Q51.
World table:

Column Name	Type
name	varchar
continent	varchar
area	Int
population	Int
gdp	Int

name is the primary key column for this table.
Each row of this table gives information about the name of a country, the continent to which it belongs, its area, the population, and its GDP value.
A country is big if:

- it has an area of at least three million (i.e., 3000000 km2), or
- it has a population of at least twenty-five million (i.e., 25000000).

Write an SQL query to report the name, population, and area of the big countries.
Return the result table in any order.
The query result format is in the following example.

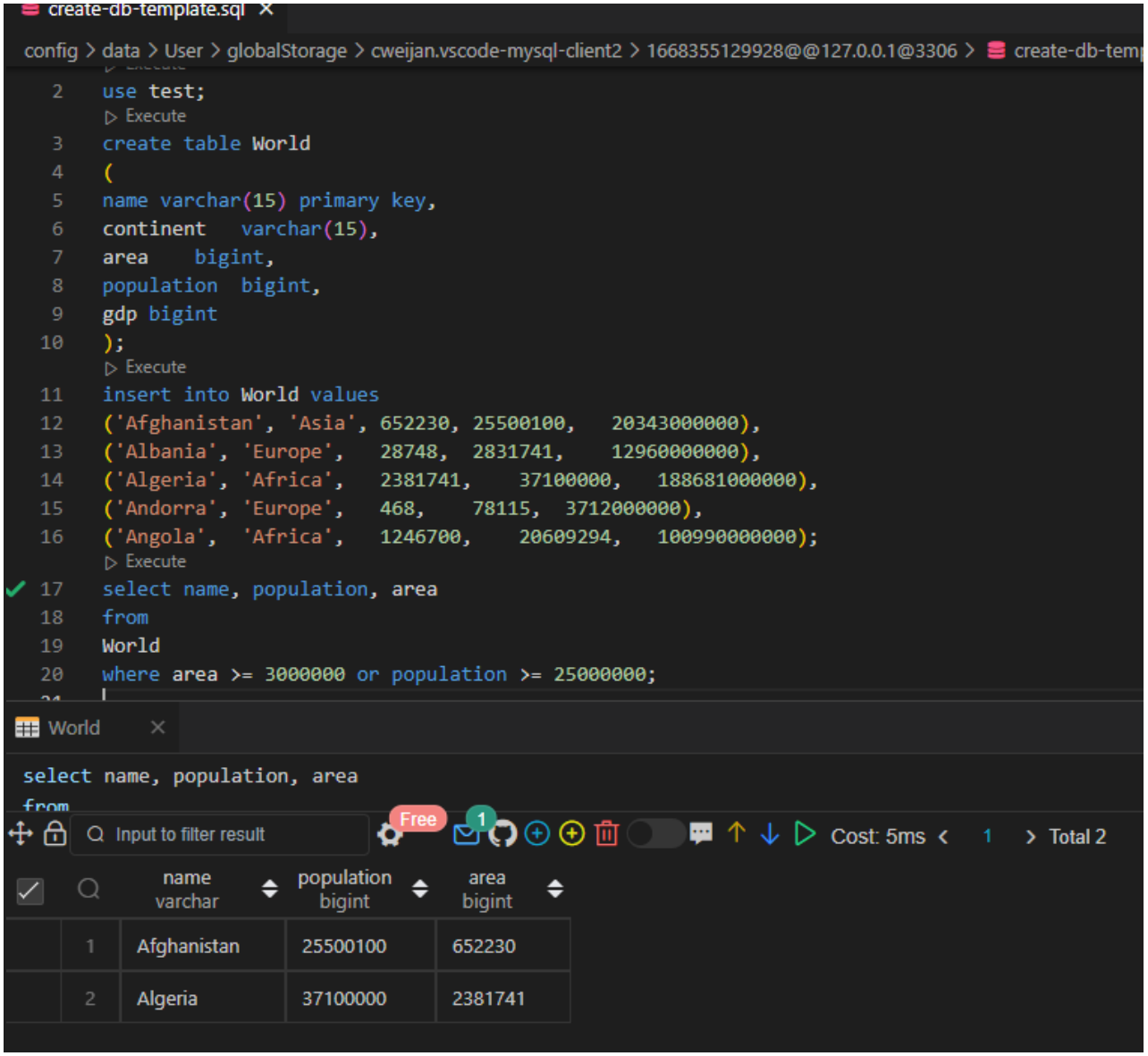
Input:
World table:

Name	continent	Area	population	gdp
Afghanistan	Asia	652230	25500100	20343000000
Albania	Europe	28748	2831741	12960000000
Algeria	Africa	2381741	37100000	188681000000
Andorra	Europe	468	78115	3712000000
Angola	Africa	1246700	20609294	100990000000

Output:

name	population	Area
Afghanistan	25500100	652230
Algeria	37100000	2381741

```
Solution:
select name, population, area
from
World
where area >= 3000000 or population >= 25000000;
```


create-db-template.sql
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-temp
use test;
Execute
create table World
(
name varchar(15) primary key,
continent varchar(15),
area bigint,
population bigint,
gdp bigint
);
Execute
insert into World values
('Afghanistan', 'Asia', 652230, 25500100, 20343000000),
('Albania', 'Europe', 28748, 2831741, 12960000000),
('Algeria', 'Africa', 2381741, 37100000, 188681000000),
('Andorra', 'Europe', 468, 78115, 3712000000),
('Angola', 'Africa', 1246700, 20609294, 100990000000);
Execute
select name, population, area
from
World
where area >= 3000000 or population >= 25000000;
World
select name, population, area
from
Input to filter result
Free
Cost: 5ms
1
Total 2
name
varchar
population
bigint
area
bigint
1
Afghanistan
25500100
652230
2
Algeria
37100000
2381741

Q52.
Table: Customer

Column Name	Type
id	int
name	varchar
referee_id	int

id is the primary key column for this table.
Each row of this table indicates the id of a customer, their name, and the id of the customer who referred them.

Write an SQL query to report the names of the customer that are not referred by the customer with id = 2.
Return the result table in any order.
The query result format is in the following example.

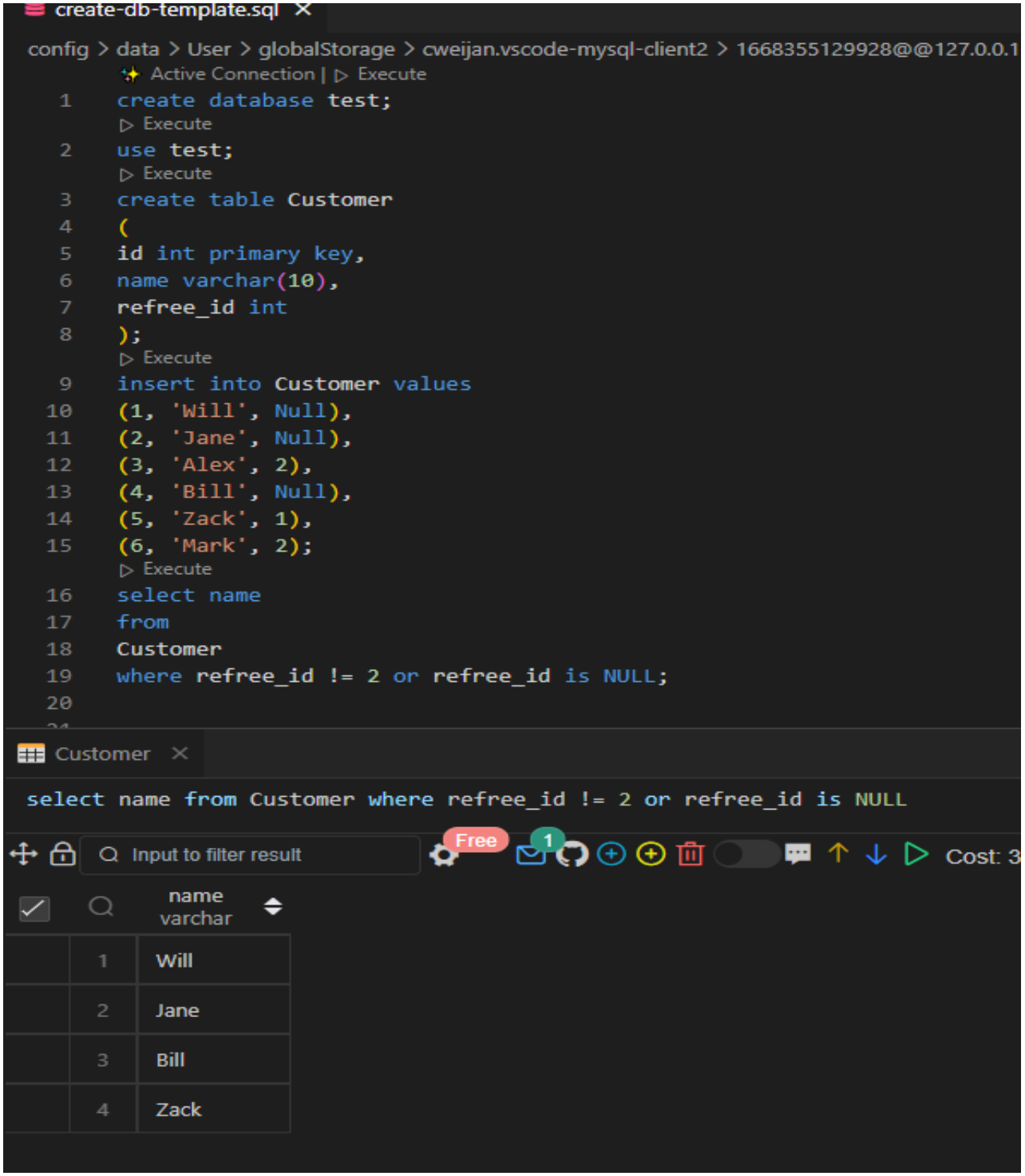
Input:
Customer table:

id	name	referee_id
1	Will	Null
2	Jane	Null
3	Alex	2
4	Bill	Null
5	Zack	1
6	Mark	2

Output:

Name
Will
Jane
Bill
Zack

```
Solution:
select name
from
Customer
where refree_id != 2 or refree_id is NULL;
```


create-db-template.sql
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1
Active Connection | ▷ Execute
1 create database test;
▷ Execute
2 use test;
▷ Execute
3 create table Customer
4 (
5 id int primary key,
6 name varchar(10),
7 refree_id int
8);
▷ Execute
9 insert into Customer values
10 (1, 'Will', Null),
11 (2, 'Jane', Null),
12 (3, 'Alex', 2),
13 (4, 'Bill', Null),
14 (5, 'Zack', 1),
15 (6, 'Mark', 2);
▷ Execute
16 select name
17 from
18 Customer
19 where refree_id != 2 or refree_id is NULL;
20
Customer
select name from Customer where refree_id != 2 or refree_id is NULL
Input to filter result
Free
1
Cost: 3
name
varchar
1 Will
2 Jane
3 Bill
4 Zack

Q53.

Table: Customers

Column Name	Type
id	int
name	varchar

id is the primary key column for this table.
Each row of this table indicates the ID and name of a customer.

Table: Orders

Column Name	Type
id	int
customerId	int

id is the primary key column for this table.
customerId is a foreign key of the ID from the Customers table.
Each row of this table indicates the ID of an order and the ID of the customer who ordered it.

Write an SQL query to report all customers who never order anything.
Return the result table in any order.
The query result format is in the following example.

Input:
Customers table:

id	name
1	Joe
2	Henry
3	Sam
4	Max

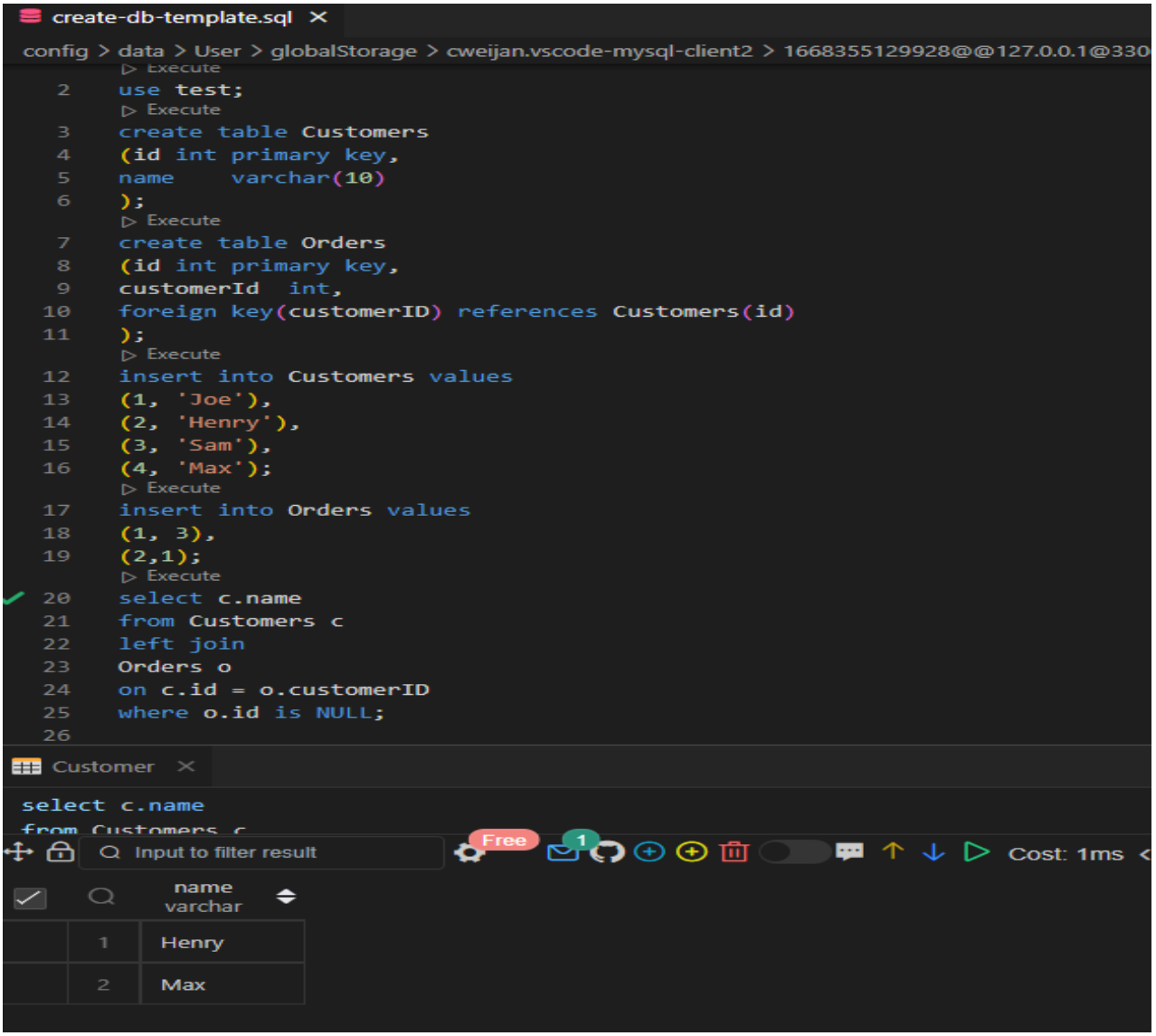
Orders table:

id	customerId
1	3
2	1

Output:

Customers
Henry
Max

```
Solution:
select c.name
from Customers c
left join
Orders o
on c.id = o.customerID
where o.id is NULL;
```



Q54.

Table: Employee

Column Name	Type
employee_id	int
team_id	int

employee_id is the primary key for this table.
Each row of this table contains the ID of each employee and their respective team.

Write an SQL query to find the team size of each of the employees.

Return result table in any order.
The query result format is in the following example.

Input:
Employee Table:

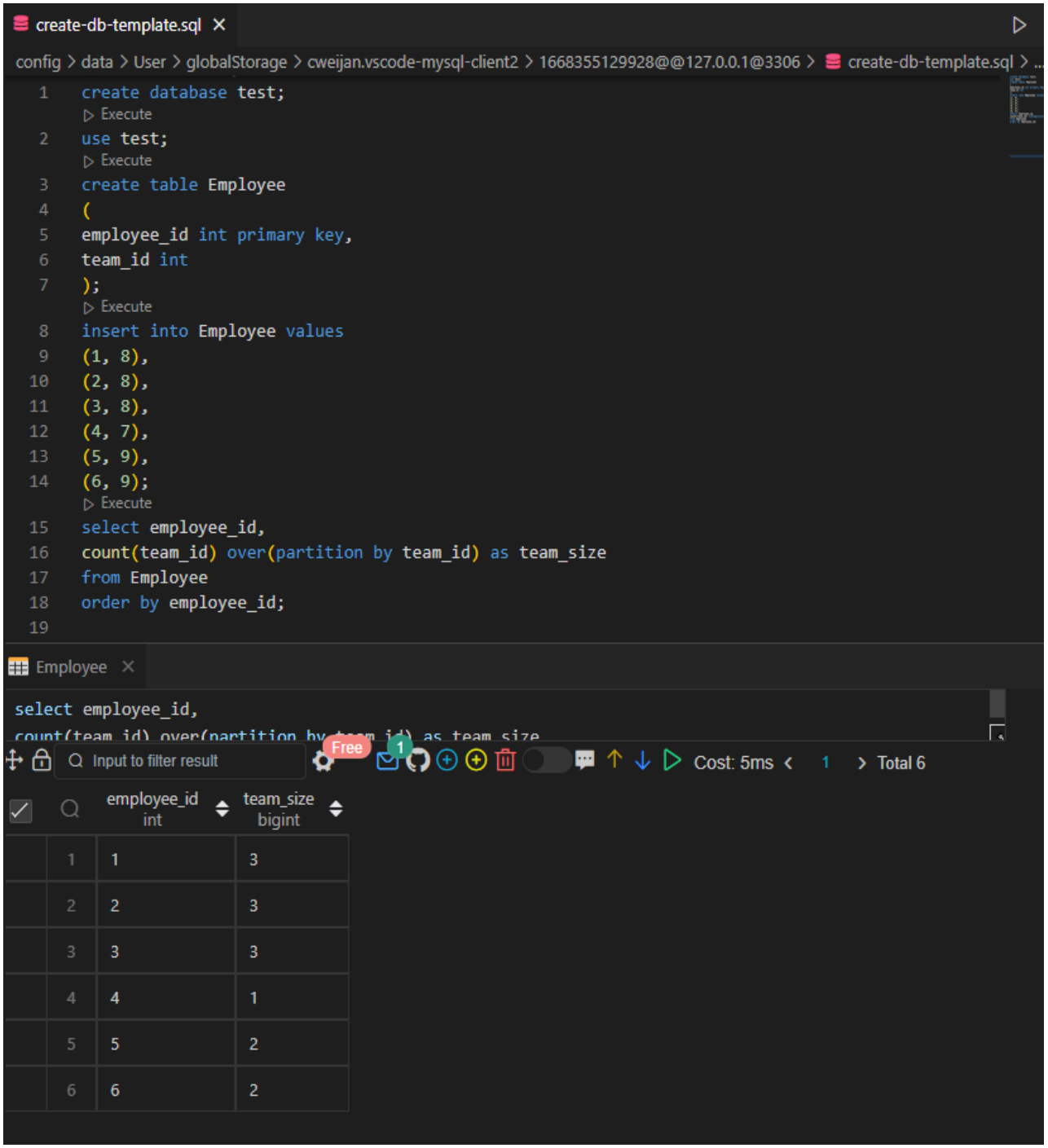
employee_id	team_id
1	8
2	8
3	8
4	7
5	9
6	9

Output:

employee_id	team_size
1	3
2	3
3	3
4	1
5	2
6	2

Explanation:
Employees with Id 1,2,3 are part of a team with team_id = 8.
Employee with Id 4 is part of a team with team_id = 7.
Employees with Id 5,6 are part of a team with team_id = 9.

```
Solution:
count(team_id) over(partition by team_id) as team_size
from Employee
order by employee_id;
```


create-db-template.sql
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-template.sql > ...
1 create database test;
Execute
2 use test;
Execute
3 create table Employee
4 (
5 employee_id int primary key,
6 team_id int
7);
Execute
8 insert into Employee values
9 (1, 8),
10 (2, 8),
11 (3, 8),
12 (4, 7),
13 (5, 9),
14 (6, 9);
Execute
15 select employee_id,
16 count(team_id) over(partition by team_id) as team_size
17 from Employee
18 order by employee_id;
19
Employee
select employee_id,
Input to filter result
Free
Cost: 5ms
1
Total 6
employee_id int
team_size bigint
1 1 3
2 2 3
3 3 3
4 4 1
5 5 2
6 6 2

Q55

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, caller 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

Calls table:

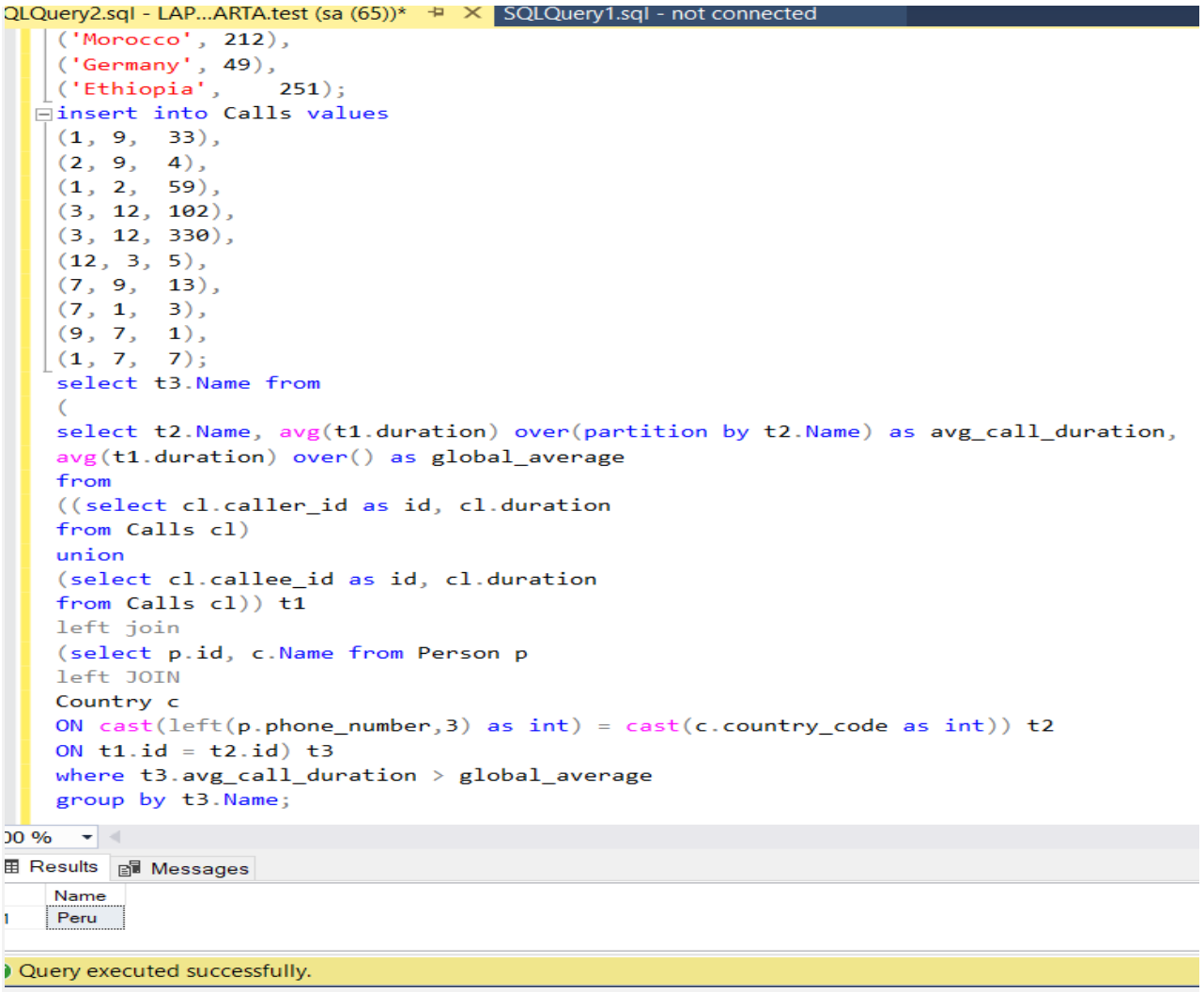
caller_id	callee_id	Duration
1	9	33
2	9	4
1	2	59
3	12	102
3	12	330
12	3	5
7	9	13
7	1	3
9	7	1
1	7	7

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;
```



Q56.

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.

Write an SQL query to report the device that is first logged in for each player.
Return the result table in any order.
The query result format is in the following example.

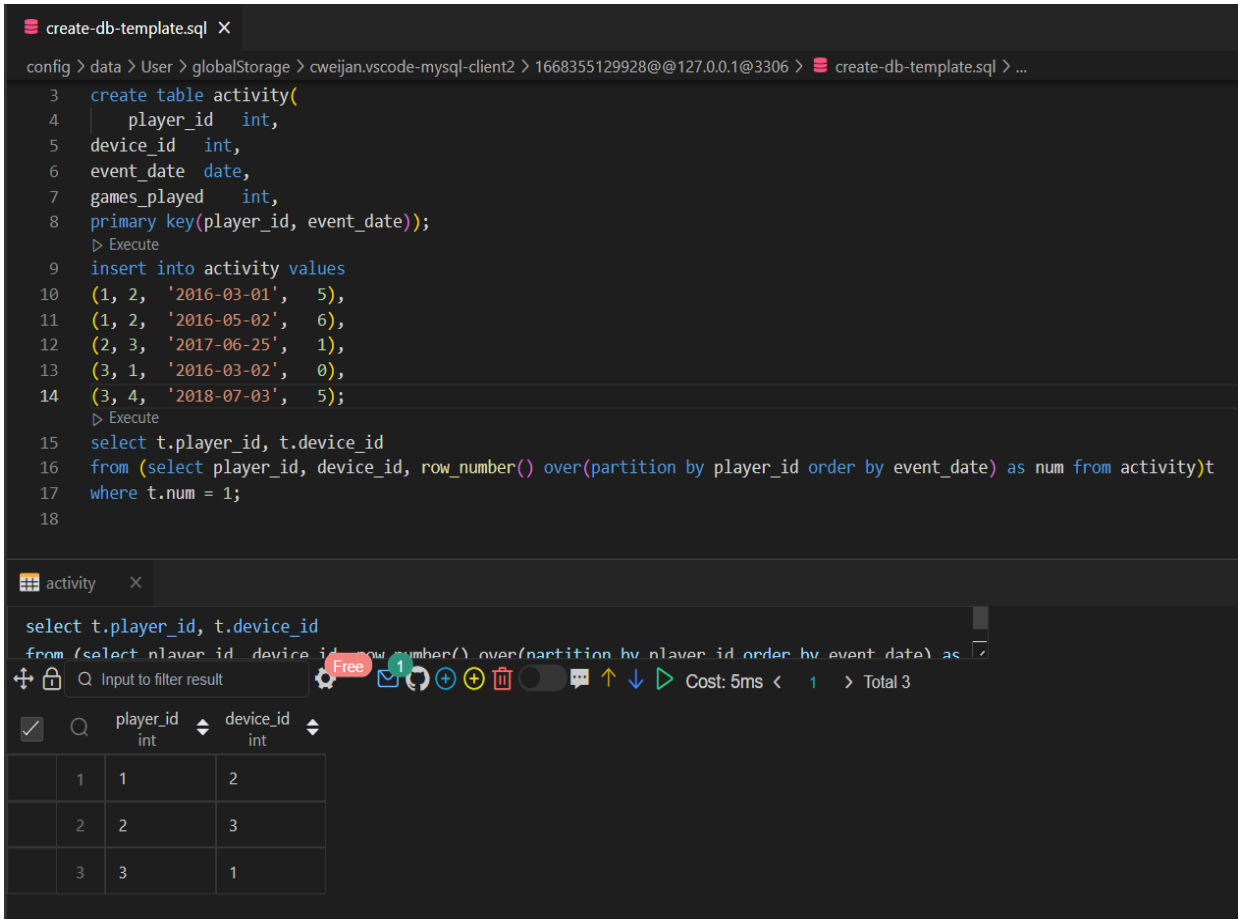
Input:
Activity
table:

player_id	device_id	event_date	games_played
1	2	2016-03-01	5
1	2	2016-05-02	6
2	3	2017-06-25	1
3	1	2016-03-02	0
3	4	2018-07-03	5

Output:

player_id	device_id
1	2
2	3
3	1

```
Solution:
select t.player_id, t.device_id
from (select player_id, device_id, row_number() over(partition by player_id
order by event_date) as num from activity)t
where t.num = 1;
```


create-db-template.sql
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-template.sql > ...
3 create table activity(
4 player_id int,
5 device_id int,
6 event_date date,
7 games_played int,
8 primary key(player_id, event_date));
Execute
9 insert into activity values
10 (1, 2, '2016-03-01', 5),
11 (1, 2, '2016-05-02', 6),
12 (2, 3, '2017-06-25', 1),
13 (3, 1, '2016-03-02', 0),
14 (3, 4, '2018-07-03', 5);
Execute
15 select t.player_id, t.device_id
16 from (select player_id, device_id, row_number() over(partition by player_id order by event_date) as num from activity)t
17 where t.num = 1;
18
activity
select t.player_id, t.device_id
Input to filter result
Free
Cost: 5ms
1
Total 3
player_id int
device_id int
1 1 2
2 2 3
3 3 1

Q57.

Table: Orders

Column Name	Type
order_number	int
customer_number	int

order_number is the primary key for this table.
This table contains information about the order ID and the customer ID.

Write an SQL query to find the customer_number for the customer who has placed the largest number of orders.
The test cases are generated so that exactly one customer will have placed more orders than any other customer.
The query result format is in the following example.

Input:
Orders table:

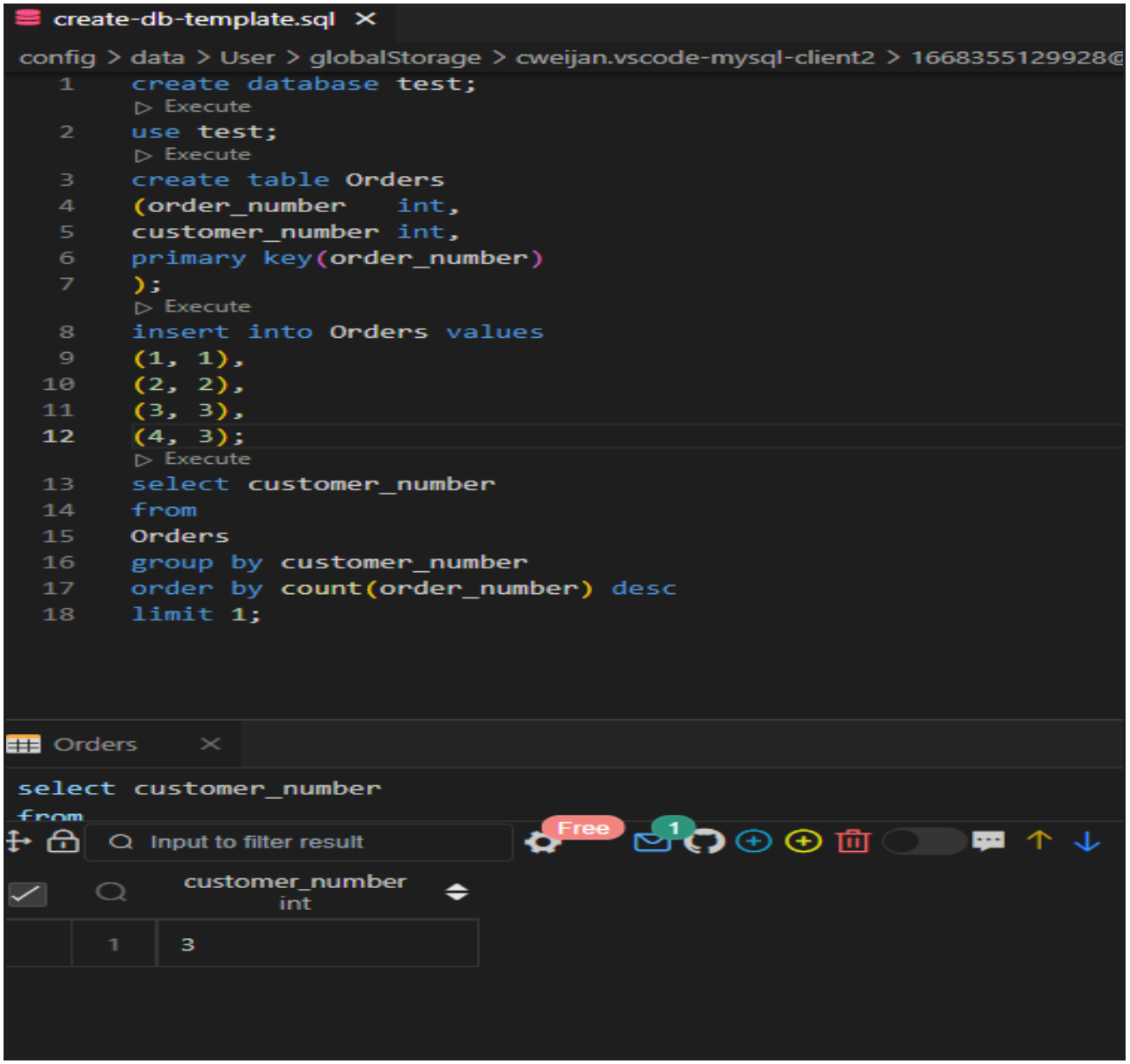
order_number	customer_number
1	1
2	2
3	3
4	3

Output:

customer_number
3

Explanation:
The customer with number 3 has two orders, which is greater than either customer 1 or 2 because each of them only has one order.
So the result is customer_number 3.

