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

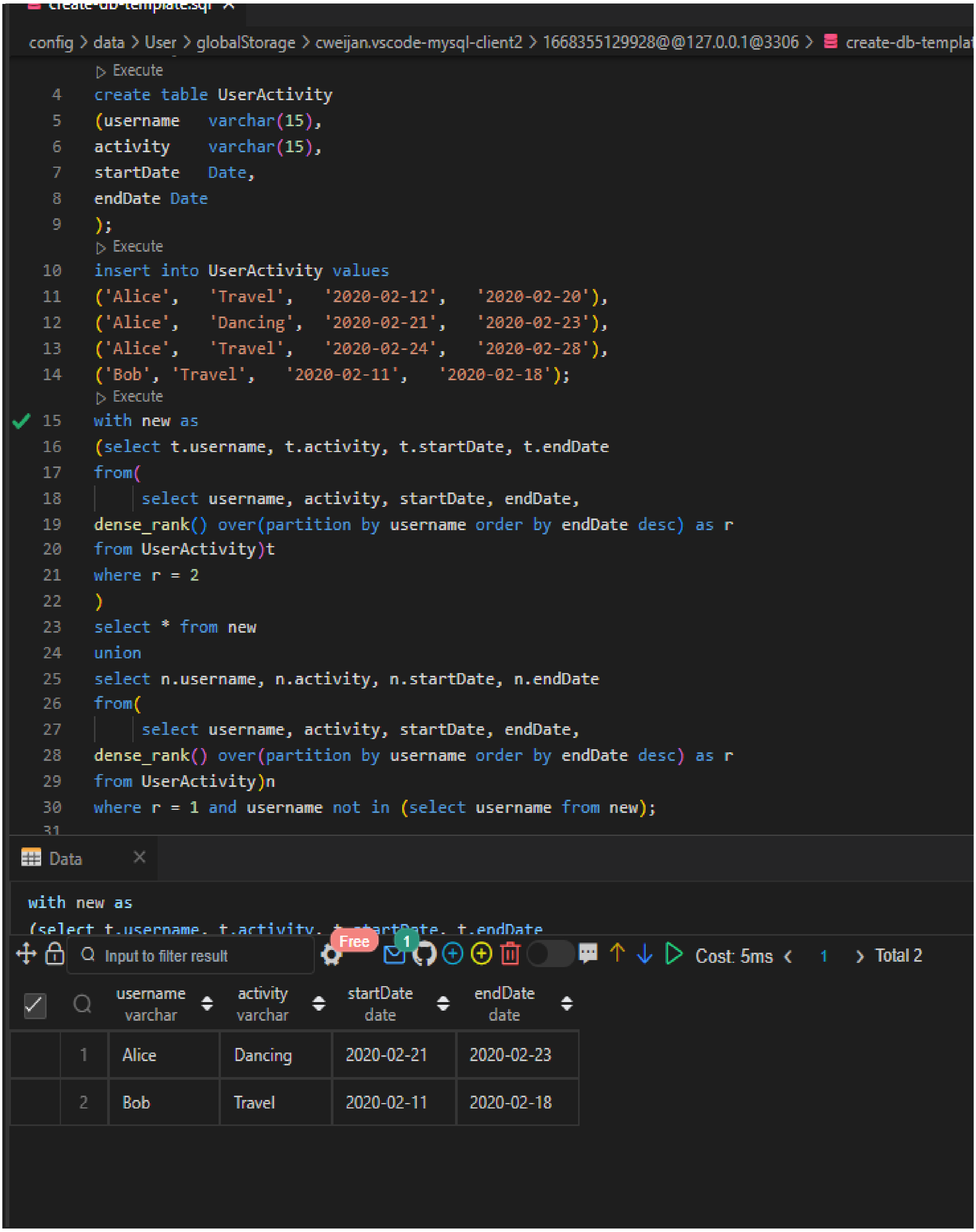
Output:

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.

Solution:

```sql
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);
```


create-db-template.sql ×
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-templat
▷ Execute
▷ Execute
▷ Execute
Data ×
Input to filter result
Free
1
Cost: 5ms
1
Total 2

```
create table UserActivity
(username   varchar(15),
activity    varchar(15),
startDate   Date,
endDate Date
);
insert into UserActivity values
('Alice',   'Travel',   '2020-02-12',   '2020-02-20'),
('Alice',   'Dancing',  '2020-02-21',   '2020-02-23'),
('Alice',   'Travel',   '2020-02-24',   '2020-02-28'),
('Bob', 'Travel',   '2020-02-11',   '2020-02-18');
with 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);
```

```
with new as
(select t.username, t.activity, t.startDate, t.endDate
```

	username varchar	activity varchar	startDate date	endDate date
1	Alice	Dancing	2020-02-21	2020-02-23
2	Bob	Travel	2020-02-11	2020-02-18

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:

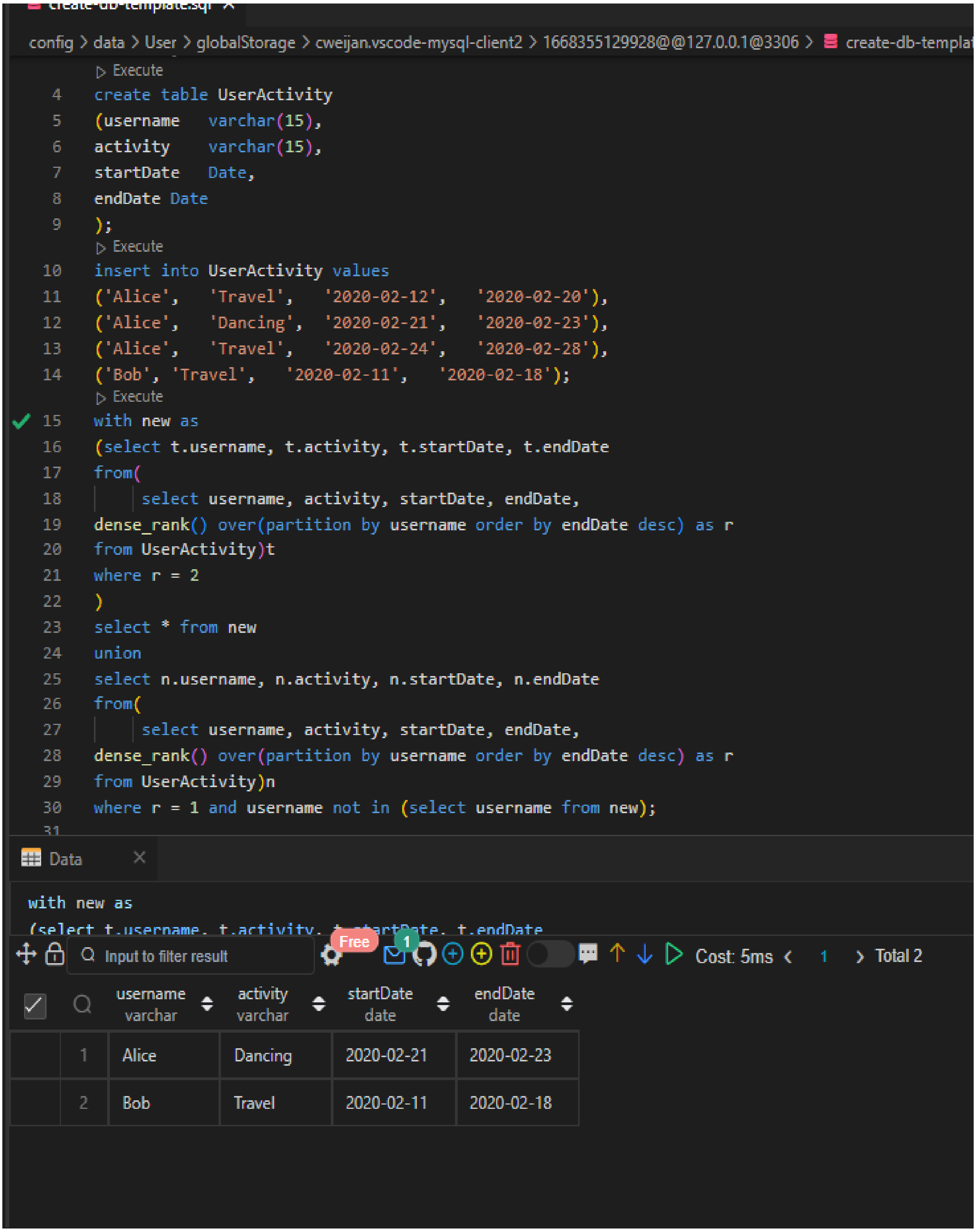
username	activity	startDate	endDate
Alice	Travel	2020-02-12	2020-02-20
Alice	Dancing	2020-02-21	2020-02-23
Alice	Travel	2020-02-24	2020-02-28
Bob	Travel	2020-02-11	2020-02-18

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.

```
Solution:
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);
```


create-db-template.sql ×
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-templat
▷ Execute
▷ Execute
▷ Execute
Data ×
Free
1
Input to filter result
Cost: 5ms
1
Total 2

```sql
create table UserActivity
(username   varchar(15),
activity    varchar(15),
startDate   Date,
endDate Date
);
insert into UserActivity values
('Alice',   'Travel',    '2020-02-12',    '2020-02-20'),
('Alice',   'Dancing',   '2020-02-21',    '2020-02-23'),
('Alice',   'Travel',    '2020-02-24',    '2020-02-28'),
('Bob', 'Travel',    '2020-02-11',    '2020-02-18');
with 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);
```

```sql
with new as
(select t.username, t.activity, t.startDate, t.endDate
```

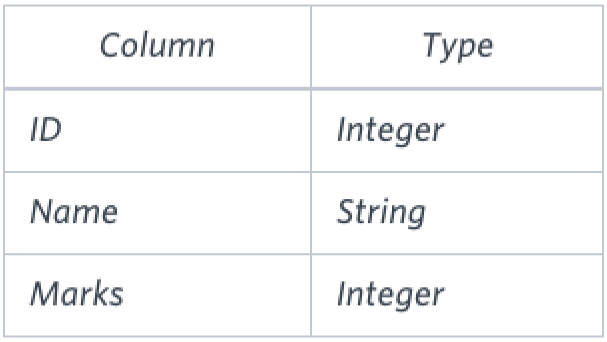
☑		username varchar	activity varchar	startDate date	endDate date
	1	Alice	Dancing	2020-02-21	2020-02-23
	2	Bob	Travel	2020-02-11	2020-02-18

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

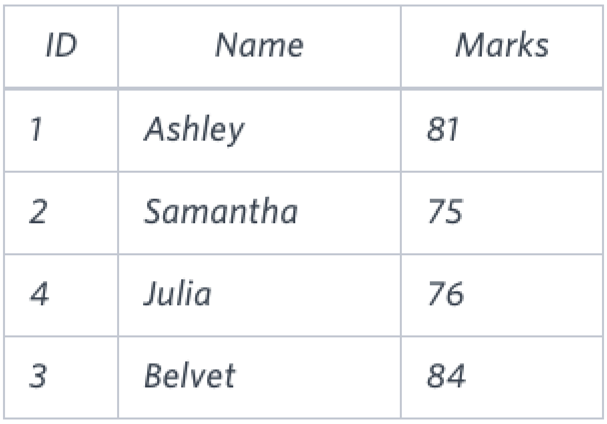
The STUDENTS table is described as follows:



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

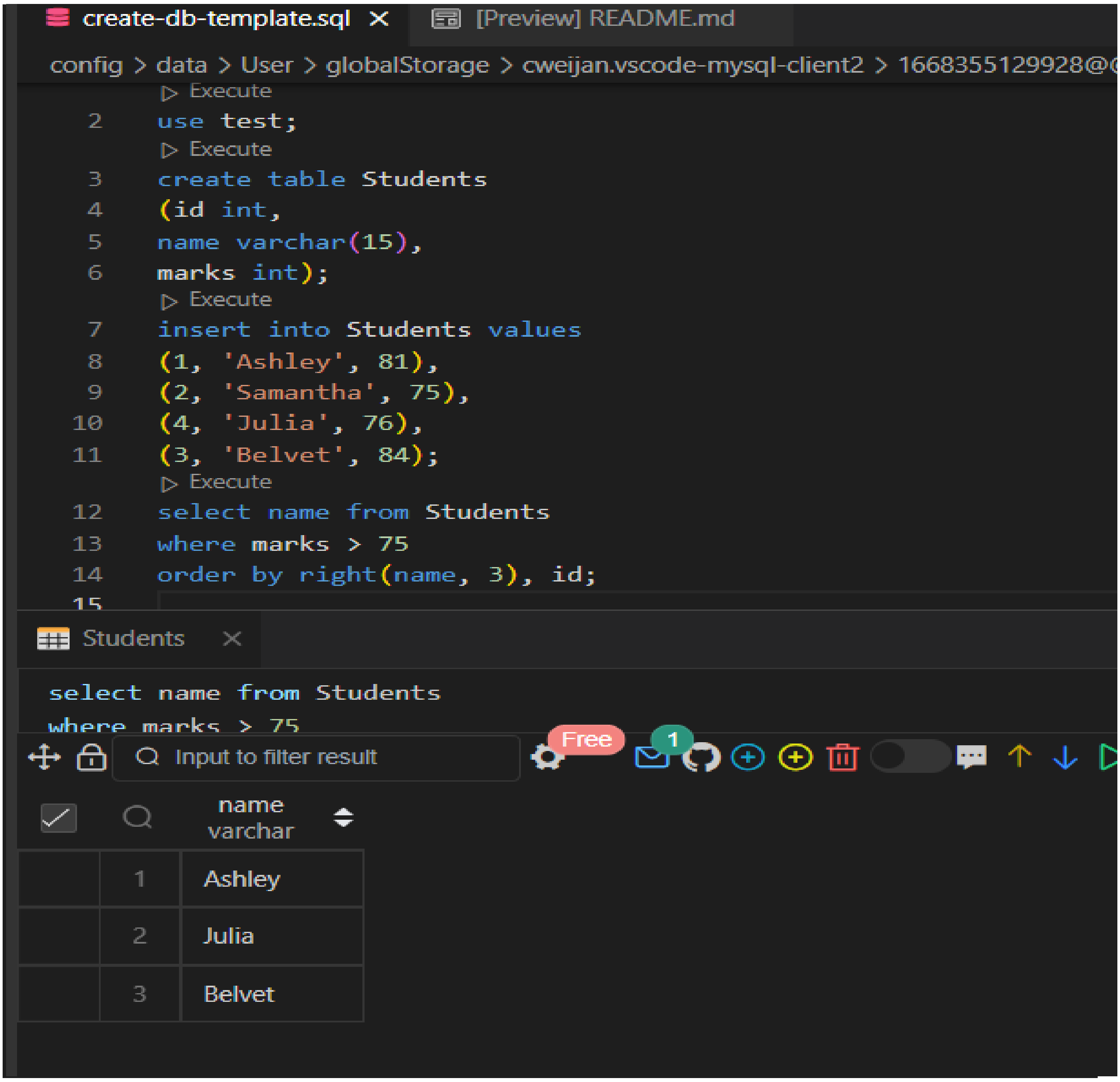
Julia

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'.

```
Solution:
select name from Students
where marks > 75
order by right(name, 3), id;
```


create-db-template.sql
[Preview] README.md
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@
▷ Execute

```
use test;
create table Students
(id int,
name varchar(15),
marks int);
insert into Students values
(1, 'Ashley', 81),
(2, 'Samantha', 75),
(4, 'Julia', 76),
(3, 'Belvet', 84);
select name from Students
where marks > 75
order by right(name, 3), id;
```

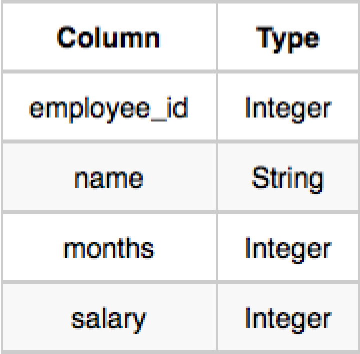
```
select name from Students
where marks > 75
```

	name varchar
1	Ashley
2	Julia
3	Belvet

Q104.

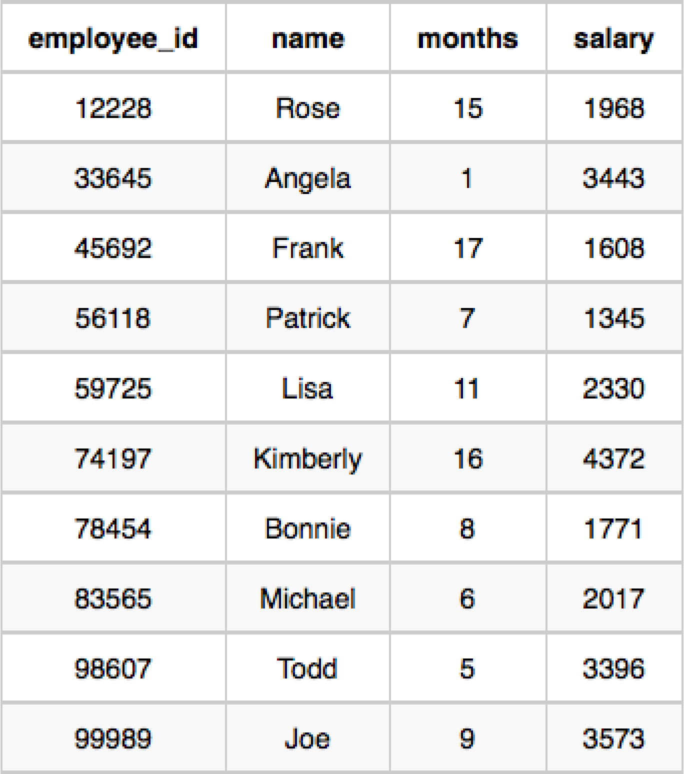
Write a query that prints a list of employee names (i.e.: the name attribute) for employees in Employee having a salary greater than $2000 per month who have been employees for less than 10 months. Sort your result by ascending employee_id.

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

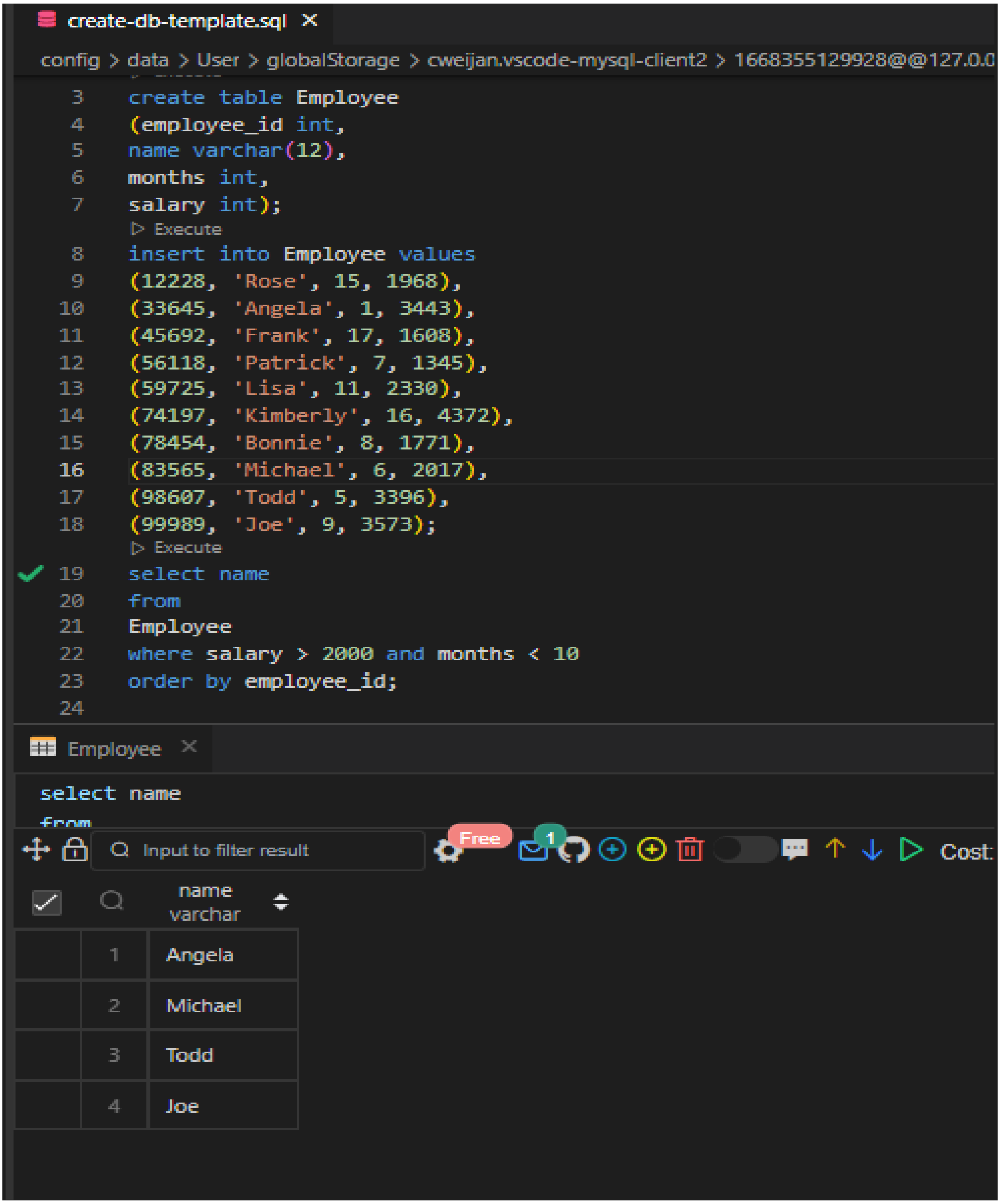
where employee_id is an employee's ID number, name is their name, months is the total number of months they've been working for the company, and salary is the their monthly salary.
Sample Input



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.

```
Solution:
select name
from
Employee
where salary > 2000 and months < 10
order by employee_id;
```


create-db-template.sql
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0
▷ Execute
Employee
Input to filter result
Free
Cost:

```sql
create table Employee
(employee_id int,
name varchar(12),
months int,
salary int);
insert into Employee values
(12228, 'Rose', 15, 1968),
(33645, 'Angela', 1, 3443),
(45692, 'Frank', 17, 1608),
(56118, 'Patrick', 7, 1345),
(59725, 'Lisa', 11, 2330),
(74197, 'Kimberly', 16, 4372),
(78454, 'Bonnie', 8, 1771),
(83565, 'Michael', 6, 2017),
(98607, 'Todd', 5, 3396),
(99989, 'Joe', 9, 3573);
select name
from
Employee
where salary > 2000 and months < 10
order by employee_id;
```

```sql
select name
from
```

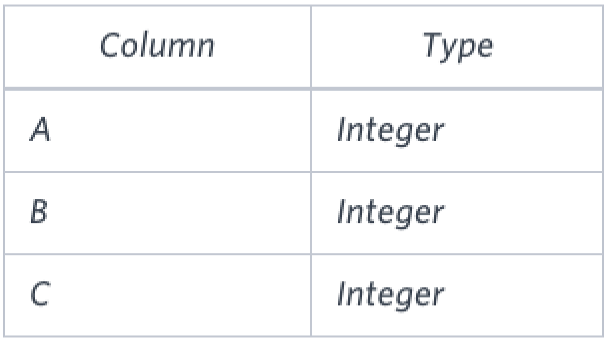
	name varchar
1	Angela
2	Michael
3	Todd
4	Joe

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:

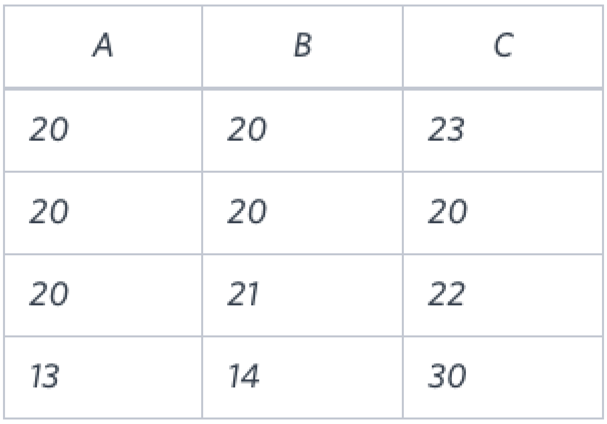
- Equilateral: It's a triangle with sides of equal length.
- Isosceles: It's a triangle with sides of equal length.
- Scalene: It's a triangle with sides of differing lengths.
- Not A Triangle: The given values of A, B, and C don't form a triangle.

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



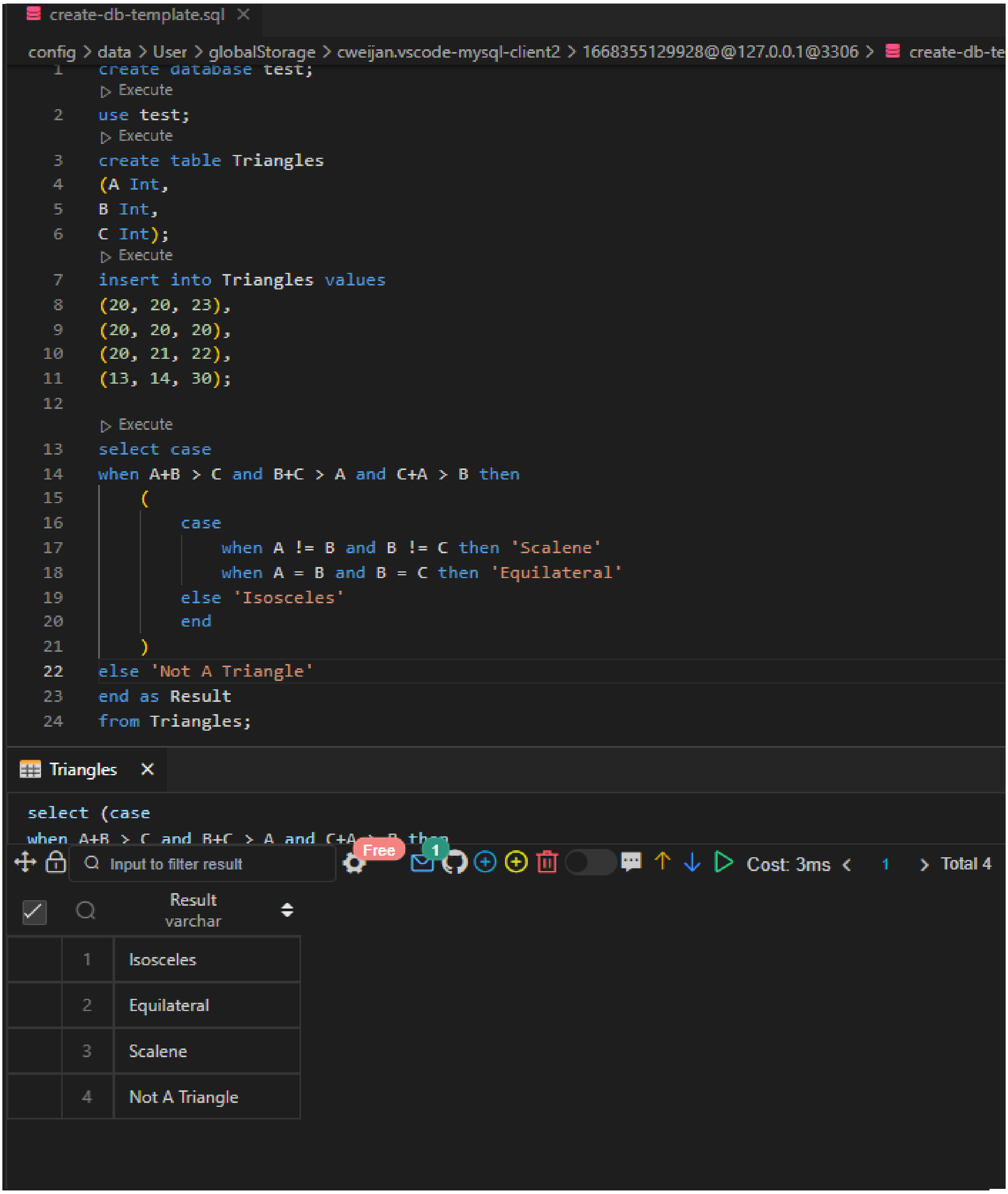
A	B	C
20	20	23
20	20	20
20	21	22
13	14	30

Sample Output
Isosceles
Equilateral
Scalene
Not A Triangle

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

```
Solution:
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;
```


create-db-template.sql
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-te
Triangles
Input to filter result
Free
1
Cost: 3ms
1
Total 4

```
create database test;
use test;
create table Triangles
(A Int,
B Int,
C Int);
insert into Triangles values
(20, 20, 23),
(20, 20, 20),
(20, 21, 22),
(13, 14, 30);

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;
```

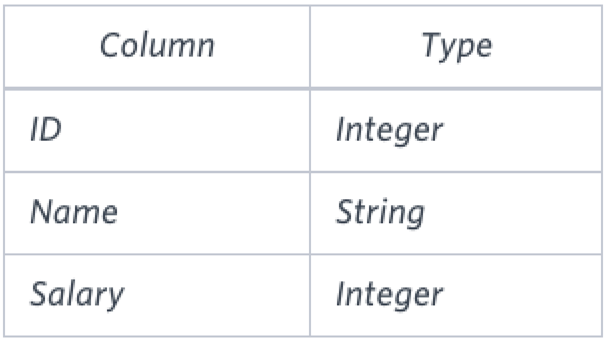
```
select (case
when A+B > C and B+C > A and C+A > B then
```

		Result varchar
	1	Isosceles
	2	Equilateral
	3	Scalene
	4	Not A Triangle

Q106.

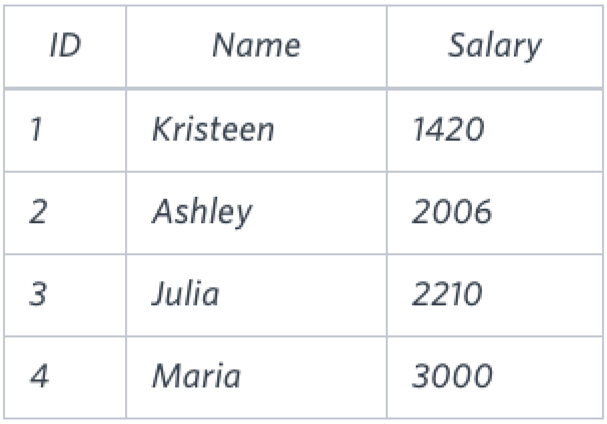
Samantha was tasked with calculating the average monthly salaries for all employees in the EMPLOYEES table, but did not realise her keyboard's 0 key was broken until after completing the calculation. She wants your help finding the difference between her miscalculation (using salaries with any zeros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

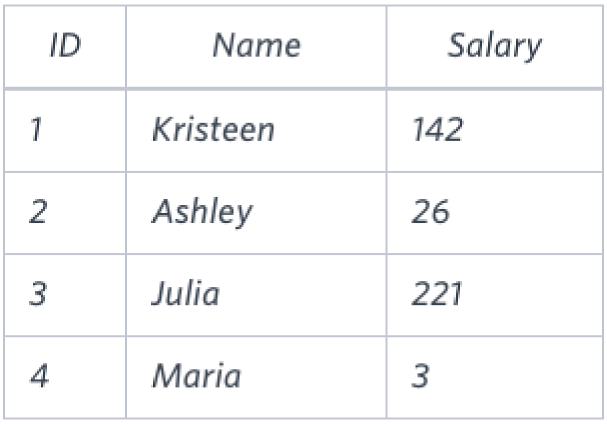
Note: Salary is per month.
Constraints
1000<salary < 10^5
Sample Input



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

Sample Output
2061

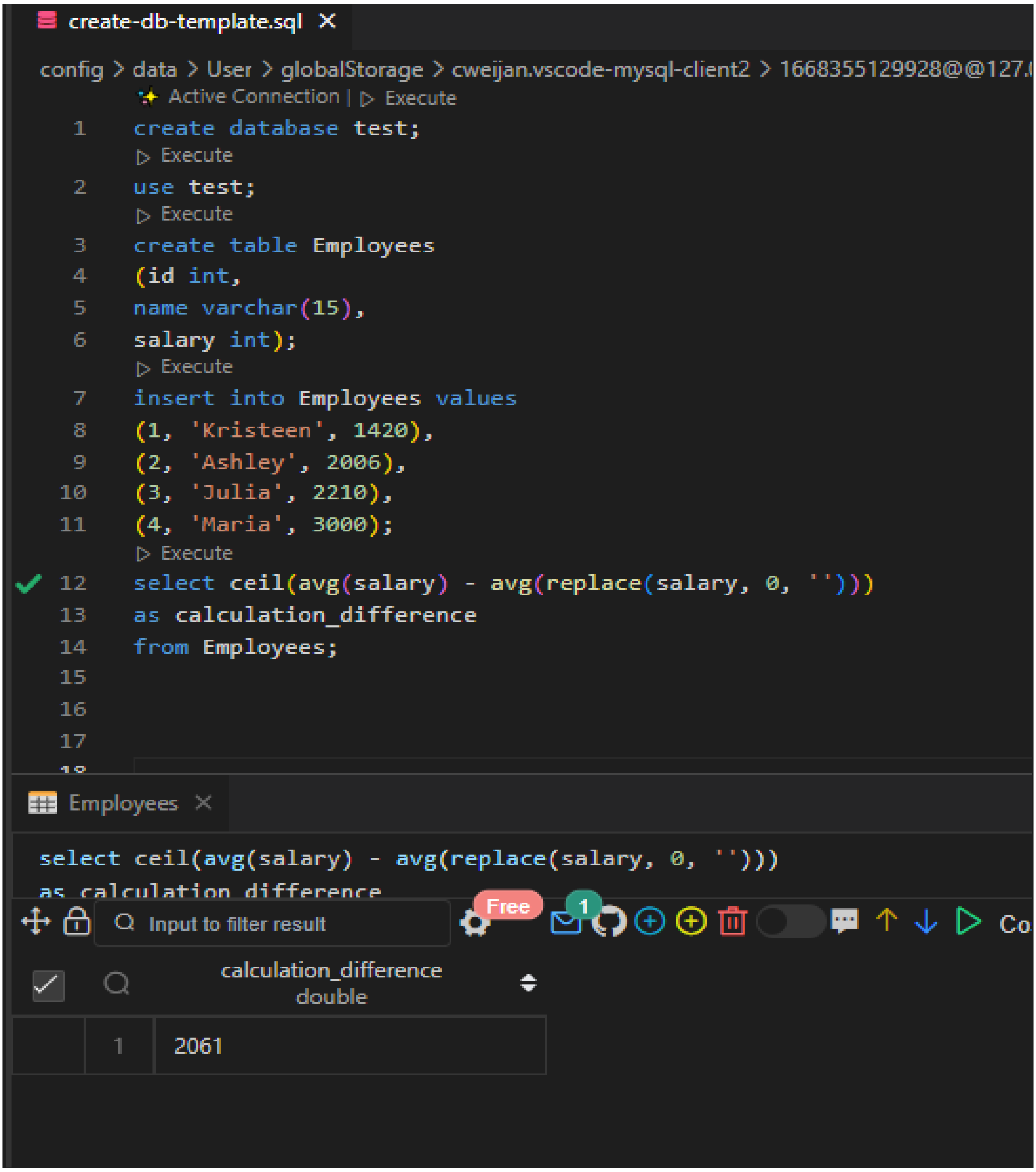
Explanation
The table below shows the salaries without zeros as they were entered by Samantha:



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.

```
Solution:
select ceil(avg(salary) - avg(replace(salary, 0, '')))
as calculation_difference
from Employees;
```


create-db-template.sql ✕
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.(
Active Connection | ▷ Execute
▷ Execute
Employees ✕
Input to filter result
Free
1
Co

```sql
create database test;
use test;
create table Employees
(id int,
name varchar(15),
salary int);
insert into Employees values
(1, 'Kristeen', 1420),
(2, 'Ashley', 2006),
(3, 'Julia', 2210),
(4, 'Maria', 3000);
select ceil(avg(salary) - avg(replace(salary, 0, '')))
as calculation_difference
from Employees;
```

```
select ceil(avg(salary) - avg(replace(salary, 0, '')))
as calculation_difference
```

```
     calculation_difference
     double
1    2061
```

Q107.

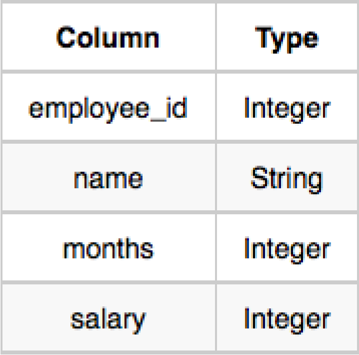
We define an employee's total earnings to be their monthly salary * months worked, and the maximum total earnings to be the maximum total earnings for any employee in the Employee table. Write a query to find the maximum total earnings for all employees as well as the total number of employees who have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

Level - Easy

Hint - Use Aggregation functions

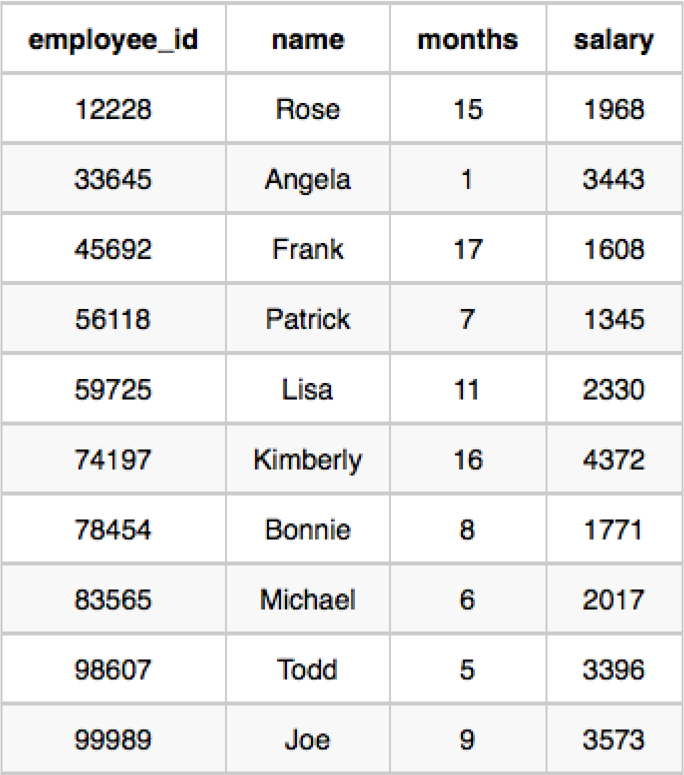
Input Format

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

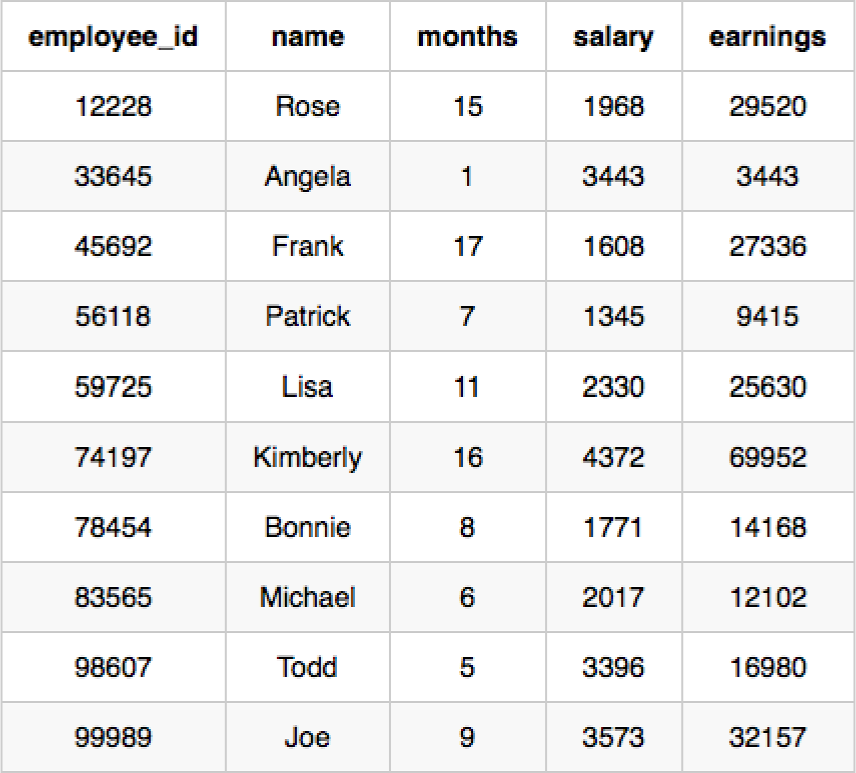


employee_id	name	months	salary
12228	Rose	15	1968
33645	Angela	1	3443
45692	Frank	17	1608
56118	Patrick	7	1345
59725	Lisa	11	2330
74197	Kimberly	16	4372
78454	Bonnie	8	1771
83565	Michael	6	2017
98607	Todd	5	3396
99989	Joe	9	3573

Sample Output
69952 1

Explanation:

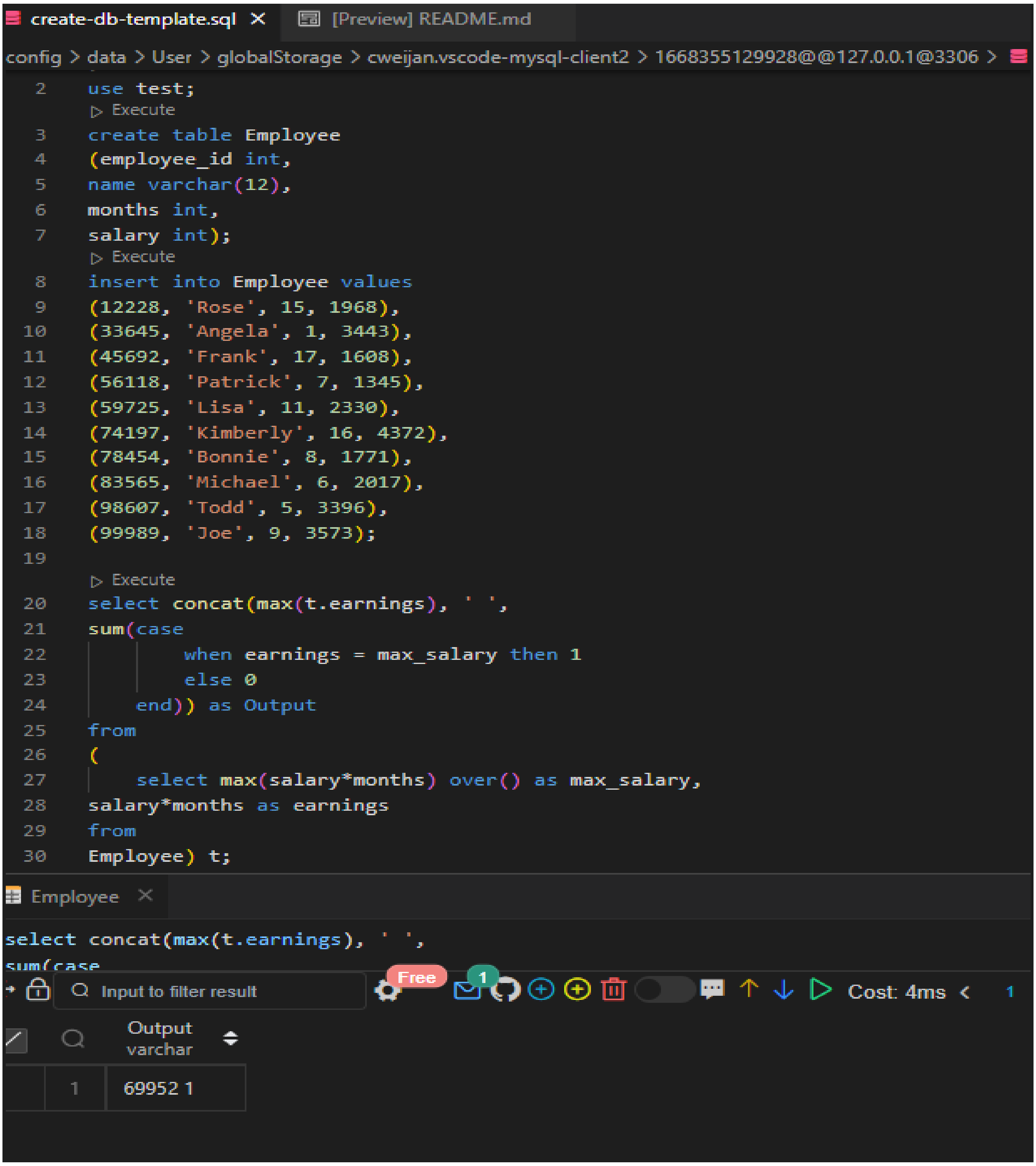
The table and earnings data is depicted in the following diagram:



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 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;
```


create-db-template.sql
[Preview] README.md
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 >
Employee
Input to filter result
Free
1
Cost: 4ms
1

```sql
use test;
▷ Execute
create table Employee
(employee_id int,
name varchar(12),
months int,
salary int);
▷ Execute
insert into Employee values
(12228, 'Rose', 15, 1968),
(33645, 'Angela', 1, 3443),
(45692, 'Frank', 17, 1608),
(56118, 'Patrick', 7, 1345),
(59725, 'Lisa', 11, 2330),
(74197, 'Kimberly', 16, 4372),
(78454, 'Bonnie', 8, 1771),
(83565, 'Michael', 6, 2017),
(98607, 'Todd', 5, 3396),
(99989, 'Joe', 9, 3573);

▷ Execute
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;
```

```
select concat(max(t.earnings), ' ',
sum(case
```

```
     Output
     varchar
1    69952 1
```

Q108.

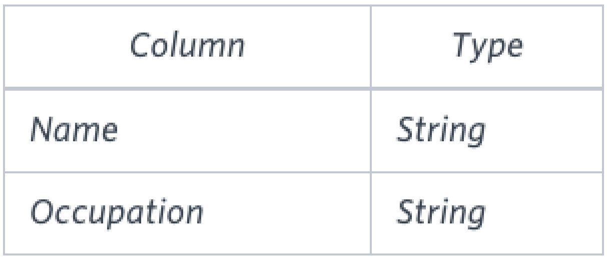
Generate the following two result sets:

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.

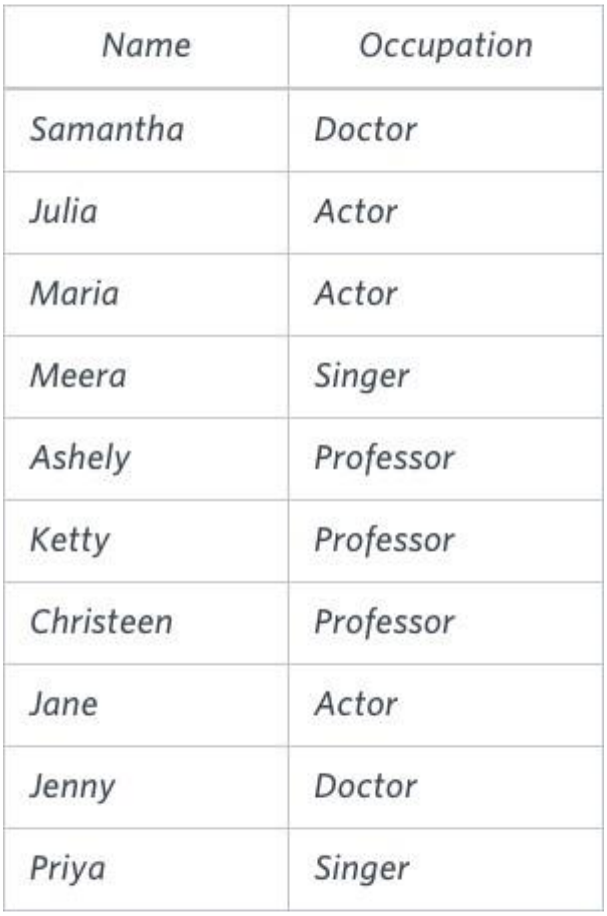
2. where [occupation_count] is the number of occurrences of an occupation in OCCUPATIONS and [occupation] is the lowercase occupation name. If more than one Occupation has the same [occupation_count], they should be ordered alphabetically.

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

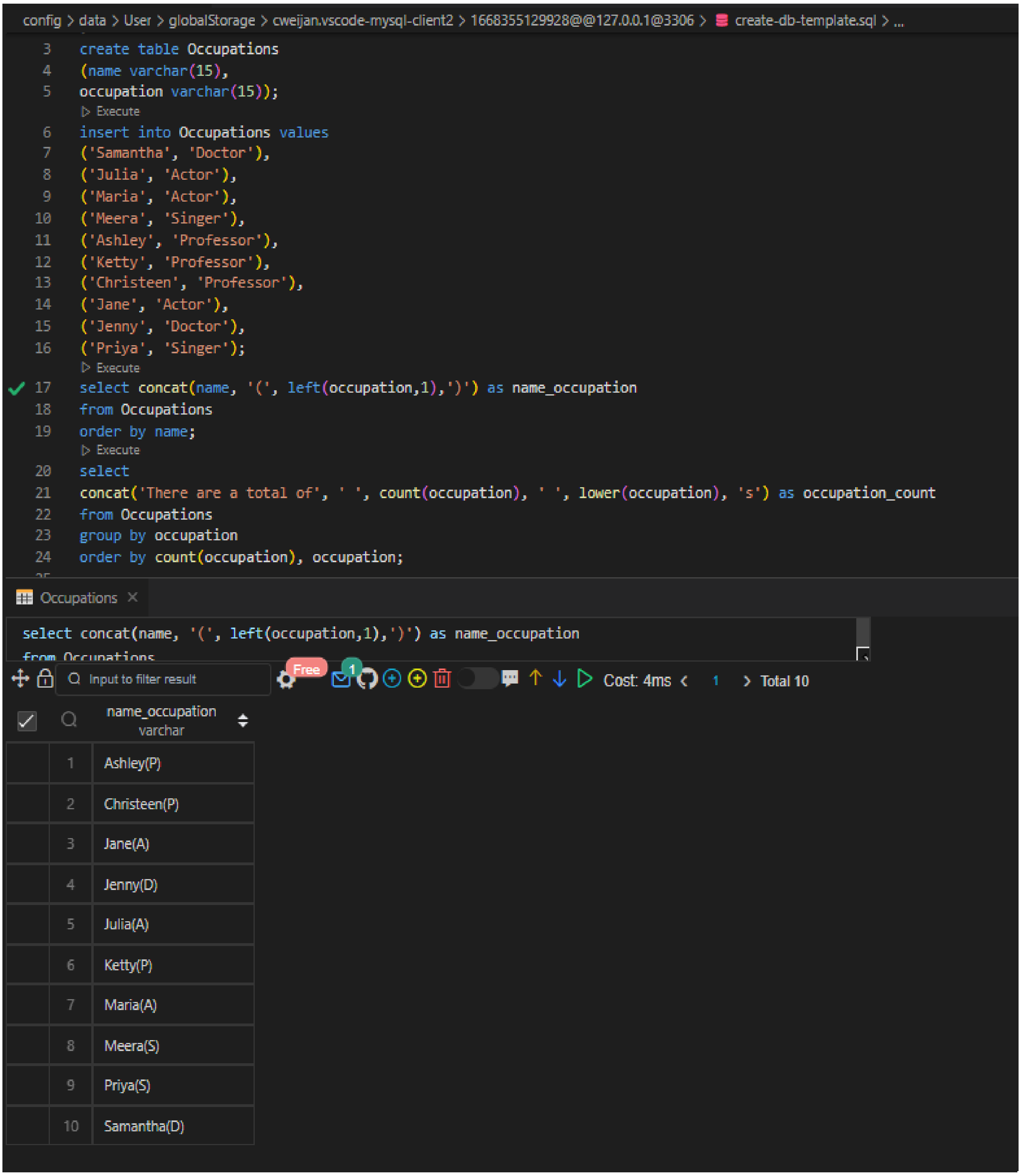
```
Ashely(P)
Christeen(P)
Jane(A)
Jenny(D)
Julia(A)
Ketty(P)
Maria(A)
Meera(S)
Priya(S)
Samantha(D)
There are a total of 2 doctors.
There are a total of 2 singers.
There are a total of 3 actors.
There are a total of 3 professors.
```

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).

