SQL Assignment

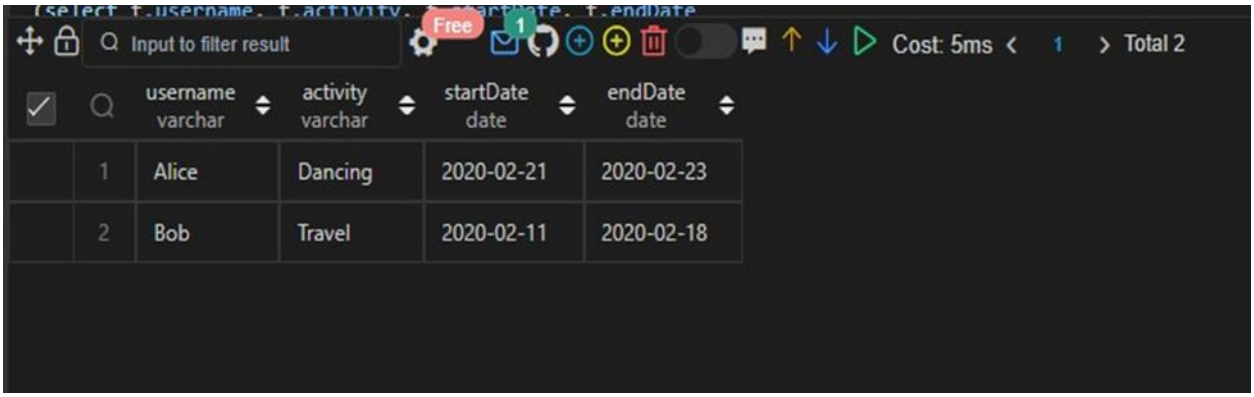
101. Write an SQL query to show the second most recent activity of each user.

If the user only has one activity, return that one. A user cannot perform more than one activity at the same time.

Return the result table in any order.

with new as
(select t.username, t.activity, t.startDate, t.endDate from(select username, activity, startDate, endDate, dense_rank() over(partition by username order by endDate desc) as r from UserActivity)t where r = 2

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

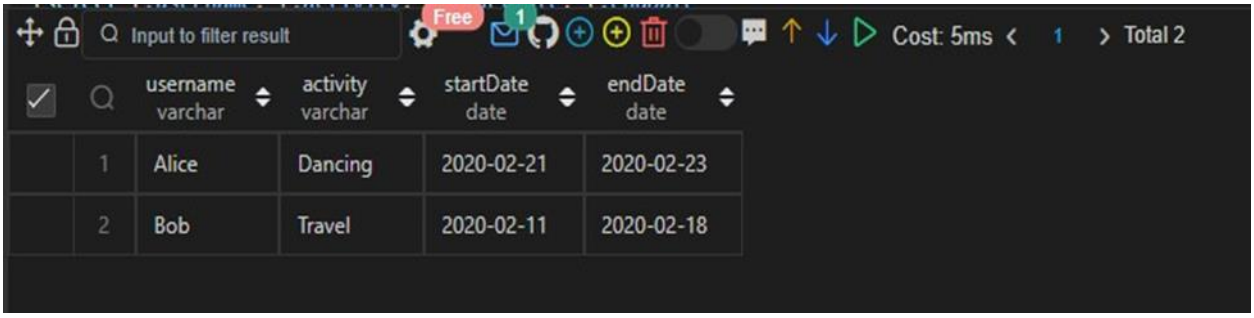


102. Write an SQL query to show the second most recent activity of each user.

If the user only has one activity, return that one. A user cannot perform more than one activity at the same time.

Return the result table in any order.

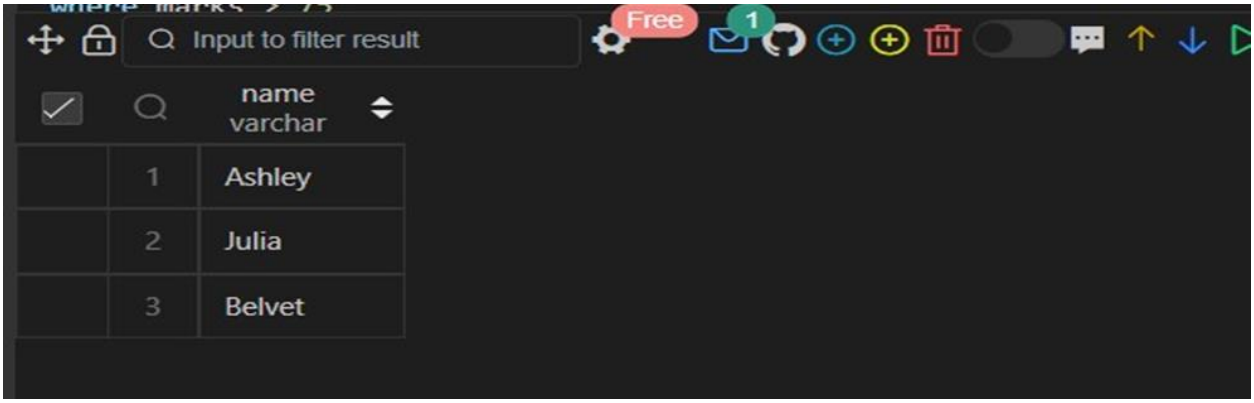
with new as
(select t.username, t.activity, t.startDate, t.endDate from(select username, activity, startDate, endDate, dense_rank() over(partition by username order by endDate desc) as r from UserActivity)t where r = 2
) select * from new union select n.username, n.activity, n.startDate, n.endDate from(select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



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

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

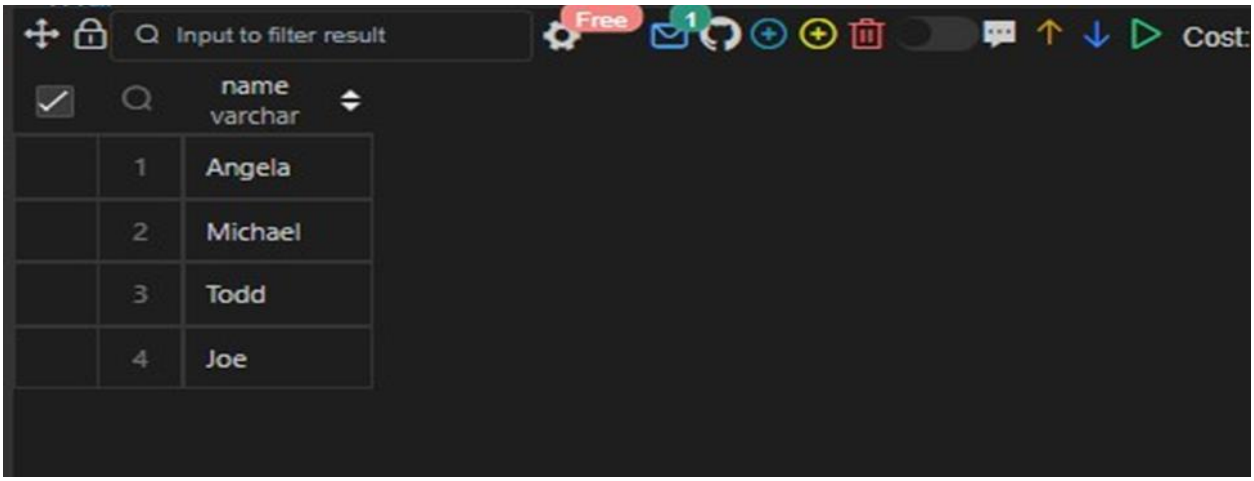
Select name from Students where marks > 75 order by right(name, 3), id;



	name varchar
1	Ashley
2	Julia
3	Belvet

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

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

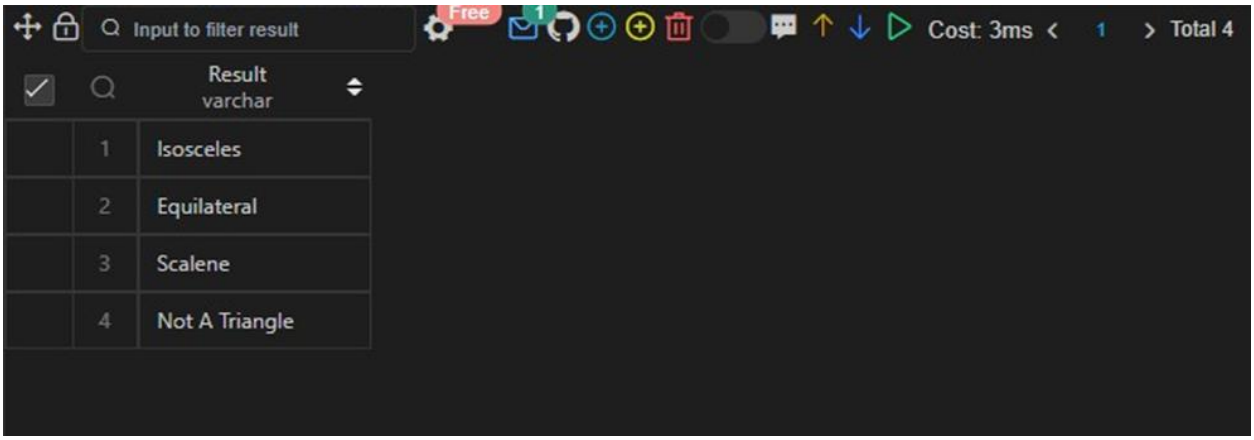


105. Write a query identifying the type of each record in the TRIANGLES table using its three side lengths.

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

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

select case when A+B > C and B+C > A and C+A > B then (case when A != B and B != C then 'Scalene' when A = B and B = C then 'Equilateral' else 'Isosceles' end) else 'Not A Triangle' end as Result from Triangles;

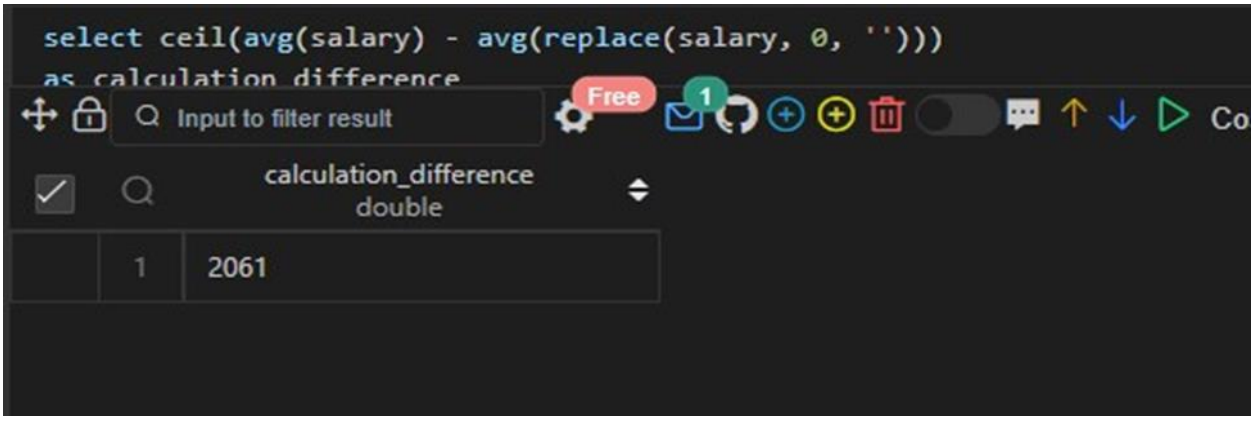


106. Samantha was tasked with calculating the average monthly salaries for all employees in the EMPLOYEES table, but did not realize her keyboard's 0 key was broken until after

completing the calculation. She wants your help finding the difference between her miscalculation (using salaries with any zeros removed), and the actual average salary.

Write a query calculating the amount of error (i.e.: actual - miscalculated average monthly salaries), and round it up to the next integer.