```
Solution:
select customer_number
from
Orders
group by customer_number
order by count(order_number) desc
limit 1;
```



Follow up: What if more than one customer has the largest number of orders, can you find all the customer_number in this case?

Ans: To find all such customers, we will use dense_rank() order by count(order_number desc). Now, all those customers who have placed the largest number of orders will get rank 1. Then, we select all those customers who are having rank 1.

Solution:

```
select t.customer_number
from
(select customer_number,
dense_rank() over(order by count(order_number) desc) as r
from
Orders
group by customer_number) t
where t.r = 1;
```

Q58.

Table: Cinema

Column Name	Type
seat_id	Int
Free	Bool

seat_id is an auto-increment primary key column for this table.
Each row of this table indicates whether the ith seat is free or not. 1 means free while 0 means occupied.

Write an SQL query to report all the consecutive available seats in the cinema.
Return the result table ordered by seat_id in ascending order.
The test cases are generated so that more than two seats are consecutively available. The query result format is in the following example.

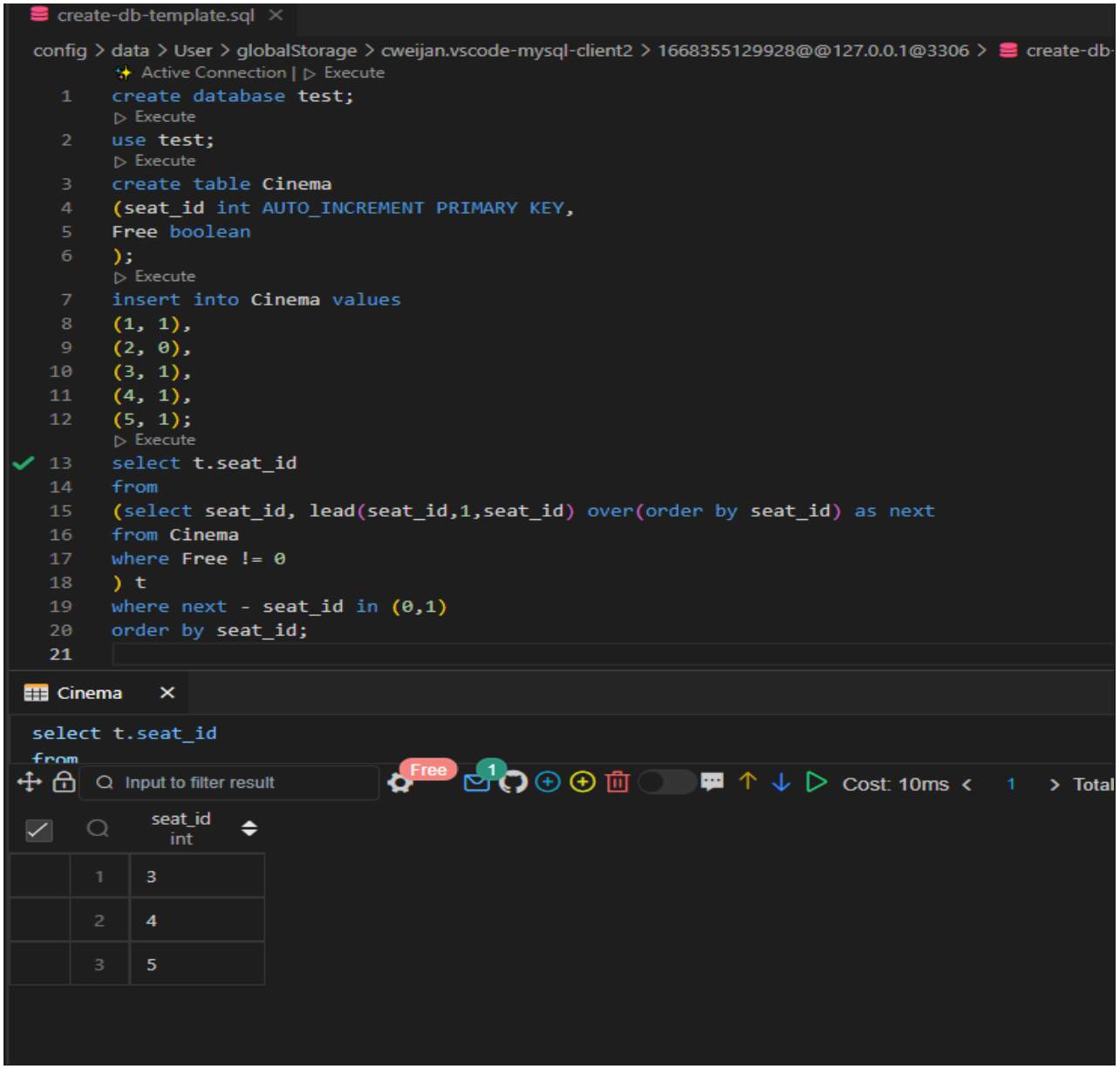
Input:
Cinema table:

seat_id	Free
1	1
2	0
3	1
4	1
5	1

Output:

seat_id
3
4
5

```
Solution:
select t.seat_id
from
(select seat_id, lead(seat_id,1,seat_id) over(order by seat_id) as next
from Cinema
where Free != 0
) t
where next - seat_id in (0,1)
order by seat_id;
```


create-db-template.sql
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-
Active Connection | Execute
create database test;
Execute
use test;
Execute
create table Cinema
(seat_id int AUTO_INCREMENT PRIMARY KEY,
Free boolean
);
Execute
insert into Cinema values
(1, 1),
(2, 0),
(3, 1),
(4, 1),
(5, 1);
Execute
select t.seat_id
from
(select seat_id, lead(seat_id,1,seat_id) over(order by seat_id) as next
from Cinema
where Free != 0
) t
where next - seat_id in (0,1)
order by seat_id;
Cinema
select t.seat_id
from
Input to filter result
Free
Cost: 10ms
Total
seat_id
int
1 3
2 4
3 5

Q59:

Table: SalesPerson

Column Name	Type
sales_id	Int
Name	varchar
Salary	Int
commission_rate	Int
hire_date	date

sales_id is the primary key column for this table.
Each row of this table indicates the name and the ID of a salesperson alongside their salary, commission rate, and hire date.

Table: Company

Column Name	Type
com_id	Int
Name	varchar
City	varchar

com_id is the primary key column for this table.
Each row of this table indicates the name and the ID of a company and the city in which the company is located.

Table: Orders

Column Name	Type
order_id	Int
order_date	Date
com_id	Int
sales_id	Int
Amount	Int

order_id is the primary key column for this table.
com_id is a foreign key to com_id from the Company table.
sales_id is a foreign key to sales_id from the SalesPerson table.
Each row of this table contains information about one order. This includes the ID of the company, the ID of the salesperson, the date of the order, and the amount paid.

Write an SQL query to report the names of all the salespersons who did not have any orders related to the company with the name "RED".
Return the result table in any order.

The query result format is in the following example.

Input:
SalesPerson table:

sales_id	Name	salary	commission_rate	hire_date
1	John	100000	6	4/1/2006
2	Amy	12000	5	5/1/2010
3	Mark	65000	12	12/25/2008
4	Pam	25000	25	1/1/2005
5	Alex	5000	10	2/3/2007

Company table:

com_id	Name	City
1	RED	Boston
2	ORANGE	New York
3	YELLOW	Boston
4	GREEN	Austin

Orders table:

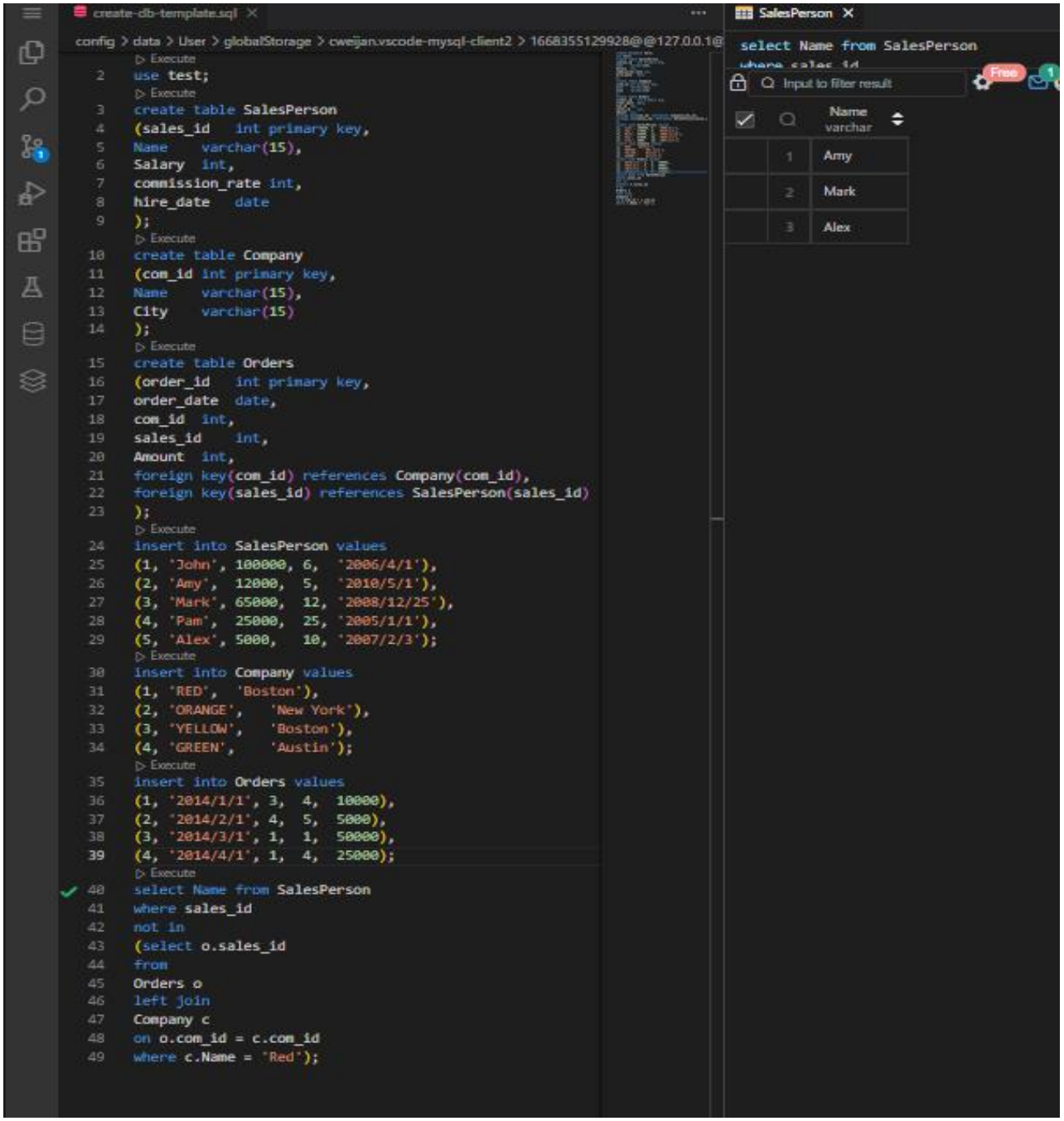
order_id	order_date	com_id	sales_id	amount
1	1/1/2014	3	4	10000
2	2/1/2014	4	5	5000
3	3/1/2014	1	1	50000
4	4/1/2014	1	4	25000

Output:

Name
Amy
Mark
Alex

Explanation:
According to orders 3 and 4 in the Orders table, it is easy to tell that only salesperson John and Pam have sales to company RED, so we report all the other names in the table salesperson.

```
Solution:
select Name from SalesPerson
where sales_id
not in
(select o.sales_id
from
Orders o
left join
Company c
on o.com_id = c.com_id
where c.Name = 'Red');
```



Q60.

Table: Triangle

Column Name	Type
X	Int
Y	Int
Z	Int

(x, y, z) is the primary key column for this table.
Each row of this table contains the lengths of three line segments.

Write an SQL query to report for every three line segments whether they can form a triangle. Return the result table in any order.
The query result format is in the following example.

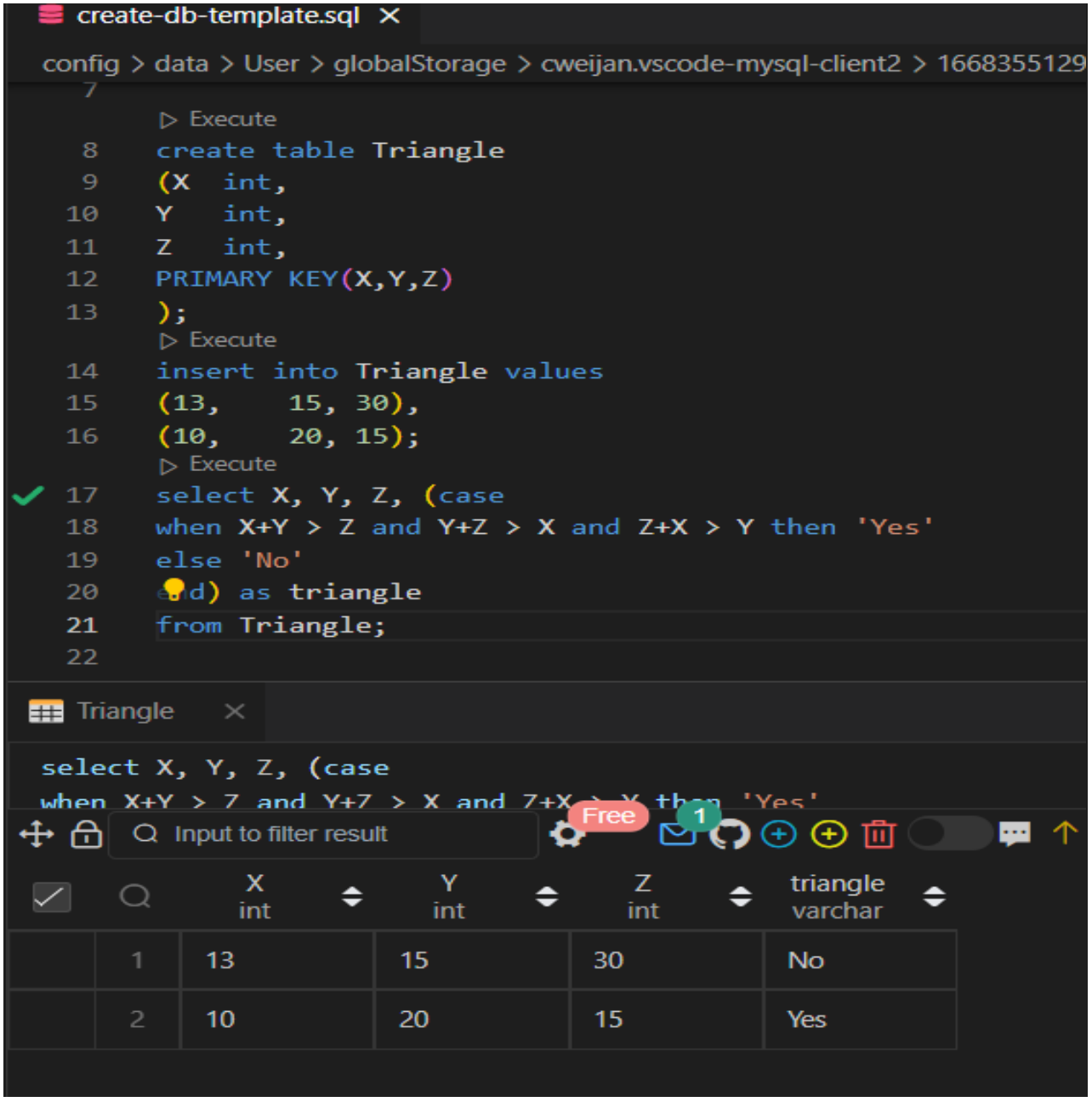
Input: Triangle
table:

X	Y	Z
13	15	30
10	20	15

Output:

X	Y	z	triangle
13	15	30	No
10	20	15	Yes

```
Solution:
select X, Y, Z, (case
when X+Y > Z and Y+Z > X and Z+X > Y then 'Yes'
else 'No'
end) as triangle
from Triangle;
```



Q61.

Table: Point

Column Name	Type
X	Int

x is the primary key column for this table.
Each row of this table indicates the position of a point on the X-axis.

Write an SQL query to report the shortest distance between any two points from the Point table. The query result format is in the following example.

Input: Point table:

x
-1
0
2

Output:

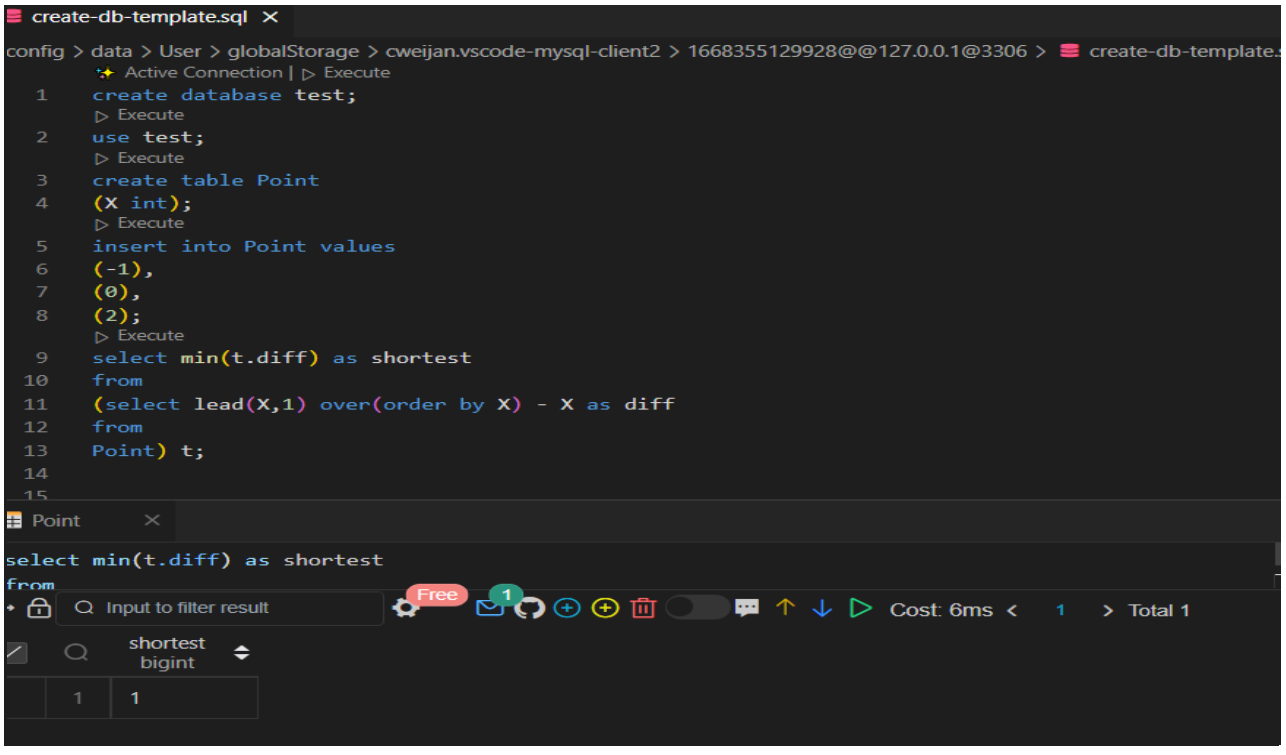
shortest
1

Explanation:
The shortest distance is between points -1 and 0 which is |(-1) - 0| = 1.

Follow up: How could you optimise your query if the Point table is ordered in ascending order?

Ans: If we arrange the points in ascending order, then we only have to check the difference between consecutive numbers, this will decrease the number of comparisons we have to check.

```
Solution:
select min(t.diff) as shortest
from
(select lead(X,1) over(order by X) - X as diff
from
Point) t;
```



Q62.

Table: ActorDirector

Column Name	Type
actor_id	Int
director_id	Int
timestamp	Int

timestamp is the primary key column for this table.

Write a SQL query for a report that provides the pairs (actor_id, director_id) where the actor has cooperated with the director at least three times.
Return the result table in any order.
The query result format is in the following example.

Input:
ActorDirector table:

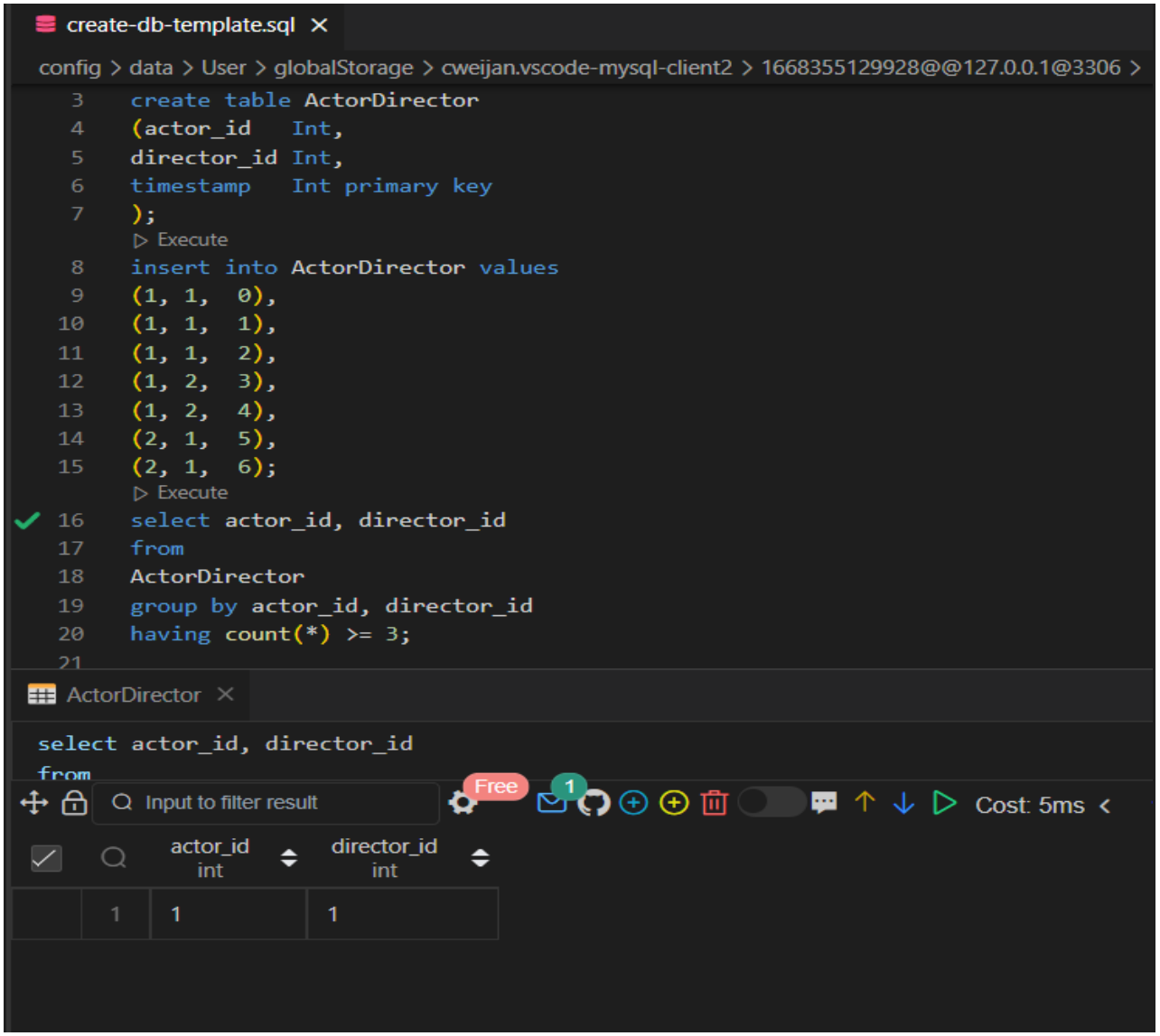
actor_id	director_id	Timestamp
1	1	0
1	1	1
1	1	2
1	2	3
1	2	4
2	1	5
2	1	6

Output:

actor_id	director_id
1	1

Explanation:
The only pair is (1, 1) where they cooperated exactly 3 times.

```
Solution:
select actor_id, director_id
from
ActorDirector
group by actor_id, director_id
having count(*) >= 3;
```


create-db-template.sql
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 >
3 create table ActorDirector
4 (actor_id Int,
5 director_id Int,
6 timestamp Int primary key
7);
Execute
8 insert into ActorDirector values
9 (1, 1, 0),
10 (1, 1, 1),
11 (1, 1, 2),
12 (1, 2, 3),
13 (1, 2, 4),
14 (2, 1, 5),
15 (2, 1, 6);
Execute
16 select actor_id, director_id
17 from
18 ActorDirector
19 group by actor_id, director_id
20 having count(*) >= 3;
21
ActorDirector
select actor_id, director_id
from
Input to filter result
Free
1
Cost: 5ms
actor_id
int
director_id
int
1
1
1

Q63

Table: Sales

Column Name	Type
sale_id	int
product_id	int
year	int
quantity	int
price	int

(sale_id, year) is the primary key of this table.
product_id is a foreign key to the Product table.
Each row of this table shows a sale on the product product_id in a certain year.Note that the price is per unit.

Table: Product

Column Name	Type
product_id	int
product_name	varchar

product_id is the primary key of this table.
Each row of this table indicates the product name of each product.

Write an SQL query that reports the product_name, year, and price for each sale_id in the Sales table.
Return the resulting table in any order.
The query result format is in the following example.

Input:
Sales table:

sale_id	product_id	year	quantity	Price
1	100	2008	10	5000
2	100	2009	12	5000
7	200	2011	15	9000

Product table:

product_id	product_name
100	Nokia
200	Apple
300	Samsung

Output:

product_name	year	Price
Nokia	2008	5000
Nokia	2009	5000
Apple	2011	9000

Explanation:
From sale_id = 1, we can conclude that Nokia was sold for 5000 in the year 2008.
From sale_id = 2, we can conclude that Nokia was sold for 5000 in the year 2009.
From sale_id = 7, we can conclude that Apple was sold for 9000 in the year 2011.

