SQL Assignment

51. Write an SQL query to report the name, population, and area of the big countries. Return the result table in any order.

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

```
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

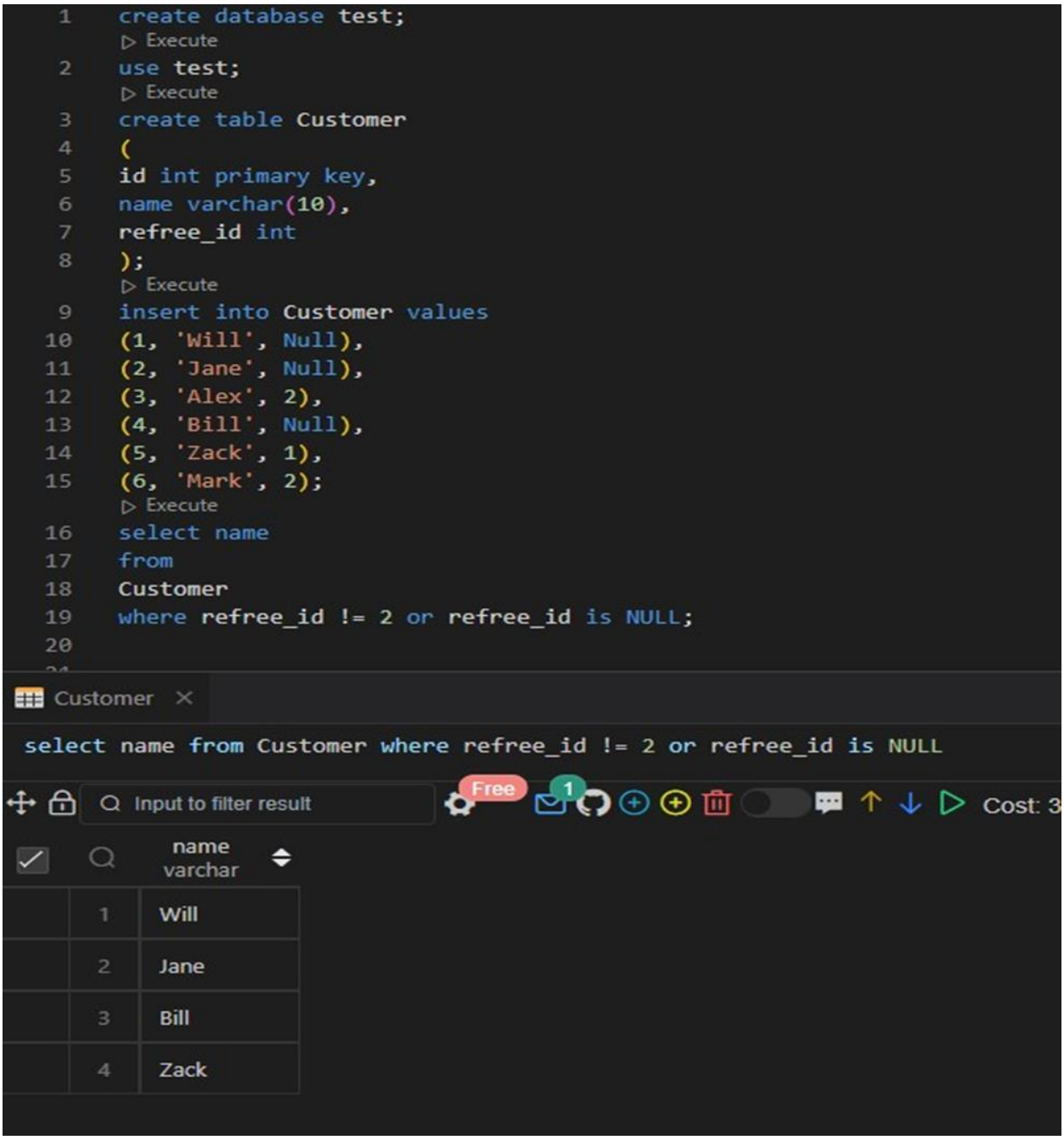
```
select name, population, area
from
```

Input to filter result | Free | Cost: 5ms | 1 | Total 2

	name varchar	population bigint	area bigint
1	Afghanistan	25500100	652230
2	Algeria	37100000	2381741

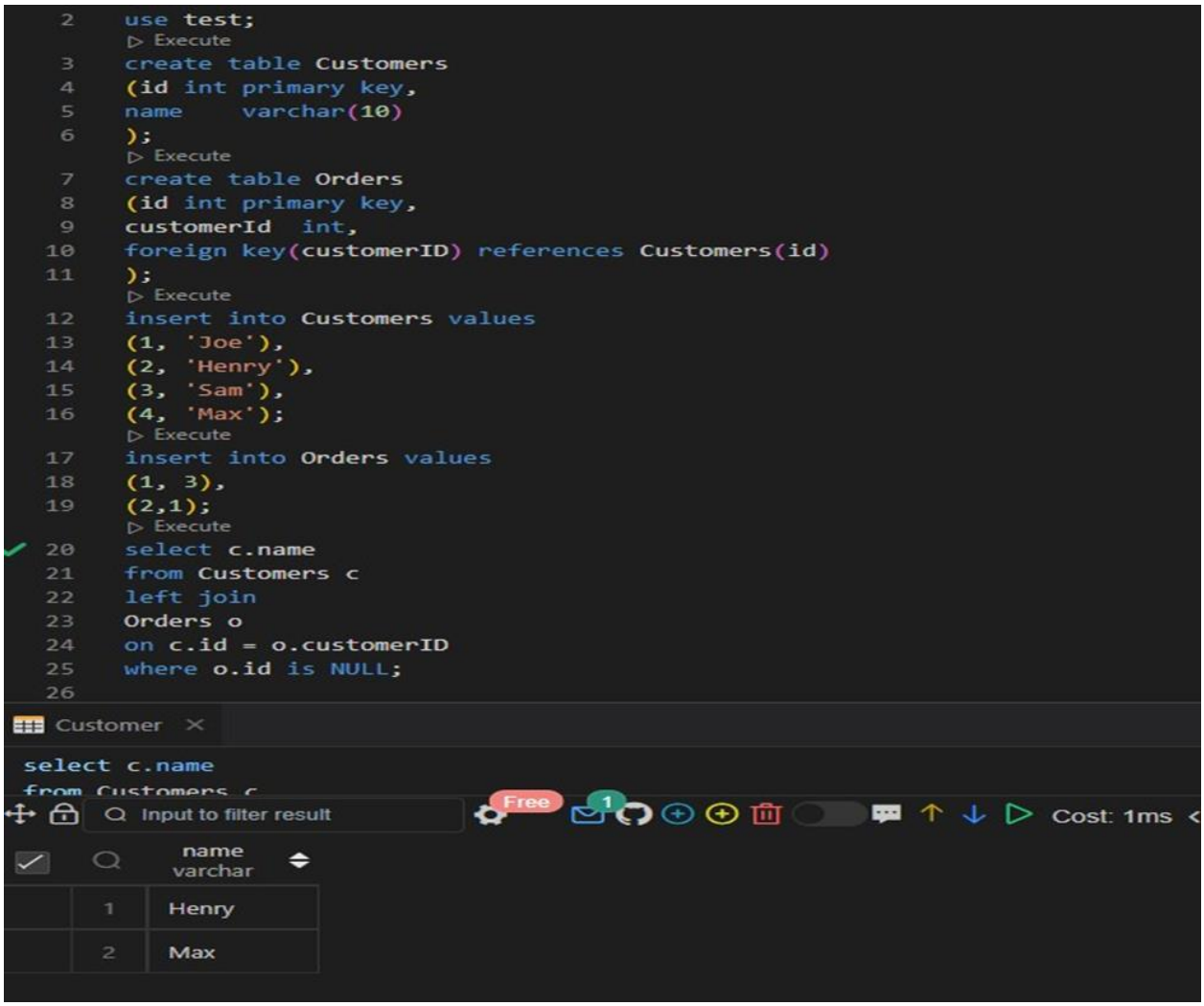
52. Write an SQL query to report the names of the customer that are not referred by the customer with id = 2. Return the result table in any order.

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



53. Write an SQL query to report all customers who never order anything.
Return the result table in any order.

Select c.name from Customers c left join Orders o on c.id = o.customerID where o.id is NULL;



54. Write an SQL query to find the team size of each of the employees.
Return result table in any order.

Select employee_id, count(team_id) over(partition by team_id) as team_size from Employee order by employee_id;

```sql
create database test;
use test;
create table Employee
(
employee_id int primary key,
team_id int
);
insert into Employee values
(1, 8),
(2, 8),
(3, 8),
(4, 7),
(5, 9),
(6, 9);
select employee_id,
count(team_id) over(partition by team_id) as team_size
from Employee
order by employee_id;
```

Employee

```sql
select employee_id,
count(team_id) over(partition by team_id) as team_size
```

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

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

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 c1.caller_id as id, c1.duration from Calls c1) union
(select c1.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;

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

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 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,  '2016-05-02',   6),
(2, 3,  '2017-06-25',   1),
(3, 1,  '2016-03-02',   0),
(3, 4,  '2018-07-03',   5);
▷ Execute
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;
```

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

57. 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.

Select
customer_number from
Orders
group by customer_number order by count(order_number) desc limit 1;

```
create database test;
▷ Execute
use test;
▷ Execute
create table Orders
(order_number    int,
customer_number int,
primary key(order_number)
);
▷ Execute
insert into Orders values
(1, 1),
(2, 2),
(3, 3),
(4, 3);
▷ Execute
select customer_number
from
Orders
group by customer_number
order by count(order_number) desc
limit 1;
```

Orders

```
select customer_number
from
```

Input to filter result

customer_number int	
1	3

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.

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;

58. 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.

