME AS Q.92

Q.132 SAME AS Q.50

Q.133 SAME AS Q.94

Q.134 SAME AS Q.94

Q.135 SAME AS Q.101

Q.136 SAME AS Q.101

Q.137 SAME AS Q.106

Q.138 SAME AS Q.105

Q.139 SAME AS Q.105

Q.140 SAME AS Q.105

Q.141 SAME AS Q.110

Q.142 SAME AS Q.111

Q143 .

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

Column	*Type*
X	*Integer*
Y	*Integer*

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

X	*Y*
20	*20*
20	*20*
20	*21*
23	*22*
22	*23*
21	*20*

Sample Output
20 20
20 21
22 23

```sql
WITH temp_functions AS (
SELECT x,y, ROW_NUMBER() OVER (ORDER BY x, y) AS row_num
FROM functions)
SELECT DISTINCT f1.x,f1.y
FROM temp_functions f1
JOIN temp_functions f2
ON f1.x = f2.y AND f1.y = f2.x
AND f1.row_num <> f2.row_num
WHERE f1.x <= f1.y
ORDER BY f1.x;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

x	y
20	20
20	21
22	23

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).

Column	Type
ID	*Integer*
Name	*String*

Students

Column	Type
ID	*Integer*
Friend_ID	*Integer*

Friends

Column	Type
ID	*Integer*
Salary	*Float*

Packages

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

ID	Friend_ID
1	2
2	3
3	4
4	1

Friends

ID	Name
1	Ashley
2	Samantha
3	Julia
4	Scarlet

Students

ID	Salary
1	15.20
2	10.06
3	11.55
4	12.12

Packages

Sample Output
Samantha
Julia
Scarlet

Explanation
See the following table:

ID	1	2	3	4
Name	Ashley	Samantha	Julia	Scarlet
Salary	15.20	10.06	11.55	12.12
Friend ID	2	3	4	1
Friend Salary	10.06	11.55	12.12	15.20

```
select  b.name,b.salary,students.name as friend_name,packages.salary as friend_salary
from (
select friend_id,name,salary as salary from friends
join students on students.id=friends.id
left join packages on packages.id=friends.id
)b
left join students on students.id=friend_id
left join packages on packages.id=friend_id
where packages.salary>b.salary
order by 4
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

name	salary	friend_name	friend_salary
SAMANTHA	10.06	JULIA	11.55
JULIA	11.55	SCARLET	12.12
SCARLET	12.12	ASHLEY	15.2

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.

Column	Type
hacker_id	Integer
name	String

- Difficulty: The difficult_level is the level of difficulty of the challenge, and score is the score of the challenge for the difficulty level.

Column	Type
difficulty_level	Integer
score	Integer

- 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.

Column	Type
challenge_id	Integer
hacker_id	Integer
difficulty_level	Integer

- 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.

Column	Type
submission_id	Integer
hacker_id	Integer
challenge_id	Integer
score	Integer

Hackers Table:

hacker_id	name
5580	Rose
8439	Angela
27205	Frank
52243	Patrick
52348	Lisa
57645	Kimberly
77726	Bonnie
83082	Michael
86870	Todd
90411	Joe

Difficulty Table:

difficulty_level	score
1	20
2	30
3	40
4	60
5	80
6	100
7	120

Challenges Table:

challenge_id	hacker_id	difficulty_level
4810	77726	4
21089	27205	1
36566	5580	7
66730	52243	6
71055	52243	2

:

submission_id	hacker_id	challenge_id	score
68628	77726	36566	30
65300	77726	21089	10
40326	52243	36566	77
8941	27205	4810	4
83554	77726	66730	30
43353	52243	66730	0
55385	52348	71055	20
39784	27205	71055	23
94613	86870	71055	30
45788	52348	36566	0
93058	86870	36566	30
7344	8439	66730	92
2721	8439	4810	36
523	5580	71055	4
49105	52348	66730	0
55877	57645	66730	80
38355	27205	66730	35
3924	8439	36566	80
97397	90411	66730	100
84162	83082	4810	40
97431	90411	71055	30

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.

```sql
select hacker_id,name,count(*) as instanceofwin
from (
select
hackers.hacker_id,name,challenges.challenge_id,sum(coalesce(difficulty.scor
e,0)) as from_score,
sum(coalesce(submissions.score,0)) as scored_in_challenge
 from hackers
left join submissions on submissions.hacker_id=hackers.hacker_id
left join challenges on challenges.challenge_id=submissions.challenge_id
left join difficulty on
difficulty.difficulty_level=challenges.difficulty_level
where challenges.challenge_id is not null
group by 1,2,3
having from_score=scored_in_challenge
)b
group by 1,2
having count(*)>1
```

