SQL PRACTICE

1. Write a solution to find the ids of products that are both low fat and recyclable.
Return the result table in any order.
Table - Products
Columns - product_id (int), low_fats (enum), recyclable (enum)

```sql
select product_id from products
where low_fats='Y' and recyclable='Y';
```

Accepted Runtime: 327 ms

Your input

```
{"headers":{"Products":["product_id","low_fats","recyclable"]},"rows":{"Products":
[["0","Y","N"],["1","Y","Y"],["2","N","Y"],["3","Y","Y"],["4","N","N"]]}}
```

Output

```
{"headers": ["product_id"], "values": [[1], [3]]}
```

Expected

```
{"headers": ["product_id"], "values": [[1], [3]]}
```

2. Find the names of the customer that are not referred by the customer with id = 2.
Return the result table in any order.
Table Customer
Colulmns- id (int), name (varchar), referee_id (int)

```sql
select name
from Customer
where referee_id <> 2 or referee_id is null;
```

Accepted Runtime: 173 ms

Your input

```
{"headers":{"Customer":["id","name","referee_id"]},"rows":{"Customer":[[1,"Will",null],
[2,"Jane",null],[3,"Alex",2],[4,"Bill",null],[5,"Zack",1],[6,"Mark",2]]}}
```

Output

```
{"headers": ["name"], "values": [["Will"], ["Jane"], ["Bill"], ["Zack"]]}
```

Expected

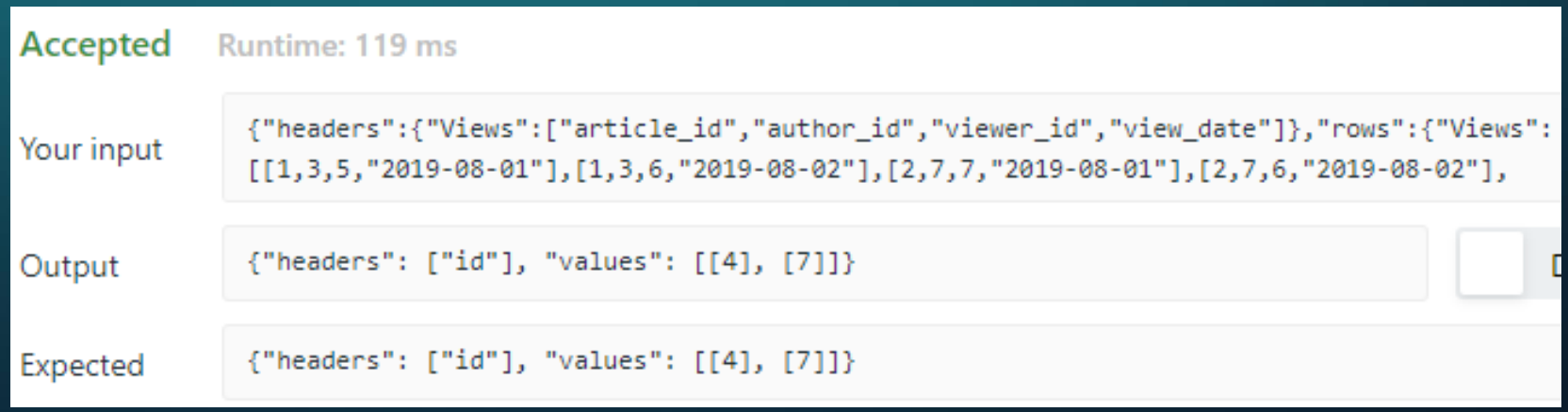
```
{"headers": ["name"], "values": [["Will"], ["Jane"], ["Bill"], ["Zack"]]}
```

3. A country is big if:
it has an area of at least three million (i.e., 3000000 km2), or
it has a population of at least twenty-five million (i.e., 25000000). Write a solution to find the name, population, and area of the big countries. Return the result table in any order.
Table-World
Columns- name (varchar), continent (varchar), area (int), population (int), Gdp (bigint)

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

4. Write a solution to find all the authors that viewed at least one of their own articles.Return the result table sorted by id in ascending order.
Note that equal author_id and viewer_id indicate the same person.
Table- Views
Columns -article_id (int), author_id (int), Viewer_id (int), view_date (date)

```sql
select
distinct author_id as id
from Views
where author_id = Viewer_id
order by author_id;
```



5. Write a solution to find the IDs of the invalid tweets. The tweet is invalid if the number of characters used in the content of the tweet is strictly greater than 15. Return the result table in any order.
Table- Tweets
Columns - tweet_id (int), content (varchar)

```sql
select
tweet_id
from Tweets
where length(content)>15;
```

Accepted Runtime: 120 ms

Your input

```
{"headers":{"Tweets":["tweet_id","content"]},"rows":{"Tweets":[[1,"Vote for Biden"],
[2,"Let us make America great again!"]]}}
```

Output

```
{"headers": ["tweet_id"], "values": [[2]]}
```

Expected

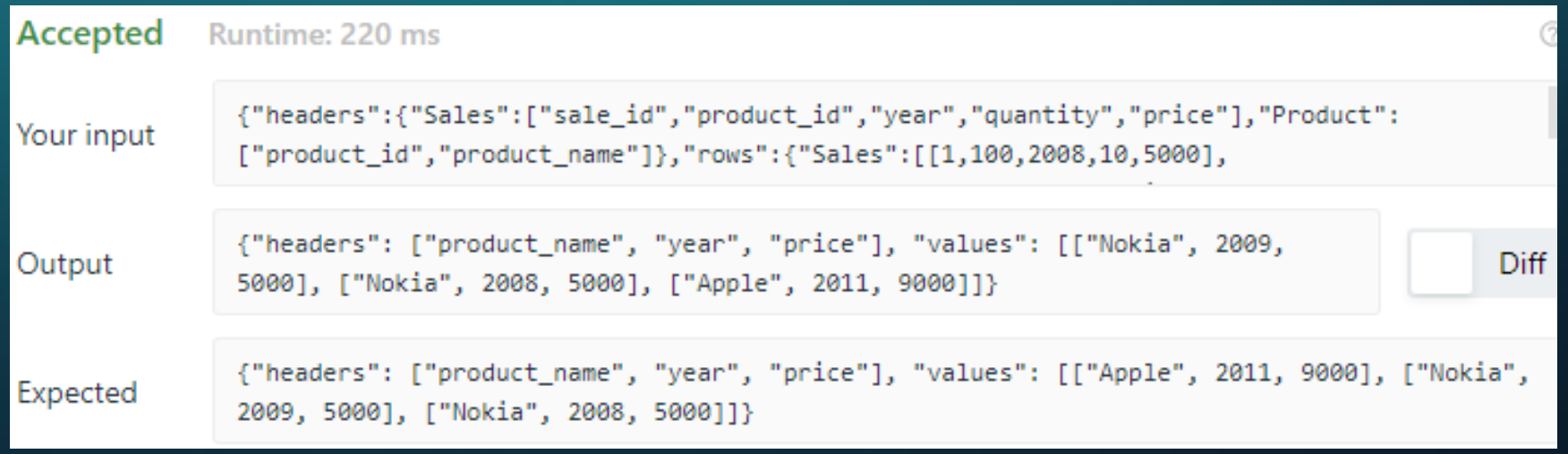
```
{"headers": ["tweet_id"], "values": [[2]]}
```

6. Write a solution to show the unique ID of each user, If a user does not have a unique ID replace just show null. Return the result table in any order.
Table1- Employees, Columns - id, name
Table2 - EmployeeUNI, Columns- id, unique_id

```sql
select
eu.unique_id, e.name
from EmployeeUNI  eu
right join Employees e on e.id = eu.id;
```

7. Write a solution to report the product_name, year, and price for each sale_id in the Sales table. Return the resulting table in any order.
Table1- Sales, Columns-sale_id (int), product_id (int), year (int), quantity (int), price (int)
Table2 - Product, Columns- product_id (int), product_name (varchar)

```sql
select
p.product_name, s.year, s.price
from sales s
join product p on s.product_id = p.product_id ;
```



8. Write a solution to find the IDs of the users who visited without making any transactions and the number of times they made these types of visits. Return the result table sorted in any order.
Table1- Visits, columns- visit_id (int), customer_id (int_
Table2- Transactions, Columns- transaction_id (int), visit_id (int), amount (int)

Sol-1

```sql
select
customer_id,
count(visit_id) as count_no_trans
from Visits v
where not exists (select visit_id from transactions t WHERE t.visit_id = v.visit_id)
group by customer_id ;
```

Sol-2

```sql
select
customer_id,
count(visit_id) as count_no_trans
from Visits
where visit_id not in (select visit_id from transactions)
group by customer_id ;
```

Sol-3

```sql
select
v.customer_id,
count(v.visit_id) as count_no_trans
from Visits v
left join Transactions t on v.visit_id = t.Visit_id
where t.transaction_id is NULL
group by v.customer_id ;
```

Accepted Runtime: 284 ms

Your input

```
{"headers":{"Visits":["visit_id","customer_id"],"Transactions":
["transaction_id","visit_id","amount"]},"rows":{"Visits":[[1,23],[2,9],[4,30],[5,54],
```

Output

```
{"headers": ["customer_id", "count_no_trans"], "values": [[30, 1], [96, 1],
[54, 2]]}
```

Expected

```
{"headers": ["customer_id", "count_no_trans"], "values": [[30, 1], [96, 1], [54, 2]]}
```

9. Write a solution to find all dates' Id with higher temperatures compared to its previous dates (yesterday). Return the result table in any order.
Table- Weather, Columns- id (int), recordDate (date), temperature (int)

```sql
SELECT w1.id
FROM Weather w1, Weather w2
WHERE DATEDIFF(w1.recordDate, w2.recordDate) = 1
AND w1.temperature > w2.temperature;
```

```sql
SELECT T1.Id
FROM Weather T1
JOIN Weather T2
ON T1.Id=(T2.Id+1)
WHERE T1.Temperature>T2.Temperature
```

Accepted Runtime: 133 ms

Your input

```
{"headers": {"Weather": ["id", "recordDate", "temperature"]}, "rows": {"Weather": [[1, "2015-01-01", 10], [2, "2015-01-02", 25], [3, "2015-01-03", 20], [4, "2015-01-04", 30]]}}
```