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;

```
1   create database test;
    ▷ Execute
2   use test;
    ▷ Execute
3   create table Cinema
4   (seat_id int AUTO_INCREMENT PRIMARY KEY,
5   Free boolean
6   );
    ▷ Execute
7   insert into Cinema values
8   (1, 1),
9   (2, 0),
10  (3, 1),
11  (4, 1),
12  (5, 1);
    ▷ Execute
13  select t.seat_id
14  from
15  (select seat_id, lead(seat_id,1,seat_id) over(order by seat_id) as next
16  from Cinema
17  where Free != 0
18  ) t
19  where next - seat_id in (0,1)
20  order by seat_id;
21
```

Cinema

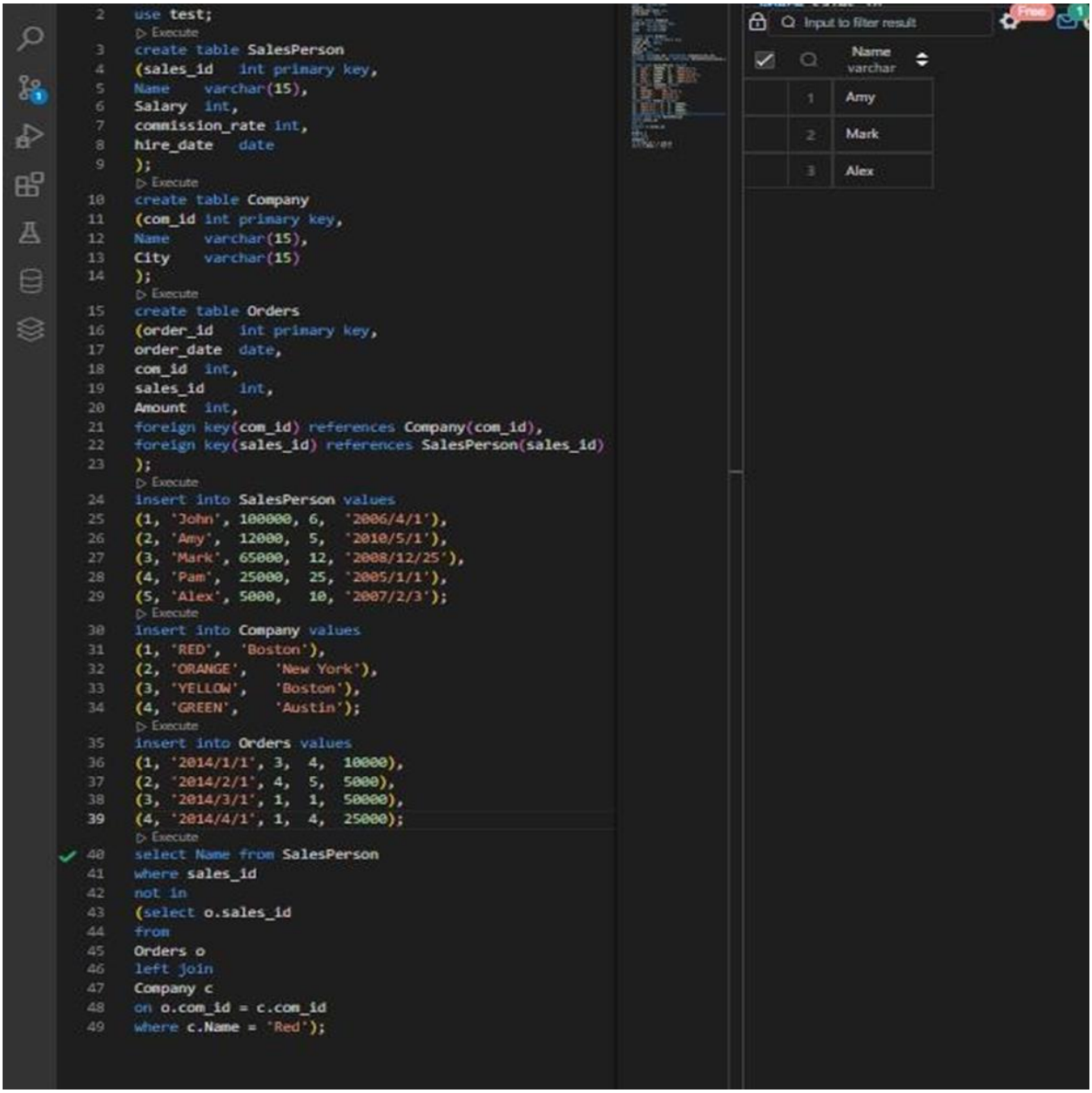
```
select t.seat_id
from
```

Input to filter result | Free | Cost: 10ms | 1 | Total

	seat_id int
1	3
2	4
3	5

59. 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.

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');



```
use test;
▷ Execute
create table SalesPerson
(sales_id   int primary key,
Name    varchar(15),
Salary  int,
commission_rate int,
hire_date   date
);
▷ Execute
create table Company
(com_id int primary key,
Name    varchar(15),
City    varchar(15)
);
▷ Execute
create table Orders
(order_id   int primary key,
order_date  date,
com_id  int,
sales_id    int,
Amount  int,
foreign key(com_id) references Company(com_id),
foreign key(sales_id) references SalesPerson(sales_id)
);
▷ Execute
insert into SalesPerson values
(1, 'John', 100000, 6,  '2006/4/1'),
(2, 'Amy',  12000,  5,  '2010/5/1'),
(3, 'Mark', 65000,  12, '2008/12/25'),
(4, 'Pam',  25000,  25, '2005/1/1'),
(5, 'Alex', 5000,   10, '2007/2/3');
▷ Execute
insert into Company values
(1, 'RED',  'Boston'),
(2, 'ORANGE',   'New York'),
(3, 'YELLOW',   'Boston'),
(4, 'GREEN',    'Austin');
▷ Execute
insert into Orders values
(1, '2014/1/1', 3,  4,  10000),
(2, '2014/2/1', 4,  5,  5000),
(3, '2014/3/1', 1,  1,  50000),
(4, '2014/4/1', 1,  4,  25000);
▷ Execute
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');
```

Input to filter result

	Name varchar
1	Amy
2	Mark
3	Alex

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

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;

```
 8   create table Triangle
 9   (X  int,
10   Y   int,
11   Z   int,
12   PRIMARY KEY(X,Y,Z)
13   );
     ▷ Execute
14   insert into Triangle values
15   (13,    15, 30),
16   (10,    20, 15);
     ▷ Execute
17   select X, Y, Z, (case
18   when X+Y > Z and Y+Z > X and Z+X > Y then 'Yes'
19   else 'No'
20   end) as triangle
21   from Triangle;
22
```

Triangle

```
select X, Y, Z, (case
when X+Y > Z and Y+Z > X and Z+X > Y then 'Yes'
```