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

Column	Type
Task_ID	Integer
Start_Date	Date
End_Date	Date

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

Task_ID	Start_Date	End_Date
1	2015-10-01	2015-10-02
2	2015-10-02	2015-10-03
3	2015-10-03	2015-10-04
4	2015-10-13	2015-10-14
5	2015-10-14	2015-10-15
6	2015-10-28	2015-10-29
7	2015-10-30	2015-10-31

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

```
wITH project_start AS
(
SELECT start_date, ROW_NUMBER() OVER() AS ps_rownum
FROm projects WHERE start_date not in (
                                                                SELECT
                                                                            end_date
                                                                            FROM
                                                                            projects
                                                                )
                    ),
    project_end AS
                  (
                          SELECT
```

```
                                end_date,
                                ROW_NUMBER() OVER() AS pe_rownum
                        FROM
                                projects
                        WHERE
                                END_date not in (
                                                SELECT
                                                        start_date
                                                FROM
                                                        projects
                                            )
                )

SELECT
        project_start.start_date,
        project_end.end_date
FROM
        project_start
INNER JOIN
        project_end
on
        project_end.pe_rownum = project_start.ps_rownum
ORDER BY
        DATEDIFF(project_start.start_date, project_end.end_date) desc,
        project_start.start_date;
```

```
Approach 2


WITH temp_project AS (
                    SELECT
                            temp.start_date,
                            temp.end_date,
                            SUM(
                                CASE
```

```
                                    WHEN previous_end_date IS NULL THEN 0
                                    WHEN DAY(end_date) -
DAY(previous_end_date) = 1 THEN 0
                                    ELSE 1
                                END
                                ) OVER(ORDER BY start_date) AS project_num
                    FROM (
                            SELECT
                                    start_date,
                                    end_date,
                                    LAG(end_date) OVER (ORDER BY
start_date) AS previous_end_date
                            FROM
                                    projects
                    ) temp
)

SELECT
        MIN(start_date) AS project_start_date,
        MAX(end_date) as project_end_date
FROM
        temp_project
GROUP BY
        project_num
ORDER BY
        DAY(MAX(end_date))-DAY(MIN(start_date));
```

Approach 3

```
WITH temp_project AS (
                        SELECT
                                temp.start_date,
                                temp.end_date,
                                SUM(
                                    CASE
                                    WHEN previous_end_date IS NULL THEN 0
```

```
                                    WHEN DAY(end_date) -
DAY(previous_end_date) = 1 THEN 0
                                    ELSE 1
                                    END
                                    )  over(order by start_date range
between unbounded preceding and current row) AS project_num
                        FROM (
                                SELECT
                                        start_date,
                                        end_date,
                                        LAG(end_date) OVER (ORDER BY
start_date) AS previous_end_date
                                FROM
                                        projects
                ) temp
            )

SELECT
        MIN(start_date) AS project_start_date,
        MAX(end_date) AS project_end_date
FROM
        temp_project
GROUP BY
        project_num
ORDER BY
        DAY(MAX(end_date))-DAY(MIN(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

```
select distinct user_id from (
select *,count(*) over (partition by user_id,check_flag) as count_instance from (
select *,date_sub(transaction_date,interval row_num day) as check_flag
from (
select *,row_number() over (partition by user_id) as row_num
 from transactions
)b
)b
)b
where count_instance>=3
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

user_id
2

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

```sql
with temp as (
select distinct
payer_id,recipient_id
from payments
)
select count(*) as cases
from temp
left join temp as a on  temp.recipient_id=a.payer_id and temp.payer_id=a.recipient_id
and temp.payer_id>temp.recipient_id
where a.payer_id is not null
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

	cases
▶	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

```
select count(*) as users_count
from (
select *,row_number() over (partition by user_id order by transaction_date asc) as trans_row
from user_transactions
)b
where trans_row=1 and spend>=50
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

users_count
1

Q150.

Assume you are given the table below containing measurement values obtained from a senso 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 particular day, in two different columns.