Output

```
{"headers": ["id"], "values": [[2], [4]]}
```

Expected

```
{"headers": ["Id"], "values": [[2], [4]]}
```

10. There is a factory website that has several machines each running the same number of processes. Write a solution to find the average time each machine takes to complete a process. The time to complete a process is the 'end' timestamp minus the 'start' timestamp. The average time is calculated by the total time to complete every process on the machine divided by the number of processes that were run.
The resulting table should have the machine_id along with the average time as processing_time, which should be rounded to 3 decimal places. Return the result table in any order.
Table- Activity
Columns- machine_id (int), process_id (int), activity_type (enum), timestamp (float)

Sol-1

```sql
SELECT
a.machine_id,
ROUND(
AVG(CASE WHEN a.activity_type = 'end' THEN a.timestamp END) -
AVG(CASE WHEN a.activity_type = 'start' THEN a.timestamp END), 3) AS processing_time
FROM activity a
GROUP BY a.machine_id;
```

Sol-2

```sql
select a1.machine_id,
round(avg(a2.timestamp-a1.timestamp), 3) as processing_time
from Activity a1
join Activity a2
on a1.machine_id=a2.machine_id and a1.process_id=a2.process_id
and a1.activity_type='start' and a2.activity_type='end'
group by a1.machine_id
```

Accepted Runtime: 143 ms

Your input

{"headers":{"Activity":["machine_id","process_id","activity_type","timestamp"]},"rows":
{"Activity":[[0,0,"start",0.712],[0,0,"end",1.52],[0,1,"start",3.14],[0,1,"end",4.12],

Output

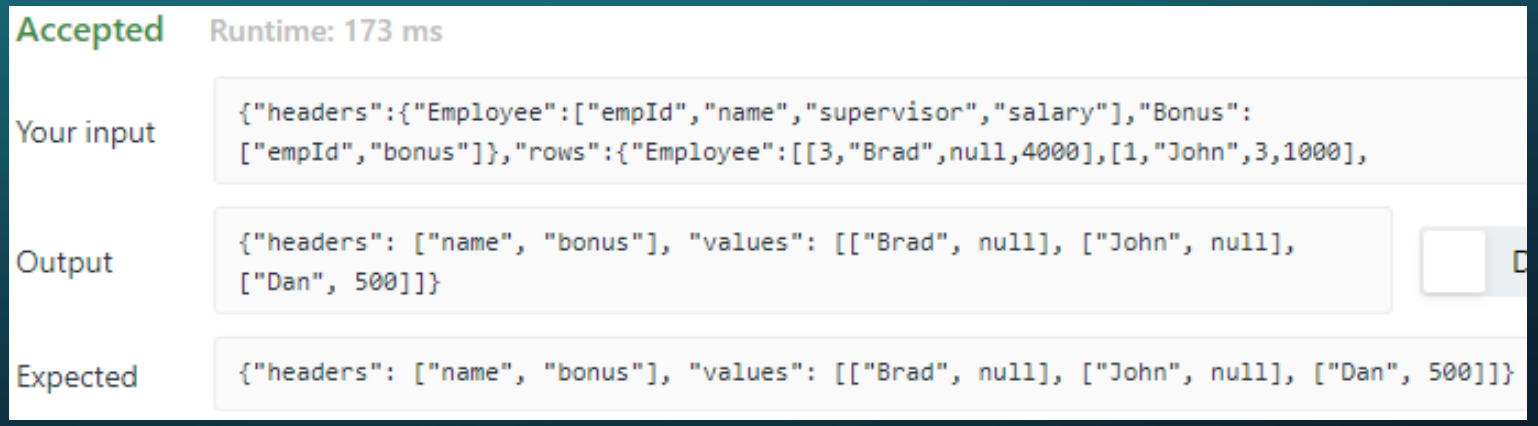
{"headers": ["machine_id", "processing_time"], "values": [[0, 0.894], [1,
0.995], [2, 1.456]]}

Expected

{"headers": ["machine_id", "processing_time"], "values": [[0, 0.894], [1, 0.995], [2,
1.456]]}

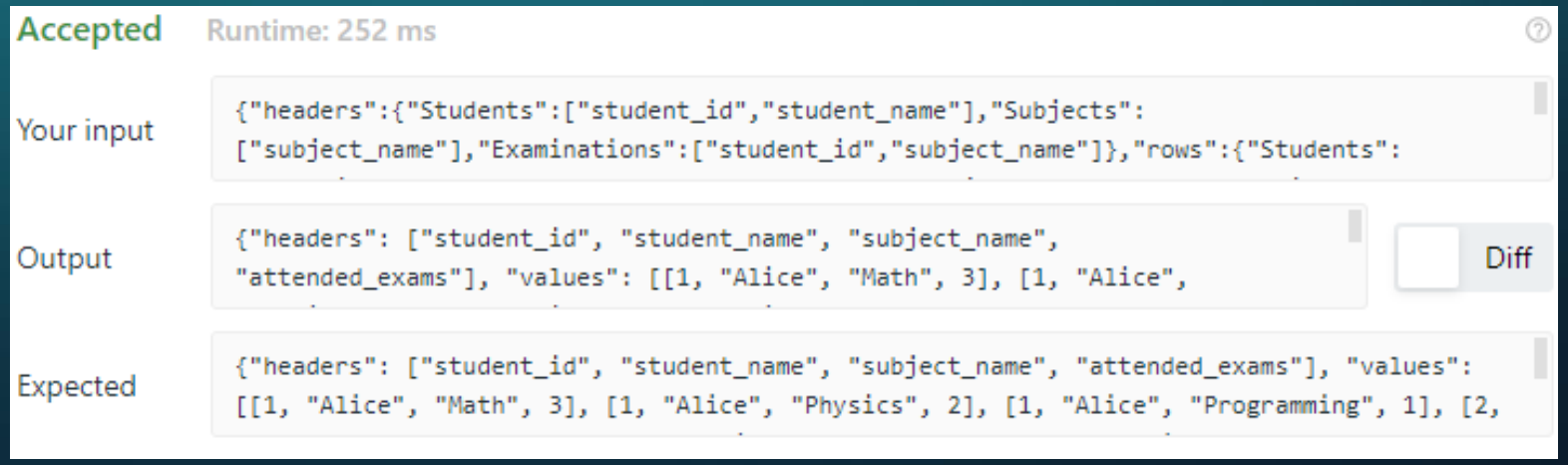
11. Write a solution to report the name and bonus amount of each employee with a bonus less than 1000. Return the result table in any order.
Table1 - Employee, Columns- EmpId (int), name (varchar), supervisor (int), salary (int)
Table2- Bonus, Columns - EmpId (int), bonus (int)

```sql
SELECT
name, bonus
from employee e
left join bonus b on e.empId = b.empId
where bonus <1000 or bonus is null;
```



12. Write a solution to find the number of times each student attended each exam. Return the result table ordered by student_id and subject_name.
Table1 - Students, Columns- student_id (int), student_name (varchar)
Table2 - Subjects, Columns- subject_name (varchar)
Table3 - Examinations, Columns - student_id (int), subject_name (varchar)

```sql
select
s.student_id, s.student_name,
sb.subject_name,
count(e.student_id) as attended_exams
from students s
cross join Subjects sb
left join examinations e on s.student_id = e.student_id
and sb.subject_name = e.subject_name
group by s.student_id, s.student_name, sb.subject_name
order by s.student_id, sb.subject_name;
```



13. Write a solution to find managers with at least five direct reports. Return the result table in any order.
Table-Employee, Columns- id (int), name (varchar), department (varchar), managerId (int)

Sol-1

```sql
select
e1.name
from Employee e1
join Employee e2 on e1.id = e2.managerId
group by e1.name
having count(e2.managerId)>=5;
```

Sol-2

```sql
select
name
from employee
where id IN (
select managerid from employee
group by managerid
having count(*)>=5);
```

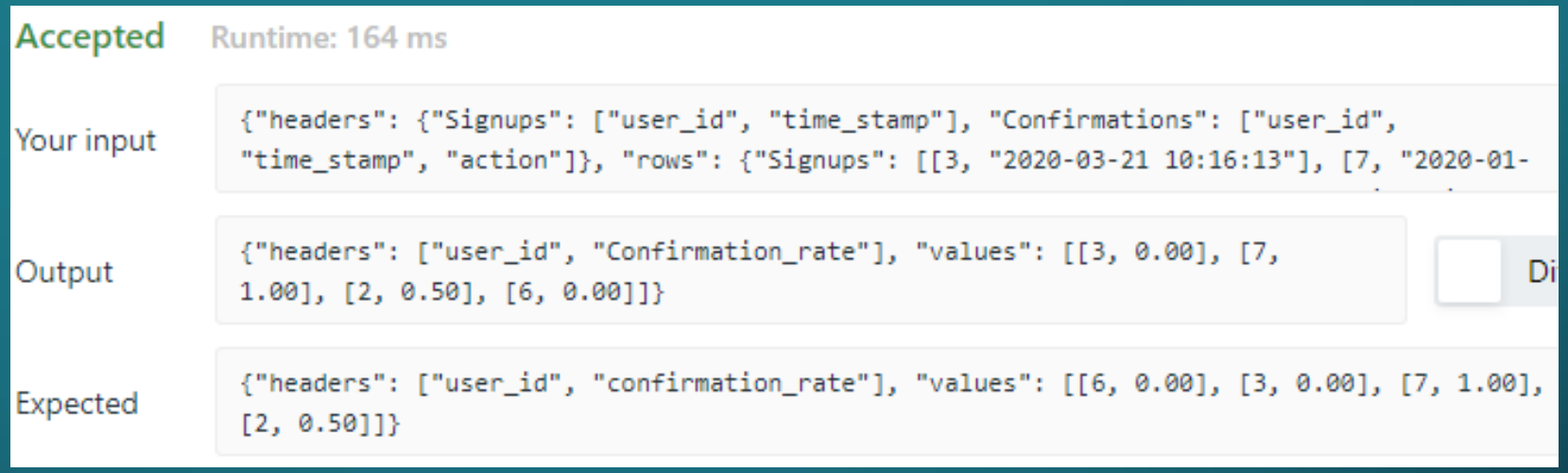
14. Write a solution to find the confirmation rate of each user. Return the result table in any order. The confirmation rate of a user is the number of 'confirmed' messages divided by the total number of requested confirmation messages. The confirmation rate of a user that did not request any confirmation messages is 0. Round the confirmation rate to two decimal places.
Table1 - Signups, Columns- user_id (int), time_stamp (datetime)
Table2 - Confirmations, Columns - user_id (int), time_stamp (datetime), action (ENUM)