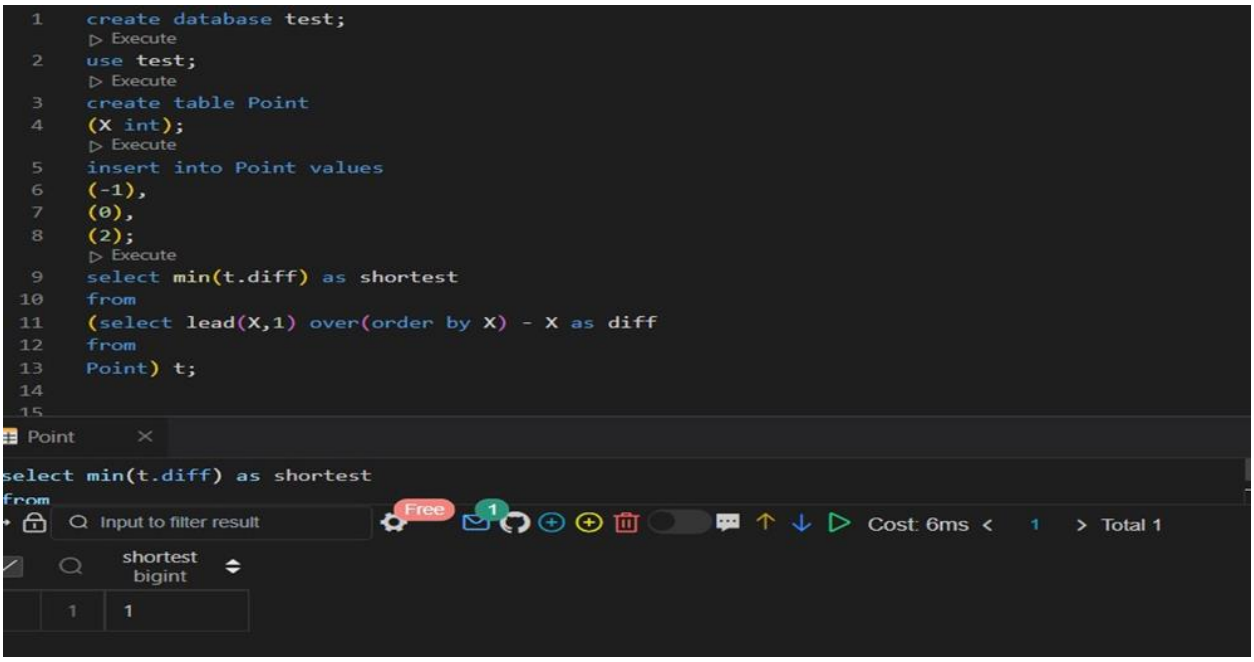
Input to filter result

Free 1

	X int	Y int	Z int	triangle varchar
1	13	15	30	No
2	10	20	15	Yes

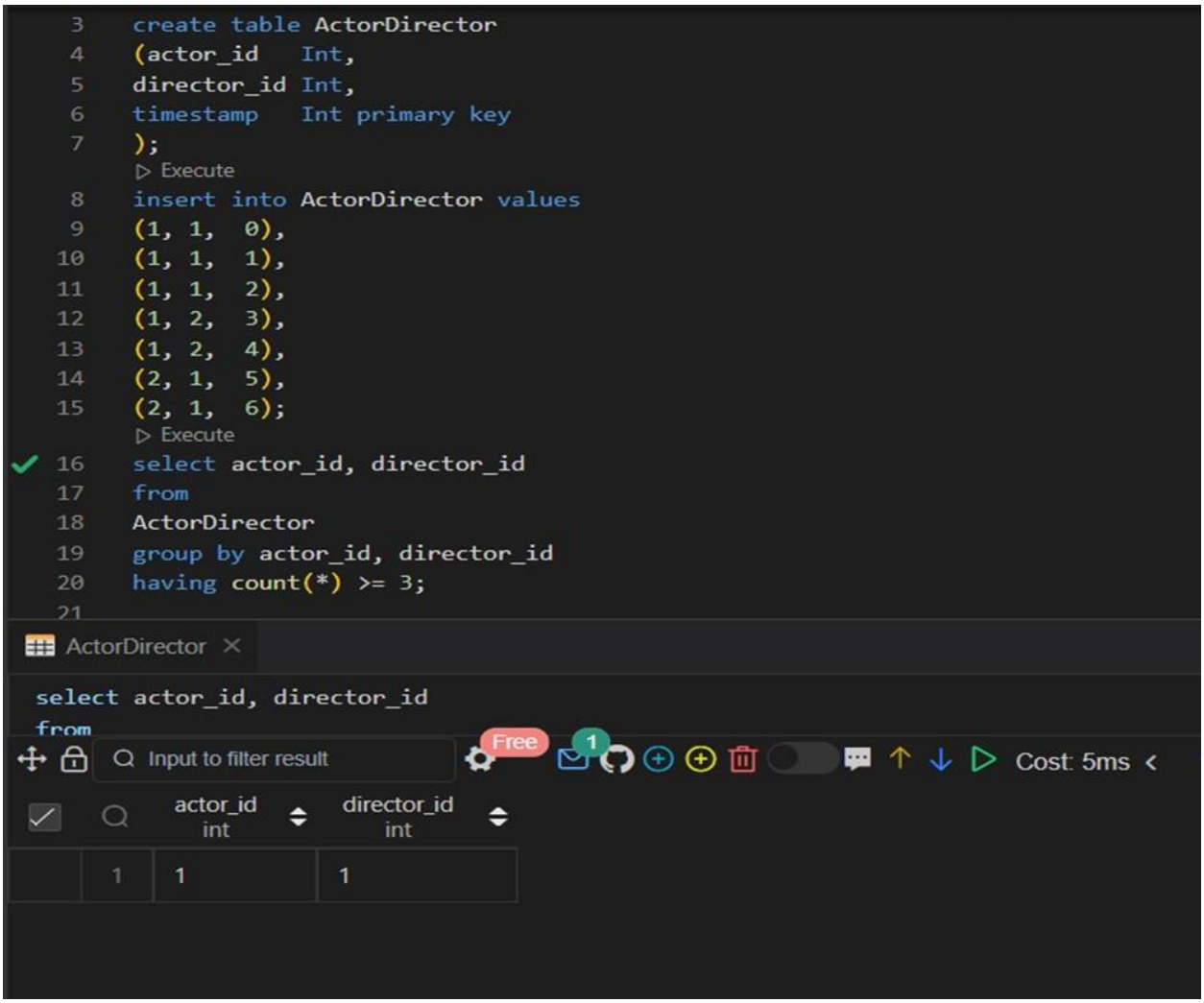
61. Write an SQL query to report the shortest distance between any two points from the Point table. The query result format is in the following example.

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



62. 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.

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



63. 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.

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;

```
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

64. 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.

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;

```sql
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

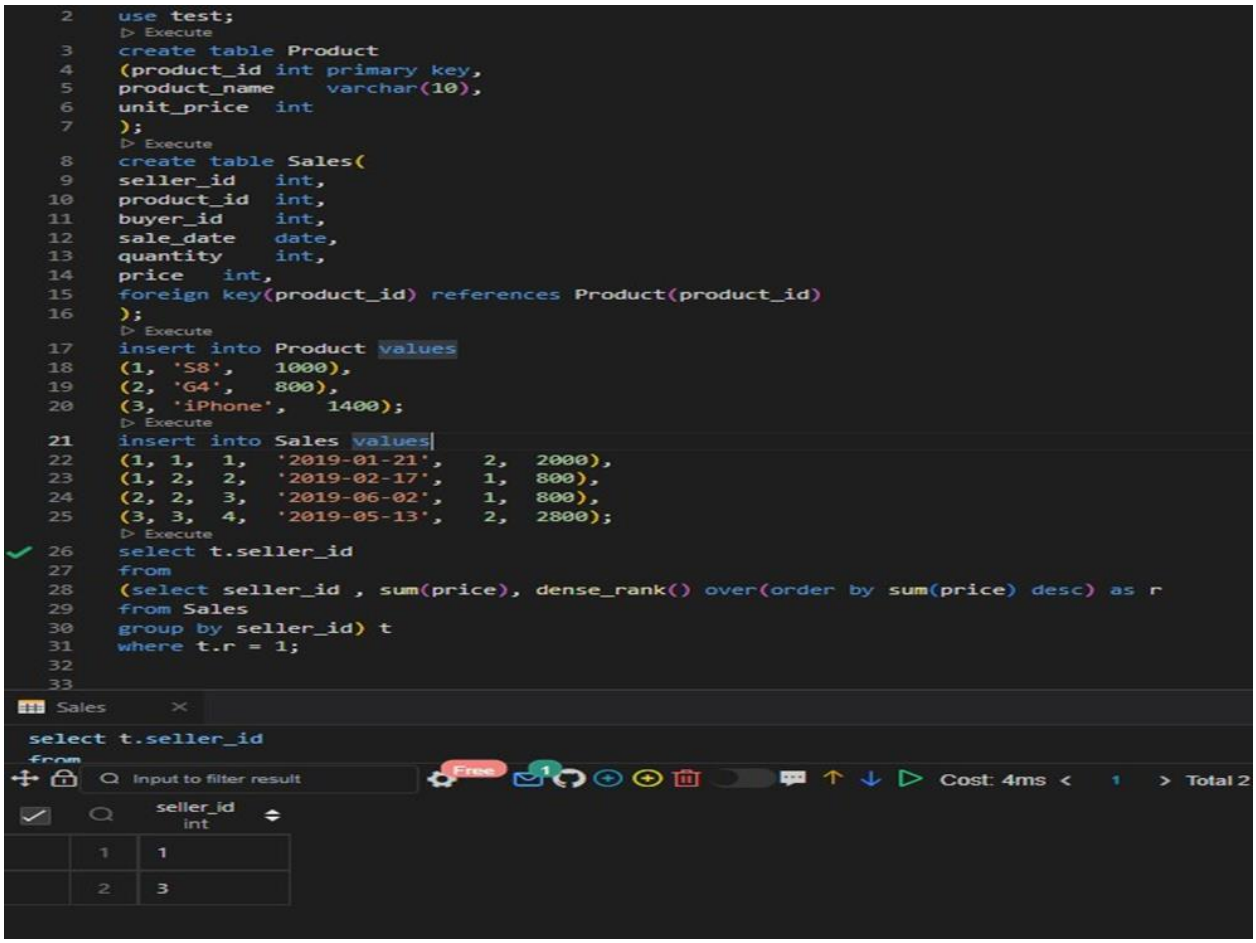
```sql
select p.project_id, round(avg(e.experience_years),2) as average_years
from
```

Input to filter result | Free | Cost: 2ms | Total 2

	project_id int	average_years newdecimal
1	1	2.00
2	2	2.50

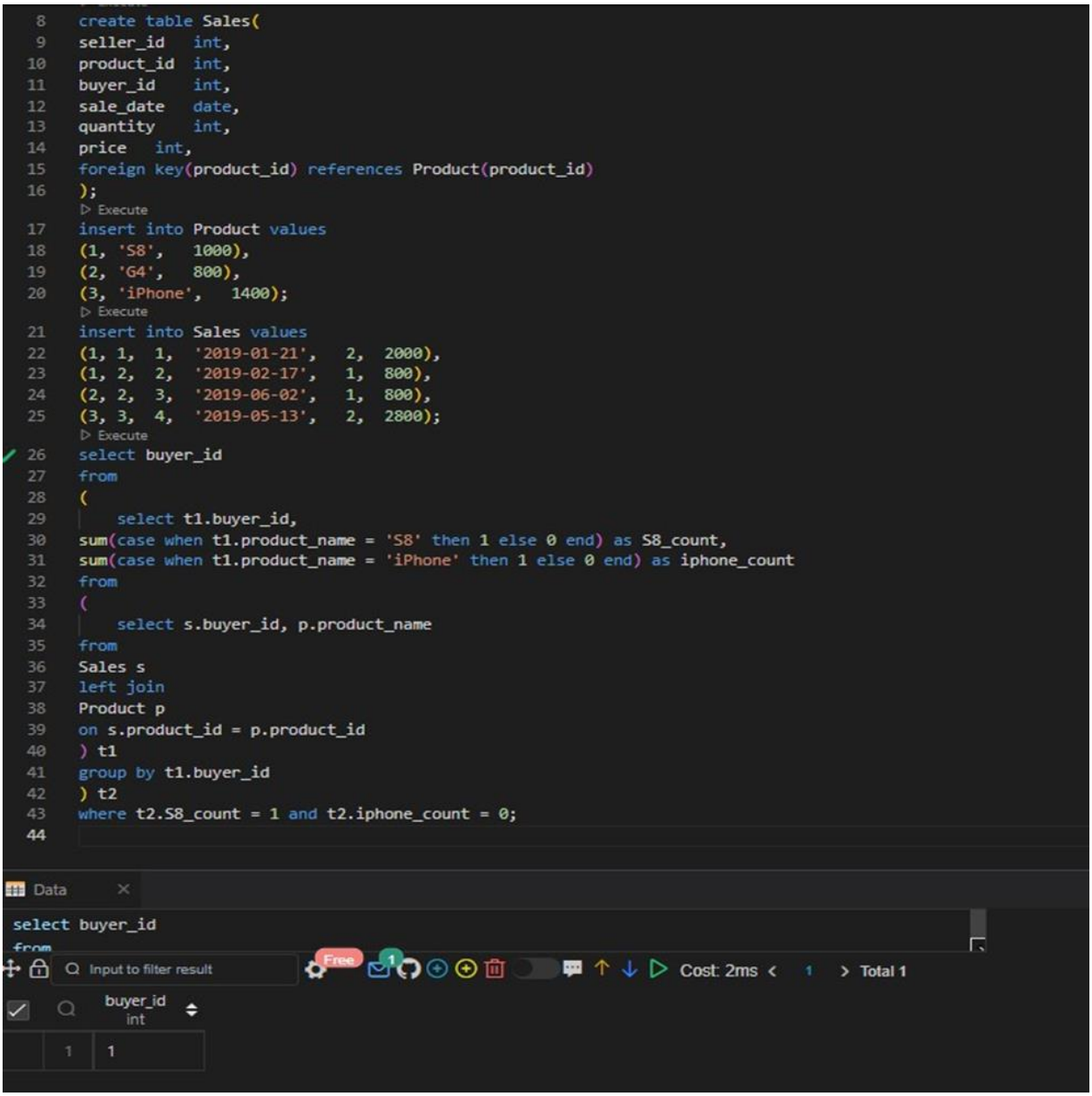
65. Write an SQL query that reports the best seller by total sales price, If there is a tie, report them all. Return the result table in any order.