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

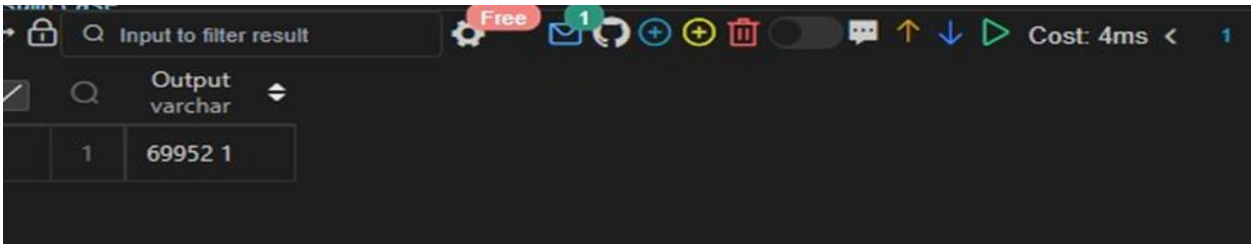


107. We define an employee's total earnings to be their monthly salary * months worked, and the maximum total earnings to be the maximum total earnings for any employee in the Employee table. Write a query to find the maximum total earnings for all employees as well as the total number of employees who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

Level - Easy

Hint - Use Aggregation functions

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



108. Generate the following two result sets:

1. Query an alphabetically ordered list of all names in OCCUPATIONS, immediately followed by the first letter of each profession as a parenthetical (i.e.: enclosed in parentheses). For example: AnActorName(A), ADoctorName(D), AProfessorName(P), and ASingerName(S). Query the number of occurrences of each occupation in OCCUPATIONS. Sort the occurrences in ascending order, and output them in the following format:

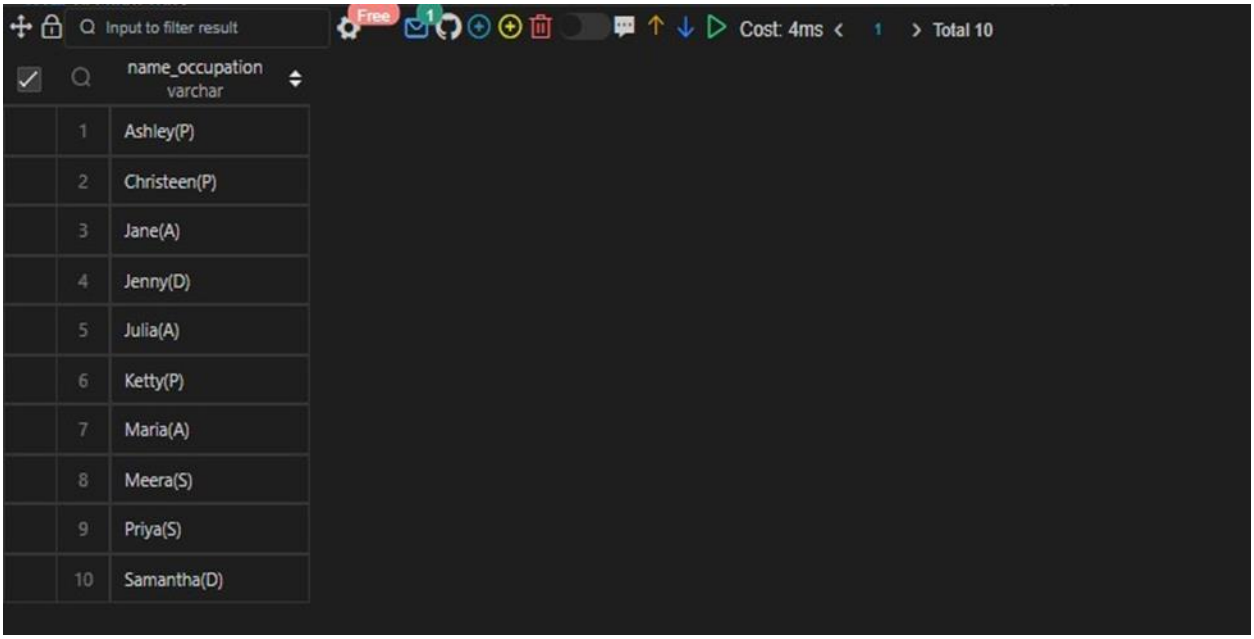
Level - Medium

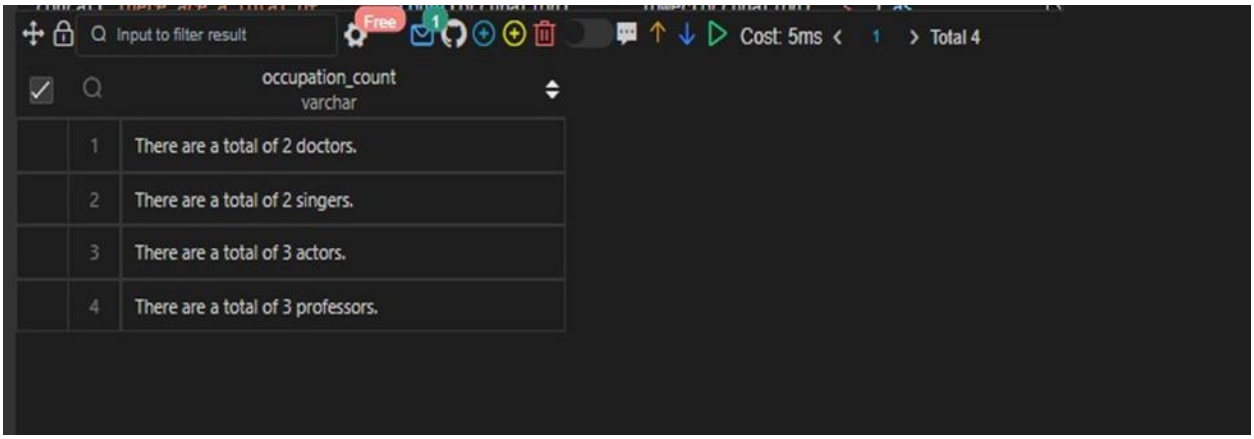
There are a total of [occupation_count] [occupation]s.

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

Note: There will be at least two entries in the table for each type of occupation.

select concat(name, '(', left(occupation,1),')') as name_occupation) from Occupations order by name; select concat('There are a total of', ' ', count(occupation), ' ', lower(occupation),
's.') as occupation_count from Occupations
group by occupation order by
count(occupation), occupation;





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

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

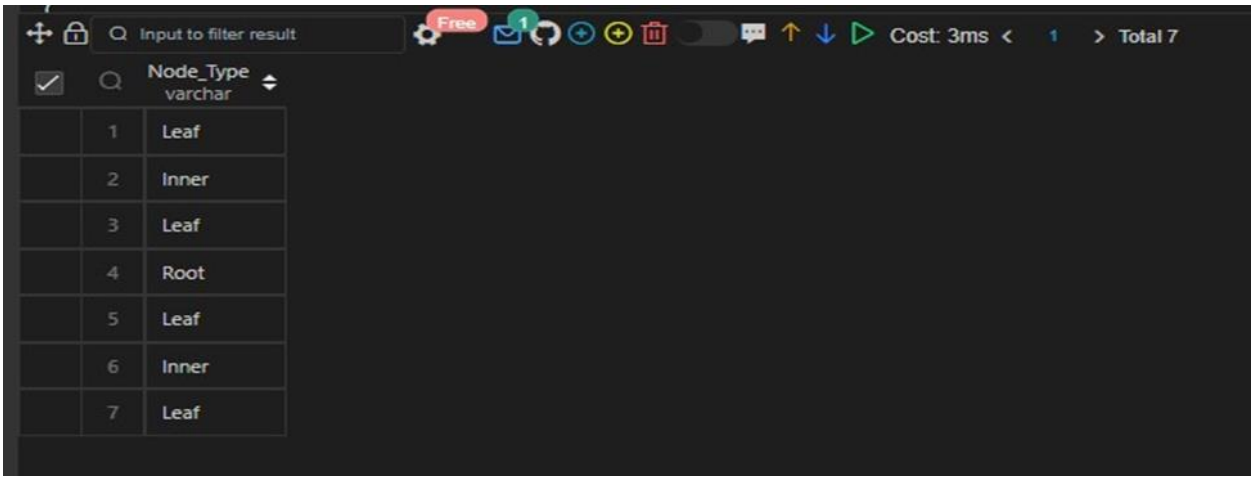
```
select max(case Occupation when 'Doctor' then Name end) as Doctors,
        max(case Occupation when 'Professor' then Name end) as Professors,
        max(case Occupation when 'Singer' then Name end) as Singers,
        max(case Occupation when 'Actor' then Name end) as Actors from  (
select occupation,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

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

110. Write a query to find the node type of Binary Tree ordered by the value of the node. Output one of the following for each node:

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

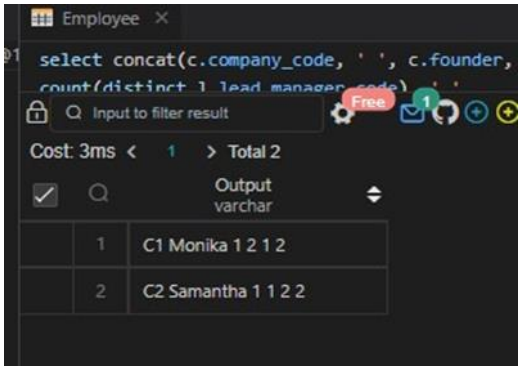
```
select  (
        cas
e
        when P is NULL then 'Root'          when N not in (select distinct P from BST
where P is not null) then 'Leaf' else 'Inner' end
) as Node_Type
from BST order
by N;
```



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

follows this hierarchy:
Given the table schemas below, write a query to print the company_code, founder name, total number of lead managers, total number of senior managers, total number of managers, and total number of employees. Order your output by ascending company_code. Level - Medium Note:
The tables may contain duplicate records.
The company_code is string, so the sorting should not be numeric. For example, if the company_codes are C_1, C_2, and C_10, then the ascending company_codes will be C_1, C_10, and C_2.

```
select concat(c.company_code, ' ', c.founder, ' ',
count(distinct l.lead_manager_code), ' ', count(distinct
s.senior_manager_code), ' ', count(distinct
m.manager_code), ' ', count(distinct e.employee_code))
as Output from Company c left outer join Lead_Manager
l on c.company_code = l.company_code left join
Senior_Manager s
on l.lead_manager_code = s.lead_manager_code left join Manager m on
s.senior_manager_code = m.senior_manager_code left join Employee e on
m.manager_code = e.manager_code group by c.company_code, c.founder order by
c.company_code;
```

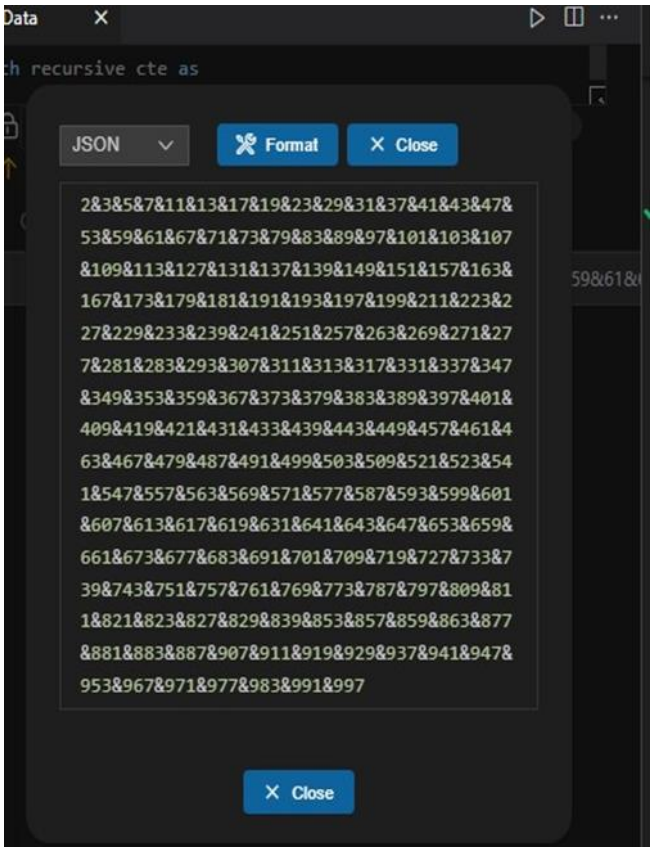


112. Write a query to print all prime numbers less than or equal to 1000. Print your result on a single line, and use the ampersand () character as your separator (instead of a space).

For example, the output for all prime numbers <=10 would be: 2&3&5&7

Hint - Firstly, select L Prime_Number from (select Level L from Dual connect Level ≤ 1000) and then do the same thing to create Level M, and then filter by M ≤ L and then group by L having count(case when L/M = truc(L/M) then ‘Y’ end) = 2 order by L

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

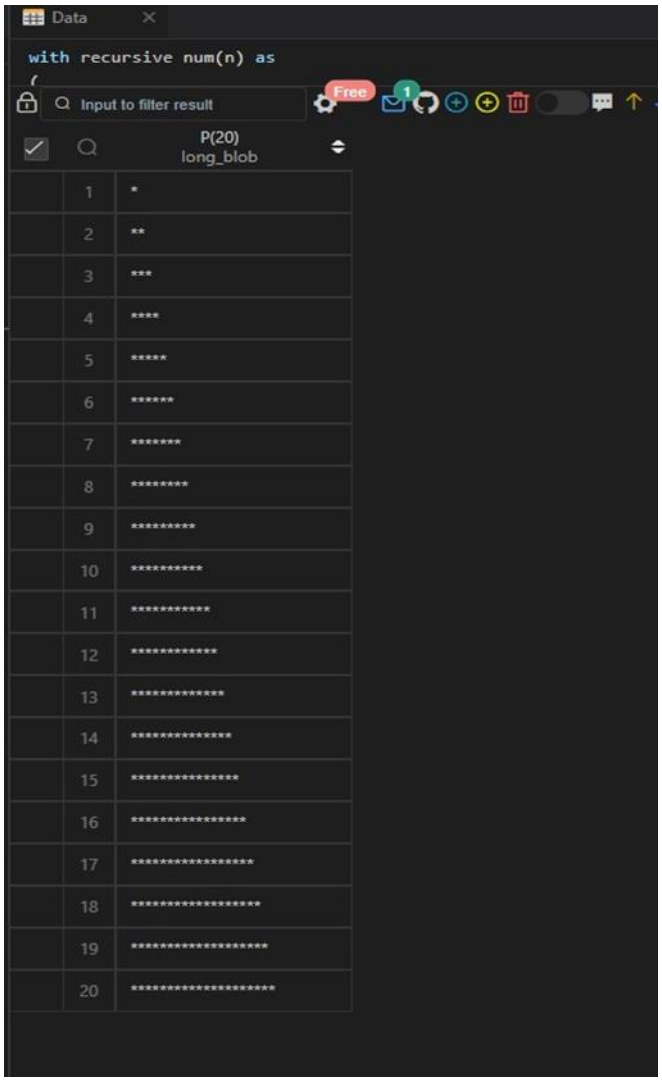


113. P(R) represents a pattern drawn by Julia in R rows. The following pattern represents P(5):

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

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