Sol-1

```
select
s.user_id,
round(avg(if(c.action = 'Confirmed',1,0)),2) as Confirmation_rate
from signups s
left join confirmations c on s.user_id = c.user_id
group by s.user_id;
```

Sol-2

```
Select s.user_id,
Round(avg(case when c.action='Confirmed' then 1 else 0 end), 2) as confirmation_rate
From Signups s
left join Confirmations c on s.user_id = c.user_id
group by s.user_id;
```



Krishna Sameera Kota

15. Write a solution to report the movies with an odd-numbered ID and a description that is not "boring". Return the result table ordered by rating in descending order.
Table- Cinema
Columns- id (int), movie (varchar), description (varchar), rating (float)

```sql
select
id, movie, description, rating
from Cinema
where id%2 != 0
and description != 'boring'
order by rating desc;
```

Accepted Runtime: 114 ms

Your input

{"headers":{"cinema":["id", "movie", "description", "rating"]},"rows":{"cinema":[[1, "War", "great 3D", 8.9], [2, "Science", "fiction", 8.5], [3, "irish", "boring", 6.2], [4,

Output

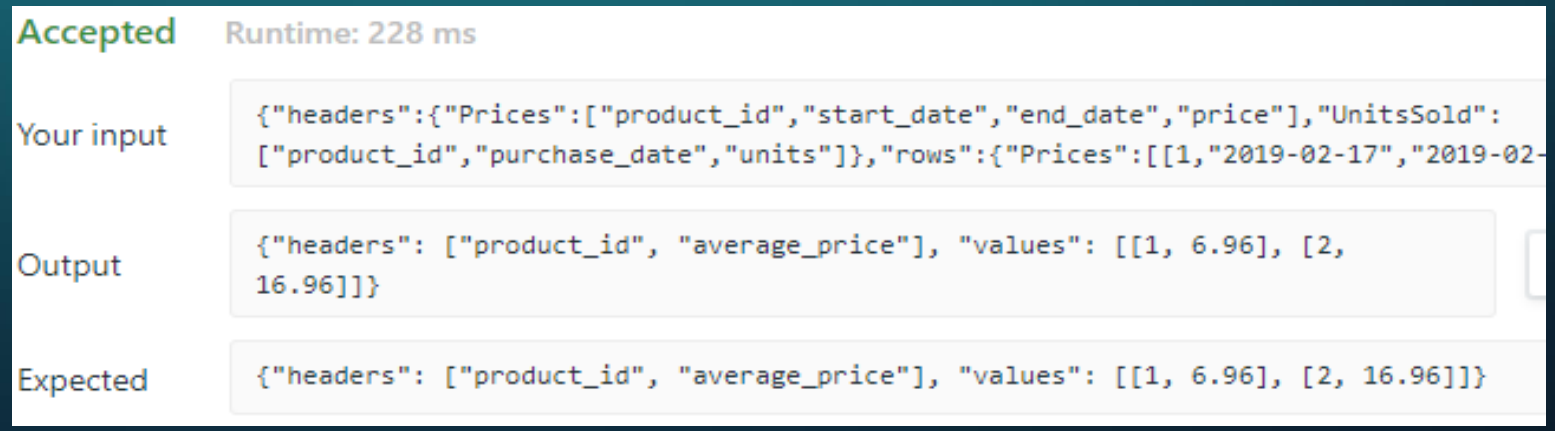
{"headers": ["id", "movie", "description", "rating"], "values": [[5, "House card", "Interesting", 9.1], [1, "War", "great 3D", 8.9]]}

Expected

{"headers": ["id", "movie", "description", "rating"], "values": [[5, "House card", "Interesting", 9.1], [1, "War", "great 3D", 8.9]]}

16. 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.
Table - Transactions
Columns - id (int), country (varchar), state (enum), amount (int), trans_date (date)
The state column is an enum of type ["approved", "declined"].

```sql
SELECT p.product_id,
IFNULL(ROUND(SUM(p.price*u.units)/SUM(u.units),2),0) AS average_price
FROM Prices AS p
LEFT JOIN UnitsSold AS u
ON p.product_id = u.product_id AND u.purchase_date
BETWEEN p.start_date and p.end_date
GROUP BY p.product_id;
```



17. 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.
Table1 - Project, Columns - project_id (int), employee_id (int)
Table2 - Employee, Columns - employee_id (int), name (varchar), experience_years (int)

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

Accepted Runtime: 507 ms

Your input

```
{"headers":{"Project":["project_id","employee_id"],"Employee":
["employee_id","name","experience_years"]},"rows":{"Project":[[1,1],[1,2],[1,3],[2,1],
```

Output

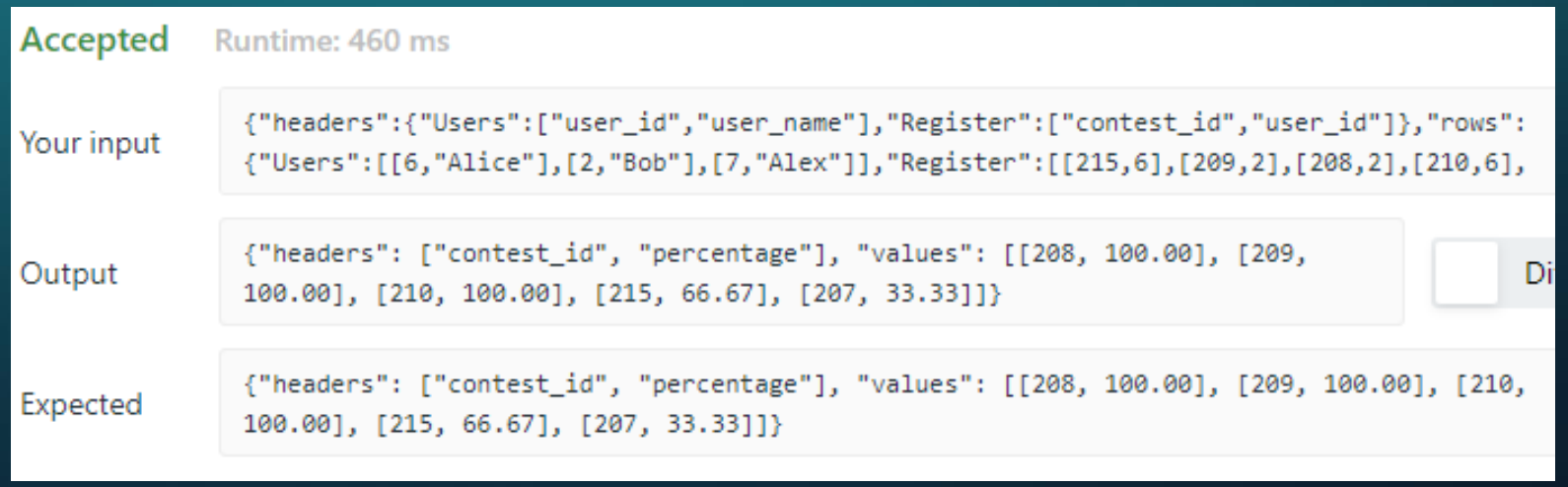
```
{"headers": ["project_id", "average_years"], "values": [[1, 2.00], [2,
2.50]]}
```

Expected

```
{"headers": ["project_id", "average_years"], "values": [[1, 2.00], [2, 2.50]]}
```

18. Write a solution to find the percentage of the users registered in each contest rounded to two decimals. Return the result table ordered by percentage in descending order. In case of a tie, order it by contest_id in ascending order.
Table1 - Users, Columns- user_id (int), user_name (int)
Table2 - Register, Columns - contest_id (int), user_id (int)

```sql
select
contest_id,
round(count(distinct user_id) *100 /
     (select count(user_id) from users), 2) as percentage
from register
group by contest_id
order by percentage desc, contest_id asc;
```



19. Write a solution to find each query_name, the quality and poor_query_percentage. Both quality and poor_query_percentage should be rounded to 2 decimal places. Return the result table in any order.
We define query quality as: The average of the ratio between query rating and its position.
We also define poor query percentage as: The percentage of all queries with rating less than 3.
Table - Queries, Columns - query_name (varchar), result (varchar), position (int), rating (int)

Sol-1 using IF

```
select
query_name,
round(AVG(rating/position),2) as quality,
round(SUM(IF(rating<3, 100, 0))/
      COUNT(query_name),2) as poor_query_percentage
from Queries
where query_name is not null
group by query_name;
```

Sol-2 using CASE

```
select
query_name,
round(AVG(rating/position),2) as quality,
round(SUM(CASE when rating<3 then 100 else 0 END)/
      COUNT(query_name),2) as poor_query_percentage
from Queries
where query_name is not null
group by query_name;
```

Accepted Runtime: 133 ms

Your input

```
{"headers":{"Queries":["query_name","result","position","rating"]},"rows":{"Queries":
[["Dog","Golden Retriever",1,5],["Dog","German Shepherd",2,5],["Dog","Mule",200,1],
```

Output

```
{"headers": ["query_name", "quality", "poor_query_percentage"], "values":
[["Dog", 2.50, 33.33], ["Cat", 0.66, 33.33]]}
```

Expected

```
{"headers": ["query_name", "quality", "poor_query_percentage"], "values": [["Dog", 2.50,
33.33], ["Cat", 0.66, 33.33]]}
```

20. 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.