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;



66. 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.

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 table Sales(
seller_id   int,
product_id  int,
buyer_id    int,
sale_date   date,
quantity    int,
price   int,
foreign key(product_id) references Product(product_id)
);
▷ Execute
insert into Product values
(1, 'S8',   1000),
(2, 'G4',   800),
(3, 'iPhone',   1400);
▷ Execute
insert into Sales values
(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);
▷ Execute
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;
```

Data

```
select buyer_id
from
```

Input to filter result | Free | Cost: 2ms | Total 1

buyer_id int
1

67. 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.

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)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 table Customer
(customer_id    int,
name    varchar(15),
visited_on  date,
amount  int,
primary key(customer_id, visited_on)
);
▷ Execute
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
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

```
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

68. 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.

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

```
create table Scores
(
player_name varchar(15),
gender  varchar(2),
day date,
score_points    Int,
primary key(gender, day)
);
▷ Execute
insert into Scores values
('Aron',    'F',    '2020-01-01',   17),
('Alice',   'F',    '2020-01-07',   23),
('Bajrang', 'M',    '2020-01-07',   7),
('Khali',   'M',    '2019-12-25',   11),
('Slaman',  'M',    '2019-12-30',   13),
('Joe',     'M',    '2019-12-31',   3),
('Jose',    'M',    '2019-12-18',   2),
('Priya',   'F',    '2019-12-31',   23),
('Priyanka','F',    '2019-12-30',   17);
▷ Execute
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;
```

Scores

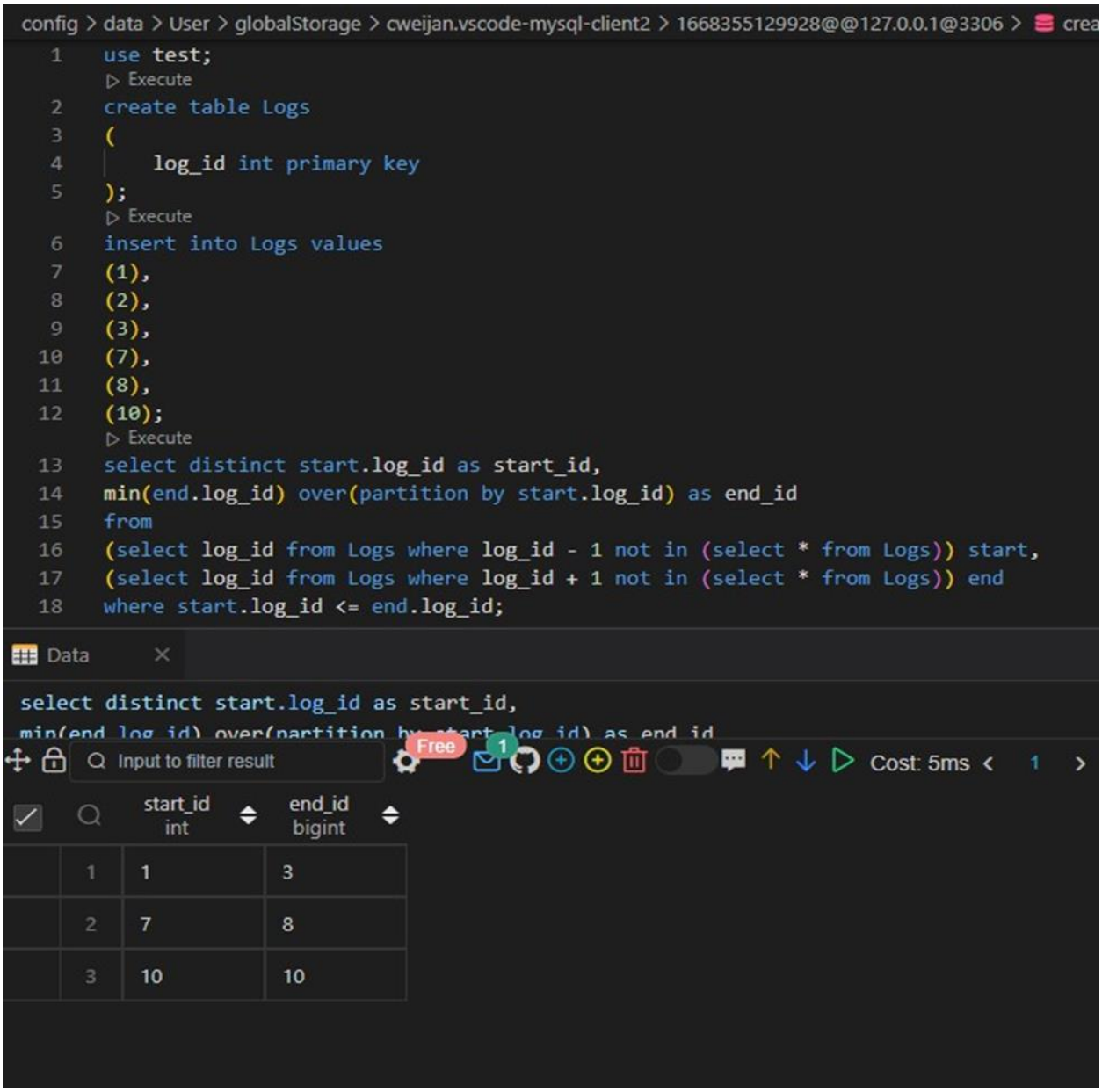
```
select gender, day, sum(score_points) over(partition by gender order by day) as total
from Scores
```

Input to filter result — Cost: 4ms — Total 9

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

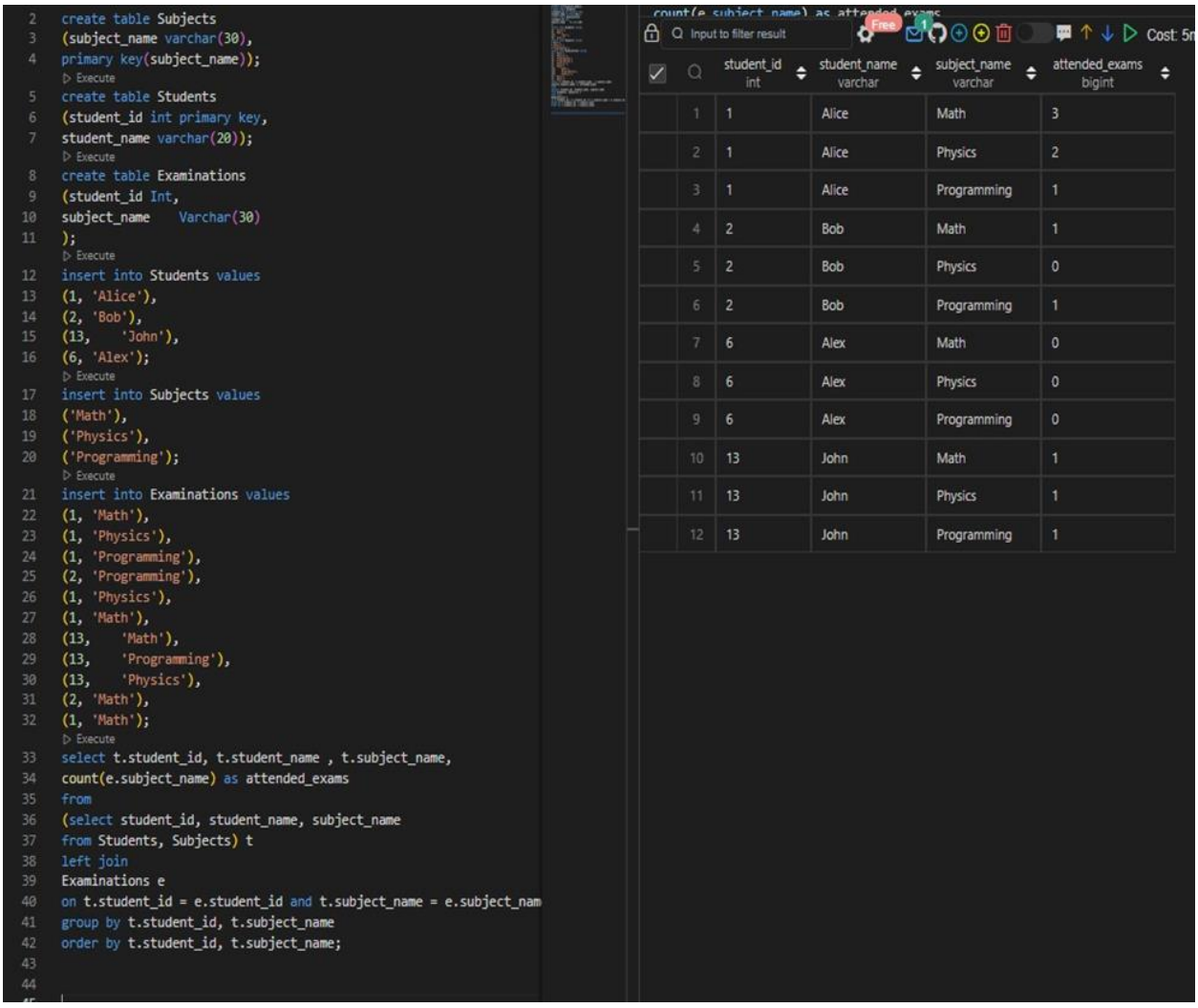
69. Write an SQL query to find the start and end number of continuous ranges in the table Logs. Return the result table ordered by start_id.