```
with recursive num(n) as
( select 1
        union
        select n + 1
        from num
        where n + 1 <= 20
) select lpad('', num.n, '*') as 'P(20)' from
num;
```



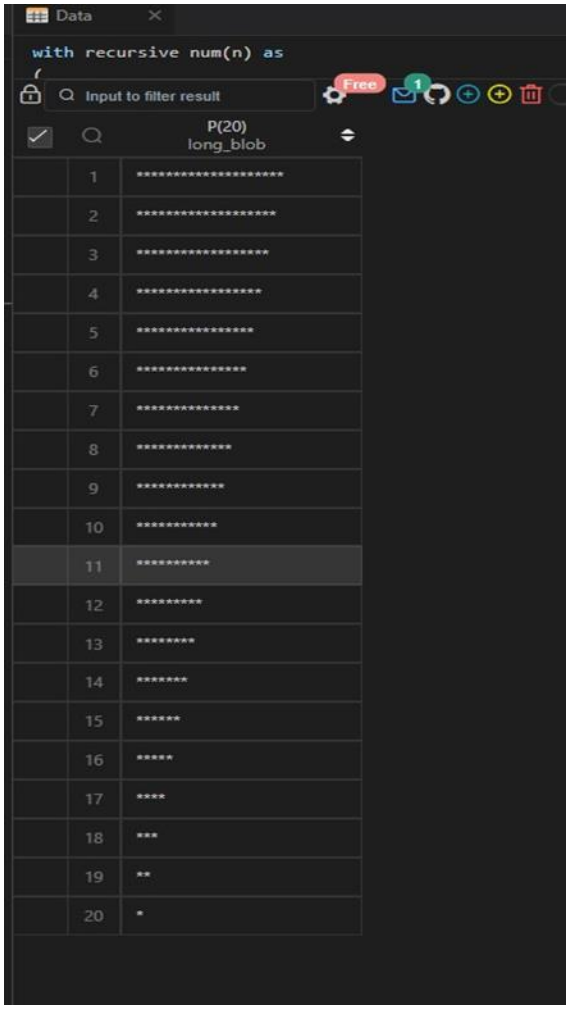
113. P(R) represents a pattern drawn by Julia in R rows. The following pattern represents P(5):

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

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

```
with recursive num(n) as
( select 20      union
          select n - 1
```

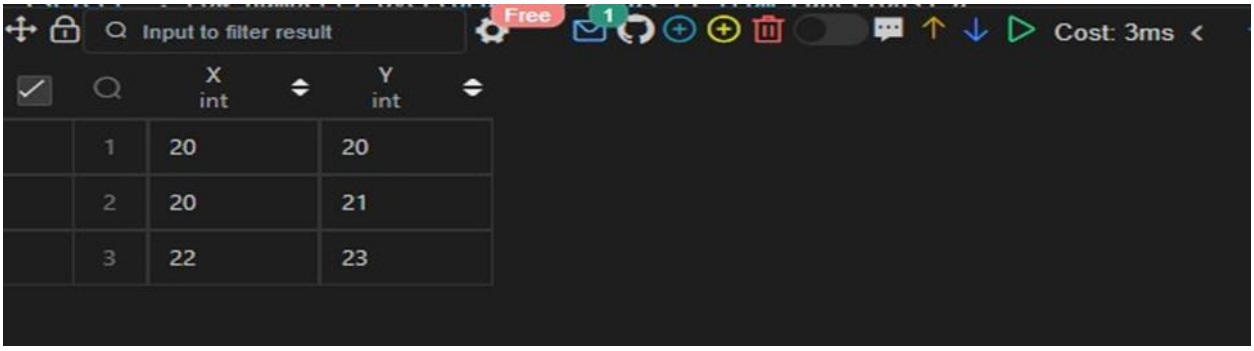
```
        from num
        where n - 1 >= 1
) select lpad('', num.n, '*') as 'P(20)' from
num;
```



114. Two pairs (X1, Y1) and (X2, Y2) are said to be symmetric pairs if X1 = Y2 and X2 = Y1. Write a query to output all such symmetric pairs in ascending order by the value of X. List the rows such that X1 ≤ Y1.

```
WITH RECURSIVE numbers AS(
  SELECT 1 AS n, '*' AS star
  UNION
  SELECT n+1,'*' AS star FROM numbers WHERE n < 20
)
SELECT
  GROUP_CONCAT(n1.star SEPARATOR '') AS stars
FROM
```

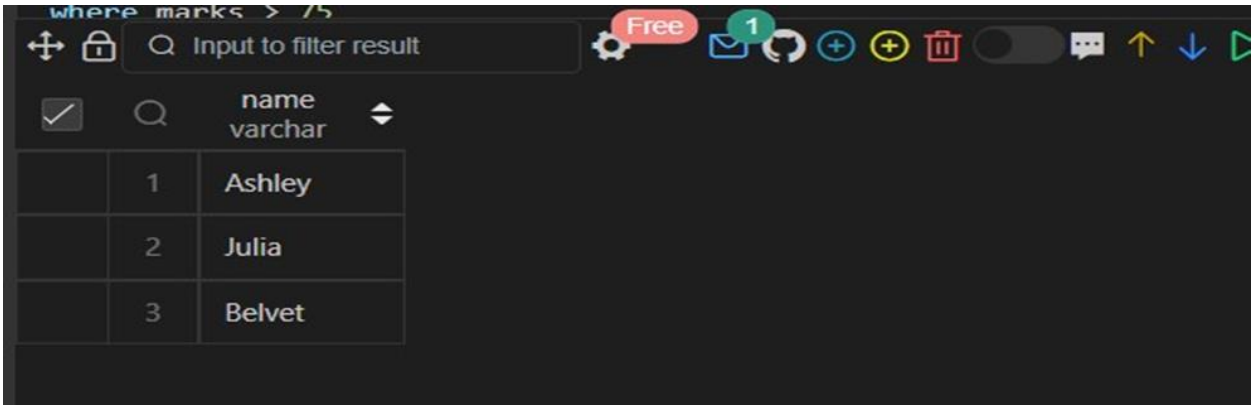
```
  numbers n1
  JOIN numbers n2 ON n1.n <= n2.n
GROUP BY
  n1.n
;
```



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

Hint - Use Like

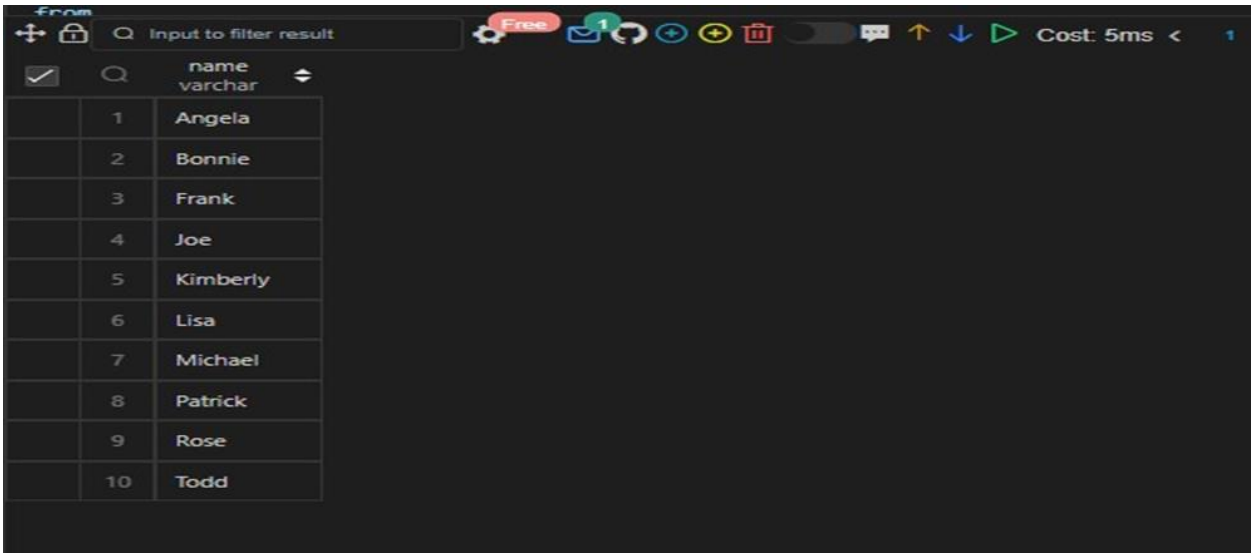
Select name from Students where marks > 75 order by right(name, 3), id;



116. 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;



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

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

```
SELECT
  name
FROM
  employee
WHERE
  salary > 2000
  AND months < 10
ORDER BY
  employee_id
;
```

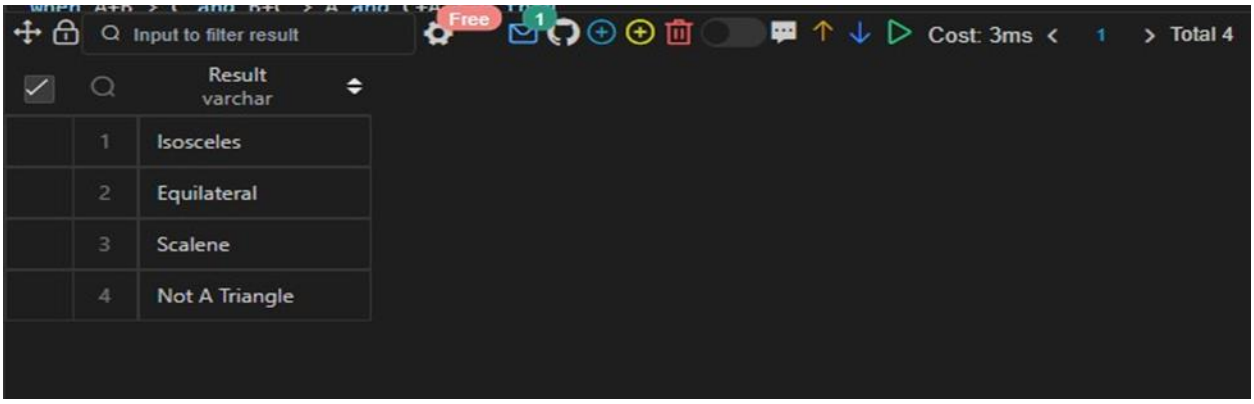
118. Write a query identifying the type of each record in the TRIANGLES table using its three side lengths. Output one of the following statements for each record in the table:

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

Level - Easy
Hint - Use predefined functions for calculation.

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

(case when A != B and B != C then 'Scalene' when A = B and B = C then 'Equilateral' else 'Isosceles' end) else 'Not A Triangle' end as Result from Triangles;

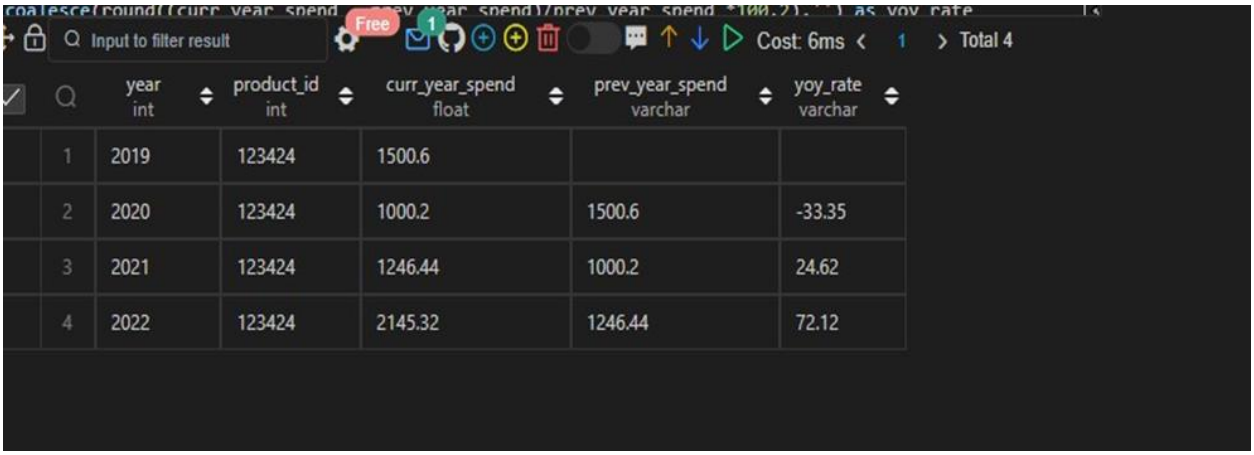


119. Assume you are given the table below containing information on user transactions for particular products. Write a query to obtain the year-on-year growth rate for the total spend of each product for each year.

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;



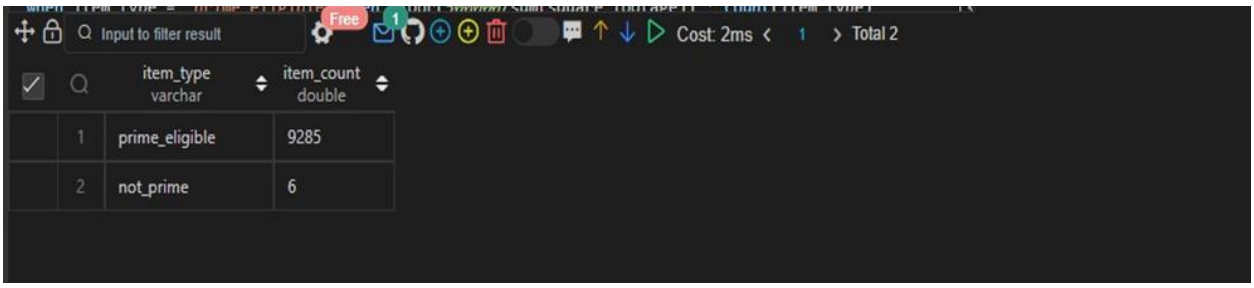
	year int	product_id int	curr_year_spend float	prev_year_spend varchar	yoy_rate varchar
1	2019	123424	1500.6		
2	2020	123424	1000.2	1500.6	-33.35
3	2021	123424	1246.44	1000.2	24.62
4	2022	123424	2145.32	1246.44	72.12

120. 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.

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;



	item_type varchar	item_count double
1	prime_eligible	9285
2	not_prime	6

121. 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

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

Data

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

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

	month int	monthly_active_users bigint
1	6	1

122. 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);
```