```
Solution:
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;
```



config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-template.sql > ...

```
create table Occupations
(name varchar(15),
occupation varchar(15));
```

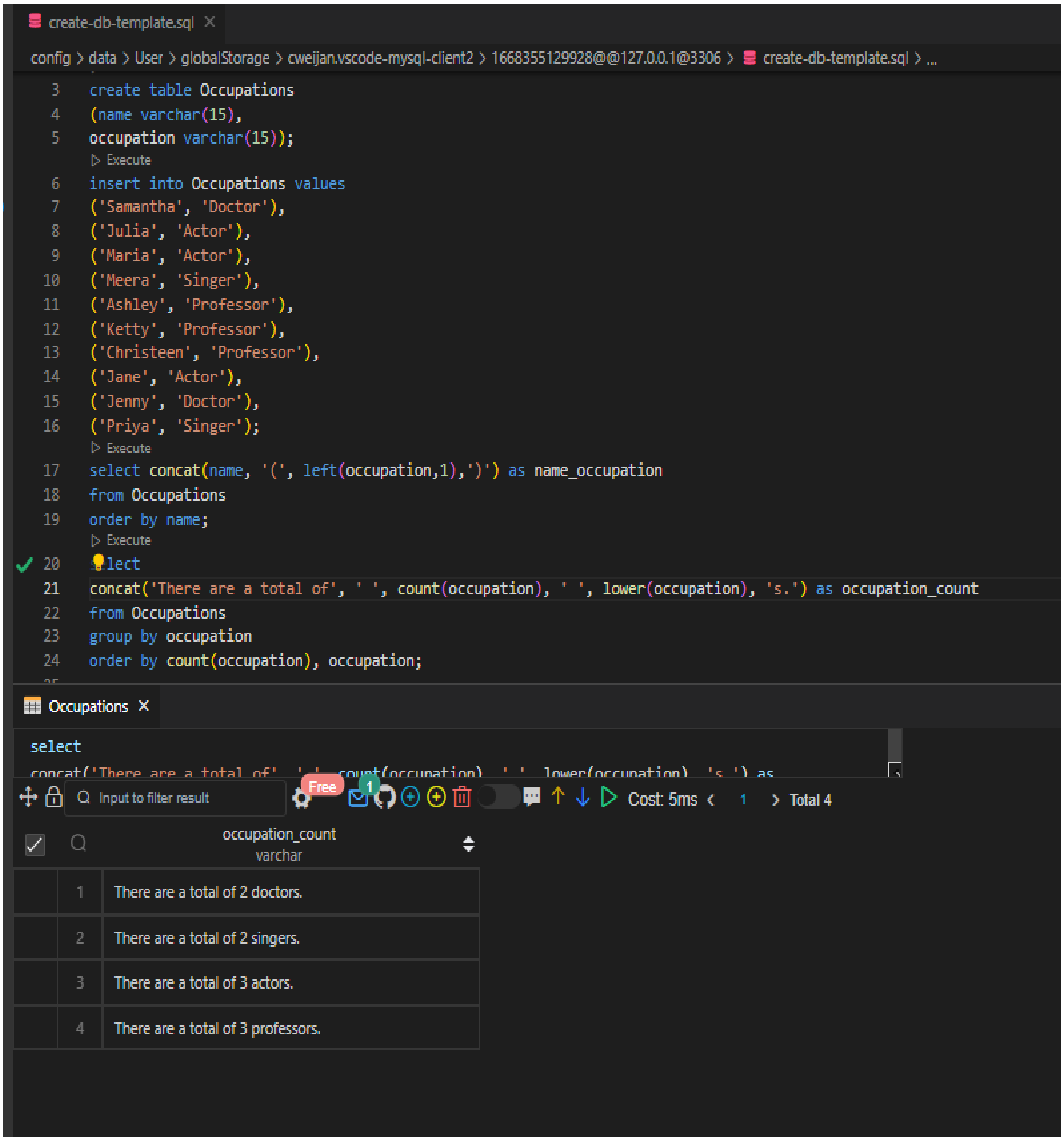
```
insert into Occupations values
('Samantha', 'Doctor'),
('Julia', 'Actor'),
('Maria', 'Actor'),
('Meera', 'Singer'),
('Ashley', 'Professor'),
('Ketty', 'Professor'),
('Christeen', 'Professor'),
('Jane', 'Actor'),
('Jenny', 'Doctor'),
('Priya', 'Singer');
```

```
select concat(name, '(', left(occupation,1),')') 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;
```

```
select concat(name, '(', left(occupation,1),')') as name_occupation
from Occupations
```

	name_occupation varchar
1	Ashley(P)
2	Christeen(P)
3	Jane(A)
4	Jenny(D)
5	Julia(A)
6	Ketty(P)
7	Maria(A)
8	Meera(S)
9	Priya(S)
10	Samantha(D)


create-db-template.sql ×
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-template.sql > ...
▷ Execute
Occupations ×
Input to filter result
Free
1
Cost: 5ms ‹ 1 › Total 4

```sql
create table Occupations
(name varchar(15),
occupation varchar(15));
insert into Occupations values
('Samantha', 'Doctor'),
('Julia', 'Actor'),
('Maria', 'Actor'),
('Meera', 'Singer'),
('Ashley', 'Professor'),
('Ketty', 'Professor'),
('Christeen', 'Professor'),
('Jane', 'Actor'),
('Jenny', 'Doctor'),
('Priya', 'Singer');
select concat(name, '(', left(occupation,1),')') as name_occupation
from Occupations
order by name;
💡lect
concat('There are a total of', ' ', count(occupation), ' ', lower(occupation), 's.') as occupation_count
from Occupations
group by occupation
order by count(occupation), occupation;
```

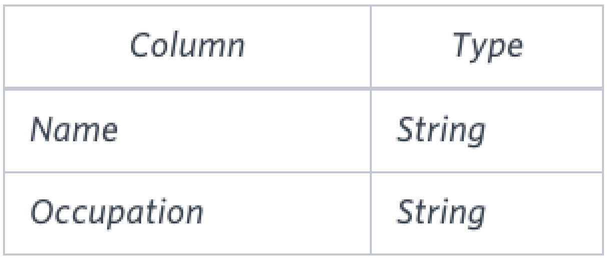
```sql
select
concat('There are a total of', ' ', count(occupation), ' ', lower(occupation), 's.') as
```

☑	occupation_count varchar
1	There are a total of 2 doctors.
2	There are a total of 2 singers.
3	There are a total of 3 actors.
4	There are a total of 3 professors.

Q109 .

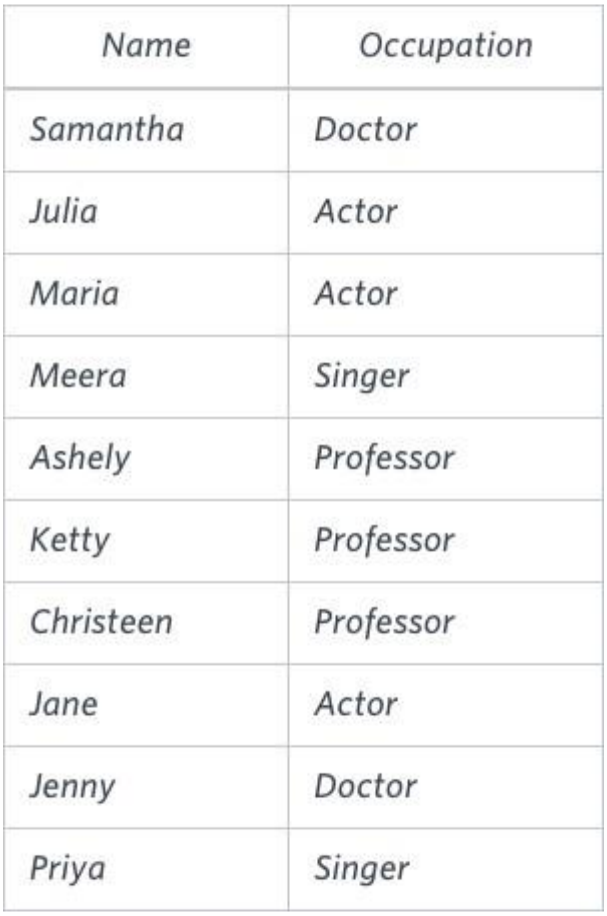
Pivot the Occupation column in OCCUPATIONS so that each Name is sorted alphabetically and displayed underneath its corresponding Occupation. The output column headers should be Doctor, Professor, Singer, and Actor, respectively.
Note: Print NULL when there are no more names corresponding to an occupation.

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



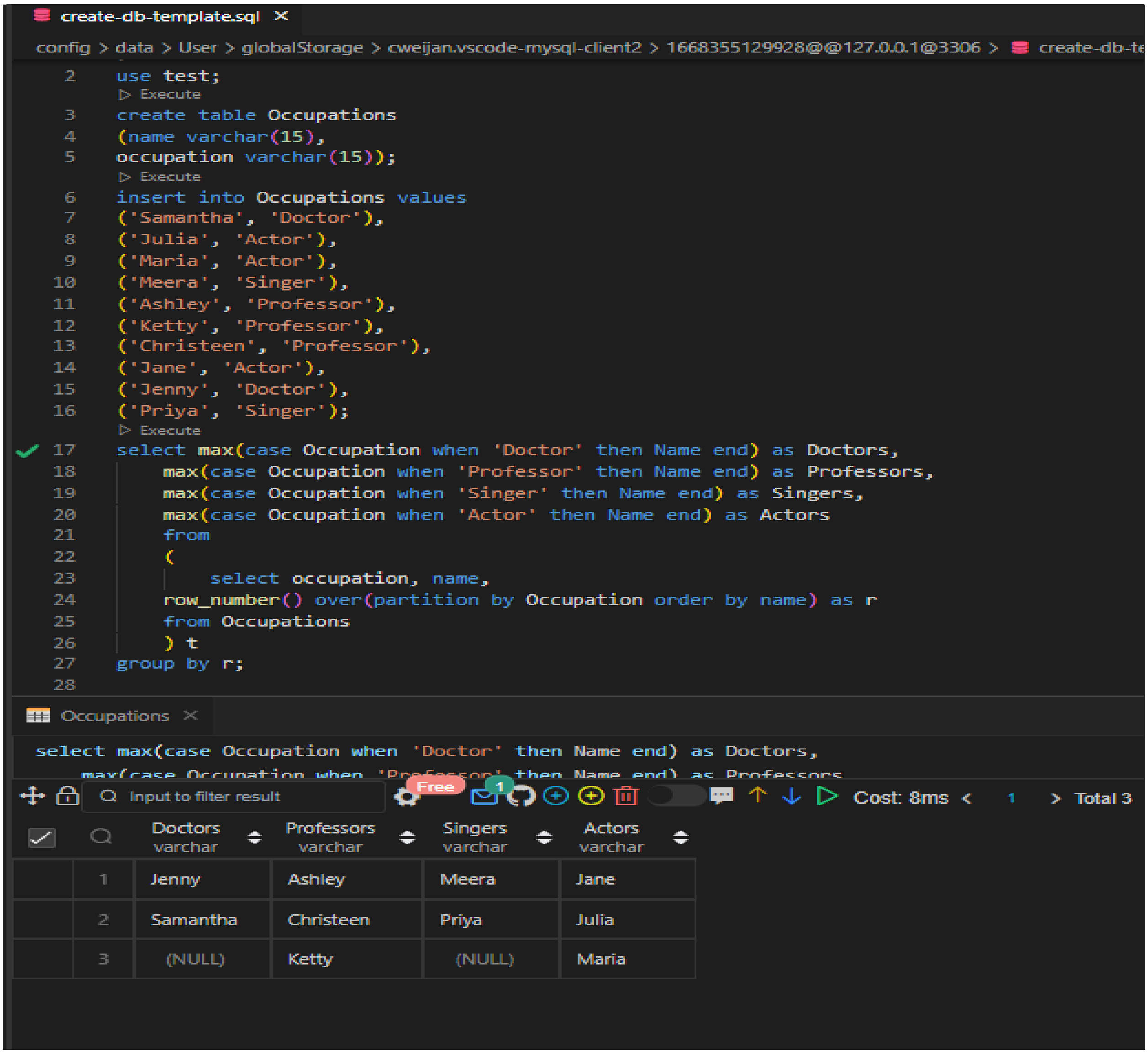
Name	Occupation
Samantha	Doctor
Julia	Actor
Maria	Actor
Meera	Singer
Ashely	Professor
Ketty	Professor
Christeen	Professor
Jane	Actor
Jenny	Doctor
Priya	Singer

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.

```
Solution:
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;
```


create-db-template.sql
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-te
Execute
Occupations
Input to filter result
Free
Cost: 8ms
Total 3

```sql
use test;
create table Occupations
(name varchar(15),
occupation varchar(15));
insert into Occupations values
('Samantha', 'Doctor'),
('Julia', 'Actor'),
('Maria', 'Actor'),
('Meera', 'Singer'),
('Ashley', 'Professor'),
('Ketty', 'Professor'),
('Christeen', 'Professor'),
('Jane', 'Actor'),
('Jenny', 'Doctor'),
('Priya', 'Singer');
select 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;
```

	Doctors varchar	Professors varchar	Singers varchar	Actors varchar
1	Jenny	Ashley	Meera	Jane
2	Samantha	Christeen	Priya	Julia
3	(NULL)	Ketty	(NULL)	Maria

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

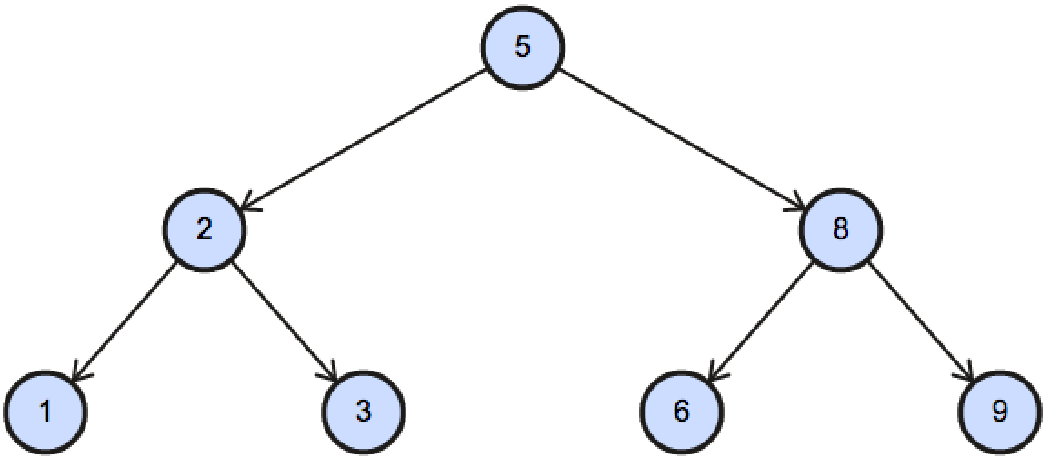
N	*P*
1	*2*
3	*2*
6	*8*
9	*8*
2	*5*
8	*5*
5	*null*

Sample Output

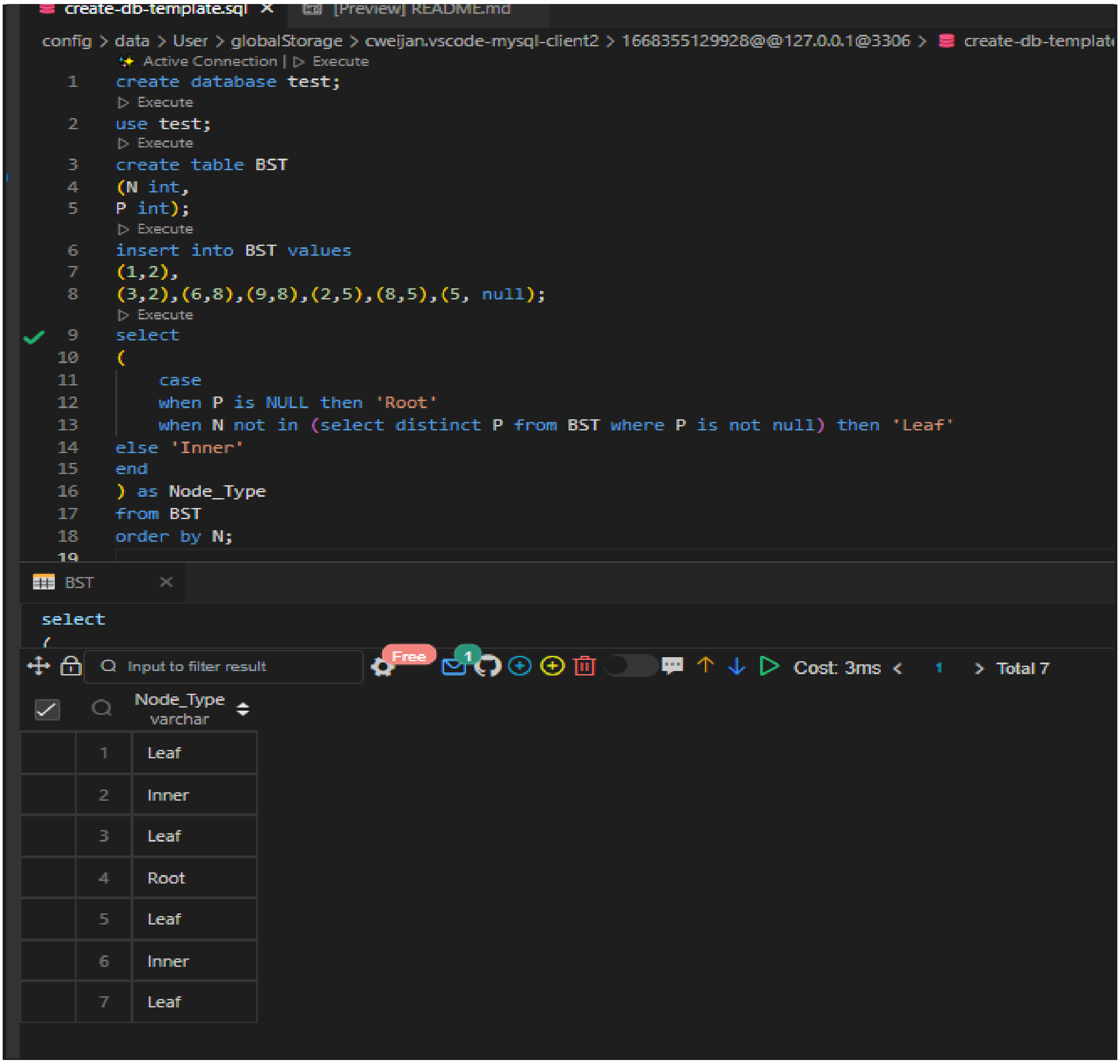
```
1 Leaf
2 Inner
3 Leaf
5 Root
6 Leaf
8 Inner
9 Leaf
```

Explanation

The Binary Tree below illustrates the sample:



```
Solution:
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;
```


create-db-template.sql ×
[Preview] README.md
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-templat
Active Connection | ▷ Execute
▷ Execute
BST ×
Input to filter result
Free
Cost: 3ms
1
Total 7

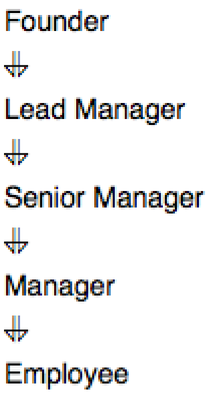
```
create database test;
use test;
create table BST
(N int,
P int);
insert into BST values
(1,2),
(3,2),(6,8),(9,8),(2,5),(8,5),(5, null);
select
(
    case
    when P is NULL then 'Root'
    when N not in (select distinct P from BST where P is not null) then 'Leaf'
else 'Inner'
end
) as Node_Type
from BST
order by N;
```

```
select
(
```

	Node_Type varchar
1	Leaf
2	Inner
3	Leaf
4	Root
5	Leaf
6	Inner
7	Leaf

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:

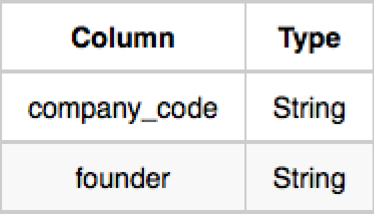
The tables may contain duplicate records.

- 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

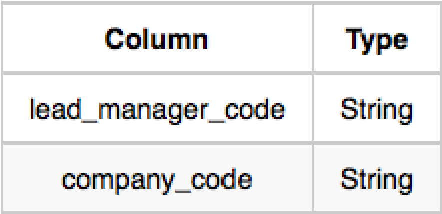
The following tables contain company data:

- Company: The company_code is the code of the company and founder is the founder of the company.



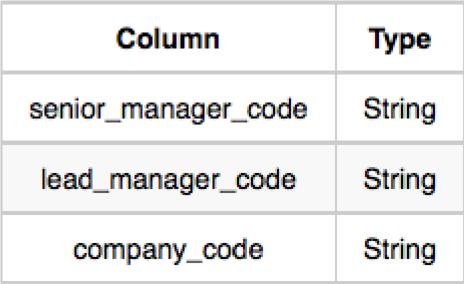
Column	Type
company_code	String
founder	String

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



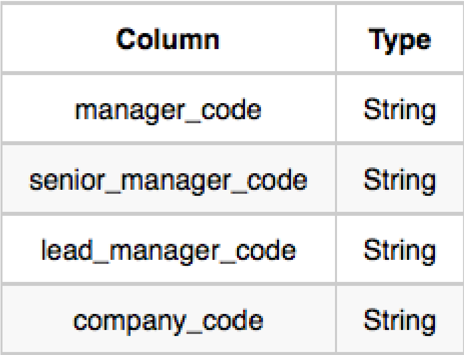
Column	Type
lead_manager_code	String
company_code	String

- 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.



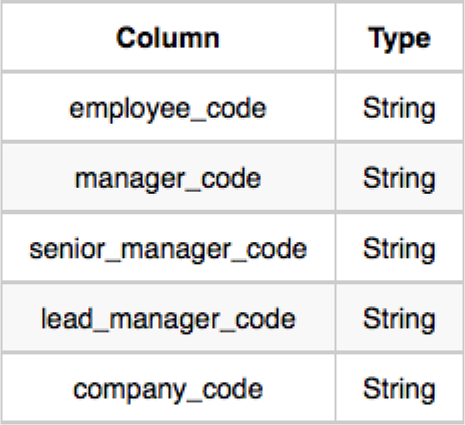
Column	Type
senior_manager_code	String
lead_manager_code	String
company_code	String

- 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

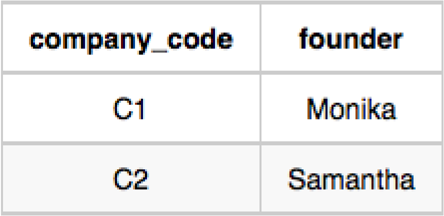
- Employee: The employee_code is the code of the employee, the manager_code is the code of its manager, the senior_manager_code is the code of its senior manager, the lead_manager_code is the code of its lead manager, and the company_code is the code of the working company.



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

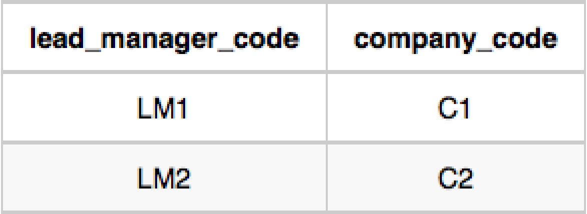
Sample Input

Company Table:



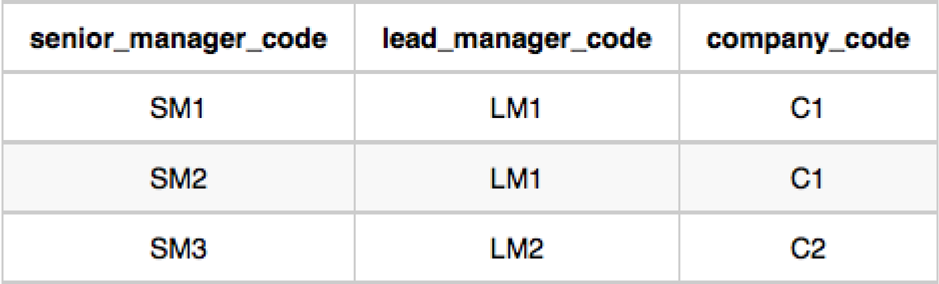
company_code	founder
C1	Monika
C2	Samantha

Lead_Manager Table:



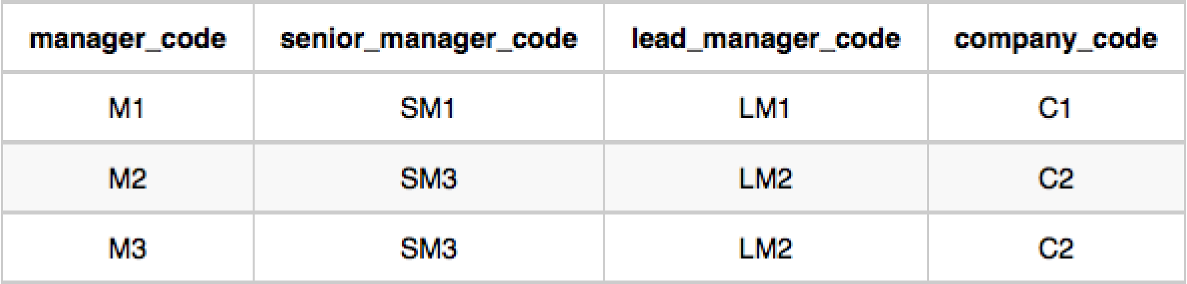
lead_manager_code	company_code
LM1	C1
LM2	C2

Senior_Manager Table:



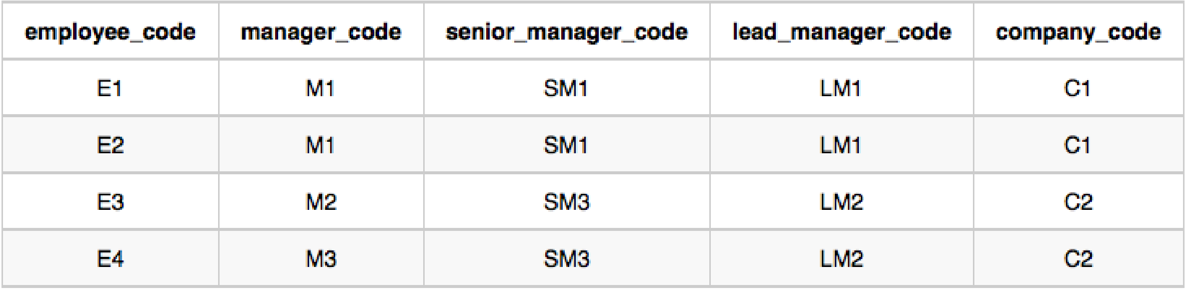
senior_manager_code	lead_manager_code	company_code
SM1	LM1	C1
SM2	LM1	C1
SM3	LM2	C2

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:

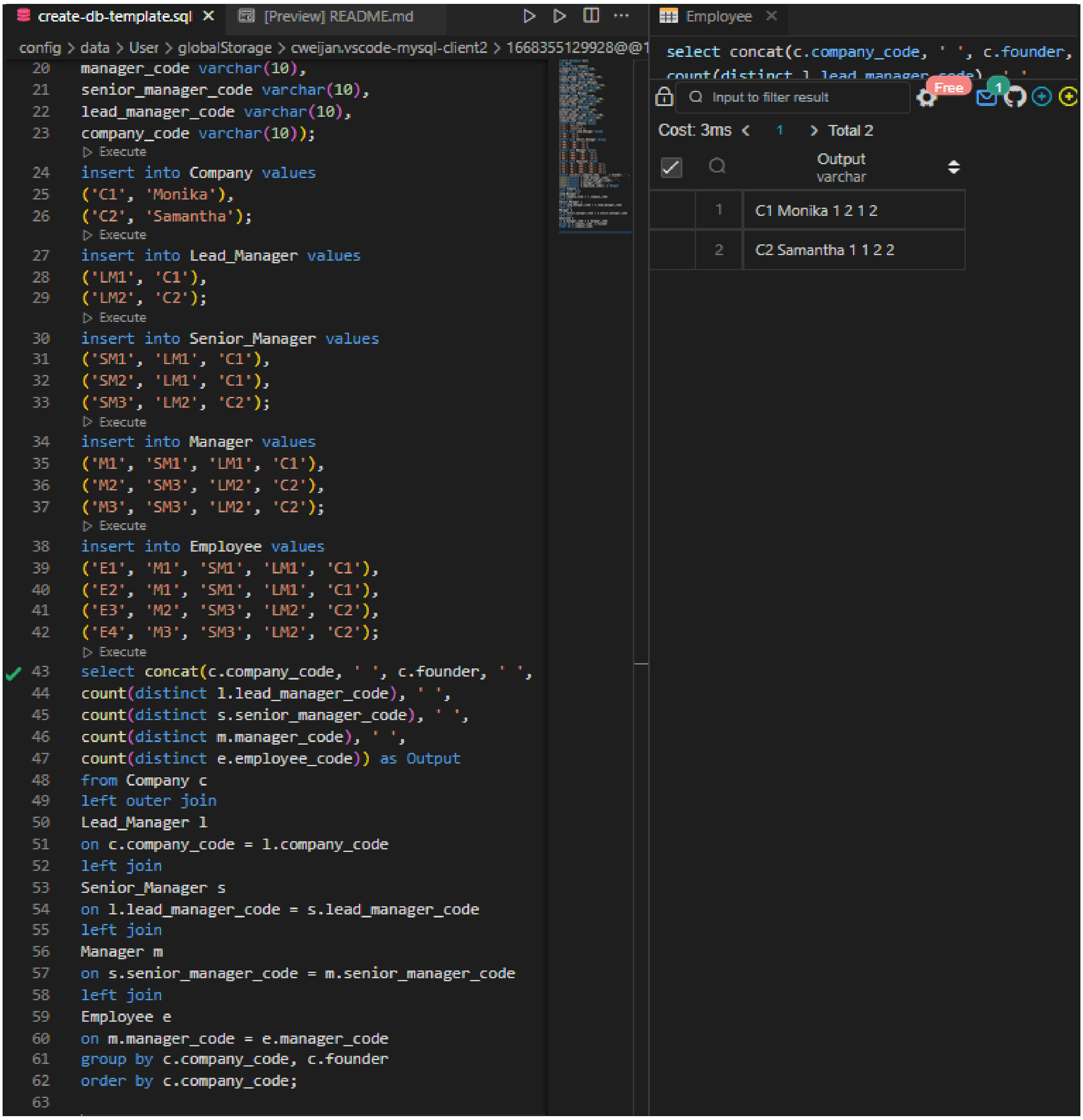


employee_code	manager_code	senior_manager_code	lead_manager_code	company_code
E1	M1	SM1	LM1	C1
E2	M1	SM1	LM1	C1
E3	M2	SM3	LM2	C2
E4	M3	SM3	LM2	C2

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.

```
Solution:
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;
```



create-db-template.sql ✕ [Preview] README.md

config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@1

```sql
manager_code varchar(10),
senior_manager_code varchar(10),
lead_manager_code varchar(10),
company_code varchar(10));
▷ Execute
insert into Company values
('C1', 'Monika'),
('C2', 'Samantha');
▷ Execute
insert into Lead_Manager values
('LM1', 'C1'),
('LM2', 'C2');
▷ Execute
insert into Senior_Manager values
('SM1', 'LM1', 'C1'),
('SM2', 'LM1', 'C1'),
('SM3', 'LM2', 'C2');
▷ Execute
insert into Manager values
('M1', 'SM1', 'LM1', 'C1'),
('M2', 'SM3', 'LM2', 'C2'),
('M3', 'SM3', 'LM2', 'C2');
▷ Execute
insert into Employee values
('E1', 'M1', 'SM1', 'LM1', 'C1'),
('E2', 'M1', 'SM1', 'LM1', 'C1'),
('E3', 'M2', 'SM3', 'LM2', 'C2'),
('E4', 'M3', 'SM3', 'LM2', 'C2');
▷ Execute
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;
```

Employee ✕

```sql
select concat(c.company_code, ' ', c.founder,
count(distinct l.lead_manager_code), ' '
```

Cost: 3ms < 1 > Total 2

	Output varchar
1	C1 Monika 1 2 1 2
2	C2 Samantha 1 1 2 2

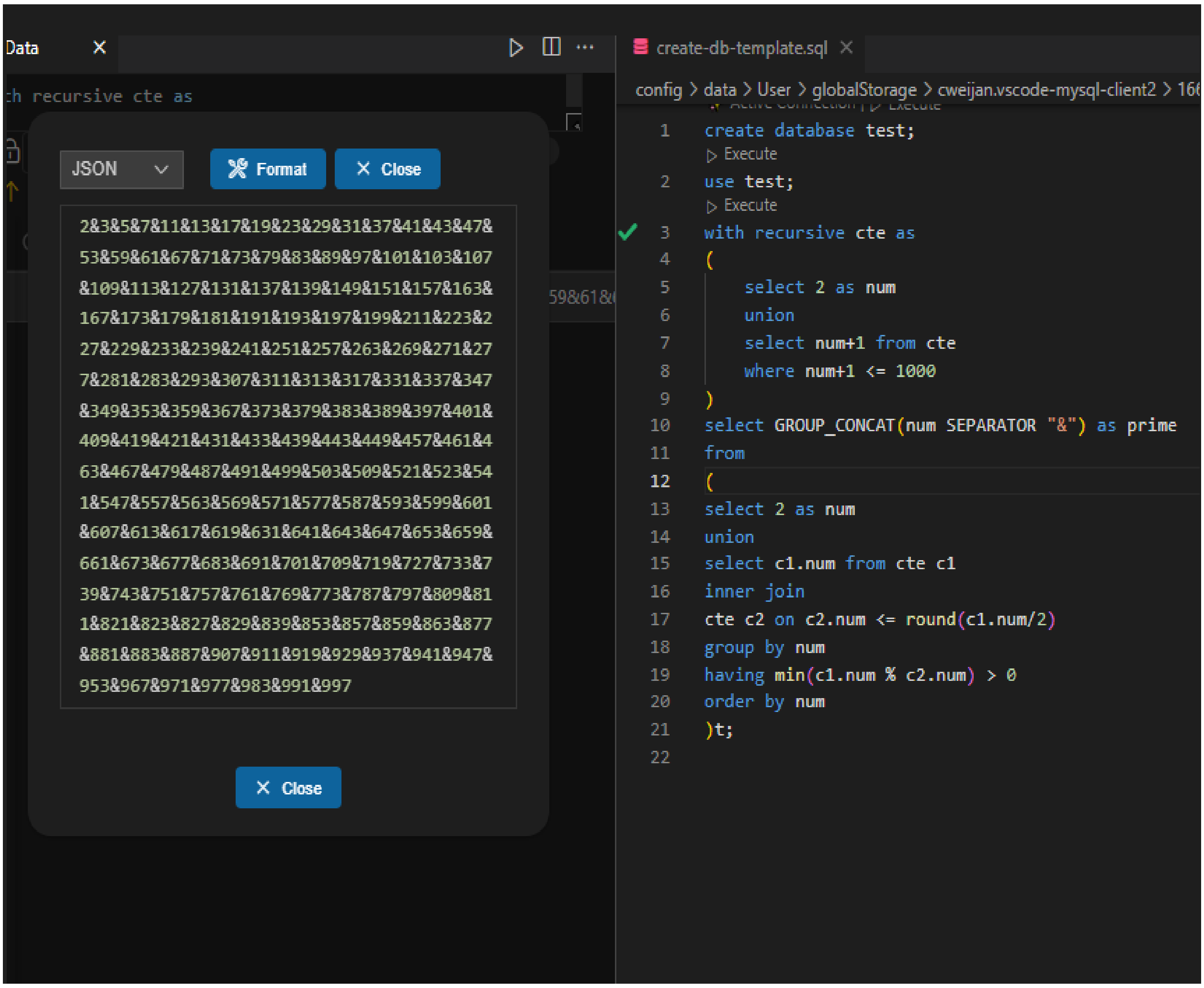
Q112.

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

```
Solution:
with recursive cte as
(
    select 2 as num
```

```sql
    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;
```


Data
create-db-template.sql
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 166
JSON
Format
Close
Execute
Close

```
2&3&5&7&11&13&17&19&23&29&31&37&41&43&47&
53&59&61&67&71&73&79&83&89&97&101&103&107
&109&113&127&131&137&139&149&151&157&163&
167&173&179&181&191&193&197&199&211&223&2
27&229&233&239&241&251&257&263&269&271&27
7&281&283&293&307&311&313&317&331&337&347
&349&353&359&367&373&379&383&389&397&401&
409&419&421&431&433&439&443&449&457&461&4
63&467&479&487&491&499&503&509&521&523&54
1&547&557&563&569&571&577&587&593&599&601
&607&613&617&619&631&641&643&647&653&659&
661&673&677&683&691&701&709&719&727&733&7
39&743&751&757&761&769&773&787&797&809&81
1&821&823&827&829&839&853&857&859&863&877
&881&883&887&907&911&919&929&937&941&947&
953&967&971&977&983&991&997
```

```sql
create database test;
use test;
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):

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

* * * *

* * * * *

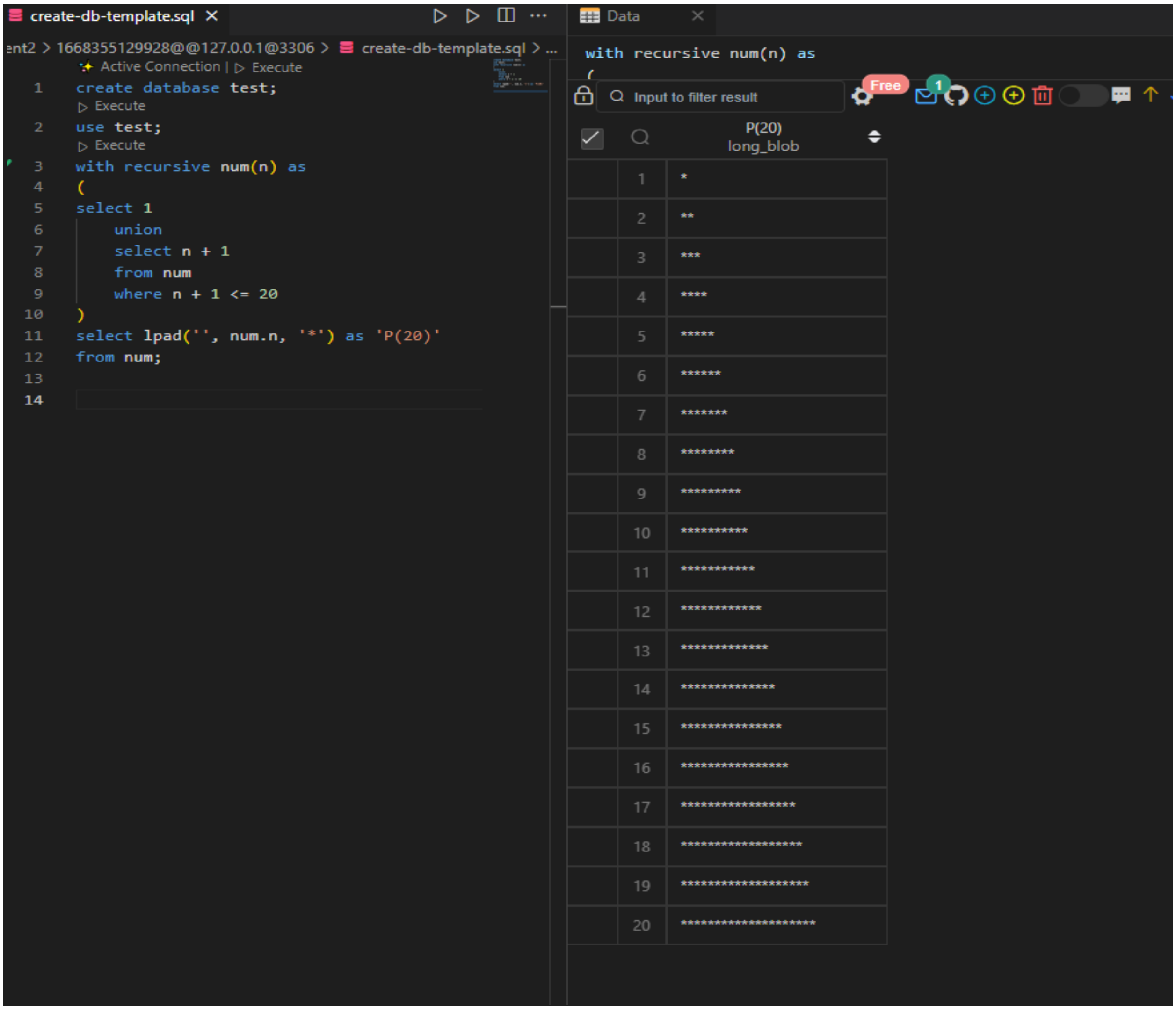
Write a query to print the pattern P(20).Level

- Easy

Source - Hackerrank

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

```
Solution:
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;
```


create-db-template.sql
ent2 > 1668355129928@@127.0.0.1@3306 > create-db-template.sql > ...
Active Connection | ▷ Execute
▷ Execute
Data
Input to filter result
Free

```
create database test;
use test;
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;
```

```
with recursive num(n) as
(
```

	P(20) long_blob
1	*
2	**
3	***
4	****
5	*****
6	******
7	*******
8	********
9	*********
10	**********
11	***********
12	************
13	*************
14	**************
15	***************
16	****************
17	*****************
18	******************
19	*******************
20	********************

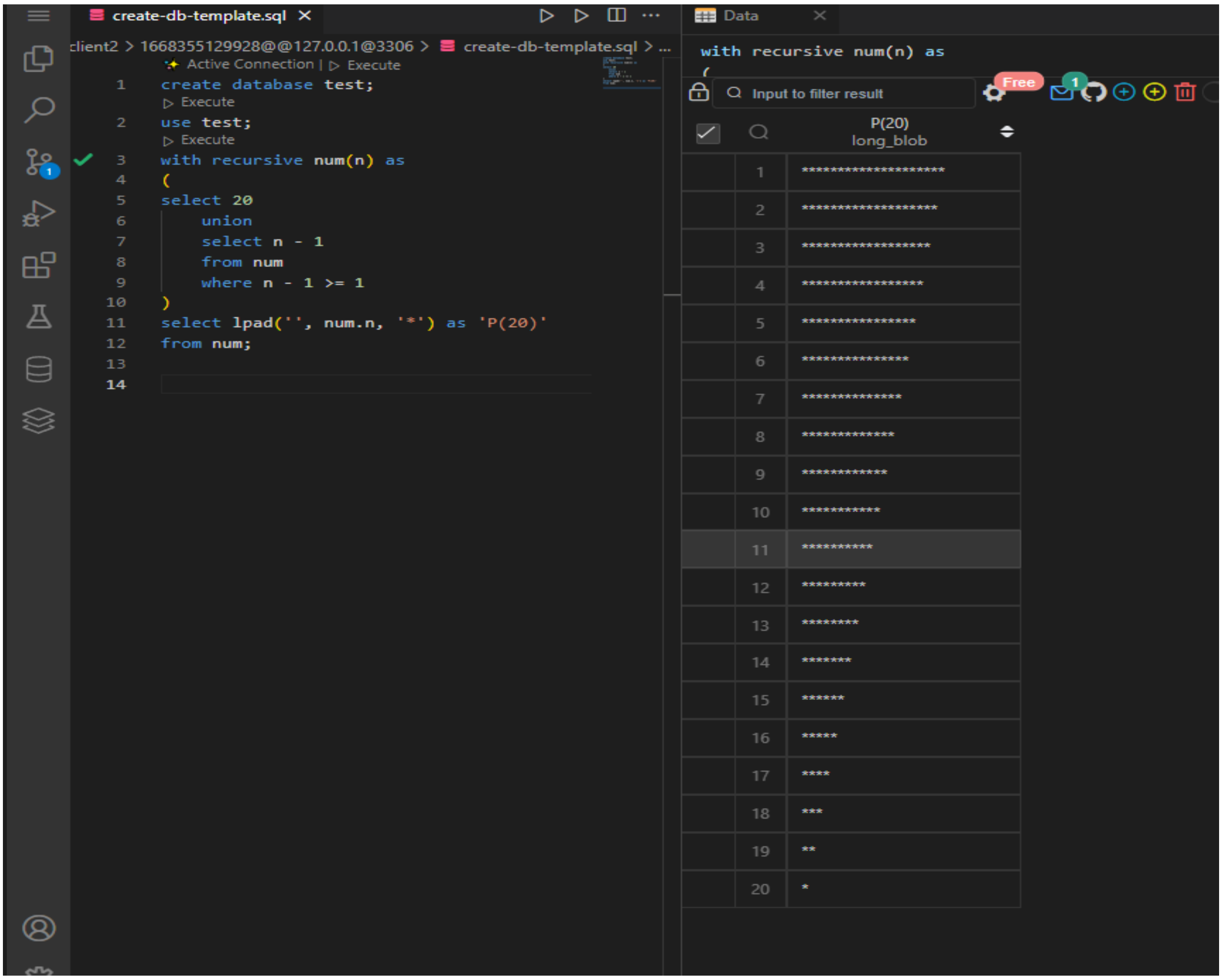
Q114.