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;



70. 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.

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;



```sql
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;
```

Input to filter result

	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

71. 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.

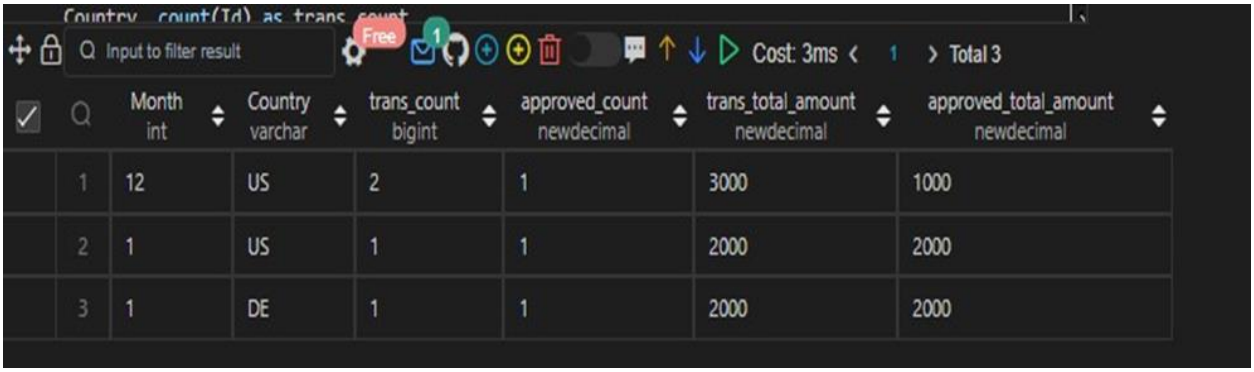
```sql
WITH RECURSIVE emp_hir AS
(
  SELECT
    employee_id,
    manager_id,
    employee_name,
    1 as lvl
  FROM
    employees
  WHERE
    employee_name = 'Boss'
```

```
  UNION
  SELECT
    em.employee_id,
    em.manager_id,
    em.employee_name,
    eh.lvl + 1 as lvl
  FROM
    emp_hir eh
    JOIN employees em ON eh.employee_id = em.manager_id
  WHERE
    em.employee_name <> 'Boss'
)
```

72. 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.

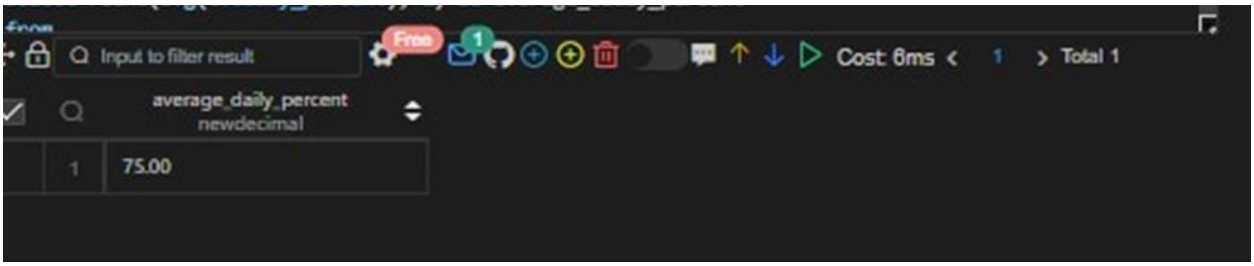
```
SELECT
  DATE_FORMAT(trans_date, '%Y-%m') AS month,
  country,
  COUNT(*) AS trans_count,
  COUNT(
    CASE
      WHEN state = 'approved'
        THEN id
    END
  ) AS approved_count,
  SUM(amount) AS trans_total_amount,
  SUM(
    CASE
      WHEN state = 'approved'
        THEN amount
    END
  ) AS approved_total_amount
FROM
  transactions
GROUP BY
  DATE_FORMAT(trans_date, '%Y-%m'),
```

country
;



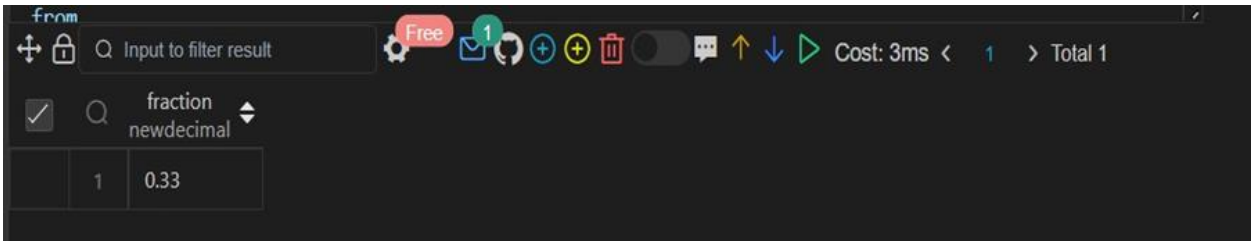
73. Write an SQL query to find the average daily percentage of posts that got removed after being reported as spam, rounded to 2 decimal places.

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;



74. 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.

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;



75. 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.

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;

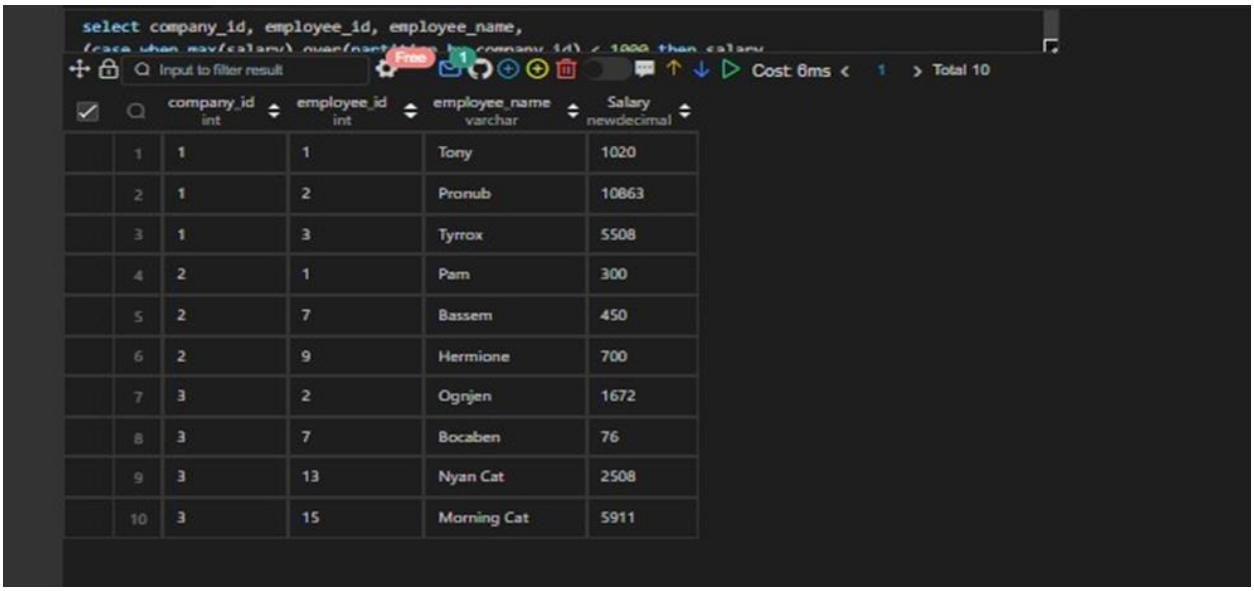
76.Write an SQL query to find the salaries of the employees after applying taxes. Round the salary to the nearest integer.

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.