Table- Transactions

Columns- id (int), country (varchar), state (enum), amount (int), trans_date (date)

The state column is an enum of type ["approved", "declined"].

```sql
select
date_format(trans_date, "%Y-%m") as Month,
country,
count(id) as trans_count,
sum(CASE when state='approved' then 1 else 0 end) as approved_count,
sum(amount) as trans_total_amount,
sum(CASE when state='approved' then amount else 0 end) as approved_total_amount
from Transactions
group by month, country ;
```

Accepted Runtime: 177 ms

Your input

```
{"headers":{"Transactions":["id","country","state","amount","trans_date"]},"rows":
{"Transactions":[[121,"US","approved",1000,"2018-12-18"],[122,"US","declined",2000,"2018-
```

Output

```
{"headers": ["Month", "country", "trans_count", "approved_count",
"trans_total_amount", "approved_total_amount"], "values": [["2018-12", "US",
```

Expected

```
{"headers": ["month", "country", "trans_count", "approved_count", "trans_total_amount",
"approved_total_amount"], "values": [["2018-12", "US", 2, 1, 3000, 1000], ["2019-01",
```

21. Write a solution to find the percentage of immediate orders in the first orders of all customers, rounded to 2 decimal places.
If the customer's preferred delivery date is the same as the order date, then the order is called immediate; otherwise, it is called scheduled. The first order of a customer is the order with the earliest order date that the customer made. It is guaranteed that a customer has precisely one first order.

Table - Delivery
Columns - delivery_id (int), customer_id (int), order_date (date), customer_pref_delivery_date (date)

Sol-1
Using CTE

```
WITH CTE AS(
select
delivery_id,
customer_id,
order_date,
customer_pref_delivery_date,
row_number() over (partition by customer_id order by order_date) as earliest_orderdate
from Delivery
)

select
round(sum(case when order_date = customer_pref_delivery_date then 1 else 0 END)*100/
      count(distinct customer_id),2) as immediate_percentage
 from CTE
 where earliest_orderdate=1;
```

Sol-2 Using Subquery

```sql
select
round(sum(case when order_date = customer_pref_delivery_date then 1 else 0 END)*100/
     count(distinct customer_id),2) as Immediate_percentage
from Delivery
where (customer_id, order_date) IN (
    select
    customer_id,
    min(order_date) as earliest_orderdate
    from Delivery
    group by customer_id)
```

Accepted Runtime: 305 ms

Your input


```
{"headers":{"Delivery":
["delivery_id","customer_id","order_date","customer_pref_delivery_date"]},"rows":
```


Output


```
{"headers": ["immediate_percentage"], "values": [[50.00]]}
```


Expected


```
{"headers": ["immediate_percentage"], "values": [[50.000000000000]]}
```

22. Write a solution 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.

Table - Activity
Columns - player_id (int), device_id (int), event_date (date), games_played (int)

Sol-1 Using CTE

```sql
with first_login as (
select
player_id,
event_date,
row_number() over (partition by player_id order by event_date) as rn
from Activity
)

select
round(count(*)/ (select count(distinct player_id) from activity),2) as fraction
from first_login f
join Activity a on f.player_id = a.player_id
and f.event_date = date_sub(a.event_date, interval 1 day)
where f.rn = 1;
```

Sol-2 Using Subquery

```sql
Select
round(count(distinct player_id) /
      (select count(distinct player_id) from Activity),2) as fraction
from Activity
where (player_id, Date_sub(event_date, interval 1 day)) IN (
       Select
       player_id,
       min(event_date) as first_login
       from Activity
       group by player_id)
```

Accepted Runtime: 122 ms

Your input

{"headers":{"Activity":["player_id","device_id","event_date","games_played"]},"rows":
{"Activity":[[1,2,"2016-03-01",5],[1,2,"2016-03-02",6],[2,3,"2017-06-25",1],[3,1,"2016-

Output

{"headers": ["fraction"], "values": [[0.33]]}

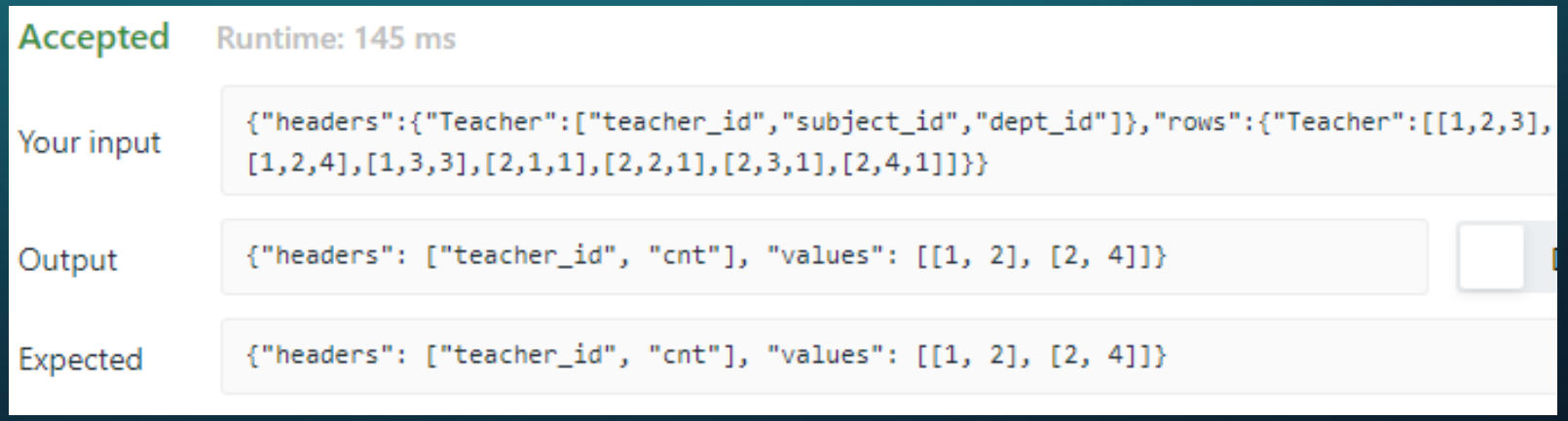
Expected

{"headers": ["fraction"], "values": [[0.33]]}

23. Write a solution to calculate the number of unique subjects each teacher teaches in the university. Return the result table in any order.

Table - Teacher
Columns - teacher_id (int), subject_id (int), dept_id (int

```sql
Select
teacher_id,
count(distinct subject_id) as cnt
from Teacher
group by teacher_id;
```



24. Write a solution to find the daily active user count for a period of 30 days ending 2019-07-27 inclusively. A user was active on someday if they made at least one activity on that day. Return the result table in any order.

Table - Activity
Columns - user_id (int), session_id (int), activity_date (date), activity_type (enum)

```sql
Select
activity_date as day,
Count(distinct user_id) as active_users
from Activity
where activity_date > '2019-06-27' and activity_date <= '2019-07-27'
group by day;
```

Accepted Runtime: 204 ms

Your input

```
{"headers":{"Activity":["user_id","session_id","activity_date","activity_type"]},"rows":
{"Activity":[[1,1,"2019-07-20","open_session"],[1,1,"2019-07-20","scroll_down"],
```

Output

```
{"headers": ["day", "active_users"], "values": [["2019-07-20", 2], ["2019-07-
21", 2]]}
```

Expected

```
{"headers": ["day", "active_users"], "values": [["2019-07-20", 2], ["2019-07-21", 2]]}
```

Notes:
Leetcode provided several ways to select a specific range of dates for the above query. After WHERE condition, we can use below ways to solve the query.

1. activity_date BETWEEN '2019-06-28' AND '2019-07-27’

2. DATEDIFF('2019-07-27', activity_date)<30 AND DATEDIFF('2019-07-27', activity_date)>=0

3. DATEDIFF('2019-07-27', activity_date) BETWEEN 0 AND 29

4. activity_date BETWEEN date_sub('2019-07-27', INTERVAL 29 DAY) AND '2019-07-27'

25. Write a solution to select the product id, year, quantity, and price for the first year of every product sold. Return the resulting table in any order.

Table1 - Sales
Columns - sale_id (int), product_id (int), year (int), quantity (int), price (int)
Table2- Product
Columns - product_id (int), product_name (varchar)

Sol-1
Using Subquery

```
Select
product_id,
year as first_year,
quantity,
price
from Sales
where (product_id, year) IN (
        select
        product_id,
        min(year) as first_year
        from Sales
        group by product_id);
```

Sol-2
Using CTE

```sql
With First_year as (
Select
product_id,
year,
quantity,
price,
Rank() over (partition by product_id order by year) as rn
from Sales)

Select
product_id,
year as first_year,
quantity,
price
from First_year
where rn=1;
```

Accepted Runtime: 187 ms

Your input


```
{"headers":{"Sales":["sale_id","product_id","year","quantity","price"],"Product":
["product_id","product_name"]},"rows":{"Sales":[[1,100,2008,10,5000],
```


Output


```
{"headers": ["product_id", "first_year", "quantity", "price"], "values":
[[100, 2008, 10, 5000], [200, 2011, 15, 9000]]}
```


Expected


```
{"headers": ["product_id", "first_year", "quantity", "price"], "values": [[100, 2008, 10,
5000], [200, 2011, 15, 9000]]}
```

26. Write a solution to find all the classes that have at least five students. Return the result table in any order.

Table- Courses
Columns - student (varchar), class (varchar)

```sql
Select
class
from Courses
group by class
having count(student) >=5;
```

Accepted Runtime: 172 ms

Your input

```
{"headers": {"Courses": ["student", "class"]}, "rows": {"Courses": [["A", "Math"], ["B", "English"], ["C", "Math"], ["D", "Biology"], ["E", "Math"], ["F", "Computer"], ["G",
```

Output

```
{"headers": ["class"], "values": [["Math"]]}
```

Expected

```
{"headers": ["class"], "values": [["Math"]]}
```

27. Write a solution that will, for each user, return the number of followers. Return the result table ordered by user_id in ascending order.

Table - Followers
Columns - user_id (int), follower_id (int)

```sql
Select
user_id,
count(follower_id) as followers_count
from Followers
group by user_id
order by user_id;
```

Accepted Runtime: 115 ms

Your input

```
{"headers":{"Followers":["user_id","follower_id"]},"rows":{"Followers":[["0","1"],["1","0"],["2","0"],["2","1"]]}}
```

Output

```
{"headers": ["user_id", "followers_count"], "values": [[0, 1], [1, 1], [2, 2]]}
```

Expected

```
{"headers": ["user_id", "followers_count"], "values": [[0, 1], [1, 1], [2, 2]]}
```

28. A single number is a number that appeared only once in the MyNumbers table. Find the largest single number. If there is no single number, report null.