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

search_frequency

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

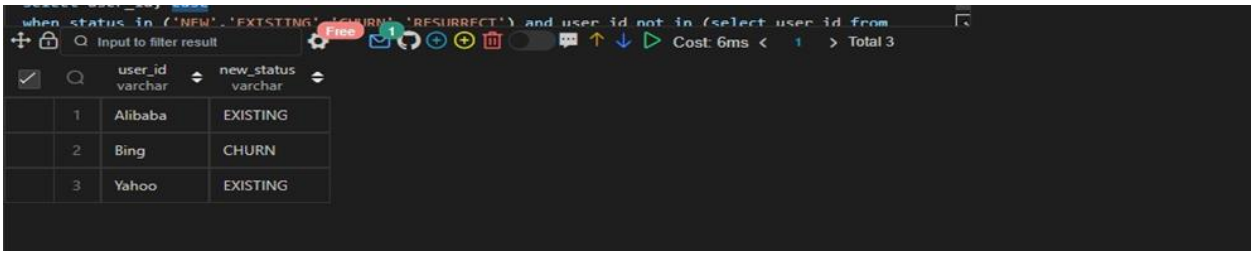
Input to filter result | Cost: 7ms | 1 | Total 1

	median newdecimal
1	2.5

123. 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;



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

124. 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-

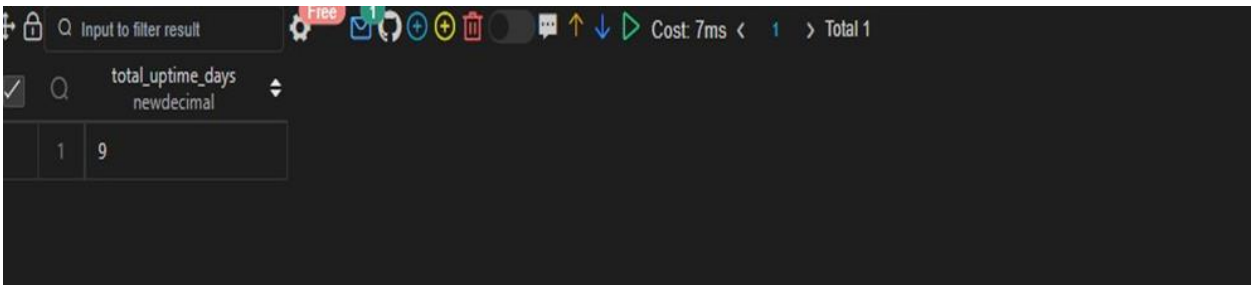
1. Calculate individual uptimes

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;



	total_uptime_days newdecimal
1	9

125. 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;

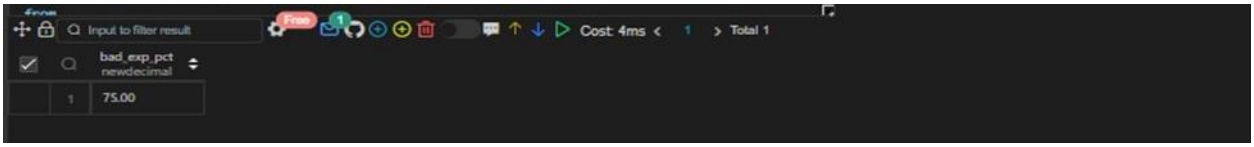
126. DoorDash's Growth Team is trying to make sure new users (those who are making orders in their first 14 days) have a great experience on all their orders in their 2 weeks on the platform. Unfortunately, many deliveries are being messed up because:

- 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;
```

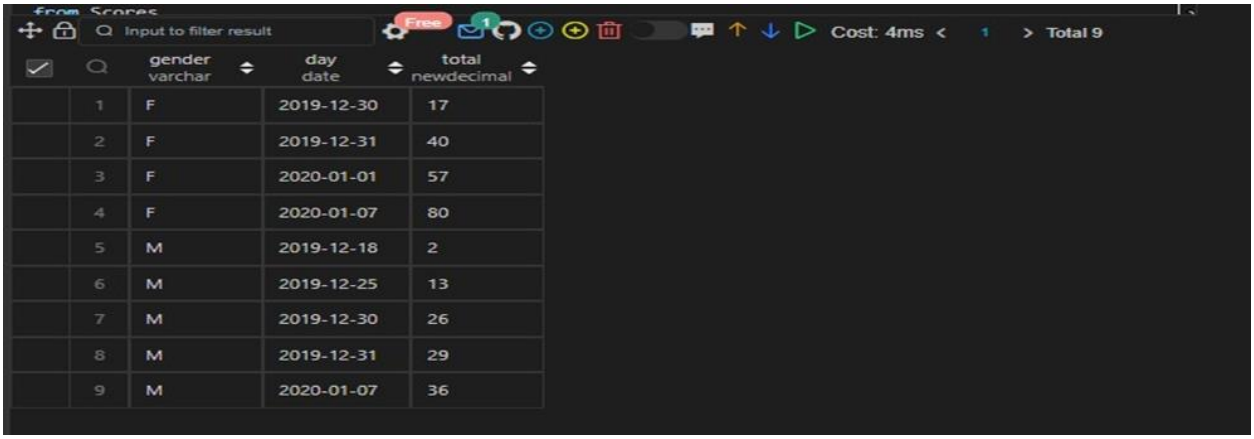


127. A competition is held between the female team and the male team.
Each row of this table indicates that a player_name and with gender has scored score_point in someday.

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.

```
SELECT
  gender,
  day,
  sum(score_points) OVER(PARTITION BY gender ORDER BY day) AS total
FROM
  scores;
```



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

128. 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.

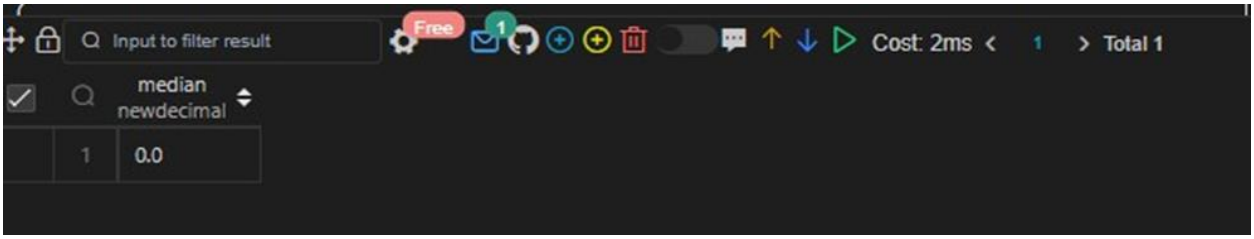
Write an SQL query to find the countries where this company can invest.

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;
```

129. 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.

```
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);
```

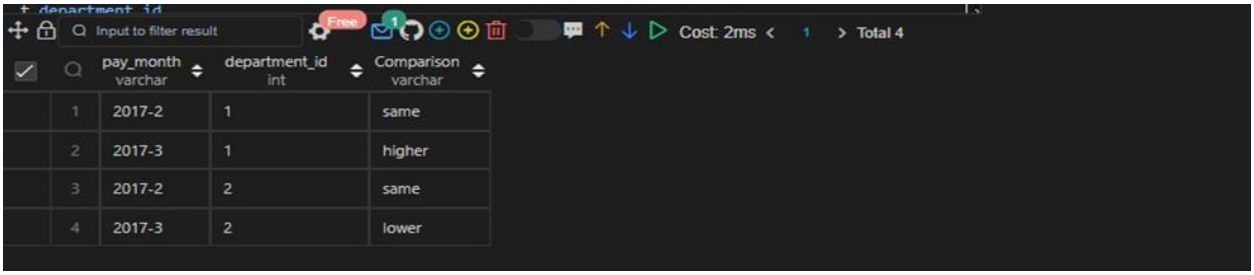


130. Write an SQL query to report the comparison result (higher/lower/same) of the average salary of employees in a department to the company's average salary.

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

131. 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;
```

Input to filter result | Free | Cost: 6ms | 1 | Total 2

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

132. 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;
```

Input to filter result | Free | Cost: 12ms | 1 | Total 3

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

133. 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.

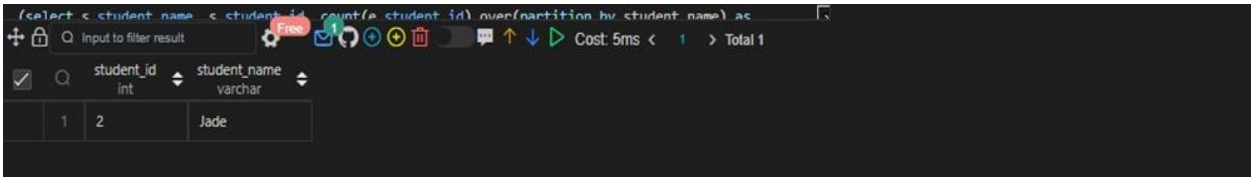
```
          select t.student_id, t.student_name
from
(select s.student_name, s.student_id, count(e.student_id) over(partition by student_name)
as exams_given, case when e.score > min(e.score) over(partition by e.exam_id) and
e.score < max(e.score) over(partition by e.exam_id) then 1 else 0 end as quiet
```

 # 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 ;

134. 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 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;

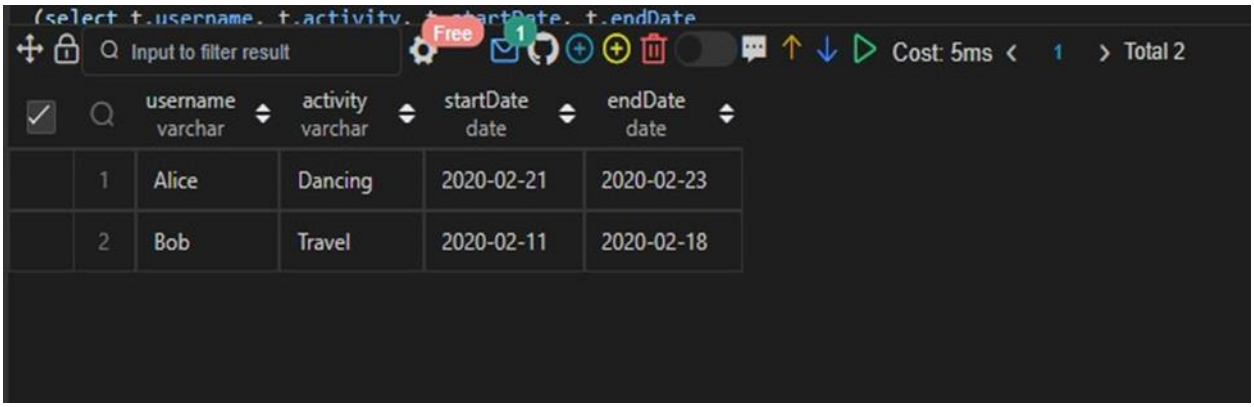


135. This table contains information about the activity performed by each user in a period of time. A person with a username performed an activity from startDate to endDate.