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;



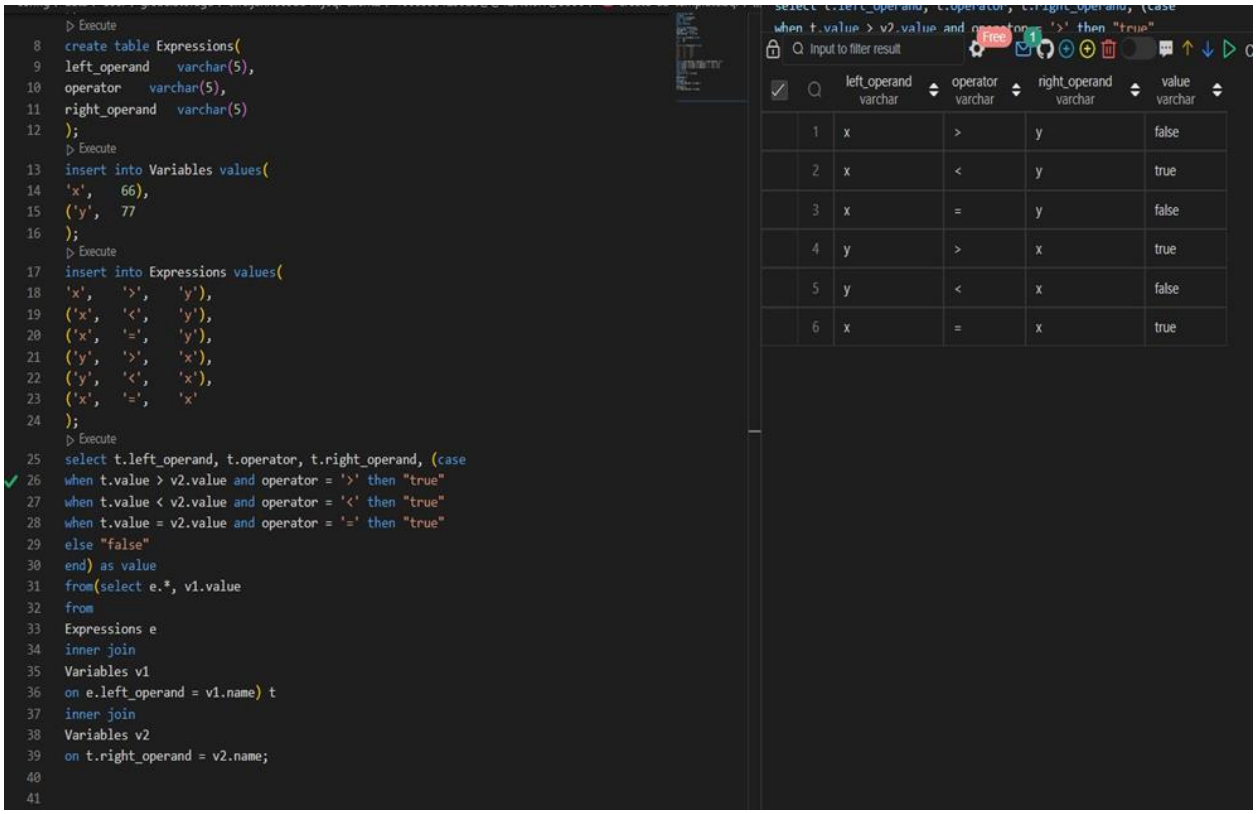
select company_id, employee_id, employee_name,

Input to filter result

Cost: 6ms < 1 > 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

77. Write an SQL query to evaluate the boolean expressions in the Expressions table. Return the result table in any order.



```
create table Expressions(
left_operand    varchar(5),
operator    varchar(5),
right_operand   varchar(5)
);
insert into Variables values(
'x',    66),
('y',   77
);
insert into Expressions values(
'x',    '>',    'y'),
('x',   '<',    'y'),
('x',   '=',    'y'),
('y',   '>',    'x'),
('y',   '<',    'x'),
('x',   '=',    'x'
);
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;
```

Input to filter result

	left_operand varchar	operator varchar	right_operand varchar	value varchar
1	x	>	y	false
2	x	<	y	true
3	x	=	y	false
4	y	>	x	true
5	y	<	x	false
6	x	=	x	true

78. 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.

```
WITH receiver_caller_calls AS(
  SELECT
    caller_id AS caller_receiver_id,
    duration
  FROM
    calls
  UNION ALL
  SELECT
    callee_id AS caller_receiver_id,
    duration
  FROM
    calls
),
call_duration_avg AS(
  SELECT
    DISTINCT cn.name,
    avg(c.duration) OVER() as global_average,
    avg(c.duration) OVER(PARTITION BY cn.name) as country_average
  FROM
    person p
    JOIN country cn
      ON  CAST(SUBSTRING_INDEX(p.phone_number, '-', 1) AS UNSIGNED) =
CAST(cn.country_code AS UNSIGNED)
    JOIN receiver_caller_calls c
      ON c.caller_receiver_id = p.id
)
```

79. 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

```
SELECT  name FROM employee ORDER BY name;
```

```
3   create table Employee
4   (employee_id int,
5   name varchar(12),
6   months int,
7   salary int);
    ▷ Execute
8   insert into Employee values
9   (12228, 'Rose', 15, 1968),
10  (33645, 'Angela', 1, 3443),
11  (45692, 'Frank', 17, 1608),
12  (56118, 'Patrick', 7, 1345),
13  (59725, 'Lisa', 11, 2330),
14  (74197, 'Kimberly', 16, 4372),
15  (78454, 'Bonnie', 8, 1771),
16  (83565, 'Michael', 6, 2017),
17  (98607, 'Todd', 5, 3396),
18  (99989, 'Joe', 9, 3573);
    ▷ Execute
19  select name
20  from |
21  Employee
22  order by name;
23
```

Employee

```
select name
from
```

Input to filter result | Free | Cost: 5ms

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

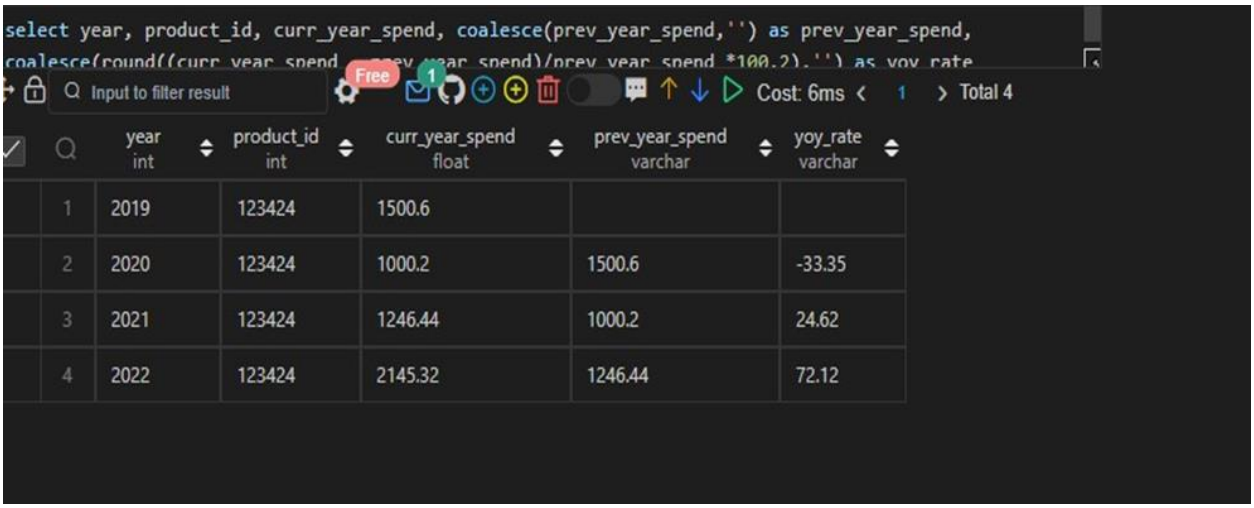
80. Assume you are given the table below containing information on user transactions for particular products. Write a query to obtain the year-on-year growth rate for the total spend of each product for each year.

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

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;



81. Amazon wants to maximize 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.

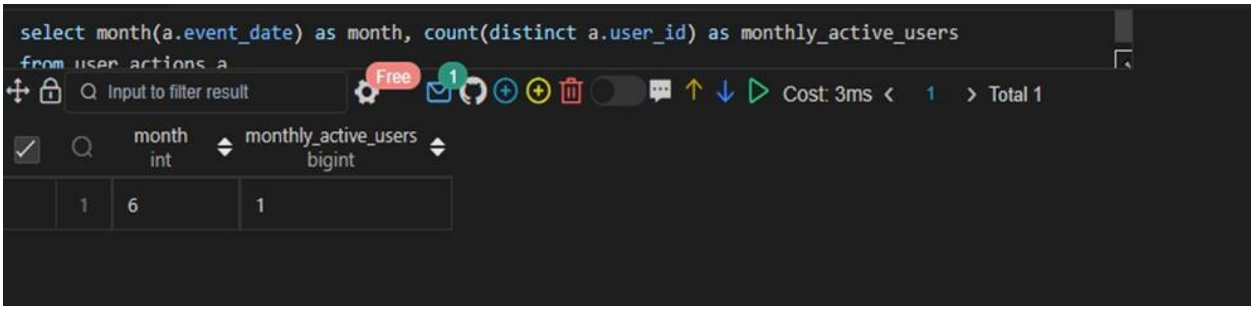
select item_type, (case when item_type = 'prime_eligible' then floor(500000/sum(square_footage)) * count(item_type) when item_type = 'not_prime' then floor((500000 -(select floor(500000/sum(square_footage)) * sum(square_footage) from inventory where item_type = 'prime_eligible'))/sum(square_footage)) * count(item_type) end) as item_count from inventory group by item_type order by count(item_type) desc;

82. 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

```
SELECT
  CAST(DATE_FORMAT(curr_month_ua.event_date, '%m') AS UNSIGNED) AS month,
  count(distinct curr_month_ua.user_id) AS monthly_active_users
FROM
  user_actions curr_month_ua
WHERE
  curr_month_ua.event_type IN ('sign-in', 'like', 'comment')
  AND DATE_FORMAT(curr_month_ua.event_date,'%Y-%m') = '2022-06'
  AND EXISTS(
    SELECT
      *
    FROM
      user_actions last_month_ua
    WHERE
      curr_month_ua.user_id = last_month_ua.user_id
      AND last_month_ua.event_type IN ('sign-in', 'like', 'comment')
      AND DATE_FORMAT(curr_month_ua.event_date, '%Y-%m') =
        DATE_FORMAT(last_month_ua.event_date + INTERVAL '1' MONTH, '%Y-%m')
  )
GROUP BY
  CAST(DATE_FORMAT(curr_month_ua.event_date, '%m') AS UNSIGNED)
;
```