Table - MyNumbers
Column - num (int)

```sql
SELECT MAX(num) AS num
FROM (
    SELECT num
    FROM MyNumbers
    GROUP BY num
    HAVING COUNT(num) = 1
) AS unique_numbers;
```

```sql
SELECT MAX(num) AS num
FROM MyNumbers
WHERE num IN(
    SELECT num
    FROM MyNumbers
    GROUP BY num
    HAVING COUNT(num) = 1
);
```

```sql
SELECT
(SELECT
 num
 FROM MyNumbers
 GROUP BY num
 HAVING COUNT(num) = 1
 ORDER BY num DESC
 LIMIT 1) AS num;
```

Accepted Runtime: 130 ms

Your input

```
{"headers": {"MyNumbers": ["num"]}, "rows": {"MyNumbers": [[8],[8],[3],[3],[1],[4],[5],
[6]]}}
```

Output

```
{"headers": ["num"], "values": [[6]]}
```

Expected

```
{"headers": ["num"], "values": [[6]]}
```

29. Write a solution to report the customer ids from the Customer table that bought all the products in the Product table. Return the result table in any order.

Table1 - Customer
Columns - customer_id (int), product_key (int)
Table2 - Product
Column - Product_key (int)

```sql
Select
Distinct Customer_id
from Customer
group by customer_id
having count(distinct product_key) = (select count(product_key) from product);
```

Accepted Runtime: 258 ms

Your input

```
{"headers":{"Customer":["customer_id","product_key"],"Product":["product_key"]},"rows":
{"Customer":[[1,5],[2,6],[3,5],[3,6],[1,6]],"Product":[[5],[6]]}}
```

Output

```
{"headers": ["Customer_id"], "values": [[1], [3]]}
```

Expected

```
{"headers": ["customer_id"], "values": [[1], [3]]}
```

30. Write a solution to report the ids and the names of all managers, the number of employees who report directly to them, and the average age of the reports rounded to the nearest integer. Return the result table ordered by employee_id. For this problem, we will consider a manager an employee who has at least 1 other employee reporting to them.
Table - Employees
Columns - employee_id (int), name (varchar), reports_to (int), age (int)

```sql
Select
e.employee_id,
e.name,
count(e.employee_id) as reports_count,
Round(avg(m.age)) as average_age
from Employees e
join Employees m on e.employee_id = m.reports_to
group by e.employee_id
order by e.employee_id;
```

Accepted Runtime: 233 ms

Your input

```
{"headers":{"Employees":["employee_id","name","reports_to","age"]},"rows":{"Employees":
[[9,"Hercy",null,43],[6,"Alice",9,41],[4,"Bob",9,36],[2,"Winston",null,37]]}}
```

Output

```
{"headers": ["employee_id", "name", "reports_count", "average_age"],
"values": [[9, "Hercy", 2, 39]]}
```

Expected

```
{"headers": ["employee_id", "name", "reports_count", "average_age"], "values": [[9,
"Hercy", 2, 39]]}
```

31. Write a solution to report all the employees with their primary department. For employees who belong to one department, report their only department. Return the result table in any order. Employees can belong to multiple departments. When the employee joins other departments, they need to decide which department is their primary department. Note that when an employee belongs to only one department, their primary column is 'N'.

Table - Employee
Columns - employee_id (int), department_id (int), primary_flag (varchar)

primary_flag is an ENUM (category) of type ('Y', 'N'). If the flag is 'Y', the department is the primary department for the employee. If the flag is 'N', the department is not the primary.

Sol-1
Using Subquery

```
Select
employee_id,
department_id
from Employee
where primary_flag = 'Y' or
employee_id IN (select
                employee_id from employee
                group by employee_id
                having count(*)=1 )
group by employee_id
```

Sol-2
Using Union

```sql
Select
employee_id, department_id
from Employee
where primary_flag = 'Y'
UNION
select
employee_id, department_id
from employee
group by employee_id
having count(employee_id)=1
```

Accepted Runtime: 156 ms

Your input

```
{"headers":{"Employee":["employee_id","department_id","primary_flag"]},"rows":
{"Employee":[["1","1","N"],["2","1","Y"],["2","2","N"],["3","3","N"],["4","2","N"],
```

Output

```
{"headers": ["employee_id", "department_id"], "values": [[1, 1], [2, 1], [3,
3], [4, 3]]}
```

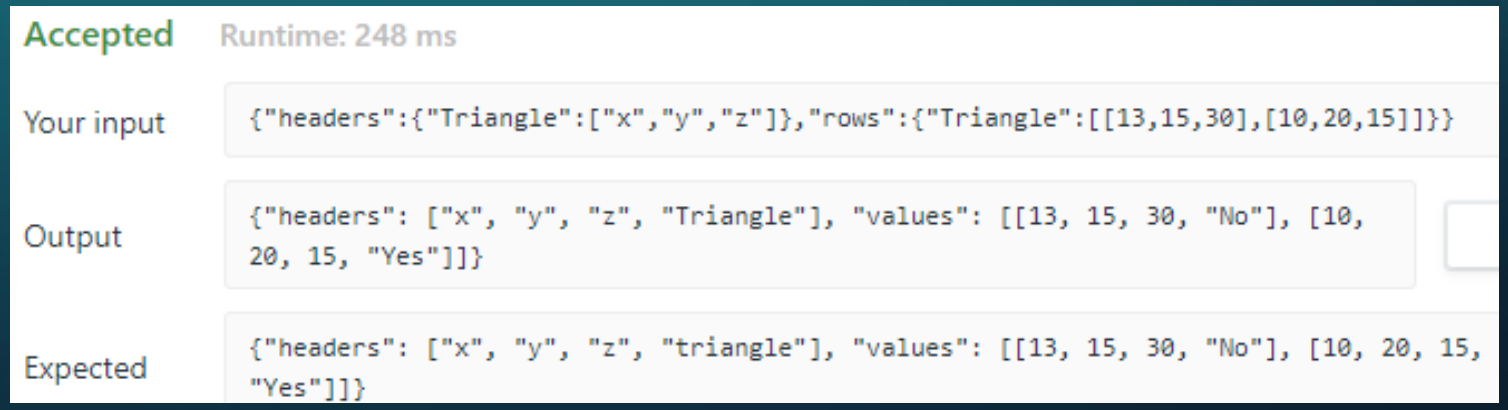
Expected

```
{"headers": ["employee_id", "department_id"], "values": [[1, 1], [2, 1], [3, 3], [4, 3]]}
```

32. Report for every three line segments whether they can form a triangle. Return the result table in any order.

Table - Triangle
Columns - x (int), y (int), z (int)
Each row of this table contains the lengths of three line segments.

```
Select
x,y,z,
CASE
WHEN x<y+z and y<x+z and z<x+y then 'Yes'
ELSE 'No'
END as Triangle
from Triangle ;
```



33. Find all numbers that appear at least three times consecutively. Return the result table in any order.

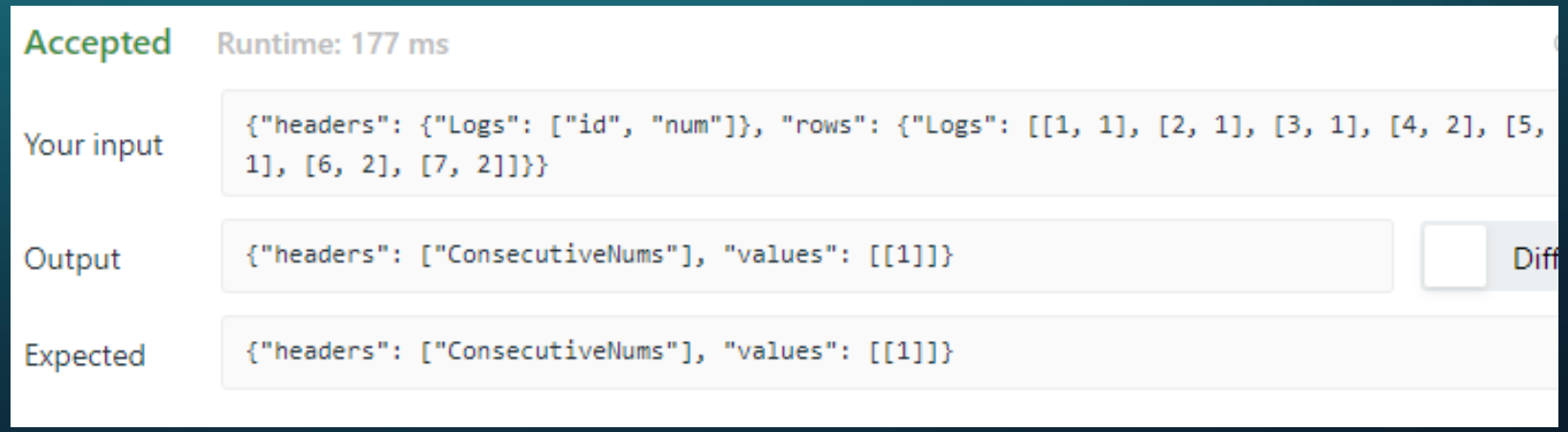
Table - Logs
Columns - id (int), num (varchar)

Sol-1
Using Self Join

```
select distinct a.num as ConsecutiveNums
from Logs a
inner join Logs b on a.id=b.id-1 and b.num=a.num
inner join Logs c on a.id=c.id-2 and c.num=a.num;
```

Sol-1
Using Lead

```
with cte as (
    select num,
    lead(num,1) over() num1,
    lead(num,2) over() num2
    from logs
)
select distinct num ConsecutiveNums
from cte
where num=num1 and num=num2
```



34. Write a solution to find the prices of all products on 2019-08-16. Assume the price of all products before any change is 10. Return the result table in any order.

Table - Products
Columns - product_id (int), new_price (int), change_date (date)

Each row of this table indicates that the price of some product was changed to a new price at some date.

```sql
SELECT
product_id,
new_price AS price
FROM Products
WHERE (product_id, change_date) IN (
    SELECT product_id, MAX(change_date)
    FROM Products
    WHERE change_date <= '2019-08-16'
    GROUP BY product_id
)
UNION
SELECT
DISTINCT product_id,
10 AS price
FROM Products
WHERE product_id NOT IN (
    SELECT product_id
    FROM Products
    WHERE change_date <= '2019-08-16'
);
```

35. Write a solution to find the person_name of the last person that can fit on the bus without exceeding the weight limit. The test cases are generated such that the first person does not exceed the weight limit.
There is a queue of people waiting to board a bus. However, the bus has a weight limit of 1000 kilograms, so there may be some people who cannot board.