```
Solution:
select p.product_name, s.year,
sum(price) as price
from
Sales s
left join
Product p
on s.product_id = p.product_id
group by p.product_name, s.year;
```

create-db-template.sql

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

```
use test;
▷ Execute
create table Product
(product_id int primary key,
product_name varchar(10)
);
▷ Execute
create table Sales
(
sale_id int,
product_id  int,
year    int,
quantity    int,
price   int,
primary key(sale_id, year),
foreign key(product_id) references Product(product_id)
);
▷ Execute
insert into Product values
(100,   'Nokia'),
(200,   'Apple'),
(300,   'Samsung');
▷ Execute
insert into Sales values
(1, 100,    2008,   10, 5000),
(2, 100,    2009,   12, 5000),
(7, 200,    2011,   15, 9000);
▷ Execute
select p.product_name, s.year,
sum(price) as price
from
Sales s
left join
Product p
on s.product_id = p.product_id
group by p.product_name, s.year;
```

ActorDirector

```
select p.product_name, s.year, sum(price) as price
from
```

Input to filter result | Free | Cost: 4ms | 1 | Total 3

	product_name varchar	year int	price newdecimal
1	Nokia	2008	5000
2	Nokia	2009	5000
3	Apple	2011	9000

Q64.

Table: Project

Column Name	Type
project_id	int
employee_id	int

(project_id, employee_id) is the primary key of this table.
employee_id is a foreign key to the Employee table.
Each row of this table indicates that the employee with employee_id is working on the project with project_id.

Table: Employee

Column Name	Type
employee_id	int
name	varchar
experience_years	int

employee_id is the primary key of this table.
Each row of this table contains information about one employee.

Write an SQL query that reports the average experience years of all the employees for each project, rounded to 2 digits.
Return the result table in any order.
The query result format is in the following example.

Input: Project
table:

project_id	employee_id
1	1
1	2
1	3
2	1
2	4

Employee table:

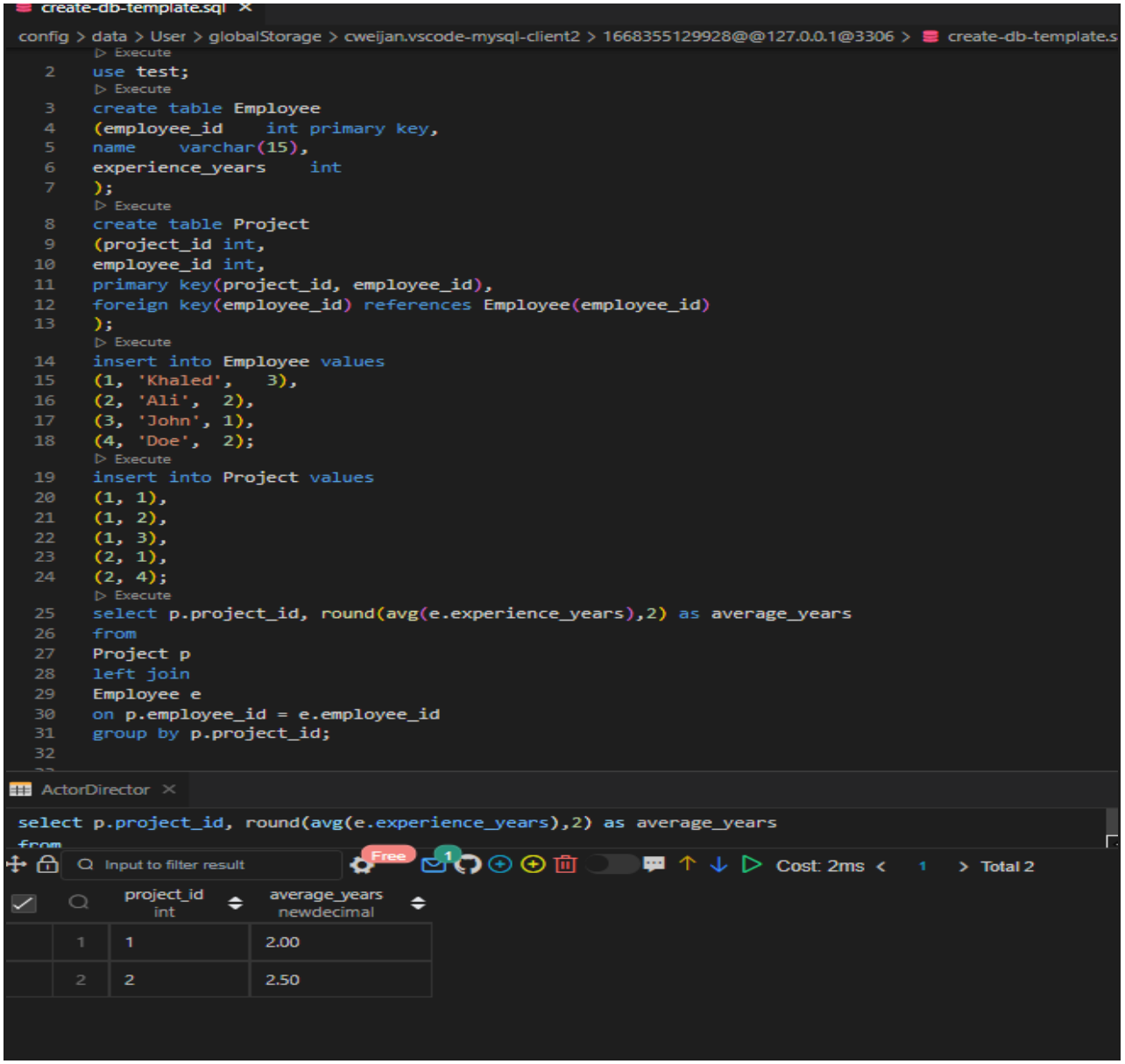
employee_id	Name	experience_years
1	Khaled	3
2	Ali	2
3	John	1
4	Doe	2

Output:

project_id	average_years
1	2
2	2.5

Explanation:
The average experience years for the first project is (3 + 2 + 1) / 3 = 2.00 and for the second project is (3 + 2) / 2 = 2.50

```
Solution:
select p.project_id, round(avg(e.experience_years),2) as average_years
from
Project p
left join
Employee e
on p.employee_id = e.employee_id
group by p.project_id;
```


create-db-template.sql
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-template.s
▷ Execute
use test;
▷ Execute
create table Employee
(employee_id int primary key,
name varchar(15),
experience_years int
);
▷ Execute
create table Project
(project_id int,
employee_id int,
primary key(project_id, employee_id),
foreign key(employee_id) references Employee(employee_id)
);
▷ Execute
insert into Employee values
(1, 'Khaled', 3),
(2, 'Ali', 2),
(3, 'John', 1),
(4, 'Doe', 2);
▷ Execute
insert into Project values
(1, 1),
(1, 2),
(1, 3),
(2, 1),
(2, 4);
▷ Execute
select p.project_id, round(avg(e.experience_years),2) as average_years
from
Project p
left join
Employee e
on p.employee_id = e.employee_id
group by p.project_id;
ActorDirector
select p.project_id, round(avg(e.experience_years),2) as average_years
from
Input to filter result
Free
Cost: 2ms
Total 2
project_id
int
average_years
newdecimal
1 1 2.00
2 2 2.50

Q65.

Table: Product

Column Name	Type
product_id	int
product_name	varchar
unit_price	int

product_id is the primary key of this table.
Each row of this table indicates the name and the price of each product.

Table: Sales

Column Name	Type
seller_id	int
product_id	int
buyer_id	int
sale_date	date
quantity	int
price	int

This table has no primary key, it can have repeated rows.product_id
is a foreign key to the Product table.
Each row of this table contains some information about one sale.

Write an SQL query that reports the best seller by total sales price, If there is a tie, report them all.
Return the result table in any order.
The query result format is in the following example.

Input: Product
table:

product_id	product_name	unit_price
1	S8	1000
2	G4	800
3	iPhone	1400

Sales table:

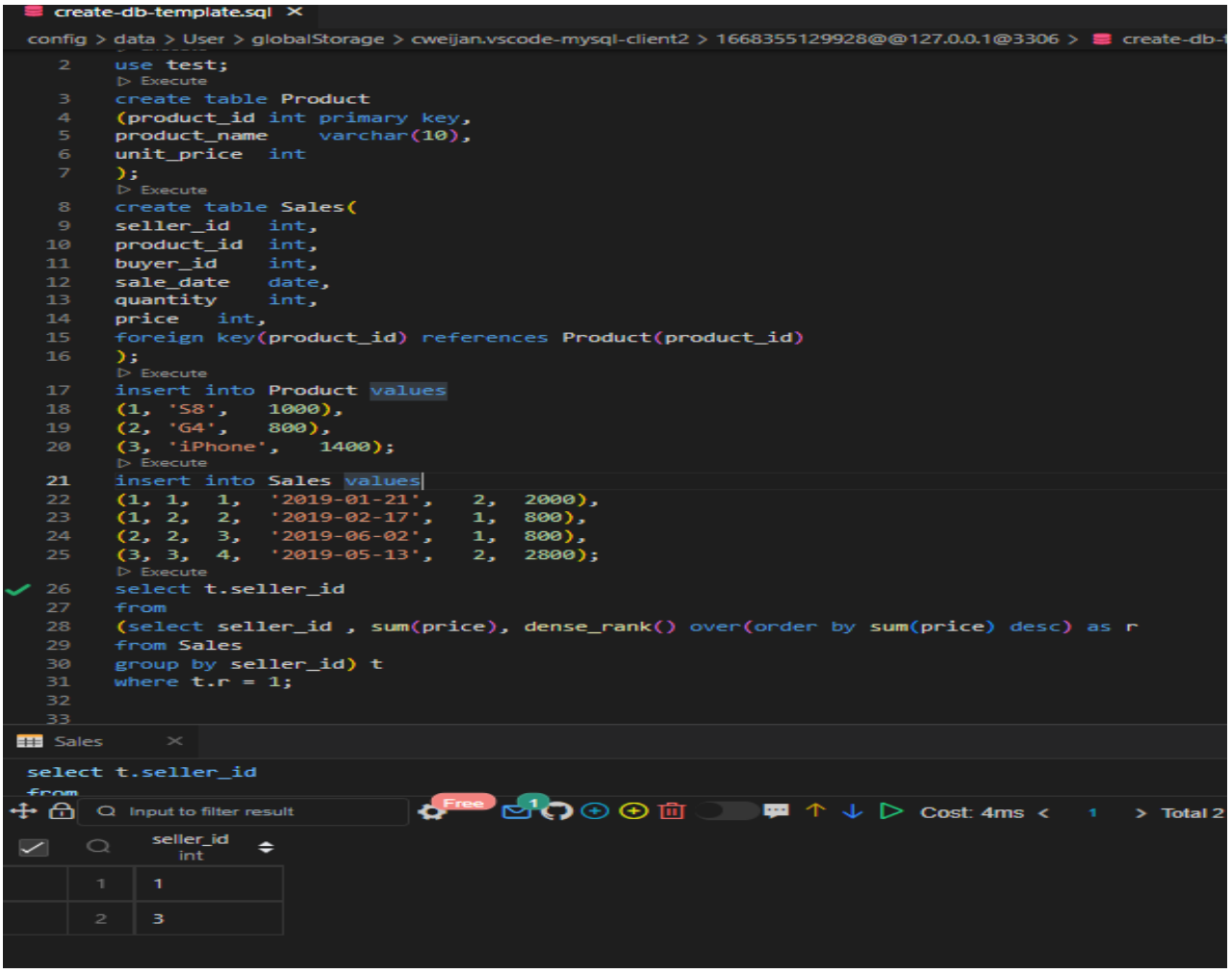
seller_id	product_id	buyer_id	sale_date	quantity	price
1	1	1	2019-01-21	2	2000
1	2	2	2019-02-17	1	800
2	2	3	2019-06-02	1	800
3	3	4	2019-05-13	2	2800

Output:

seller_id
1
3

Explanation: Both sellers with id 1 and 3 sold products with the most total price of 2800.

```
Solution:
select t.seller_id
from
(select seller_id , sum(price),
dense_rank() over(order by sum(price) desc) as r
from Sales
group by seller_id) t
where t.r = 1;
```



Q66.

Table: Product

Column Name	Type
product_id	int
product_name	varchar
unit_price	int

product_id is the primary key of this table.
Each row of this table indicates the name and the price of each product.

Table: Sales

Column Name	Type
seller_id	int
product_id	int
buyer_id	int
sale_date	date
quantity	int
price	int

This table has no primary key, it can have repeated rows.product_id
is a foreign key to the Product table.
Each row of this table contains some information about one sale.

Write an SQL query that reports the buyers who have bought S8 but not iPhone. Note that S8 and iPhone are products present in the Product table.
Return the result table in any order.
The query result format is in the following example.

Input: Product
table:

product_id	product_name	unit_price
1	S8	1000
2	G4	800
3	iPhone	1400

Sales table:

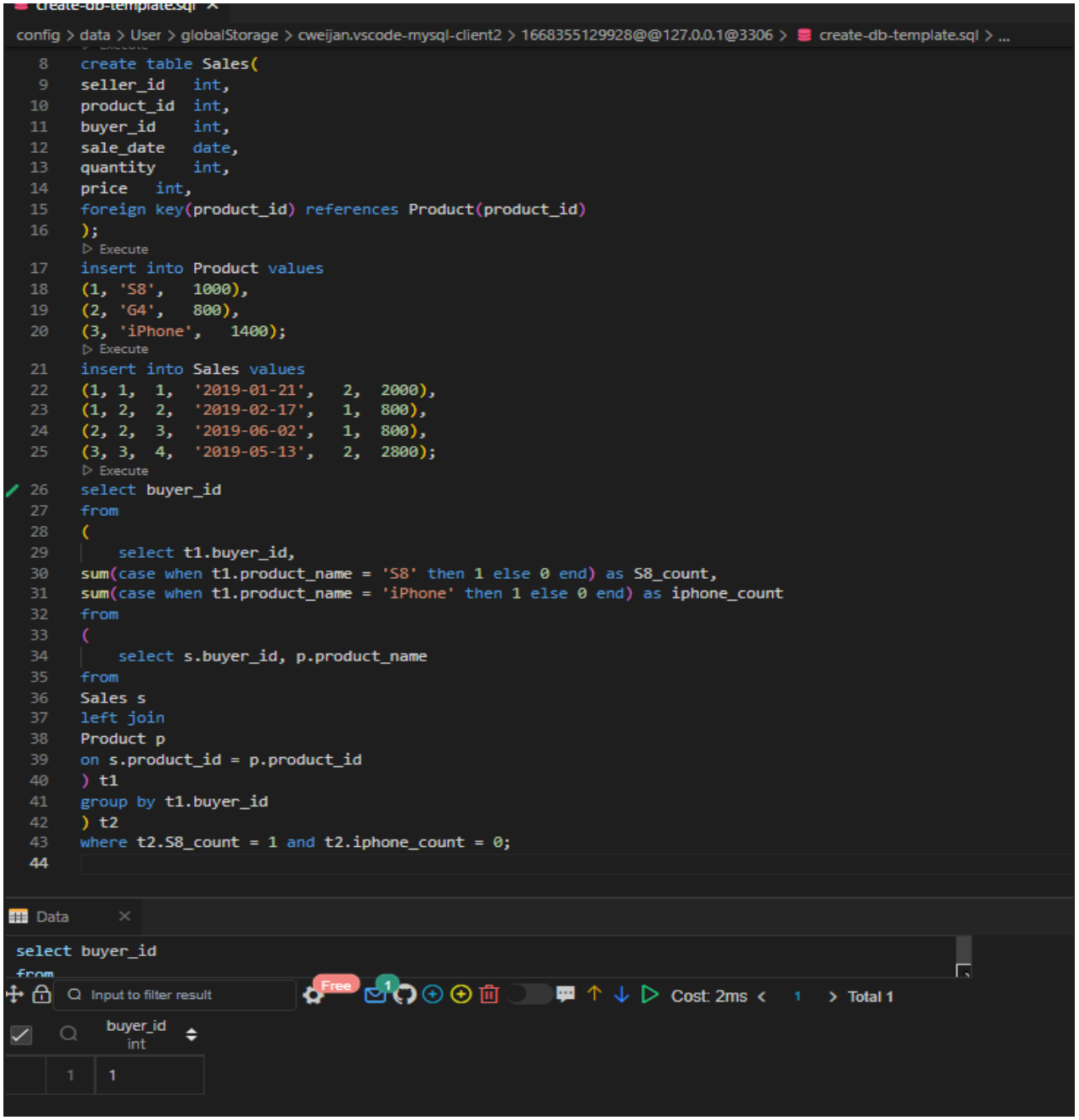
seller_id	product_id	buyer_id	sale_date	quantity	price
1	1	1	2019-01-21	2	2000
1	2	2	2019-02-17	1	800
2	2	3	2019-06-02	1	800
3	3	4	2019-05-13	2	2800

Output:

buyer_id
1

Explanation:
The buyer with id 1 bought an S8 but did not buy an iPhone. The buyer with id 3 bought both.

```
Soltion:
select buyer_id
from
(
    select t1.buyer_id,
sum(case when t1.product_name = 'S8' then 1 else 0 end) as S8_count,
sum(case when t1.product_name = 'iPhone' then 1 else 0 end) as iphone_count
from
(
    select s.buyer_id, p.product_name
from
Sales s
left join
Product p
on s.product_id = p.product_id
) t1
group by t1.buyer_id
) t2
where t2.S8_count = 1 and t2.iphone_count = 0;
```


create-db-template.sql
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-template.sql > ...
8 create table Sales(
9 seller_id int,
10 product_id int,
11 buyer_id int,
12 sale_date date,
13 quantity int,
14 price int,
15 foreign key(product_id) references Product(product_id)
16);
Execute
17 insert into Product values
18 (1, 'S8', 1000),
19 (2, 'G4', 800),
20 (3, 'iPhone', 1400);
Execute
21 insert into Sales values
22 (1, 1, 1, '2019-01-21', 2, 2000),
23 (1, 2, 2, '2019-02-17', 1, 800),
24 (2, 2, 3, '2019-06-02', 1, 800),
25 (3, 3, 4, '2019-05-13', 2, 2800);
Execute
26 select buyer_id
27 from
28 (
29 select t1.buyer_id,
30 sum(case when t1.product_name = 'S8' then 1 else 0 end) as S8_count,
31 sum(case when t1.product_name = 'iPhone' then 1 else 0 end) as iphone_count
32 from
33 (
34 select s.buyer_id, p.product_name
35 from
36 Sales s
37 left join
38 Product p
39 on s.product_id = p.product_id
40) t1
41 group by t1.buyer_id
42) t2
43 where t2.S8_count = 1 and t2.iphone_count = 0;
44
Data
select buyer_id
from
Input to filter result
Free
Cost: 2ms
1
Total 1
buyer_id
int
1
1

Q67.

Table: Customer

Column Name	Type
customer_id	Int
Name	Varchar
visited_on	Date
Amount	Int

(customer_id, visited_on) is the primary key for this table.
This table contains data about customer transactions in a restaurant.
visited_on is the date on which the customer with ID (customer_id) has visited the restaurant.
amount is the total paid by a customer.

You are the restaurant owner and you want to analyse a possible expansion (there will be at least one customer every day).
Write an SQL query to compute the moving average of how much the customer paid in a seven days window (i.e., current day + 6 days before). average_amount should be rounded to two decimal places.
Return result table ordered by visited_on in ascending order.
The query result format is in the following example.

Input:
Customer table:

customer_id	Name	visited_on	amount
1	Jhon	2019-01-01	100
2	Daniel	2019-01-02	110
3	Jade	2019-01-03	120
4	Khaled	2019-01-04	130
5	Winston	2019-01-05	110
6	Elvis	2019-01-06	140
7	Anna	2019-01-07	150
8	Maria	2019-01-08	80
9	Jaze	2019-01-09	110
1	Jhon	2019-01-10	130
3	Jade	2019-01-10	150

Output:

visited_on	amount	average_amount
2019-01-07	860	122.86

2019-01-08	840	120
2019-01-09	840	120
2019-01-10	1000	142.86

Explanation:
1st moving average from 2019-01-01 to 2019-01-07 has an average_amount of (100 + 110 + 120 + 130 + 110 + 140 + 150)/7 = 122.86
2nd moving average from 2019-01-02 to 2019-01-08 has an average_amount of (110 + 120 + 130 + 110 + 140 + 150 + 80)/7 = 120
3rd moving average from 2019-01-03 to 2019-01-09 has an average_amount of (120 + 130 + 110 + 140 + 150 + 80 + 110)/7 = 120
4th moving average from 2019-01-04 to 2019-01-10 has an average_amount of (130 + 110 + 140 + 150 + 80 + 110 + 130 + 150)/7 = 142.86

```
Solution:
select t2.visited_on, t2.amount, t2.average_amount
from
(select t1.visited_on, t1.prev_date_interval_6,
round(sum(amount) over(order by visited_on range between interval '6' day
preceding and current row),2) as amount,
round(avg(amount) over(order by visited_on range between interval '6' day
preceding and current row),2) as average_amount
from
(select visited_on, sum(amount) as amount,
lag(visited_on,6) over(order by visited_on) as prev_date_interval_6
from Customer
group by visited_on
order by visited_on) t1
) t2
where prev_date_interval_6 is not null;
```

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 Customer
(customer_id    int,
name    varchar(15),
visited_on  date,
amount  int,
primary key(customer_id, visited_on)
);
```

▷ Execute

```sql
insert into Customer values
(1, 'Jhon', '2019-01-01',   100),
(2, 'Daniel',   '2019-01-02',   110),
(3, 'Jade', '2019-01-03',   120),
(4, 'Khaled',   '2019-01-04',   130),
(5, 'Winston',  '2019-01-05',   110),
(6, 'Elvis',    '2019-01-06',   140),
(7, 'Anna', '2019-01-07',   150),
(8, 'Maria',    '2019-01-08',   80),
(9, 'Jaze', '2019-01-09',   110),
(1, 'Jhon'  ,'2019-01-10',  130),
(3, 'Jade', '2019-01-10',   150);
```

▷ Execute

```sql
select t2.visited_on, t2.amount, t2.average_amount
from
(select t1.visited_on, t1.prev_date_interval_6,
round(sum(amount) over(order by visited_on range between interval '6' day preceding and current row),2) as amount,
round(avg(amount) over(order by visited_on range between interval '6' day preceding and current row),2) as average_amount
from
(select visited_on, sum(amount) as amount,
lag(visited_on,6) over(order by visited_on) as prev_date_interval_6
from Customer
group by visited_on
order by visited_on) t1
) t2
where prev_date_interval_6 is not null;
```

Customer

```sql
select t2.visited_on, t2.amount, t2.average_amount
from
```

Input to filter result | Free | Cost: 7ms | 1 | Total 4

		visited_on date	amount newdecimal	average_amount newdecimal
	1	2019-01-07	860	122.86
	2	2019-01-08	840	120.00
	3	2019-01-09	840	120.00
	4	2019-01-10	1000	142.86

Q68.

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:

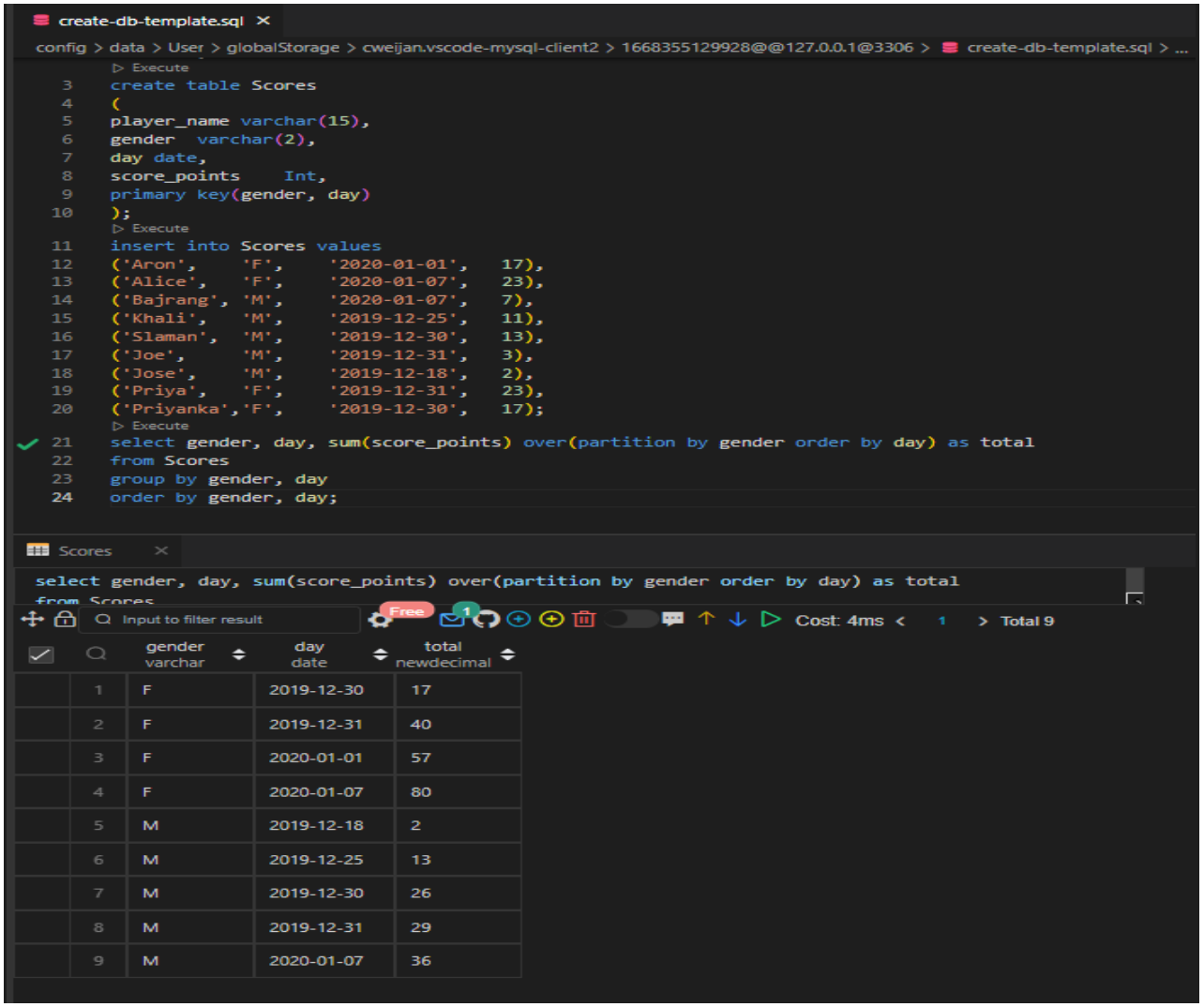
gender	day	Total
F	2019-12-30	17
F	2019-12-31	40
F	2020-01-01	57
F	2020-01-07	80
M	2019-12-18	2
M	2019-12-25	13

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;
```



Q69.

Table: Logs

Column Name	Type
log_id	int

log_id is the primary key for this table.
Each row of this table contains the ID in a log Table.

Write an SQL query to find the start and end number of continuous ranges in the table Logs.
Return the result table ordered by start_id.
The query result format is in the following example.

Input: Logs table:

log_id
1
2
3
7
8
10

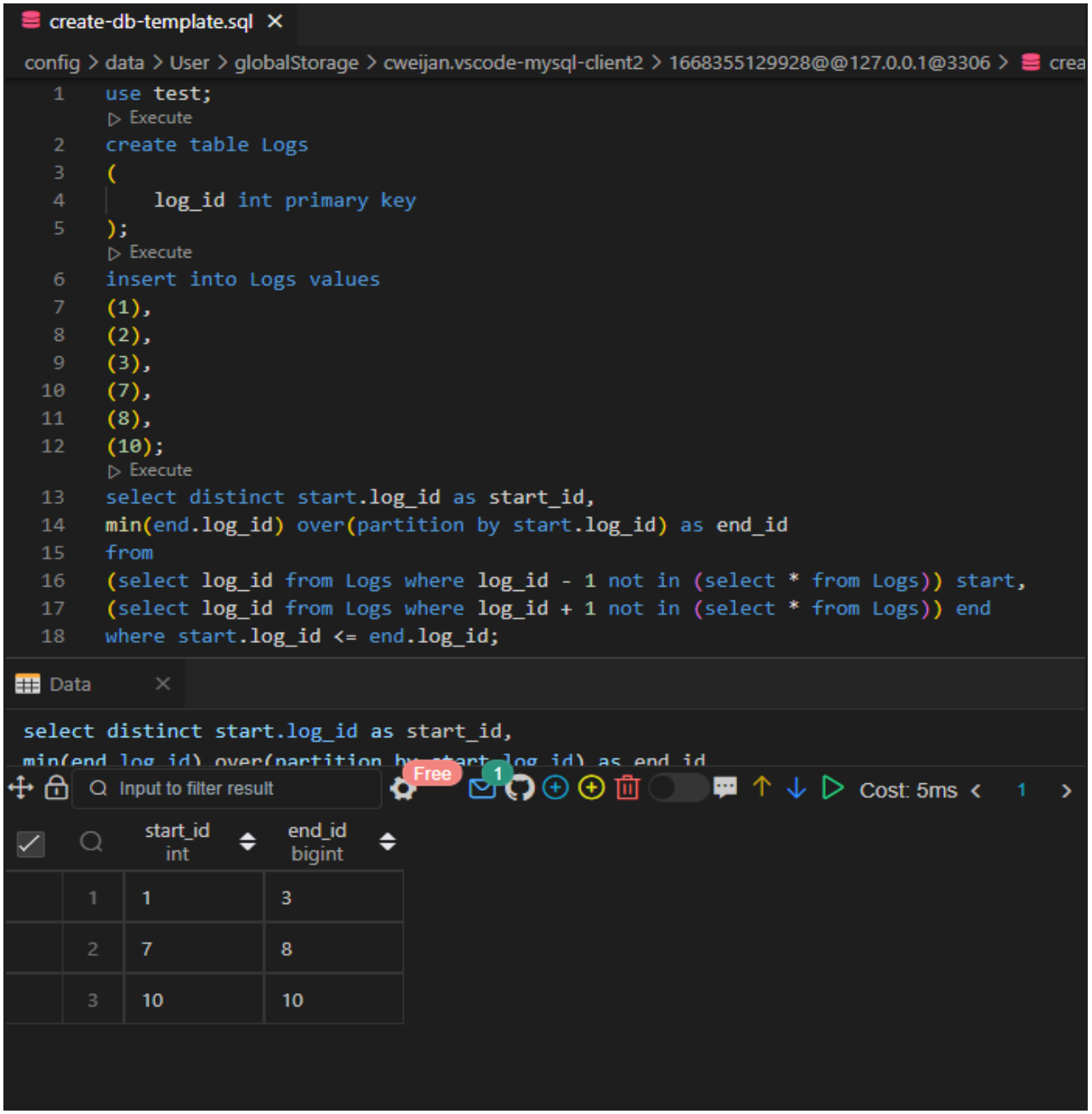
Output:

start_id	end_id
1	3
7	8
10	10

Explanation:
The result table should contain all ranges in table Logs.
From 1 to 3 is contained in the table.
From 4 to 6 is missing in the table
From 7 to 8 is contained in the table.
Number 9 is missing from the table.
Number 10 is contained in the table.

Solution:

```
select distinct start.log_id as start_id,
min(end.log_id) over(partition by start.log_id) as end_id
from
(select log_id from Logs where log_id - 1 not in (select * from Logs)) start,
(select log_id from Logs where log_id + 1 not in (select * from Logs)) end
where start.log_id <= end.log_id;
```



Q70.

Table: Students

Column Name	Type
student_id	Int
student_name	Varchar

student_id is the primary key for this table.
Each row of this table contains the ID and the name of one student in the school.

Table: Subjects

Column Name	Type
subject_name	Varchar

subject_name is the primary key for this table.
Each row of this table contains the name of one subject in the school.

Table: Examinations

Column Name	Type
student_id	Int
subject_name	Varchar

There is no primary key for this table. It may contain duplicates.
Each student from the Students table takes every course from the Subjects table.
Each row of this table indicates that a student with ID student_id attended the exam of subject_name.

Write an SQL query to find the number of times each student attended each exam.
Return the result table ordered by student_id and subject_name.

The query result format is in the following example.

Input: Students table:

student_id	student_name
1	Alice
2	Bob
13	John
6	Alex

Subjects table:

subject_name
Math
Physics
Programming

Examinations table:

student_id	subject_name
1	Math
1	Physics
1	Programming
2	Programming
1	Physics
1	Math
13	Math
13	Programming
13	Physics
2	Math
1	Math

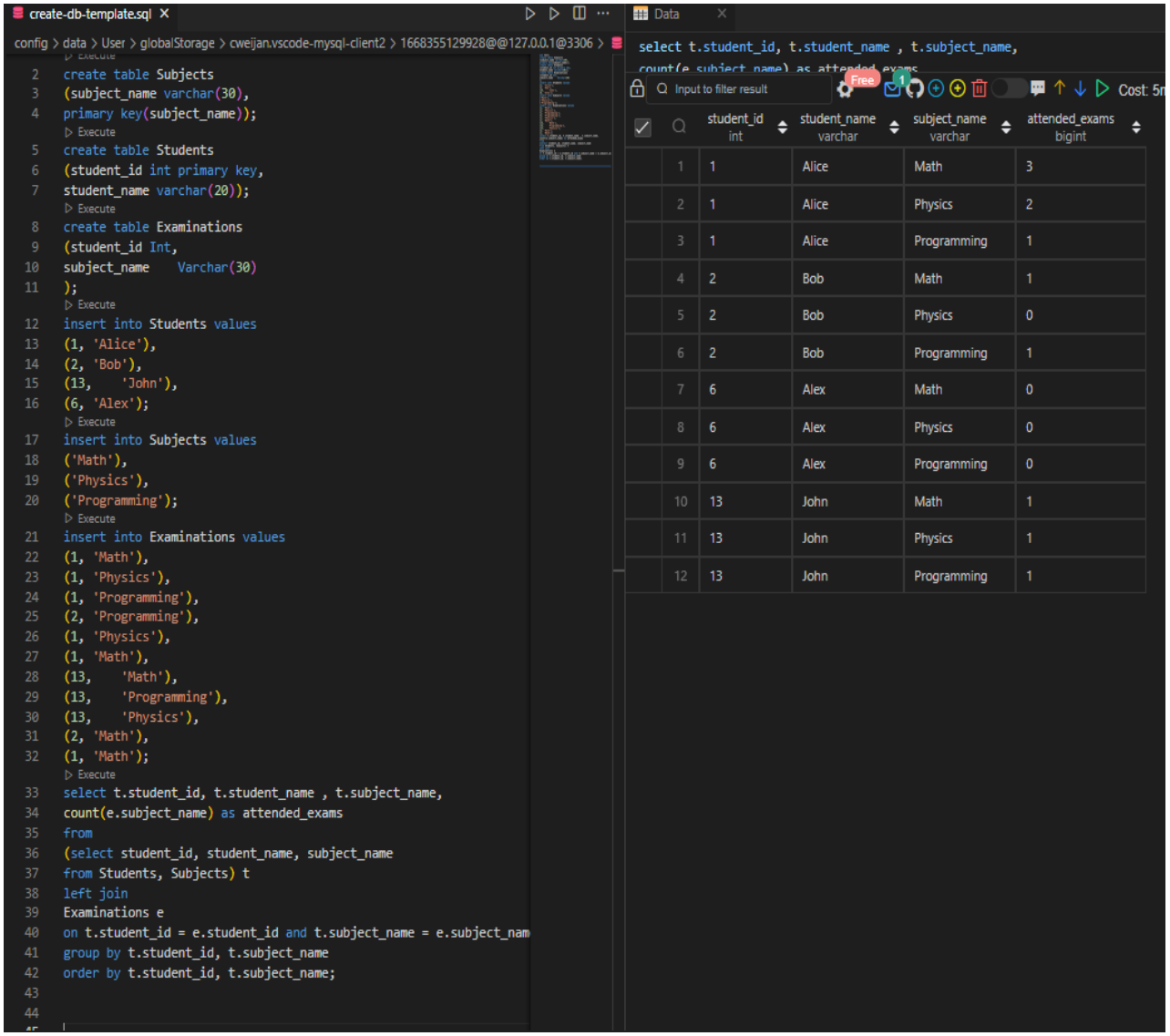
Output:

student_id	student_name	subject_name	attended_exams
1	Alice	Math	3
1	Alice	Physics	2
1	Alice	Programming	1
2	Bob	Math	1
2	Bob	Physics	0

2	Bob	Programming	1
6	Alex	Math	0
6	Alex	Physics	0
6	Alex	Programming	0
13	John	Math	1
13	John	Physics	1
13	John	Programming	1

Explanation:
The result table should contain all students and all subjects.
Alice attended the Math exam 3 times, the Physics exam 2 times, and the Programming exam 1 time.
Bob attended the Math exam 1 time, the Programming exam 1 time, and did not attend the Physics exam.
Alex did not attend any exams.
John attended the Math exam 1 time, the Physics exam 1 time, and the Programming exam 1 time.

```
Solution:
select t.student_id, t.student_name , t.subject_name,
count(e.subject_name) as attended_exams
from
(select student_id, student_name, subject_name
from Students, Subjects) t
left join
Examinations e
on t.student_id = e.student_id and t.subject_name = e.subject_name
group by t.student_id, t.subject_name
order by t.student_id, t.subject_name;
```


create-db-template.sql
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 >
Execute
create table Subjects
(subject_name varchar(30),
primary key(subject_name));
Execute
create table Students
(student_id int primary key,
student_name varchar(20));
Execute
create table Examinations
(student_id Int,
subject_name Varchar(30)
);
Execute
insert into Students values
(1, 'Alice'),
(2, 'Bob'),
(13, 'John'),
(6, 'Alex');
Execute
insert into Subjects values
('Math'),
('Physics'),
('Programming');
Execute
insert into Examinations values
(1, 'Math'),
(1, 'Physics'),
(1, 'Programming'),
(2, 'Programming'),
(1, 'Physics'),
(1, 'Math'),
(13, 'Math'),
(13, 'Programming'),
(13, 'Physics'),
(2, 'Math'),
(1, 'Math');
Execute
select t.student_id, t.student_name , t.subject_name,
count(e.subject_name) as attended_exams
from
(select student_id, student_name, subject_name
from Students, Subjects) t
left join
Examinations e
on t.student_id = e.student_id and t.subject_name = e.subject_nam
group by t.student_id, t.subject_name
order by t.student_id, t.subject_name;
Data
select t.student_id, t.student_name , t.subject_name,
Input to filter result
Free
Cost: 5r
student_id int
student_name varchar
subject_name varchar
attended_exams bigint
1 1 Alice Math 3
2 1 Alice Physics 2
3 1 Alice Programming 1
4 2 Bob Math 1
5 2 Bob Physics 0
6 2 Bob Programming 1
7 6 Alex Math 0
8 6 Alex Physics 0
9 6 Alex Programming 0
10 13 John Math 1
11 13 John Physics 1
12 13 John Programming 1

Q71.

Table: Employees

Column Name	Type
employee_id	Int
employee_name	Varchar
manager_id	int

employee_id is the primary key for this table.
Each row of this table indicates that the employee with ID employee_id and name employee_name reports his work to his/her direct manager with manager_id
The head of the company is the employee with employee_id = 1.

Write an SQL query to find employee_id of all employees that directly or indirectly report their work to the head of the company.
The indirect relation between managers will not exceed three managers as the company is small.
Return the result table in any order.
The query result format is in the following example.

Input:
Employees table:

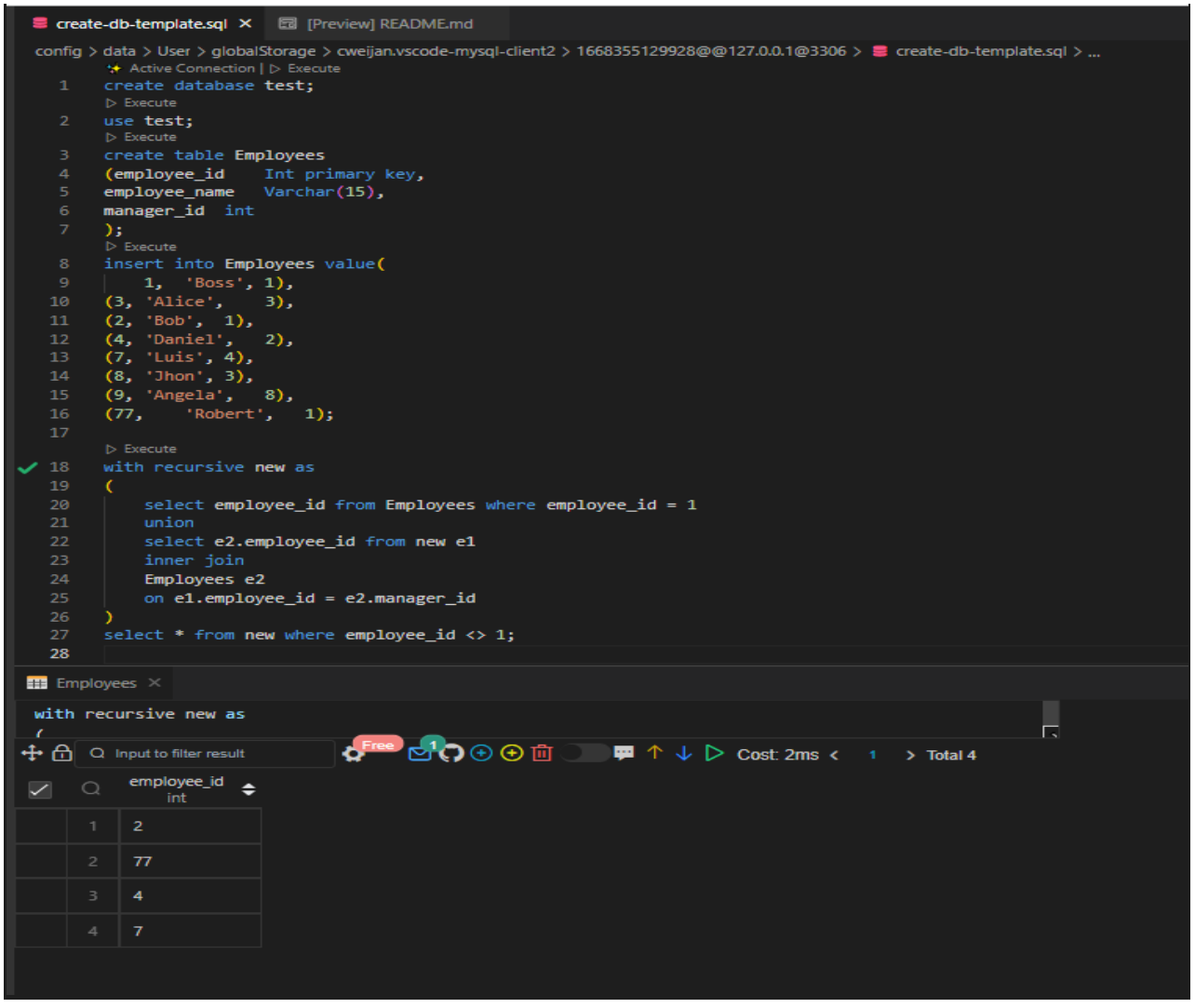
employee_id	employee_name	manager_id
1	Boss	1
3	Alice	3
2	Bob	1
4	Daniel	2
7	Luis	4
8	Jhon	3
9	Angela	8
77	Robert	1

Output:

employee_id
2
77
4
7

Explanation:
The head of the company is the employee with employee_id 1.
The employees with employee_id 2 and 77 report their work directly to the head of the company.
The employee with employee_id 4 reports their work indirectly to the head of the company 4 --> 2 --> 1. The employee with employee_id 7 reports their work indirectly to the head of the company 7 --> 4 --> 2 --> 1.
The employees with employee_id 3, 8, and 9 do not report their work to the head of the company directly or indirectly.

```
Solution:
with recursive new as
(
    select employee_id from Employees where employee_id = 1
    union
    select e2.employee_id from new e1
    inner join
    Employees e2
    on e1.employee_id = e2.manager_id
)
select * from new where employee_id <> 1;
```



Q72.

Table: Transactions

Column Name	Type
Id	Int
Country	Varchar
State	Enum
Amount	Int
trans_date	Date

id is the primary key of this table.
The table has information about incoming transactions.
The state column is an enum of type ["approved", "declined"].

Write an SQL query to find for each month and country, the number of transactions and their total amount, the number of approved transactions and their total amount.
Return the result table in any order.
The query result format is in the following example.

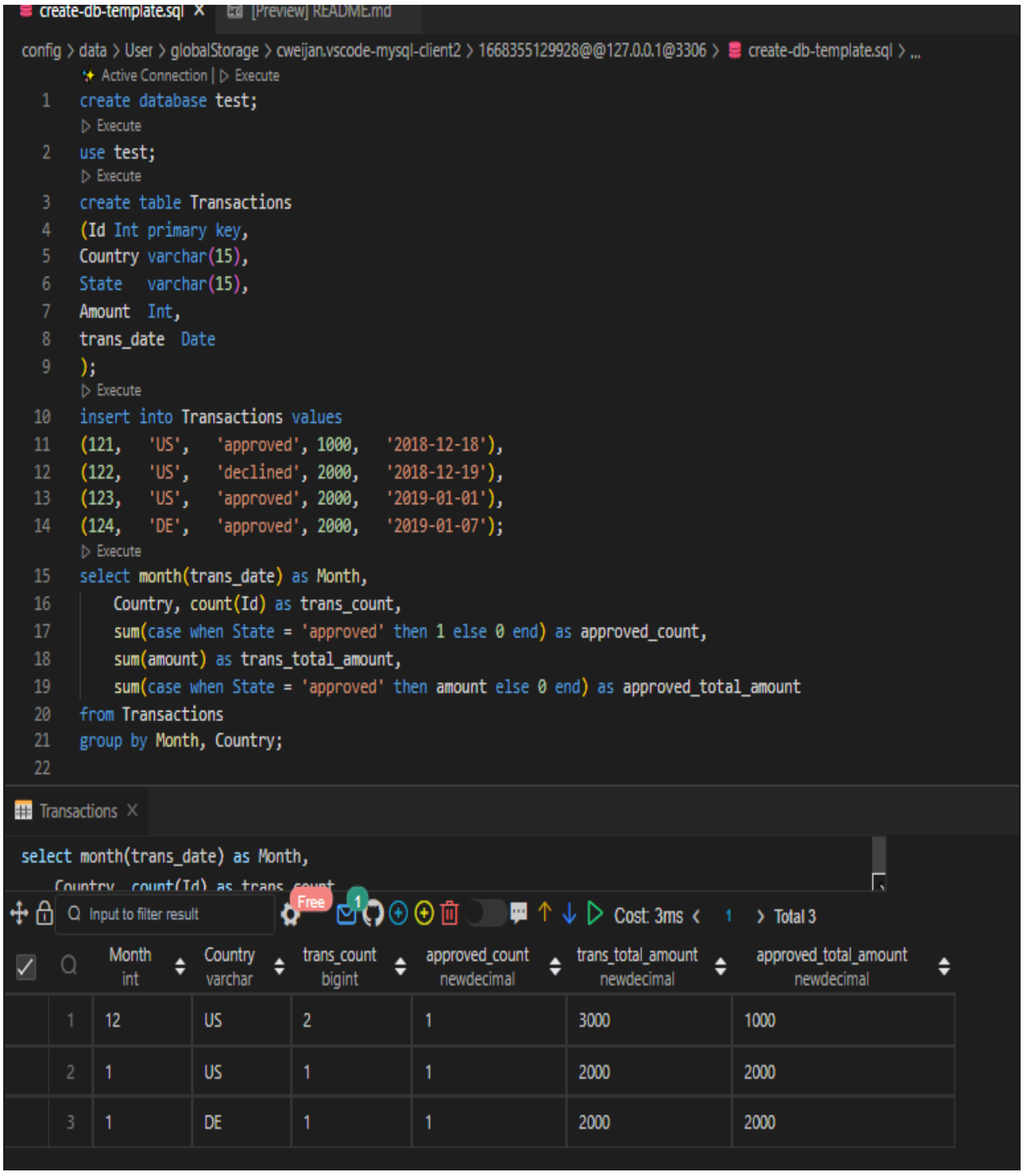
Input: Transactions
table:

id	country	state	amount	trans_date
121	US	approved	1000	2018-12-18
122	US	declined	2000	2018-12-19
123	US	approved	2000	2019-01-01
124	DE	approved	2000	2019-01-07

Output:

month	Country	trans_count	approved_count	trans_total_amount	roved_total_am
12	US	2	1	3000	1000
1	US	1	1	2000	2000
1	DE	1	1	2000	2000

```
Solution:
select month(trans_date) as Month,
    Country, count(Id) as trans_count,
    sum(case when State = 'approved' then 1 else 0 end) as approved_count,
    sum(amount) as trans_total_amount,
    sum(case when State = 'approved' then amount else 0 end) as
approved_total_amount
from Transactions
group by Month, Country;
```



Q73.

Table: Actions

Column Name	Type
user_id	Int
post_id	Int
action_date	Date
action	Enum
extra	Varchar

There is no primary key for this table, it may have duplicate rows.
The action column is an ENUM type of ('view', 'like', 'reaction', 'comment', 'report', 'share').
The extra column has optional information about the action, such as a reason for the report or a type of reaction.

Table: Removals

Column Name	Type
post_id	Int
remove_date	Date

post_id is the primary key of this table.
Each row in this table indicates that some post was removed due to being reported or as a result of an admin review.

Write an SQL query to find the average daily percentage of posts that got removed after being reported as spam, rounded to 2 decimal places.
The query result format is in the following example.

Input:
Actions table:

user_id	post_id	action_date	action	extra
1	1	2019-07-01	view	null
1	1	2019-07-01	like	null
1	1	2019-07-01	share	null
2	2	2019-07-04	view	null
2	2	2019-07-04	report	spam
3	4	2019-07-04	view	null
3	4	2019-07-04	report	spam
4	3	2019-07-02	view	null
4	3	2019-07-02	report	spam

5	2	2019-07-03	view	null
5	2	2019-07-03	report	racism
5	5	2019-07-03	view	null
5	5	2019-07-03	report	racism

Removals table:

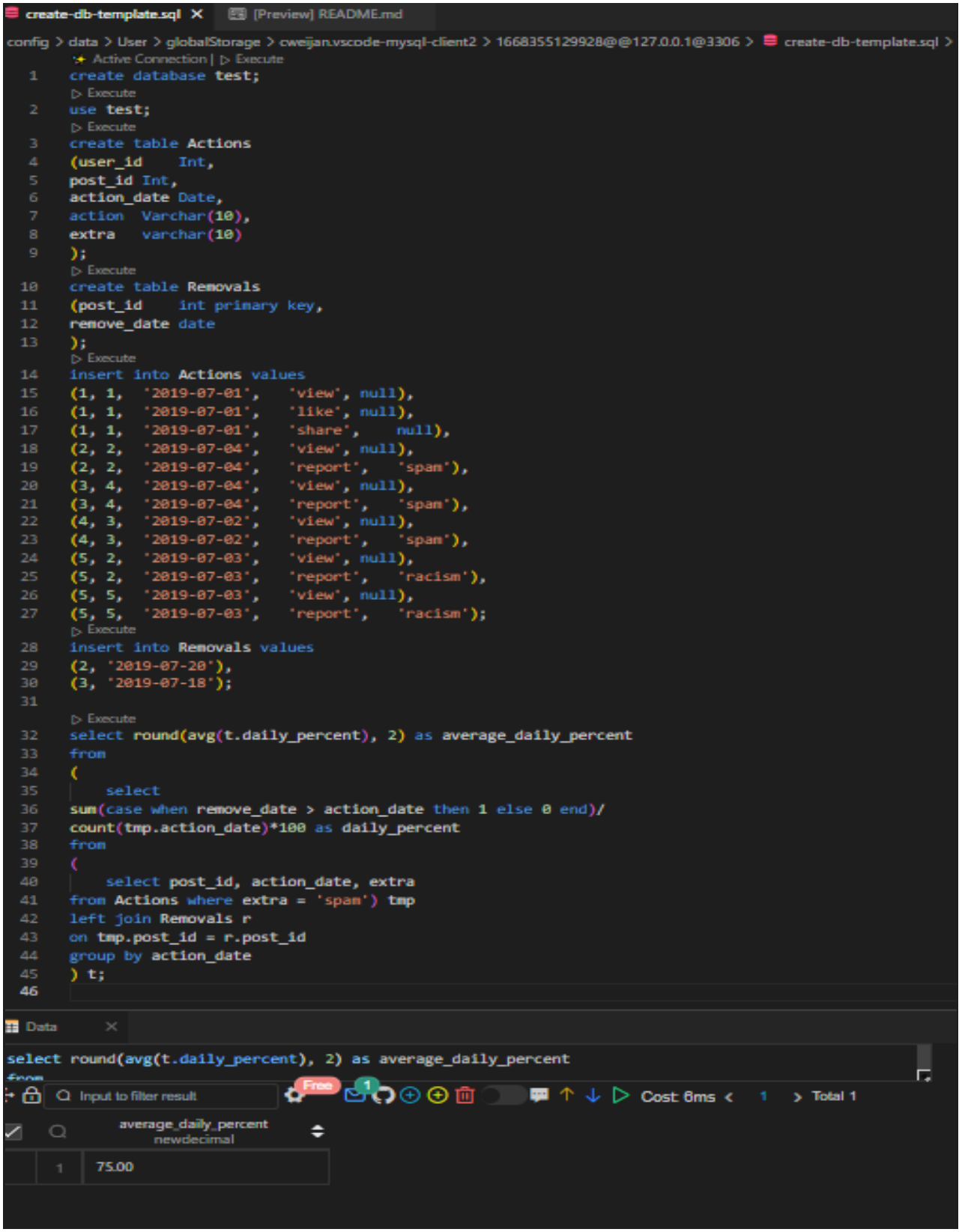
post_id	remove_date
2	2019-07-20
3	2019-07-18

Output:

average_daily_percent
75

Explanation:
The percentage for 2019-07-04 is 50% because only one post of two spam reported posts were removed.
The percentage for 2019-07-02 is 100% because one post was reported as spam and it was removed. The other days had no spam reports so the average is (50 + 100) / 2 = 75%
Note that the output is only one number and that we do not care about the remove dates.

```
Solution:
select round(avg(t.daily_percent), 2) as average_daily_percent
from
(
    select
sum(case when remove_date > action_date then 1 else 0 end)/
count(tmp.action_date)*100 as daily_percent
from
(
    select post_id, action_date, extra
from Actions where extra = 'spam') tmp
left join Removals r
on tmp.post_id = r.post_id
group by action_date
) t;
```



Q74.

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.

Write an SQL query to report the fraction of players that logged in again on the day after the day they first logged in, rounded to 2 decimal places. In other words, you need to count the number of players that logged in for at least two consecutive days starting from their first login date, then divide that number by the total number of players.

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	2016-03-02	0
3	4	2018-07-03	5

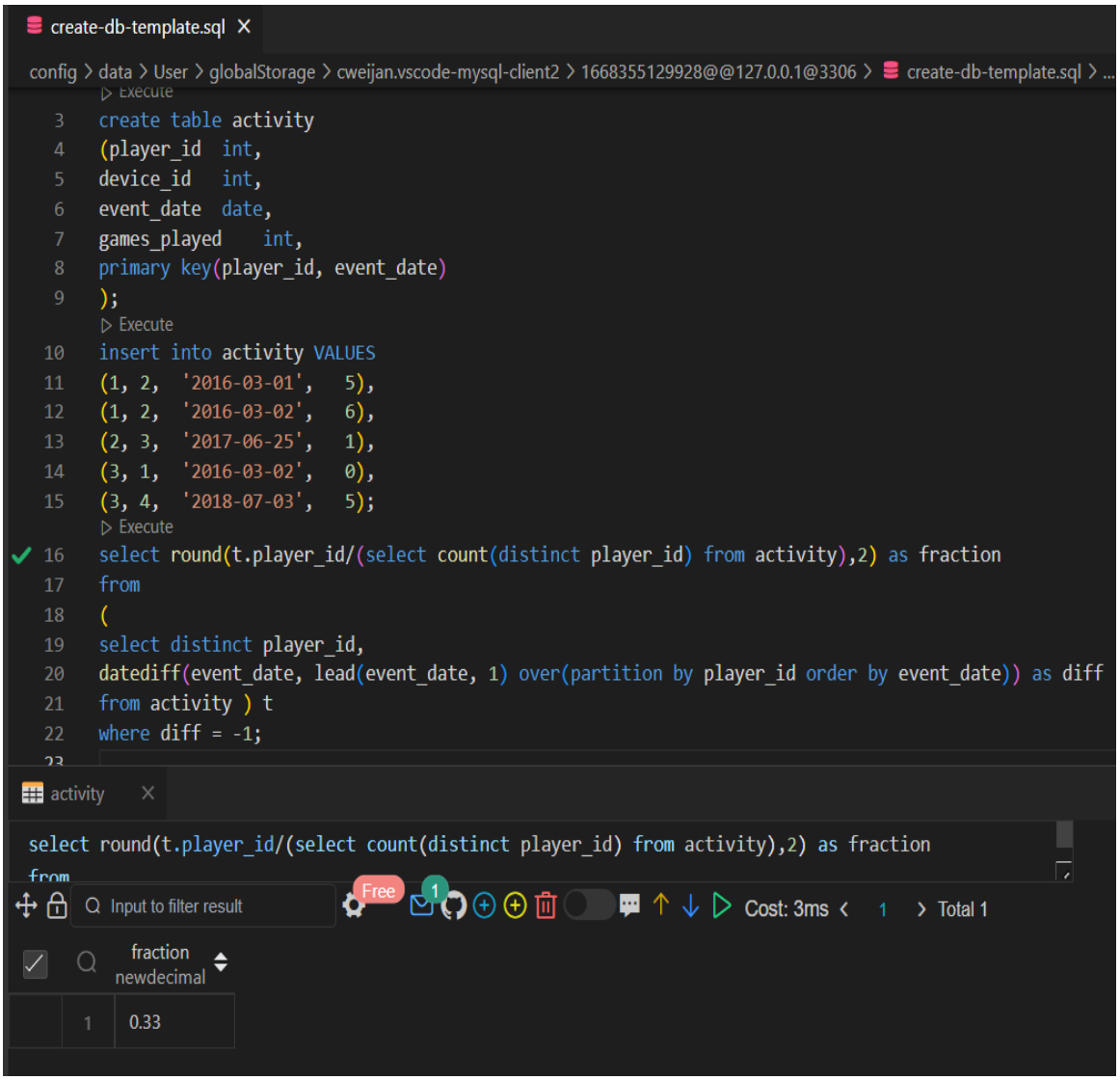
Output:

fraction
0.33

Explanation:

Only the player with id 1 logged back in after the first day he had logged in so the answer is 1/3 = 0.33