83. 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.

```
 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);
```

84. 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.

```
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;

85. 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-

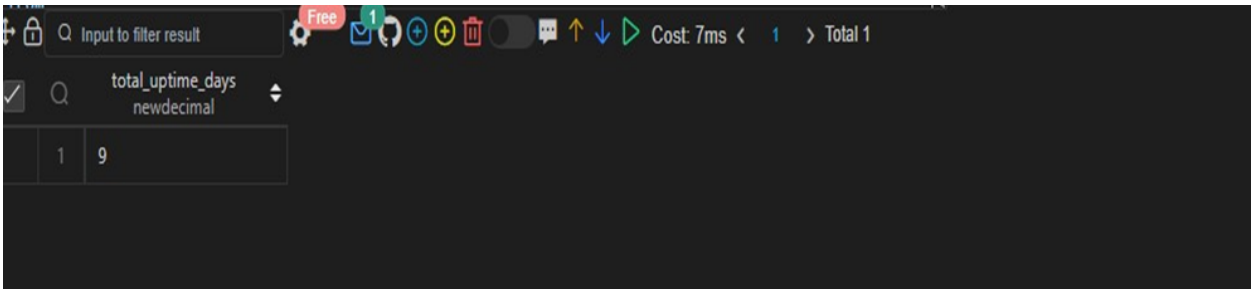
1. Calculate individual uptimes

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.

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;



86. 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.

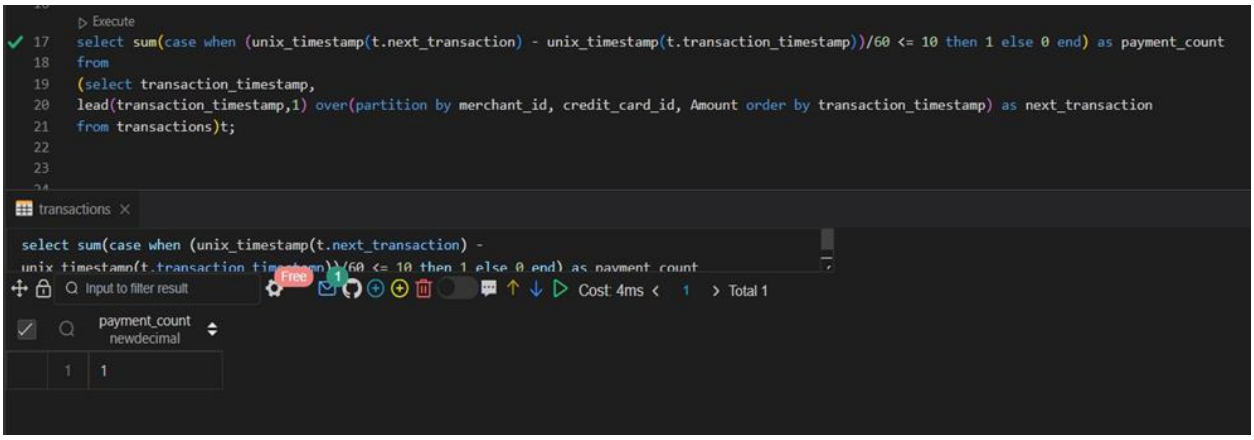
Using the transactions table, identify any payments made at the same merchant with the same credit card for the same amount within 10 minutes of each other. Count such repeated payments.

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.

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;



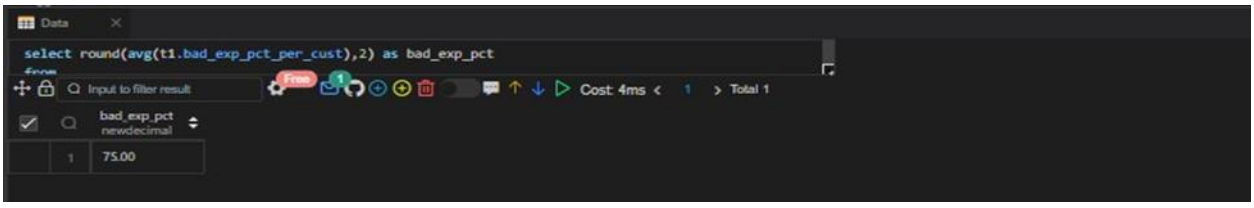
87. DoorDash's Growth Team is trying to make sure new users (those who are making orders in their first 14 days) have a great experience on all their orders in their 2 weeks on the platform. Unfortunately, many deliveries are being messed up because:

- 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.

```
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;
```



88. 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.

```
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;

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

Input to filter result | Cost: 4ms | Total 9

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

89. 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.

select t3.Name
from
(
select t2.Name, avg(t1.duration) over(partition by t2.Name) as
avg_call_duration, avg(t1.duration) over() as global_average from
((select cl.caller_id as id, cl.duration
from Calls cl) union
(select cl.callee_id as id, cl.duration
from Calls cl)) t1 left join
(select p.id, c.Name from Person
p left JOIN Country c
ON cast(left(p.phone_number,3) as int) = cast(c.country_code as int)) t2
ON t1.id = t2.id) t3
where t3.avg_call_duration > global_average
group by t3.Name;

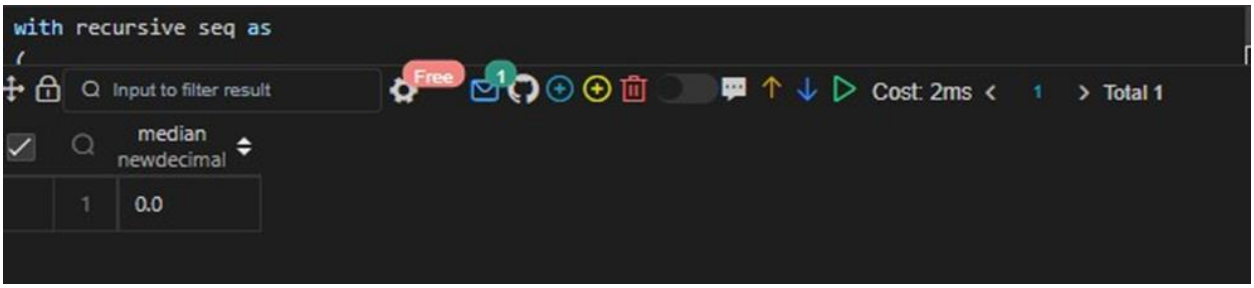
	Name
1	Peru

90. 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.

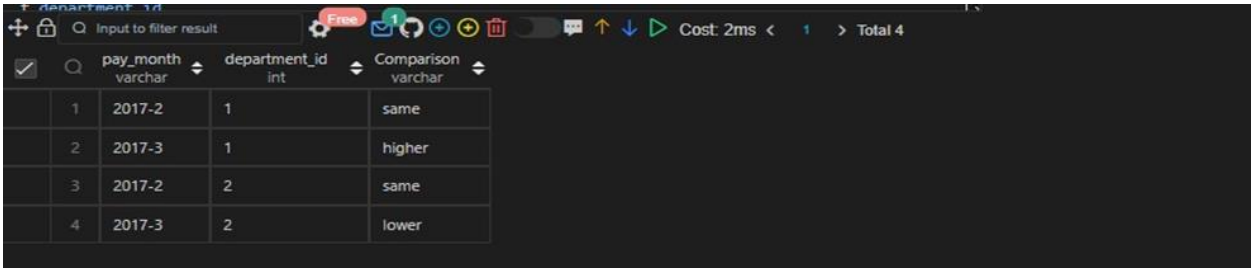
```
 with recursive seq as
(       select num, frequency, 1 as c from Numbers      union
        select num, frequency, c+1 from seq where c < frequency
) select round(avg(t.num),1) as median  from (    select num,row_number()
over(order by num, c) as r1,          row_number() over(order by num desc,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



91. 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.

```
WITH department_company_avg_monthly AS(
  SELECT
    DISTINCT DATE_FORMAT(s.pay_date, '%Y-%m') AS pay_month,
    department_id,
    AVG(amount) OVER(PARTITION BY DATE_FORMAT(s.pay_date, '%Y-%m')) as
company_avg,
    AVG(amount) OVER(PARTITION BY DATE_FORMAT(s.pay_date, '%Y-%m'),
department_id) as department_avg
  FROM
    salary s
    JOIN employee e ON s.employee_id = e.employee_id
)
SELECT
  pay_month,
  department_id,
  CASE
    WHEN department_avg > company_avg
      THEN 'higher'
    WHEN department_avg < company_avg
      THEN 'lower'
```

```
    ELSE
      'same'
  END AS comparison
FROM
  department_company_avg_monthly
ORDER BY
  department_id
;
```

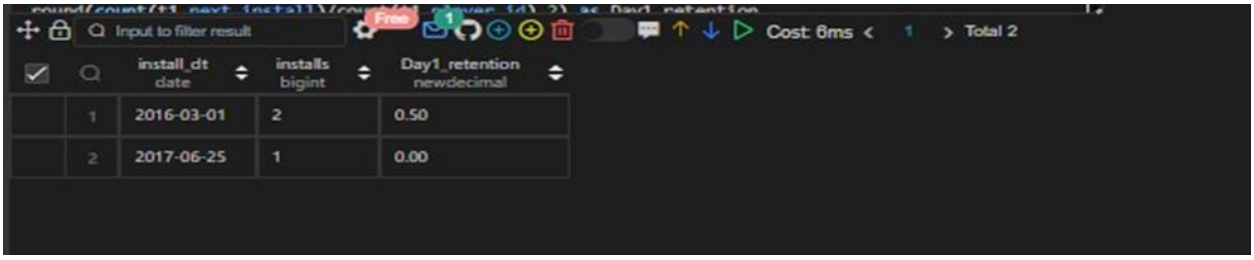


	pay_month varchar	department_id int	Comparison varchar
1	2017-2	1	same
2	2017-3	1	higher
3	2017-2	2	same
4	2017-3	2	lower

92. 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.