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

with new as
(select t.username, t.activity, t.startDate, t.endDate from(select username,
activity, startDate, endDate, dense_rank() over(partition by username order by
endDate desc) as r from UserActivity)t where r = 2
) select * from new union select n.username, n.activity,
n.startDate, n.endDate from(select username, activity,
startDate, endDate,
dense_rank() over(partition by username order by endDate desc) as r from
UserActivity)n

where r = 1 and username not in (select username from new);



136.

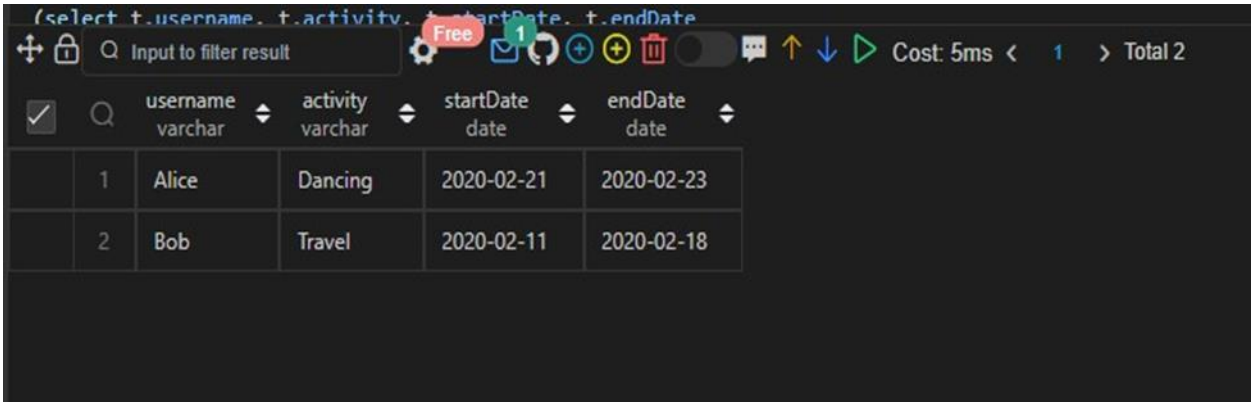
Table: UserActivity

Column Name	Type
username	varchar
activity	varchar
startDate	Date
endDate	Date

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

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

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

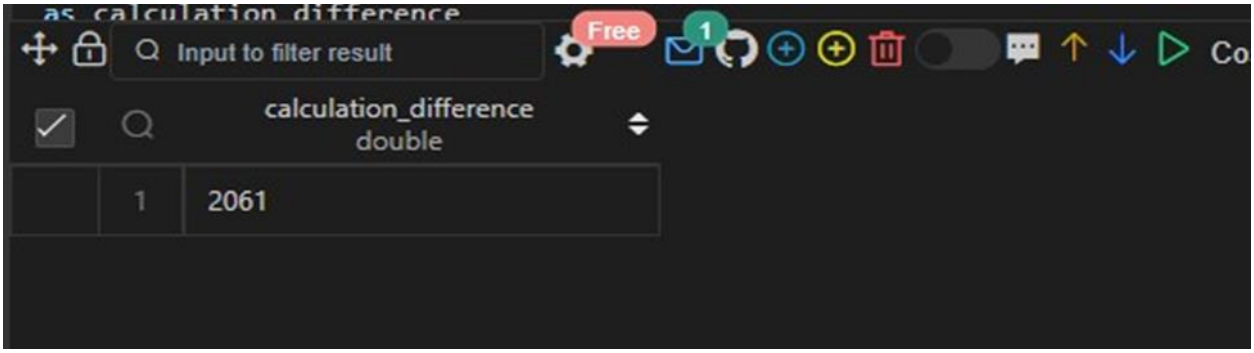


	username varchar	activity varchar	startDate date	endDate date
1	Alice	Dancing	2020-02-21	2020-02-23
2	Bob	Travel	2020-02-11	2020-02-18

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

Write a query calculating the amount of error (i.e.: actual - miscalculated average monthly salaries), and round it up to the next integer.

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



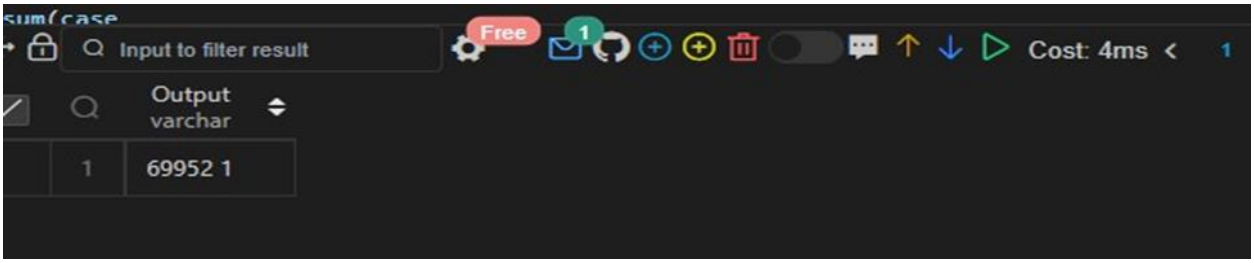
	calculation_difference double
1	2061

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

Level - Easy

Hint - Use Aggregation functions

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

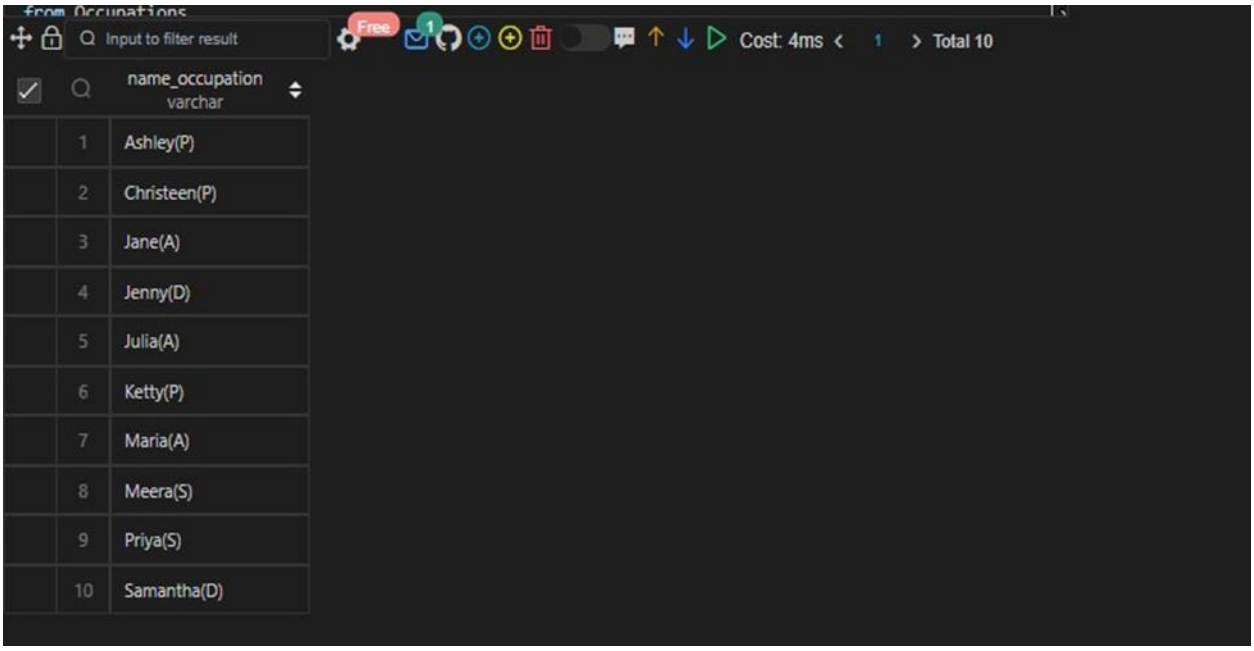


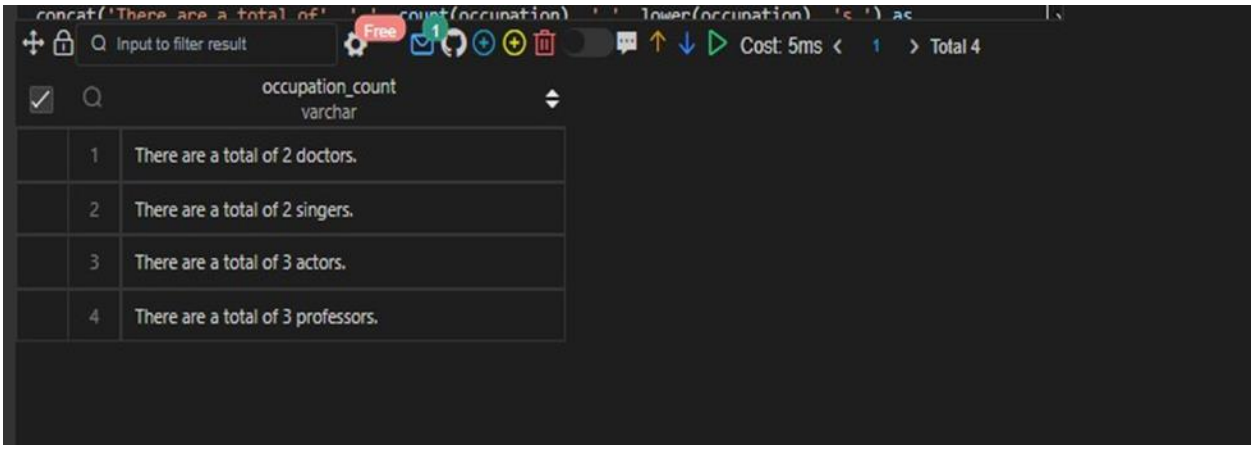
139. Generate the following two result sets:

1. Query an alphabetically ordered list of all names in OCCUPATIONS, immediately followed by the first letter of each profession as a parenthetical (i.e.: enclosed in parentheses). For example: AnActorName(A), ADoctorName(D), AProfessorName(P), and ASingerName(S). Query the number of occurrences of each occupation in OCCUPATIONS. Sort the occurrences in ascending order, and output them in the following format:
Level - Medium
There are a total of [occupation_count] [occupation]s.

2. where [occupation_count] is the number of occurrences of an occupation in OCCUPATIONS and [occupation] is the lowercase occupation name. If more than one Occupation has the same [occupation_count], they should be ordered alphabetically.
Note: There will be at least two entries in the table for each type of occupation.

select concat(name, '(', left(occupation,1),')') as name_occupation) from Occupations order by name; select concat('There are a total of', ' ', count(occupation), ' ', lower(occupation), 's.') as occupation_count from Occupations
group by occupation order by
count(occupation), occupation;



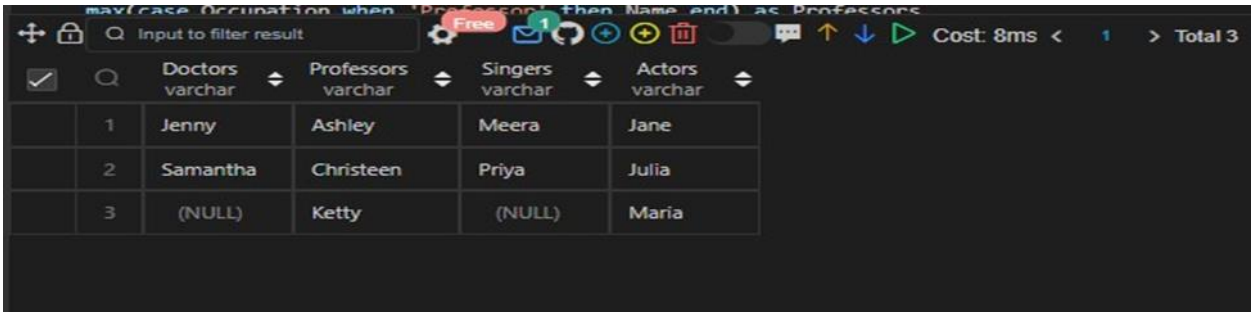


	occupation_count varchar
1	There are a total of 2 doctors.
2	There are a total of 2 singers.
3	There are a total of 3 actors.
4	There are a total of 3 professors.

140. Pivot the Occupation column in OCCUPATIONS so that each Name is sorted alphabetically and displayed underneath its corresponding Occupation. The output column headers should be Doctor, Professor, Singer, and Actor, respectively.

Note: Print NULL when there are no more names corresponding to an occupation.