Table - Queue
Columns - person_id (int), person_name (varchar), weight (int), turn (int)

The person_id and turn columns will contain all numbers from 1 to n, where n is the number of rows in the table. turn determines the order of which the people will board the bus, where turn=1 denotes the first person to board and turn=n denotes the last person to board.

Sol-1
Using Self Join

```sql
SELECT
q1.person_name
FROM Queue q1
JOIN Queue q2 ON q1.turn >= q2.turn
GROUP BY q1.turn
HAVING SUM(q2.weight) <= 1000
ORDER BY SUM(q2.weight) DESC
LIMIT 1;
```

Sol-2
Using CTE

```sql
WITH CTE AS (
    SELECT
    turn, person_name, weight,
    SUM(weight) OVER(ORDER BY turn) AS total_weight
    FROM Queue
    ORDER BY turn
)
SELECT person_name
FROM Queue q
WHERE q.turn = (SELECT MAX(turn) FROM CTE WHERE total_weight <= 1000);
```

Accepted Runtime: 119 ms

Your input

```
{"headers":{"Queue":["person_id","person_name","weight","turn"]},"rows":{"Queue":
[[5,"Alice",250,1],[4,"Bob",175,5],[3,"Alex",350,2],[6,"John Cena",400,3],
```

Output

```
{"headers": ["person_name"], "values": [["John Cena"]]}
```

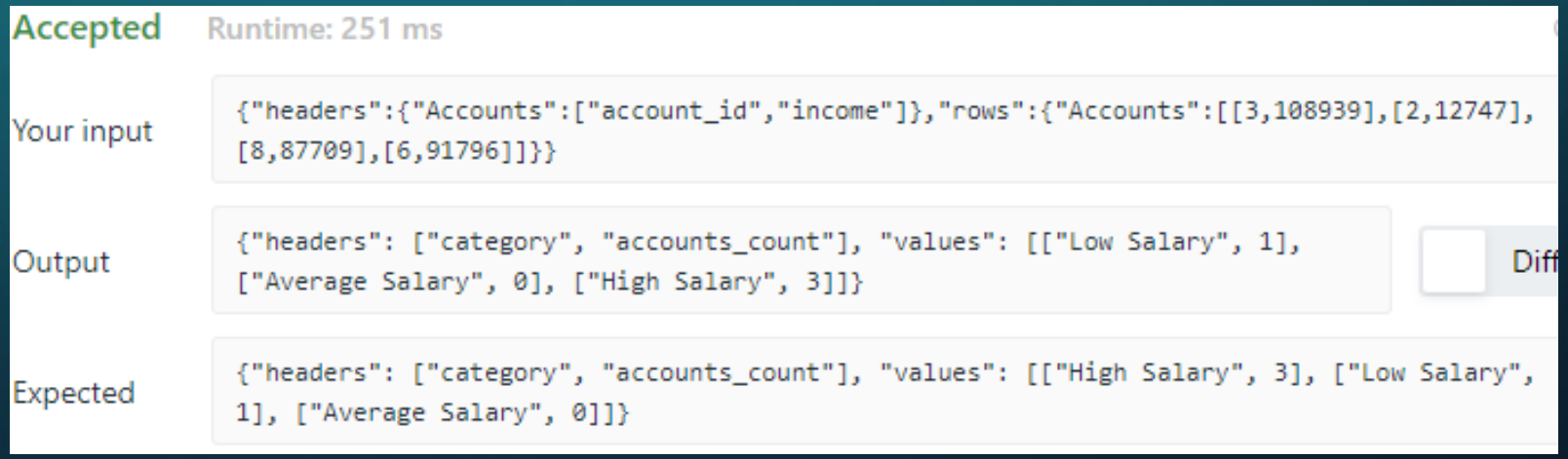
Expected

```
{"headers": ["person_name"], "values": [["John Cena"]]}
```

36. Write a solution to calculate the number of bank accounts for each salary category. The salary categories are:

"Low Salary": All the salaries strictly less than $20000.
"Average Salary": All the salaries in the inclusive range [$20000, $50000].
"High Salary": All the salaries strictly greater than $50000.
The result table must contain all three categories. If there are no accounts in a category, return 0.
Return the result table in any order.

Table - Accounts
Columns - account_id (int), income (int)



---------- Krishna Sameera Kota

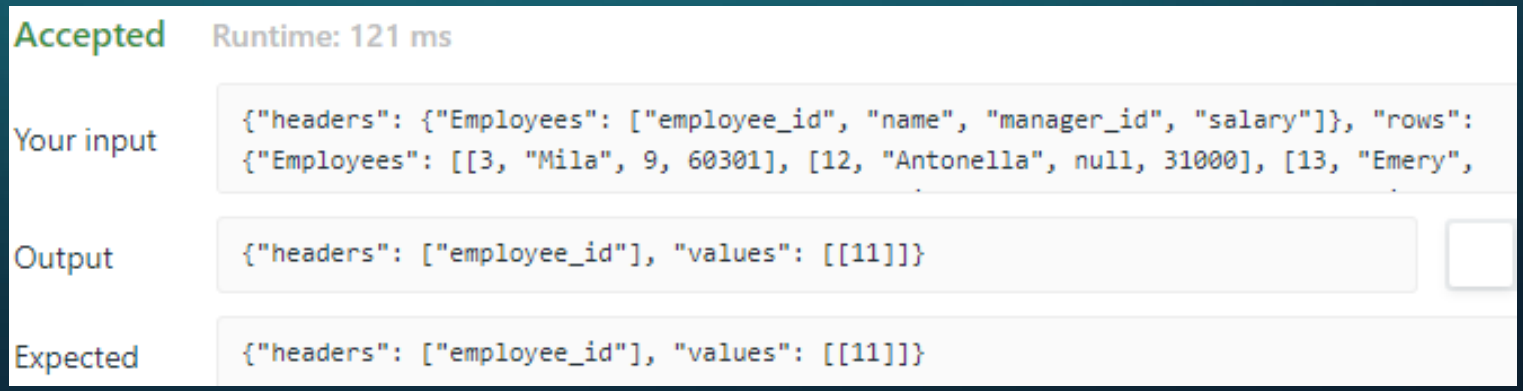
```sql
SELECT
'Low Salary' AS category,
COUNT(CASE WHEN income < 20000 THEN 1 END) AS accounts_count
FROM Accounts
UNION ALL
SELECT
'Average Salary' AS category,
COUNT(CASE WHEN income BETWEEN 20000 AND 50000 THEN 1 END) AS accounts_count
FROM Accounts
UNION ALL
SELECT
'High Salary' AS category,
COUNT(CASE WHEN income > 50000 THEN 1 END) AS accounts_count
FROM Accounts;
```

37. Find the IDs of the employees whose salary is strictly less than $30000 and whose manager left the company. When a manager leaves the company, their information is deleted from the Employees table, but the reports still have their manager_id set to the manager that left. Return the result table ordered by employee_id.

Table - Employees
Columns - employee_id (int), name (varchar), manager_id (int), salary (int)

```sql
SELECT employee_id
FROM Employees
WHERE salary < 30000 AND
manager_id NOT IN
(SELECT employee_id
FROM Employees)
ORDER BY employee_id;
```

```sql
SELECT e1.employee_id
FROM Employees e1
LEFT JOIN Employees e2
ON e1.manager_id = e2.employee_id
WHERE e1.salary < 30000 AND e2.employee_id IS NULL AND e1.manager_id IS NOT NULL
ORDER BY employee_id;
```



38. Write a solution to swap the seat id of every two consecutive students. If the number of students is odd, the id of the last student is not swapped. Return the result table ordered by id in ascending order.

Table - Seat
Columns - id (int), student (varchar)

```sql
SELECT
CASE
    WHEN id = (SELECT MAX(id) FROM seat) AND id % 2 = 1 THEN id
    WHEN id % 2 = 1 THEN id + 1
    ELSE id - 1
END AS id,
student
FROM seat
ORDER BY id
```

Accepted Runtime: 278 ms

Your input

```
{"headers": {"Seat": ["id","student"]}, "rows": {"Seat": [[1,"Abbot"],[2,"Doris"],
[3,"Emerson"],[4,"Green"],[5,"Jeames"]]}}
```

Output

```
{"headers": ["id", "student"], "values": [[1, "Doris"], [2, "Abbot"], [3,
"Green"], [4, "Emerson"], [5, "Jeames"]]}
```

Expected

```
{"headers": ["id", "student"], "values": [[1, "Doris"], [2, "Abbot"], [3, "Green"], [4,
"Emerson"], [5, "Jeames"]]}
```

39. Write a solution to:
Find the name of the user who has rated the greatest number of movies. In case of a tie, return the lexicographically smaller user name.
Find the movie name with the highest average rating in February 2020. In case of a tie, return the lexicographically smaller movie name.