```
Solution:
select round(t.player_id/(select count(distinct player_id) from activity),2) as
fraction
from
(
select distinct player_id,
datediff(event_date, lead(event_date, 1) over(partition by player_id order by
event_date)) as diff
from activity ) t
where diff = -1;
```


create-db-template.sql
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-template.sql > ...
Execute
create table activity
(player_id int,
device_id int,
event_date date,
games_played int,
primary key(player_id, event_date)
);
Execute
insert into activity VALUES
(1, 2, '2016-03-01', 5),
(1, 2, '2016-03-02', 6),
(2, 3, '2017-06-25', 1),
(3, 1, '2016-03-02', 0),
(3, 4, '2018-07-03', 5);
Execute
select round(t.player_id/(select count(distinct player_id) from activity),2) as fraction
from
(
select distinct player_id,
datediff(event_date, lead(event_date, 1) over(partition by player_id order by event_date)) as diff
from activity) t
where diff = -1;
activity
select round(t.player_id/(select count(distinct player_id) from activity),2) as fraction
from
Input to filter result
Free
Cost: 3ms
1
Total 1
fraction
newdecimal
1
0.33

Q75.

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.

Write an SQL query to report the fraction of players that logged in again on the day after the day they first logged in, rounded to 2 decimal places. In other words, you need to count the number of players that logged in for at least two consecutive days starting from their first login date, then divide that number by the total number of players.
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	2016-03-02	0
3	4	2018-07-03	5

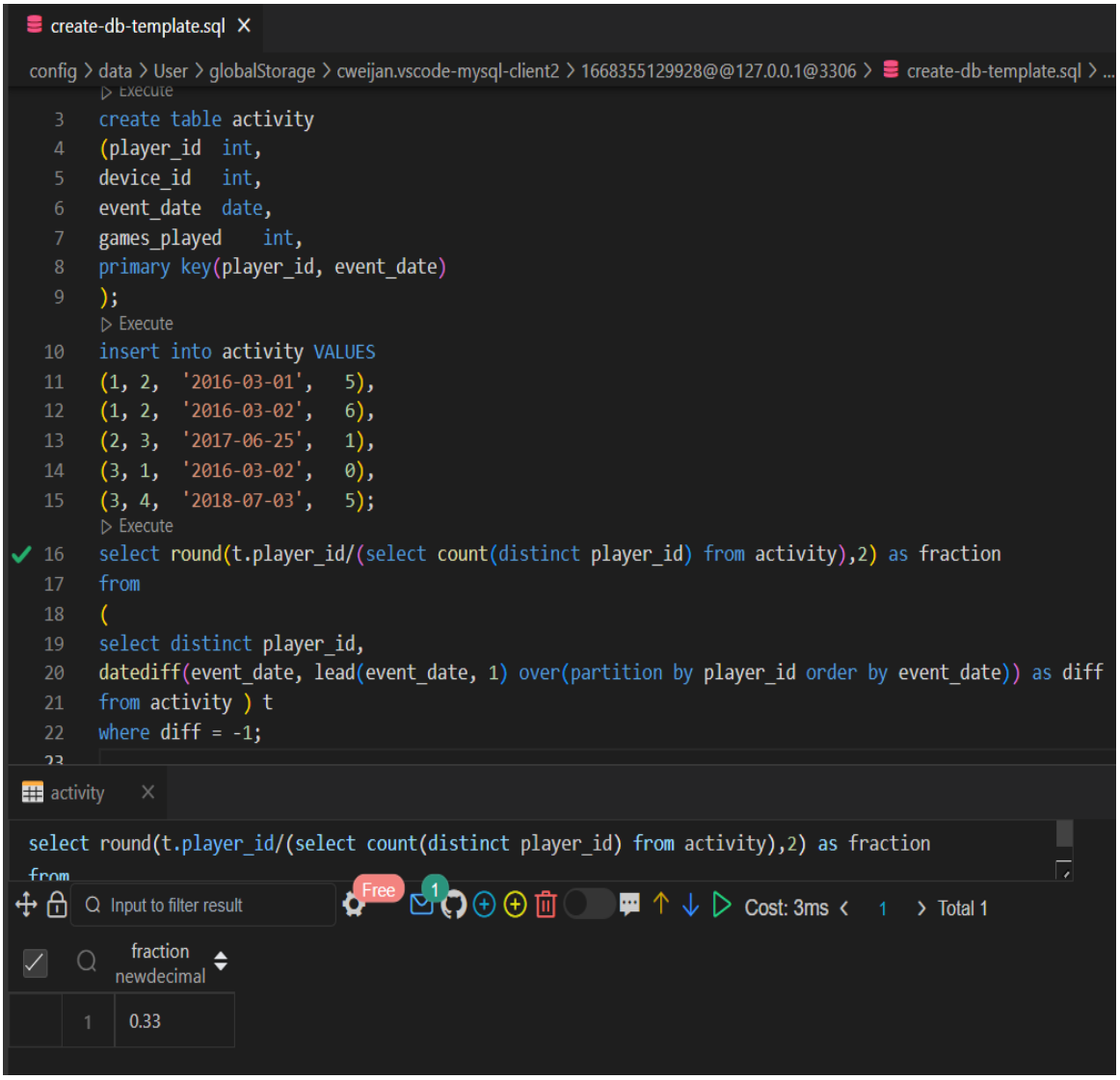
Output:

fraction
0.33

Explanation:
Only the player with id 1 logged back in after the first day he had logged in so the answer is 1/3 = 0.33

Solution:

```
select round(t.player_id/(select count(distinct player_id) from activity),2) as
fraction
from
(
select distinct player_id,
datediff(event_date, lead(event_date, 1) over(partition by player_id order by
event_date)) as diff
from activity ) t
where diff = -1;
```


create-db-template.sql
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-template.sql > ...
Execute
create table activity
(player_id int,
device_id int,
event_date date,
games_played int,
primary key(player_id, event_date)
);
Execute
insert into activity VALUES
(1, 2, '2016-03-01', 5),
(1, 2, '2016-03-02', 6),
(2, 3, '2017-06-25', 1),
(3, 1, '2016-03-02', 0),
(3, 4, '2018-07-03', 5);
Execute
select round(t.player_id/(select count(distinct player_id) from activity),2) as fraction
from
(
select distinct player_id,
datediff(event_date, lead(event_date, 1) over(partition by player_id order by event_date)) as diff
from activity) t
where diff = -1;
activity
select round(t.player_id/(select count(distinct player_id) from activity),2) as fraction
from
Input to filter result
Free
Cost: 3ms
1
Total 1
fraction
newdecimal
1
0.33

Q76.

Table Salaries:

Column Name	Type
company_id	int
employee_id	int
employee_name	varchar
salary	int

(company_id, employee_id) is the primary key for this table.
This table contains the company id, the id, the name, and the salary for an employee.

Write an SQL query to find the salaries of the employees after applying taxes. Round the salary to the nearest integer.
The tax rate is calculated for each company based on the following criteria:

- 0% If the max salary of any employee in the company is less than $1000.
- 24% If the max salary of any employee in the company is in the range [1000, 10000] inclusive.
- 49% If the max salary of any employee in the company is greater than $10000.

Return the result table in any order.
The query result format is in the following example.

Input: Salaries
table:

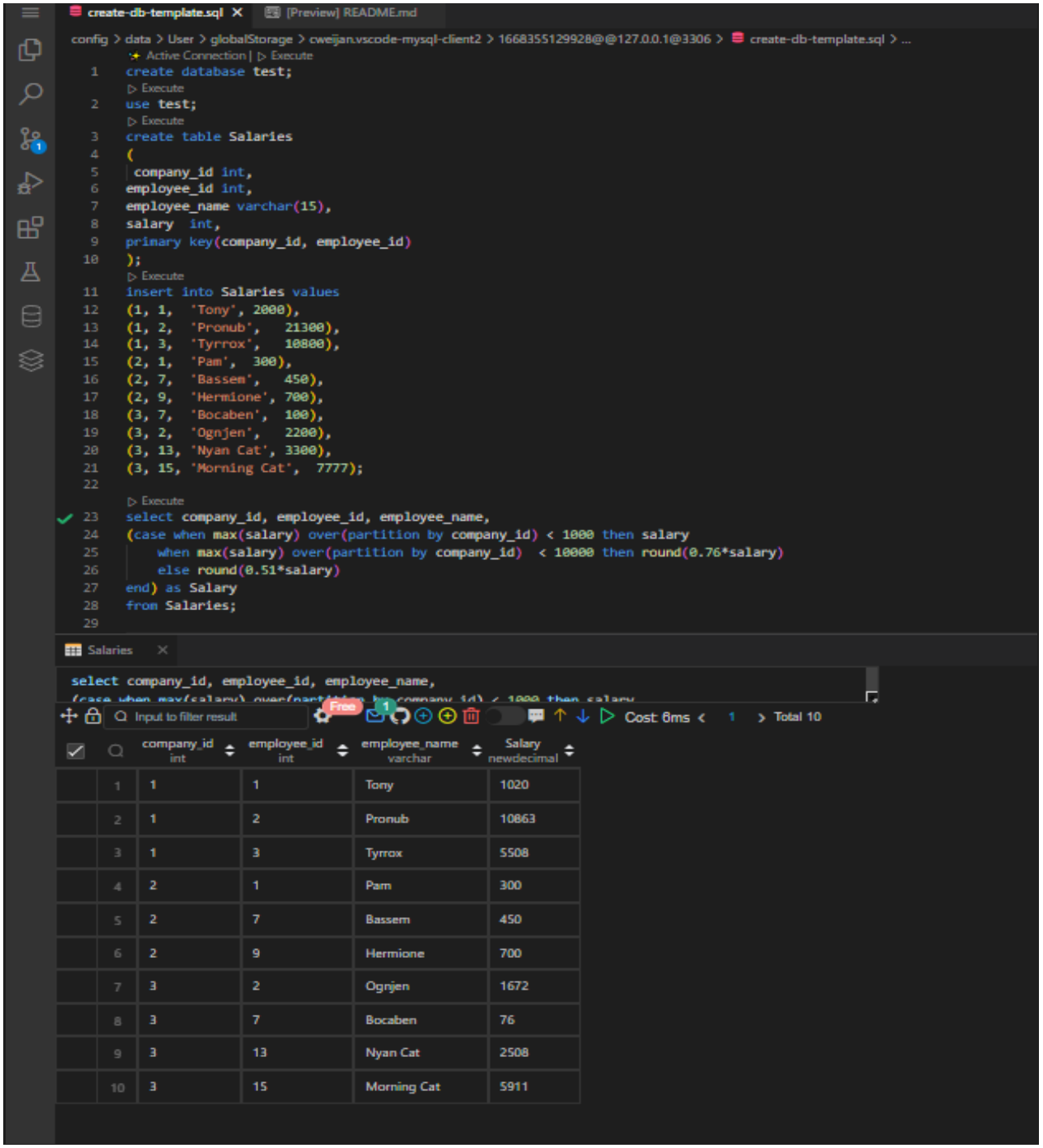
company_id	employee_id	employee_name	salary
1	1	Tony	2000
1	2	Pronub	21300
1	3	Tyrrox	10800
2	1	Pam	300
2	7	Bassem	450
2	9	Hermione	700
3	7	Bocaben	100
3	2	Ognjen	2200
3	13	Nyan Cat	3300
3	15	Morning Cat	7777

Output:

company_id	employee_id	employee_name	Salary
1	1	Tony	1020
1	2	Pronub	10863
1	3	Tyrrox	5508
2	1	Pam	300
2	7	Bassem	450
2	9	Hermione	700
3	7	Bocaben	76
3	2	Ognjen	1672
3	13	Nyan Cat	2508
3	15	Morning Cat	5911

Explanation:
For company 1, Max salary is 21300. Employees in company 1 have taxes = 49%
For company 2, Max salary is 700. Employees in company 2 have taxes = 0%
For company 3, Max salary is 7777. Employees in company 3 have taxes = 24%
The salary after taxes = salary - (taxes percentage / 100) * salary
For example, Salary for Morning Cat (3, 15) after taxes = 7777 - 7777 * (24 / 100) = 7777 - 1866.48 = 5910.52, which is rounded to 5911.

```
Solution:
select company_id, employee_id, employee_name,
(case when max(salary) over(partition by company_id) < 1000 then salary
    when max(salary) over(partition by company_id)  < 10000 then
round(0.76*salary)
    else round(0.51*salary)
end) as Salary
from Salaries;
```


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
create database test;
Execute
use test;
Execute
create table Salaries
(
company_id int,
employee_id int,
employee_name varchar(15),
salary int,
primary key(company_id, employee_id)
);
Execute
insert into Salaries values
(1, 1, 'Tony', 2000),
(1, 2, 'Pronub', 21300),
(1, 3, 'Tyrrox', 10800),
(2, 1, 'Pam', 300),
(2, 7, 'Bassem', 450),
(2, 9, 'Hermione', 700),
(3, 7, 'Bocaben', 100),
(3, 2, 'Ognjen', 2200),
(3, 13, 'Nyan Cat', 3300),
(3, 15, 'Morning Cat', 7777);
Execute
select company_id, employee_id, employee_name,
(case when max(salary) over(partition by company_id) < 1000 then salary
when max(salary) over(partition by company_id) < 10000 then round(0.76*salary)
else round(0.51*salary)
end) as Salary
from Salaries;
Salaries
Input to filter result
Free
Cost 6ms
Total 10
company_id int
employee_id int
employee_name varchar
Salary newdecimal
1 1 1 Tony 1020
2 1 2 Pronub 10863
3 1 3 Tyrrox 5508
4 2 1 Pam 300
5 2 7 Bassem 450
6 2 9 Hermione 700
7 3 2 Ognjen 1672
8 3 7 Bocaben 76
9 3 13 Nyan Cat 2508
10 3 15 Morning Cat 5911

Q77.

Table Variables:

Column Name	Type
name	varchar
value	int

name is the primary key for this table.
This table contains the stored variables and their values.

Table Expressions:

Column Name	Type
left_operand	varchar
operator	enum
right_operand	varchar

(left_operand, operator, right_operand) is the primary key for this table.
This table contains a boolean expression that should be evaluated.
operator is an enum that takes one of the values ('<', '>', '=')
The values of left_operand and right_operand are guaranteed to be in the Variables table.

Write an SQL query to evaluate the boolean expressions in Expressions table.
Return the result table in any order.
The query result format is in the following example.

Input: Variables
table:

name	value
x	66
y	77

Expressions table:

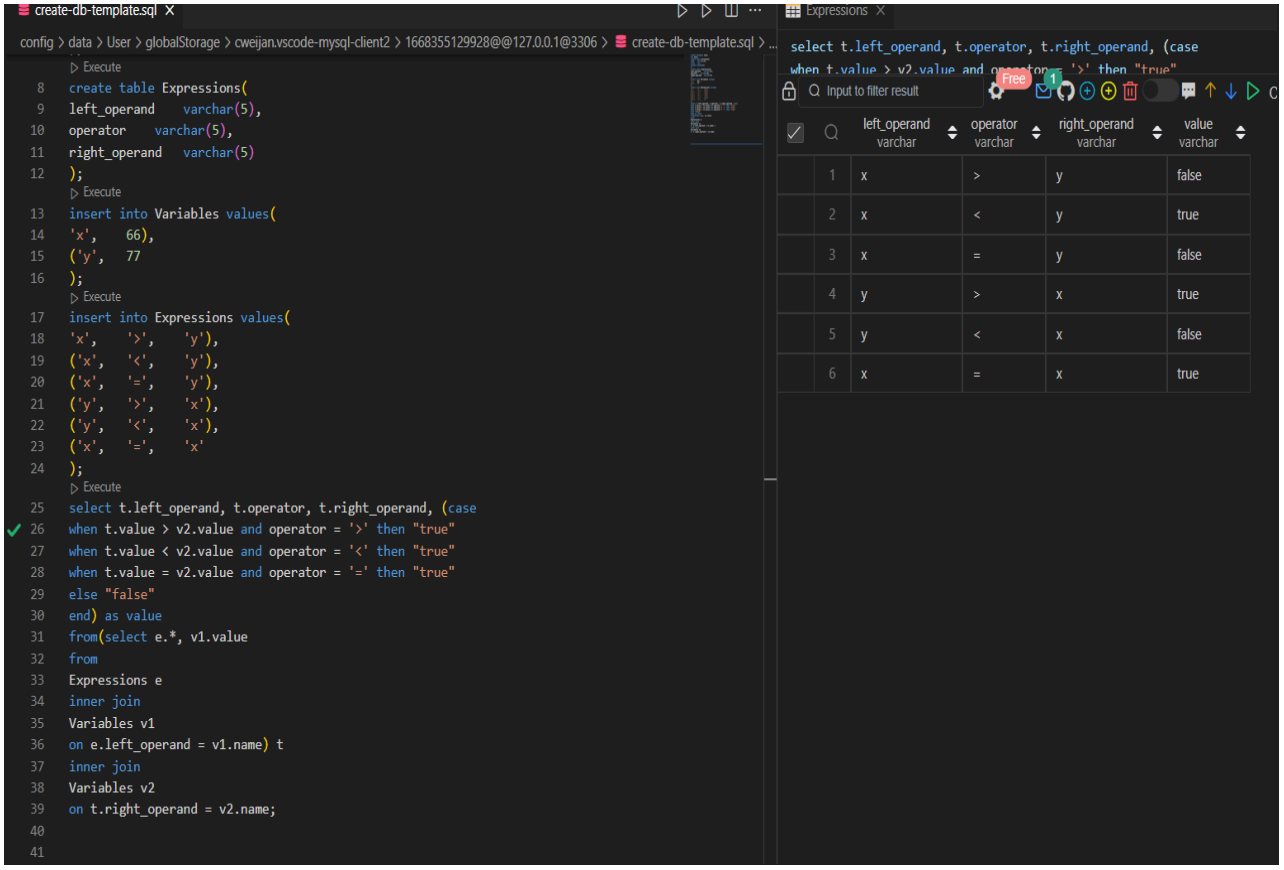
left_operand	operator	right_operand
x	>	y
x	<	y
x	=	y
y	>	x
y	<	x
x	=	x

Output:

left_operand	operator	right_operand	value
x	>	y	false
x	<	y	true
x	=	y	false
y	>	x	true
y	<	x	false
x	=	x	true

Explanation:
As shown, you need to find the value of each boolean expression in the table using the variables table.

```
Solution:
select t.left_operand, t.operator, t.right_operand, (case
when t.value > v2.value and operator = '>' then "true"
when t.value < v2.value and operator = '<' then "true"
when t.value = v2.value and operator = '=' then "true"
else "false"
end) as value
from(select e.*, v1.value
from
Expressions e
inner join
Variables v1
on e.left_operand = v1.name) t
inner join
Variables v2
on t.right_operand = v2.name;
```



Q78.

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

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

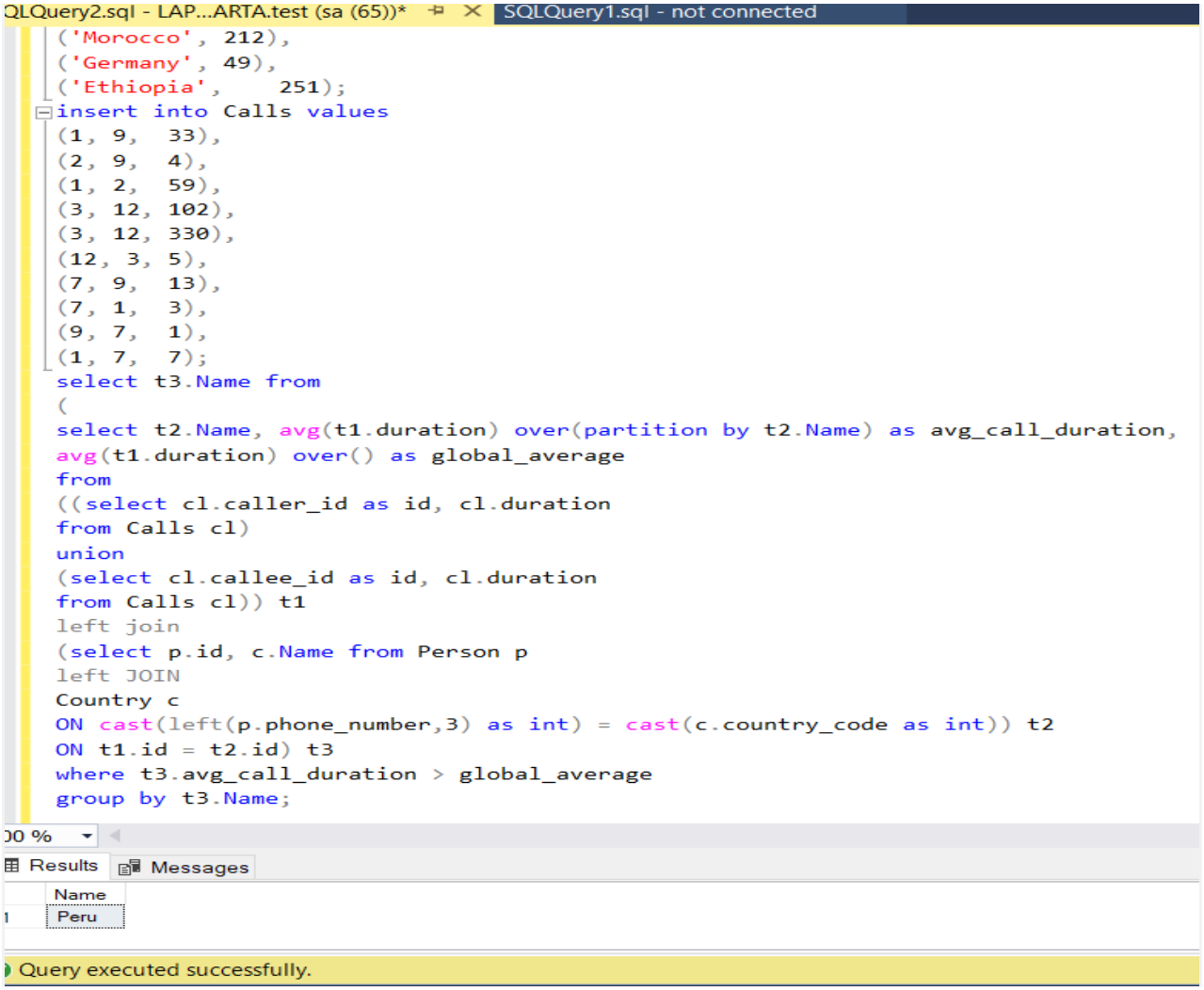
7	1	3
9	7	1
1	7	7

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;
```



Q79.

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.

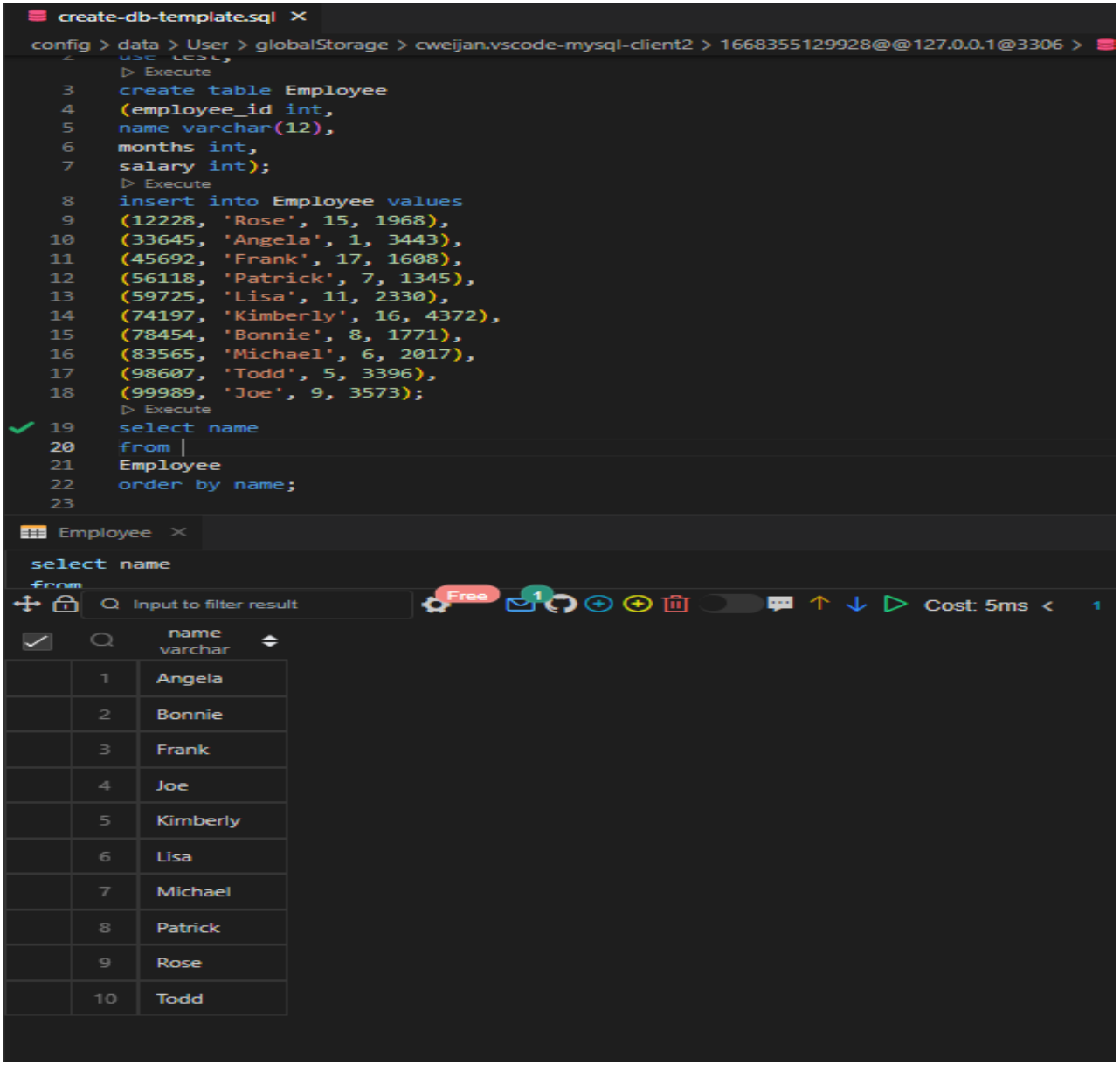
Sample Input

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;
```



Q80.

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:

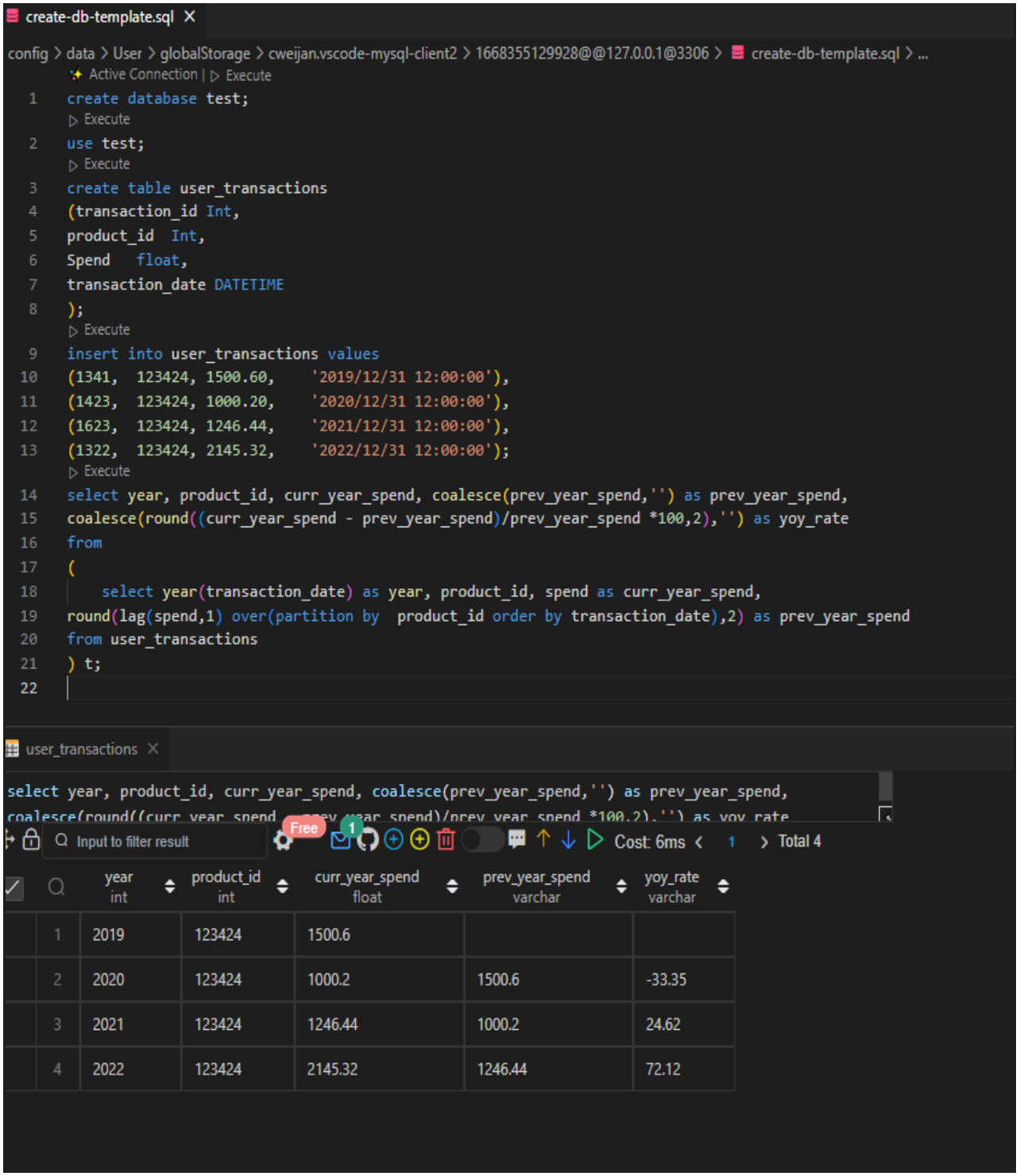
transaction_id	product_id	spend	transaction_date
1341	123424	1500.60	12/31/2019 12:00:00
1423	123424	1000.20	12/31/2020 12:00:00
1623	123424	1246.44	12/31/2021 12:00:00
1322	123424	2145.32	12/31/2022 12:00:00

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;
```



Q81.