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



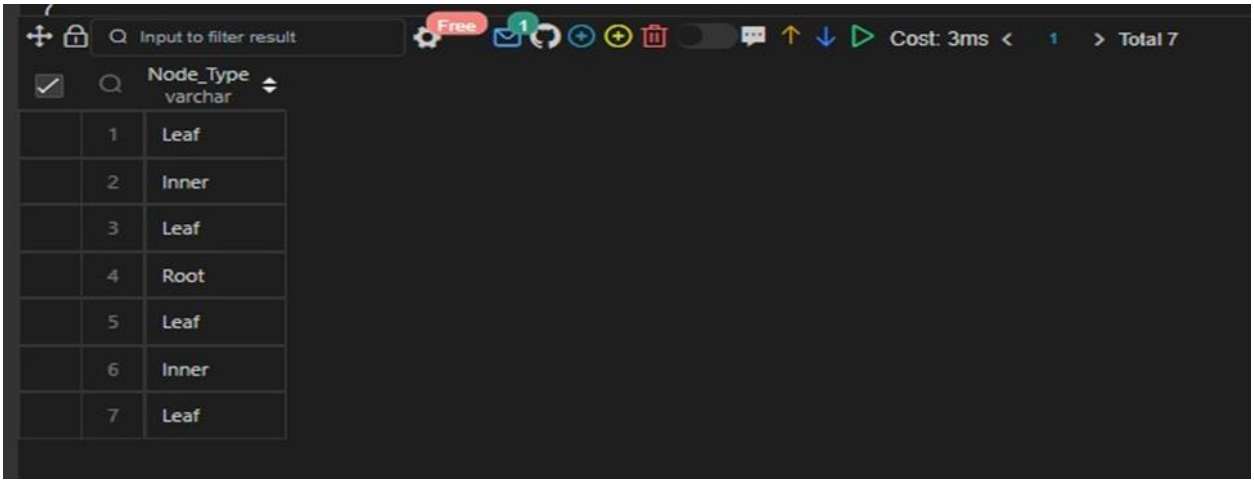
	Doctors varchar	Professors varchar	Singers varchar	Actors varchar
1	Jenny	Ashley	Meera	Jane
2	Samantha	Christeen	Priya	Julia
3	(NULL)	Ketty	(NULL)	Maria

141. Write a query to find the node type of Binary Tree ordered by the value of the node. Output one of the following for each node:

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

```
select  (
		case
e
		when P is NULL then 'Root'	when N not in (select distinct P from BST where P is not null) then 'Leaf' else 'Inner' end
```

) as Node_Type
from BST order
by N;



142. Amber's conglomerate corporation just acquired some new companies. Each of the companies follows this hierarchy:
Given the table schemas below, write a query to print the company_code, founder name, total number of lead managers, total number of senior managers, total number of managers, and total number of employees. Order your output by ascending company_code. Level - Medium
Note:

- The tables may contain duplicate records.
- The company_code is string, so the sorting should not be numeric. For example, if the company_codes are C_1, C_2, and C_10, then the ascending company_codes will be C_1, C_10, and C_2.

Input Format
The following tables contain company data:

- Company: The company_code is the code of the company and founder is the founder of the company.
- Lead_Manager: The lead_manager_code is the code of the lead manager, and the company_code is the code of the working company.
- Senior_Manager: The senior_manager_code is the code of the senior manager, the lead_manager_code is the code of its lead manager, and the company_code is the code of the working company.
- Manager: The manager_code is the code of the manager, the senior_manager_code is the code of its senior manager, the lead_manager_code is the code of its lead manager, and the company_code is the code of the working company.

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

Founder

⇓

Lead Manager

⇓

Senior Manager

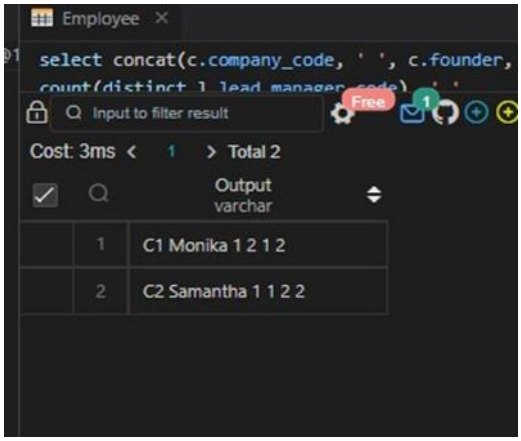
⇓

Manager

⇓

Employee

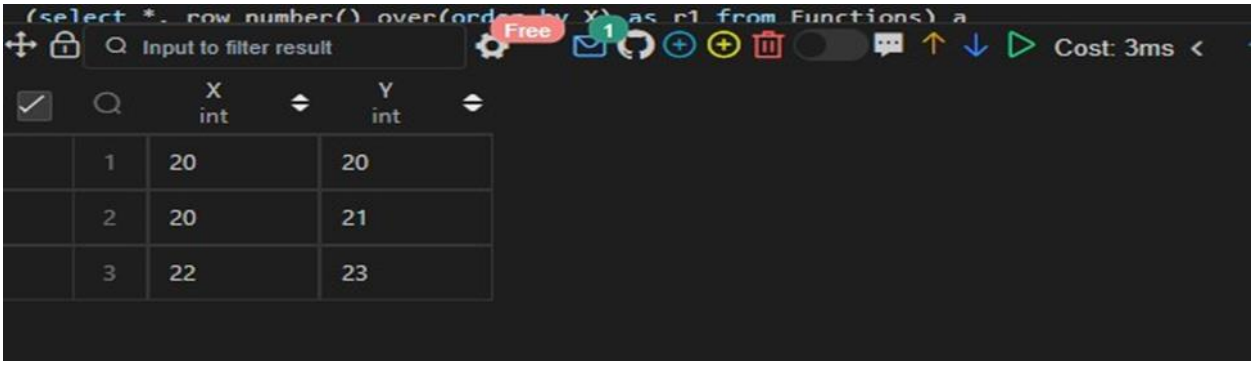
```
select concat(c.company_code, ' ', c.founder, ' ',
count(distinct l.lead_manager_code), ' ', count(distinct
s.senior_manager_code), ' ', count(distinct m.manager_code),
' ', count(distinct e.employee_code)) as Output from Company
c left outer join Lead_Manager l on c.company_code =
l.company_code left join Senior_Manager s
on l.lead_manager_code = s.lead_manager_code left join
Manager m on s.senior_manager_code =
m.senior_manager_code left join Employee e
on m.manager_code = e.manager_code group by
c.company_code, c.founder order by
c.company_code;
```



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

```
WITH functions_serialized AS(
  SELECT
    *,
```

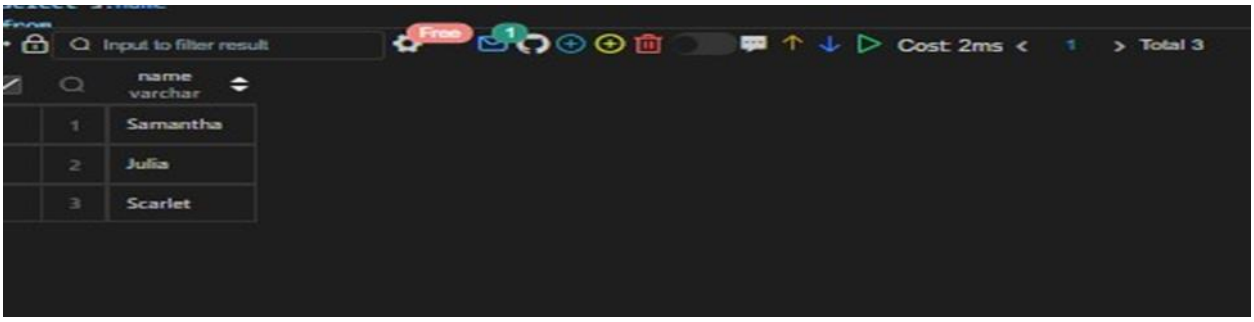
```
    ROW_NUMBER() OVER(ORDER BY x) AS serial
  FROM
    functions
)
SELECT
  DISTINCT f1.x,
  f1.y
FROM
  functions_serialized f1
WHERE
  EXISTS(
    SELECT
      *
    FROM
      functions_serialized f2
    WHERE
      f1.serial <> f2.serial
      AND f1.x = f2.y
      AND f1.y = f2.x
  )
  AND f1.x <= f1.y
;
```



144. You are given three tables: Students, Friends and Packages. Students contain two columns: ID and Name. Friends contains two columns: ID and Friend_ID (ID of the ONLY best friend). Packages contain two columns: ID and Salary (offered salary in $ thousands per month).

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

select s.name from Students s join Friends f on s.id = f.id join Packages sp on sp.id = s.id join Packages fp on fp.id = f.friend_id where fp.salary > sp.salary order by fp.salary;



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

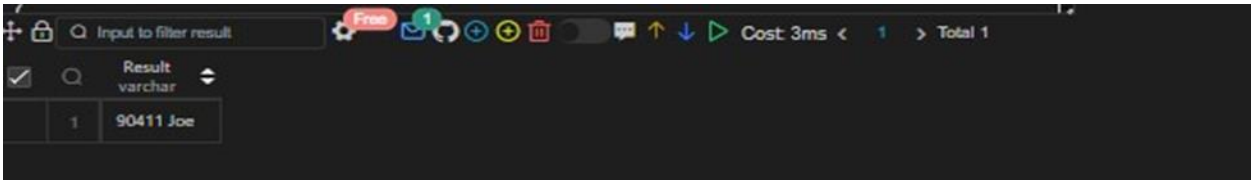
Level - Medium

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

The following tables contain contest data:

- Hackers: The hacker_id is the id of the hacker, and name is the name of the hacker.
- Difficulty: The difficult_level is the level of difficulty of the challenge, and score is the score of the challenge for the difficulty level.
- Challenges: The challenge_id is the id of the challenge, the hacker_id is the id of the hacker who created the challenge, and difficulty_level is the level of difficulty of the challenge.
- Submissions: The submission_id is the id of the submission, hacker_id is the id of the hacker who made the submission, challenge_id is the id of the challenge that the submission belongs to, and score is the score of the submission.

```
select concat(t1.hacker_id, ' ', t1.name) as Result from
(        select t.hacker_id, t.name,  dense_rank() over(order by
full_score_challenge_count desc) as r from    (        select h.hacker_id, h.name,
count(h.hacker_id) as full_score_challenge_count     from
      Submissions s     join      Hackers h        on
s.hacker_id = h.hacker_id        join      Challenges c
          on s.challenge_id = c.challenge_id          join
      Difficulty d
          on d.difficulty_level = c.difficulty_level           where
s.score = d.score         group by h.hacker_id, h.name
          having full_score_challenge_count > 1
          ) t ) t1
where t1.r = 1
order by t1.hacker_id;
```



146. You are given a table, Projects, containing three columns: Task_ID, Start_Date and End_Date. It is guaranteed that the difference between the End_Date and the Start_Date is equal to 1 day for each row in the table.

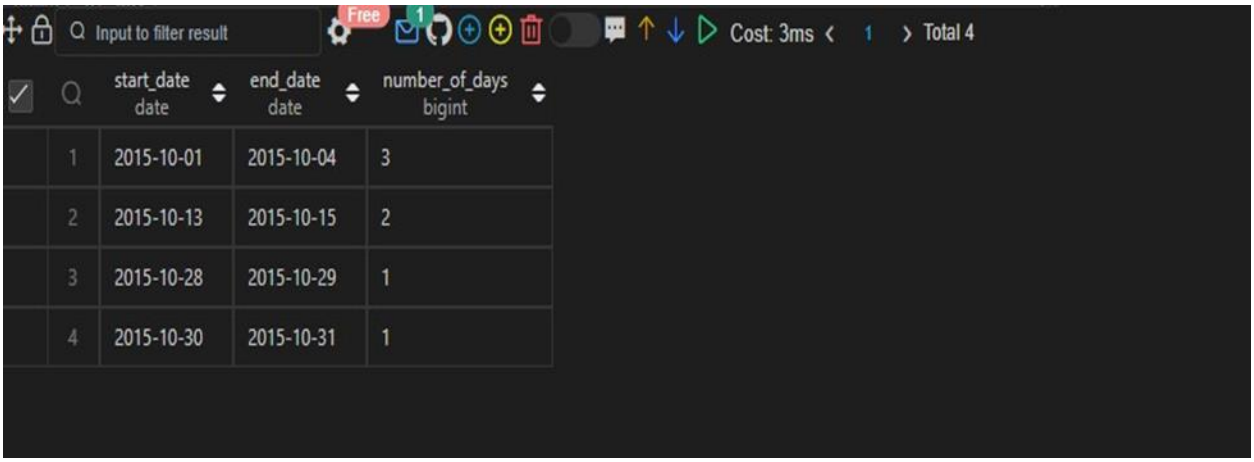
Level - Medium

Hint - Use Advance join

If the End_Date of the tasks are consecutive, then they are part of the same project. Samantha is interested in finding the total number of different projects completed.