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

```
Solution:
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;
```

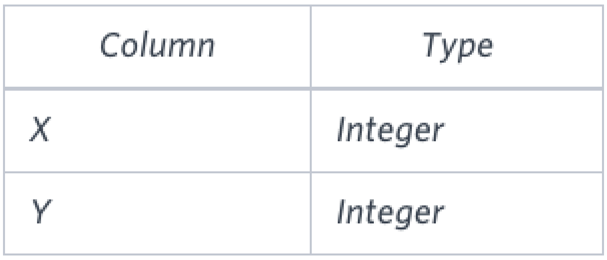

create-db-template.sql
client2 > 1668355129928@@127.0.0.1@3306 > create-db-template.sql > ...
Active Connection | ▷ Execute
▷ Execute
▷ Execute
Data
Input to filter result
Free

```
create database test;
use test;
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;
```

```
with recursive num(n) as
(
```

	P(20) long_blob
1	********************
2	*******************
3	******************
4	*****************
5	****************
6	***************
7	**************
8	*************
9	************
10	***********
11	**********
12	*********
13	********
14	*******
15	******
16	*****
17	****
18	***
19	**
20	*

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

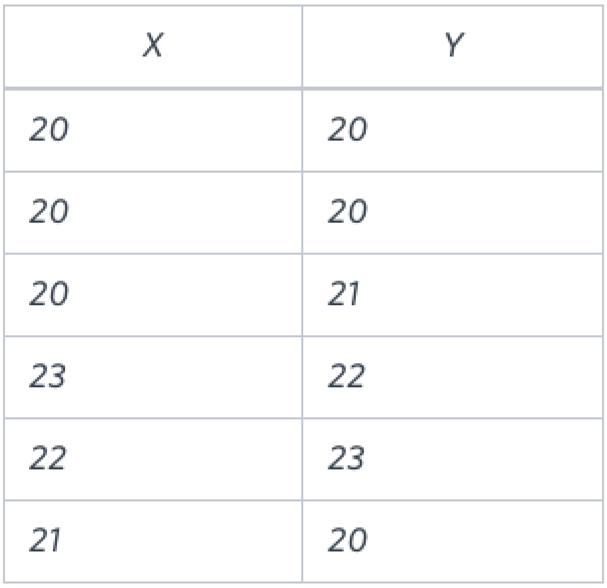


Column	Type
X	Integer
Y	Integer

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.

.

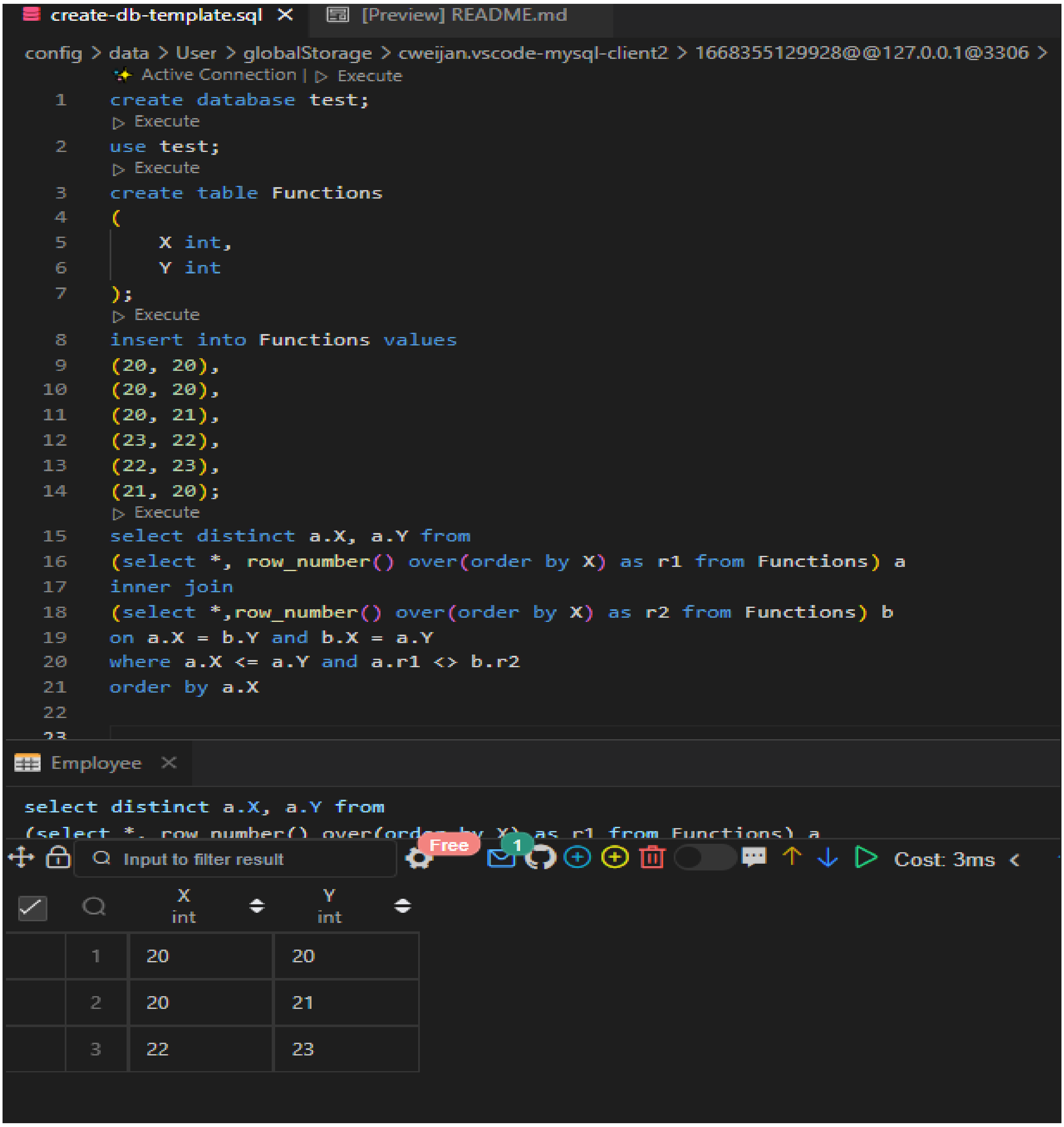
Sample Input



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

Sample Output
20 20
20 21
22 23

```
Solution:
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
```


create-db-template.sql
[Preview] README.md
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 >
Active Connection | ▷ Execute
▷ Execute
Employee
Input to filter result
Free
1
Cost: 3ms

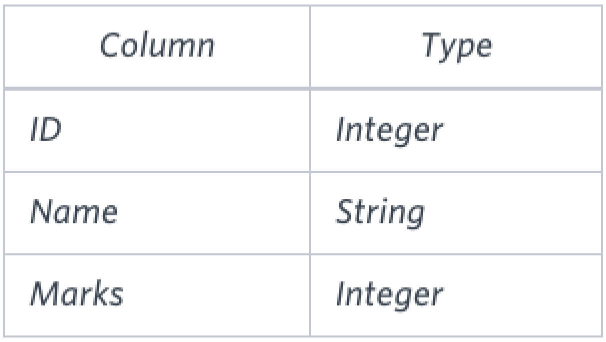
```
create database test;
use test;
create table Functions
(
    X int,
    Y int
);
insert into Functions values
(20, 20),
(20, 20),
(20, 21),
(23, 22),
(22, 23),
(21, 20);
select distinct a.X, a.Y from
(select *, row_number() over(order by X) as r1 from Functions) a
inner join
(select *,row_number() 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
```

```
select distinct a.X, a.Y from
(select *, row_number() over(order by X) as r1 from Functions) a
```

	X int	Y int
1	20	20
2	20	21
3	22	23

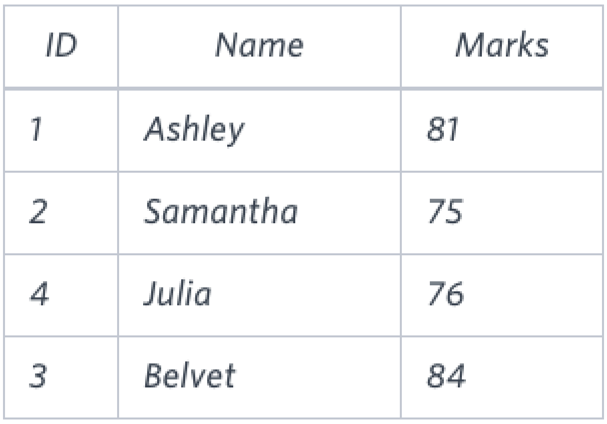
Q115.

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



Column	Type
ID	Integer
Name	String
Marks	Integer

The STUDENTS table is described as follows:
The Name column only contains uppercase (A-Z) and lowercase (a-z) letters.
Sample Input

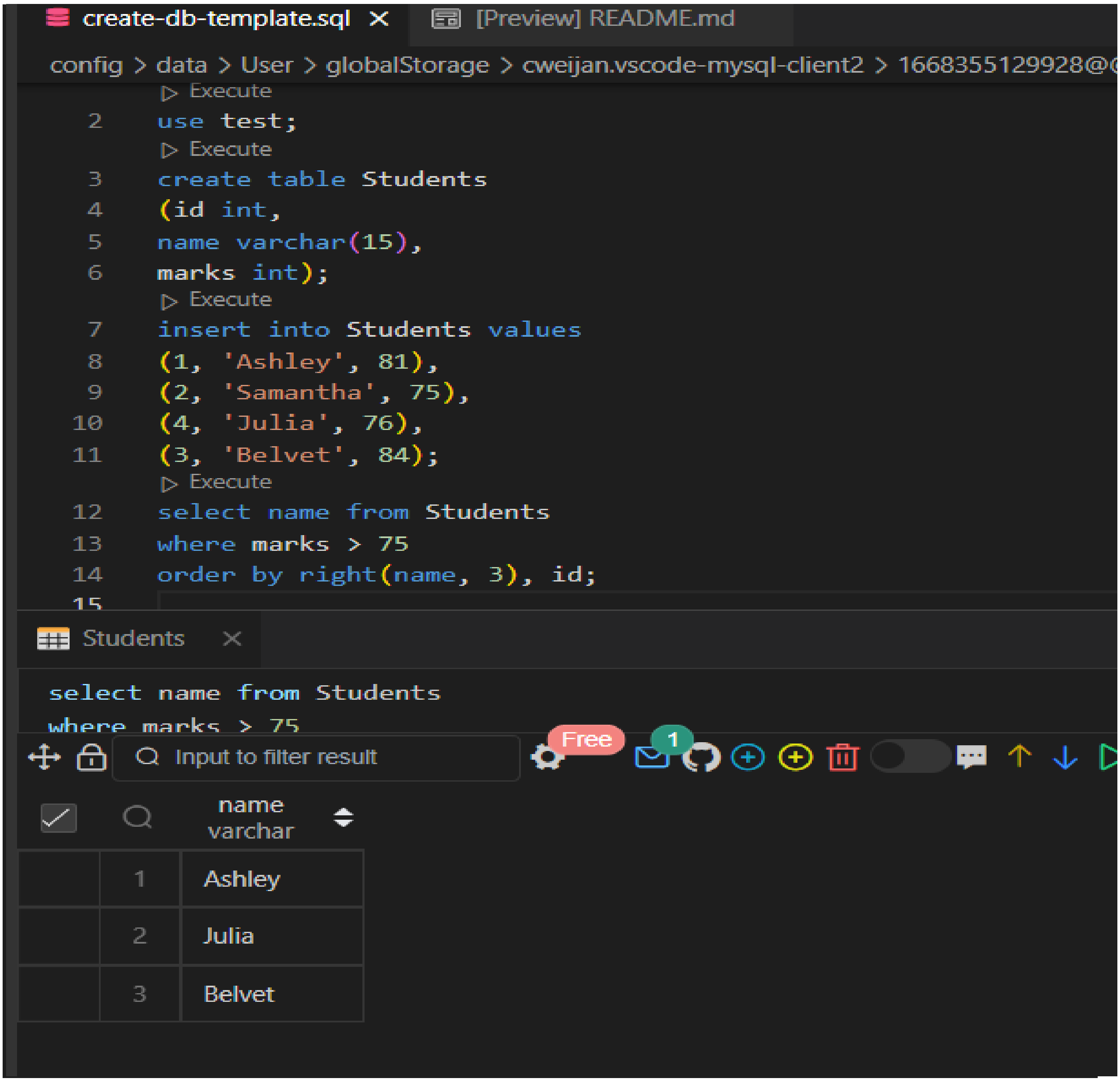


ID	Name	Marks
1	Ashley	81
2	Samantha	75
4	Julia	76
3	Belvet	84

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'.

```
Solution:
select name from Students
where marks > 75
order by right(name, 3), id;
```


create-db-template.sql
[Preview] README.md
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@

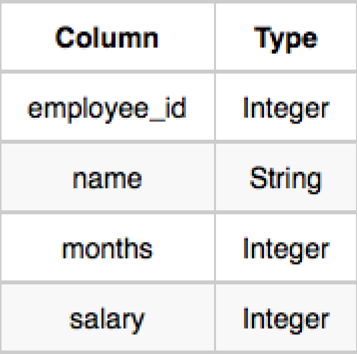
```sql
use test;
create table Students
(id int,
name varchar(15),
marks int);
insert into Students values
(1, 'Ashley', 81),
(2, 'Samantha', 75),
(4, 'Julia', 76),
(3, 'Belvet', 84);
select name from Students
where marks > 75
order by right(name, 3), id;
```

```sql
select name from Students
where marks > 75
```

	name varchar
1	Ashley
2	Julia
3	Belvet

Q116.

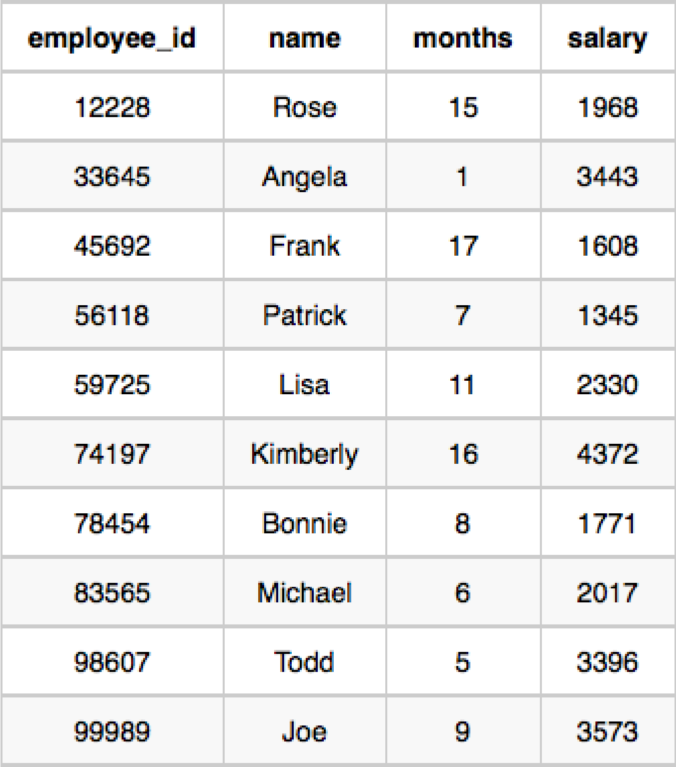
Write a query that prints a list of employee names (i.e.: the name attribute) from the Employee table in alphabetical order.
Level - Easy
Hint - Use ORDER BY
Input Format
The Employee table containing employee data for a company is described as follows:



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 their monthly salary.

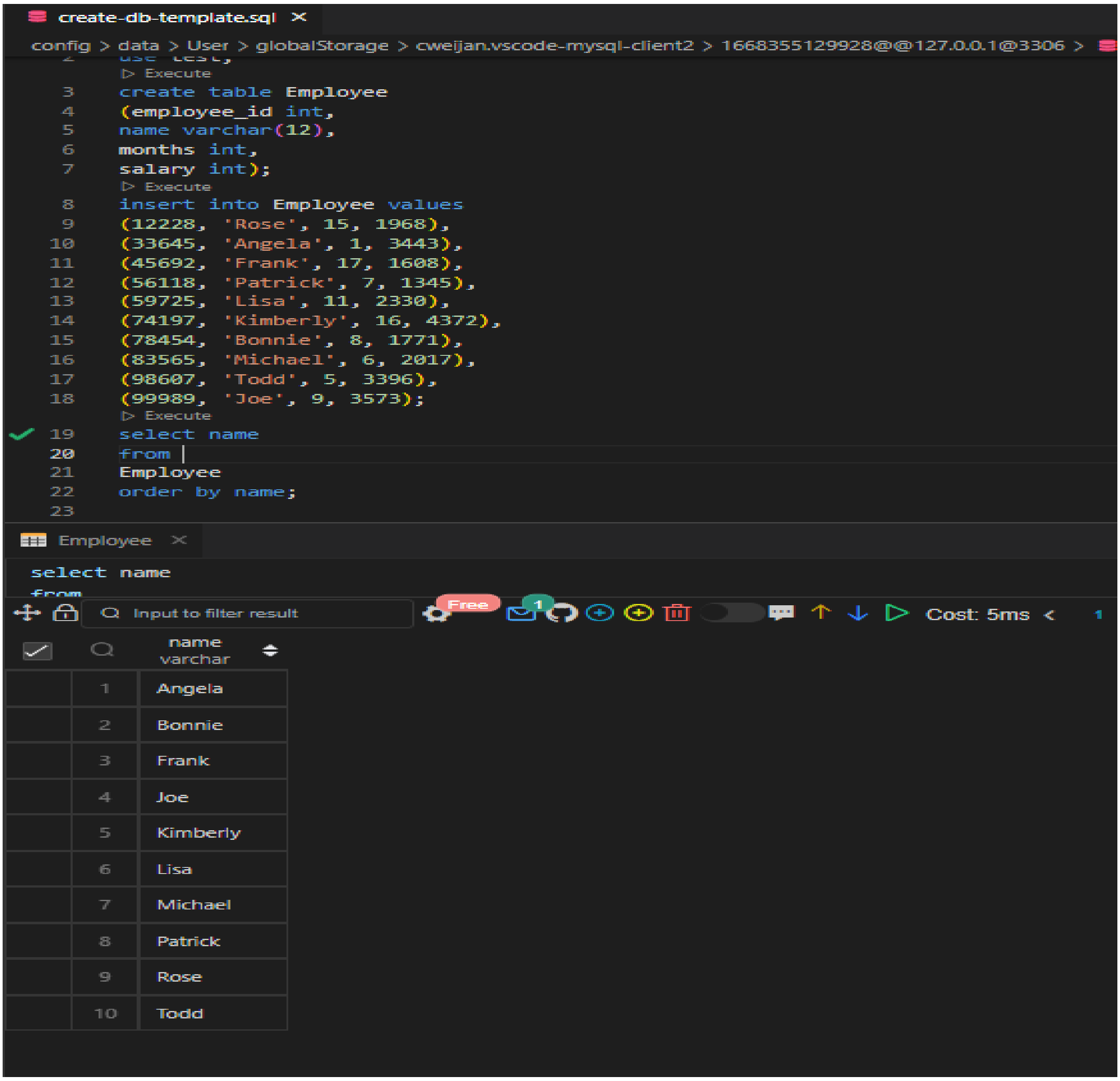
Sample Input



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
Bonnie
Frank
Joe
Kimberly
Lisa
Michael
Patrick
Rose
Todd

```
Solution:
select name
from
Employee
order by name;
```


create-db-template.sql ✕
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 >

```
use test;
▷ Execute
create table Employee
(employee_id int,
name varchar(12),
months int,
salary int);
▷ Execute
insert into Employee values
(12228, 'Rose', 15, 1968),
(33645, 'Angela', 1, 3443),
(45692, 'Frank', 17, 1608),
(56118, 'Patrick', 7, 1345),
(59725, 'Lisa', 11, 2330),
(74197, 'Kimberly', 16, 4372),
(78454, 'Bonnie', 8, 1771),
(83565, 'Michael', 6, 2017),
(98607, 'Todd', 5, 3396),
(99989, 'Joe', 9, 3573);
▷ Execute
select name
from
Employee
order by name;
```

```
select name
from
```

	name varchar
1	Angela
2	Bonnie
3	Frank
4	Joe
5	Kimberly
6	Lisa
7	Michael
8	Patrick
9	Rose
10	Todd

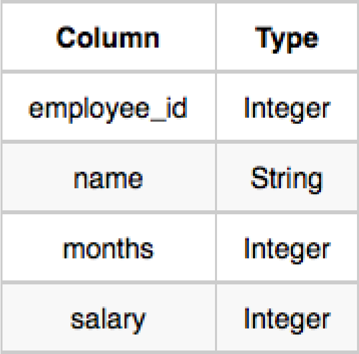
Q117. Write a query that prints a list of employee names (i.e.: the name attribute) for employees in Employee having a salary greater than $2000 per month who have been employees for less than 10 months. Sort your result by ascending employee_id.

Level - Easy

Hint - Use AscendingInput

Format

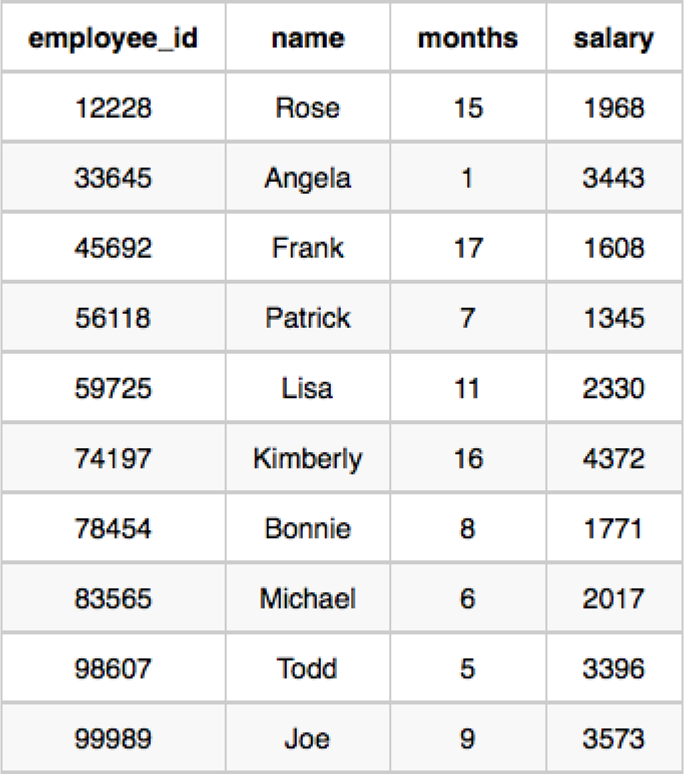
The Employee table containing employee data for a company is described as follows:



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.

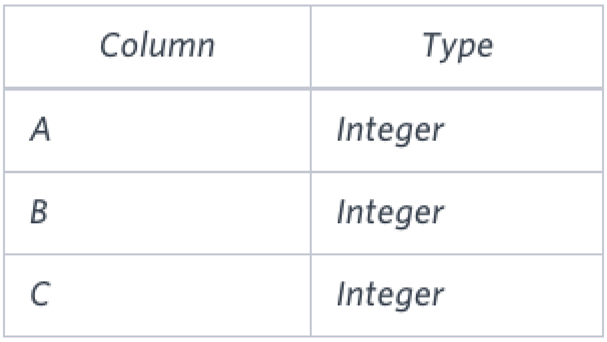
Todd has been an employee for 5 months and earns $3396 per month.
Joe has been an employee for 9 months and earns $3573 per month.
We order our output by ascending employee_id.

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.

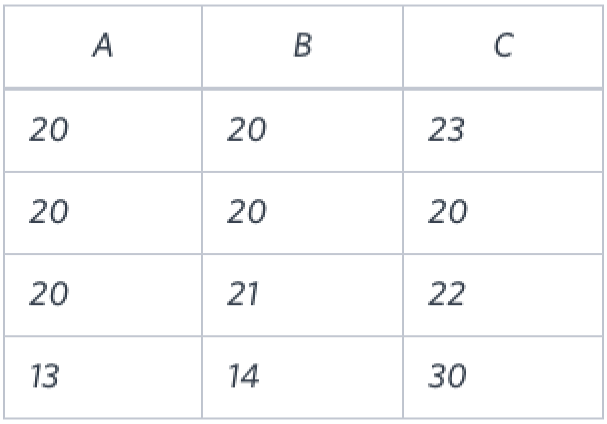
Level - Easy
Hint - Use predefined functions for calculation.

Input Format
The TRIANGLES table is described as follows:



Column	Type
A	Integer
B	Integer
C	Integer

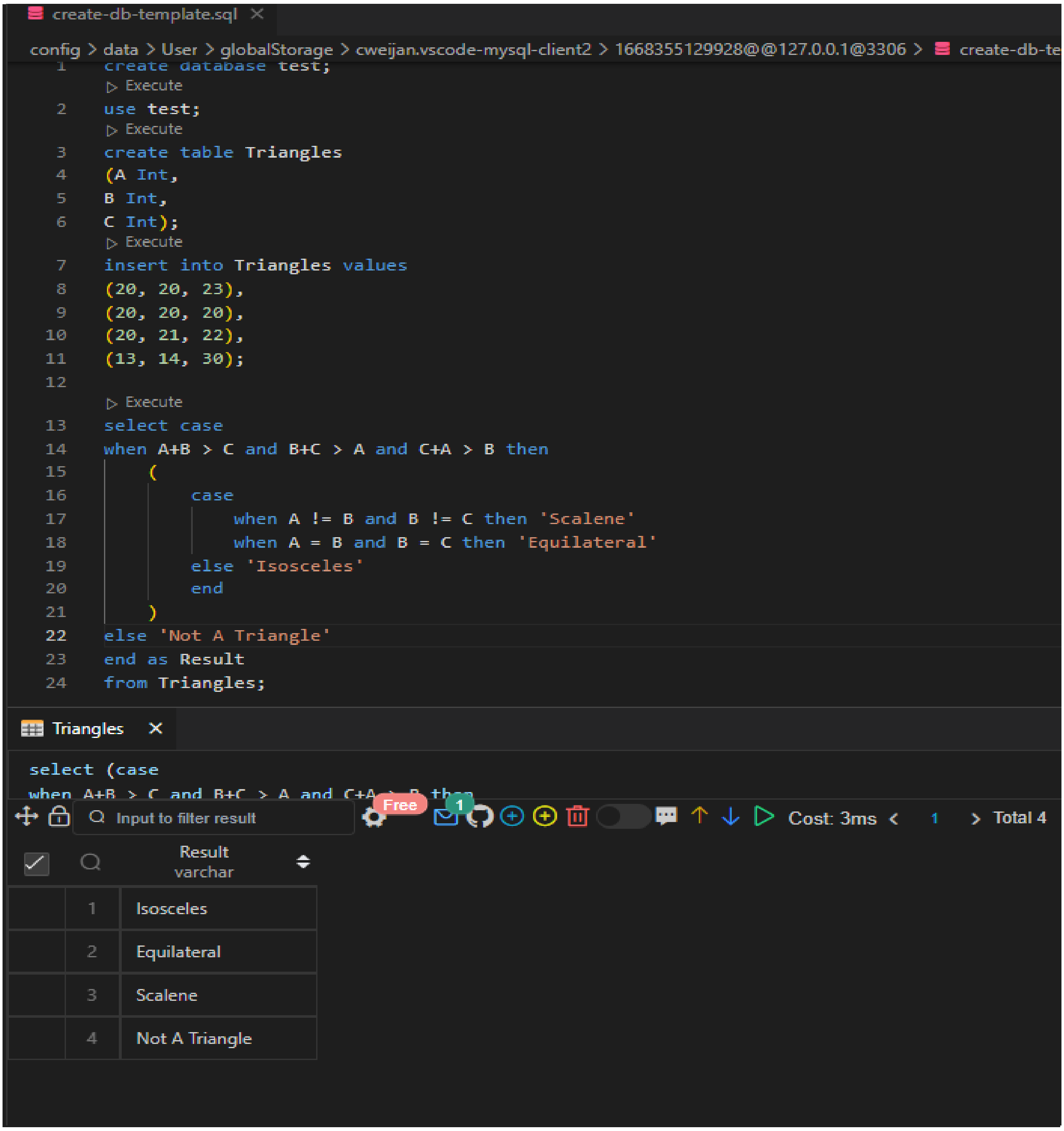
Each row in the table denotes the lengths of each of a triangle's three sides.
Sample Input



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 ≡ 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 .

```
Solution:
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;
```


create-db-template.sql
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-te
Triangles
Input to filter result
Free
Cost: 3ms
1
Total 4

```
create database test;
  ▷ Execute
use test;
  ▷ Execute
create table Triangles
(A Int,
B Int,
C Int);
  ▷ Execute
insert into Triangles values
(20, 20, 23),
(20, 20, 20),
(20, 21, 22),
(13, 14, 30);

  ▷ Execute
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;
```

```
select (case
when A+B > C and B+C > A and C+A > B then
```

	Result varchar
1	Isosceles
2	Equilateral
3	Scalene
4	Not A Triangle

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

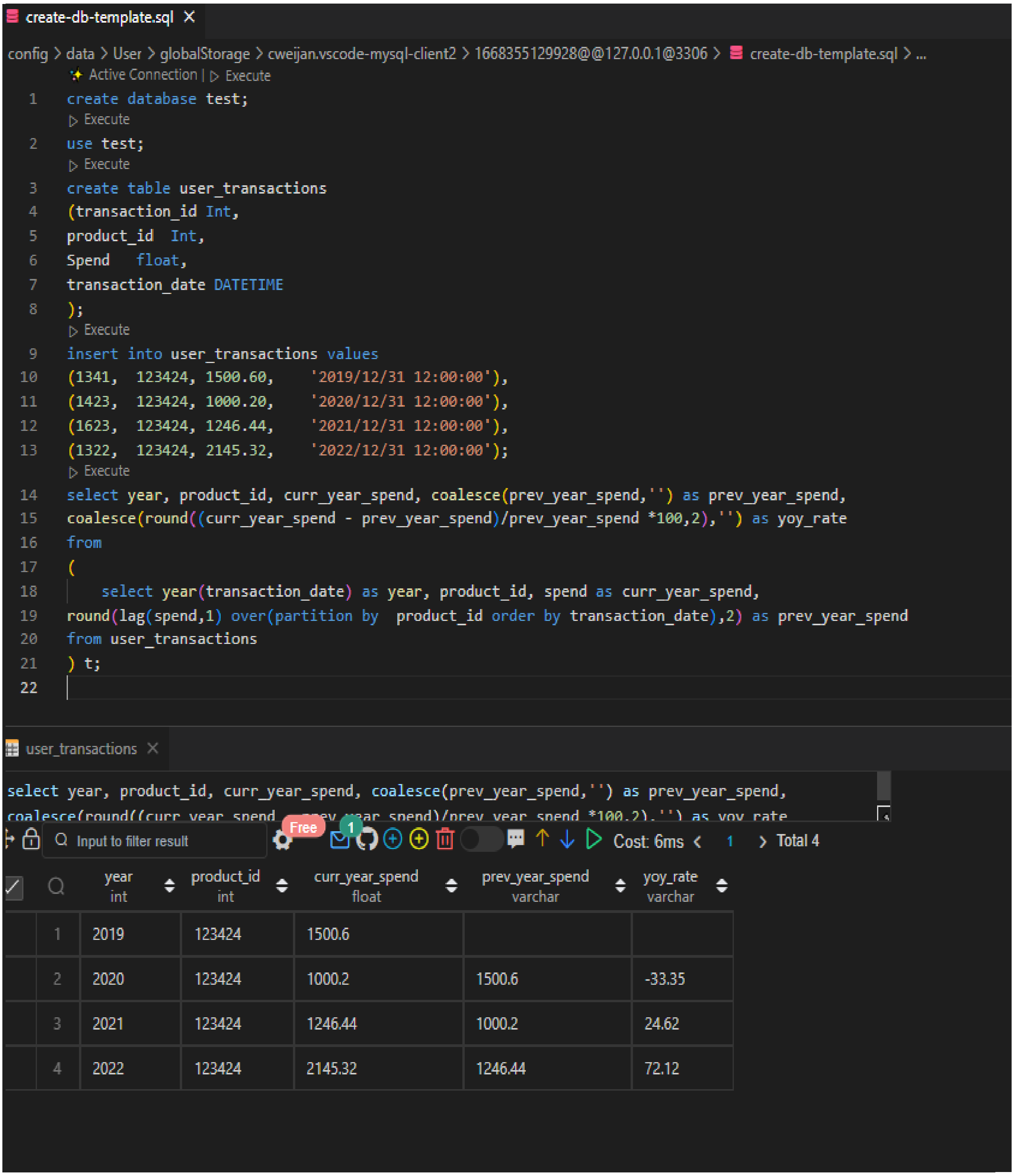
user_transactions Example Input:

transaction_id	product_id	spend	transaction_date
1341	123424	1500.60	12/31/2019 12:00:00
1423	123424	1000.20	12/31/2020 12:00:00
1623	123424	1246.44	12/31/2021 12:00:00
1322	123424	2145.32	12/31/2022 12:00:00

Example Output:

y	product_id	curr_year_spend	prev_year_spend	yoy_rate
2	123424	1500.60		
2	123424	1000.20	1500.60	-33.35
2	123424	1246.44	1000.20	24.62
2	123424	2145.32	1246.44	72.12

```
Solution:
select year, product_id, curr_year_spend, coalesce(prev_year_spend,'') as
prev_year_spend,
coalesce(round((curr_year_spend - prev_year_spend)/prev_year_spend *100,2),'')
as yoy_rate
from
(
    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;
```


create-db-template.sql
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-template.sql > ...
Active Connection | ▷ Execute
user_transactions
Input to filter result
Free
Cost: 6ms
Total 4

```sql
create database test;
use test;
create table user_transactions
(transaction_id Int,
product_id  Int,
Spend   float,
transaction_date DATETIME
);
insert into user_transactions values
(1341,  123424, 1500.60,    '2019/12/31 12:00:00'),
(1423,  123424, 1000.20,    '2020/12/31 12:00:00'),
(1623,  123424, 1246.44,    '2021/12/31 12:00:00'),
(1322,  123424, 2145.32,    '2022/12/31 12:00:00');
select year, product_id, curr_year_spend, coalesce(prev_year_spend,'') as prev_year_spend,
coalesce(round((curr_year_spend - prev_year_spend)/prev_year_spend *100,2),'') as yoy_rate
from
(
    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;
```

```sql
select year, product_id, curr_year_spend, coalesce(prev_year_spend,'') as prev_year_spend,
```

	year int	product_id int	curr_year_spend float	prev_year_spend varchar	yoy_rate varchar
1	2019	123424	1500.6		
2	2020	123424	1000.2	1500.6	-33.35
3	2021	123424	1246.44	1000.2	24.62
4	2022	123424	2145.32	1246.44	72.12

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

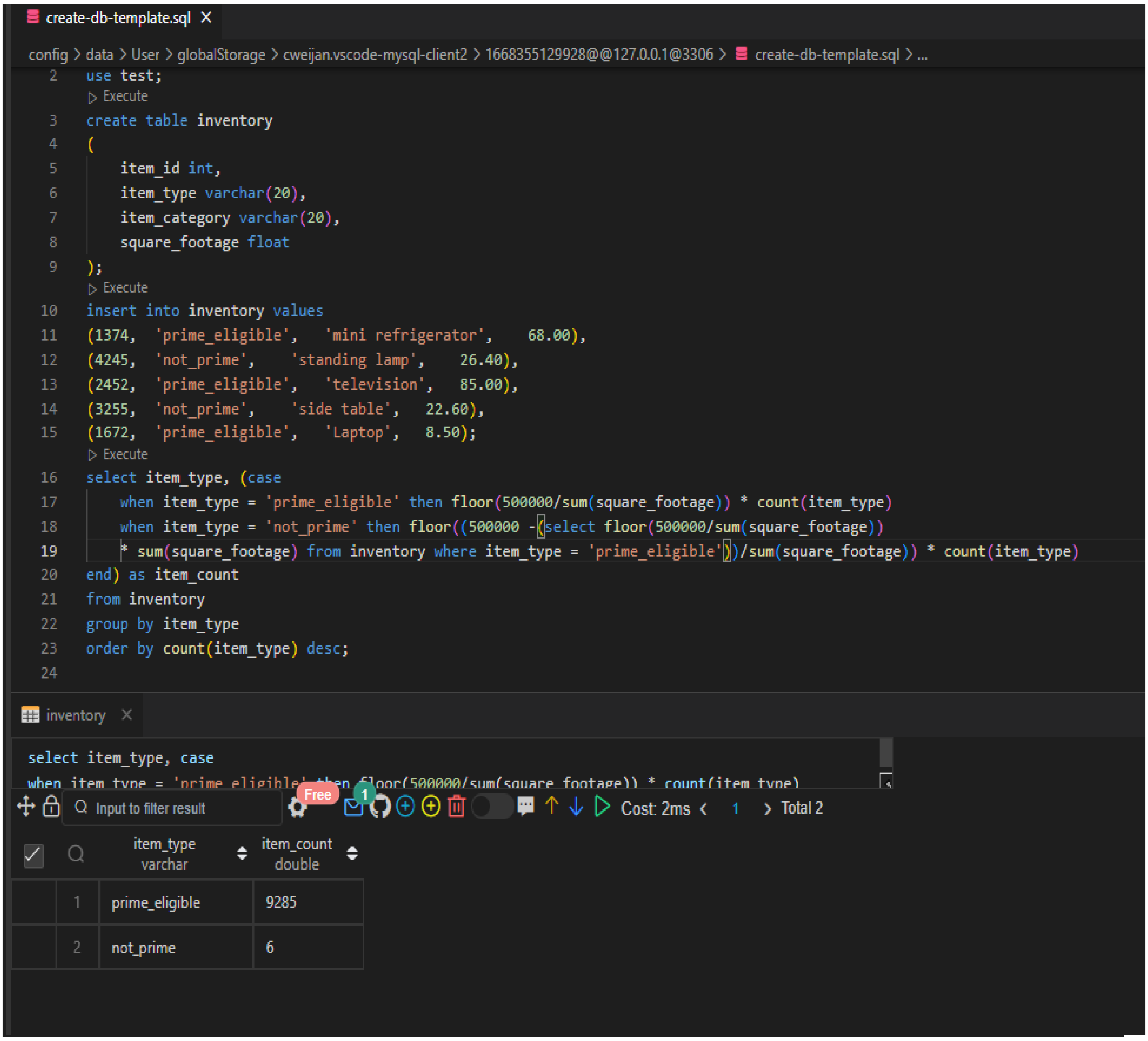
inventory Example Input:

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

```
Solution:
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;
```


create-db-template.sql
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-template.sql > ...
▷ Execute
inventory
Input to filter result
Free
Cost: 2ms
1
Total 2

```sql
use test;
create table inventory
(
    item_id int,
    item_type varchar(20),
    item_category varchar(20),
    square_footage float
);
insert into inventory values
(1374,  'prime_eligible',   'mini refrigerator',    68.00),
(4245,  'not_prime',    'standing lamp',    26.40),
(2452,  'prime_eligible',   'television',   85.00),
(3255,  'not_prime',    'side table',   22.60),
(1672,  'prime_eligible',   'Laptop',   8.50);
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;
```

```
select item_type, case
when item_type = 'prime_eligible' then floor(500000/sum(square_footage)) * count(item_type)
```

		item_type varchar	item_count double
	1	prime_eligible	9285
	2	not_prime	6

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

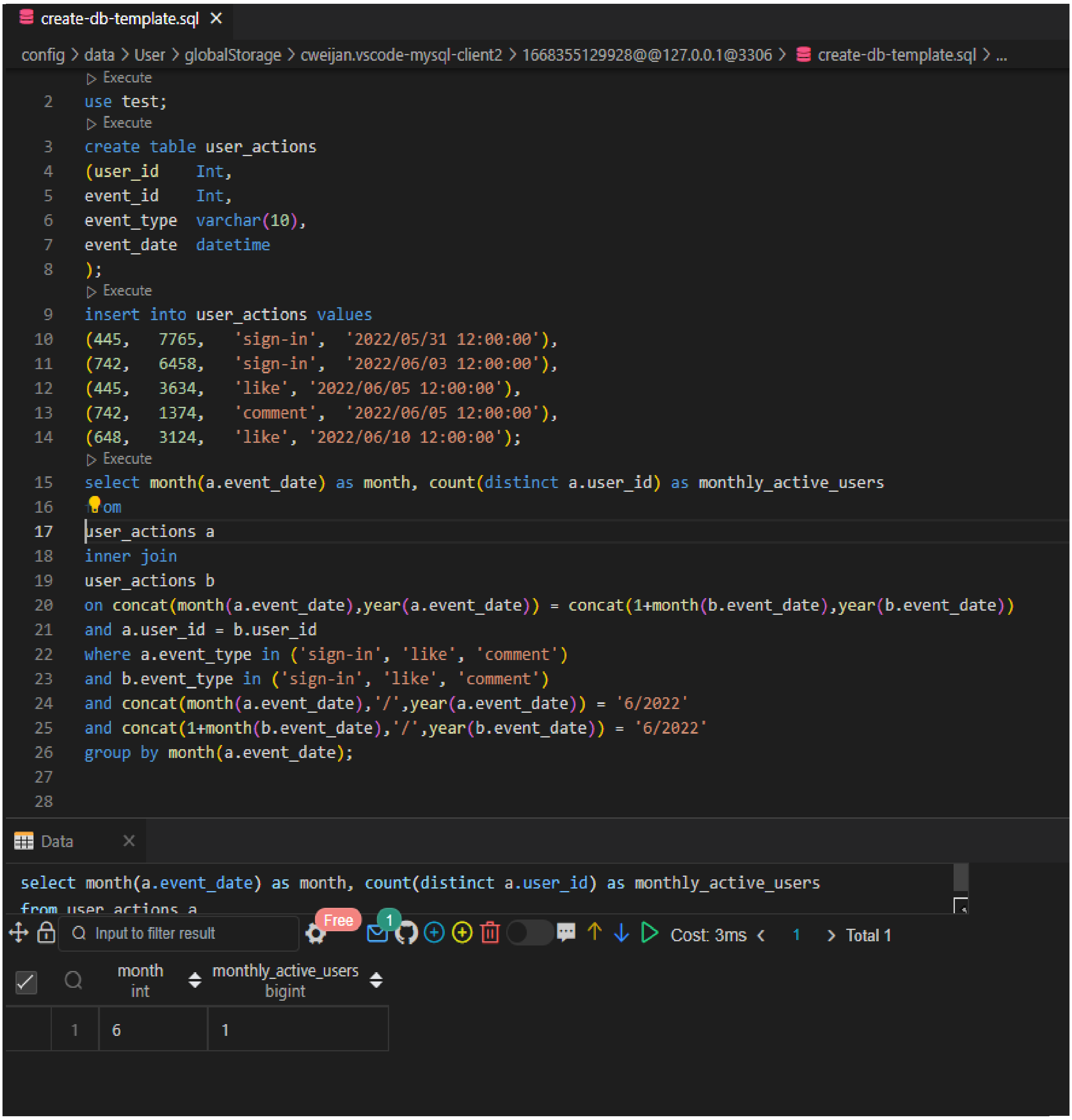
Example Output for June 2022:

month	monthly_active_users
6	1

```
Solution: For July Month
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);
```

```
Solution: For June Month
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'
```

```sql
and concat(1+month(b.event_date),'/',year(b.event_date)) = '6/2022'
group by month(a.event_date);
```



create-db-template.sql ✕
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-template.sql > ...
▷ Execute
Data ✕
Input to filter result
Free
Cost: 3ms
Total 1

```sql
use test;
create table user_actions
(user_id    Int,
event_id    Int,
event_type  varchar(10),
event_date  datetime
);
insert into user_actions values
(445,   7765,   'sign-in',  '2022/05/31 12:00:00'),
(742,   6458,   'sign-in',  '2022/06/03 12:00:00'),
(445,   3634,   'like', '2022/06/05 12:00:00'),
(742,   1374,   'comment',  '2022/06/05 12:00:00'),
(648,   3124,   'like', '2022/06/10 12:00:00');
select month(a.event_date) as month, count(distinct a.user_id) as monthly_active_users
om
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);
```

```
select month(a.event_date) as month, count(distinct a.user_id) as monthly_active_users
from user_actions a
```

	month int	monthly_active_users bigint
1	6	1

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

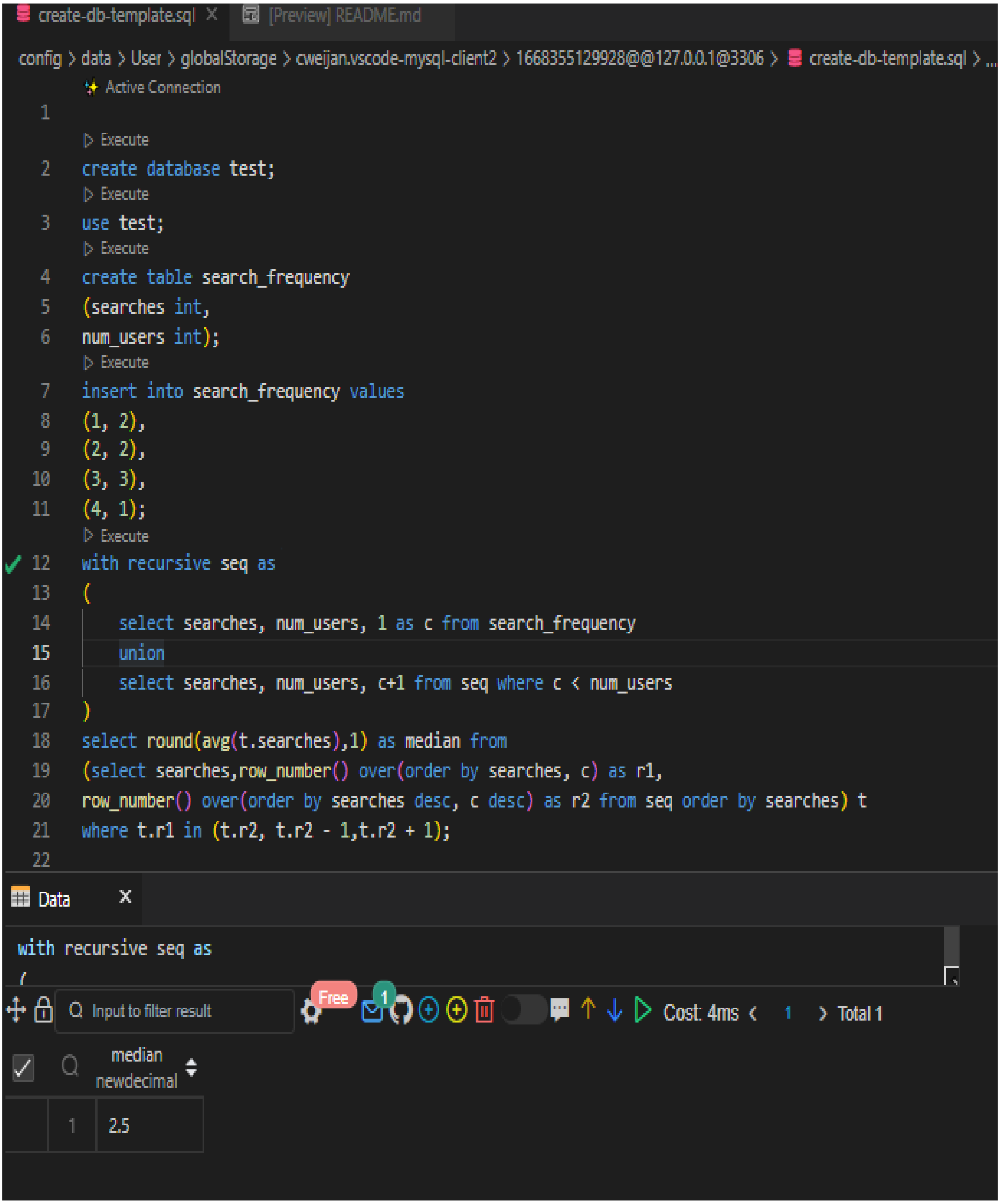
search_frequency Example Input:

searches	num_users
1	2
2	2
3	3
4	1

Example Output:

median
2.5

```
Solution:
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);
```