Table1 - Movies, Columns - movie_id (int), title (varchar)
Table2 - Users, Columns- user_id (name), name (varchar)
Table3 - MovieRationg, Columns- movie_id (int), user_id (int), rating (int), created_at (date)

```sql
(Select
name as results
from movierating mr
join users u on u.user_id = mr.user_id
group by name
order by count(*) desc, name
limit 1)

UNION ALL

(Select
title as results
from movierating mr
join movies m on m.movie_id = mr.movie_id
where date_format(mr.created_at, '%Y-%m') = '2020-02'
group by title
order by avg(mr.rating) desc, title
limit 1)
```

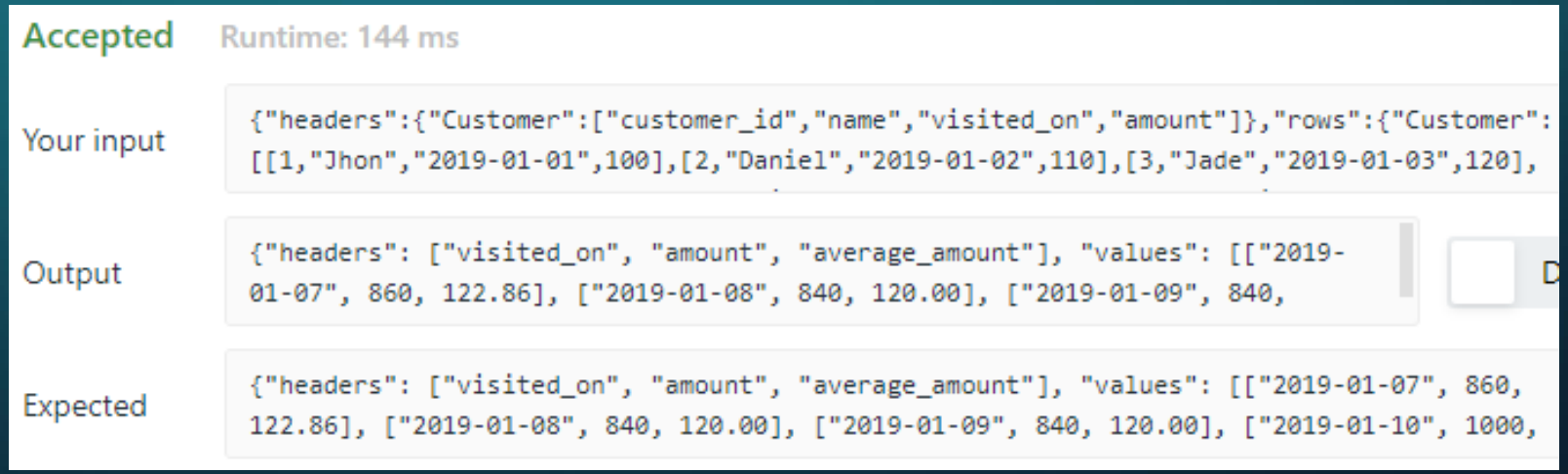
```sql
with cte1 as
(select
 u.name,
 rank() over(order by count(mr.movie_id) desc, u.name) as rnk1
from MovieRating mr
join Users u
on mr.user_id = u.user_id
group by mr.user_id),

cte2 as
(select m.title,
 rank() over(order by avg(mr.rating) desc, m.title) as rnk2
from Movies m
join MovieRating mr
on m.movie_id = mr.movie_id
where date_format(mr.created_at, "%Y-%m") = "2020-02"
group by mr.movie_id)

select name as results
from cte1
where rnk1 = 1
union all
select title as results
from cte2
where rnk2 = 1
```

40. You are the restaurant owner and you want to analyze a possible expansion (there will be at least one customer every day).

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 the result table ordered by visited_on in ascending order.

Table- Customer
Columns- customer_id (int), name (varchar), visited_on (date), amount (int)



---------- Krishna Sameera Kota

```sql
WITH cte1 AS(
SELECT visited_on,
SUM(amount) AS total_amount
FROM Customer
GROUP BY visited_on
),

cte2 as (
SELECT
visited_on,
SUM(total_amount) OVER(ORDER BY visited_on RANGE BETWEEN INTERVAL 6 DAY PRECEDING
                       AND CURRENT ROW) AS amount,
RANK() over (order by visited_on) as rnk
FROM cte1
)

Select
visited_on,
amount,
round(amount/7 , 2) as average_amount
from cte2
where rnk > 6
order by visited_on;
```

41. Write a solution to find the people who have the most friends and the most friends number. The test cases are generated so that only one person has the most friends.

Table-RequestAccepted
Columns- requester_id (int), accepter_id (int), accept_date (date)
This table contains the ID of the user who sent the request, the ID of the user who received the request, and the date when the request was accepted.

```
Select
x.id,  count(*) as num from
(select requester_id as id from RequestAccepted
 union all
 select accepter_id as id from RequestAccepted) as x
group by x.id
order by num desc
limit 1
```

```sql
With Cte as (
select requester_id as id from RequestAccepted
union all
select accepter_id as id from RequestAccepted
)

Select id,
count(*) as num from Cte
group by id
order by num desc
limit 1
```

Accepted Runtime: 124 ms

Your input

```
{"headers":{"RequestAccepted":["requester_id","accepter_id","accept_date"]},"rows":
{"RequestAccepted":[[1,2,"2016/06/03"],[1,3,"2016/06/08"],[2,3,"2016/06/08"],
```

Output

```
{"headers": ["id", "num"], "values": [[3, 3]]}
```

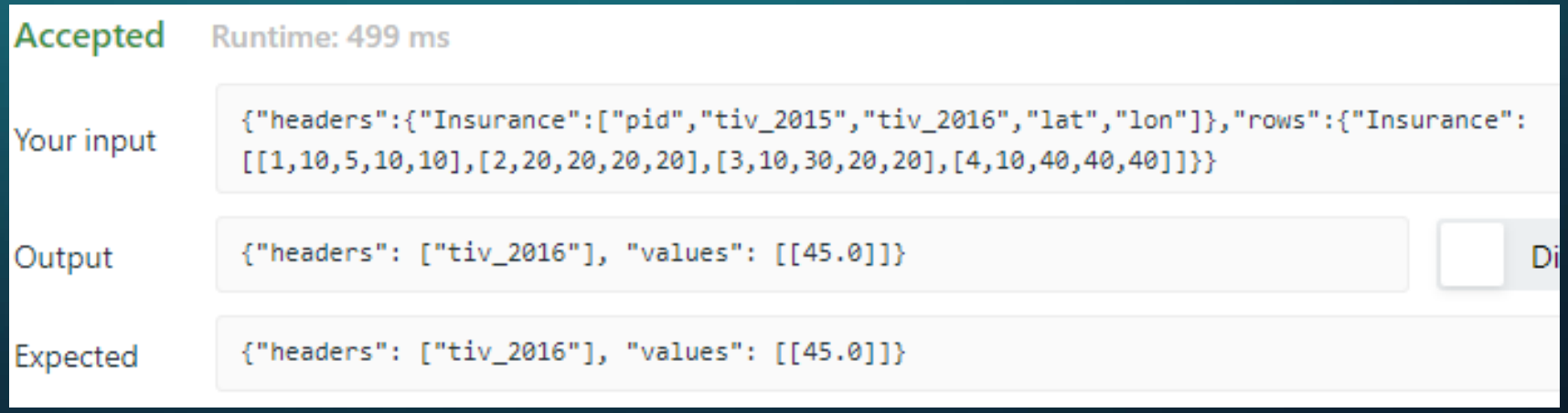
Expected

```
{"headers": ["id", "num"], "values": [[3, 3]]}
```

---------- Krishna Sameera Kota

42. Write a solution to report the sum of all total investment values in 2016 tiv_2016, for all policyholders who:
have the same tiv_2015 value as one or more other policyholders, and
are not located in the same city as any other policyholder (i.e., the (lat, lon) attribute pairs must be unique).
Round tiv_2016 to two decimal places.

Table- Insurance
Columns - pid (int), tiv_2015 (float), tiv_2016 (flaot), lat (float), lon (float)
pid is the policyholder's policy ID.
tiv_2015 is the total investment value in 2015 and tiv_2016 is the total investment value in 2016.



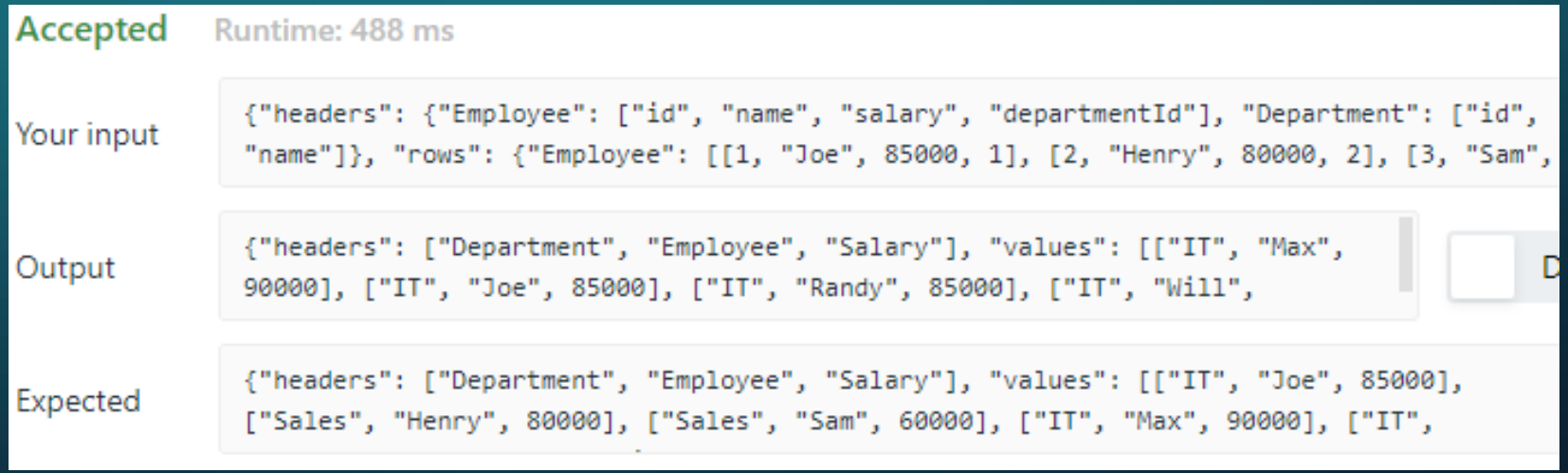
---------- Krishna Sameera Kota

```sql
Select
round(sum(tiv_2016),2) as tiv_2016
from Insurance
where tiv_2015 IN (
    Select tiv_2015
    from Insurance
    group by tiv_2015
    having count(*) >1)

and (lat,lon) IN (
    Select lat, lon
    from Insurance
    group by lat, lon
    having count(*) = 1)
```

43. A company's executives are interested in seeing who earns the most money in each of the company's departments. A high earner in a department is an employee who has a salary in the top three unique salaries for that department.

Write a solution to find the employees who are high earners in each of the departments. Return the result table in any order.