Write a query to output the start and end dates of projects listed by the number of days it took to complete the project in ascending order. If there is more than one project that have the same number of completion days, then order by the start date of the project.

```
select s.start_date, min(e.end_date) as end_date, (min(e.end_date) -
s.start_date) as number_of_days  from
(select start_date from Projects where start_date - 1 not in (select start_date from Projects)) s,
(select end_date from Projects where end_date + 1 not in (select end_date from Projects)) e
where s.start_date <= e.end_date group by
s.start_date;
```

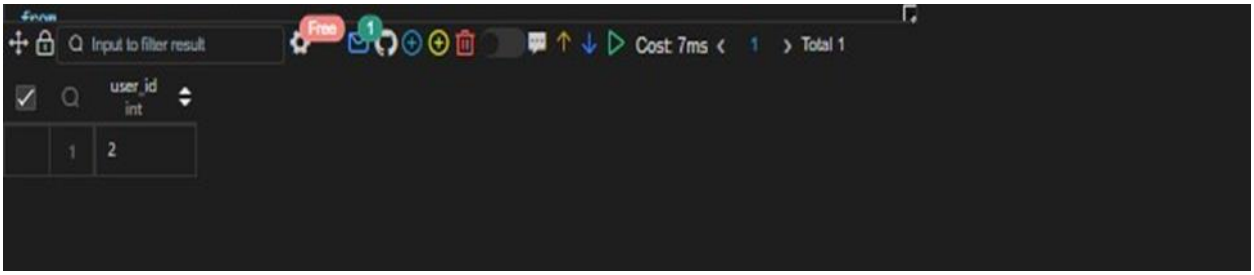


147. In an effort to identify high-value customers, Amazon asked for your help to obtain data about users who go on shopping sprees. A shopping spree occurs when a user makes purchases on 3 or more consecutive days.

List the user IDs who have gone on at least 1 shopping spree in ascending order.

```
SELECT
 *,
 DATEDIFF(LEAD(transaction_date) OVER(PARTITION BY user_id ORDER BY transaction_date),
transaction_date) AS diff1,
 DATEDIFF(transaction_date, LAG(transaction_date) OVER(PARTITION BY user_id ORDER BY
transaction_date)) AS diff2
```

```
FROM
  transactions;

SELECT
  distinct user_id
FROM(
  SELECT
    *,
    DATEDIFF(LEAD(transaction_date) OVER(PARTITION BY user_id ORDER BY transaction_date),
transaction_date) AS diff1,
    DATEDIFF(transaction_date, LAG(transaction_date) OVER(PARTITION BY user_id ORDER BY
transaction_date)) AS diff2
  FROM
    transactions
) AS trx_with_date_diff
WHERE
  trx_with_date_diff.diff1 = 1
  AND trx_with_date_diff.diff2 = 1
;
```



148. You are given a table of PayPal payments showing the payer, the recipient, and the amount paid. A two-way unique relationship is established when two people send money back and forth. Write a query to find the number of two-way unique relationships in this data.

Assumption:

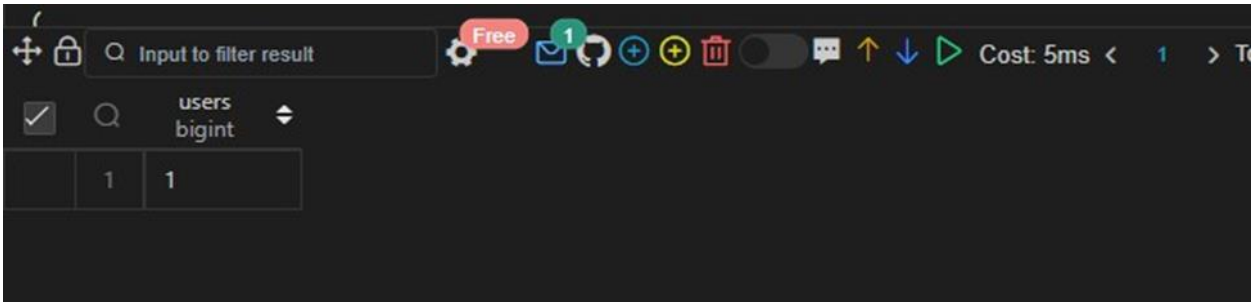
- A payer can send money to the same recipient multiple times.

```
SELECT
  ROUND(COUNT(
    distinct p1.payer_id,
    p1.recipient_id
  )/2) AS unique_relationships
FROM
  payments p1
  JOIN payments p2 ON p1. payer_id = p2.recipient_id
    AND p1.recipient_id = p2.payer_id;
```

149. Assume you are given the table below on user transactions. Write a query to obtain the list of customers whose first transaction was valued at $50 or more. Output the number of users.
Clarification:

- Use the transaction_date field to determine which transaction should be labeled as the first for each user.
- Use a specific function (we can't give too much away!) to account for scenarios where a user had multiple transactions on the same day, and one of those was the first.

```
WITH user_trx_serialized AS(
  SELECT
    *,
    DENSE_RANK() OVER(PARTITION BY user_id ORDER BY transaction_date) as serial
  FROM
    user_transactions
)
SELECT
  COUNT(*) AS users
FROM
  user_trx_serialized
WHERE
  serial = 1
  AND spend >= 50
;
```

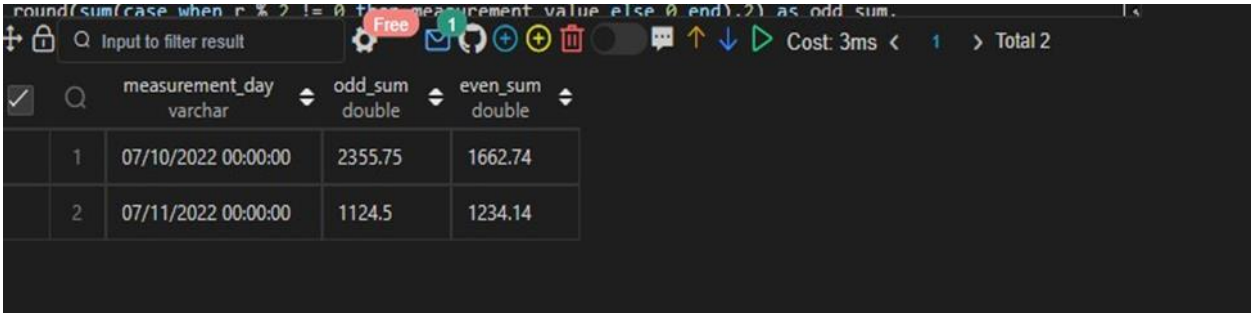


150. Assume you are given the table below containing measurement values obtained from a sensor over several days. Measurements are taken several times within a given day.

Write a query to obtain the sum of the odd-numbered and even-numbered measurements on a particular day, in two different columns.

Note that the 1st, 3rd, 5th measurements within a day are considered odd-numbered measurements and the 2nd, 4th, 6th measurements are even-numbered measurements.

```
select measurement_day, round(sum(case when r % 2 != 0 then measurement_value else 0
end),2) as odd_sum, round(sum(case when r % 2 = 0 then measurement_value else 0 end),2) as
even_sum  from (        select date_format(measurement_time, '%m/%d/%Y 00:00:00') as
measurement_day,  measurement_value, row_number() over(partition by
date(measurement_time) order by measurement_time) as r from measurements
)t group by
measurement_day;
```



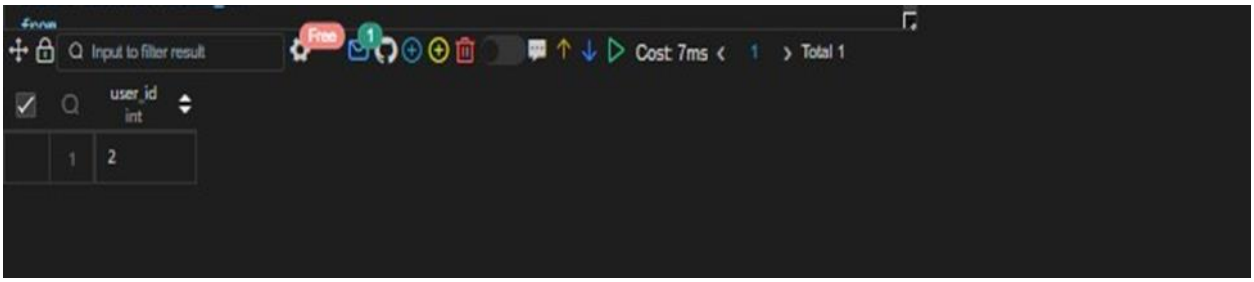
151. In an effort to identify high-value customers, Amazon asked for your help to obtain data about users who go on shopping sprees. A shopping spree occurs when a user makes purchases on 3 or more consecutive days.

List the user IDs who have gone on at least 1 shopping spree in ascending order.

Level - Medium

Hint - Use self join

```
select distinct t.user_id from (            select user_id, transaction_date as
first,      lead(transaction_date,1) over(partition by user_id order by
transaction_date) as second,  lead(transaction_date,2) over(partition by
user_id order by transaction_date) as third     from transactions
) t  where timestampdiff(day, first, second) = 1 and timestampdiff(day, second, third) = 1;
```

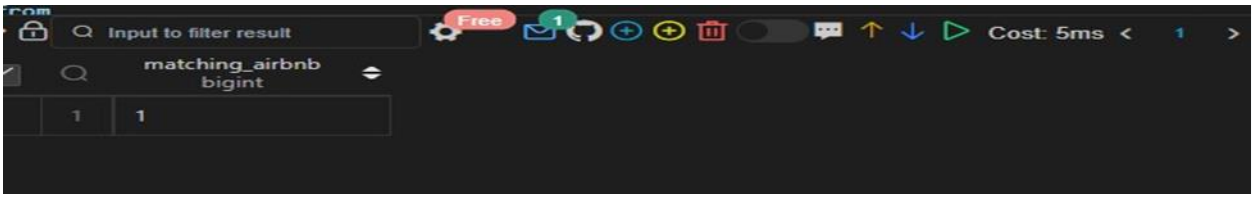


152. The Airbnb Booking Recommendations team is trying to understand the "substitutability" of two rentals and whether one rental is a good substitute for another. They want you to write a query to find the unique combination of two Airbnb rentals with the same exact amenities offered. Output the count of the unique combination of Airbnb rentals.

Level - Medium
Hint - Use unique statement
Assumptions:

- If property 1 has a kitchen and pool, and property 2 has a kitchen and pool too, it is a good substitute and represents a unique matching rental.
- If property 3 has a kitchen, pool and fireplace, and property 4 only has a pool and fireplace, then it is not a good substitute.

```
select count(t1.amenity_count) as matching_airbnb from (    select t.amenities, count(*) as
amenity_count            from     (        select rental_id, group_concat(amenity order by
amenity) amenities      from rental_amenities              group by rental_id
            )t          group by
t.amenities
)t1 where t1.amenity_count>1;
```



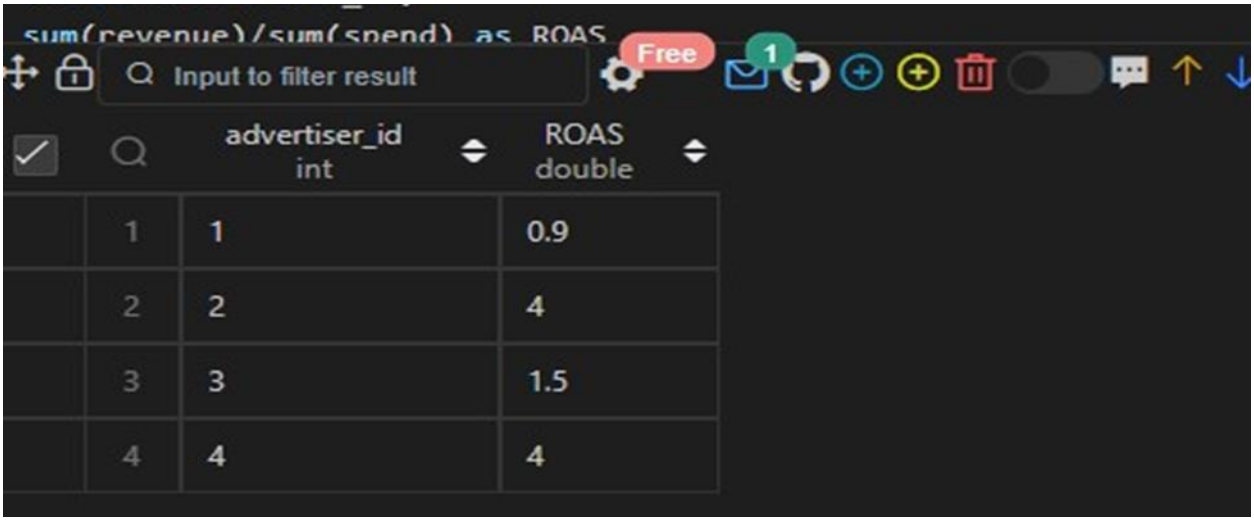
153. Google marketing managers are analysing the performance of various advertising accounts over the last month. They need your help to gather the relevant data.

Write a query to calculate the return on ad spend (ROAS) for each advertiser across all ad campaigns. Round your answer to 2 decimal places, and order your output by the advertiser_id.