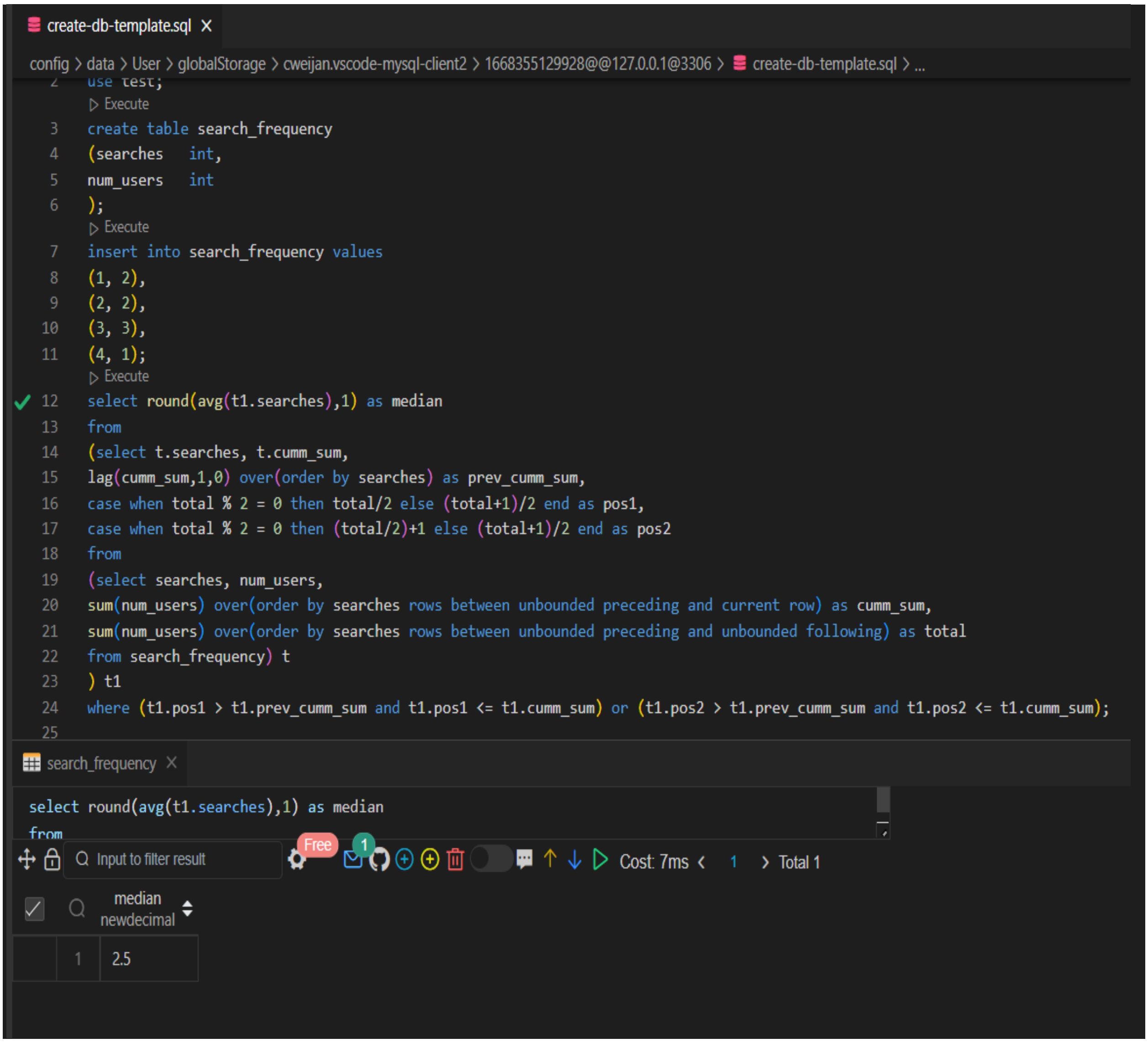

create-db-template.sql
[Preview] README.md
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-template.sql > ...
Active Connection
Execute
Data
Input to filter result
Free
Cost: 4ms
1
Total 1

```sql

create database test;
use test;
create table search_frequency
(searches int,
num_users int);
insert into search_frequency values
(1, 2),
(2, 2),
(3, 3),
(4, 1);
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);

```

```
with recursive seq as
(
```

```
     median
     newdecimal
1    2.5
```

```
Solution: using cumulative sum
select round(avg(t1.searches),1) as median
from
(select t.searches, t.cumm_sum,
lag(cumm_sum,1,0) over(order by searches) as prev_cumm_sum,
case when total % 2 = 0 then total/2 else (total+1)/2 end as pos1,
case when total % 2 = 0 then (total/2)+1 else (total+1)/2 end as pos2
from
(select searches, num_users,
sum(num_users) over(order by searches rows between unbounded preceding and
current row) as cumm_sum,
sum(num_users) over(order by searches rows between unbounded preceding and
unbounded following) as total
from search_frequency) t
) t1
where (t1.pos1 > t1.prev_cumm_sum and t1.pos1 <= t1.cumm_sum) or (t1.pos2 >
t1.prev_cumm_sum and t1.pos2 <= t1.cumm_sum);
```



create-db-template.sql X
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-template.sql > ...

```
use test;
▷ Execute
create table search_frequency
(searches   int,
num_users   int
);
▷ Execute
insert into search_frequency values
(1, 2),
(2, 2),
(3, 3),
(4, 1);
▷ Execute
select round(avg(t1.searches),1) as median
from
(select t.searches, t.cumm_sum,
lag(cumm_sum,1,0) over(order by searches) as prev_cumm_sum,
case when total % 2 = 0 then total/2 else (total+1)/2 end as pos1,
case when total % 2 = 0 then (total/2)+1 else (total+1)/2 end as pos2
from
(select searches, num_users,
sum(num_users) over(order by searches rows between unbounded preceding and current row) as cumm_sum,
sum(num_users) over(order by searches rows between unbounded preceding and unbounded following) as total
from search_frequency) t
) t1
where (t1.pos1 > t1.prev_cumm_sum and t1.pos1 <= t1.cumm_sum) or (t1.pos2 > t1.prev_cumm_sum and t1.pos2 <= t1.cumm_sum);
```

```
select round(avg(t1.searches),1) as median
from
```

		median newdecimal
	1	2.5

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 possibletransitions 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

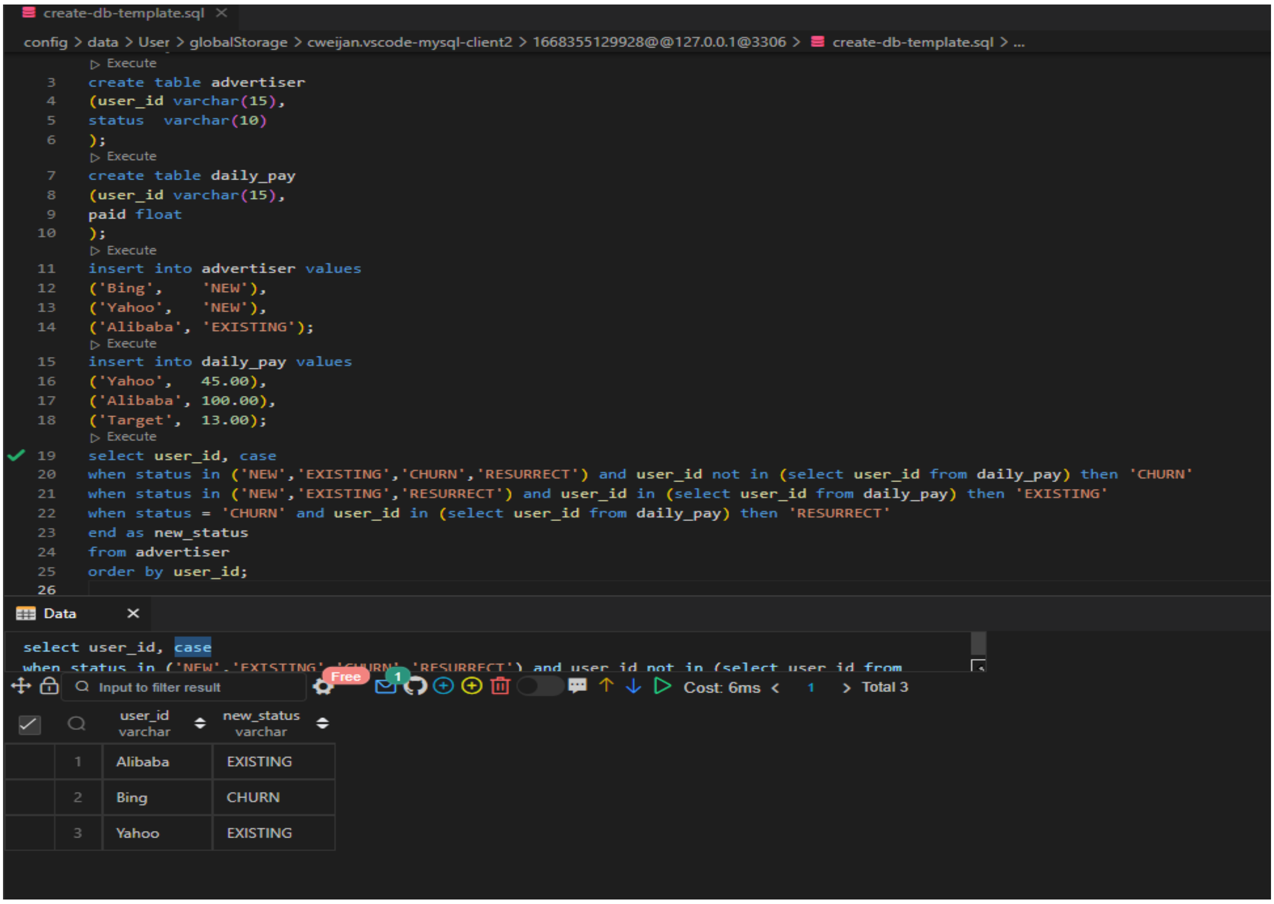
1. Row 2, 4, 6, 8: As long as the user has not paid on day T, the end status is updated to CHURN regardless of the previous status.
2. Row 1, 3, 5, 7: When the user paid on day T, the end status is updated to either EXISTING or RESURRECT, depending on their previous state. RESURRECT is only possible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

Solution:

Conditions used in case when:

Previous Status	Condition	Next Status
New, Existing, Churn, Resurrect	Didn't pay on day T	Churn
New, Existing, Resurrect	Paid on day T	Existing
Churn	Paid on day T	Resurrect

```
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;
```



create-db-template.sql ×
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-template.sql > ...
▷ Execute
Data ×
Input to filter result
Free
1
Cost: 6ms < 1 > Total 3

```sql
create table advertiser
(user_id varchar(15),
status  varchar(10)
);
create table daily_pay
(user_id varchar(15),
paid float
);
insert into advertiser values
('Bing',    'NEW'),
('Yahoo',   'NEW'),
('Alibaba', 'EXISTING');
insert into daily_pay values
('Yahoo',   45.00),
('Alibaba', 100.00),
('Target',  13.00);
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;
```

	user_id varchar	new_status varchar
1	Alibaba	EXISTING
2	Bing	CHURN
3	Yahoo	EXISTING

Q124. Amazon Web Services (AWS) is powered by fleets of servers. Senior management has requested data-driven solutions to optimise server usage.

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

Level - Hard
Hint-

1. Calculate individual uptimes

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

Assumptions:

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

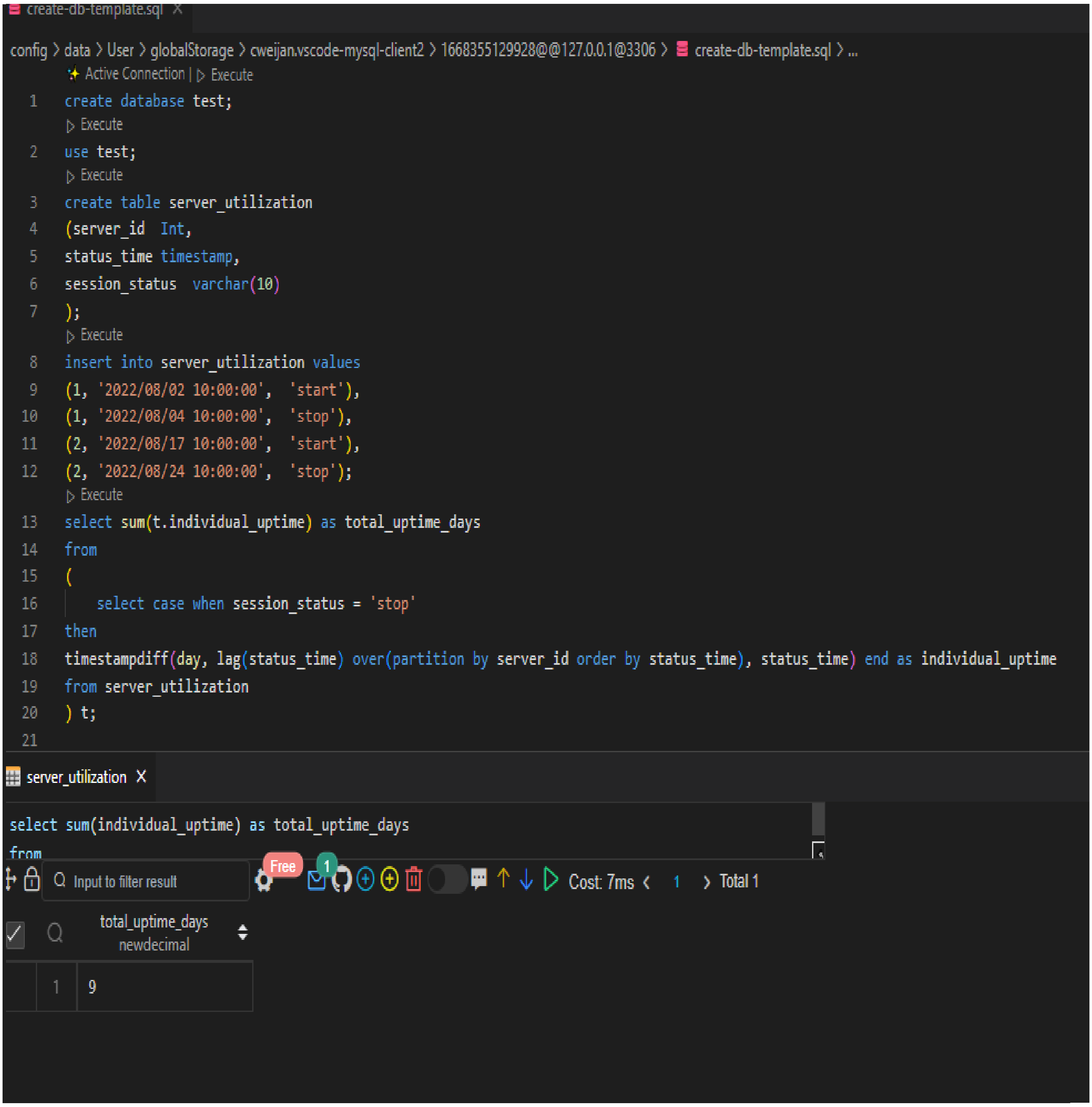
server_utilization Table:

Column Name	Type
server_id	integer
status_time	timestamp
session_status	string

server_utilization Example Input:

server_id	status_time	session_status
1	08/02/2022 10:00:00	start
1	08/04/2022 10:00:00	stop
2	08/17/2022 10:00:00	start
2	08/24/2022 10:00:00	stop

```
Solution:
select sum(t.individual_uptime) as total_uptime_days
from
(
    select case when session_status = '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;
```



create-db-template.sql ×
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-template.sql > ...
Active Connection | ▷ Execute
▷ Execute
server_utilization ×
Input to filter result
Free
1
Cost: 7ms
1
Total 1

```sql
create database test;
use test;
create table server_utilization
(server_id  Int,
status_time timestamp,
session_status  varchar(10)
);
insert into server_utilization values
(1, '2022/08/02 10:00:00',  'start'),
(1, '2022/08/04 10:00:00',  'stop'),
(2, '2022/08/17 10:00:00',  'start'),
(2, '2022/08/24 10:00:00',  'stop');
select sum(t.individual_uptime) as total_uptime_days
from
(
    select case when session_status = '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;

```

```
select sum(individual_uptime) as total_uptime_days
from
```

```
   total_uptime_days
   newdecimal
1  9
```

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

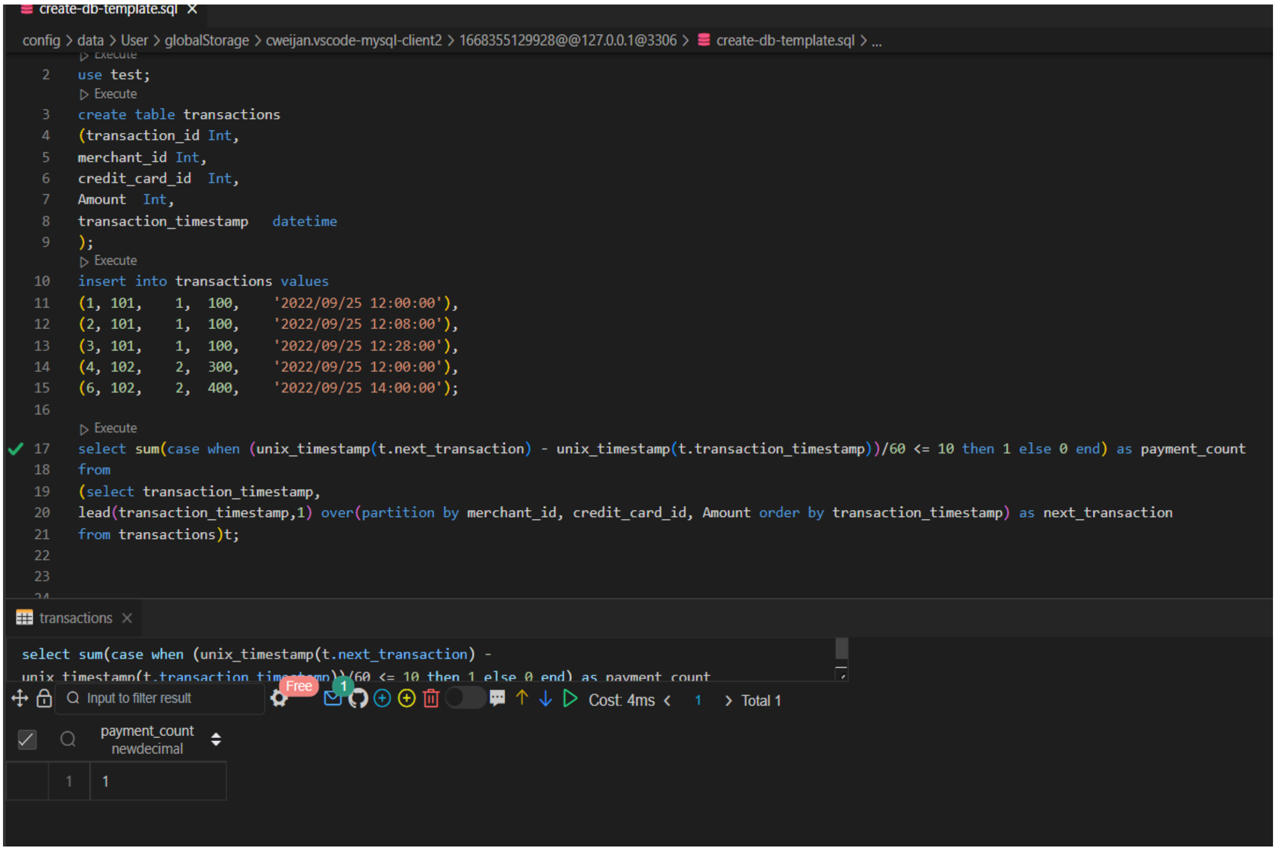
transactions Example Input:

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

```
Solution:
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;
```


create-db-template.sql ×
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-template.sql > ...
▷ Execute
transactions ×
Input to filter result
Free
Cost: 4ms
Total 1

```sql
use test;
create table transactions
(transaction_id Int,
merchant_id Int,
credit_card_id  Int,
Amount  Int,
transaction_timestamp   datetime
);
insert into transactions values
(1, 101,    1,  100,    '2022/09/25 12:00:00'),
(2, 101,    1,  100,    '2022/09/25 12:08:00'),
(3, 101,    1,  100,    '2022/09/25 12:28:00'),
(4, 102,    2,  300,    '2022/09/25 12:00:00'),
(6, 102,    2,  400,    '2022/09/25 14:00:00');

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;
```

```sql
select sum(case when (unix_timestamp(t.next_transaction) -
unix_timestamp(t.transaction_timestamp))/60 <= 10 then 1 else 0 end) as payment_count
```

	payment_count newdecimal
1	1

Q126. DoorDash's Growth Team is trying to make sure new users (those who are making orders in their first 14 days) have a great experience on all their orders in their 2 weeks on the platform. Unfortunately, many deliveries are being messed up because:

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

Hint- Use Where Clause and joins

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

orders Table:

Column Name	Type
order_id	integer
customer_id	integer
trip_id	integer
status	string ('completed successfully', 'completed incorrectly', 'never received')
order_timestamp	timestamp

orders Example Input:

order_id	customer_id	trip_id	status	order_timestamp
727424	8472	100463	completed successfully	06/05/2022 09:12:00
242513	2341	100482	completed incorrectly	06/05/2022 14:40:00
141367	1314	100362	completed incorrectly	06/07/2022 15:03:00
582193	5421	100657	never_received	07/07/2022 15:22:00
253613	1314	100213	completed successfully	06/12/2022 13:43:00

trips Table:

Column Name	Type
dasher_id	integer
trip_id	integer
estimated_delivery_timestamp	timestamp
actual_delivery_timestamp	timestamp

trips Example Input:

dasher_id	trip_id	estimated_delivery_timestamp	actual_delivery_timestamp
101	100463	06/05/2022 09:42:00	06/05/2022 09:38:00
102	100482	06/05/2022 15:10:00	06/05/2022 15:46:00
101	100362	06/07/2022 15:33:00	06/07/2022 16:45:00
102	100657	07/07/2022 15:52:00	-
103	100213	06/12/2022 14:13:00	06/12/2022 14:10:00

customers Table:

Column Name	Type
customer_id	integer
signup_timestamp	timestamp

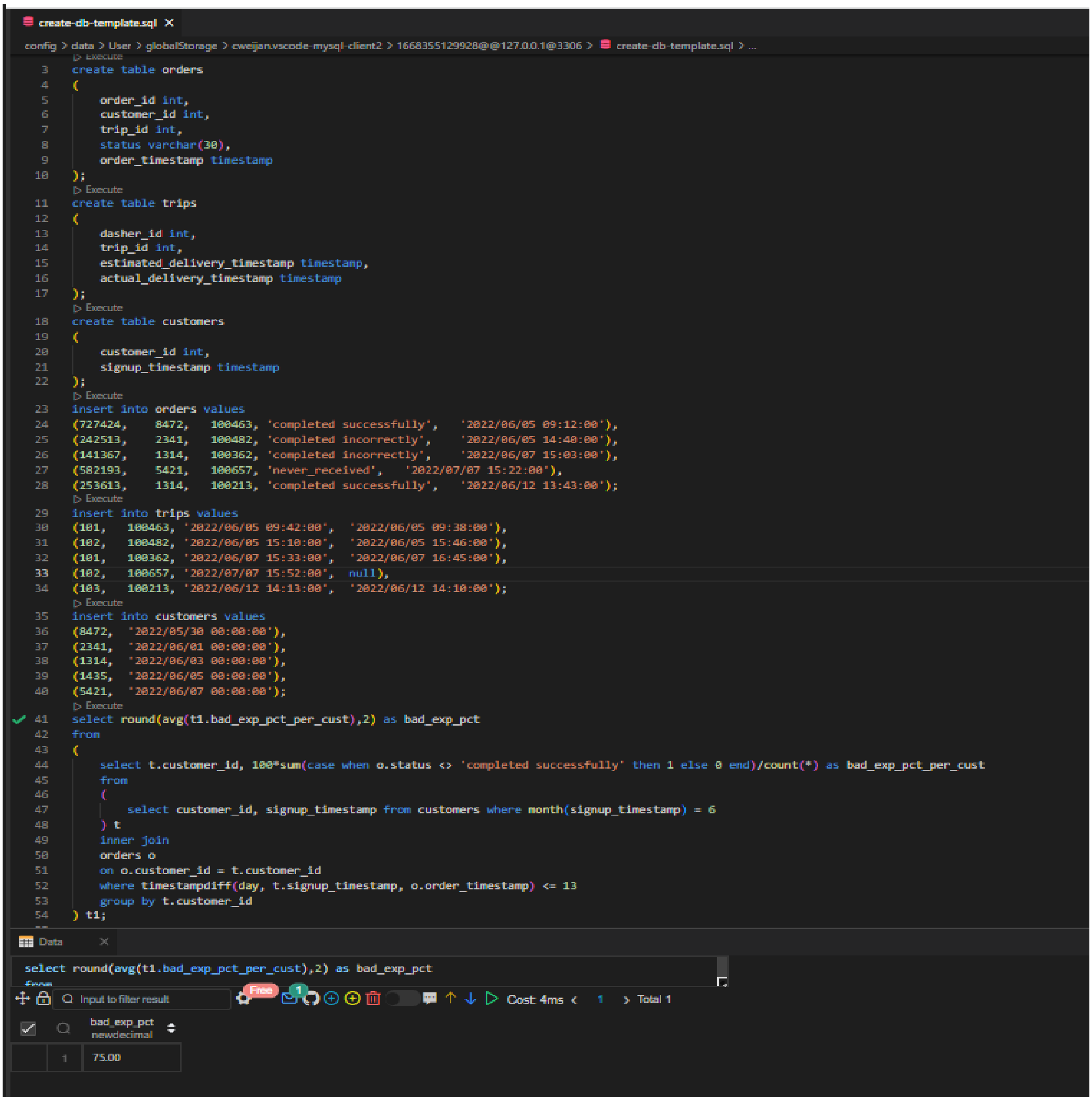
customers Example Input:

customer_id	signup_timestamp
8472	05/30/2022 00:00:00
2341	06/01/2022 00:00:00
1314	06/03/2022 00:00:00
1435	06/05/2022 00:00:00
5421	06/07/2022 00:00:00

Example Output:

bad_experience_pct
75.00

```
Solution:
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;
```


create-db-template.sql
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-template.sql > ...
▷ Execute
Data
Input to filter result
Free
Cost: 4ms
Total 1

```
create table orders
(
    order_id int,
    customer_id int,
    trip_id int,
    status varchar(30),
    order_timestamp timestamp
);
create table trips
(
    dasher_id int,
    trip_id int,
    estimated_delivery_timestamp timestamp,
    actual_delivery_timestamp timestamp
);
create table customers
(
    customer_id int,
    signup_timestamp timestamp
);
insert into orders values
(727424,    8472,   100463, 'completed successfully',   '2022/06/05 09:12:00'),
(242513,    2341,   100482, 'completed incorrectly',    '2022/06/05 14:40:00'),
(141367,    1314,   100362, 'completed incorrectly',    '2022/06/07 15:03:00'),
(582193,    5421,   100657, 'never_received',   '2022/07/07 15:22:00'),
(253613,    1314,   100213, 'completed successfully',   '2022/06/12 13:43:00');
insert into trips values
(101,   100463, '2022/06/05 09:42:00',  '2022/06/05 09:38:00'),
(102,   100482, '2022/06/05 15:10:00',  '2022/06/05 15:46:00'),
(101,   100362, '2022/06/07 15:33:00',  '2022/06/07 16:45:00'),
(102,   100657, '2022/07/07 15:52:00',  null),
(103,   100213, '2022/06/12 14:13:00',  '2022/06/12 14:10:00');
insert into customers values
(8472,  '2022/05/30 00:00:00'),
(2341,  '2022/06/01 00:00:00'),
(1314,  '2022/06/03 00:00:00'),
(1435,  '2022/06/05 00:00:00'),
(5421,  '2022/06/07 00:00:00');
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;
```

```
select round(avg(t1.bad_exp_pct_per_cust),2) as bad_exp_pct
```

	bad_exp_pct newdecimal
1	75.00

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

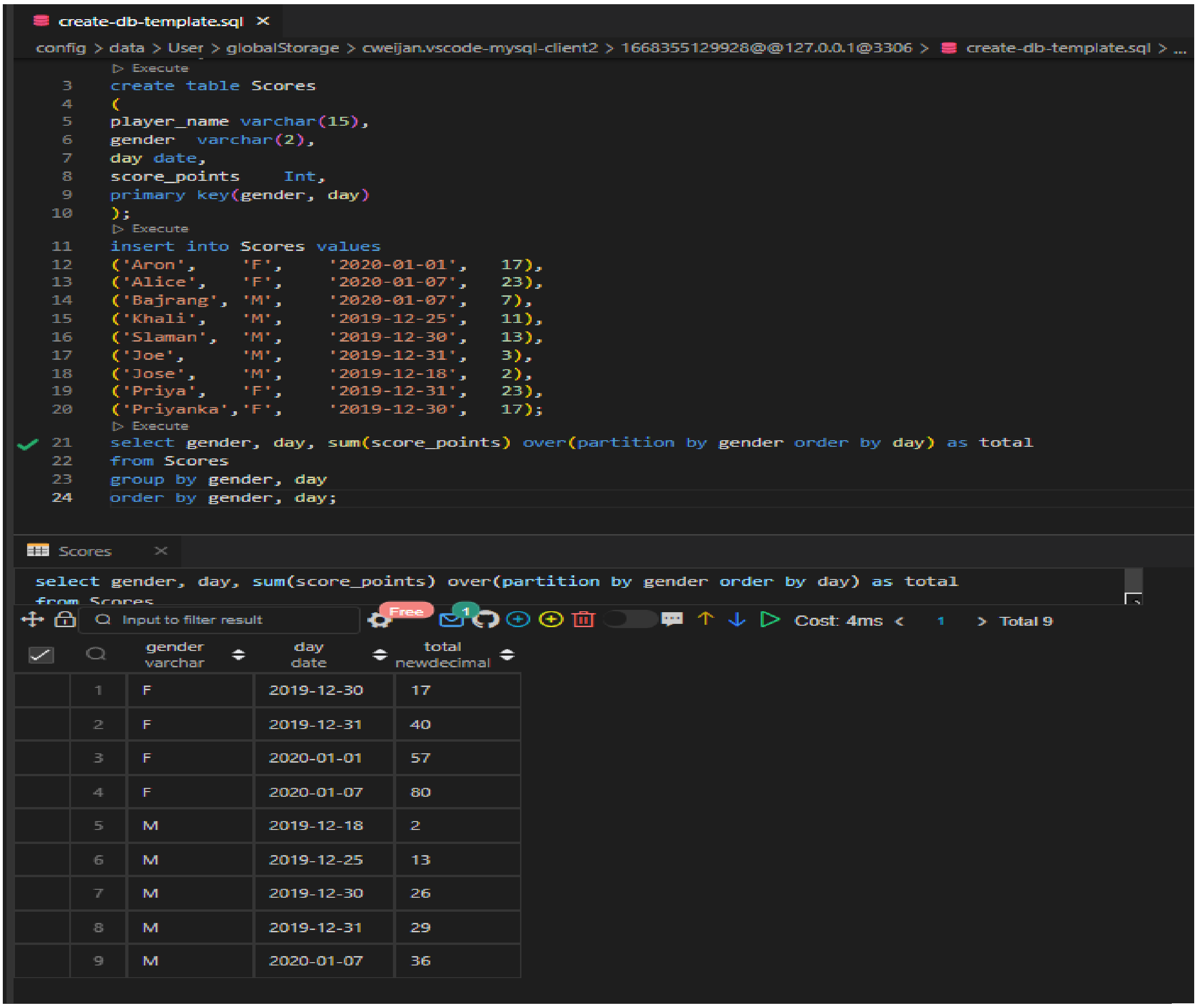
M	2019-12-30	26
M	2019-12-31	29
M	2020-01-07	36

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.

```
Solution:
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;
```


create-db-template.sql
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-template.sql > ...
▷ Execute
Scores
Input to filter result
Free
Cost: 4ms
1
Total 9

```sql
create table Scores
(
player_name varchar(15),
gender  varchar(2),
day date,
score_points    Int,
primary key(gender, day)
);
insert into Scores values
('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);
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;
```

```sql
select gender, day, sum(score_points) over(partition by gender order by day) as total
from Scores
```

	gender varchar	day date	total newdecimal
1	F	2019-12-30	17
2	F	2019-12-31	40
3	F	2020-01-01	57
4	F	2020-01-07	80
5	M	2019-12-18	2
6	M	2019-12-25	13
7	M	2019-12-30	26
8	M	2019-12-31	29
9	M	2020-01-07	36

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

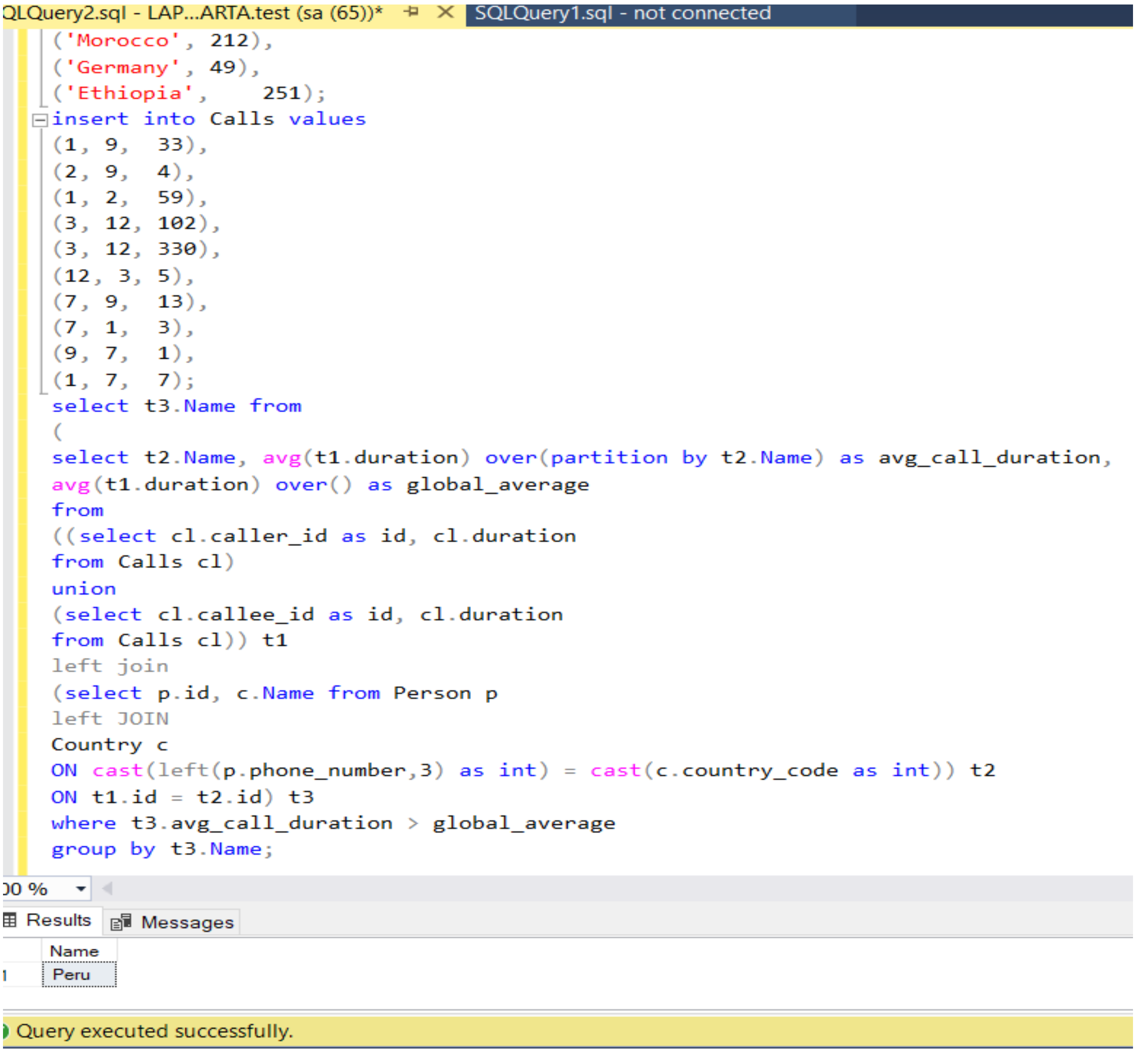
1	2	59
3	12	102
3	12	330
12	3	5
7	9	13
7	1	3
9	7	1
1	7	7

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.

```
Solution:
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;
```


QLQuery2.sql - LAP...ARTA.test (sa (65))*
SQLQuery1.sql - not connected
)0 %
Results
Messages

```
('Morocco', 212),
('Germany', 49),
('Ethiopia',    251);
insert into Calls values
(1, 9,  33),
(2, 9,  4),
(1, 2,  59),
(3, 12, 102),
(3, 12, 330),
(12, 3, 5),
(7, 9,  13),
(7, 1,  3),
(9, 7,  1),
(1, 7,  7);
select 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;
```

```
   Name
1  Peru
```

Query executed successfully.

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:

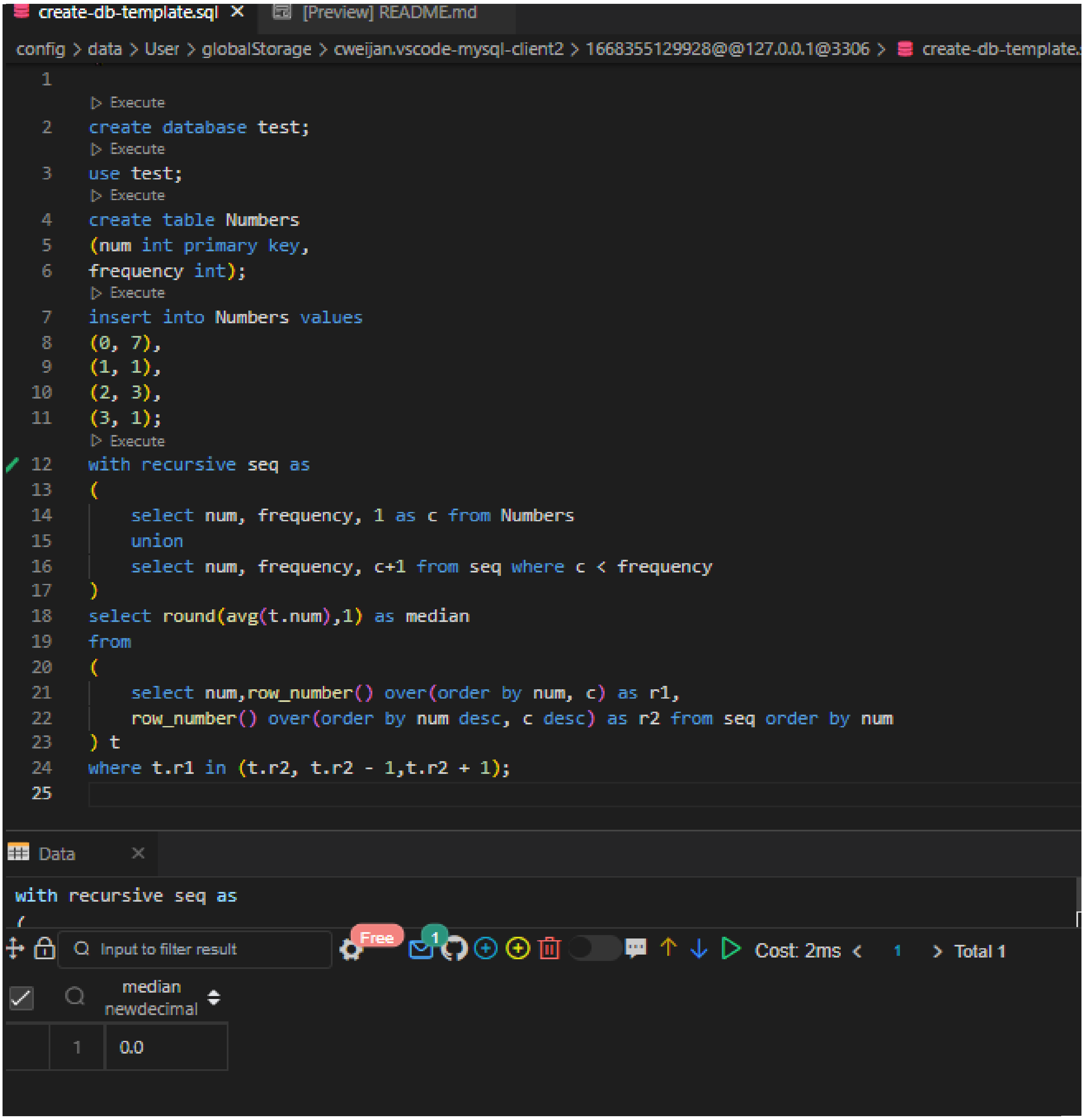
num	frequency
0	7
1	1
2	3
3	1

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.

```
Solution:
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);
```

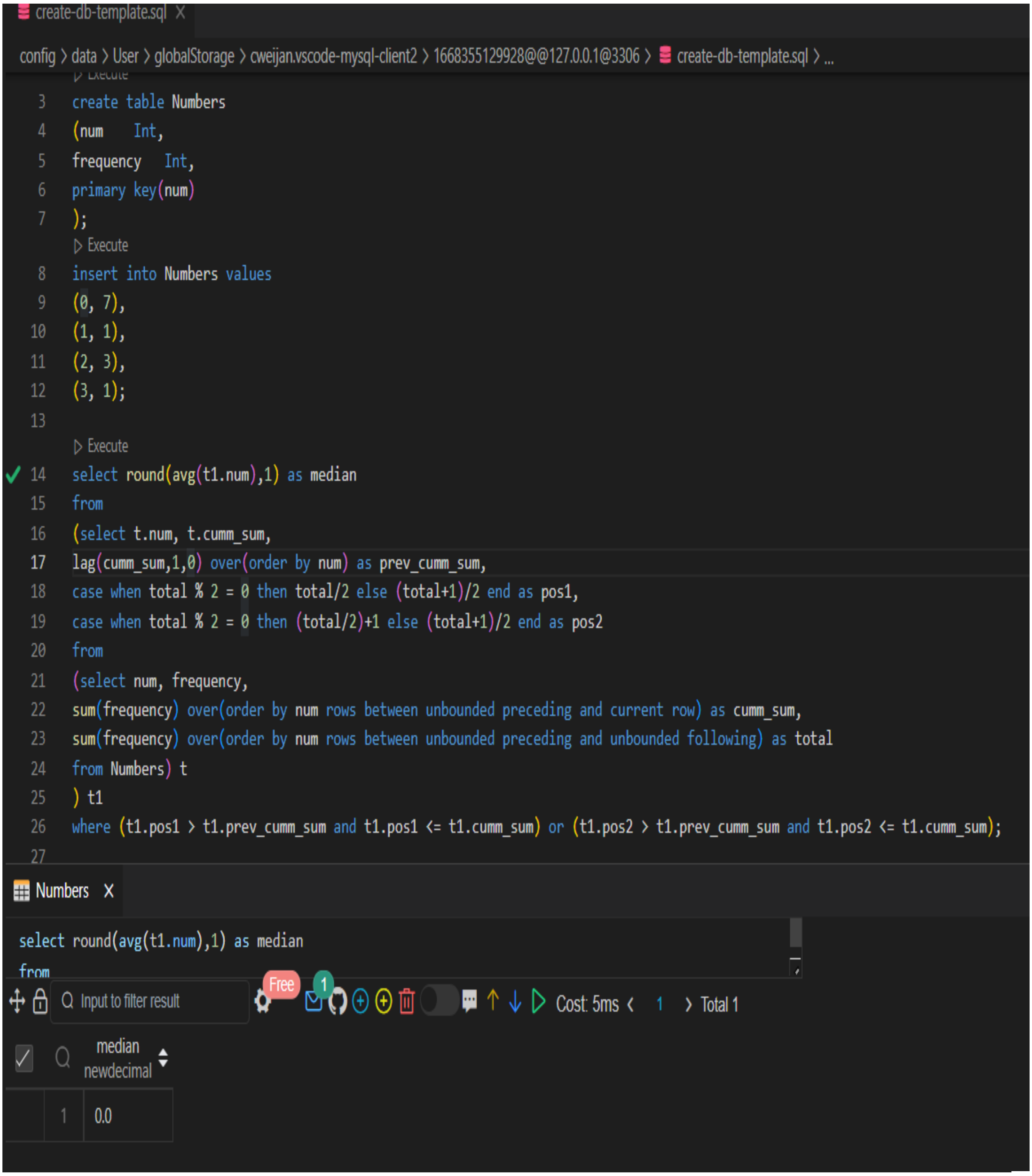

create-db-template.sql
[Preview] README.md
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-template.s
▷ Execute
Data
Input to filter result
Free
Cost: 2ms
Total 1

```sql

create database test;
use test;
create table Numbers
(num int primary key,
frequency int);
insert into Numbers values
(0, 7),
(1, 1),
(2, 3),
(3, 1);
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);

```

```
with recursive seq as
(
```

		median newdecimal
	1	0.0

```
Solution: using cumulative sum
select round(avg(t1.num),1) as median
from
(select t.num, t.cumm_sum,
lag(cumm_sum,1,0) over(order by num) as prev_cumm_sum,
case when total % 2 = 0 then total/2 else (total+1)/2 end as pos1,
case when total % 2 = 0 then (total/2)+1 else (total+1)/2 end as pos2
from
(select num, frequency,
sum(frequency) over(order by num rows between unbounded preceding and current
row) as cumm_sum,
sum(frequency) over(order by num rows between unbounded preceding and unbounded
following) as total

from Numbers) t
) t1
where (t1.pos1 > t1.prev_cumm_sum and t1.pos1 <= t1.cumm_sum) or (t1.pos2 >
t1.prev_cumm_sum and t1.pos2 <= t1.cumm_sum);
```


create-db-template.sql X
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-template.sql > ...
▷ Execute

```
create table Numbers
(num    Int,
frequency   Int,
primary key(num)
);
```

```
insert into Numbers values
(0, 7),
(1, 1),
(2, 3),
(3, 1);

```

```
select round(avg(t1.num),1) as median
from
(select t.num, t.cumm_sum,
lag(cumm_sum,1,0) over(order by num) as prev_cumm_sum,
case when total % 2 = 0 then total/2 else (total+1)/2 end as pos1,
case when total % 2 = 0 then (total/2)+1 else (total+1)/2 end as pos2
from
(select num, frequency,
sum(frequency) over(order by num rows between unbounded preceding and current row) as cumm_sum,
sum(frequency) over(order by num rows between unbounded preceding and unbounded following) as total
from Numbers) t
) t1
where (t1.pos1 > t1.prev_cumm_sum and t1.pos1 <= t1.cumm_sum) or (t1.pos2 > t1.prev_cumm_sum and t1.pos2 <= t1.cumm_sum);

```

```
select round(avg(t1.num),1) as median
from
```

	median newdecimal
1	0.0

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:

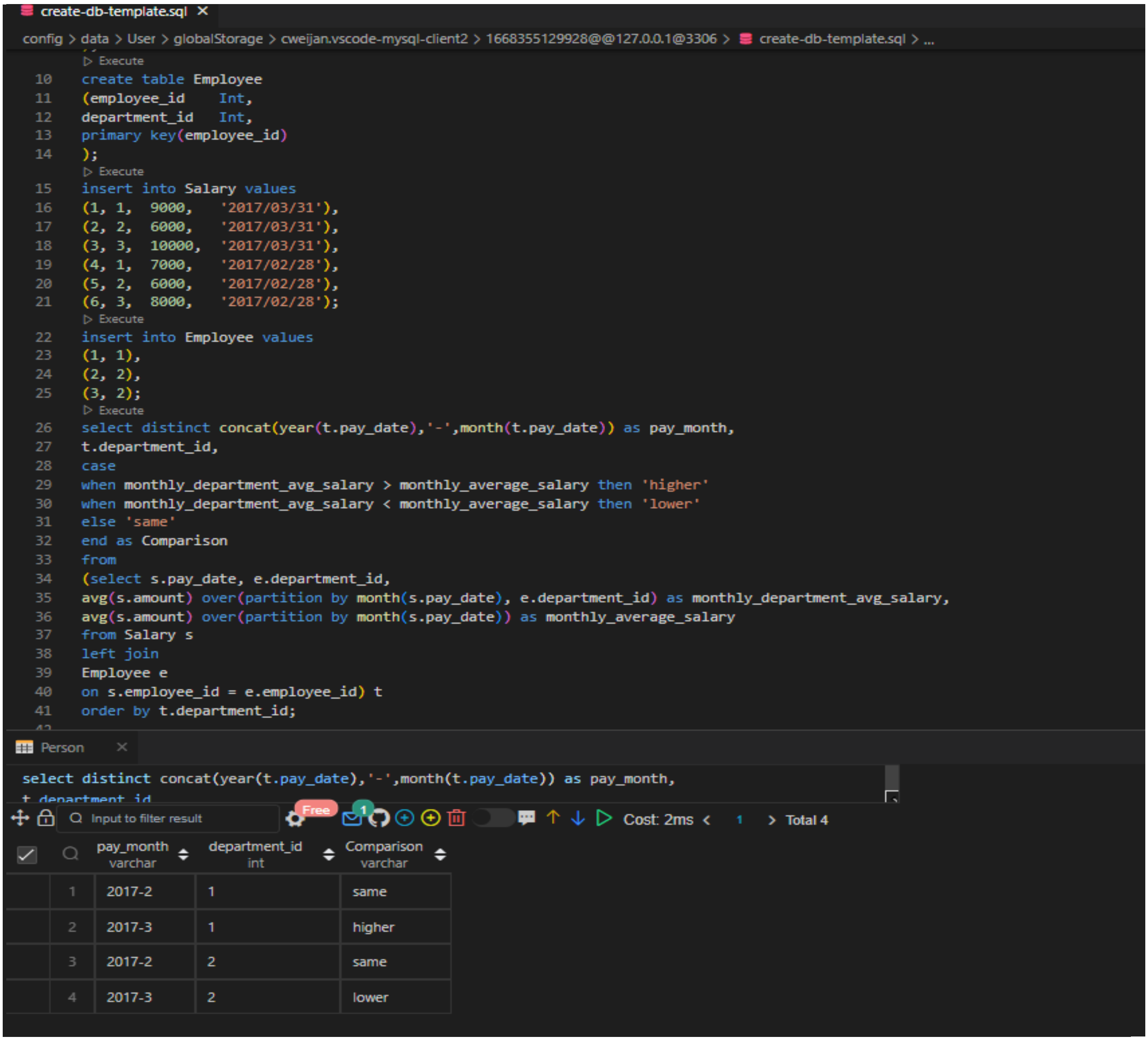
pay_month	department_id	comparison
2017-02	1	same
2017-03	1	higher
2017-02	2	same
2017-03	2	lower

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.

```
Solution:
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,
```

```sql
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;
```


create-db-template.sql
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-template.sql > ...
▷ Execute
Person
Input to filter result
Free
Cost: 2ms
1
Total 4

```sql
create table Employee
(employee_id    Int,
department_id   Int,
primary key(employee_id)
);
insert into Salary values
(1, 1,  9000,   '2017/03/31'),
(2, 2,  6000,   '2017/03/31'),
(3, 3,  10000,  '2017/03/31'),
(4, 1,  7000,   '2017/02/28'),
(5, 2,  6000,   '2017/02/28'),
(6, 3,  8000,   '2017/02/28');
insert into Employee values
(1, 1),
(2, 2),
(3, 2);
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;
```

```sql
select distinct concat(year(t.pay_date),'-',month(t.pay_date)) as pay_month,
t.department_id
```

	pay_month varchar	department_id int	Comparison varchar
1	2017-2	1	same
2	2017-3	1	higher
3	2017-2	2	same
4	2017-3	2	lower

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:

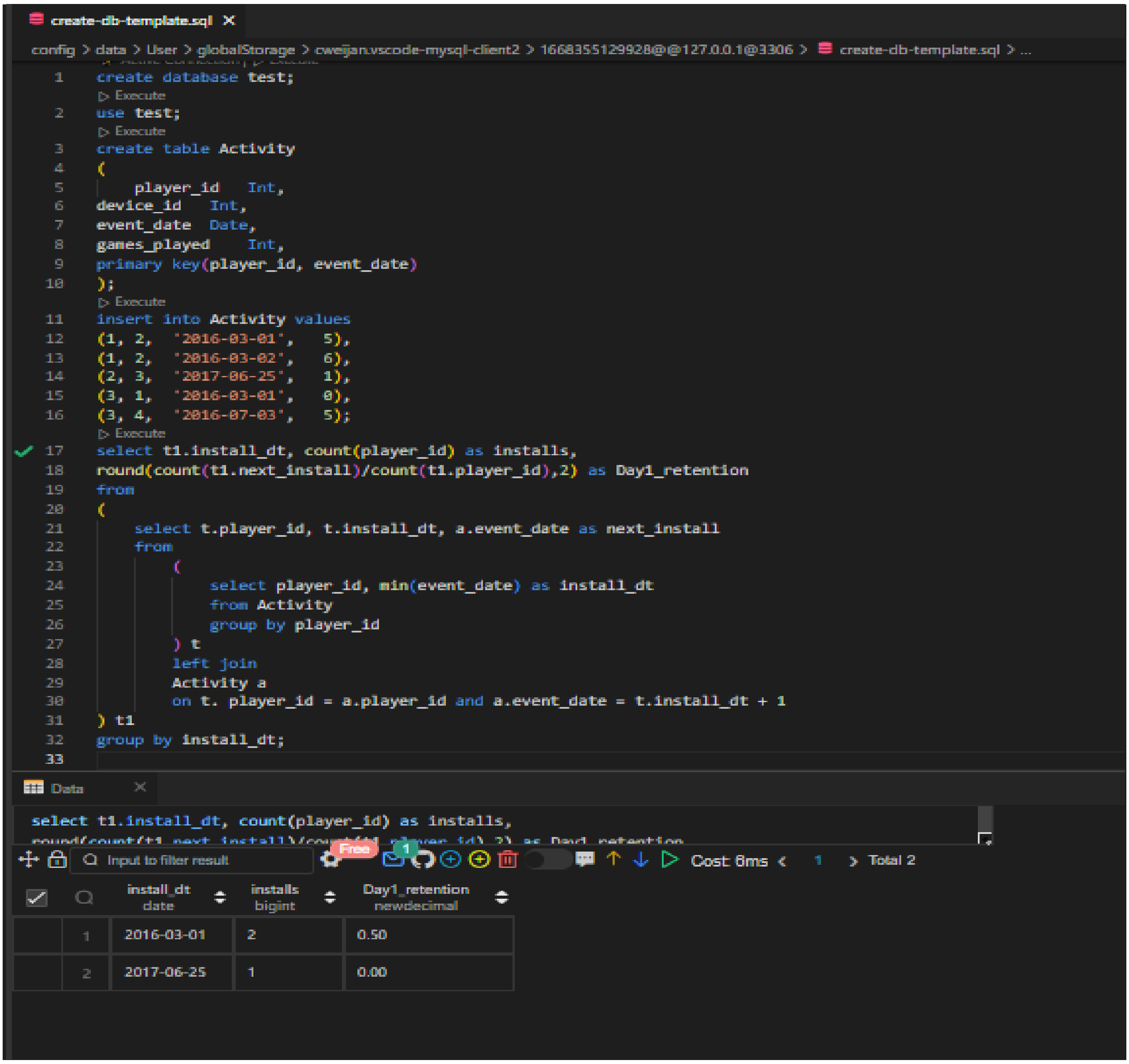
player_id	device_id	event_date	games_played
1	2	2016-03-01	5
1	2	2016-03-02	6
2	3	2017-06-25	1
3	1	2016-03-01	0
3	4	2016-07-03	5

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

```
Solution:
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;
```


create-db-template.sql
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-template.sql > ...
▷ Execute
Data
Input to filter result
Free
Cost: 6ms
Total 2

```
create database test;
use test;
create table Activity
(
    player_id   Int,
device_id   Int,
event_date  Date,
games_played    Int,
primary key(player_id, event_date)
);
insert into Activity values
(1, 2,  '2016-03-01',   5),
(1, 2,  '2016-03-02',   6),
(2, 3,  '2017-06-25',   1),
(3, 1,  '2016-03-01',   0),
(3, 4,  '2016-07-03',   5);
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;
```

```
select t1.install_dt, count(player_id) as installs,
```

	install_dt date	installs bigint	Day1_retention newdecimal
1	2016-03-01	2	0.50
2	2017-06-25	1	0.00

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

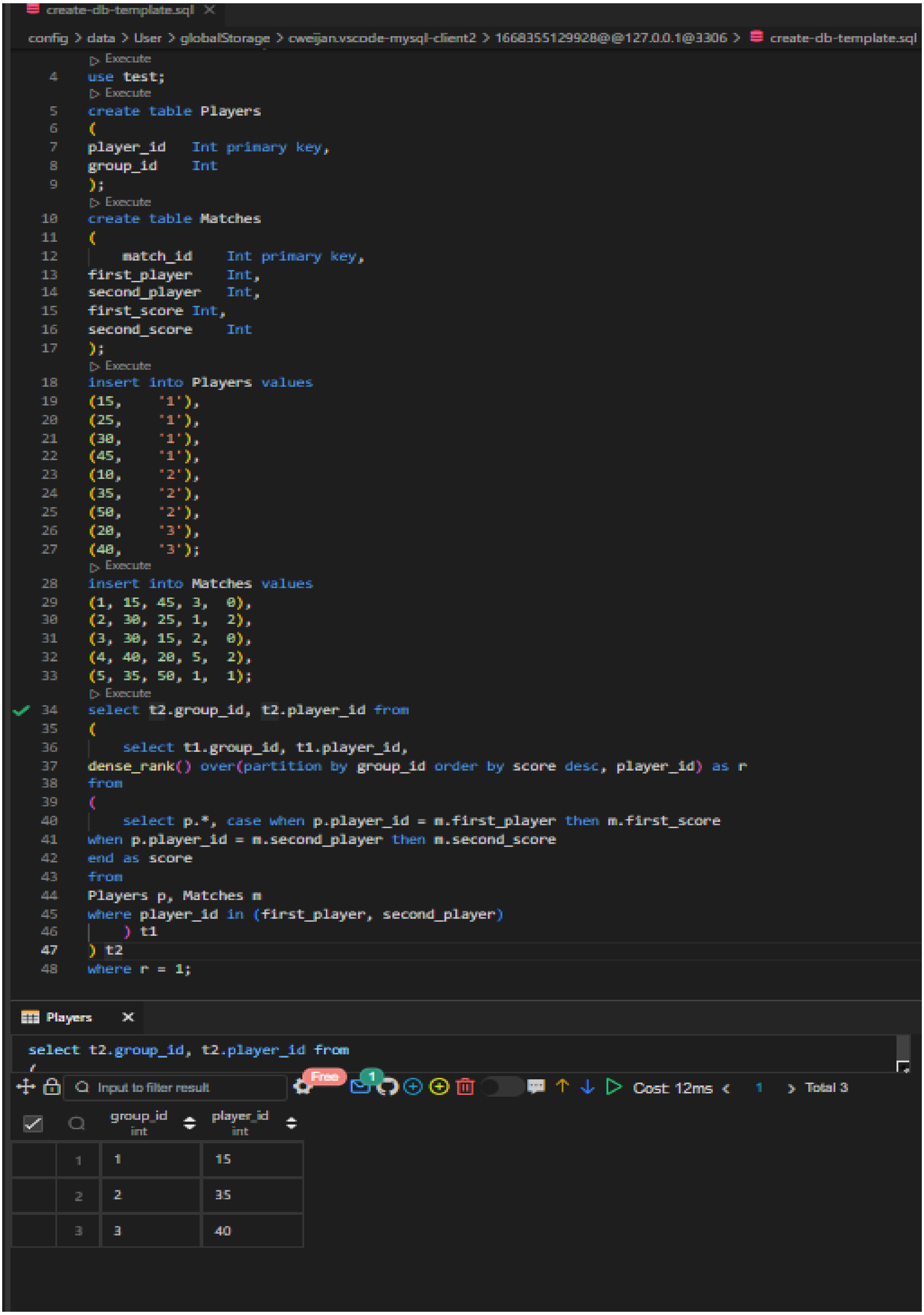
Matches table:

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

```
Solution:
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;
```


create-db-template.sql
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-template.sql
▷ Execute
Players
Input to filter result
Free
Cost: 12ms
1
Total 3

```
use test;
create table Players
(
player_id   Int primary key,
group_id    Int
);
create table Matches
(
    match_id    Int primary key,
first_player    Int,
second_player   Int,
first_score Int,
second_score    Int
);
insert into Players values
(15,    '1'),
(25,    '1'),
(30,    '1'),
(45,    '1'),
(10,    '2'),
(35,    '2'),
(50,    '2'),
(20,    '3'),
(40,    '3');
insert into Matches values
(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);
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;
```

```
select t2.group_id, t2.player_id from
(
```

	group_id int	player_id int
1	1	15
2	2	35
3	3	40

Q133.

Table: Student

Column Name	Type
student_id	int
student_name	varchar

student_id is the primary key for this table.
student_name is the name of the student.

Table: Exam

Column Name	Type
exam_id	int
student_id	int
score	int

(exam_id, student_id) is the primary key for this table.
Each row of this table indicates that the student with student_id had a score points in the exam with id exam_id.

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

Write an SQL query to report the students (student_id, student_name) being quiet in all exams. Do not return the student who has never taken any exam.
Return the result table ordered by student_id.
The query result format is in the following example.

Input:
Student table:

student_id	student_name
1	Daniel
2	Jade
3	Stella
4	Jonathan
5	Will

Exam table:

exam_id	student_id	score
10	1	70
10	2	80
10	3	90
20	1	80
30	1	70
30	3	80
30	4	90
40	1	60
40	2	70
40	4	80

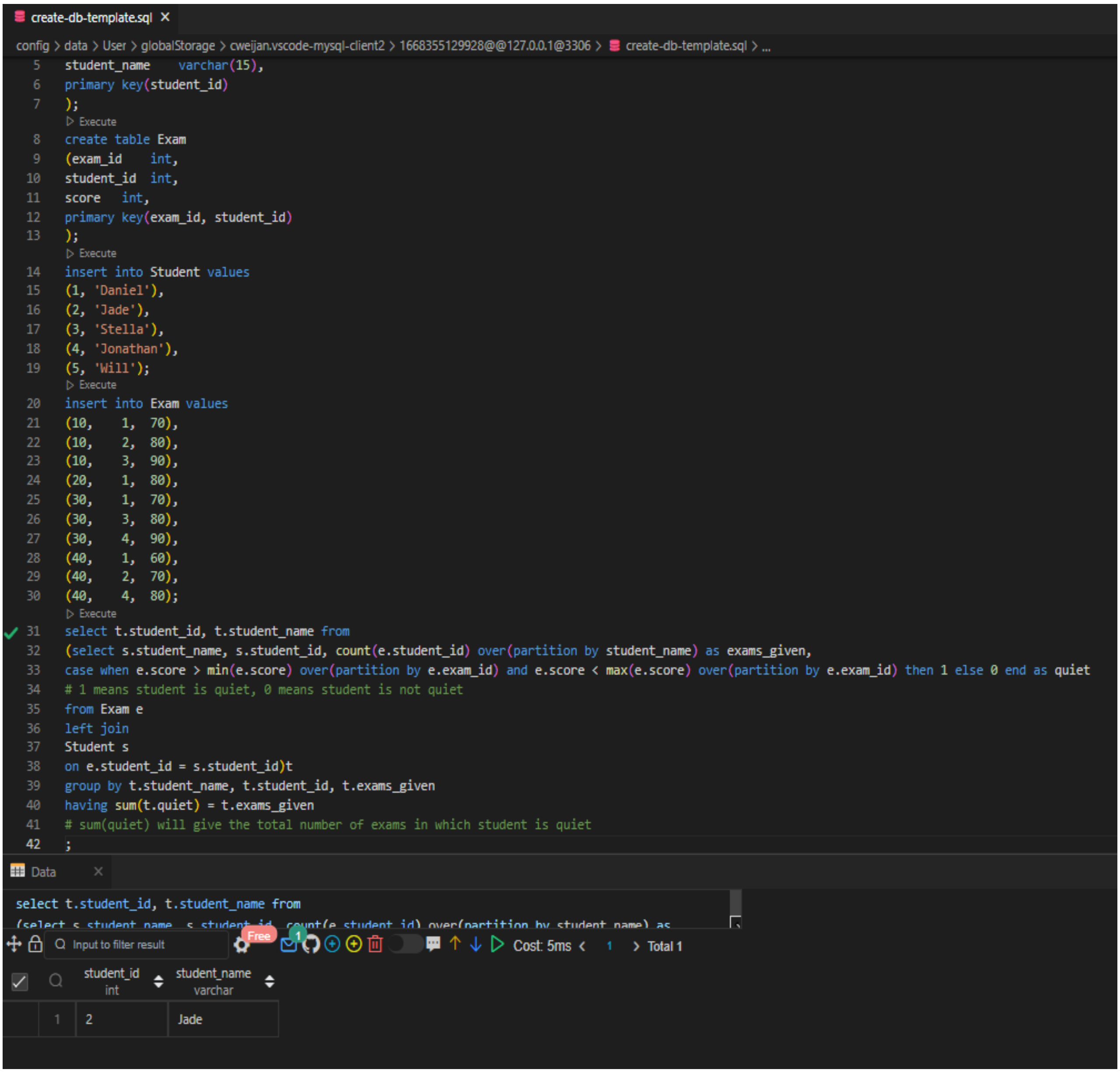
Output:

student_id	student_name
2	Jade

Explanation:
For exam 1: Student 1 and 3 hold the lowest and high scores respectively.For exam 2: Student 1 holds both the highest and lowest score.
For exam 3 and 4: Student 1 and 4 hold the lowest and high scores respectively.Students 2 and 5 have never got the highest or lowest in any of the exams.
Since student 5 is not taking any exam, he is excluded from the result.So, we only return the information of Student 2.

Solution:

```
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
# 1 means student is quiet, 0 means student is not 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
# sum(quiet) will give the total number of exams in which student is quiet
```


create-db-template.sql ×
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-template.sql > ...
Execute
Data ×
Input to filter result
Free
Cost: 5ms < 1 > Total 1

```
student_name    varchar(15),
primary key(student_id)
);
create table Exam
(exam_id    int,
student_id  int,
score   int,
primary key(exam_id, student_id)
);
insert into Student values
(1, 'Daniel'),
(2, 'Jade'),
(3, 'Stella'),
(4, 'Jonathan'),
(5, 'Will');
insert into Exam values
(10,    1,  70),
(10,    2,  80),
(10,    3,  90),
(20,    1,  80),
(30,    1,  70),
(30,    3,  80),
(30,    4,  90),
(40,    1,  60),
(40,    2,  70),
(40,    4,  80);
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
# 1 means student is quiet, 0 means student is not 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
# sum(quiet) will give the total number of exams in which student is quiet
;
```

```
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
```

	student_id int	student_name varchar
1	2	Jade

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

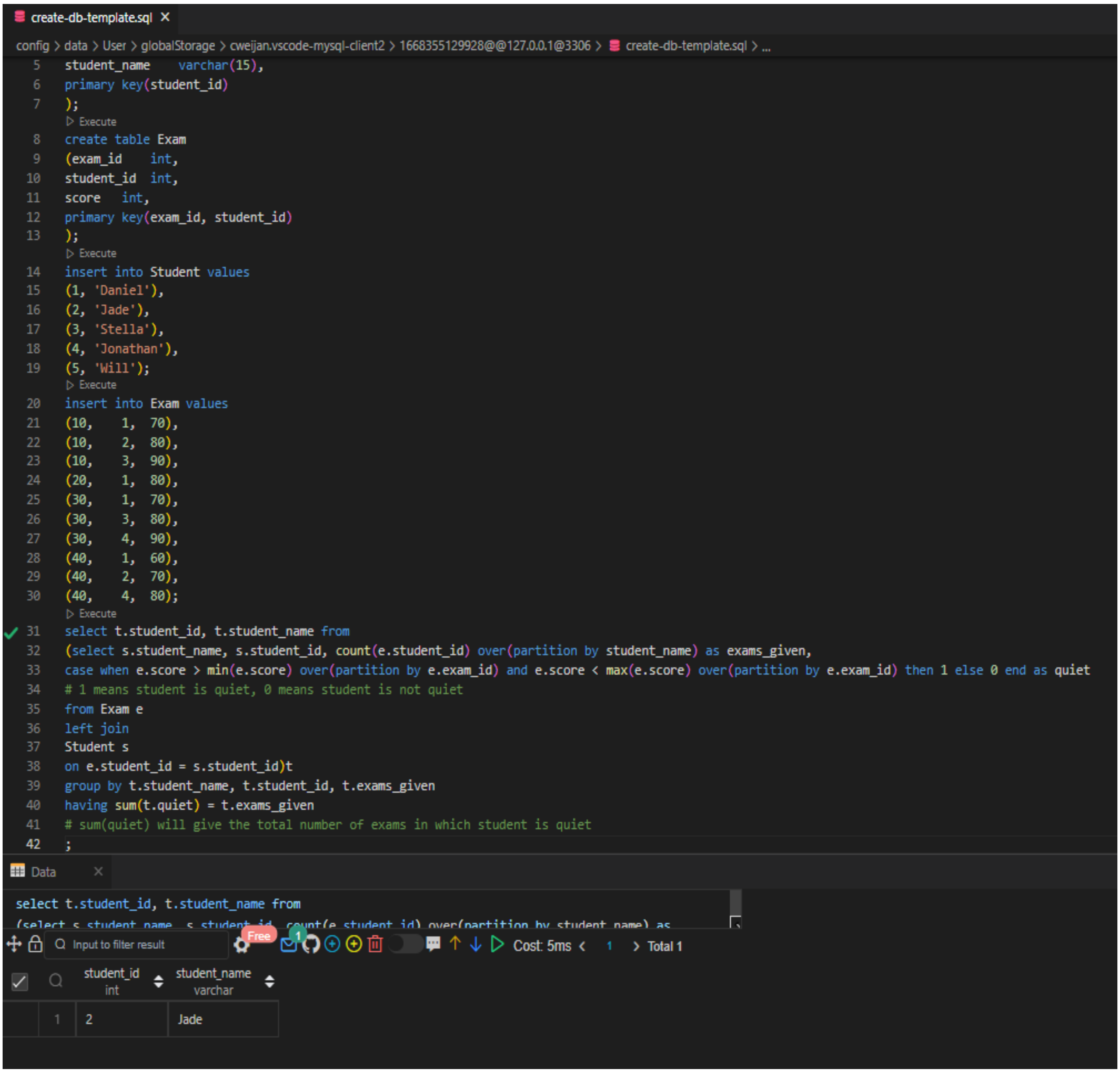
30	1	70
30	3	80
30	4	90
40	1	60
40	2	70
40	4	80

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.

```
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
# 1 means student is quiet, 0 means student is not 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
# sum(quiet) will give the total number of exams in which student is quiet
```


create-db-template.sql ×
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-template.sql > ...
▷ Execute
Data ×
Input to filter result
Free
Cost: 5ms
1
Total 1

```sql
student_name     varchar(15),
primary key(student_id)
);
create table Exam
(exam_id    int,
student_id  int,
score   int,
primary key(exam_id, student_id)
);
insert into Student values
(1, 'Daniel'),
(2, 'Jade'),
(3, 'Stella'),
(4, 'Jonathan'),
(5, 'Will');
insert into Exam values
(10,    1,  70),
(10,    2,  80),
(10,    3,  90),
(20,    1,  80),
(30,    1,  70),
(30,    3,  80),
(30,    4,  90),
(40,    1,  60),
(40,    2,  70),
(40,    4,  80);
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
# 1 means student is quiet, 0 means student is not 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
# sum(quiet) will give the total number of exams in which student is quiet
;
```

```sql
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
```

	student_id int	student_name varchar
1	2	Jade

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

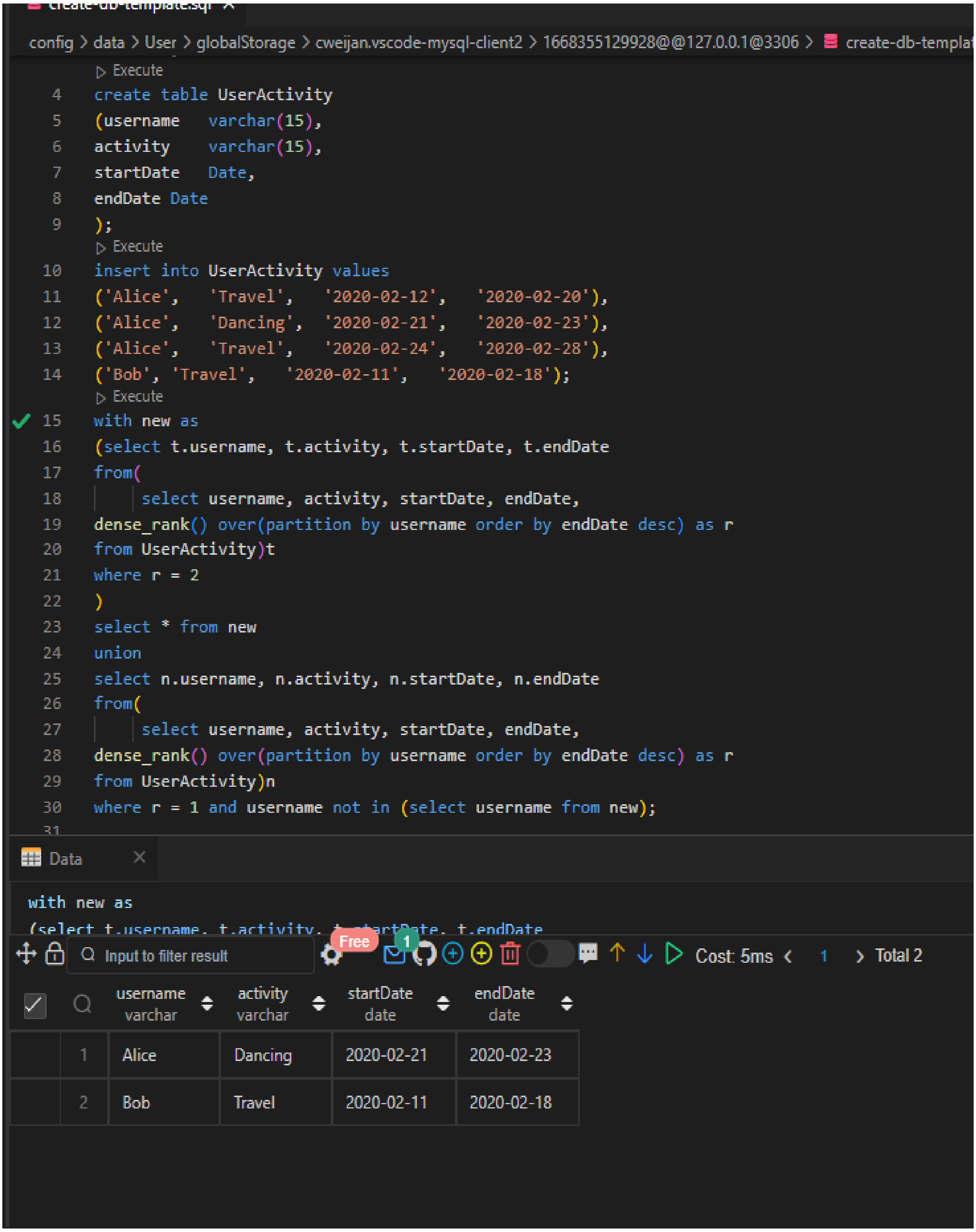
Alice	Dancing	2020-02-21	2020-02-23
Alice	Travel	2020-02-24	2020-02-28
Bob	Travel	2020-02-11	2020-02-18

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.

```
Solution:
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);
```



config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-templat

```
▷ Execute
create table UserActivity
(username   varchar(15),
activity    varchar(15),
startDate   Date,
endDate Date
);
▷ Execute
insert into UserActivity values
('Alice',   'Travel',   '2020-02-12',   '2020-02-20'),
('Alice',   'Dancing',  '2020-02-21',   '2020-02-23'),
('Alice',   'Travel',   '2020-02-24',   '2020-02-28'),
('Bob', 'Travel',   '2020-02-11',   '2020-02-18');
▷ Execute
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);
```

Data

```
with new as
(select t.username, t.activity, t.startDate, t.endDate
```

		username varchar	activity varchar	startDate date	endDate date
	1	Alice	Dancing	2020-02-21	2020-02-23
	2	Bob	Travel	2020-02-11	2020-02-18

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:

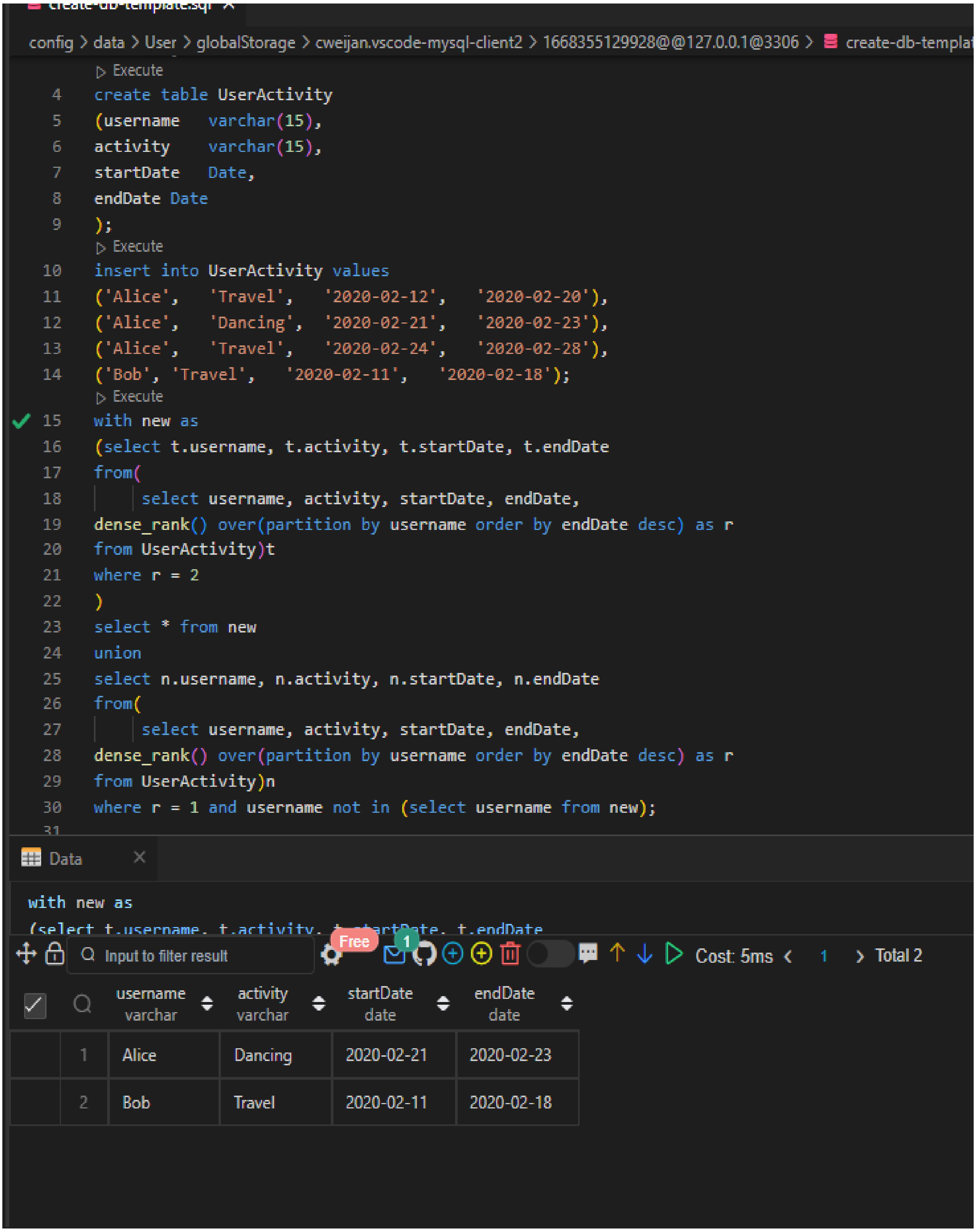
username	activity	startDate	endDate
Alice	Travel	2020-02-12	2020-02-20
Alice	Dancing	2020-02-21	2020-02-23
Alice	Travel	2020-02-24	2020-02-28
Bob	Travel	2020-02-11	2020-02-18

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.

```
Solution:
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);
```


create-db-template.sql ×
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-templat
▷ Execute
Data ×
Input to filter result
Free
1
Cost: 5ms
1
Total 2

```sql
create table UserActivity
(username   varchar(15),
activity    varchar(15),
startDate   Date,
endDate Date
);
```

```sql
insert into UserActivity values
('Alice',   'Travel',   '2020-02-12',   '2020-02-20'),
('Alice',   'Dancing',  '2020-02-21',   '2020-02-23'),
('Alice',   'Travel',   '2020-02-24',   '2020-02-28'),
('Bob', 'Travel',   '2020-02-11',   '2020-02-18');
```

```sql
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);
```

```sql
with new as
(select t.username, t.activity, t.startDate, t.endDate
```

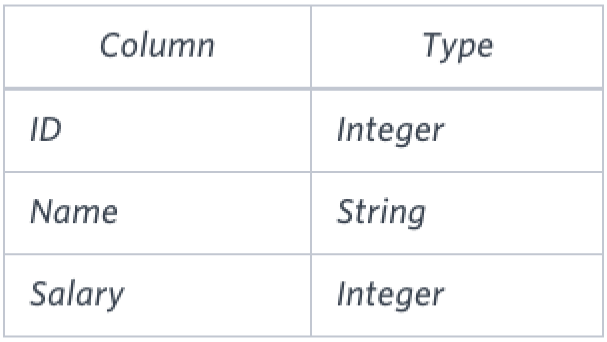
	username varchar	activity varchar	startDate date	endDate date
1	Alice	Dancing	2020-02-21	2020-02-23
2	Bob	Travel	2020-02-11	2020-02-18

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.

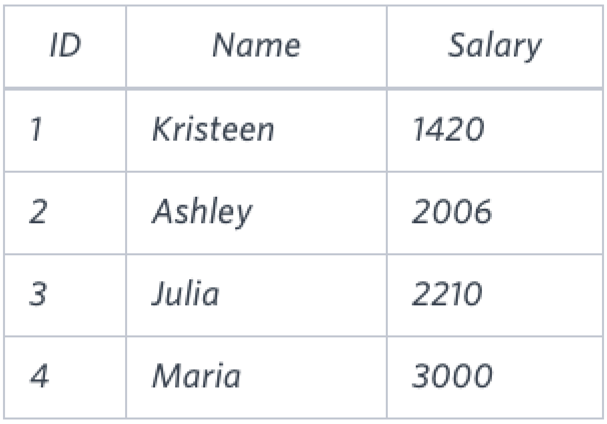
Input Format

The EMPLOYEES table is described as follows:



Column	Type
ID	Integer
Name	String
Salary	Integer

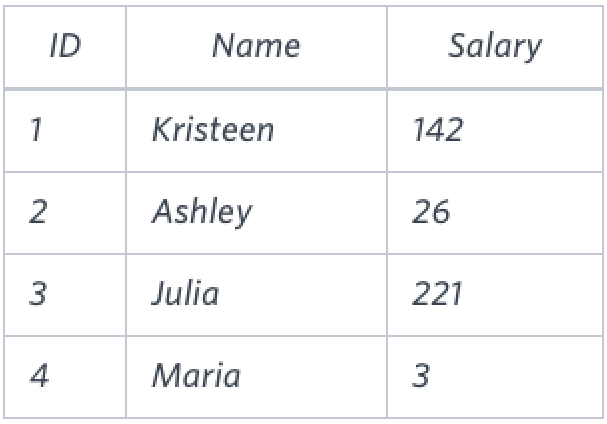
Note: Salary is per month.
Constraints
1000<salary < 10^5
Sample Input



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

Sample Output
2061

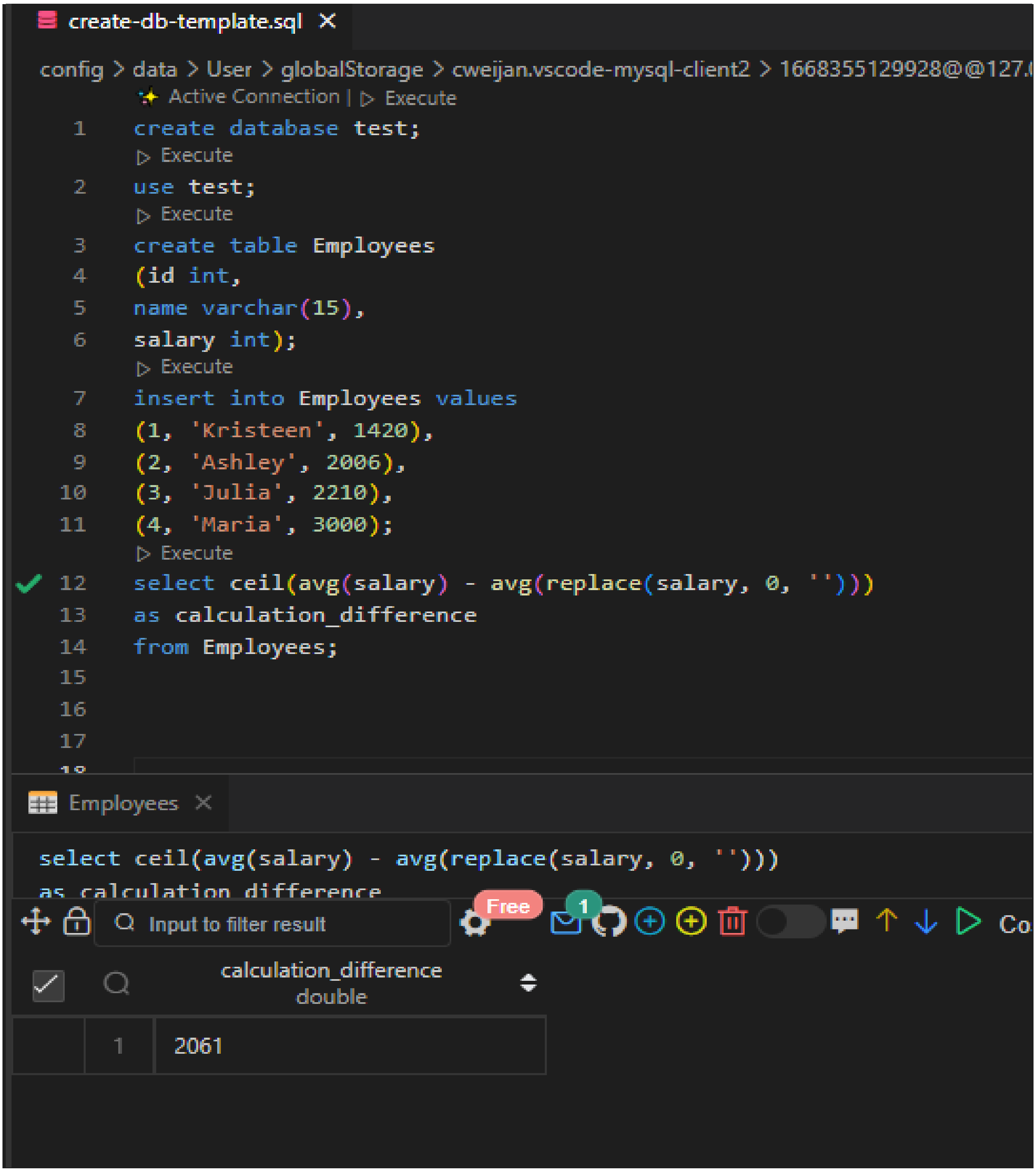
Explanation
The table below shows the salaries without zeros as they were entered by Samantha:



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.

```
Solution:
select ceil(avg(salary) - avg(replace(salary, 0, '')))
as calculation_difference
from Employees;
```



create-db-template.sql
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.
Active Connection | ▷ Execute
▷ Execute

```
create database test;
use test;
create table Employees
(id int,
name varchar(15),
salary int);
insert into Employees values
(1, 'Kristeen', 1420),
(2, 'Ashley', 2006),
(3, 'Julia', 2210),
(4, 'Maria', 3000);
select ceil(avg(salary) - avg(replace(salary, 0, '')))
as calculation_difference
from Employees;
```

Employees
Input to filter result
Free
1
Co

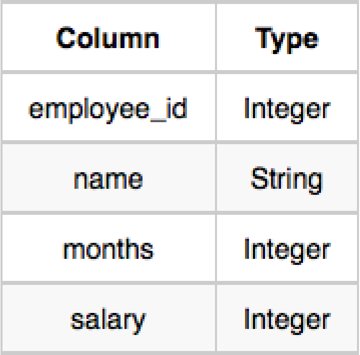
```
select ceil(avg(salary) - avg(replace(salary, 0, '')))
as calculation_difference
```

		calculation_difference double
	1	2061

Q138.

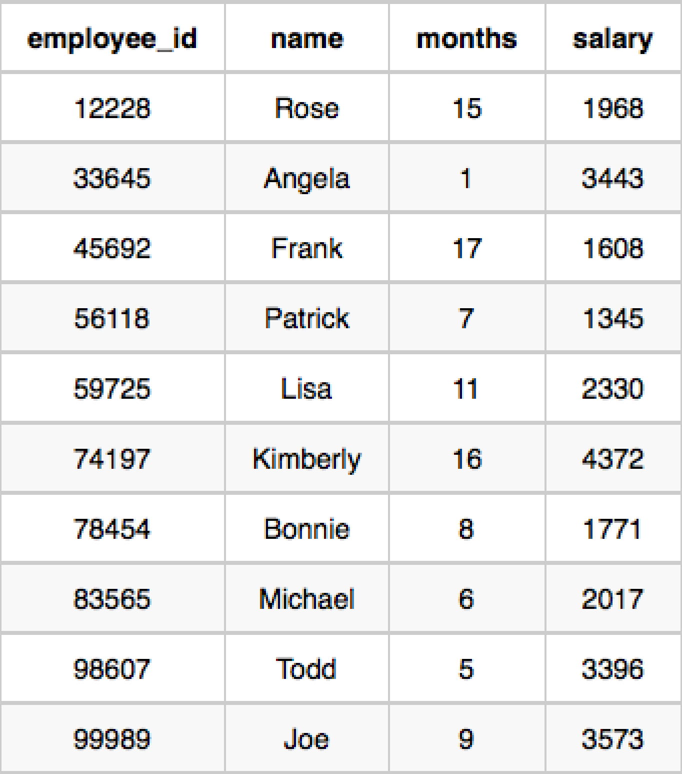
We define an employee's total earnings to be their monthly salary * months worked, and the maximum total earnings to be the maximum total earnings for any employee in the Employee table. Write a query to find the maximum total earnings for all employees as well as the total number of employees who have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



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

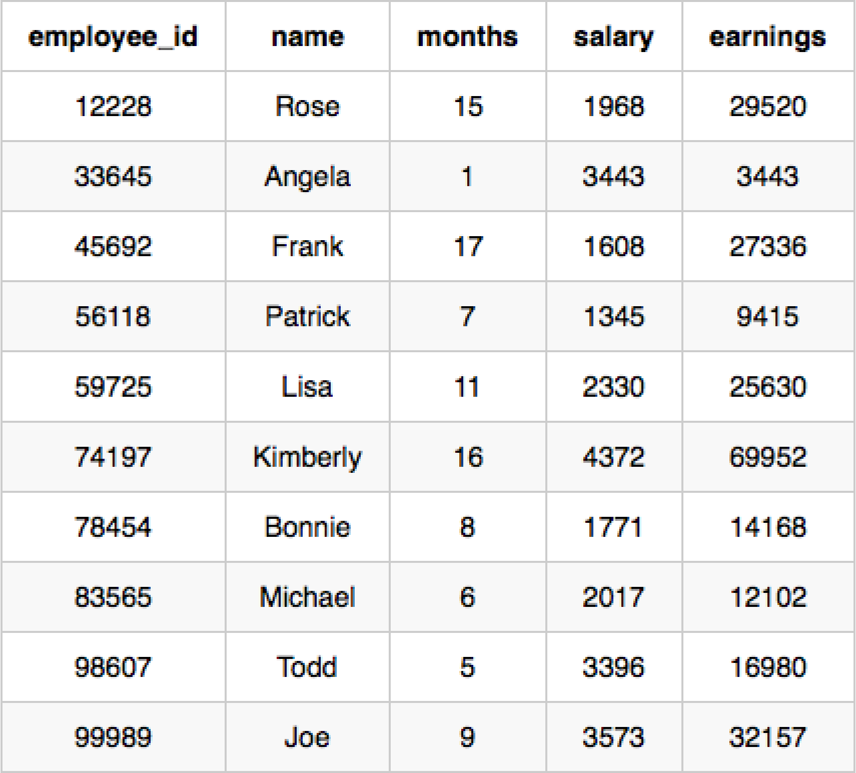


employee_id	name	months	salary
12228	Rose	15	1968
33645	Angela	1	3443
45692	Frank	17	1608
56118	Patrick	7	1345
59725	Lisa	11	2330
74197	Kimberly	16	4372
78454	Bonnie	8	1771
83565	Michael	6	2017
98607	Todd	5	3396
99989	Joe	9	3573

Sample Output

```
69952 1
```

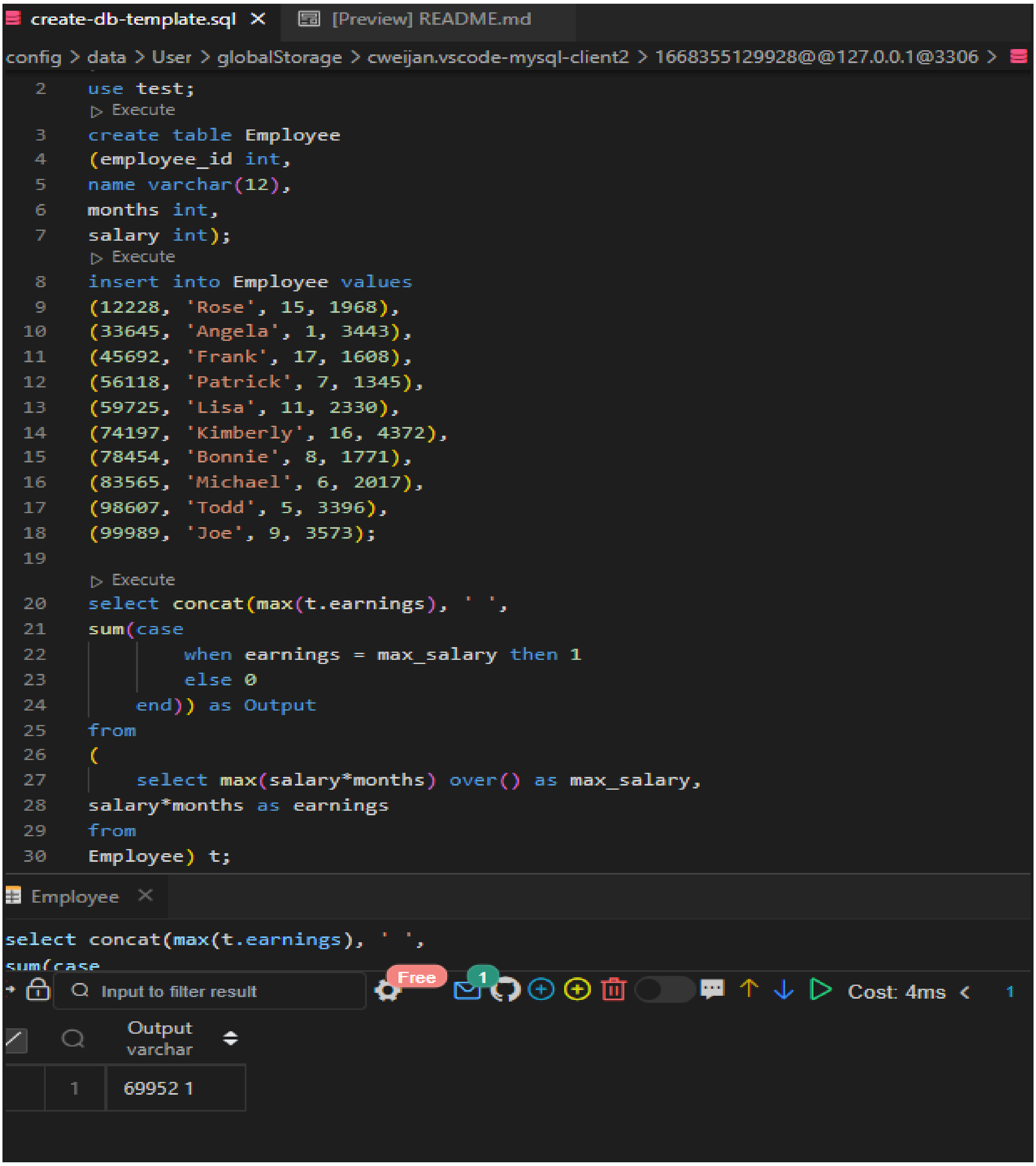
Explanation:
The table and earnings data is depicted in the following diagram:



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 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;
```


create-db-template.sql ×
[Preview] README.md
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 >
▷ Execute
Employee ×
Input to filter result
Free
1
Cost: 4ms
1

```
use test;
create table Employee
(employee_id int,
name varchar(12),
months int,
salary int);
insert into Employee values
(12228, 'Rose', 15, 1968),
(33645, 'Angela', 1, 3443),
(45692, 'Frank', 17, 1608),
(56118, 'Patrick', 7, 1345),
(59725, 'Lisa', 11, 2330),
(74197, 'Kimberly', 16, 4372),
(78454, 'Bonnie', 8, 1771),
(83565, 'Michael', 6, 2017),
(98607, 'Todd', 5, 3396),
(99989, 'Joe', 9, 3573);

select 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;
```

```
select concat(max(t.earnings), ' ',
sum(case
```

	Output varchar
1	69952 1

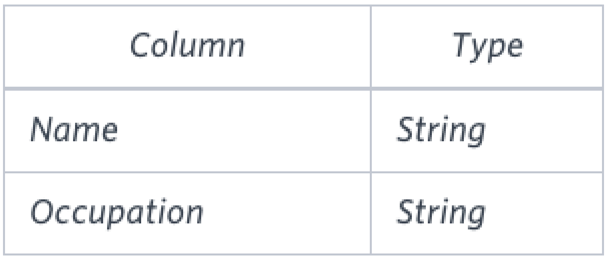
Q139.

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).

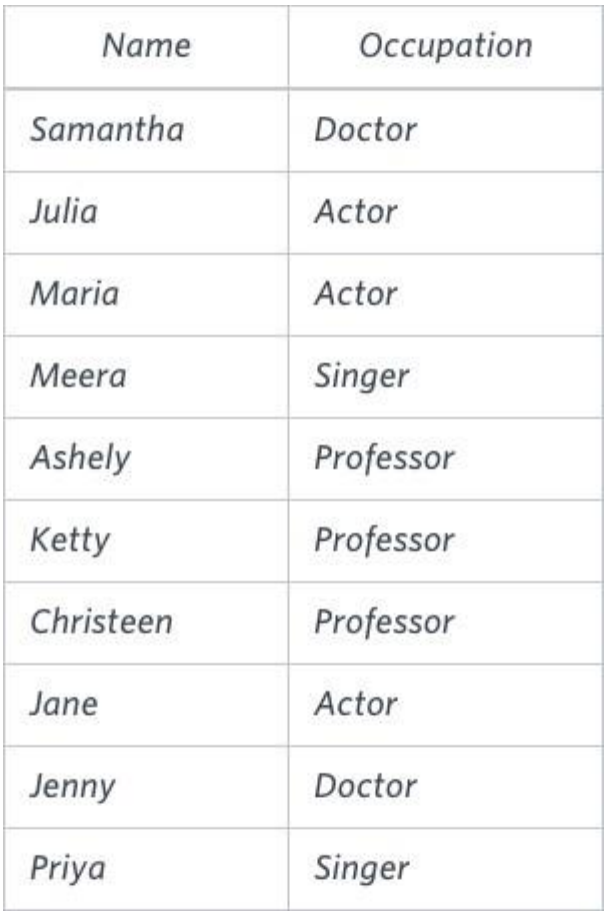
Query the number of occurrences of each occupation in OCCUPATIONS. Sort the occurrences in ascending order, and output them in the following format:
Level - Medium
There are a total of [occupation_count] [occupation]s.

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

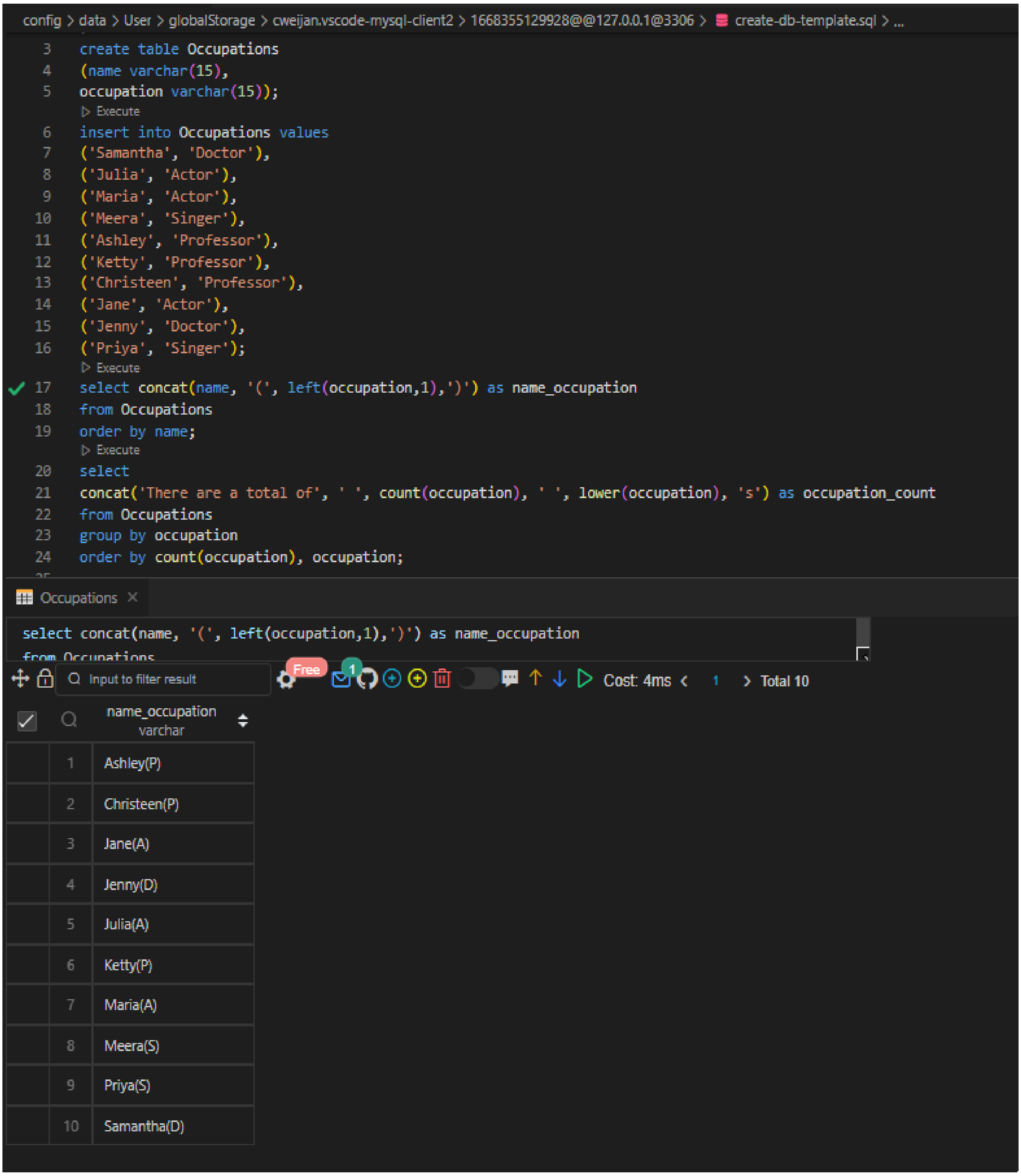
```
Ashely(P)
Christeen(P)
Jane(A)
Jenny(D)
Julia(A)
Ketty(P)
Maria(A)
Meera(S)
Priya(S)
Samantha(D)
There are a total of 2 doctors.
There are a total of 2 singers.
There are a total of 3 actors.
There are a total of 3 professors.
```

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).

```
Solution:
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;
```

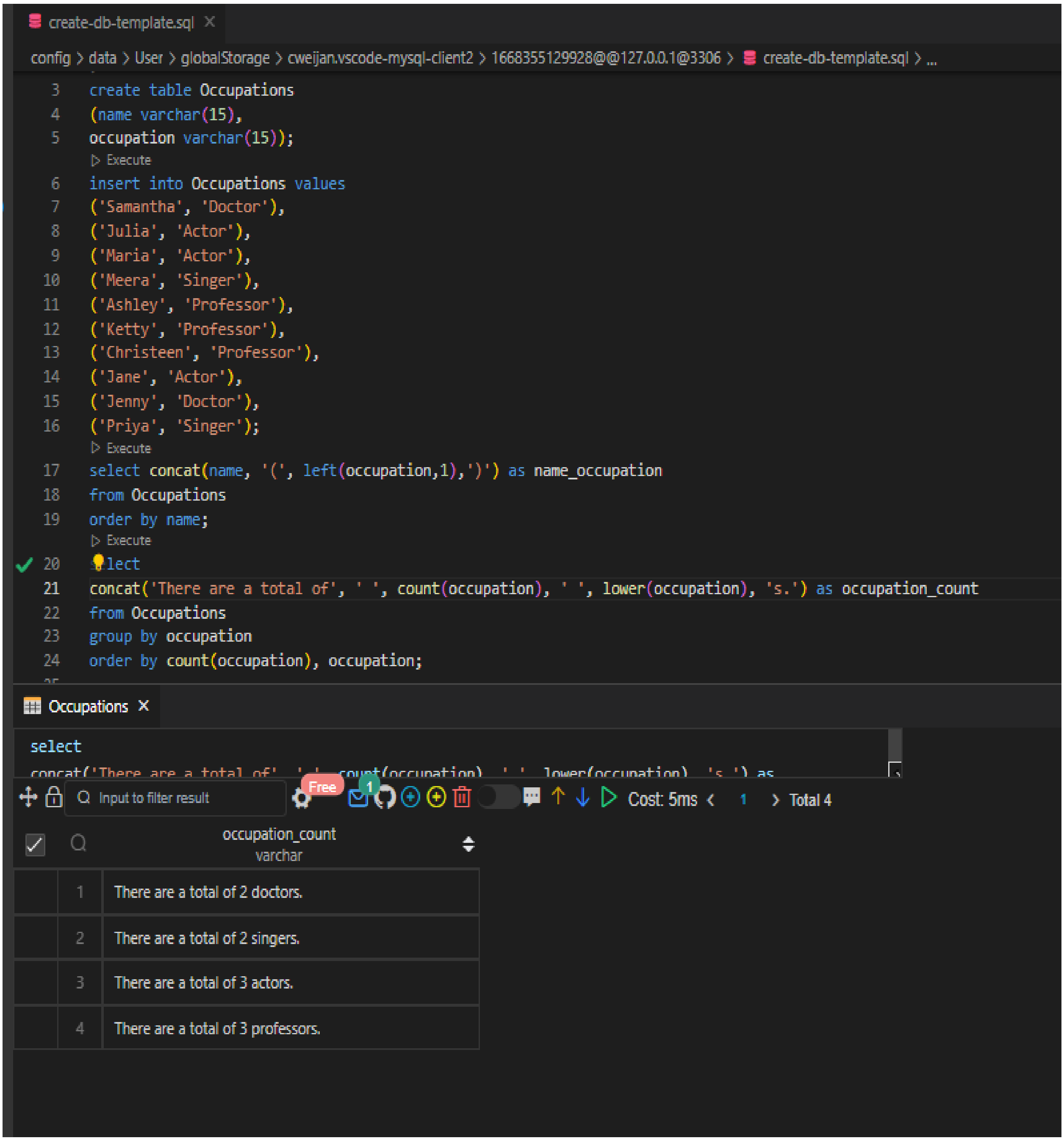


config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-template.sql > ...

```
create table Occupations
(name varchar(15),
occupation varchar(15));
▷ Execute
insert into Occupations values
('Samantha', 'Doctor'),
('Julia', 'Actor'),
('Maria', 'Actor'),
('Meera', 'Singer'),
('Ashley', 'Professor'),
('Ketty', 'Professor'),
('Christeen', 'Professor'),
('Jane', 'Actor'),
('Jenny', 'Doctor'),
('Priya', 'Singer');
▷ Execute
select concat(name, '(', left(occupation,1),')') as name_occupation
from Occupations
order by name;
▷ Execute
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;
```

```
select concat(name, '(', left(occupation,1),')') as name_occupation
from Occupations
```

		name_occupation varchar
	1	Ashley(P)
	2	Christeen(P)
	3	Jane(A)
	4	Jenny(D)
	5	Julia(A)
	6	Ketty(P)
	7	Maria(A)
	8	Meera(S)
	9	Priya(S)
	10	Samantha(D)



create-db-template.sql ×
config › data › User › globalStorage › cweijan.vscode-mysql-client2 › 1668355129928@@127.0.0.1@3306 › create-db-template.sql › ...

```sql
create table Occupations
(name varchar(15),
occupation varchar(15));
▷ Execute
insert into Occupations values
('Samantha', 'Doctor'),
('Julia', 'Actor'),
('Maria', 'Actor'),
('Meera', 'Singer'),
('Ashley', 'Professor'),
('Ketty', 'Professor'),
('Christeen', 'Professor'),
('Jane', 'Actor'),
('Jenny', 'Doctor'),
('Priya', 'Singer');
▷ Execute
select concat(name, '(', left(occupation,1),')') as name_occupation
from Occupations
order by name;
▷ Execute
lect
concat('There are a total of', ' ', count(occupation), ' ', lower(occupation), 's.') as occupation_count
from Occupations
group by occupation
order by count(occupation), occupation;
```

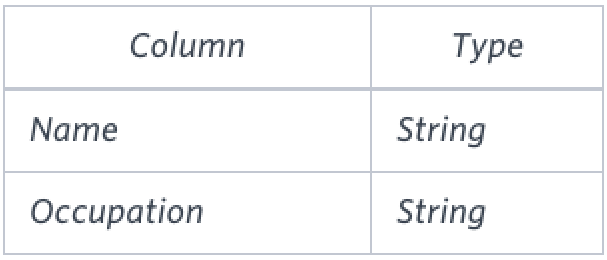
```sql
select
concat('There are a total of', ' ', count(occupation), ' ', lower(occupation), 's.') as
```

	occupation_count varchar
1	There are a total of 2 doctors.
2	There are a total of 2 singers.
3	There are a total of 3 actors.
4	There are a total of 3 professors.

Q140 .

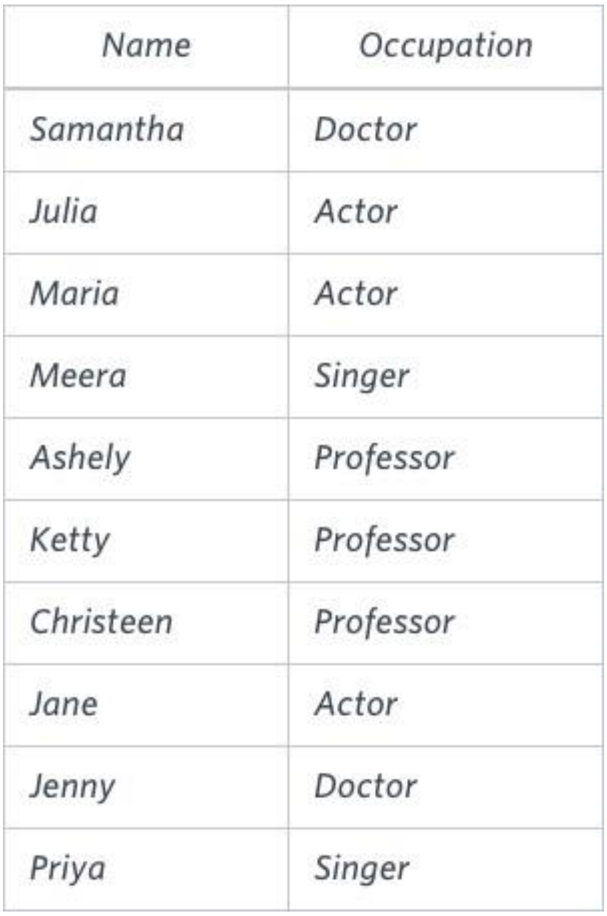
Pivot the Occupation column in OCCUPATIONS so that each Name is sorted alphabetically and displayed underneath its corresponding Occupation. The output column headers should be Doctor, Professor, Singer, and Actor, respectively.
Note: Print NULL when there are no more names corresponding to an occupation.

Input Format
The OCCUPATIONS table is described as follows:



Column	Type
Name	String
Occupation	String

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



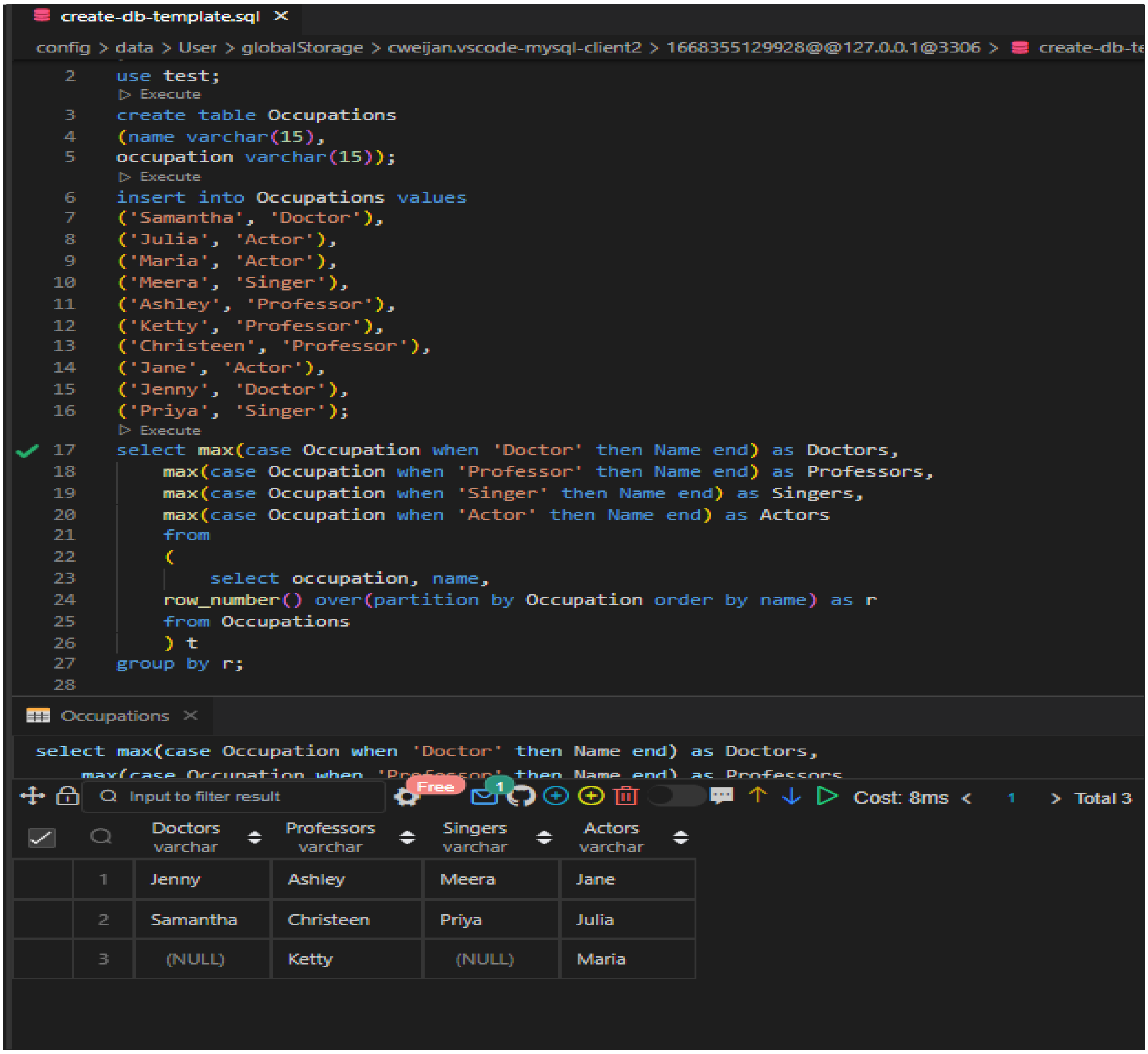
Name	Occupation
Samantha	Doctor
Julia	Actor
Maria	Actor
Meera	Singer
Ashely	Professor
Ketty	Professor
Christeen	Professor
Jane	Actor
Jenny	Doctor
Priya	Singer

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.

```
Solution:
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;
```



create-db-template.sql
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-te
Execute
Occupations
Input to filter result
Free
Cost: 8ms
1
Total 3

```sql
use test;
create table Occupations
(name varchar(15),
occupation varchar(15));
insert into Occupations values
('Samantha', 'Doctor'),
('Julia', 'Actor'),
('Maria', 'Actor'),
('Meera', 'Singer'),
('Ashley', 'Professor'),
('Ketty', 'Professor'),
('Christeen', 'Professor'),
('Jane', 'Actor'),
('Jenny', 'Doctor'),
('Priya', 'Singer');
select 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;
```

```sql
select max(case Occupation when 'Doctor' then Name end) as Doctors,
    max(case Occupation when 'Professor' then Name end) as Professors
```

	Doctors varchar	Professors varchar	Singers varchar	Actors varchar
1	Jenny	Ashley	Meera	Jane
2	Samantha	Christeen	Priya	Julia
3	(NULL)	Ketty	(NULL)	Maria

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.

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:

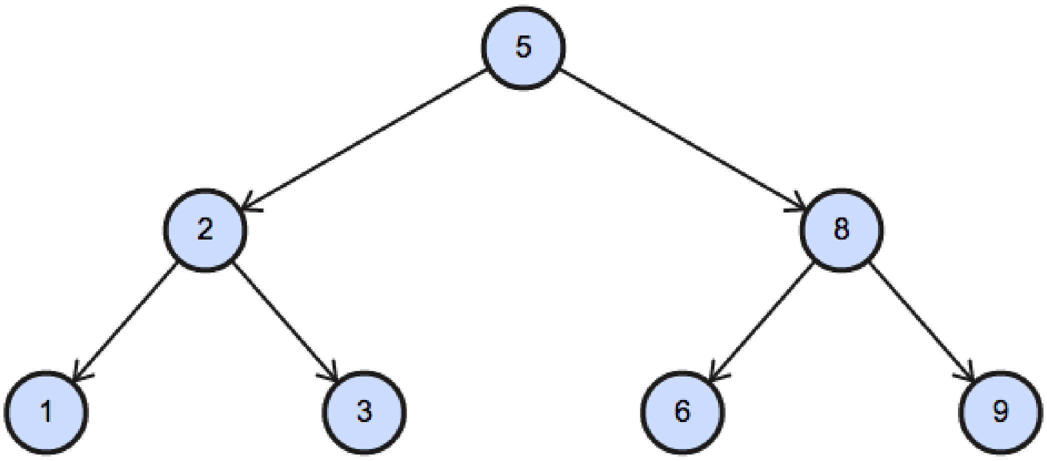
- Root: If node is root node.
- Leaf: If node is leaf node.
- Inner: If node is neither root nor leaf node.

Sample Input

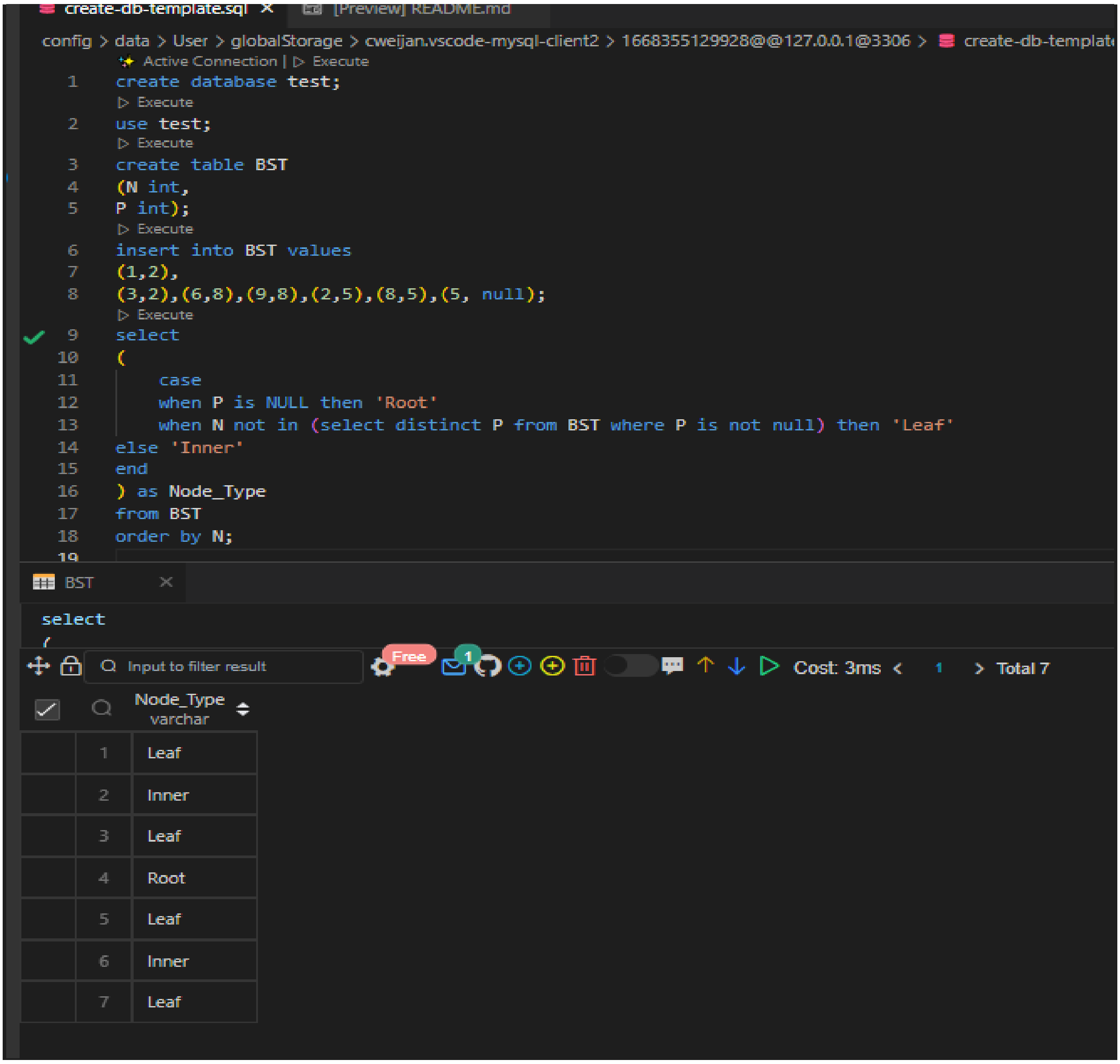
N	P
1	2
3	2
6	8
9	8
2	5
8	5
5	null

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

Explanation
The Binary Tree below illustrates the sample:



```
Solution:
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;
```


create-db-template.sql ×
[Preview] README.md
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-templat
Active Connection | ▷ Execute
▷ Execute
BST ×
Input to filter result
Free
Cost: 3ms
Total 7

```
create database test;
use test;
create table BST
(N int,
P int);
insert into BST values
(1,2),
(3,2),(6,8),(9,8),(2,5),(8,5),(5, null);
select
(
    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;
```

```
select
(
```

	Node_Type varchar
1	Leaf
2	Inner
3	Leaf
4	Root
5	Leaf
6	Inner
7	Leaf

Q142 .

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:

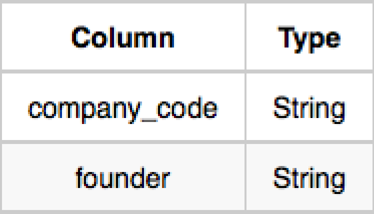
- The tables may contain duplicate records.

- 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

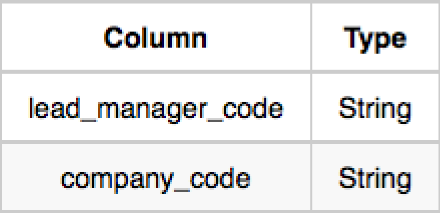
The following tables contain company data:

- Company: The company_code is the code of the company and founder is the founder of the company.



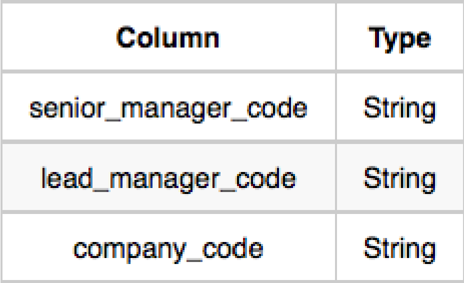
Column	Type
company_code	String
founder	String

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



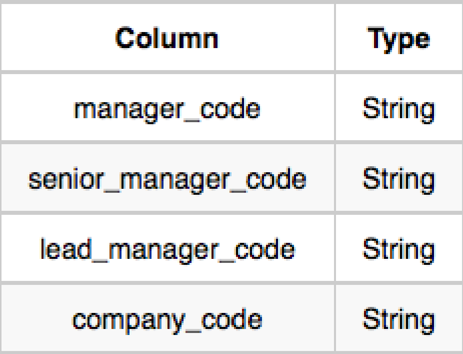
Column	Type
lead_manager_code	String
company_code	String

- 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.



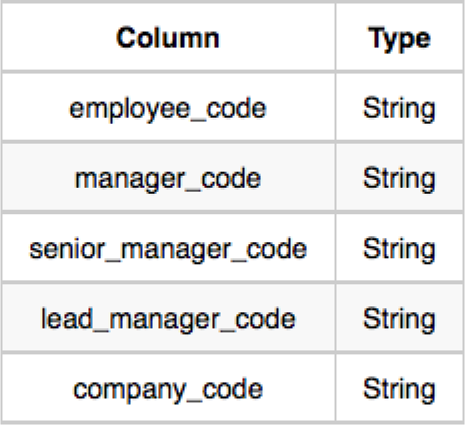
Column	Type
senior_manager_code	String
lead_manager_code	String
company_code	String

- 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

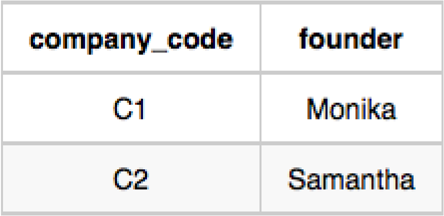
- Employee: The employee_code is the code of the employee, the manager_code is the code of its manager, the senior_manager_code is the code of its senior manager, the lead_manager_code is the code of its lead manager, and the company_code is the code of the working company.



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

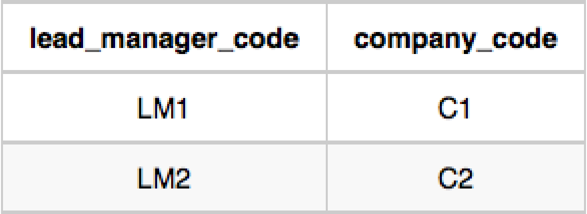
Sample Input

Company Table:



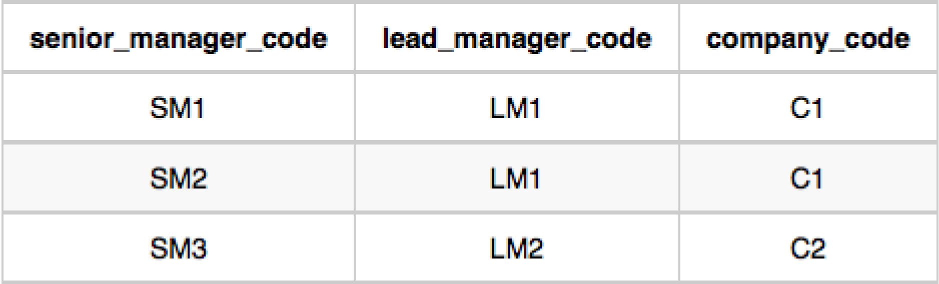
company_code	founder
C1	Monika
C2	Samantha

Lead_Manager Table:



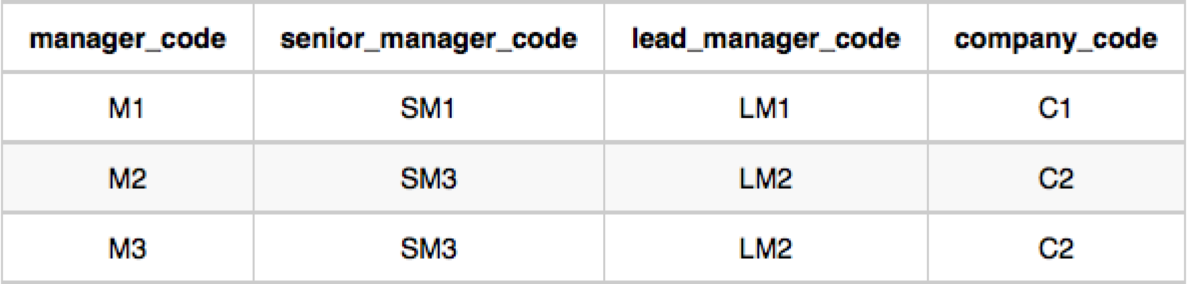
lead_manager_code	company_code
LM1	C1
LM2	C2

Senior_Manager Table:



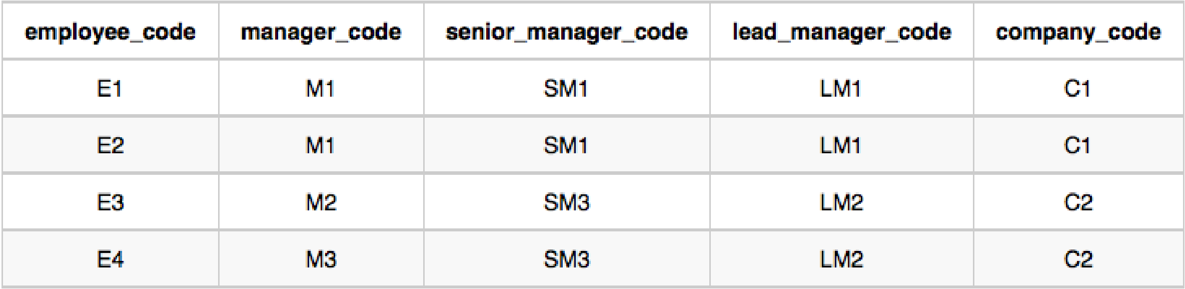
senior_manager_code	lead_manager_code	company_code
SM1	LM1	C1
SM2	LM1	C1
SM3	LM2	C2

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:

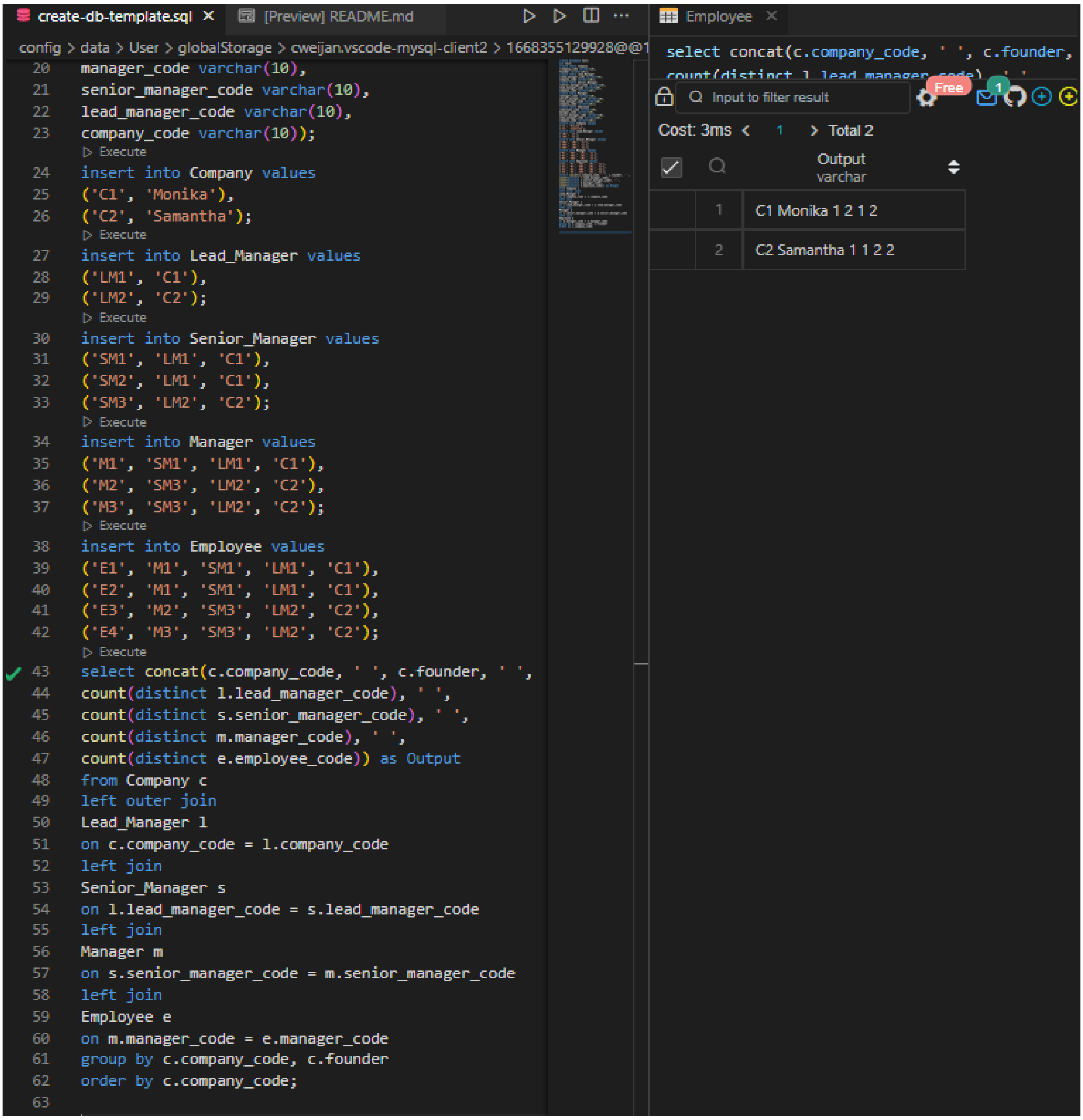


employee_code	manager_code	senior_manager_code	lead_manager_code	company_code
E1	M1	SM1	LM1	C1
E2	M1	SM1	LM1	C1
E3	M2	SM3	LM2	C2
E4	M3	SM3	LM2	C2

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.

```
Solution:
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;
```


create-db-template.sql
[Preview] README.md
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@1
Employee
Input to filter result
Free
Execute

```sql
manager_code varchar(10),
senior_manager_code varchar(10),
lead_manager_code varchar(10),
company_code varchar(10));
insert into Company values
('C1', 'Monika'),
('C2', 'Samantha');
insert into Lead_Manager values
('LM1', 'C1'),
('LM2', 'C2');
insert into Senior_Manager values
('SM1', 'LM1', 'C1'),
('SM2', 'LM1', 'C1'),
('SM3', 'LM2', 'C2');
insert into Manager values
('M1', 'SM1', 'LM1', 'C1'),
('M2', 'SM3', 'LM2', 'C2'),
('M3', 'SM3', 'LM2', 'C2');
insert into Employee values
('E1', 'M1', 'SM1', 'LM1', 'C1'),
('E2', 'M1', 'SM1', 'LM1', 'C1'),
('E3', 'M2', 'SM3', 'LM2', 'C2'),
('E4', 'M3', 'SM3', 'LM2', 'C2');
select 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;
```

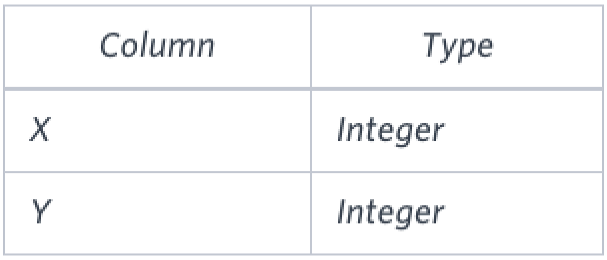
```sql
select concat(c.company_code, ' ', c.founder,
count(distinct l.lead_manager_code), ' '
```

Cost: 3ms < 1 > Total 2

	Output varchar
1	C1 Monika 1 2 1 2
2	C2 Samantha 1 1 2 2

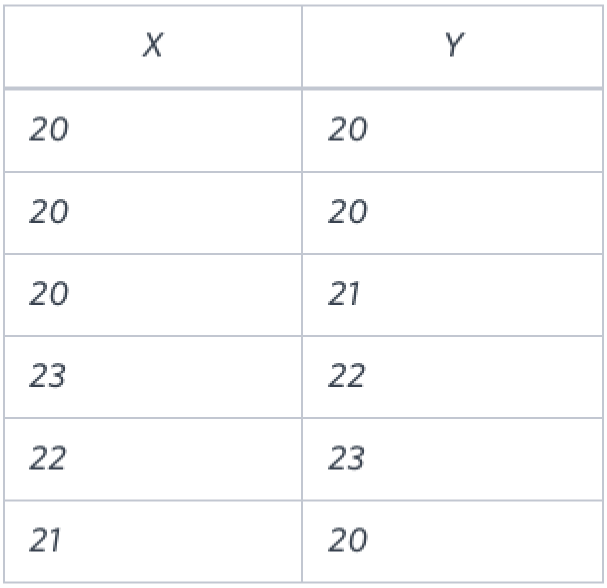
Q143 .

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



Column	Type
X	Integer
Y	Integer

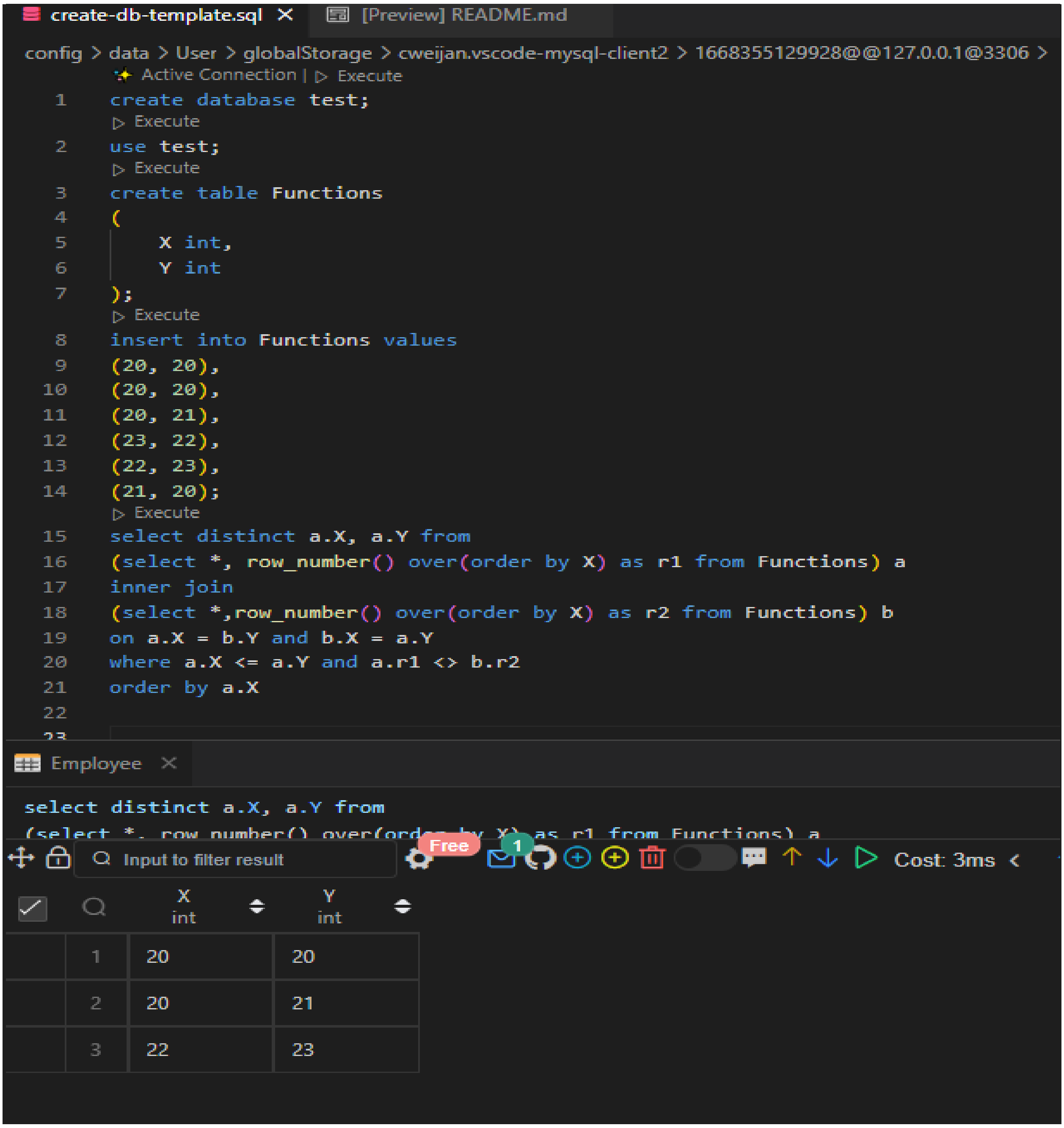
Two pairs (X1, Y1) and (X2, Y2) are said to be symmetric pairs if X1 = Y2 and X2 = Y1.
Write a query to output all such symmetric pairs in ascending order by the value of X. List the rows such that X1 ≤ Y1.
Level - Medium
Source - Hackerrank
Hint - Use group by and having clause .
Sample Input



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

Sample Output
20 20
20 21
22 23

```
Solution:
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
```


create-db-template.sql
[Preview] README.md
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 >
Active Connection | ▷ Execute
▷ Execute
Employee
Input to filter result
Free
1
Cost: 3ms

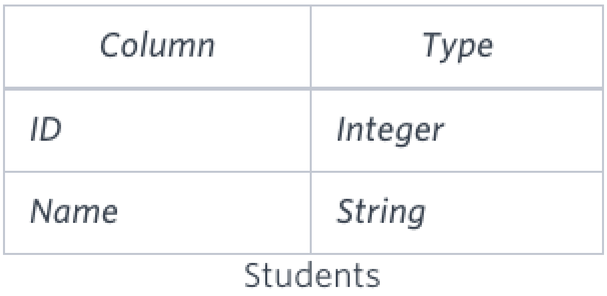
```sql
create database test;
use test;
create table Functions
(
    X int,
    Y int
);
insert into Functions values
(20, 20),
(20, 20),
(20, 21),
(23, 22),
(22, 23),
(21, 20);
select distinct a.X, a.Y from
(select *, row_number() over(order by X) as r1 from Functions) a
inner join
(select *,row_number() 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
```