```
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;
```



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

93. 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.

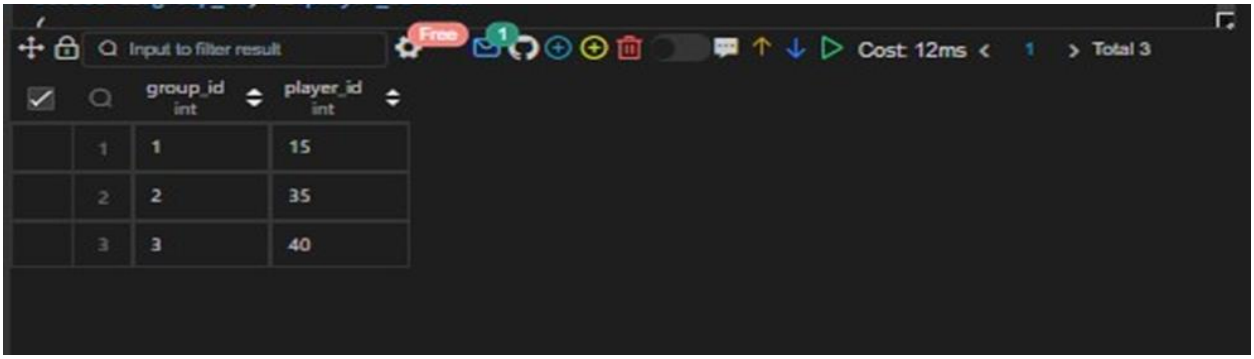
You may assume that, in each match, players belong to the same group.

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.

```
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;
```

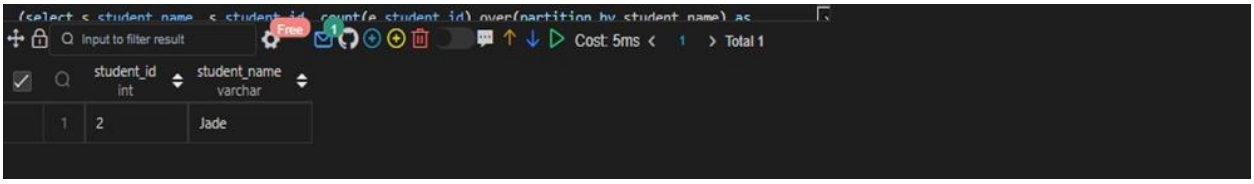


94. Each row of this table indicates that the student with student_id had a score points in the exam with id exam_id.

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.

```
        select t.student_id, t.student_name
from
(select s.student_name, s.student_id, count(e.student_id) over(partition by
student_name) as exams_given, case when e.score > min(e.score) over(partition
```

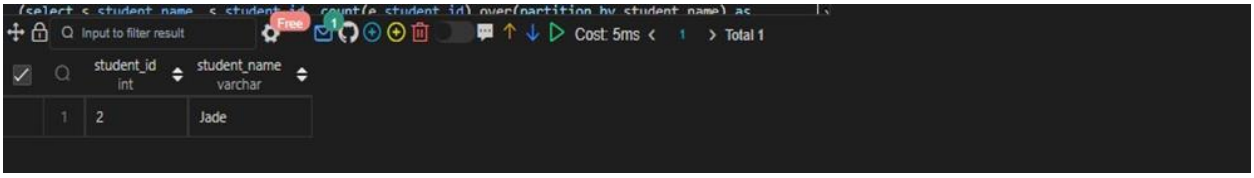
by e.exam_id) and e.score < max(e.score) over(partition by e.exam_id) then 1 else 0 end as quiet
from Exam e left join Student s on e.student_id = s.student_id)t group by t.student_name, t.student_id, t.exams_given having sum(t.quiet) = t.exams_given;



95. Each row of this table indicates that the student with student_id had a score in the exam with id exam_id.

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

select t.student_id, t.student_name from
(select s.student_name, s.student_id, count(e.student_id) over(partition by student_name) as exams_given, case when e.score > min(e.score) over(partition by e.exam_id) and e.score < max(e.score) over(partition by e.exam_id) then 1 else 0 end as quiet
from Exam e left join Student s on e.student_id = s.student_id)t
group by t.student_name, t.student_id, t.exams_given having sum(t.quiet) = t.exams_given;



96. 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.

```
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;
```

	user_id int	song_id int	song_plays newdecimal
1	777	1238	12
2	695	4520	2
3	125	9630	1

97. New TikTok users sign up with their emails, so each signup requires a text confirmation to activate the new user's account.

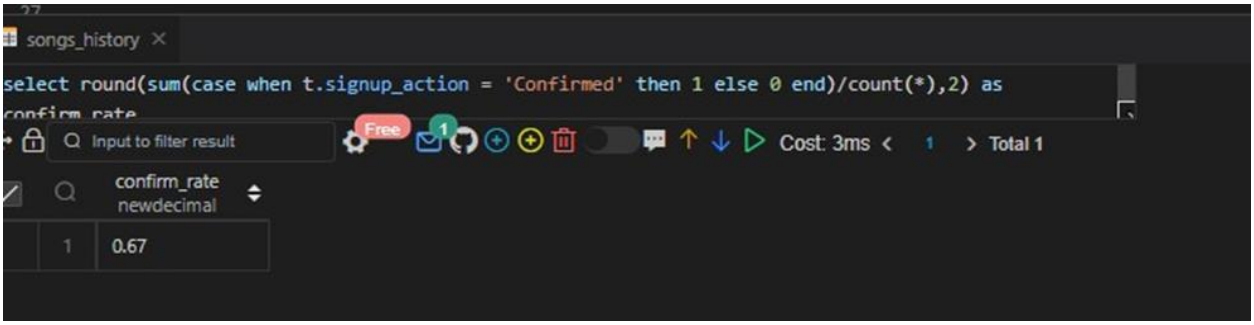
Write a query to find the confirmation rate of users who confirmed their signups with text messages. Round the result to 2 decimal places.

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.

```
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;
```

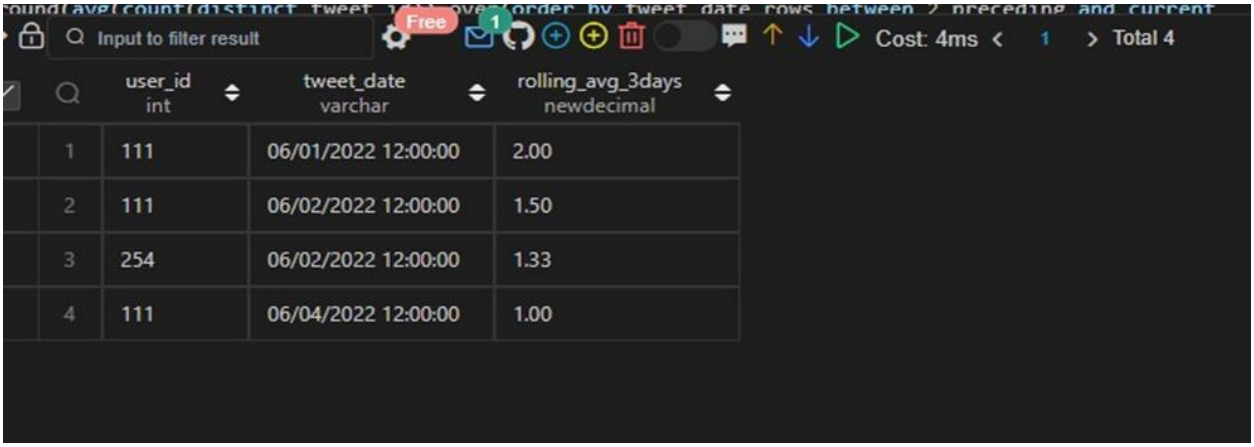


98. The table below contains information about tweets over a given period of time. Calculate the 3-day rolling average of tweets published by each user for each date that a tweet was posted. Output the user id, tweet date, and rolling averages rounded to 2 decimal places. Hint- Use Count and group by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

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;



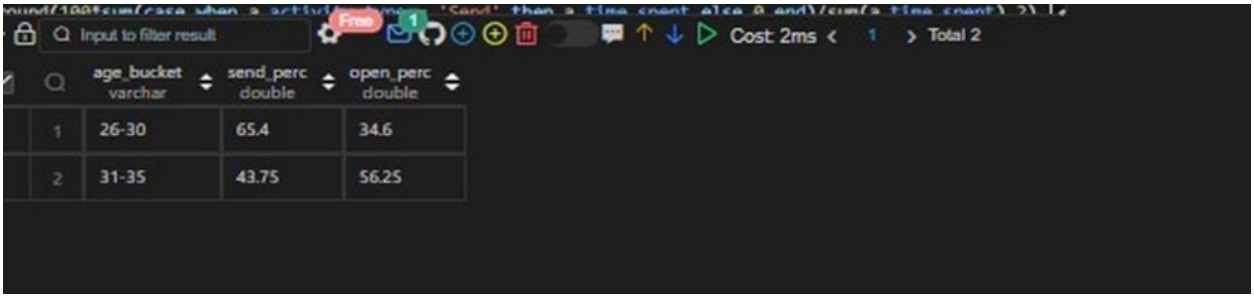
99. Assume you are given the tables below containing information on Snapchat users, their ages, and their time spent sending and opening snaps. Write a query to obtain a breakdown of the time spent sending vs. opening snaps (as a percentage of total time spent on these activities) for each age group.

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.

```
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;
```



100. 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 the largest follower base.

```
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;