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

```
WITH campaign_per_advertiser AS(
SELECT
  advertiser_id,
  SUM(spend) AS total_spend,
  SUM(revenue) AS total_revenue
FROM
  ad_campaigns
GROUP BY
  advertiser_id
)
SELECT
  advertiser_id,
  ROUND(total_revenue/total_spend, 2) AS ROAS
FROM
  campaign_per_advertiser
ORDER BY
  advertiser_id
```

;



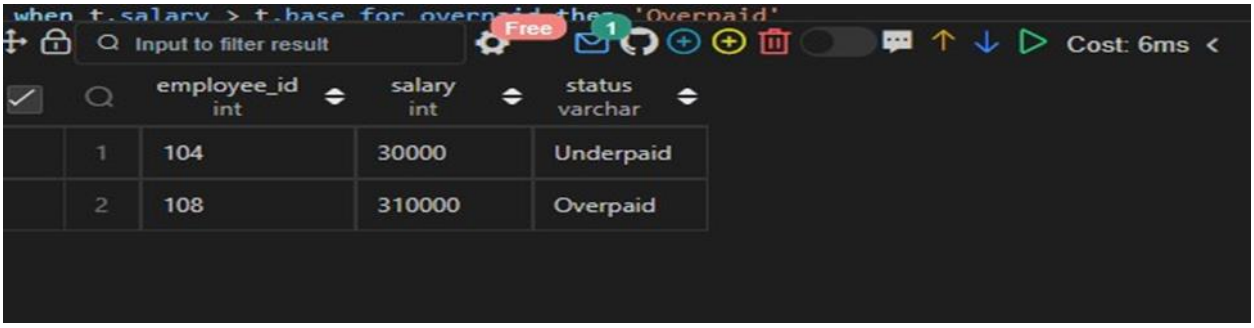
154. Your team at Accenture is helping a Fortune 500 client revamp their compensation and benefits program. The first step in this analysis is to manually review employees who are potentially overpaid or underpaid.

An employee is considered to be potentially overpaid if they earn more than 2 times the average salary for people with the same title. Similarly, an employee might be underpaid if they earn less than half of the average for their title. We'll refer to employees who are both underpaid and overpaid as compensation outliers for the purposes of this problem.

Write a query that shows the following data for each compensation outlier: employee ID, salary, and whether they are potentially overpaid or potentially underpaid (refer to Example Output below).

Hint: ROAS = Ad Revenue / Ad Spend

select t.employee_id, t.salary, case when t.salary >
t.base_for_overpaid then 'Overpaid' when t.salary <
t.base_for_underpaid then 'Underpaid' end as status from
(select employee_id, salary, 2*avg(salary) over(partition by title) as base_for_overpaid,
0.5*avg(salary) over(partition by title) as base_for_underpaid from employee_pay
)t having status is not null
order by t.employee_id;

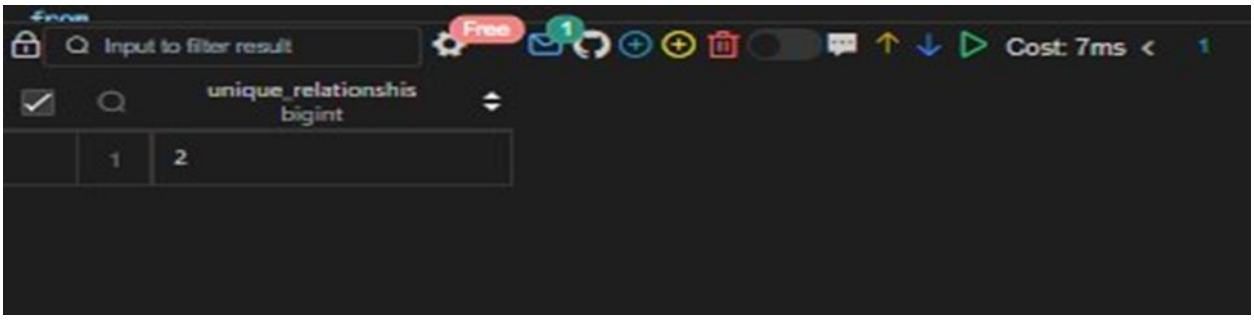


155.You are given a table of PayPal payments showing the payer, the recipient, and the amount paid. A two-way unique relationship is established when two people send money back and forth. Write a query to find the number of two-way unique relationships in this data. Assumption:

- A payer can send money to the same recipient multiple times.

Hint- Use the INTERSECT set operator.

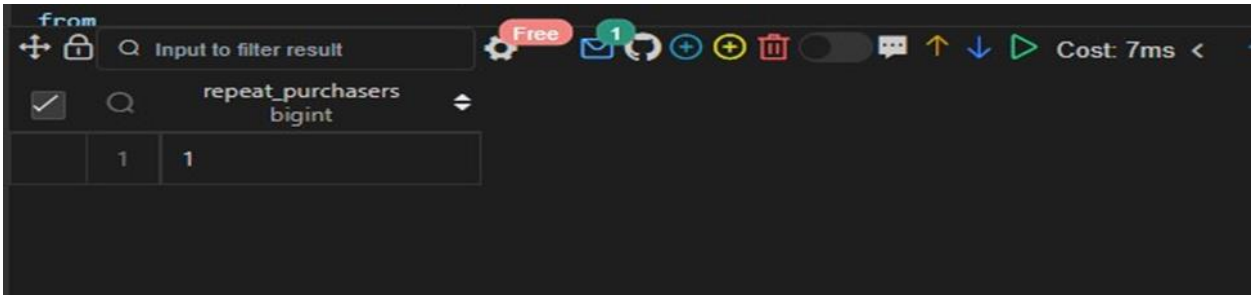
```
SELECT
  COUNT(DISTINCT user_id) AS repeat_purchasers
FROM(
  SELECT
    user_id
  FROM
    purchases
  GROUP BY
    user_id,
    product_id
  HAVING
    COUNT(DISTINCT DATE_FORMAT(purchase_date,'%Y-%m-%d')) > 1
) repeat_purchase_product
;
```



156. Assume you are given the table below containing information on user purchases. Write a query to obtain the number of users who purchased the same product on two or more different days. Output the number of unique users. *PS. On 26 Oct 2022, we expanded the purchases data set, thus the oficial output may vary from before.*

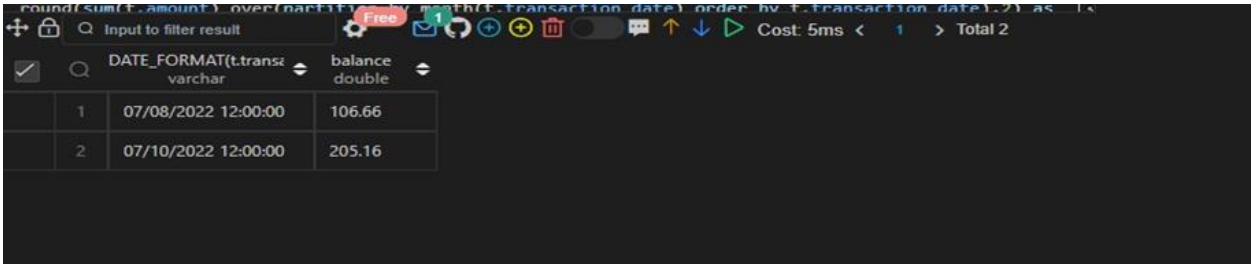
Hint- Count the distinct number of dates formatted into the DATE format in the COUNT(DISTINCT).

```
select count(distinct t.user_id) as repeat_purchasers from (          select user_id, product_id,
count(*) as c from purchases group by user_id, product_id having c > 1) t;
```



157. Say you have access to all the transactions for a given merchant account. Write a query to print the cumulative balance of the merchant account at the end of each day, with the total balance reset back to zero at the end of the month. Output the transaction date and cumulative balance. Hint-You should use CASE.

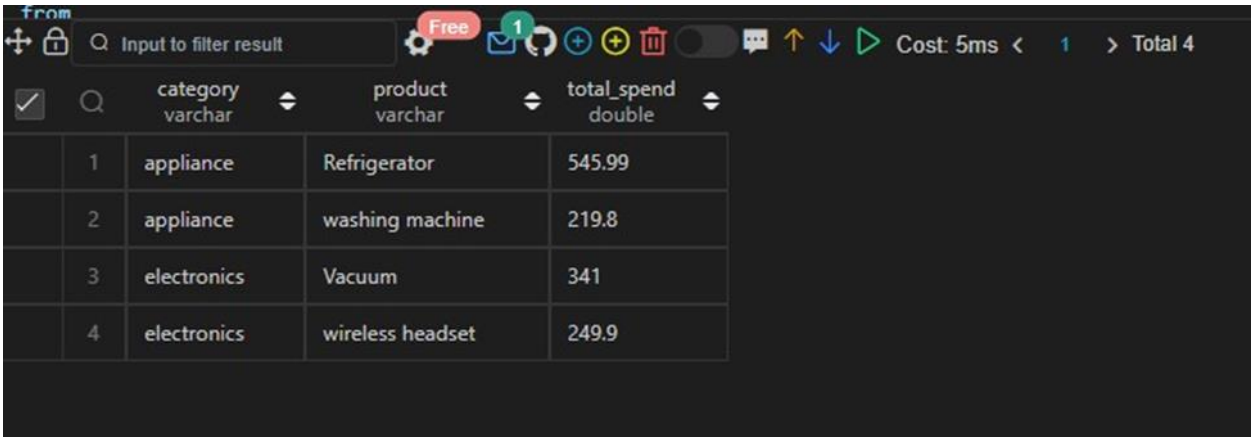
```
WITH daily_balance AS (
  SELECT
    DATE_FORMAT(transaction_date, '%Y-%m') AS transaction_month,
    DATE_FORMAT(transaction_date, '%m/%d/%Y') AS transaction_date,
    SUM(CASE
      WHEN type = 'deposit'
        THEN amount
      WHEN type = 'withdrawal'
        THEN amount * (-1)
    END) AS balance

  FROM
    transactions
  GROUP BY
    DATE_FORMAT(transaction_date, '%m/%d/%Y'),
    DATE_FORMAT(transaction_date, '%Y-%m')
)
SELECT
  CONCAT(transaction_date , ' 12:00:00') AS transaction_date,
  SUM(balance) OVER( PARTITION BY transaction_month ORDER BY transaction_date) AS
balance
FROM
  daily_balance
;
```



158. Assume you are given the table below containing information on Amazon customers and their spend on products belonging to various categories. Identify the top two highest-grossing products within each category in 2022. Output the category, product, and total spend.

Hint- Use where ,and, group by .

```
WITH category_product_spend_2022 AS(
  SELECT
    category,
    product,
    sum(spend) AS total_spend,
    DENSE_RANK() OVER(PARTITION BY category ORDER BY sum(spend)) AS serial
  FROM
    product_spend
  WHERE
    DATE_FORMAT(transaction_date, '%Y') = '2022'
  GROUP BY
    category,
    product
)
SELECT
  category,
  product,
  total_spend
FROM
  category_product_spend_2022
WHERE
  serial <= 2;
```



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

159. Facebook is analysing its user signup data for June 2022. Write a query to generate the churn rate by week in June 2022. Output the week number (1, 2, 3, 4, ...) and the corresponding churn rate rounded to 2 decimal places.

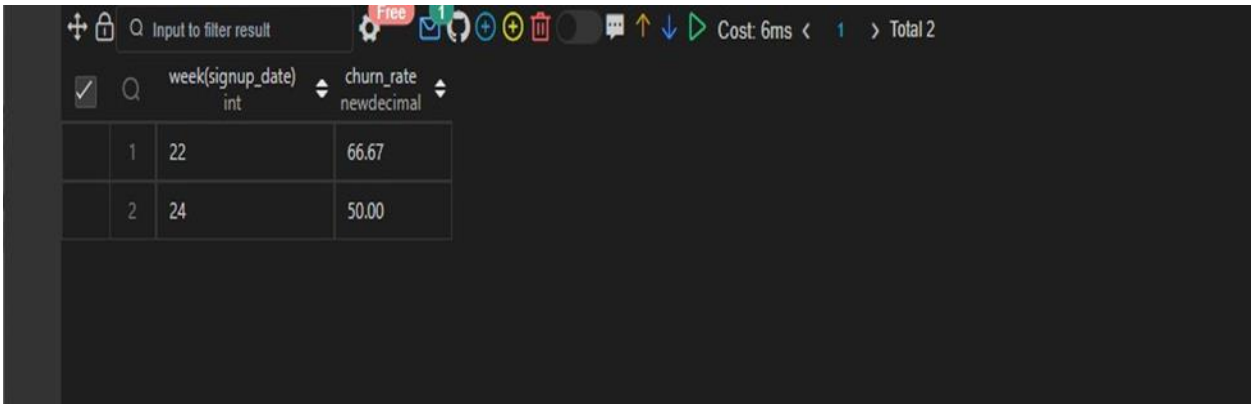
For example, week number 1 represents the dates from 30 May to 5 Jun, and week 2 is from 6 Jun to 12 Jun.

Hint- Use Extract.

Assumptions:

- If the last_login date is within 28 days of the signup_date, the user can be considered churned.
- If the last_login is more than 28 days after the signup date, the user didn't churn.

select week(signup_date),
round(100*sum(case when timestampdiff(day,signup_date,last_login) <= 28 then 1 else 0 end)/count(*),2) as churn_rate from users group by week(signup_date);



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