```sql
select distinct a.X, a.Y from
(select *, row_number() over(order by X) as r1 from Functions) a
```

	X int	Y int
1	20	20
2	20	21
3	22	23

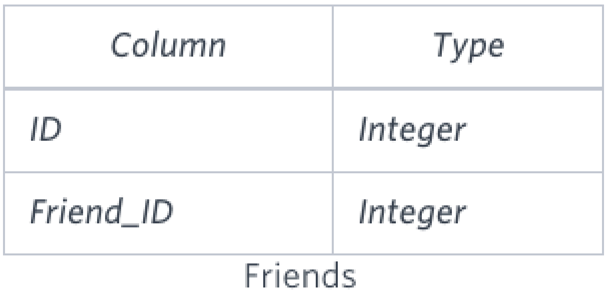
Q144 .

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).



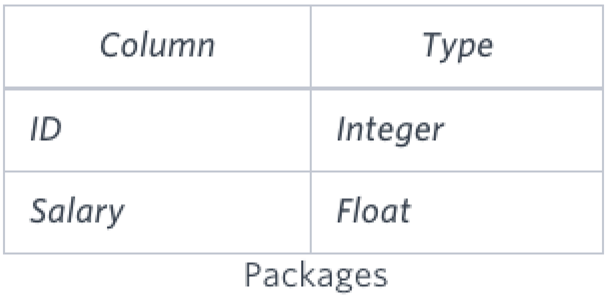
Column	Type
ID	Integer
Name	String

Students



Column	Type
ID	Integer
Friend_ID	Integer

Friends

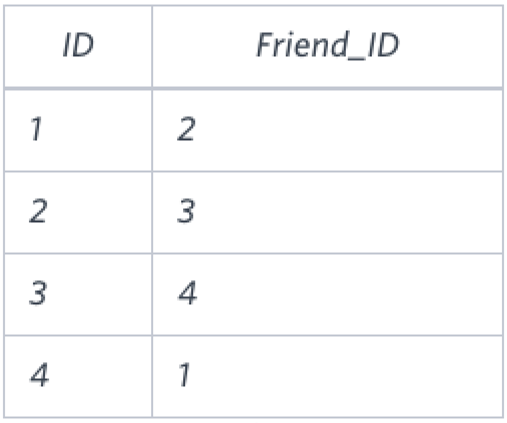


Column	Type
ID	Integer
Salary	Float

Packages

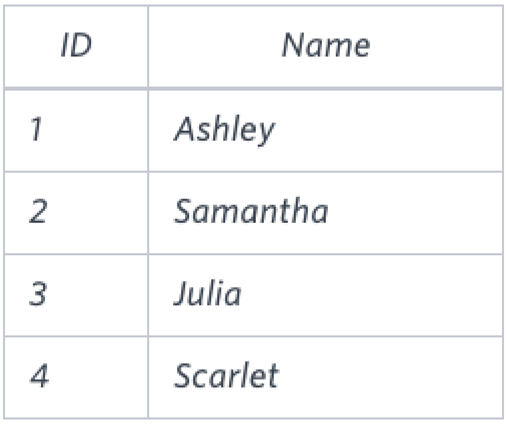
Write a query to output the names of those students whose best friends got offered a higher salary than them. Names must be ordered by the salary amount offered to the best friends. It is guaranteed that no two students get the same salary offer.

Sample Input



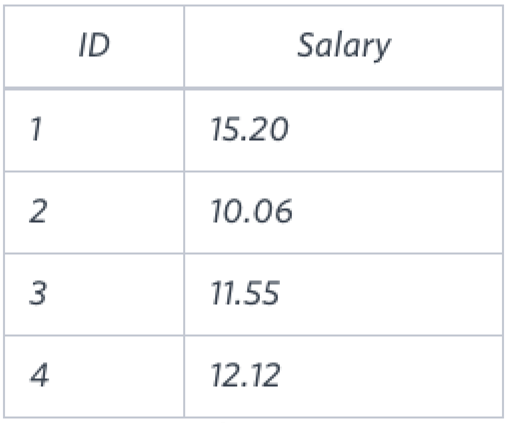
ID	Friend_ID
1	2
2	3
3	4
4	1

Friends



ID	Name
1	Ashley
2	Samantha
3	Julia
4	Scarlet

Students



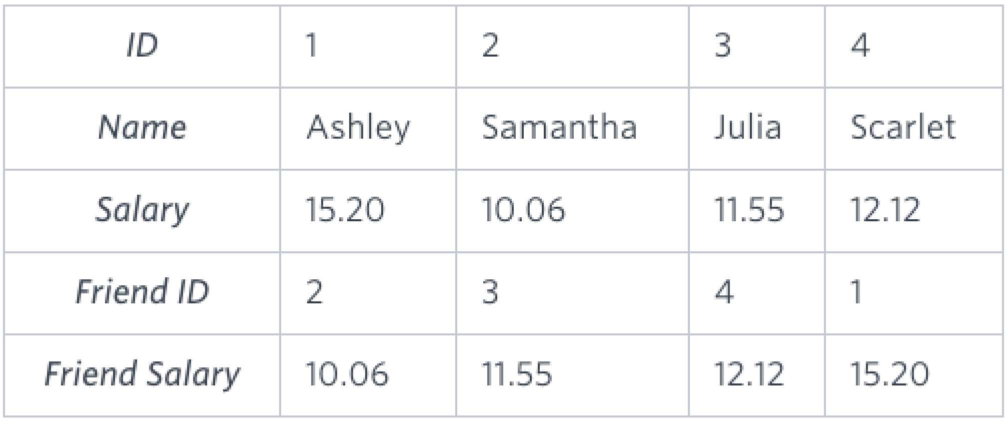
ID	Salary
1	15.20
2	10.06
3	11.55
4	12.12

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

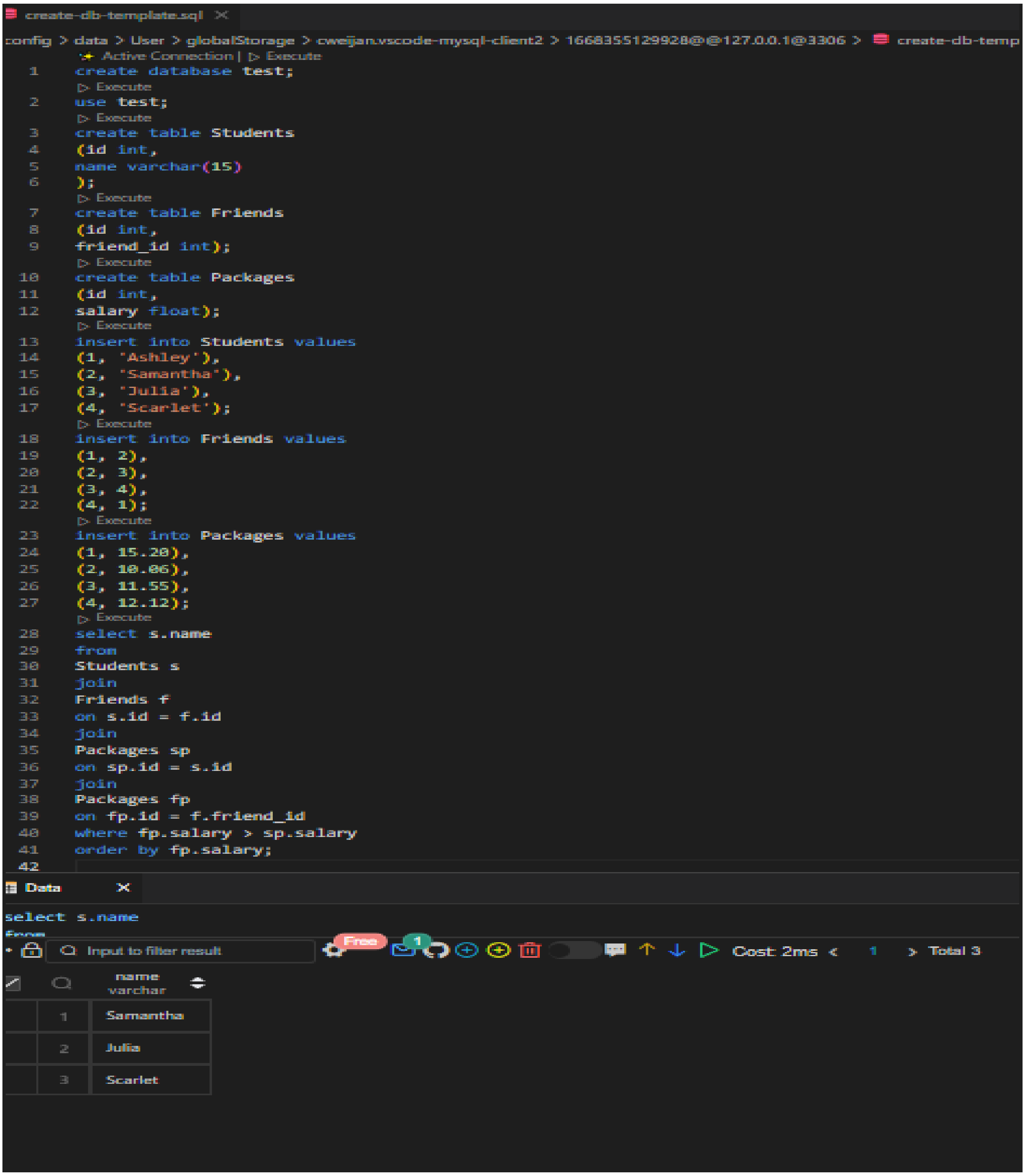
Now,

- 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

```
Solution:
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;
```


create-db-template.sql
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-temp
Active Connection | ▷ Execute
▷ Execute
Data
Input to filter result
Free
Cost: 2ms
Total 3

```sql
create database test;
use test;
create table Students
(id int,
name varchar(15)
);
create table Friends
(id int,
friend_id int);
create table Packages
(id int,
salary float);
insert into Students values
(1, 'Ashley'),
(2, 'Samantha'),
(3, 'Julia'),
(4, 'Scarlet');
insert into Friends values
(1, 2),
(2, 3),
(3, 4),
(4, 1);
insert into Packages values
(1, 15.20),
(2, 10.06),
(3, 11.55),
(4, 12.12);
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;
```

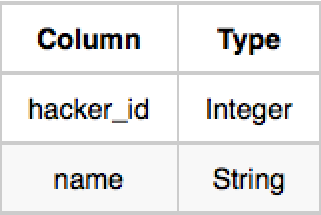
```
select s.name
from
```

```
   name
   varchar
1  Samantha
2  Julia
3  Scarlet
```

Q145.

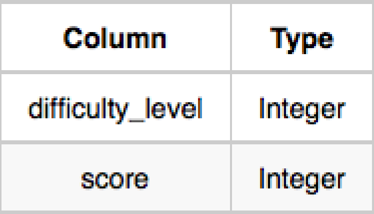
Julia just finished conducting a coding contest, and she needs your help assembling the leaderboard! Write a query to print the respective hacker_id and name of hackers who achieved full scores for more than one challenge. Order your output in descending order by the total number of challenges in which 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.



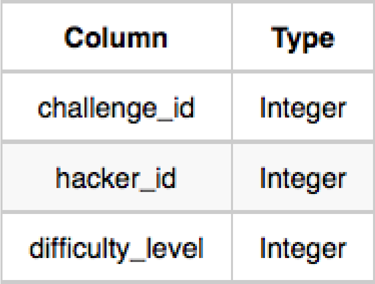
Column	Type
hacker_id	Integer
name	String

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



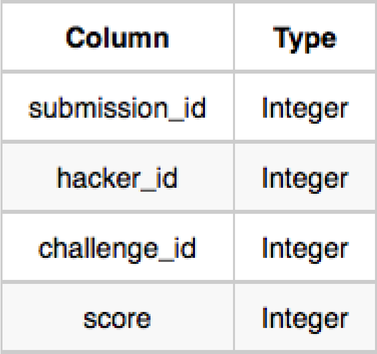
Column	Type
difficulty_level	Integer
score	Integer

- 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

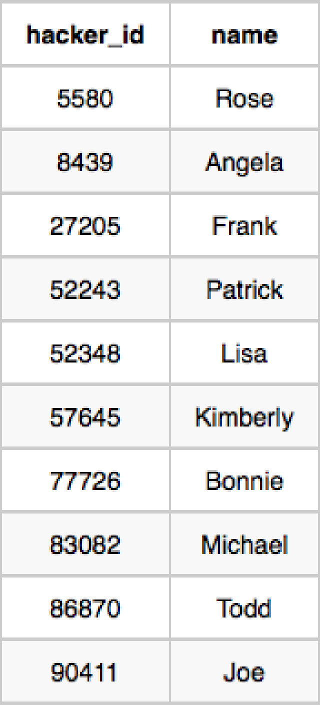
- Submissions: The submission_id is the id of the submission, hacker_id is the id of the hacker who made the submission, challenge_id is the id of the challenge that the submission belongs to, and score is the score of the submission.



Column	Type
submission_id	Integer
hacker_id	Integer
challenge_id	Integer
score	Integer

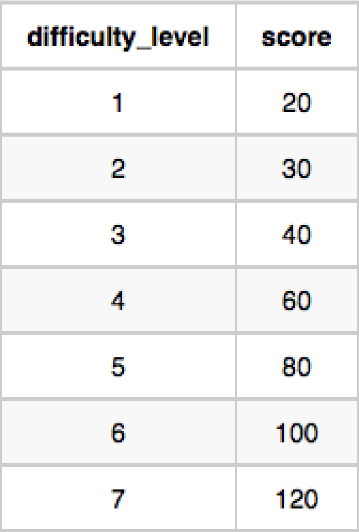
Sample Input

Hackers Table:



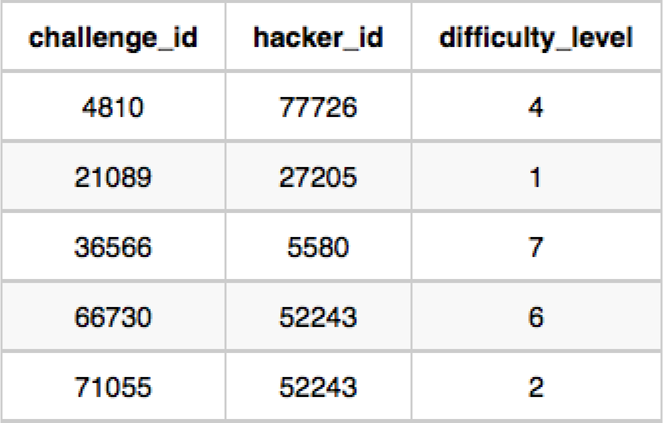
hacker_id	name
5580	Rose
8439	Angela
27205	Frank
52243	Patrick
52348	Lisa
57645	Kimberly
77726	Bonnie
83082	Michael
86870	Todd
90411	Joe

Difficulty Table:



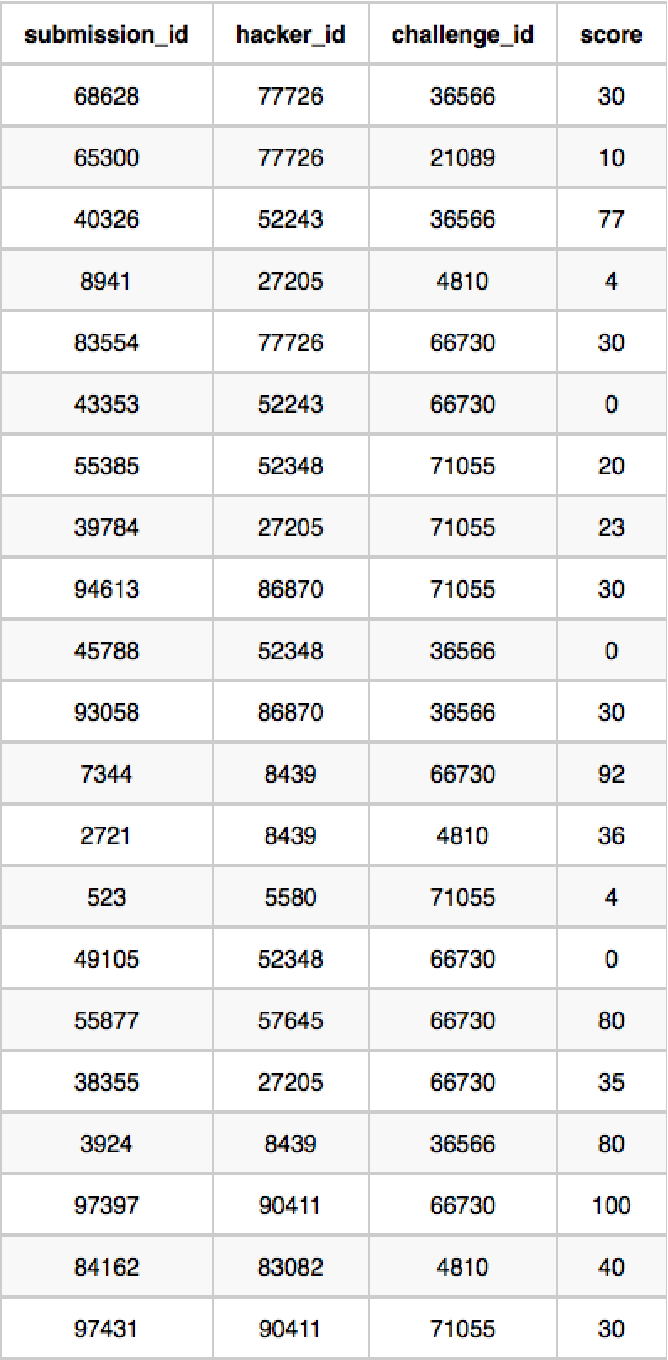
difficulty_level	score
1	20
2	30
3	40
4	60
5	80
6	100
7	120

Challenges Table:



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

:



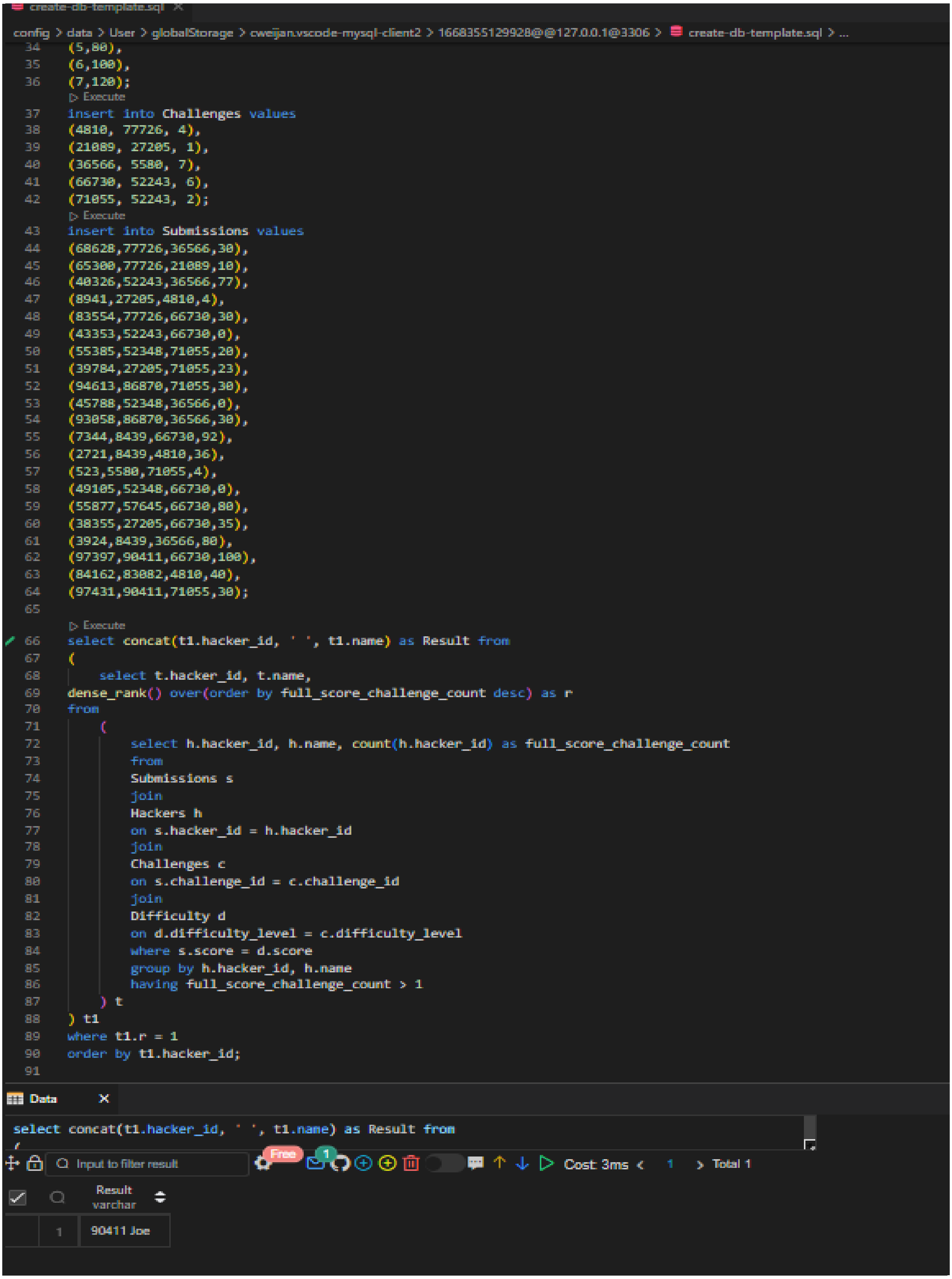
submission_id	hacker_id	challenge_id	score
68628	77726	36566	30
65300	77726	21089	10
40326	52243	36566	77
8941	27205	4810	4
83554	77726	66730	30
43353	52243	66730	0
55385	52348	71055	20
39784	27205	71055	23
94613	86870	71055	30
45788	52348	36566	0
93058	86870	36566	30
7344	8439	66730	92
2721	8439	4810	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.

```
Solution:
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;
```


create-db-template.sql
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-template.sql > ...
▷ Execute
Data
Input to filter result
Free
Cost: 3ms
Total 1

```
(5,80),
(6,100),
(7,120);
insert into Challenges values
(4810, 77726, 4),
(21089, 27205, 1),
(36566, 5580, 7),
(66730, 52243, 6),
(71055, 52243, 2);
insert into Submissions values
(68628,77726,36566,30),
(65300,77726,21089,10),
(40326,52243,36566,77),
(8941,27205,4810,4),
(83554,77726,66730,30),
(43353,52243,66730,0),
(55385,52348,71055,20),
(39784,27205,71055,23),
(94613,86870,71055,30),
(45788,52348,36566,0),
(93058,86870,36566,30),
(7344,8439,66730,92),
(2721,8439,4810,36),
(523,5580,71055,4),
(49105,52348,66730,0),
(55877,57645,66730,80),
(38355,27205,66730,35),
(3924,8439,36566,80),
(97397,90411,66730,100),
(84162,83082,4810,40),
(97431,90411,71055,30);

select 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;
```

```
select concat(t1.hacker_id, ' ', t1.name) as Result from
(
```

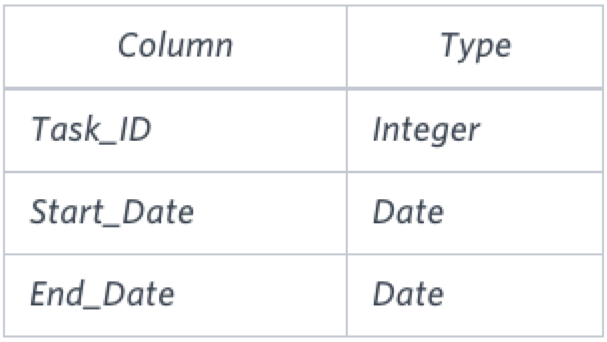
	Result varchar
1	90411 Joe

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

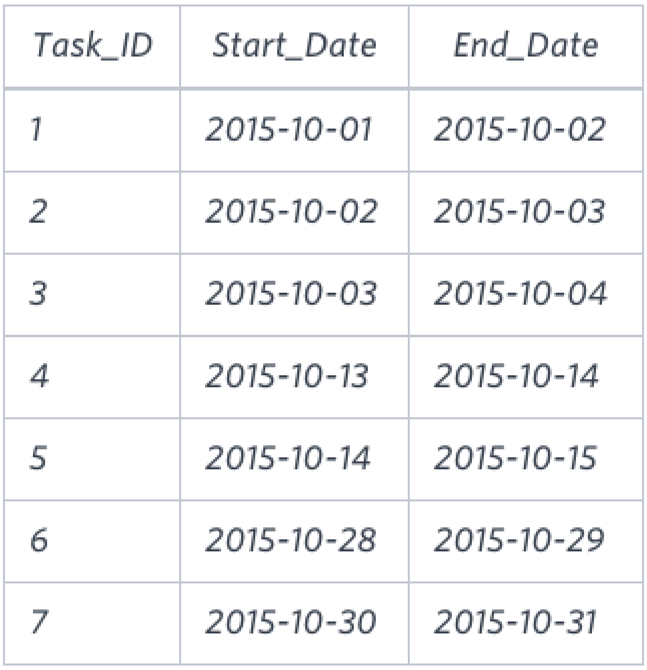


Column	Type
Task_ID	Integer
Start_Date	Date
End_Date	Date

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

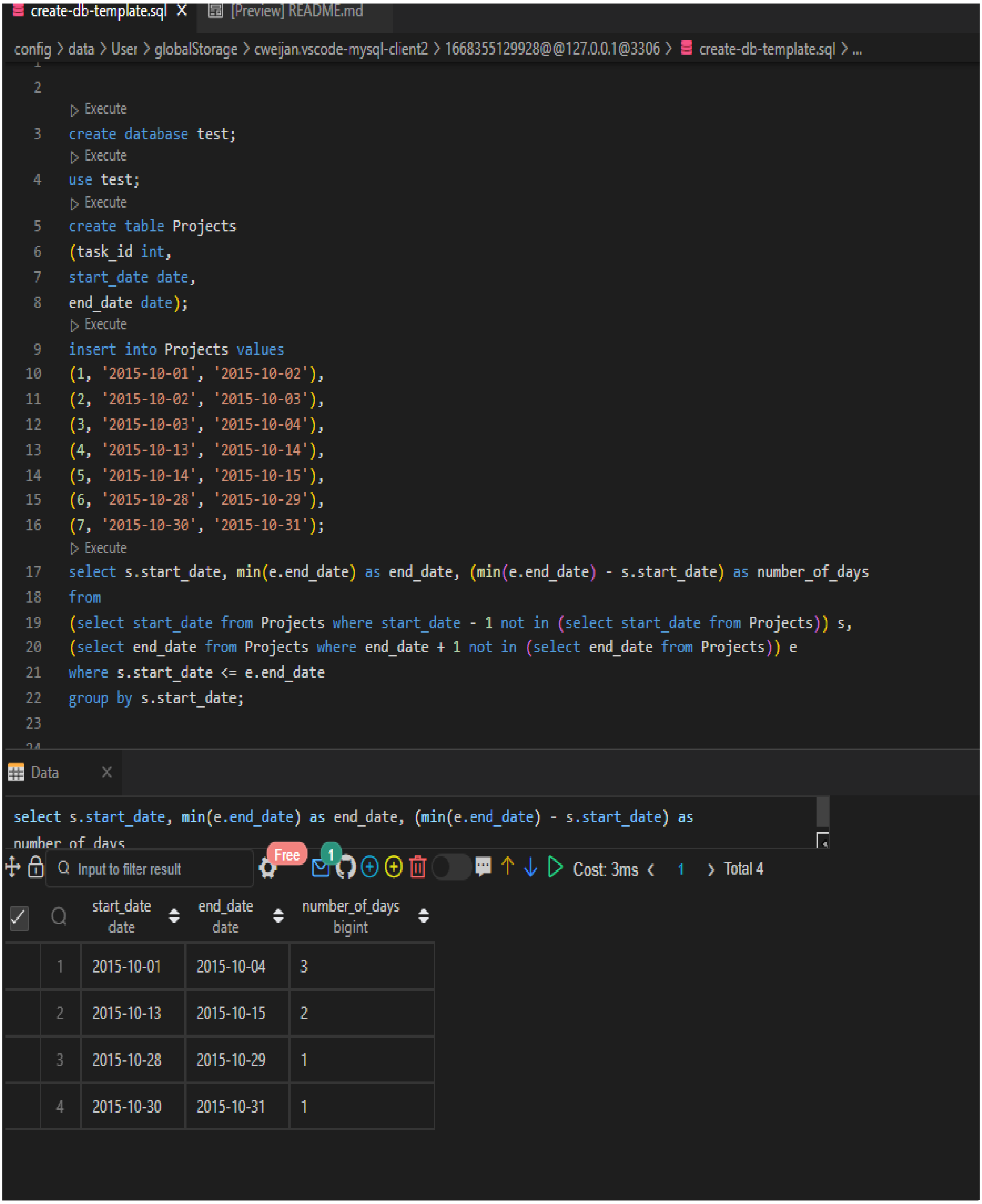
```
2015-10-28  2015-10-29
2015-10-30  2015-10-31
2015-10-13  2015-10-15
2015-10-01  2015-10-04
```

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.

```
Solution:
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;
```


create-db-template.sql
[Preview] README.md
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-template.sql > ...
Execute
Data
Input to filter result
Free
1
Cost: 3ms
1
Total 4

```sql
create database test;
use test;
create table Projects
(task_id int,
start_date date,
end_date date);
insert into Projects values
(1, '2015-10-01', '2015-10-02'),
(2, '2015-10-02', '2015-10-03'),
(3, '2015-10-03', '2015-10-04'),
(4, '2015-10-13', '2015-10-14'),
(5, '2015-10-14', '2015-10-15'),
(6, '2015-10-28', '2015-10-29'),
(7, '2015-10-30', '2015-10-31');
select 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;
```

```
select s.start_date, min(e.end_date) as end_date, (min(e.end_date) - s.start_date) as
number_of_days
```

	start_date date	end_date date	number_of_days bigint
1	2015-10-01	2015-10-04	3
2	2015-10-13	2015-10-15	2
3	2015-10-28	2015-10-29	1
4	2015-10-30	2015-10-31	1

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

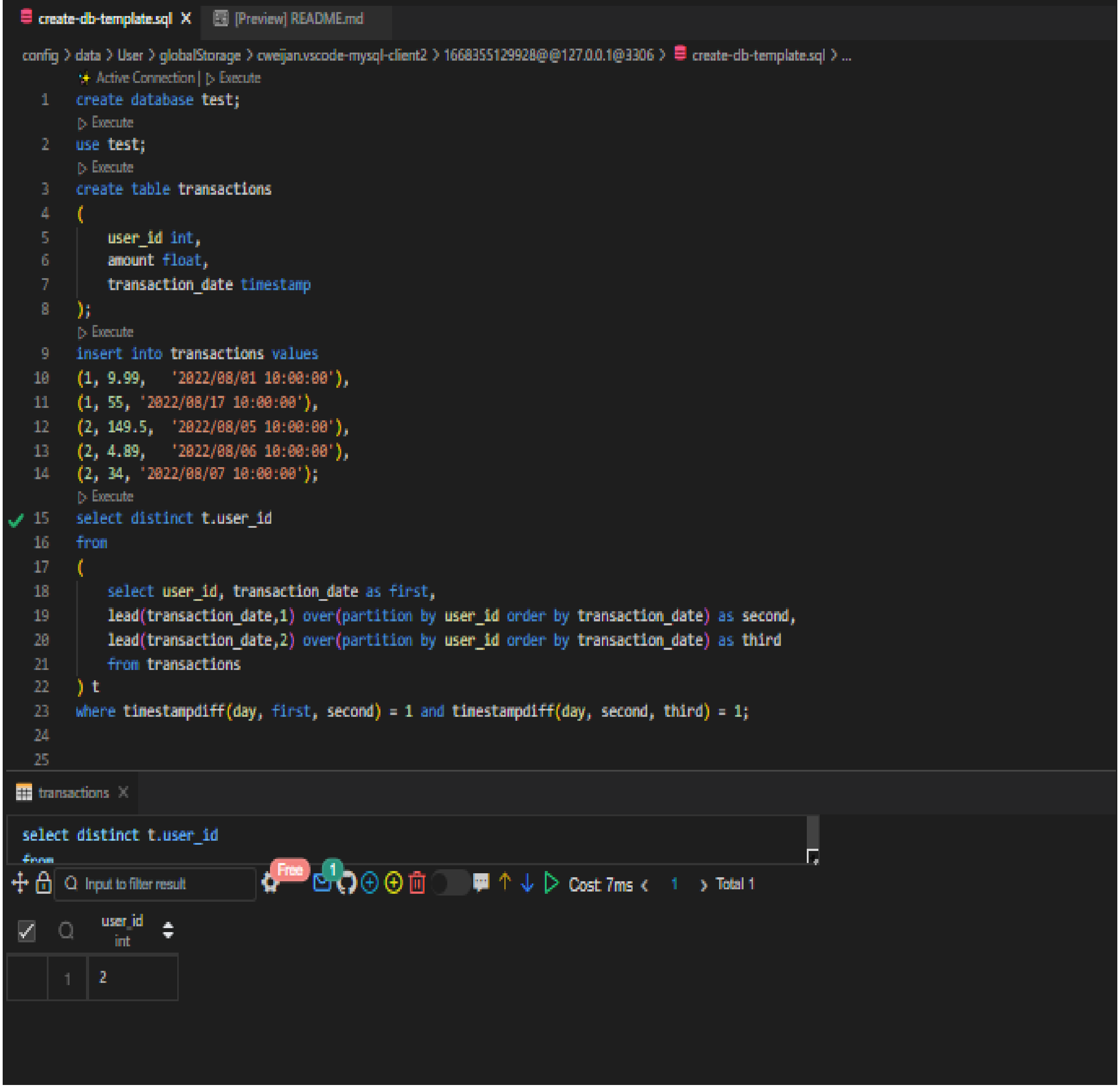
transactions Example Input:

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

```
Solution:
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;
```


create-db-template.sql
[Preview] README.md
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-template.sql > ...
Active Connection | ▷ Execute
▷ Execute
transactions
Input to filter result
Free
Cost 7ms
Total 1

```
create database test;
use test;
create table transactions
(
    user_id int,
    amount float,
    transaction_date timestamp
);
insert into transactions values
(1, 9.99,   '2022/08/01 10:00:00'),
(1, 55, '2022/08/17 10:00:00'),
(2, 149.5,  '2022/08/05 10:00:00'),
(2, 4.89,   '2022/08/06 10:00:00'),
(2, 34, '2022/08/07 10:00:00');
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;
```

```
   user_id
   int
1  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

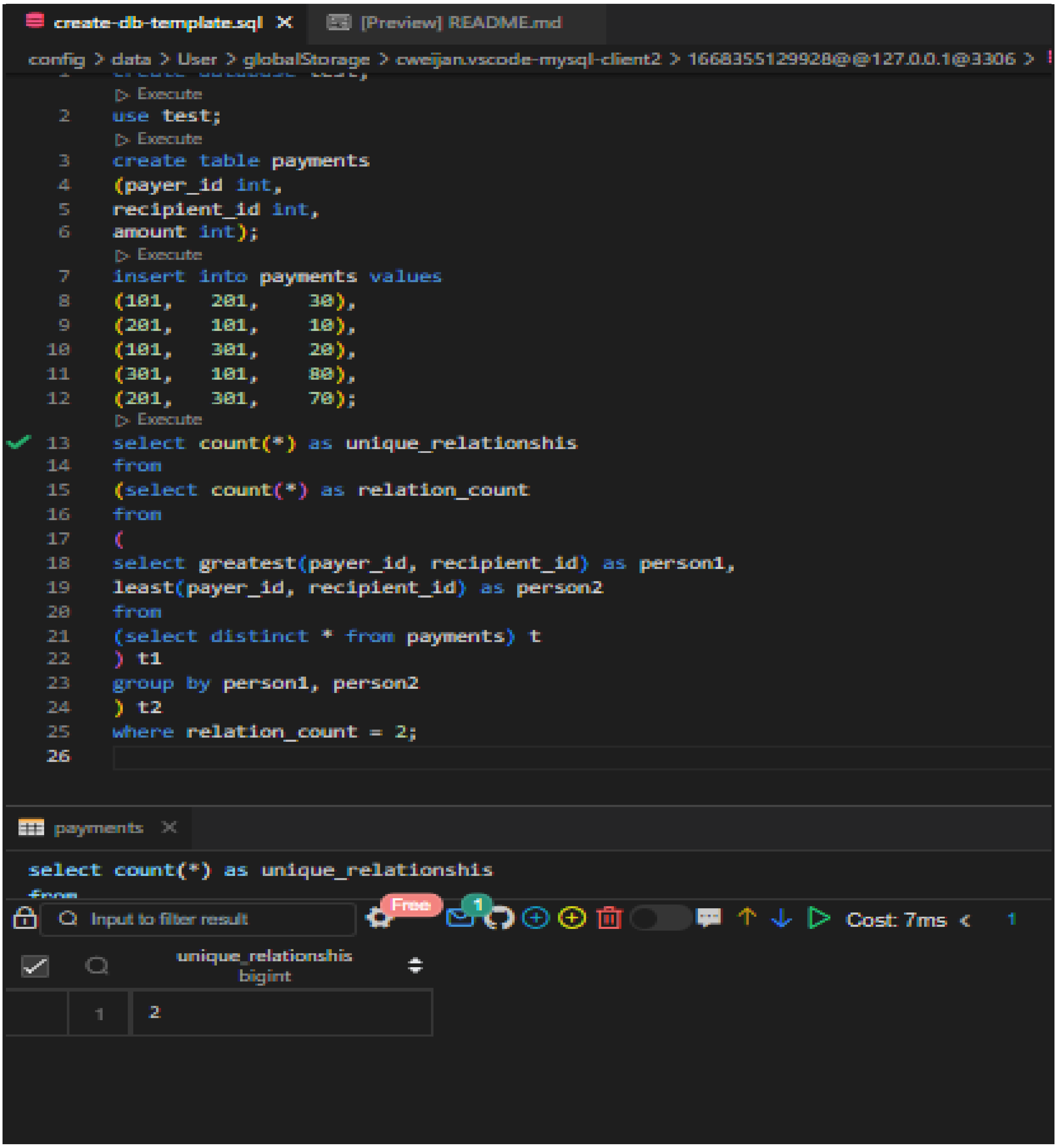
payments Example Input:

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

```
Solution:
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;
```



create-db-template.sql
[Preview] README.md
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 >
Execute
payments
Input to filter result
Free
Cost: 7ms

```sql
create database test;
use test;
create table payments
(payer_id int,
recipient_id int,
amount int);
insert into payments values
(101,   201,    30),
(201,   101,    10),
(101,   301,    20),
(301,   101,    80),
(201,   301,    70);
select 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;
```

```
select count(*) as unique_relationshis
from
```

	unique_relationshis bigint
1	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

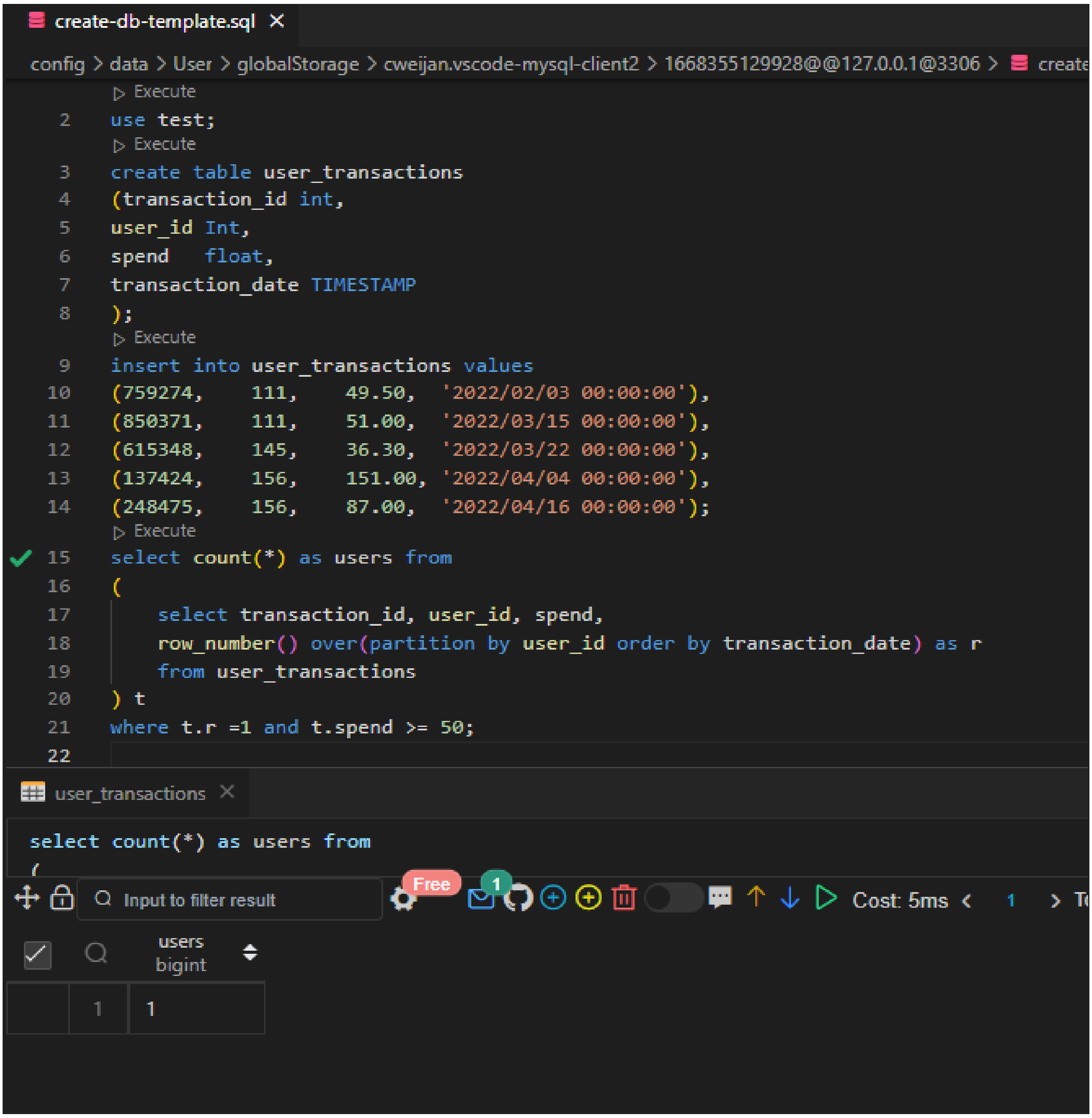
user_transactions Example Input:

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

```
Solution:
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;
```


create-db-template.sql
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create
▷ Execute
user_transactions
Input to filter result
Free
1
Cost: 5ms
1
T

```sql
use test;
create table user_transactions
(transaction_id int,
user_id Int,
spend   float,
transaction_date TIMESTAMP
);
insert into user_transactions values
(759274,    111,    49.50,  '2022/02/03 00:00:00'),
(850371,    111,    51.00,  '2022/03/15 00:00:00'),
(615348,    145,    36.30,  '2022/03/22 00:00:00'),
(137424,    156,    151.00, '2022/04/04 00:00:00'),
(248475,    156,    87.00,  '2022/04/16 00:00:00');
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;

```

```sql
select count(*) as users from
(
```

	users bigint
1	1

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:

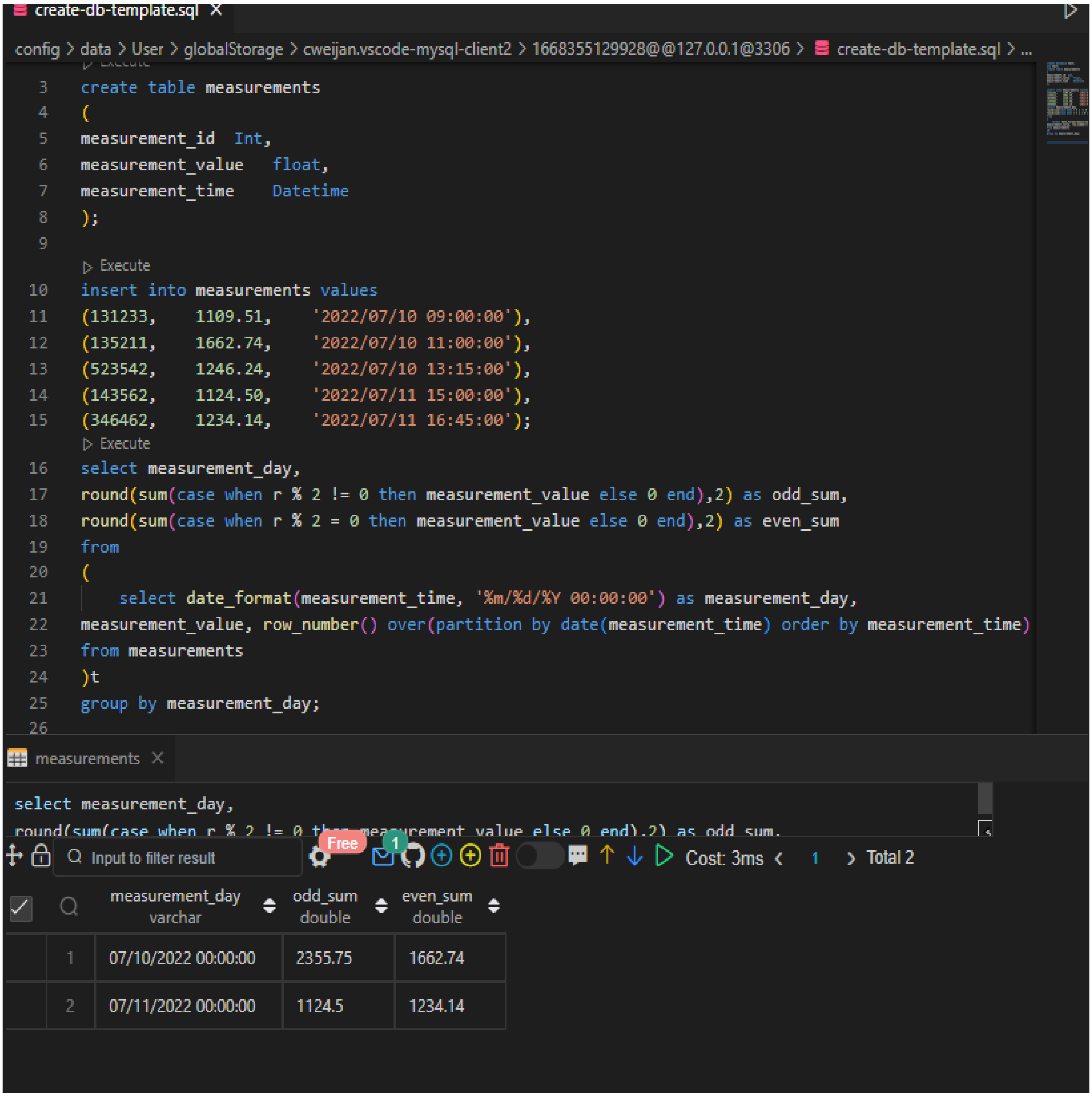
measurement_id	measurement_value	measurement_time
131233	1109.51	07/10/2022 09:00:00
135211	1662.74	07/10/2022 11:00:00
523542	1246.24	07/10/2022 13:15:00
143562	1124.50	07/11/2022 15:00:00
346462	1234.14	07/11/2022 16:45:00

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

```
Solution:
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;
```



create-db-template.sql ✕
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-template.sql > ...

```
create table measurements
(
measurement_id  Int,
measurement_value   float,
measurement_time    Datetime
);

▷ Execute
insert into measurements values
(131233,    1109.51,    '2022/07/10 09:00:00'),
(135211,    1662.74,    '2022/07/10 11:00:00'),
(523542,    1246.24,    '2022/07/10 13:15:00'),
(143562,    1124.50,    '2022/07/11 15:00:00'),
(346462,    1234.14,    '2022/07/11 16:45:00');
▷ Execute
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)
from measurements
)t
group by measurement_day;
```

```
select measurement_day,
round(sum(case when r % 2 != 0 then measurement_value else 0 end),2) as odd_sum,
```

	measurement_day varchar	odd_sum double	even_sum double
1	07/10/2022 00:00:00	2355.75	1662.74
2	07/11/2022 00:00:00	1124.5	1234.14

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

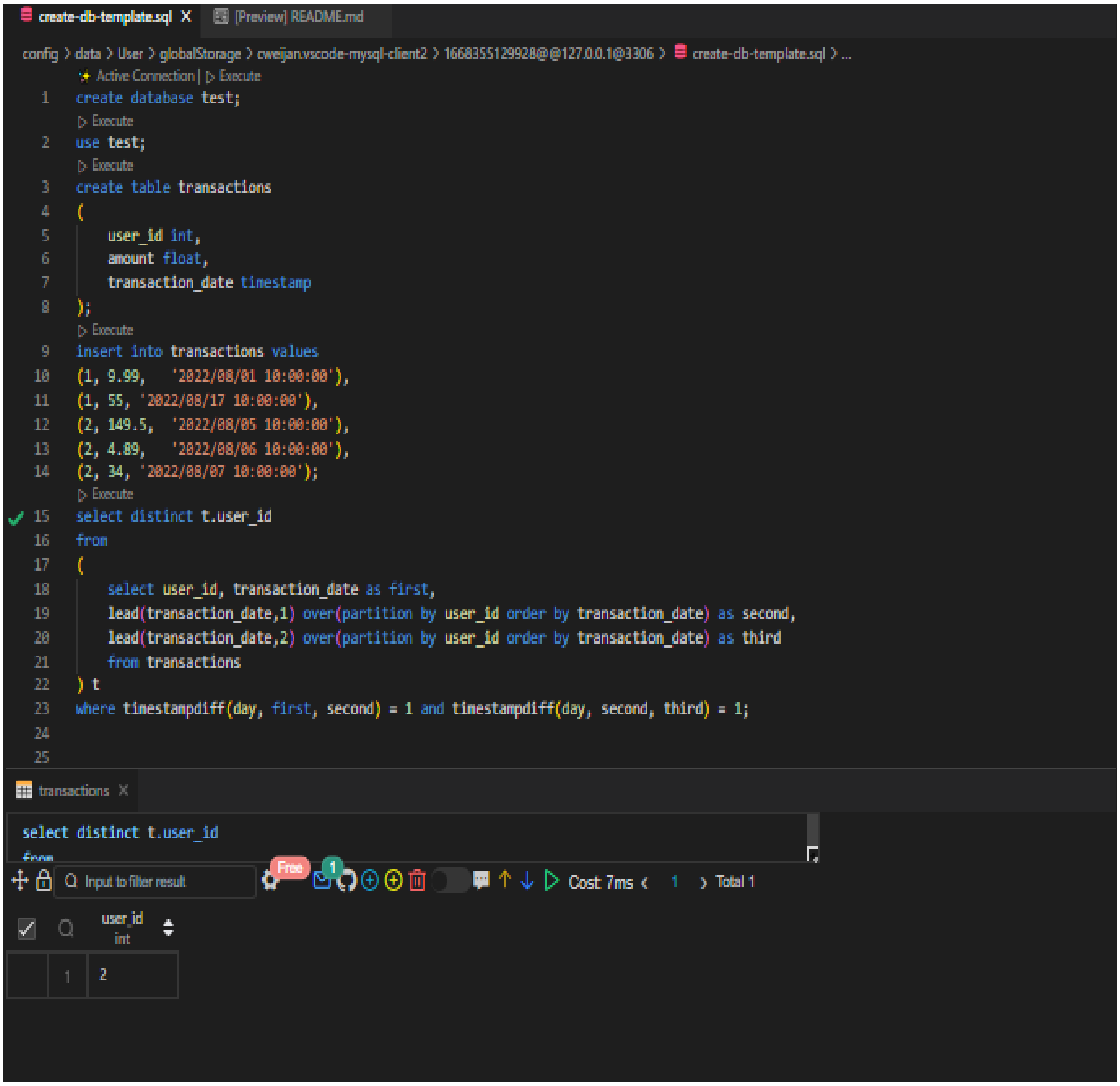
transactions Example Input:

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

```
Solution:
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;
```


create-db-template.sql
[Preview] README.md
config › data › User › globalStorage › cweijan.vscode-mysql-client2 › 1668355129928@@127.0.0.1@3306 › create-db-template.sql › ...
Active Connection | ▷ Execute
▷ Execute
transactions
Input to filter result
Free
Cost 7ms
Total 1

```sql
create database test;
use test;
create table transactions
(
    user_id int,
    amount float,
    transaction_date timestamp
);
insert into transactions values
(1, 9.99,   '2022/08/01 10:00:00'),
(1, 55, '2022/08/17 10:00:00'),
(2, 149.5,  '2022/08/05 10:00:00'),
(2, 4.89,   '2022/08/06 10:00:00'),
(2, 34, '2022/08/07 10:00:00');
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;
```

```
select distinct t.user_id
from
```

```
   user_id
   int
1  2
```

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:

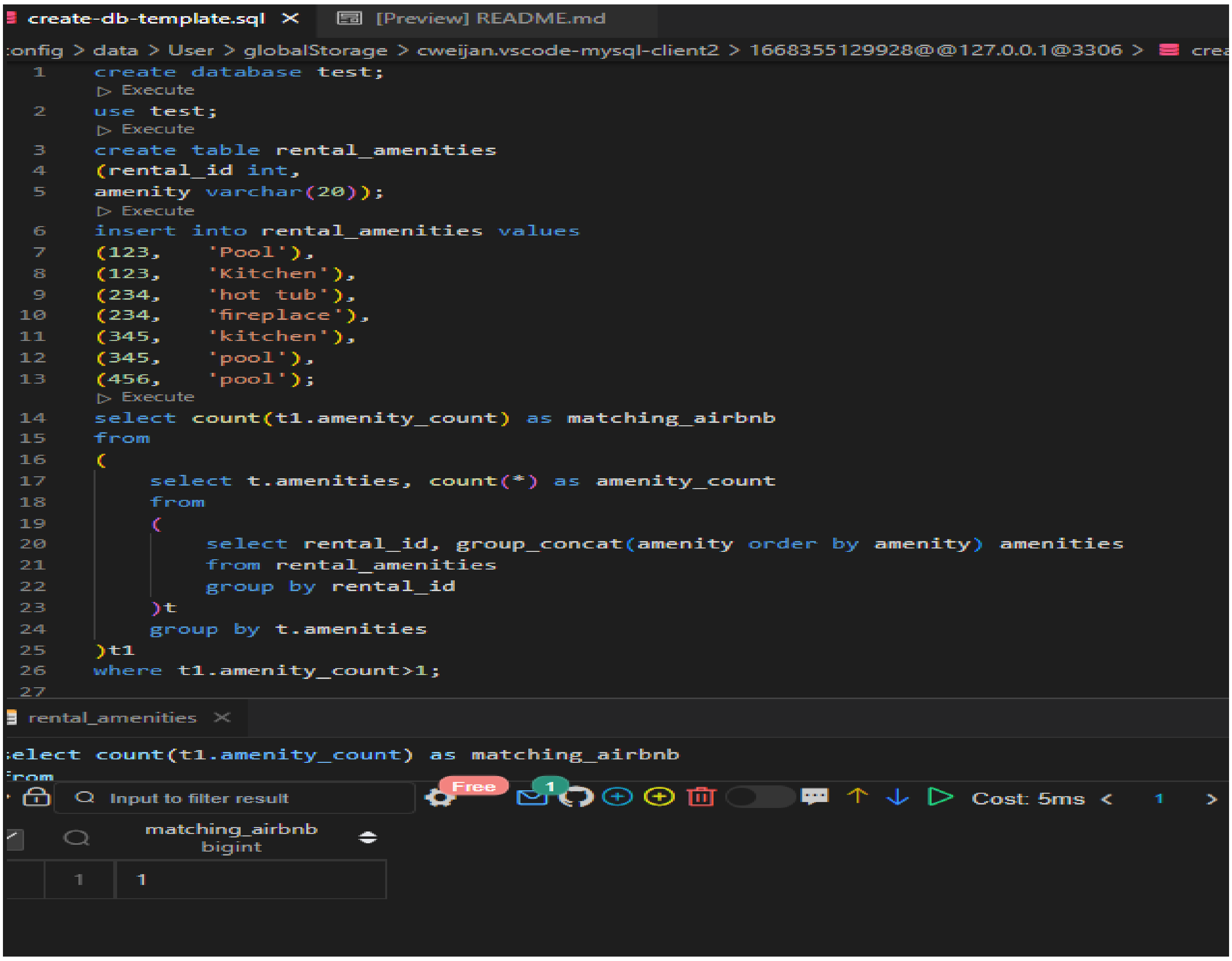
rental_id	Amenity
123	Pool
123	Kitchen
234	hot tub
234	fireplace
345	Kitchen

345	Pool
456	Pool

Example Output:

matching_airbnb
1

```
Solution:
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;
```


create-db-template.sql
[Preview] README.md
:onfig > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > crea
▷ Execute
rental_amenities
Input to filter result
Free
Cost: 5ms

```
create database test;
use test;
create table rental_amenities
(rental_id int,
amenity varchar(20));
insert into rental_amenities values
(123,   'Pool'),
(123,   'Kitchen'),
(234,   'hot tub'),
(234,   'fireplace'),
(345,   'kitchen'),
(345,   'pool'),
(456,   'pool');
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;
```

```
:elect count(t1.amenity_count) as matching_airbnb
'rom
```

	matching_airbnb bigint
1	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.
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

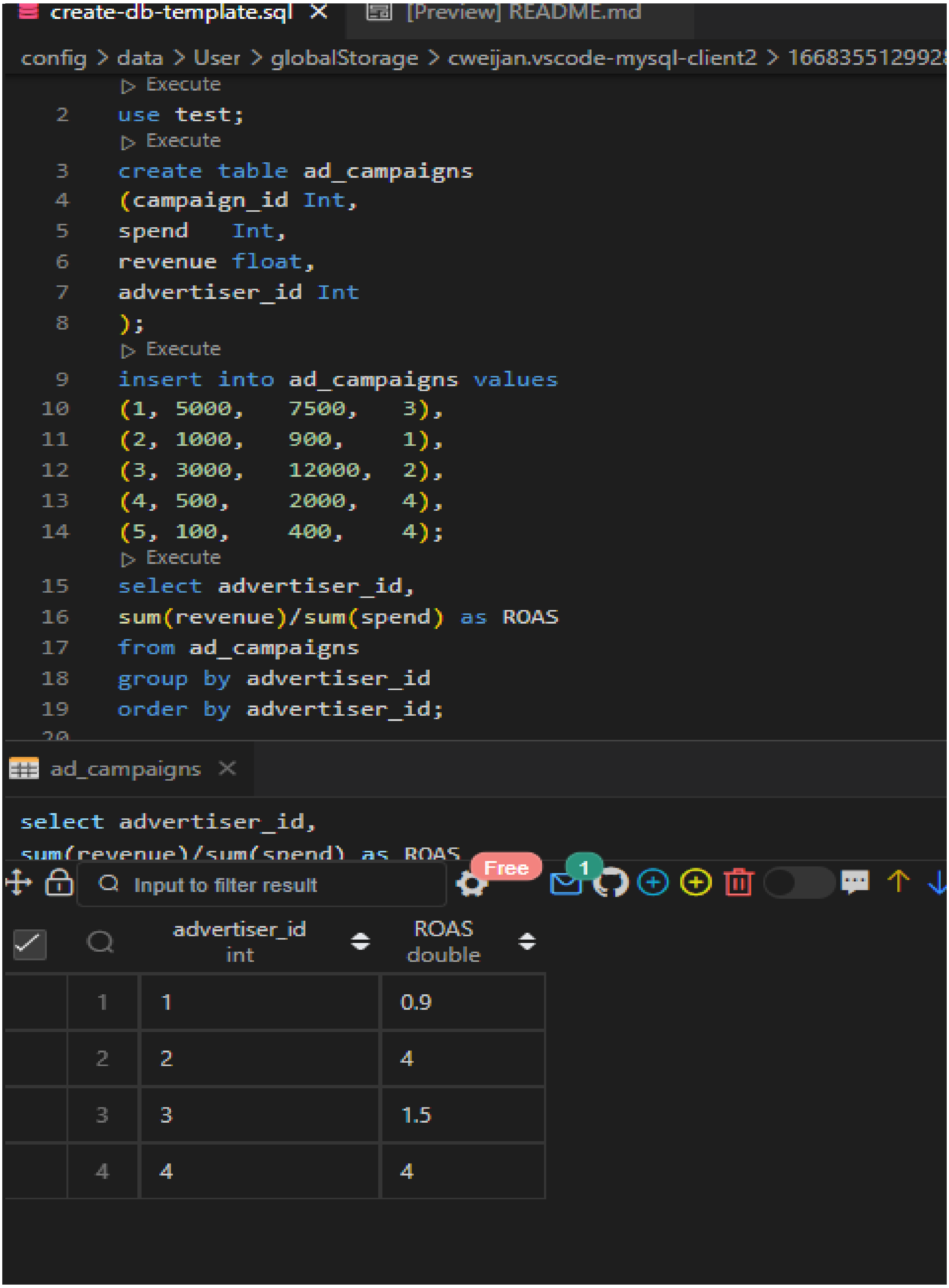
ad_campaigns Example Input:

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

```
Solution:
select advertiser_id,
sum(revenue)/sum(spend) as ROAS
from ad_campaigns
group by advertiser_id
order by advertiser_id;
```



```sql
use test;
create table ad_campaigns
(campaign_id Int,
spend   Int,
revenue float,
advertiser_id Int
);
insert into ad_campaigns values
(1, 5000,   7500,   3),
(2, 1000,   900,    1),
(3, 3000,   12000,  2),
(4, 500,    2000,   4),
(5, 100,    400,    4);
select advertiser_id,
sum(revenue)/sum(spend) as ROAS
from ad_campaigns
group by advertiser_id
order by advertiser_id;
```

```sql
select advertiser_id,
sum(revenue)/sum(spend) as ROAS
```

	advertiser_id int	ROAS double
1	1	0.9
2	2	4
3	3	1.5
4	4	4

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:

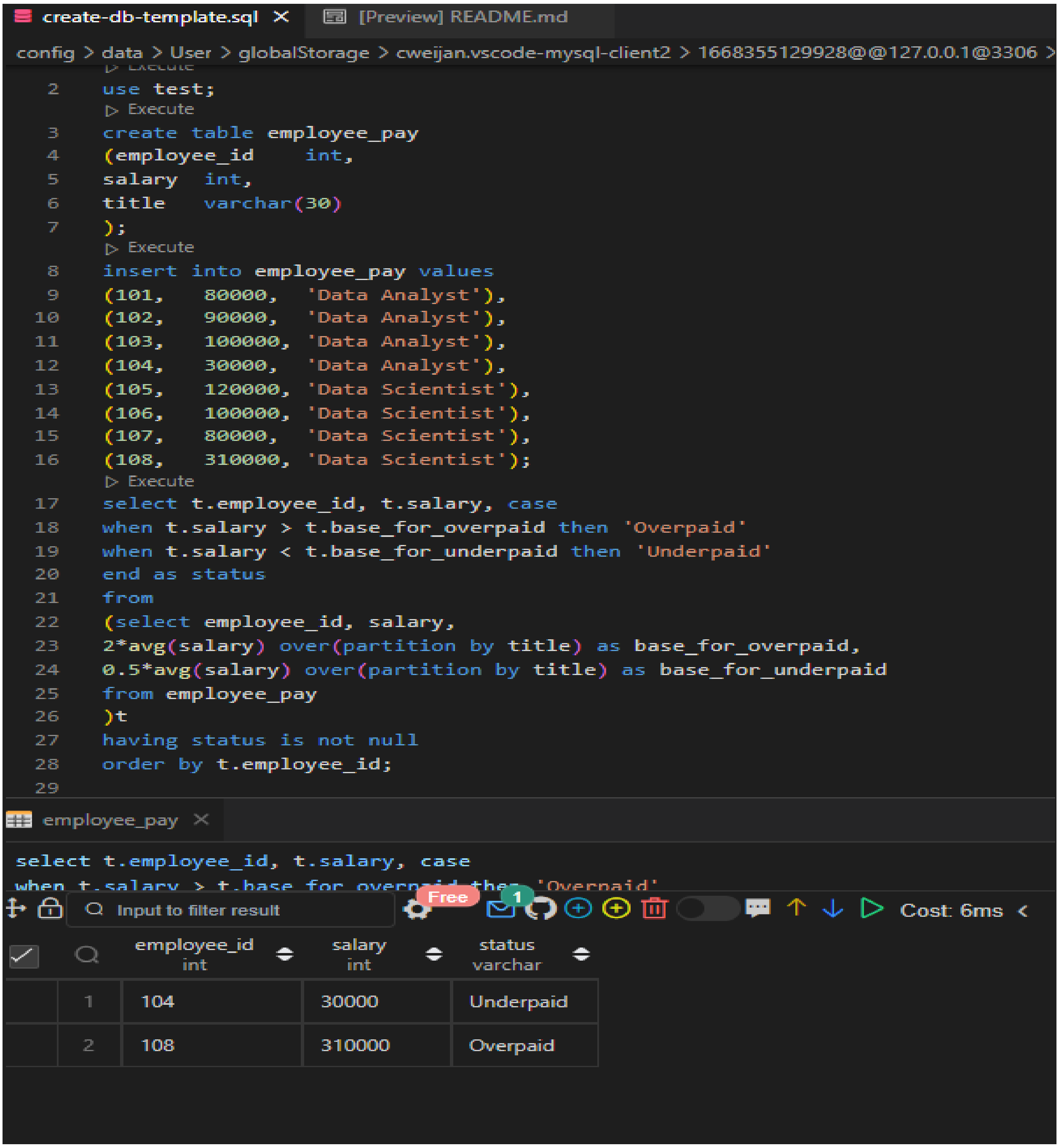
employee_id	salary	Title
101	80000	Data Analyst
102	90000	Data Analyst
103	100000	Data Analyst
104	30000	Data Analyst

105	120000	Data Scientist
106	100000	Data Scientist
107	80000	Data Scientist
108	310000	Data Scientist

employee_id	Salary	status
104	30000	Underpaid
108	310000	Overpaid

Example Output:

```
Solution:
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;
```


create-db-template.sql
[Preview] README.md
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 >
Execute
employee_pay
Input to filter result
Free
1
Cost: 6ms

```sql
use test;
create table employee_pay
(employee_id    int,
salary  int,
title   varchar(30)
);
insert into employee_pay values
(101,   80000,  'Data Analyst'),
(102,   90000,  'Data Analyst'),
(103,   100000, 'Data Analyst'),
(104,   30000,  'Data Analyst'),
(105,   120000, 'Data Scientist'),
(106,   100000, 'Data Scientist'),
(107,   80000,  'Data Scientist'),
(108,   310000, 'Data Scientist');
select 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;
```

```sql
select t.employee_id, t.salary, case
when t.salary > t.base_for_overpaid then 'Overpaid'
```

	employee_id int	salary int	status varchar
1	104	30000	Underpaid
2	108	310000	Overpaid

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

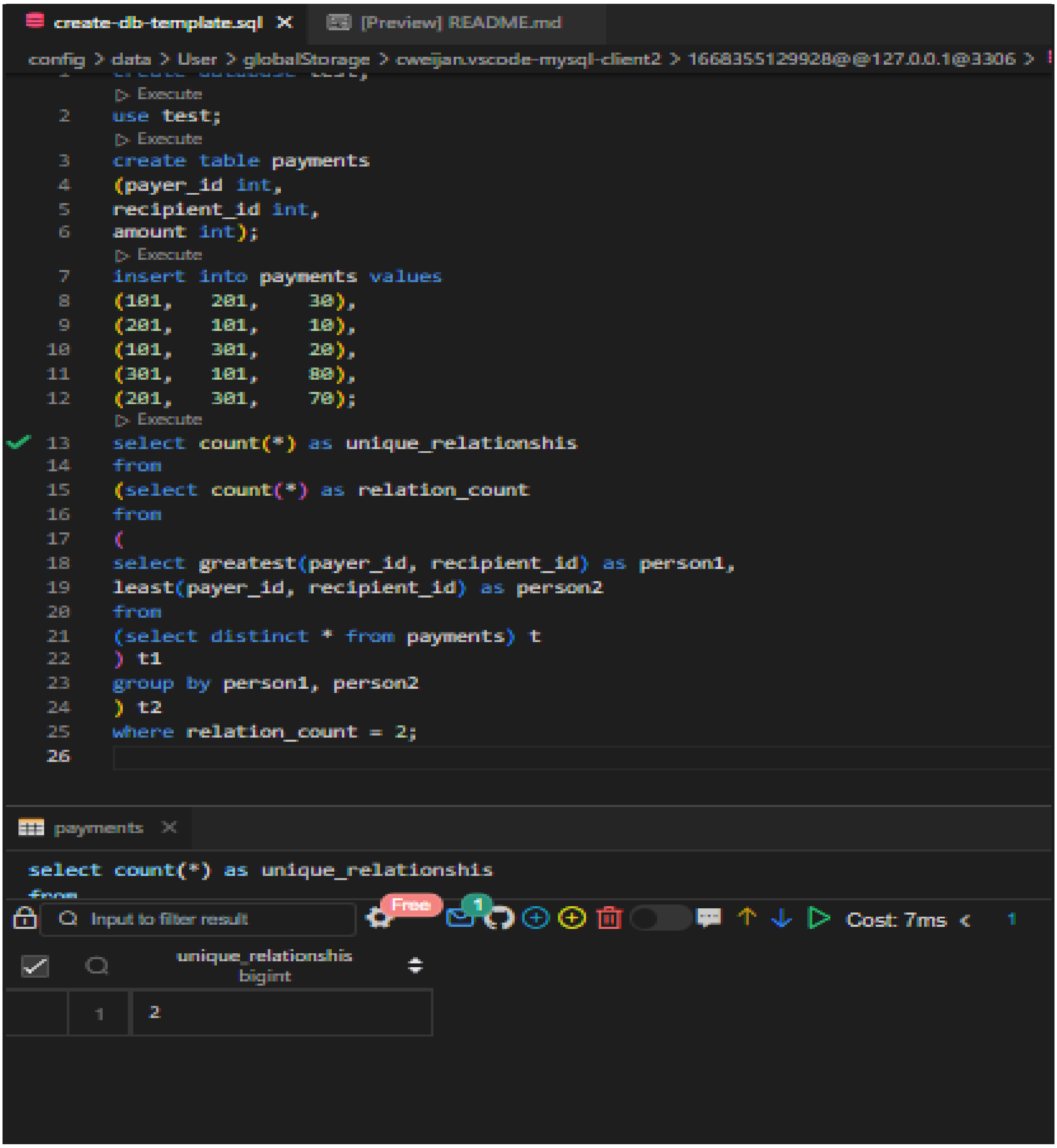
payments Example Input:

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

```
Solution:
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;
```


create-db-template.sql
[Preview] README.md
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 >
▷ Execute
payments
Input to filter result
Free
Cost: 7ms

```
create database test;
use test;
create table payments
(payer_id int,
recipient_id int,
amount int);
insert into payments values
(101,    201,    30),
(201,    101,    10),
(101,    301,    20),
(301,    101,    80),
(201,    301,    70);
select 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;
```

```
select count(*) as unique_relationshis
from
```

	unique_relationshis bigint
1	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 oficial output may vary from before.

Hint- Count the distinct number of dates formatted into the DATE format in the COUNT(DISTINCT).

purchases Table:

Column Name	Type

user_id	Integer
product_id	Integer
quantity	Integer
purchase_date	Datetime

purchasesExample Input:

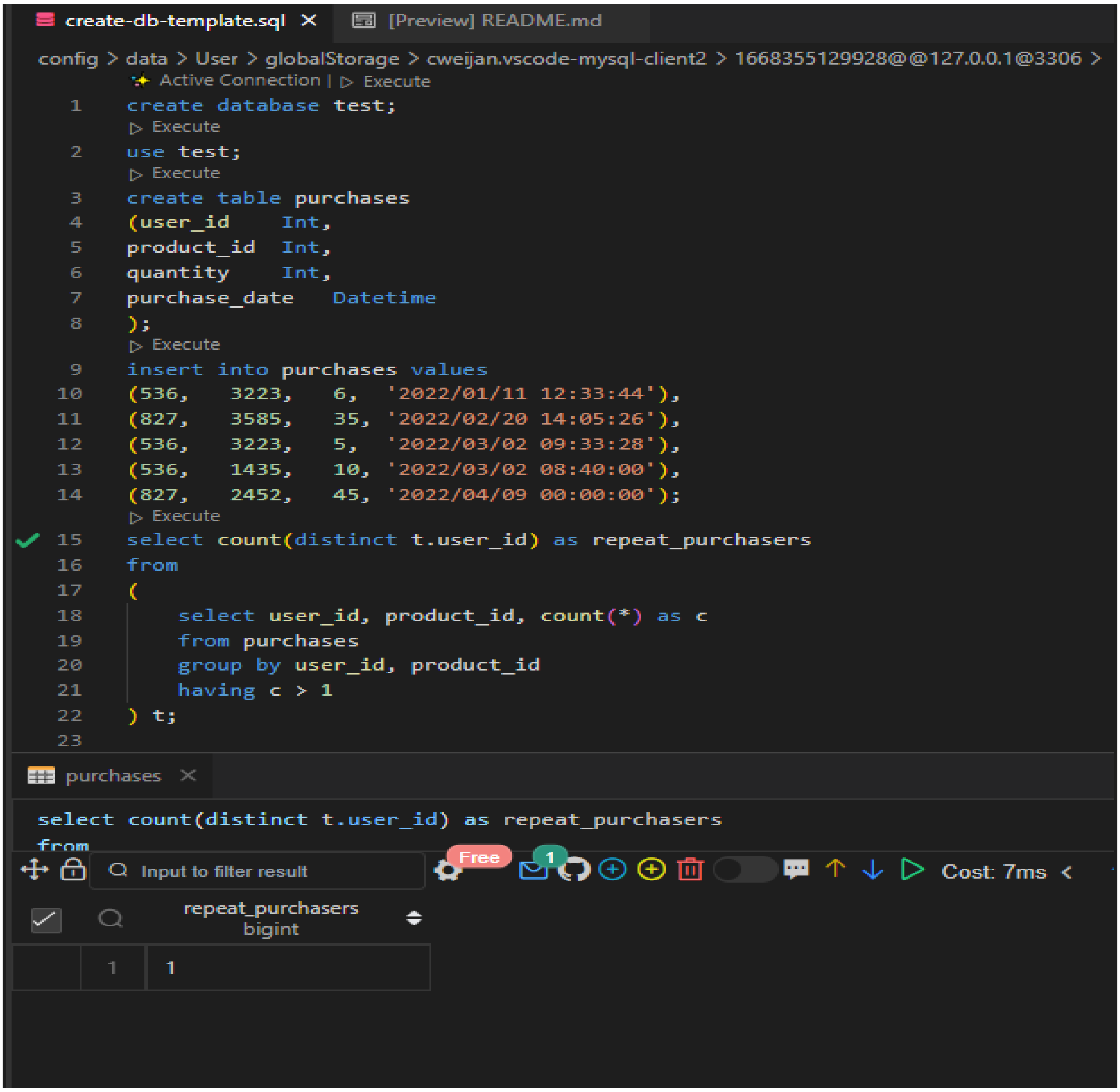
user_id	product_id	quantity	purchase_date
536	3223	6	01/11/2022 12:33:44

827	3585	35	02/20/2022 14:05:26
536	3223	5	03/02/2022 09:33:28
536	1435	10	03/02/2022 08:40:00
827	2452	45	04/09/2022 00:00:00

Example Output:

repeat_purchasers
1

```
Solution:
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;
```


create-db-template.sql
[Preview] README.md
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 >
Active Connection | ▷ Execute
▷ Execute
purchases
Input to filter result
Free
Cost: 7ms

```sql
create database test;
use test;
create table purchases
(user_id    Int,
product_id  Int,
quantity    Int,
purchase_date   Datetime
);
insert into purchases values
(536,   3223,   6,  '2022/01/11 12:33:44'),
(827,   3585,   35, '2022/02/20 14:05:26'),
(536,   3223,   5,  '2022/03/02 09:33:28'),
(536,   1435,   10, '2022/03/02 08:40:00'),
(827,   2452,   45, '2022/04/09 00:00:00');
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;
```

```sql
select count(distinct t.user_id) as repeat_purchasers
from
```

```
   repeat_purchasers
   bigint
1  1
```

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

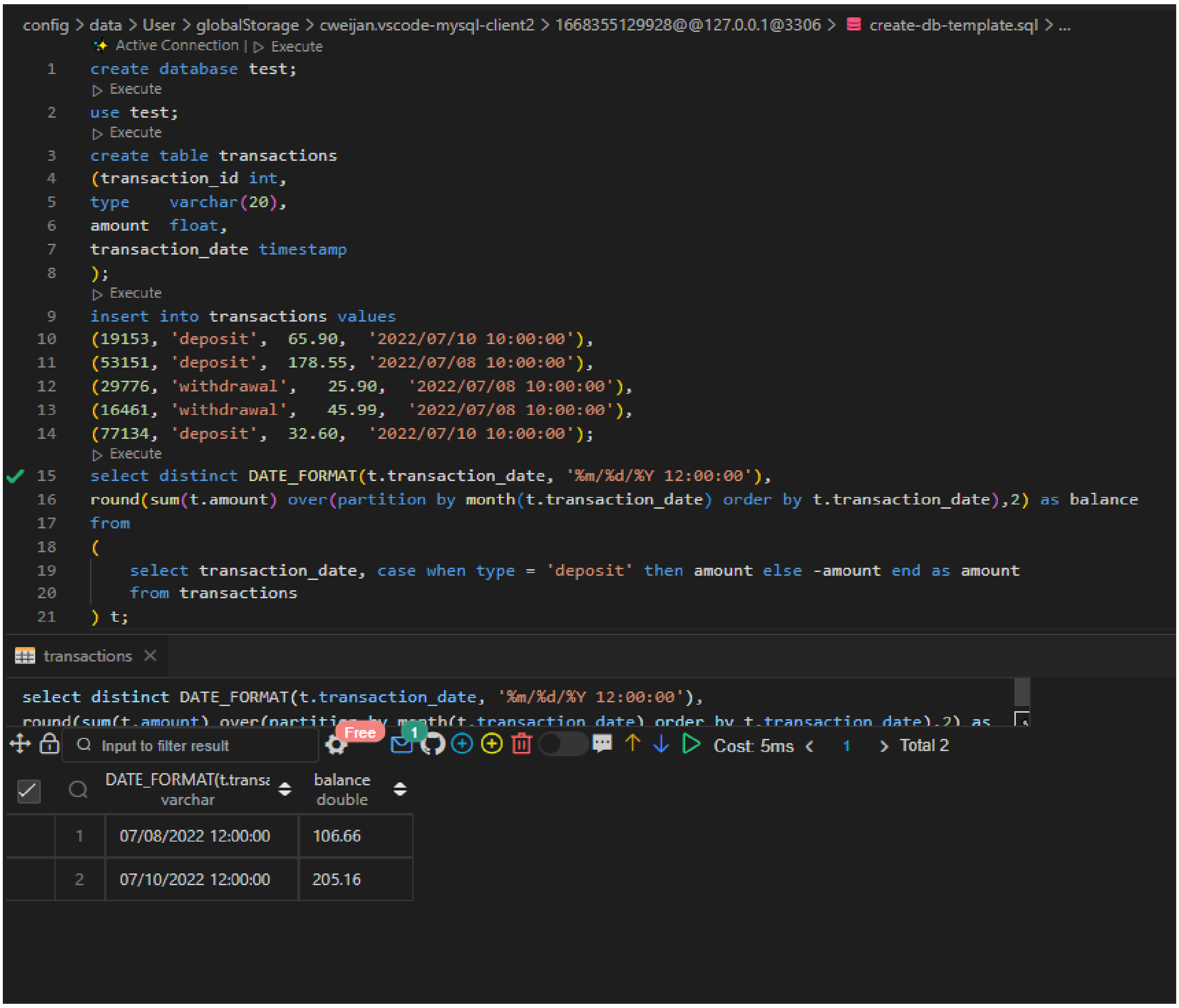
29776	Withdrawal	25.90	07/08/2022 10:00:00
16461	Withdrawal	45.99	07/08/2022 10:00:00
77134	Deposit	32.60	07/10/2022 10:00:00

Example Output:

transaction_date	balance
07/08/2022 12:00:00	106.66
07/10/2022 12:00:00	205.16

Solution:

```
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;
```


config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-template.sql > ...
Active Connection | ▷ Execute
▷ Execute

```
create database test;
use test;
create table transactions
(transaction_id int,
type    varchar(20),
amount  float,
transaction_date timestamp
);
insert into transactions values
(19153, 'deposit',  65.90,  '2022/07/10 10:00:00'),
(53151, 'deposit',  178.55, '2022/07/08 10:00:00'),
(29776, 'withdrawal',   25.90,  '2022/07/08 10:00:00'),
(16461, 'withdrawal',   45.99,  '2022/07/08 10:00:00'),
(77134, 'deposit',  32.60,  '2022/07/10 10:00:00');
select distinct DATE_FORMAT(t.transaction_date, '%m/%d/%Y 12:00:00'),
round(sum(t.amount) over(partition by month(t.transaction_date) order by t.transaction_date),2) as balance
from
(
    select transaction_date, case when type = 'deposit' then amount else -amount end as amount
    from transactions
) t;
```

```
select distinct DATE_FORMAT(t.transaction_date, '%m/%d/%Y 12:00:00'),
round(sum(t.amount) over(partition by month(t.transaction_date) order by t.transaction_date),2) as
```

	DATE_FORMAT(t.transa varchar	balance double
1	07/08/2022 12:00:00	106.66
2	07/10/2022 12:00:00	205.16

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:

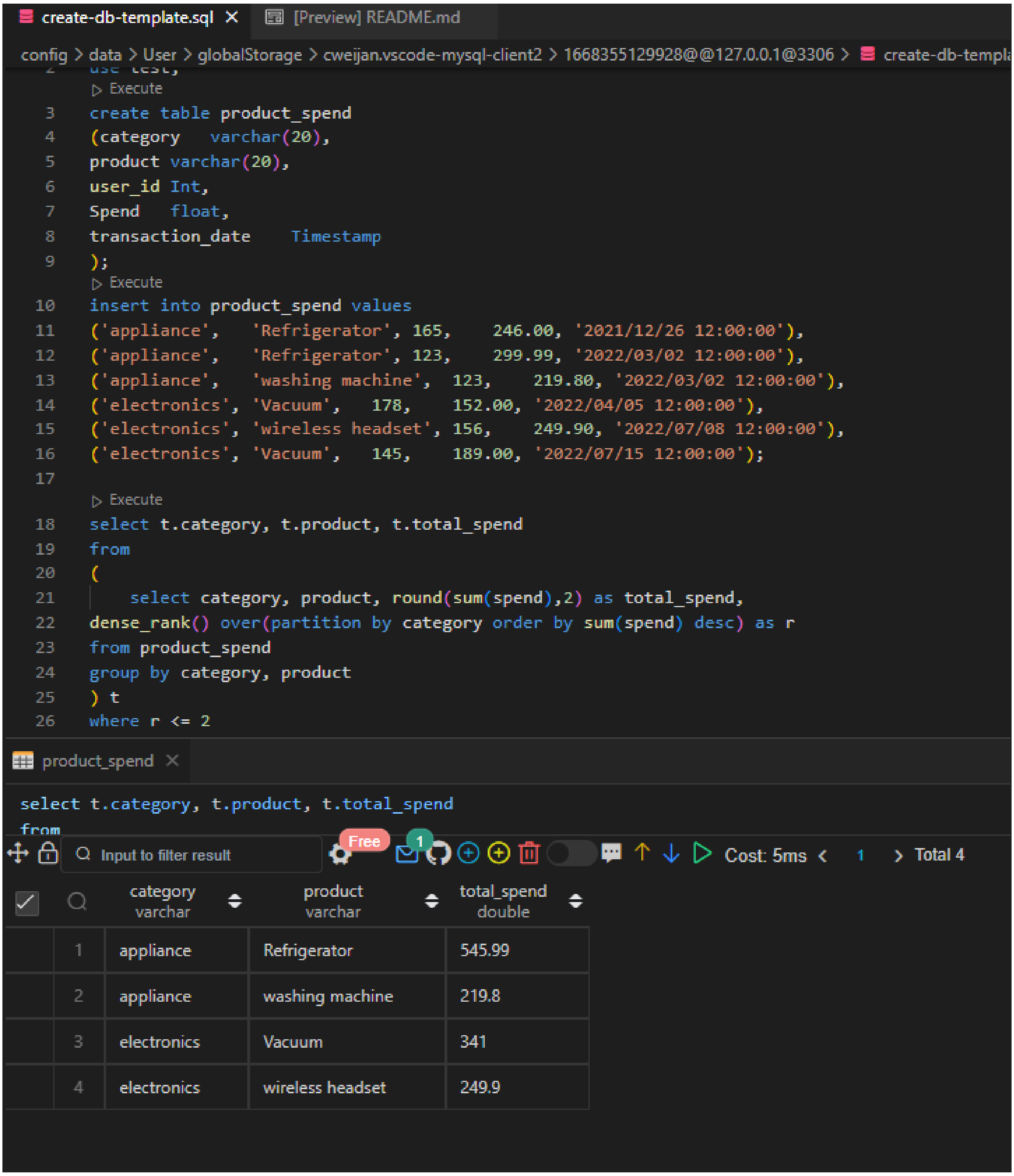
Category	Product	user_id	spend	transaction_date

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	545.99
Appliance	washing machine	219.80
electronics	Vacuum	341.00
electronics	wireless headset	249.90

```
Solution:
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
```



```sql
use test;
create table product_spend
(category   varchar(20),
product varchar(20),
user_id Int,
Spend   float,
transaction_date    Timestamp
);
insert into product_spend values
('appliance',   'Refrigerator', 165,    246.00, '2021/12/26 12:00:00'),
('appliance',   'Refrigerator', 123,    299.99, '2022/03/02 12:00:00'),
('appliance',   'washing machine',  123,    219.80, '2022/03/02 12:00:00'),
('electronics', 'Vacuum',   178,    152.00, '2022/04/05 12:00:00'),
('electronics', 'wireless headset', 156,    249.90, '2022/07/08 12:00:00'),
('electronics', 'Vacuum',   145,    189.00, '2022/07/15 12:00:00');

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
```

```sql
select t.category, t.product, t.total_spend
from
```

	category varchar	product varchar	total_spend double
1	appliance	Refrigerator	545.99
2	appliance	washing machine	219.8
3	electronics	Vacuum	341
4	electronics	wireless headset	249.9

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

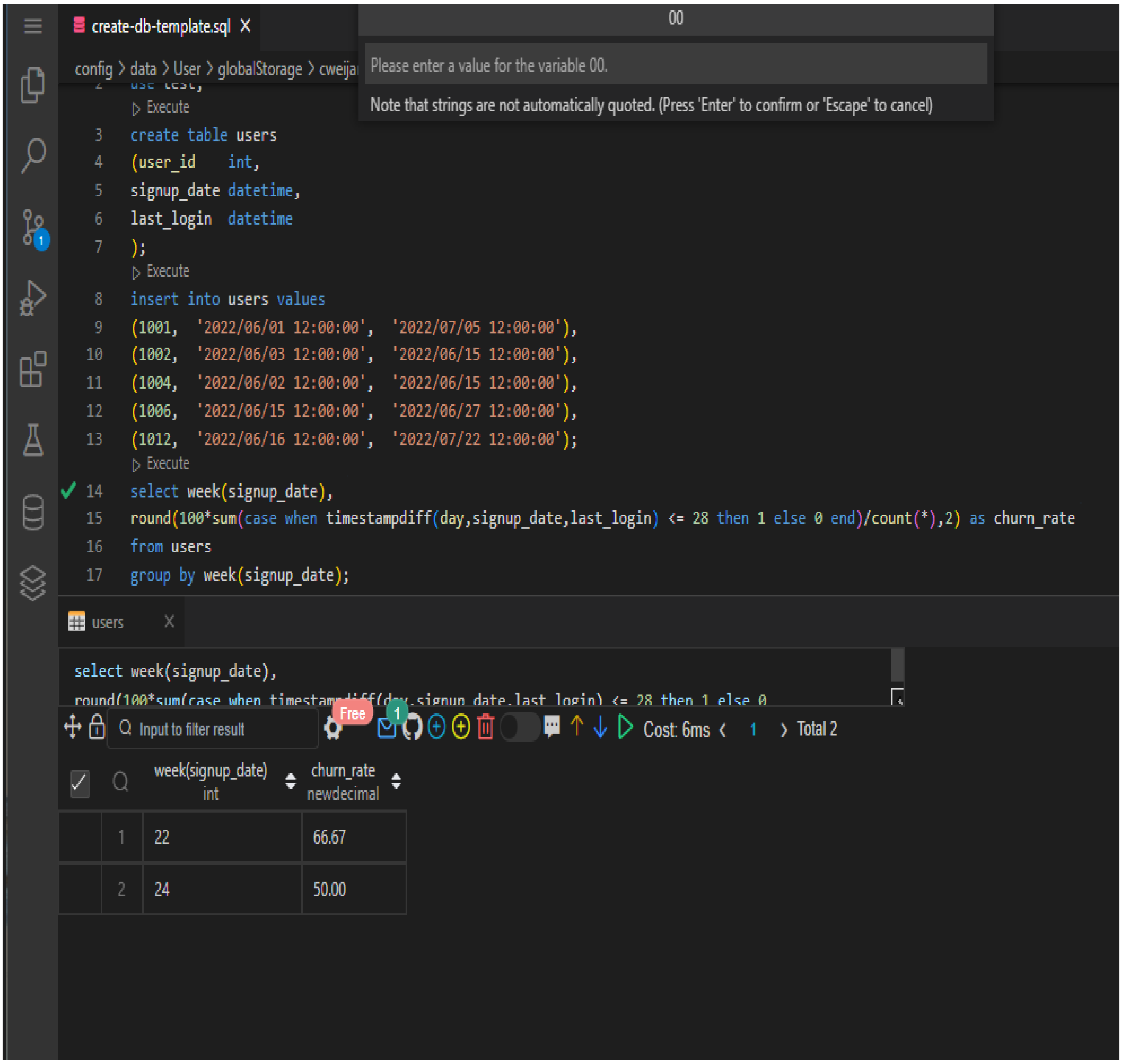
Example Output:

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%.

```
Solution:  according to week of year
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);
```



00

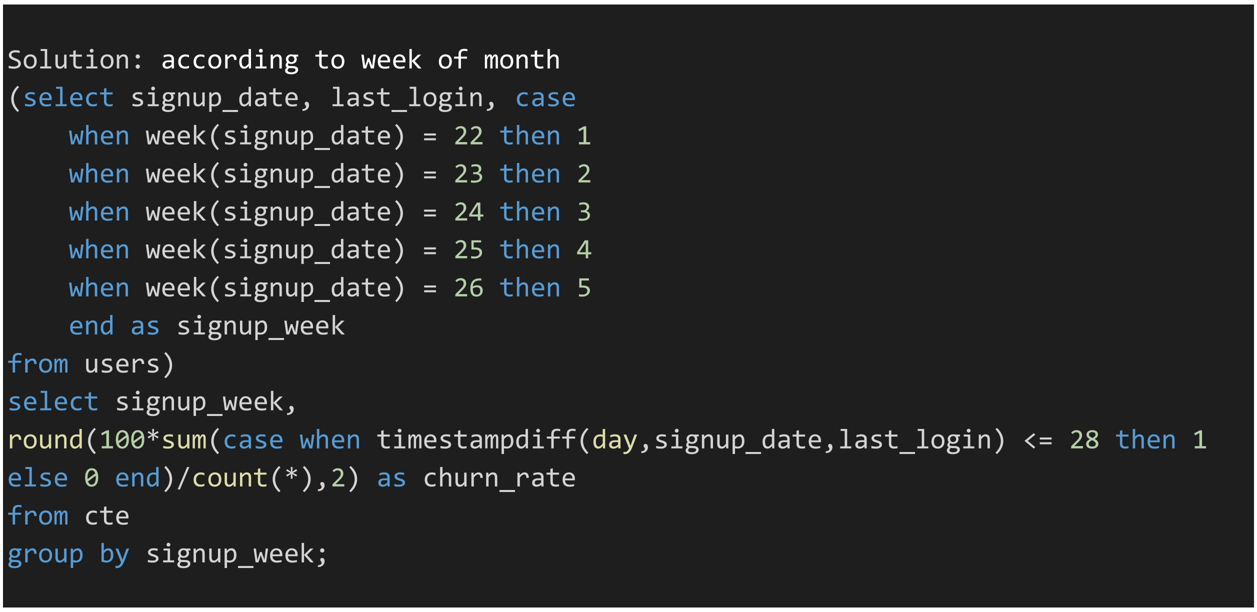
Please enter a value for the variable 00.

Note that strings are not automatically quoted. (Press 'Enter' to confirm or 'Escape' to cancel)

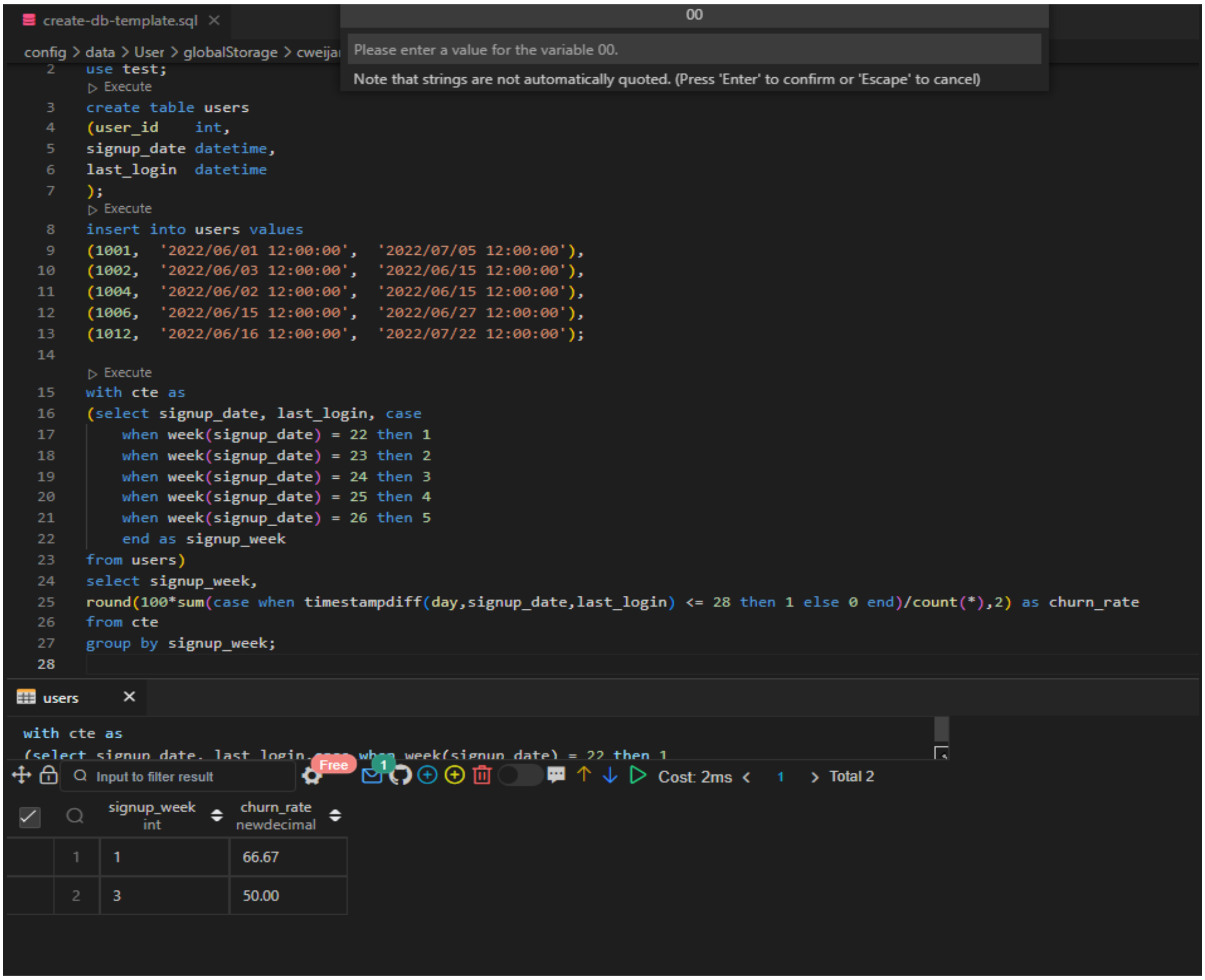
```
use test;
▷ Execute
create table users
(user_id    int,
signup_date datetime,
last_login  datetime
);
▷ Execute
insert into users values
(1001,  '2022/06/01 12:00:00',  '2022/07/05 12:00:00'),
(1002,  '2022/06/03 12:00:00',  '2022/06/15 12:00:00'),
(1004,  '2022/06/02 12:00:00',  '2022/06/15 12:00:00'),
(1006,  '2022/06/15 12:00:00',  '2022/06/27 12:00:00'),
(1012,  '2022/06/16 12:00:00',  '2022/07/22 12:00:00');
▷ Execute
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);
```

```
select week(signup_date),
round(100*sum(case when timestampdiff(day,signup_date,last_login) <= 28 then 1 else 0
```

☑		week(signup_date) int	churn_rate newdecimal
	1	22	66.67
	2	24	50.00



```
Solution: according to week of month
(select signup_date, last_login, case
    when week(signup_date) = 22 then 1
    when week(signup_date) = 23 then 2
    when week(signup_date) = 24 then 3
    when week(signup_date) = 25 then 4
    when week(signup_date) = 26 then 5
    end as signup_week
from users)
select signup_week,
round(100*sum(case when timestampdiff(day,signup_date,last_login) <= 28 then 1
else 0 end)/count(*),2) as churn_rate
from cte
group by signup_week;
```



00

Please enter a value for the variable 00.

Note that strings are not automatically quoted. (Press 'Enter' to confirm or 'Escape' to cancel)

```sql
use test;
create table users
(user_id    int,
signup_date datetime,
last_login  datetime
);
insert into users values
(1001,  '2022/06/01 12:00:00',  '2022/07/05 12:00:00'),
(1002,  '2022/06/03 12:00:00',  '2022/06/15 12:00:00'),
(1004,  '2022/06/02 12:00:00',  '2022/06/15 12:00:00'),
(1006,  '2022/06/15 12:00:00',  '2022/06/27 12:00:00'),
(1012,  '2022/06/16 12:00:00',  '2022/07/22 12:00:00');

with cte as
(select signup_date, last_login, case
    when week(signup_date) = 22 then 1
    when week(signup_date) = 23 then 2
    when week(signup_date) = 24 then 3
    when week(signup_date) = 25 then 4
    when week(signup_date) = 26 then 5
    end as signup_week
from users)
select signup_week,
round(100*sum(case when timestampdiff(day,signup_date,last_login) <= 28 then 1 else 0 end)/count(*),2) as churn_rate
from cte
group by signup_week;
```

```sql
with cte as
(select signup_date, last_login, case when week(signup_date) = 22 then 1
```

	signup_week int	churn_rate newdecimal
1	1	66.67
2	3	50.00