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

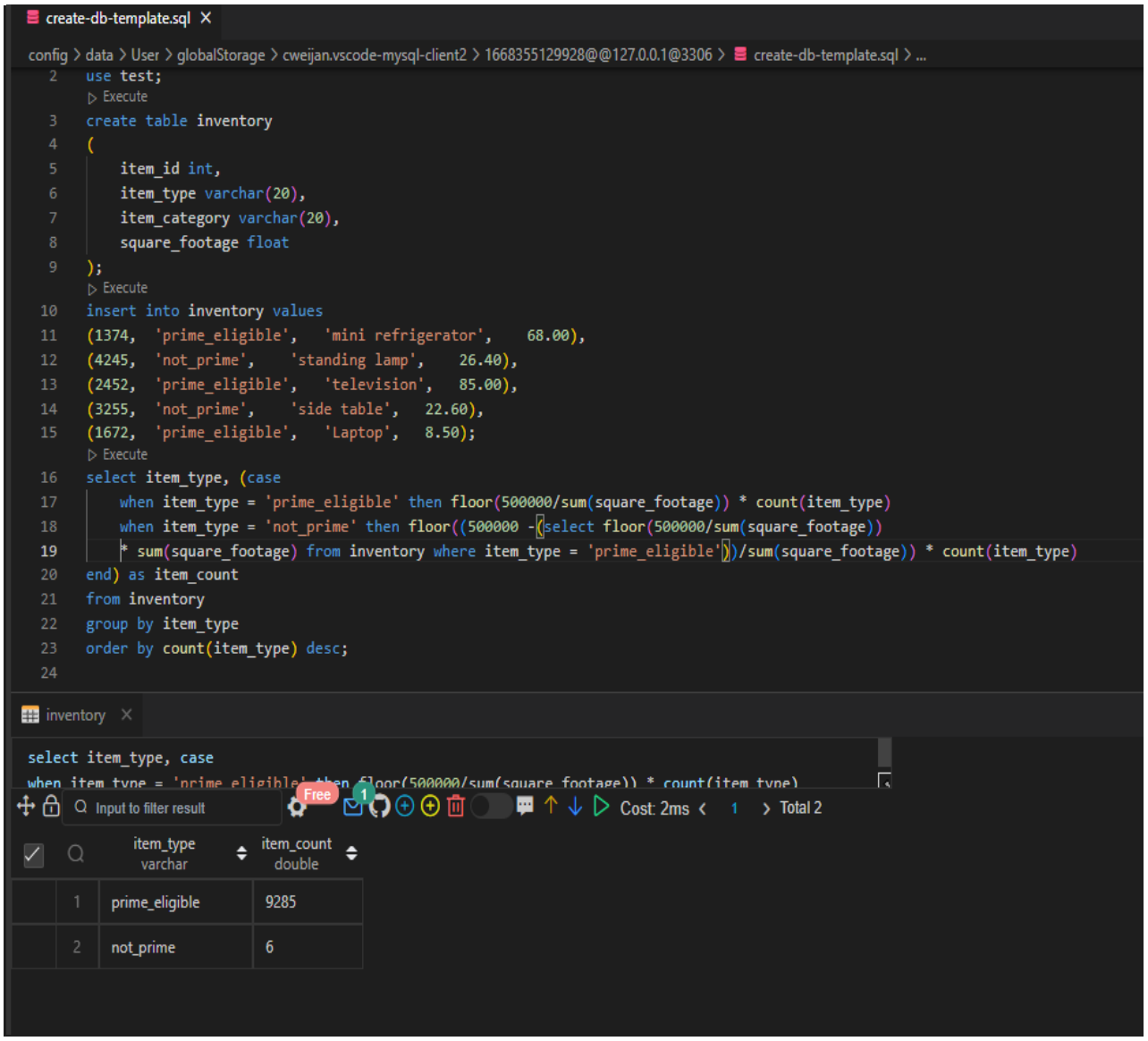
`inventory` Example Input:

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;
```



Q82.

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:

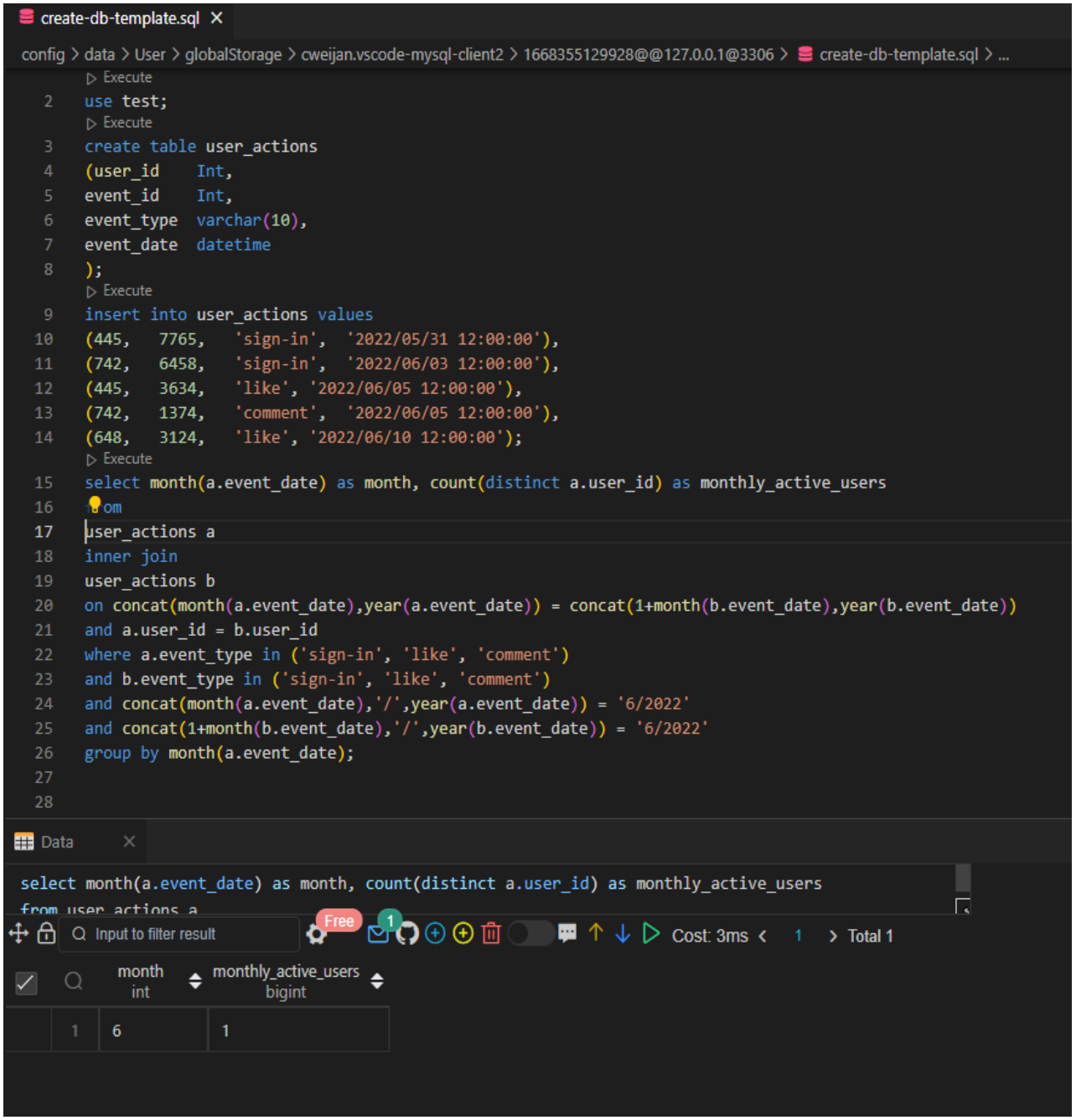
month	monthly_active_users
6	1

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



Q83.

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:

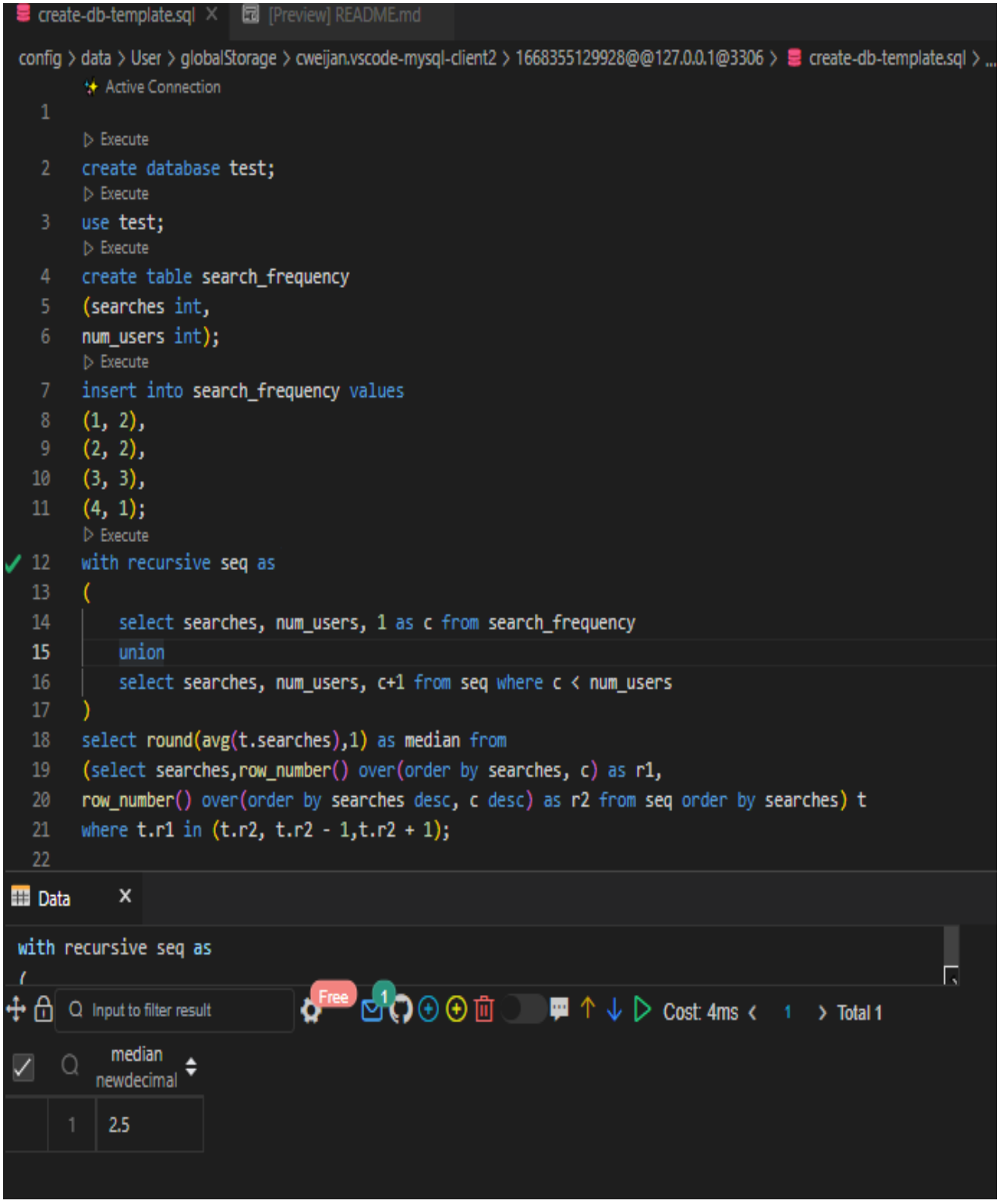
searches	num_users
1	2
2	2
3	3
4	1

Example Output:

median
2.5

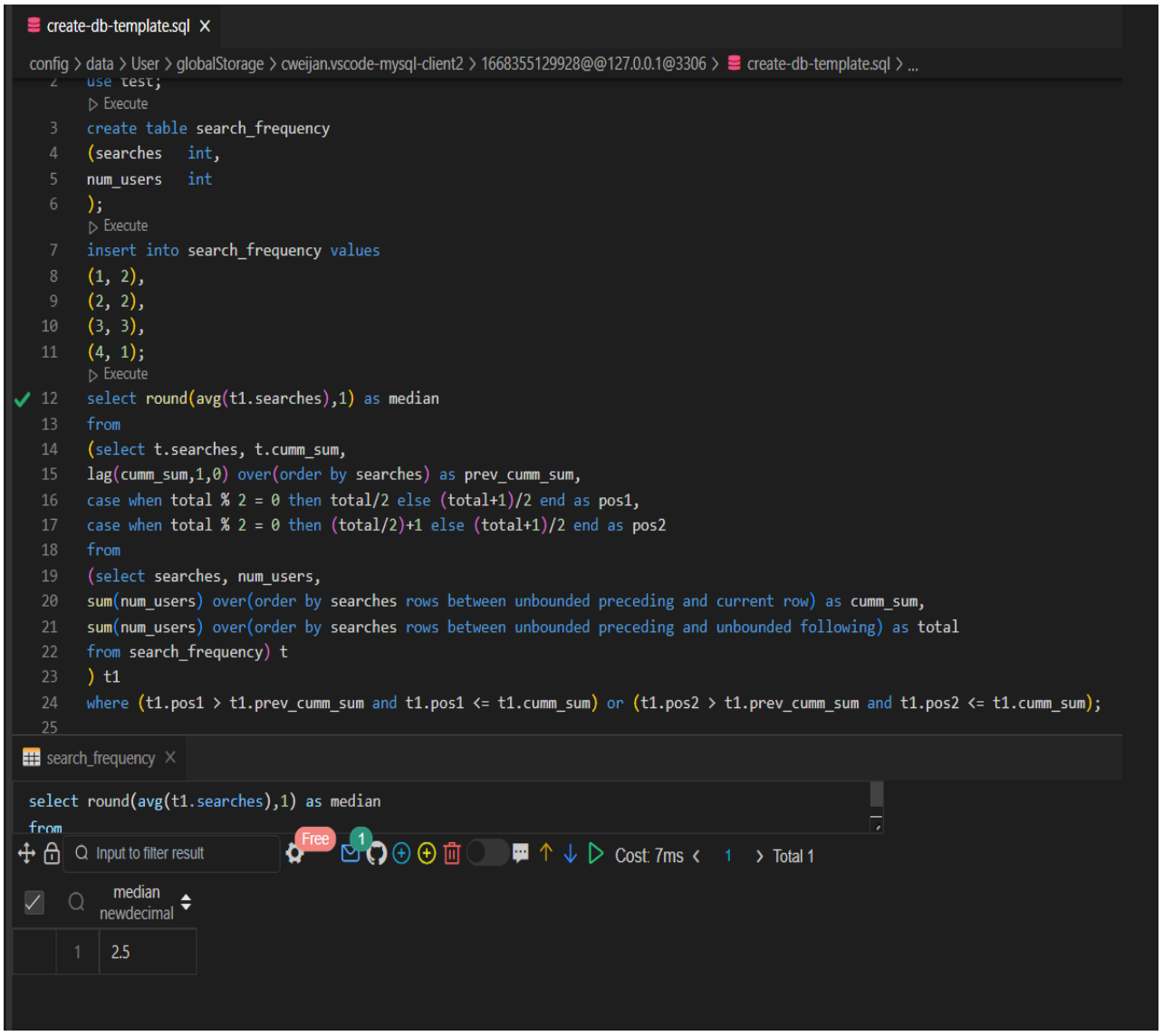
Solution: **using recursive cte**

```
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
create database test;
Execute
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
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);
Data
with recursive seq as
(
Input to filter result
Free
Cost: 4ms
1
Total 1
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);
```



Q84.

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

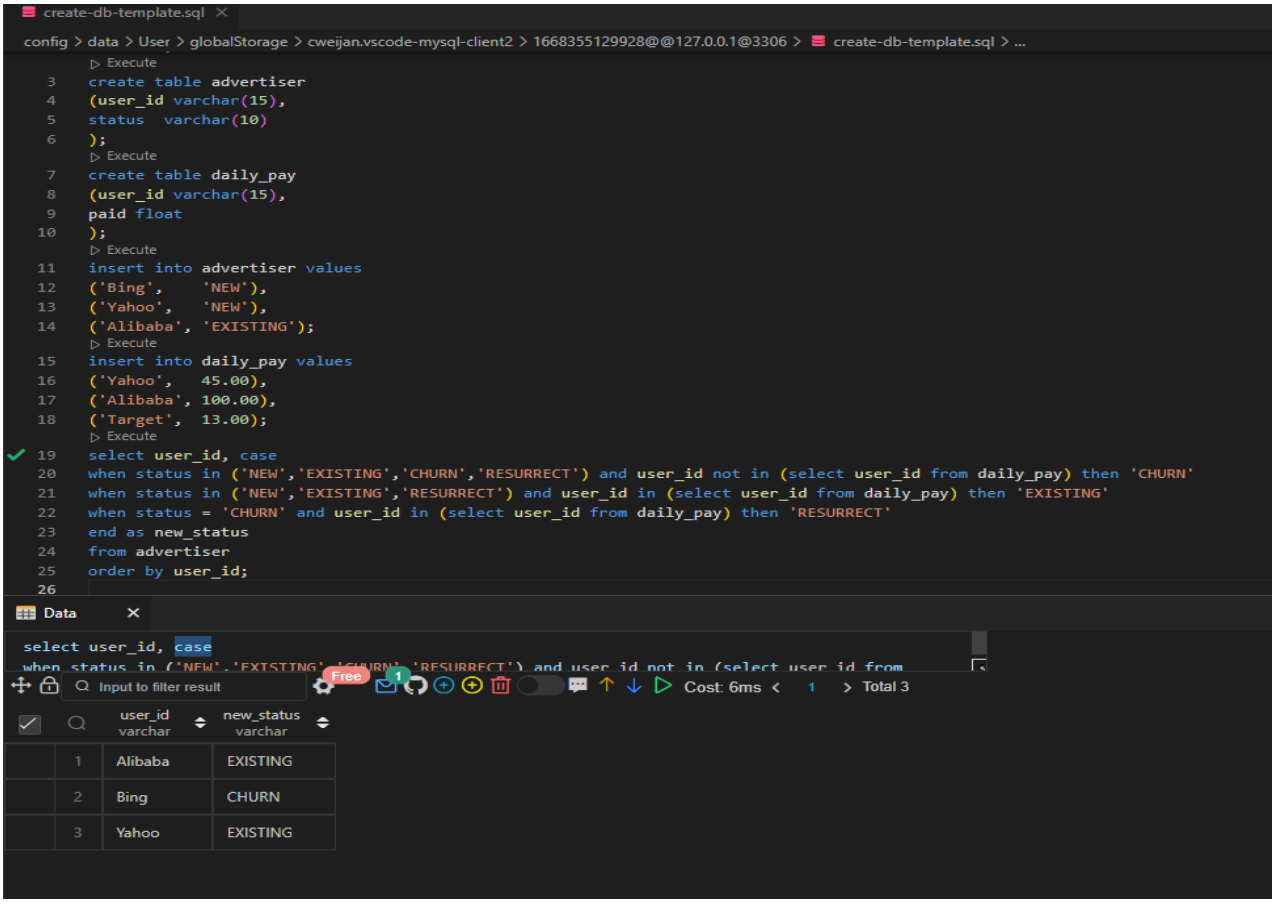
1. Row 2, 4, 6, 8: As long as the user has not paid on day T, the end status is updated to CHURN regardless of the previous status.
2. Row 1, 3, 5, 7: When the user paid on day T, the end status is updated to either EXISTING or RESURRECT, depending on their previous state. RESURRECT is only possible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



Q85.

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.

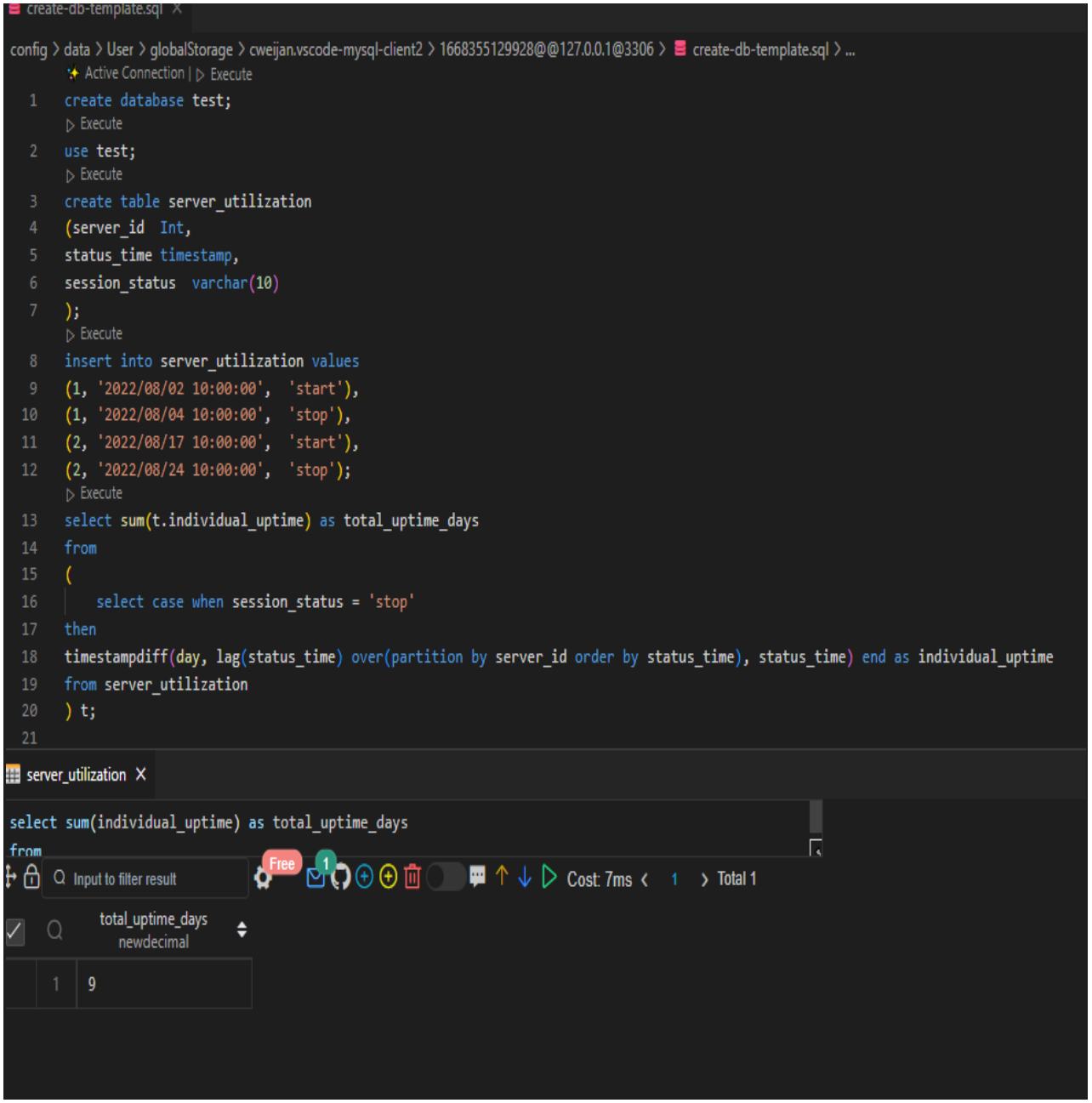
`server_utilization` Table:

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;
```



Q86.

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

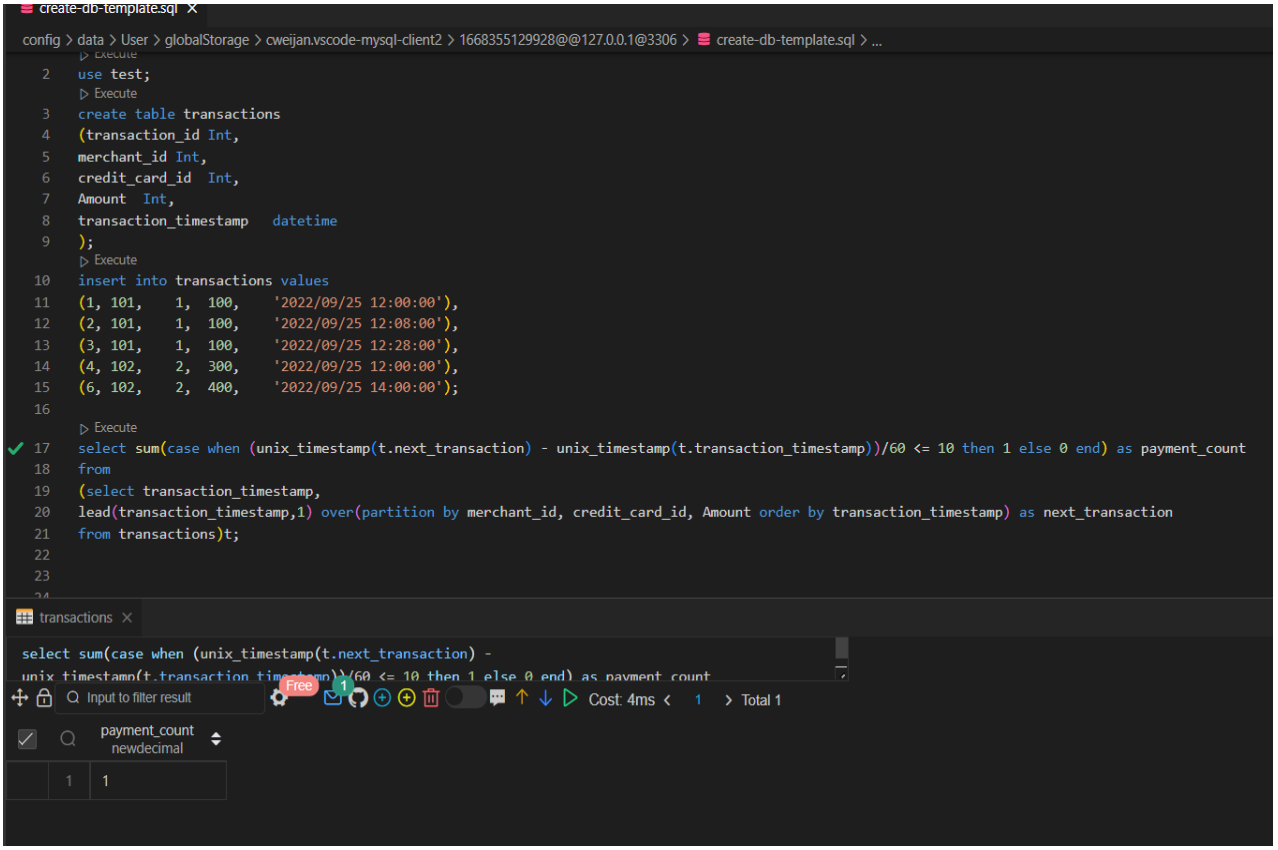
transactions Example Input:

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;
```



Q87.
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

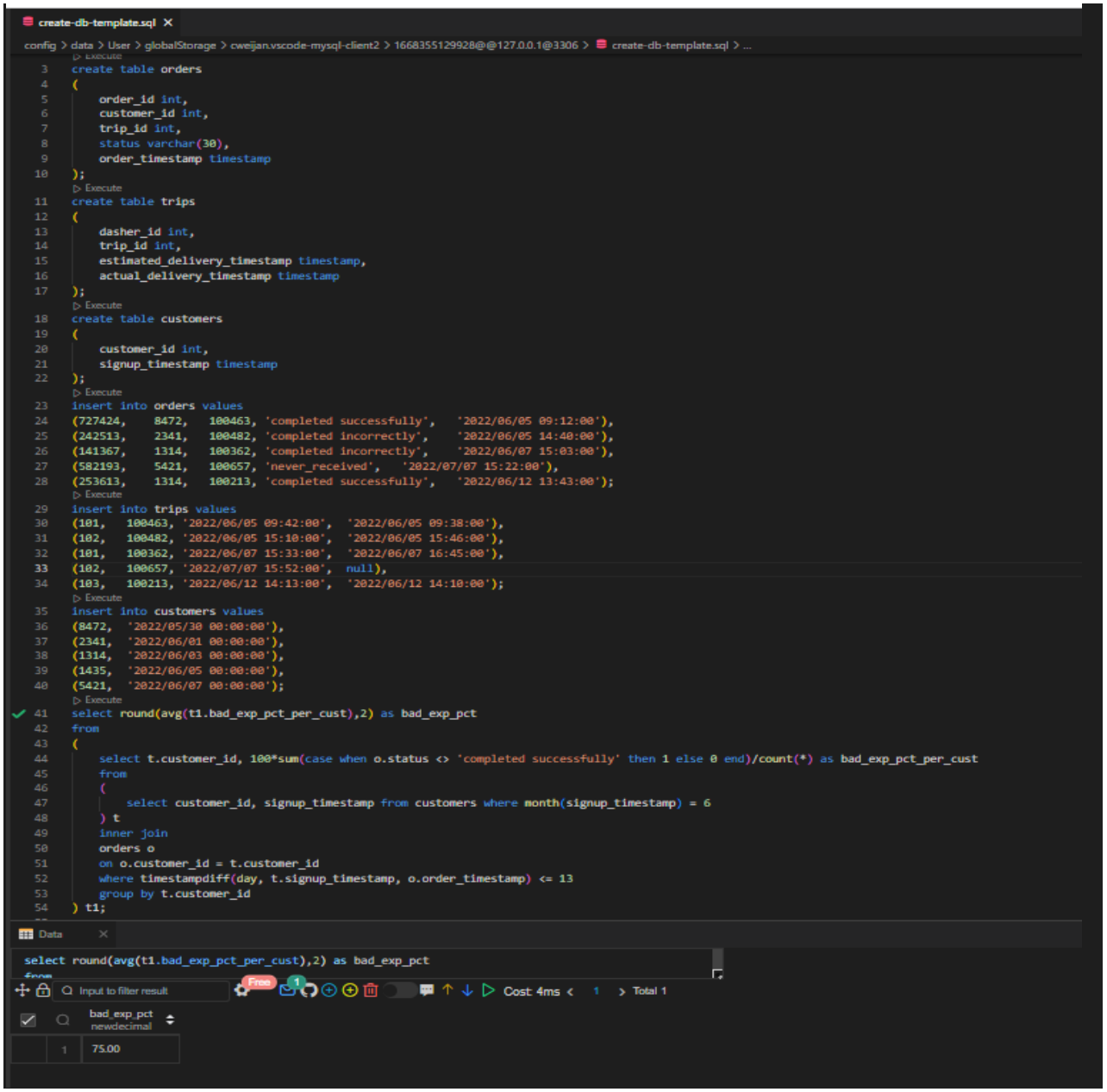
`customers` Example Input:

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
create table orders
(
order_id int,
customer_id int,
trip_id int,
status varchar(30),
order_timestamp timestamp
);
Execute
create table trips
(
dasher_id int,
trip_id int,
estimated_delivery_timestamp timestamp,
actual_delivery_timestamp timestamp
);
Execute
create table customers
(
customer_id int,
signup_timestamp timestamp
);
Execute
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');
Execute
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');
Execute
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');
Execute
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;
Data
select round(avg(t1.bad_exp_pct_per_cust),2) as bad_exp_pct
from
Input to filter result
Free
Cost 4ms
Total 1
bad_exp_pct
newdecimal
1
75.00

Q88

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

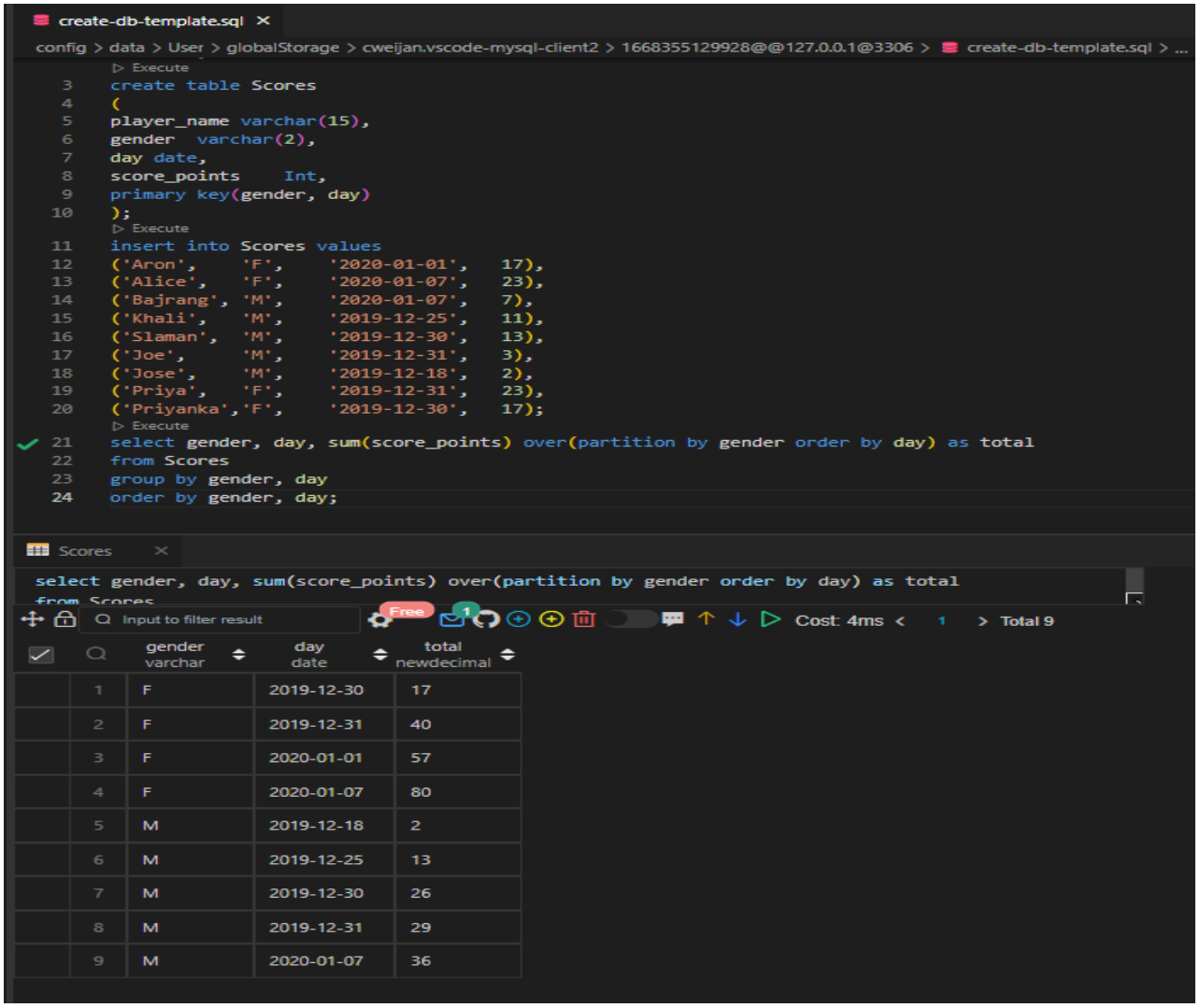
M	2019-12-30	26
M	2019-12-31	29
M	2020-01-07	36

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;
```



Q89.