Table1 - Employee, Columns - id (int), name (varchar), salary (int), departmentId (int)
Table2 - Department, Columns - id (int), name (varchar)



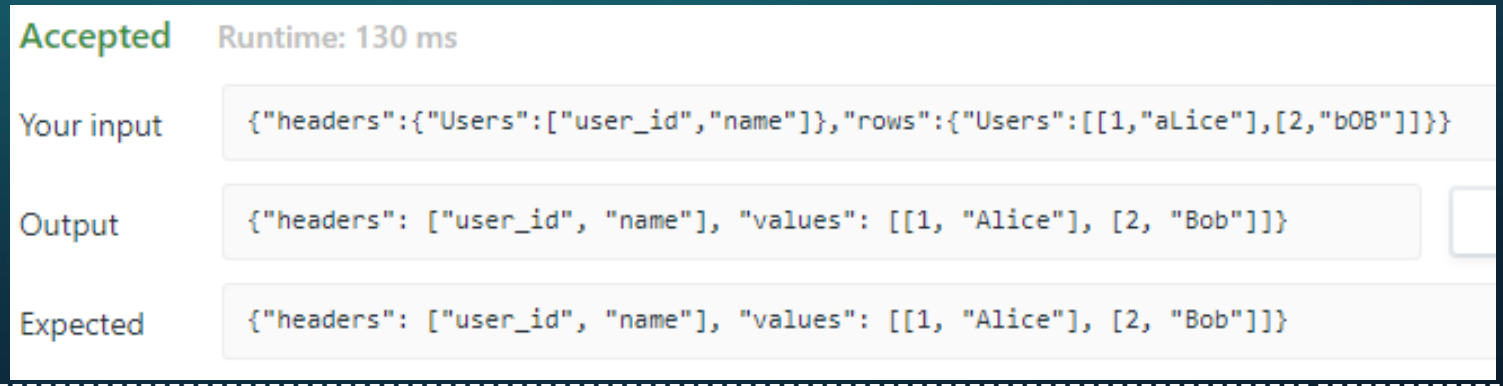
```sql
SELECT
Department,
Employee,
Salary
FROM (SELECT
      d.name AS Department,
      e.name AS Employee,
      e.salary AS Salary,
      DENSE_RANK() OVER (PARTITION BY d.name ORDER BY Salary DESC) AS rnk
      FROM Employee e
      JOIN Department d ON e.departmentId = d.id) AS rnk1
WHERE rnk <= 3;
```

44. Write a solution to fix the names so that only the first character is uppercase and the rest are lowercase. Return the result table ordered by user_id.

Table- Users
Columns- user_id (int), name (varchar)

```sql
Select
user_id,
CONCAT(UPPER(LEFT(name,1)), LOWER(RIGHT(name, LENGTH(name)-1))) as name
from Users
order by user_id
```

```sql
Select
user_id,
CONCAT(UPPER(SUBSTRING(name,1,1)), LOWER(SUBSTRING(name,2,LENGTH(name)))) as name
from Users
order by user_id
```



45. Write a solution to find the patient_id, patient_name, and conditions of the patients who have Type I Diabetes. Type I Diabetes always starts with DIAB1 prefix. Return the result table in any order.

Table - Patients
Columns- patient_id (int), patient_name (varchar), conditions (varchar)
'conditions' contains 0 or more code separated by spaces.

```sql
Select
patient_id,
patient_name,
conditions
from Patients
where conditions like '% DIAB1%' or conditions like 'DIAB1%'
```

Accepted Runtime: 242 ms

Your input

```
{"headers": {"Patients": ["patient_id", "patient_name", "conditions"]}, "rows":
{"Patients": [[1, "Daniel", "YFEV COUGH"], [2, "Alice", ""], [3, "Bob", "DIAB100 MYOP"],
```

Output

```
{"headers": ["patient_id", "patient_name", "conditions"], "values": [[3,
"Bob", "DIAB100 MYOP"], [4, "George", "ACNE DIAB100"]]}
```

Expected

```
{"headers": ["patient_id", "patient_name", "conditions"], "values": [[3, "Bob", "DIAB100
MYOP"], [4, "George", "ACNE DIAB100"]]}
```

46. Write a solution to delete all duplicate emails, keeping only one unique email with the smallest id. Please note that you are supposed to write a DELETE statement and not a SELECT one.

Table - Person
Columns - id (int), email (varchar)

```
DELETE p1
from person p1, person p2
where p1.email=p2.email and p1.id>p2.id;
```

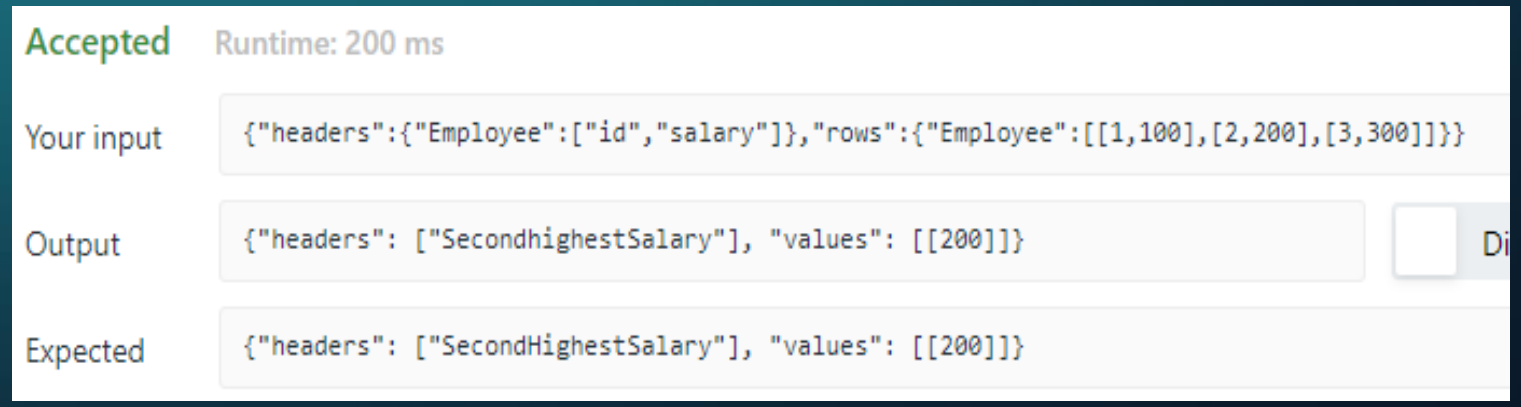
47. Write a solution to find the second highest salary from the Employee table. If there is no second highest salary, return null

Table - Employee
Columns- id (int), salary (int)

```sql
select
max(salary) as SecondhighestSalary
from employee
where salary <> (select max(salary) from employee)
```

```sql
select
(select distinct Salary
 from Employee
 order by salary desc
 limit 1 offset 1)
as SecondHighestSalary;
```

```sql
select
IFNULL(
(select distinct Salary
 from Employee
 order by salary desc
 limit 1 offset 1), NULL)
as SecondHighestSalary;
```



48. Write a solution to find for each date the number of different products sold and their names. The sold products names for each date should be sorted lexicographically. Return the result table ordered by sell_date.
Table - Activities
Columns - sell_date (date), product (varchar)

syntax : GROUP_CONCAT(column_name ORDER BY some_column SEPARATOR 'separator')

```
Select
sell_date,
COUNT(distinct product) as num_sold,
GROUP_CONCAT( DISTINCT product order by product ASC separator ',' ) as products
from Activities
group by sell_date
order by sell_date ;
```

Accepted Runtime: 247 ms

Your input

```
{"headers":{"Activities":["sell_date","product"]},"rows":{"Activities":[["2020-05-
30","Headphone"],["2020-06-01","Pencil"],["2020-06-02","Mask"],["2020-05-
```

Output

```
{"headers": ["sell_date", "num_sold", "products"], "values": [["2020-05-
30", 3, "Basketball,Headphone,T-Shirt"], ["2020-06-01", 2, "Bible,Pencil"],
```

Expected

```
{"headers": ["sell_date", "num_sold", "products"], "values": [["2020-05-30", 3,
"Basketball,Headphone,T-Shirt"], ["2020-06-01", 2, "Bible,Pencil"], ["2020-06-02", 1,
```

49. Write a solution to get the names of products that have at least 100 units ordered in February 2020 and their amount. Return the result table in any order.
Table1 - Products
Columns- product_id (int), product_name (varchar), product_category (varchar)
Table2 - Orders
Columns- product_id (int), order_date (date), unit (int)

```sql
Select
p.product_name,
sum(o.unit) as unit
from Products p
join Orders o on p.product_id = o.product_id
where date_format(o.order_date, '%Y-%m') = '2020-02'
group by p.product_name
having unit >=100
```

Accepted Runtime: 375 ms

Your input

```
{"headers": {"Products": ["product_id", "product_name", "product_category"], "Orders":
["product_id", "order_date", "unit"]}, "rows": {"Products": [[1, "Leetcode Solutions",
```

Output

```
{"headers": ["product_name", "unit"], "values": [["Leetcode Solutions", 130],
["Leetcode Kit", 100]]}
```

Expected

```
{"headers": ["product_name", "unit"], "values": [["Leetcode Solutions", 130], ["Leetcode
Kit", 100]]}
```

50. Write a solution to find the users who have valid emails.
A valid e-mail has a prefix name and a domain where:
The prefix name is a string that may contain letters (upper or lower case), digits, underscore '_', period '.', and/or dash '-'. The prefix name must start with a letter. The domain is '@leetcode.com'.
Return the result table in any order.
Table - Users
Columns- user_id (int), name (varchar), mail (varchar)

```sql
Select
user_id,
name,
mail
from Users
where mail regexp '^[A-Za-z][A-Za-z0-9_.-]*@leetcode[.]com$'
```

Accepted Runtime: 202 ms

Your input

```
{"headers":{"Users":["user_id","name","mail"]},"rows":{"Users":
[[1,"Winston","winston@leetcode.com"],[2,"Jonathan","jonathanisgreat"],
```

Output

```
{"headers": ["user_id", "name", "mail"], "values": [[1, "Winston",
"winston@leetcode.com"], [3, "Annabelle", "bella-@leetcode.com"], [4,
```

Expected

```
{"headers": ["user_id", "name", "mail"], "values": [[1, "Winston",
"winston@leetcode.com"], [3, "Annabelle", "bella-@leetcode.com"], [4, "Sally",
```

THANKYOU