Note that the 1st, 3rd, 5th measurements within a day are considered odd-numbered measurer 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

```
select date(measurment_time), round(sum(case when row_nums%2=0 then measurment_value end),1) as even_sum,
round(sum(case when row_nums%2!=0 then measurment_value end),1) as odd_sum from (
select *,row_number() over (partition by date(measurment_time) order by measurment_time asc) as row_nums
from measurments
)b
group by 1
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

date(measurment_time)	even_sum	odd_sum
2022-07-10	1662.7	2355.8
2022-07-11	1234.1	1124.5

Answer 151 SAME AS Q.147

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

```
select sum(pairs) from (
select gid,sum(distinct instances) as pairs from (
select max(rental_id) as gid,count_num,amenity,count(*) div 2 as instances
from (
select *,count(*) over (partition by rental_id) as count_num from
rental_amenities
)b
group by 2,3
having instances>=1
)b
group by 1
)b
```

Q153.

Google marketing managers are analysing the performance of various advertising accounts over the last month. They need your help to gather the relevant data.
Write a query to calculate the return on ad spend (ROAS) for each advertiser across all ad campaigns. Round your answer to 2 decimal places, and order your output by the advertiser_id.

Level - Medium
Hint: ROAS = Ad Revenue / Ad Spend

ad_campaigns Table:

Column Name	Type
campaign_id	integer
spend	integer
revenue	float
advertiser_id	integer

ad_campaigns Example Input:

campaign_id	spend	revenue	advertiser_id
1	5000	7500	3
2	1000	900	1
3	3000	12000	2
4	500	2000	4
5	100	400	4

Example Output:

advertiser_id	ROAS
1	0.9
2	4
3	1.5
4	4

```
select advertiser_id,sum(revenue)/sum(spend) as roas
 from ad_campaigns
group by 1
order by 1
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

advertiser_id	sum(revenue)/sum(spend)
1	0.9
2	4
3	1.5
4	1

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

```sql
select * from (
select *,case when salary<0.5*avg_title_salary then 'Underpaid'
when salary >2*avg_title_salary then 'Overpaid' end as remark from (
select *,avg(salary) over (partition by title) avg_title_salary
 from employee_pay
)b )b
where remark is not null
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

employee_id	salary	title	avg_title_salary	remark
104	30000	DATA ANALYST	75000.0000	Underpaid
108	310000	DATA SCIENTIST	152500.0000	Overpaid

Q.155 SAME AS 148

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

```sql
select distinct user_id from (
select user_id,product_id,count(distinct date(purchase_date)) from purchases
group by 1,2
having count(distinct date(purchase_date))>1)b
-- select *,dense_rank() over (partition by user_id,product_id order by date(purchase_date)) as purchase_instance
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

user_id
536

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

```
3729 • select *,sum(net_bal) over (order by trans_date) as cumulative_sum from (
3730   select date(transaction_date) as trans_date,sum(case when type='DEPOSIT' then amount else -1*amount end) as net_bal
3731    from transactions
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

trans_date	net_bal	cumulative_sum
2022-07-08	106.66000175476074	106.66000175476074
2022-07-10	98.5	205.16000175476074

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

```sql
select * from (
select *,row_number() over (partition by category order by total_spend desc) as ranking from (
select category,product,round(sum(spend),1) as total_spend
from product_spend
where extract(year from transaction_date)=2022
group by 1,2)b
)b
where ranking<3
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

category	product	total_spend	ranking
APPLIANCE	REFRIGERATOR	300	1
APPLIANCE	WASHING MACHINE	219.8	2
ELECTRONICS	VACUUM	341	1
ELECTRONICS	WIRELESS HEADSET	249.9	2

Result 456

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%.

```
WITH temp_churn_rate AS (
                            SELECT
                                    user_id,
                                    signup_date,
                                    last_login,
                                    DATEDIFF(last_login, signup_date) diff,
                                    EXTRACT(WEEK FROM signup_date) AS
week_no,
                                    DENSE_RANK() OVER(ORDER BY EXTRACT(WEEK
FROM signup_date)) ranking
                            FROM
                                    users
                            WHERE
```

```
                                        EXTRACT(MONTH FROM signup_date) = 6
                                        AND
                                        EXTRACT(YEAR FROM signup_date) = 2022
        ),
    temp_churn_rate2 AS (
                                SELECT
                                        ranking,
                                        COUNT(ranking) AS total_users,
                                        COUNT(
                                        CASE
                                            WHEN diff <= 28 THEN 1
                                            END
                                            ) AS total_churns
                                FROM
                                        temp_churn_rate
                                GROUP BY
                                        ranking
        )

SELECT
        ranking AS week,
        ROUND((total_churns/total_users) * 100 ,2) AS churn_rate
FROM
        temp_churn_rate2
ORDER BY
        ranking;
```