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

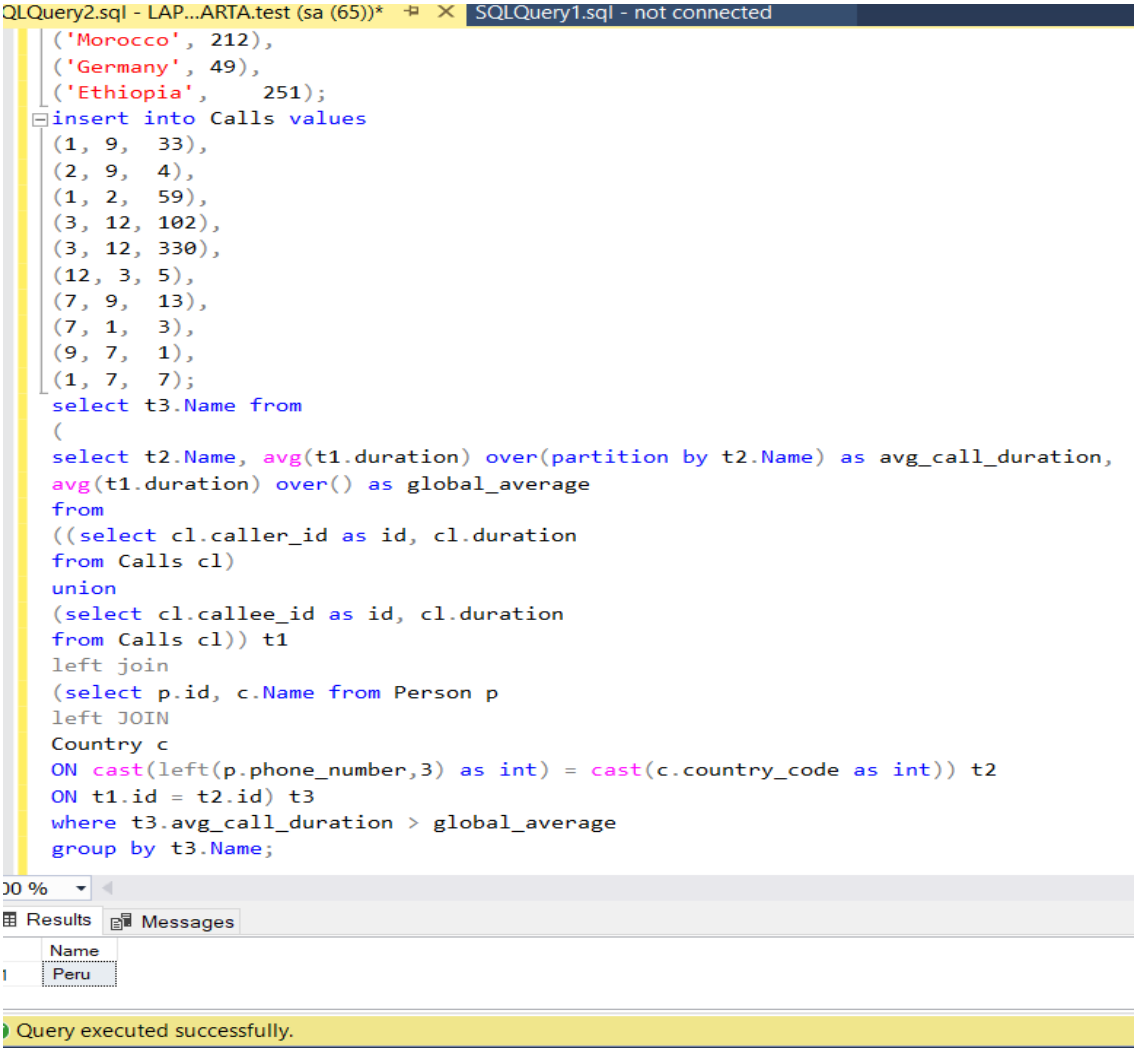
1	2	59
3	12	102
3	12	330
12	3	5
7	9	13
7	1	3
9	7	1
1	7	7

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;
```



Q90.

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.

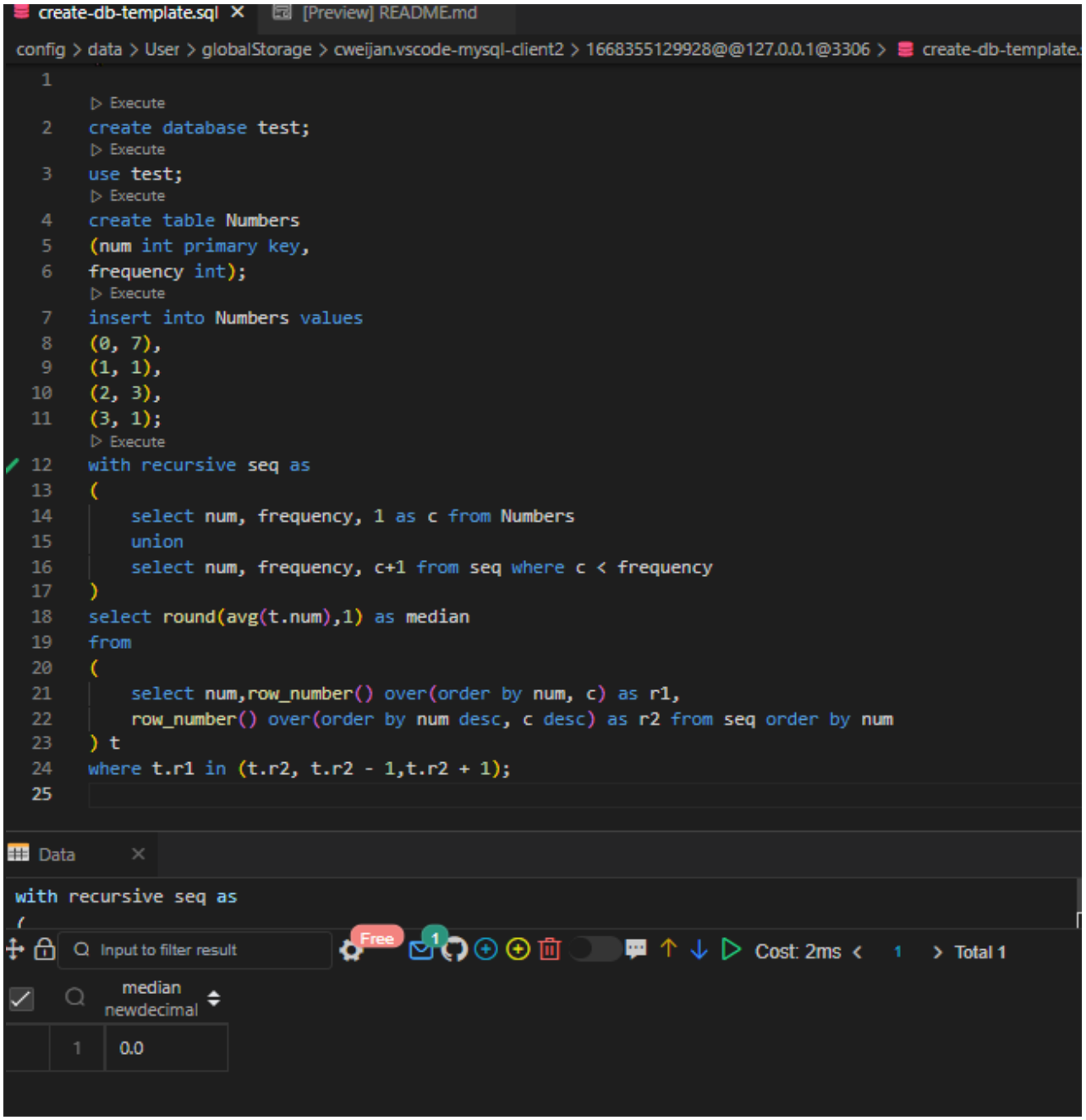
num	frequency
0	7
1	1
2	3
3	1

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.

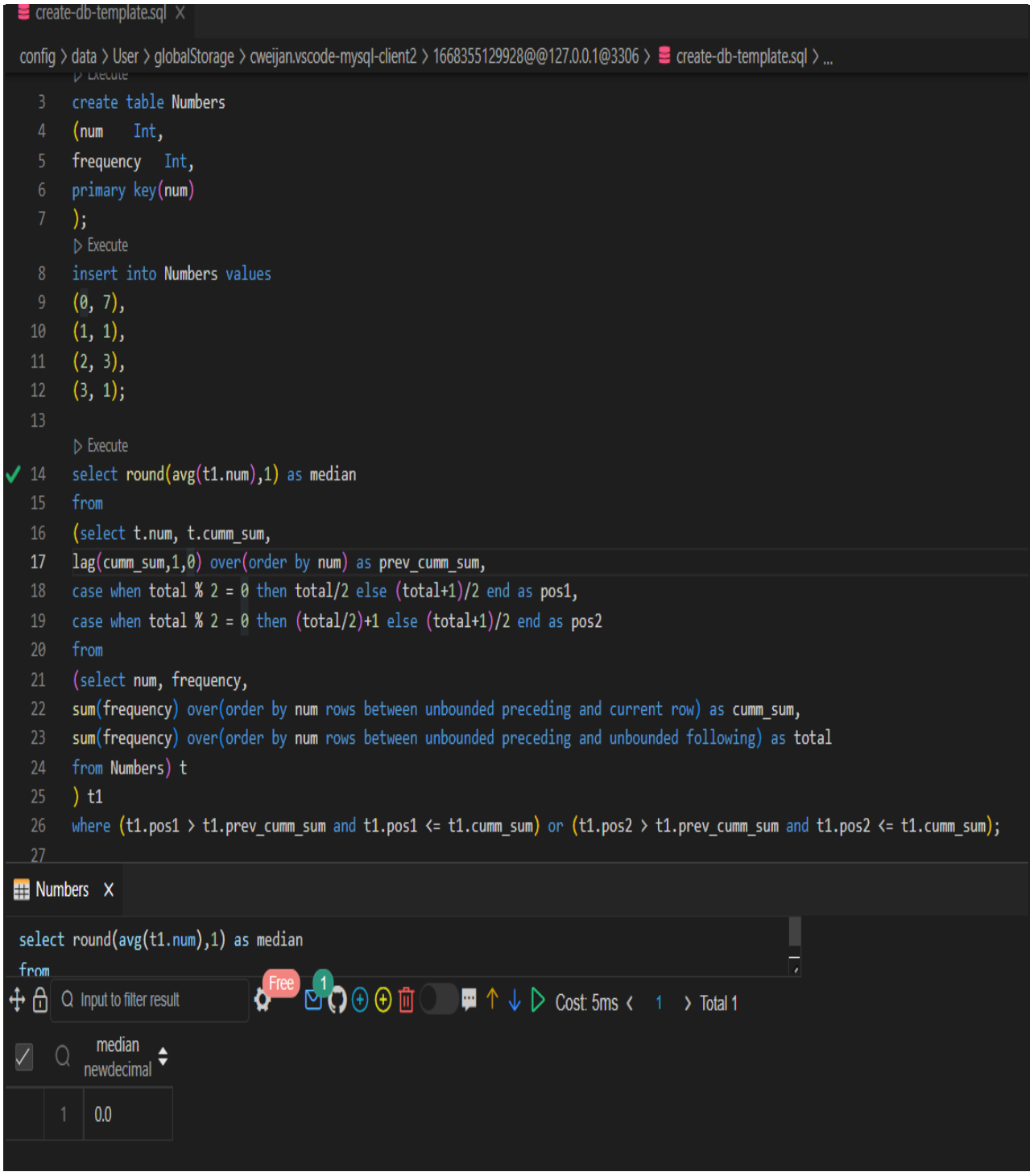
```
Solution: using recursive cte
with recursive seq as
(
    select num, frequency, 1 as c from Numbers
    union
    select num, frequency, c+1 from seq where c < frequency
)
select round(avg(t.num),1) as median
from
(
    select num,row_number() over(order by num, c) as r1,
    row_number() over(order 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);
```



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);
```



Q91.

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.

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:

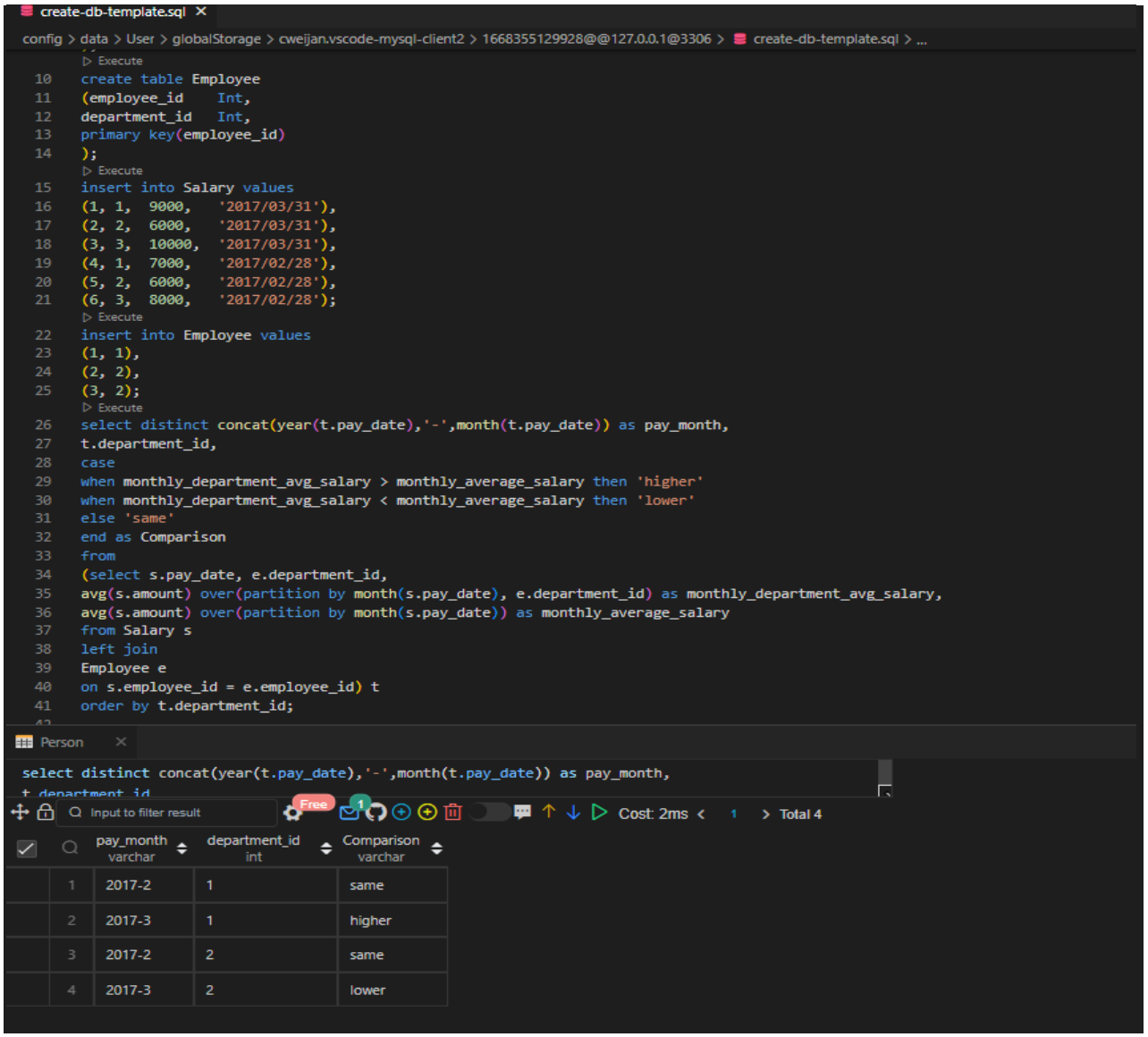
pay_month	department_id	comparison
2017-02	1	same
2017-03	1	higher
2017-02	2	same
2017-03	2	lower

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:

```mysql
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;
```



Q92

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

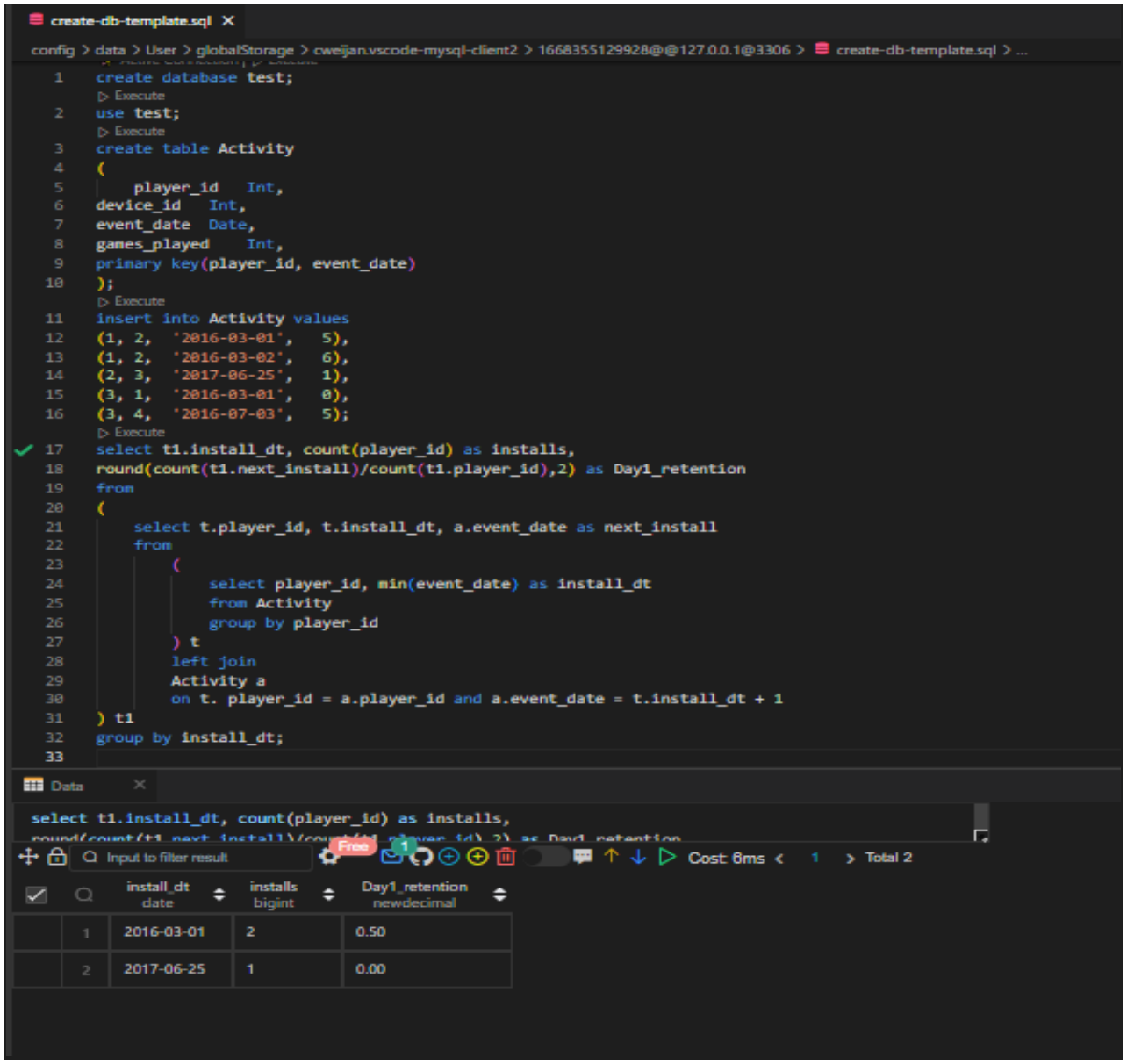
Output:

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:

```sql
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;
```



Q93.

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

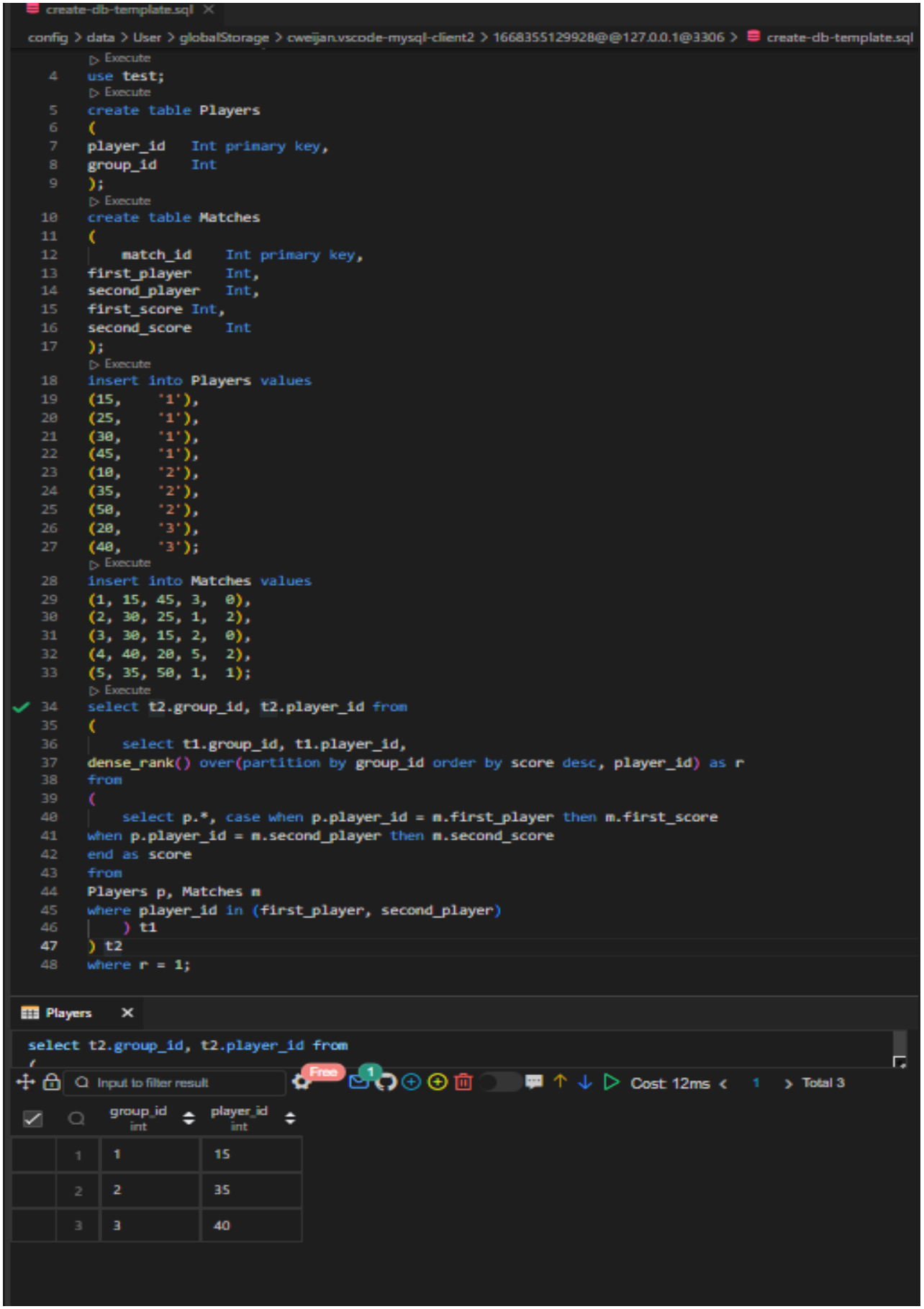
Matches table:

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;
```



Q94.

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

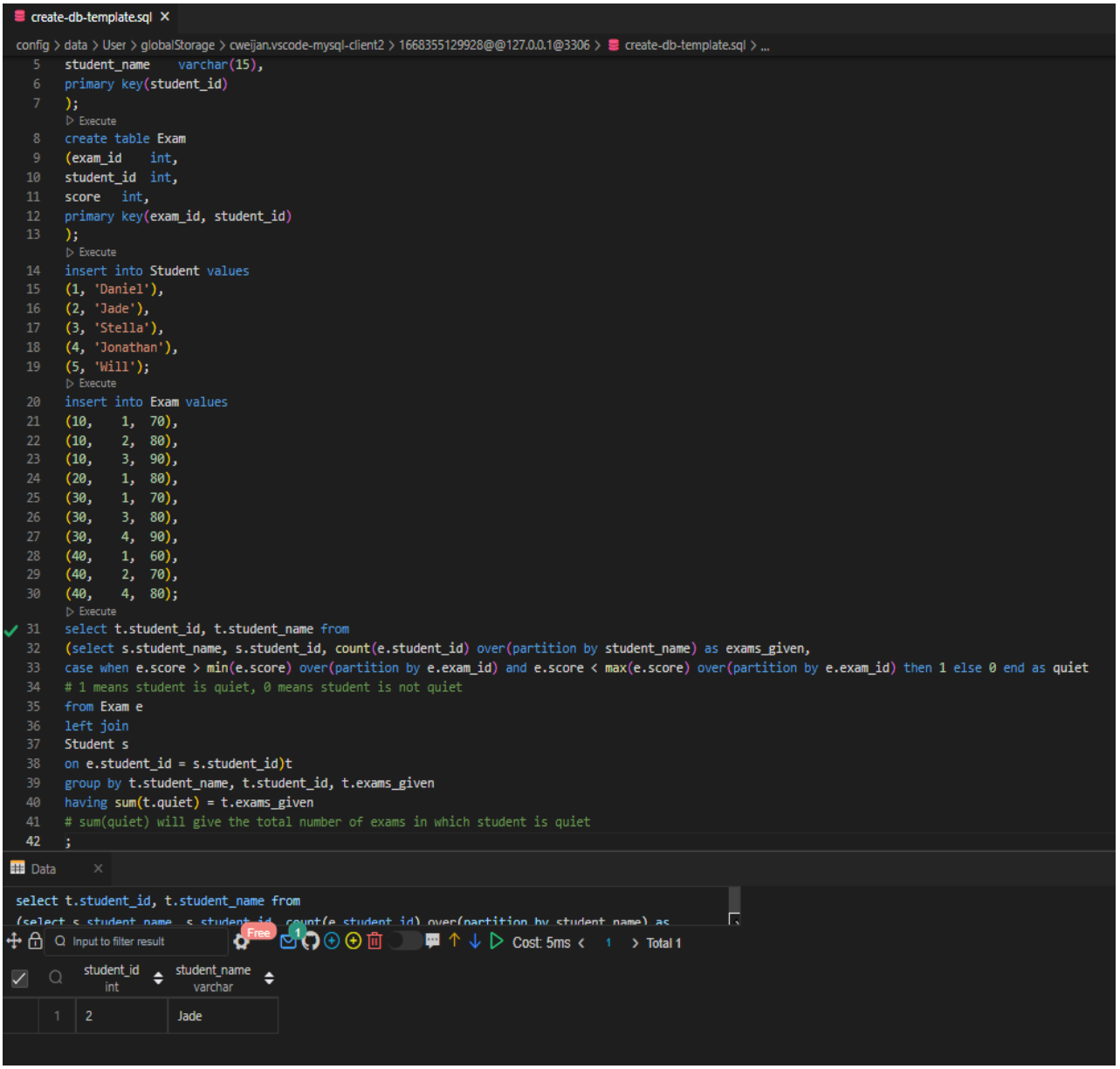
Output:

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:

```mysql
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
```



Q95.

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

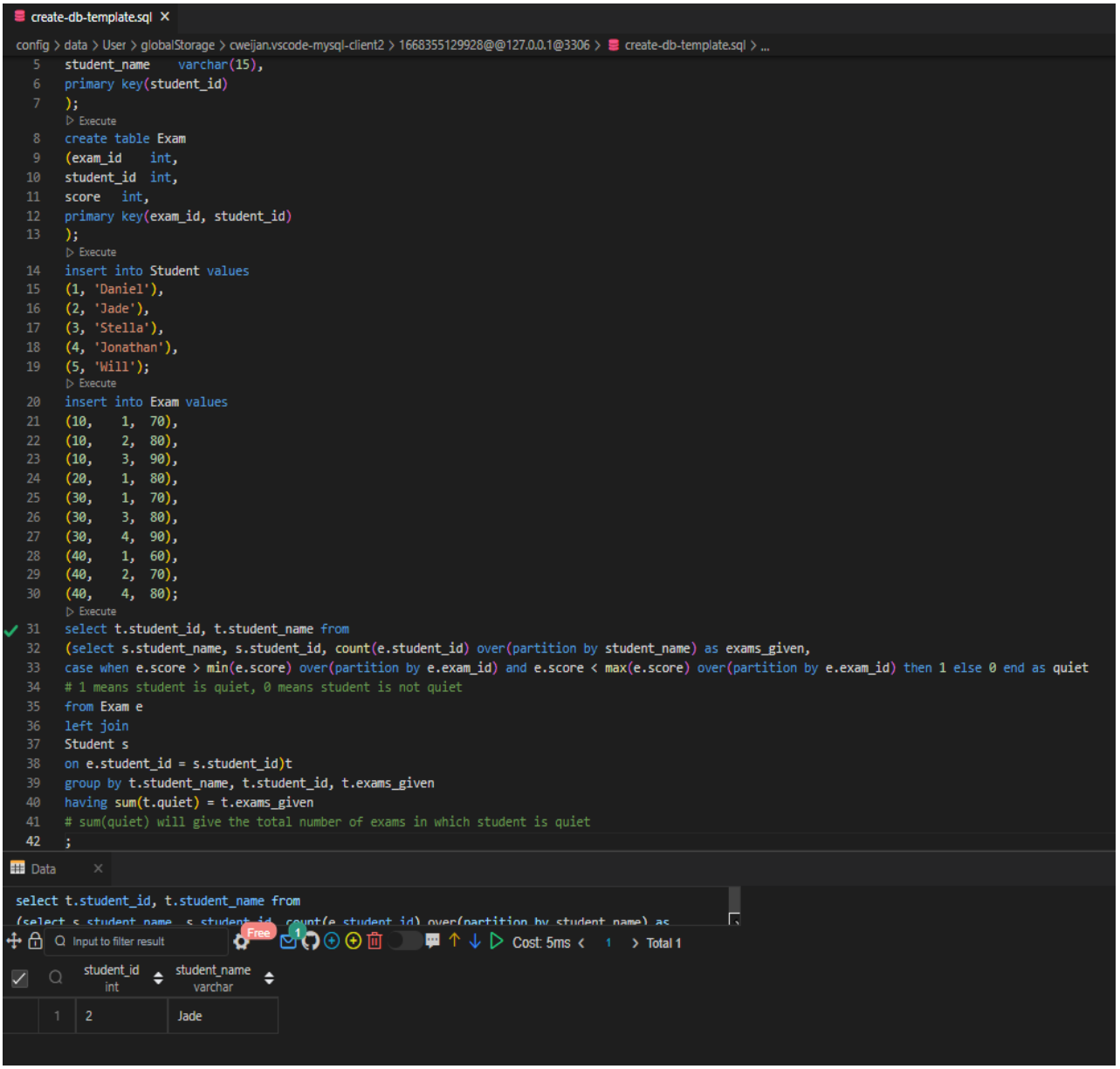
30	3	80
30	4	90
40	1	60
40	2	70
40	4	80

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 > ...
student_name varchar(15),
primary key(student_id)
);
Execute
create table Exam
(exam_id int,
student_id int,
score int,
primary key(exam_id, student_id)
);
Execute
insert into Student values
(1, 'Daniel'),
(2, 'Jade'),
(3, 'Stella'),
(4, 'Jonathan'),
(5, 'Will');
Execute
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);
Execute
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
;
Data
select t.student_id, t.student_name from
Input to filter result
Free
Cost: 5ms
Total 1
student_id int
student_name varchar
1
2
Jade

Q96.

You're given two tables on Spotify users' streaming data. songs_history table contains the historical streaming data and songs_weekly table contains the current week's streaming data.
Write a query to output the user id, song id, and cumulative count of song plays as of 4 August 2022 sorted in descending order.

Hint- Use group by

Definitions:

- song_weekly table currently holds data from 1 August 2022 to 7 August 2022.

- songs_history table currently holds data up to to 31 July 2022. The output should include the historical data in this table.

Assumption:

- There may be a new user or song in the songs_weekly table not present in the songs_history table.

songs_history Table:

Column Name	Type
history_id	Integer
user_id	Integer
song_id	Integer
song_plays	Integer

songs_history Example Input:

history_id	user_id	song_id	song_plays
10011	777	1238	11
12452	695	4520	1

song_plays: Refers to the historical count of streaming or song plays by the user.

songs_weekly Table:

Column Name	Type
user_id	Integer
song_id	Integer
listen_time	Datetime

songs_weekly Example Input:

user_id	song_id	listen_time
777	1238	08/01/2022 12:00:00
695	4520	08/04/2022 08:00:00

125	9630	08/04/2022 16:00:00
695	9852	08/07/2022 12:00:00

user_id	song_id	song_plays
777	1238	12
695	4520	2
125	9630	1

```
Solution:
select t.user_id, t.song_id, sum(t.song_plays) as song_plays
from
(
    select user_id, song_id, song_plays
from songs_history
union all
select user_id, song_id, 1 as song_plays
from songs_weekly
where date(listen_time) <= '2022/08/04') t
group by user_id, song_id;
```

Q97.

New TikTok users sign up with their emails, so each signup requires a text confirmation to activate the new user's account.
Write a query to find the confirmation rate of users who confirmed their signups with text messages. Round the result to 2 decimal places.

Hint- Use Joins

Assumptions:

- A user may fail to confirm several times with text. Once the signup is confirmed for a user, they will not be able to initiate the signup again.
- A user may not initiate the signup confirmation process at all.

`emails` Table:

Column Name	Type
email_id	Integer
user_id	Integer
signup_date	Datetime

`emails` Example Input:

email_id	user_id	signup_date
125	7771	06/14/2022 00:00:00
236	6950	07/01/2022 00:00:00
433	1052	07/09/2022 00:00:00

Texts Table:

Column Name	Type
text_id	Integer
email_id	Integer
signup_action	varchar

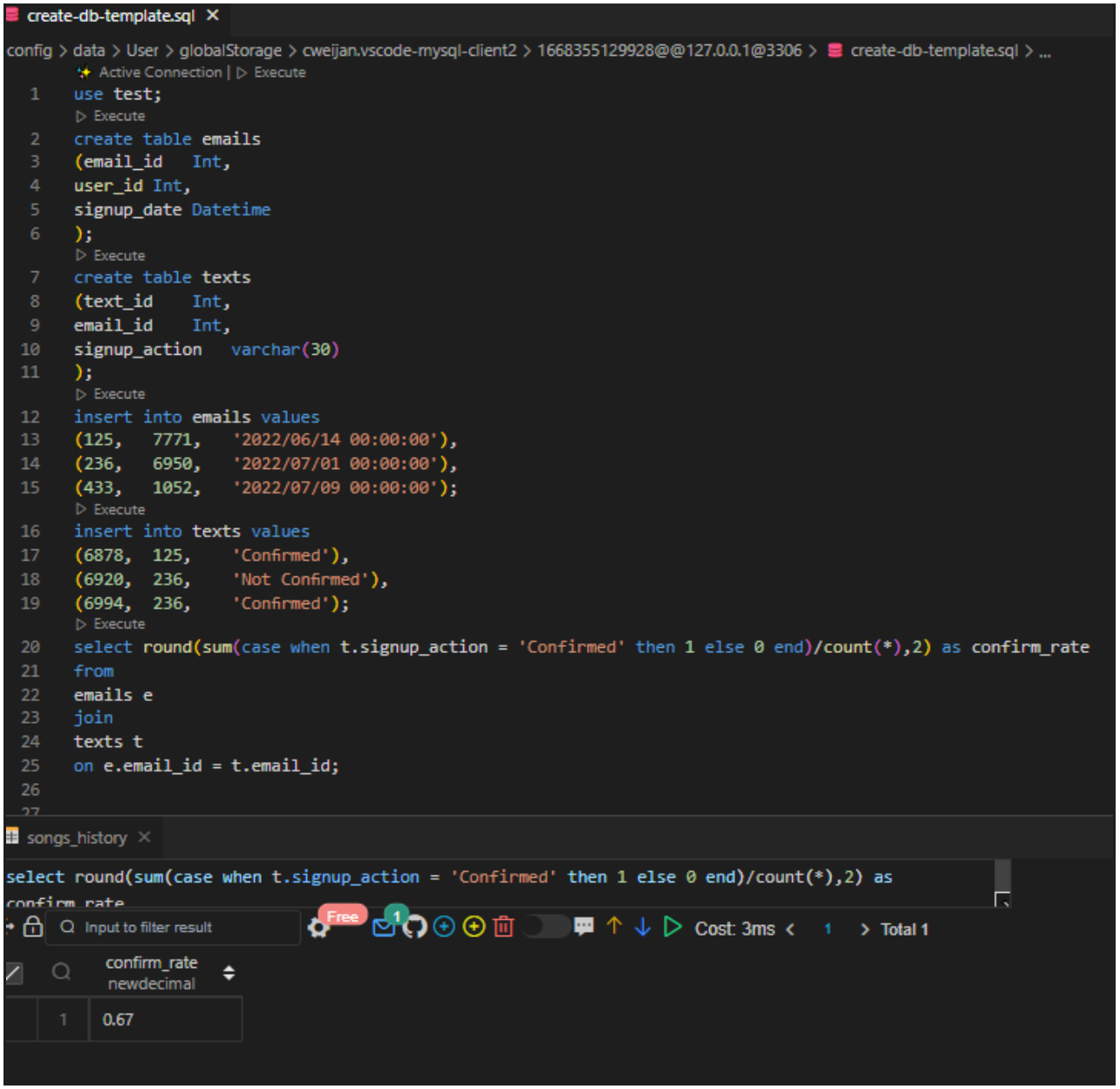
texts Example Input:

text_id	email_id	signup_action
6878	125	Confirmed
6920	236	Not Confirmed
6994	236	Confirmed

Example Output:

confirm_rate
0.67

```
Solution:
select round(sum(case when t.signup_action = 'Confirmed' then 1 else 0
end)/count(*),2) as confirm_rate
from
emails e
join
texts t
on e.email_id = t.email_id;
```



Q98.

The table below contains information about tweets over a given period of time. Calculate the 3-day rolling average of tweets published by each user for each date that a tweet was posted. Output the user id, tweet date, and rolling averages rounded to 2 decimal places.
Hint- Use Count and group by

Important Assumptions:

- Rows in this table are *consecutive* and ordered by date.
- Each row represents a different day
- A day that does not correspond to a row in this table is not counted. The most recent day is the next row above the current row.

Note: Rolling average is a metric that helps us analyze data points by creating a series of averages based on different subsets of a dataset. It is also known as a moving average, running average, moving mean, or rolling mean.

`tweets` Table:

Column Name	Type
tweet_id	Integer
user_id	Integer
tweet_date	Timestamp

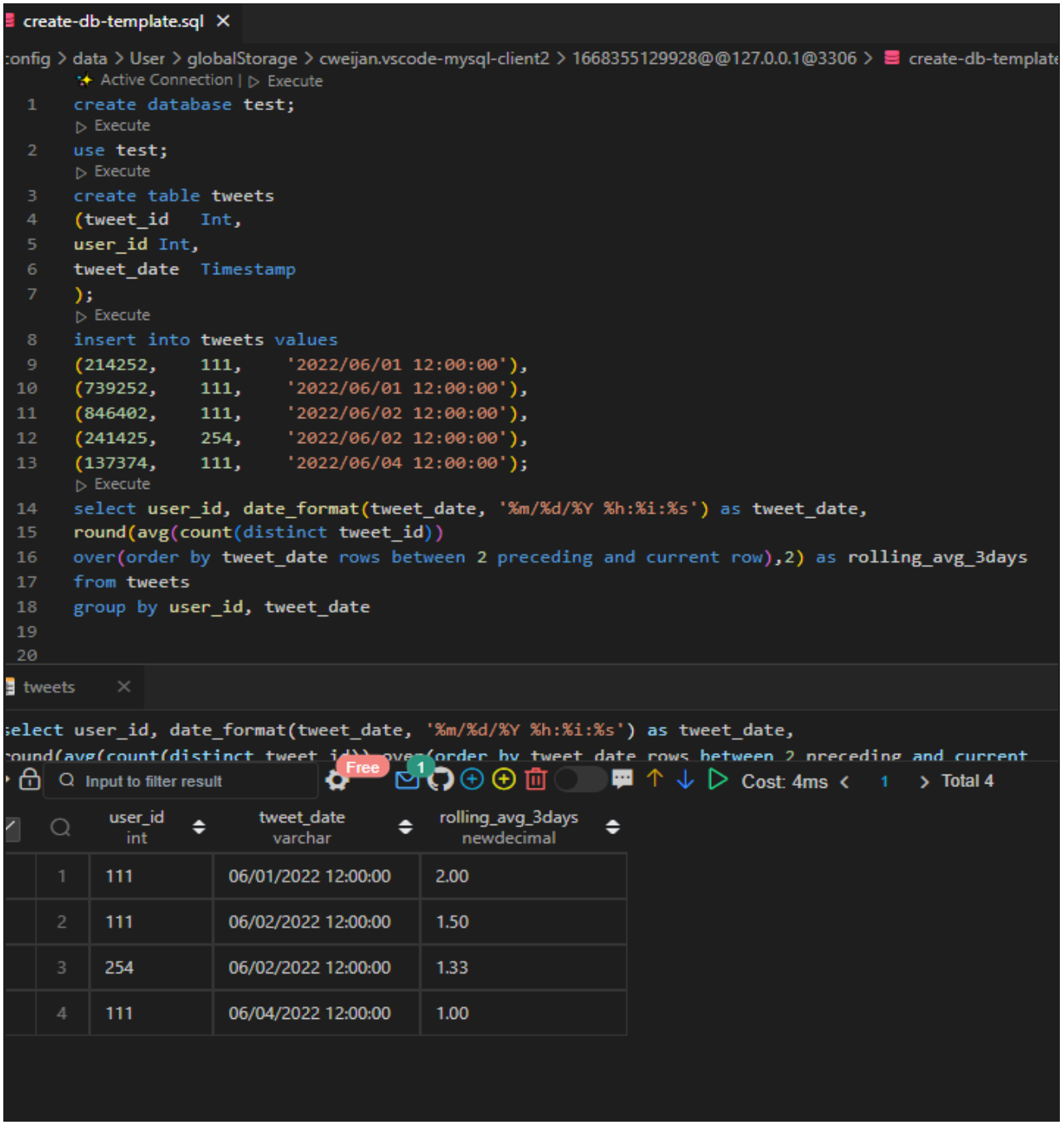
`tweets` Example Input:

tweet_id	user_id	tweet_date
214252	111	06/01/2022 12:00:00
739252	111	06/01/2022 12:00:00
846402	111	06/02/2022 12:00:00
241425	254	06/02/2022 12:00:00
137374	111	06/04/2022 12:00:00

Example Output:

user_id	tweet_date	rolling_avg_3days
111	06/01/2022 12:00:00	2.00
111	06/02/2022 12:00:00	1.50
111	06/04/2022 12:00:00	1.33
254	06/02/2022 12:00:00	1.00

```
Solution:
select user_id, date_format(tweet_date, '%m/%d/%Y %h:%i:%s') as tweet_date,
round(avg(count(distinct tweet_id)) over(order by tweet_date rows between 2
preceding and current row),2) as rolling_avg_3days
from tweets
group by user_id, tweet_date
```



Q99.

Assume you are given the tables below containing information on Snapchat users, their ages, and their time spent sending and opening snaps. Write a query to obtain a breakdown of the time spent sending vs. opening snaps (as a percentage of total time spent on these activities) for each age group.

Hint- Use join and case

Output the age bucket and percentage of sending and opening snaps. Round the percentage to 2 decimal places.

Notes:

- You should calculate these percentages:
    - time sending / (time sending + time opening)
    - time opening / (time sending + time opening)
- To avoid integer division in percentages, multiply by 100.0 and not 100.

`activities` Table:

Column Name	Type
activity_id	Integer
user_id	Integer
activity_type	string ('send', 'open', 'chat')
time_spent	float
activity_date	Datetime

`activities` Example Input:

activity_id	user_id	activity_type	time_spent	activity_date
7274	123	Open	4.50	06/22/2022 12:00:00
2425	123	Send	3.50	06/22/2022 12:00:00
1413	456	Send	5.67	06/23/2022 12:00:00
1414	789	Chat	11.00	06/25/2022 12:00:00
2536	456	Open	3.00	06/25/2022 12:00:00

`age_breakdown` Table:

Column Name	Type
user_id	Integer
age_bucket	string ('21-25', '26-30', '31-25')

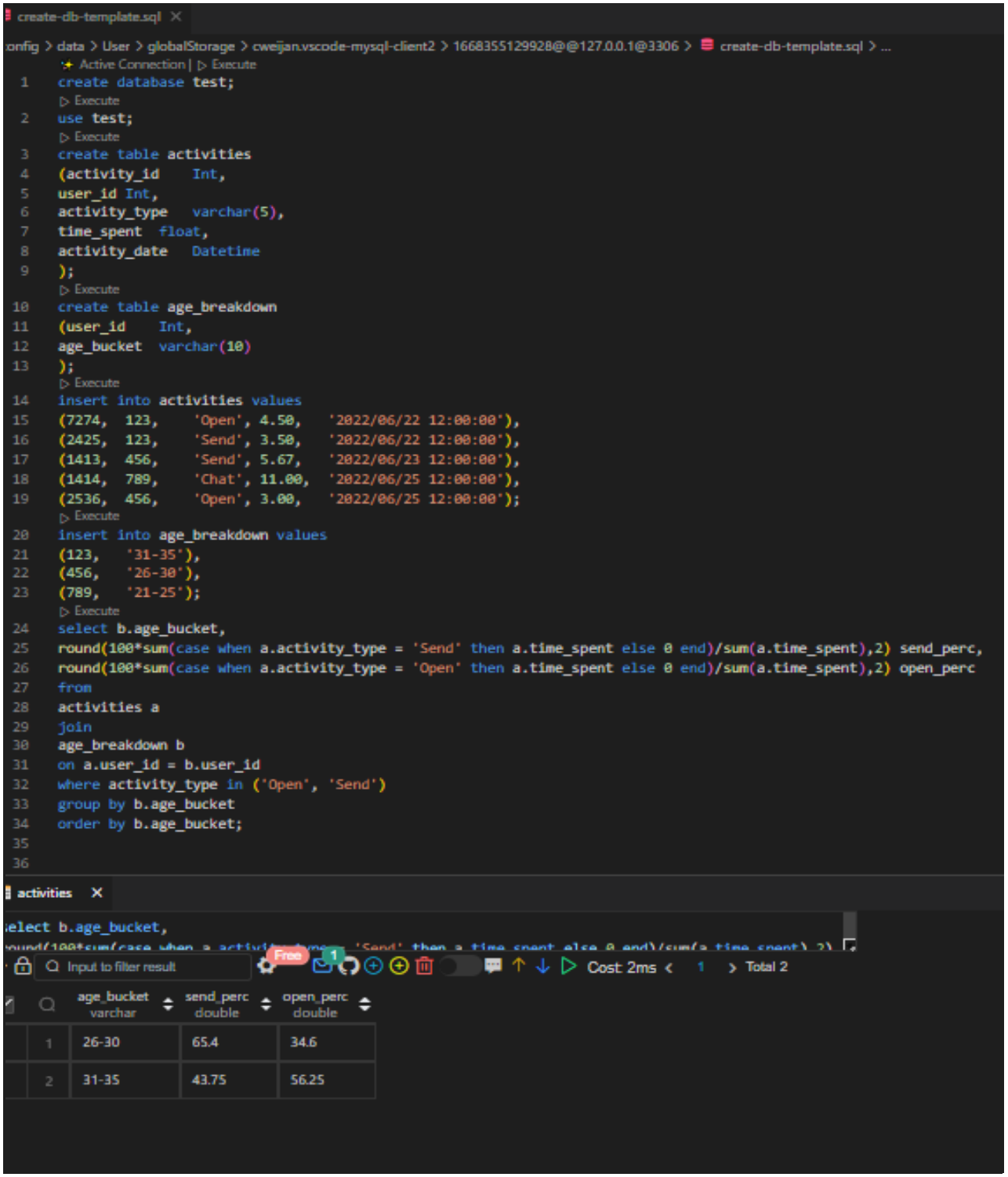
`age_breakdown` Example Input:

user_id	age_bucket
123	31-35
456	26-30
789	21-25

Example Output:

age_bucket	send_perc	open_perc
26-30	65.40	34.60
31-35	43.75	56.25

```
Solution:
select b.age_bucket,
round(100*sum(case when a.activity_type = 'Send' then a.time_spent else 0
end)/sum(a.time_spent),2) send_perc,
round(100*sum(case when a.activity_type = 'Open' then a.time_spent else 0
end)/sum(a.time_spent),2) open_perc
from
activities a
join
age_breakdown b
on a.user_id = b.user_id
where activity_type in ('Open', 'Send')
group by b.age_bucket
order by b.age_bucket;
```



Q100 .

The LinkedIn Creator team is looking for power creators who use their personal profile as a company or influencer page. This means that if someone's Linkedin page has more followers than all the companies they work for, we can safely assume that person is a Power Creator. Keep in mind that if a person works at multiple companies, we should take into account the company with the most followers.

Level - Medium
Hint- Use join and group by

Write a query to return the IDs of these LinkedIn power creators in ascending order.

Assumptions:

- A person can work at multiple companies.
- In the case of multiple companies, use the one with largest follower base.

`personal_profiles` Table:

Column Name	Type
profile_id	integer
name	string
followers	integer

`personal_profiles` Example Input:

profile_id	name	followers
1	Nick Singh	92,000
2	Zach Wilson	199,000
3	Daliana Liu	171,000
4	Ravit Jain	107,000
5	Vin Vashishta	139,000
6	Susan Wojcicki	39,000

`employee_company` Table:

Column Name	Type
personal_profile_id	integer
company_id	integer

`employee_company` Example Input:

personal_profile_id	company_id
1	4
1	9
2	2
3	1
4	3
5	6
6	5

`company_pages` Table:

Column Name	Type
company_id	integer

name	string
followers	integer

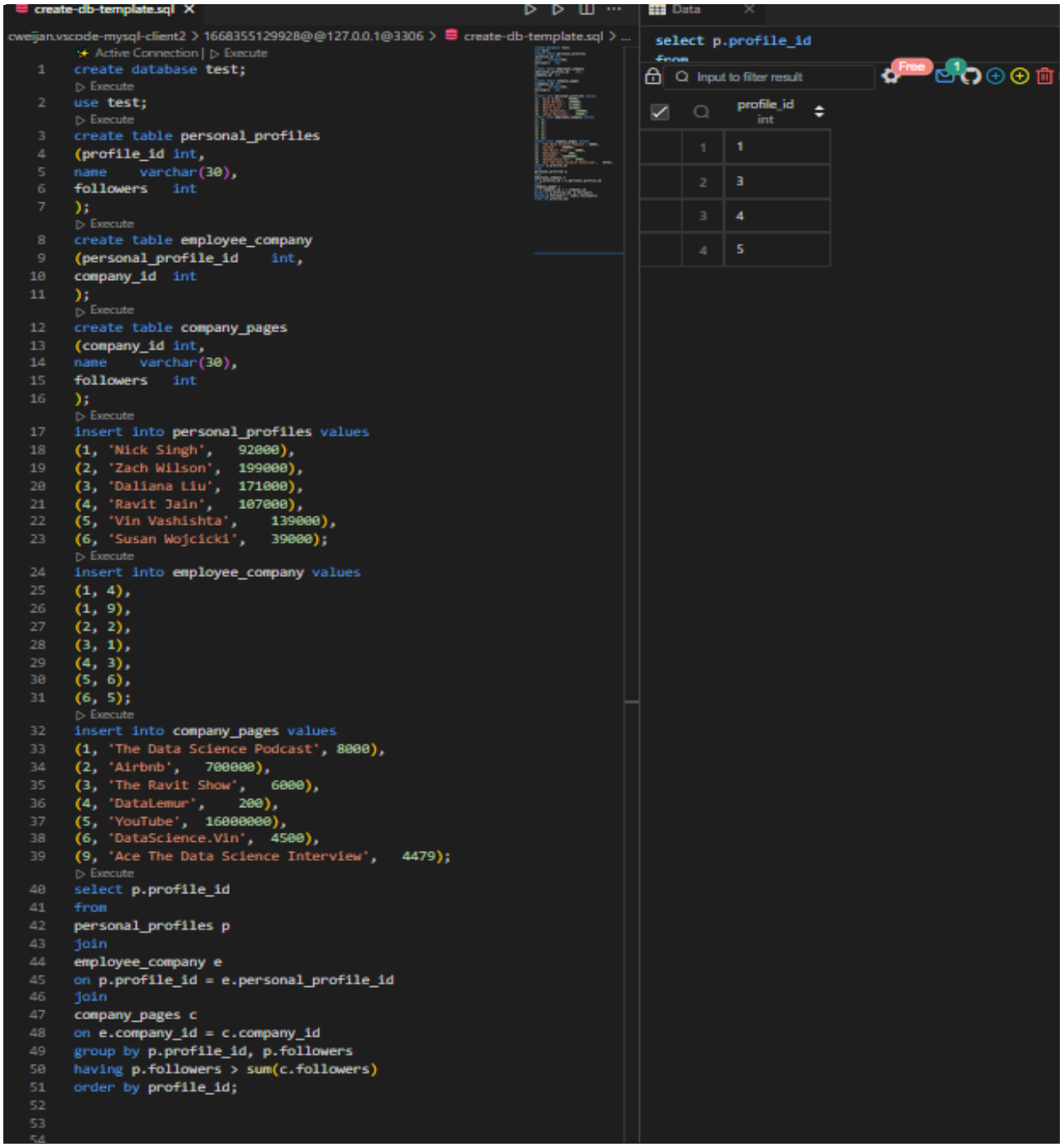
`company_pages` Example Input:

company_id	Name	followers
1	The Data Science Podcast	8,000
2	Airbnb	700,000
3	The Ravit Show	6,000
4	DataLemur	200
5	YouTube	1,6000,000
6	DataScience.Vin	4,500
9	Ace The Data Science Interview	4479

Example Output:

profile_id
1
3
4
5

```
Solution:
select p.profile_id
from
personal_profiles p
join
employee_company e
on p.profile_id = e.personal_profile_id
join
company_pages c
on e.company_id = c.company_id
group by p.profile_id, p.followers
having p.followers > sum(c.followers)
order by profile_id;
```


create-db-template.sql
cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-template.sql > ...
Active Connection | Execute
create database test;
Execute
use test;
Execute
create table personal_profiles
(profile_id int,
name varchar(30),
followers int
);
Execute
create table employee_company
(personal_profile_id int,
company_id int
);
Execute
create table company_pages
(company_id int,
name varchar(30),
followers int
);
Execute
insert into personal_profiles values
(1, 'Nick Singh', 92000),
(2, 'Zach Wilson', 199000),
(3, 'Daliana Liu', 171000),
(4, 'Ravit Jain', 107000),
(5, 'Vin Vashishta', 139000),
(6, 'Susan Wojcicki', 39000);
Execute
insert into employee_company values
(1, 4),
(1, 9),
(2, 2),
(3, 1),
(4, 3),
(5, 6),
(6, 5);
Execute
insert into company_pages values
(1, 'The Data Science Podcast', 8000),
(2, 'Airbnb', 700000),
(3, 'The Ravit Show', 6000),
(4, 'DataLemur', 200),
(5, 'YouTube', 16000000),
(6, 'DataScience.Vin', 4500),
(9, 'Ace The Data Science Interview', 4479);
Execute
select p.profile_id
from
personal_profiles p
join
employee_company e
on p.profile_id = e.personal_profile_id
join
company_pages c
on e.company_id = c.company_id
group by p.profile_id, p.followers
having p.followers > sum(c.followers)
order by profile_id;
Data
select p.profile_id
Input to filter result
Free
profile_id
int
1
2
3
4
1
3
4
5