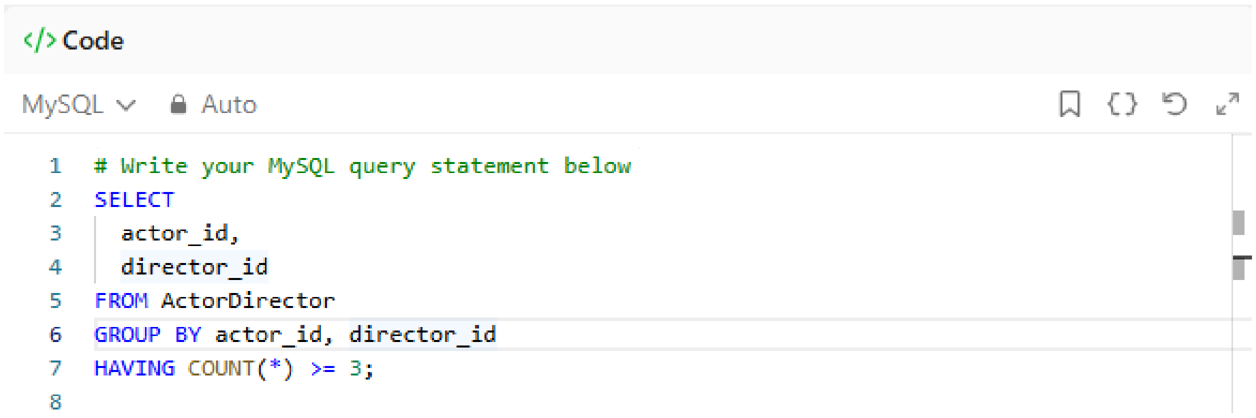
Reem Aboutaleb

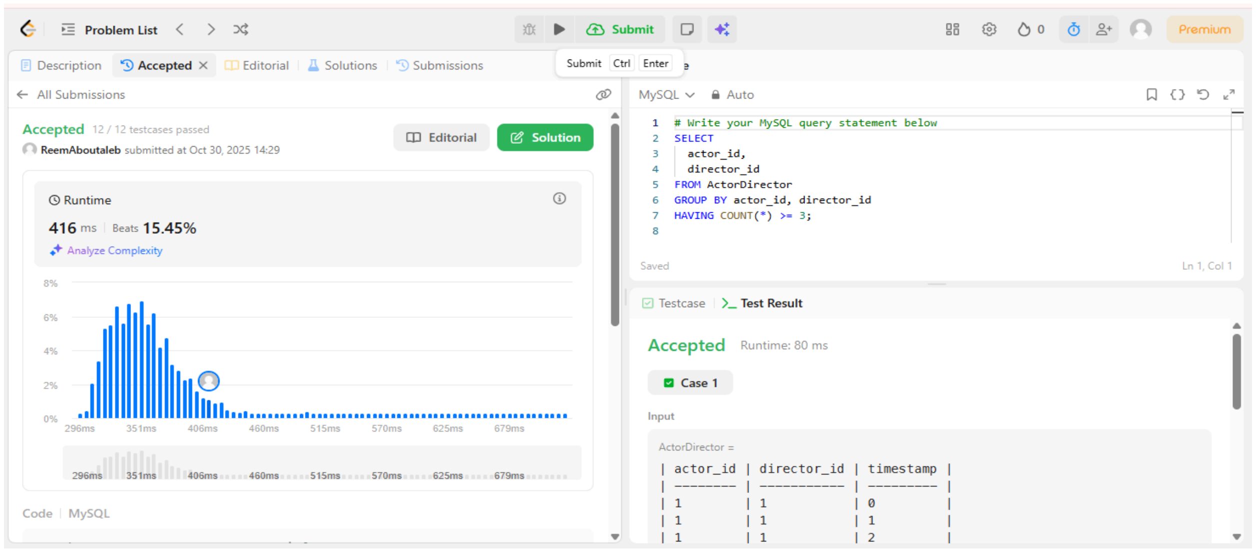
Homework#4 Bootcamp

Answer # 1 : [1050. Actors and Directors Who Cooperated At Least Three Times]

[Code I used:]



```sql
# Write your MySQL query statement below
SELECT
    actor_id,
    director_id
FROM ActorDirector
GROUP BY actor_id, director_id
HAVING COUNT(*) >= 3;
```



Accepted 12 / 12 testcases passed

ReemAboutaleb submitted at Oct 30, 2025 14:29

Runtime

416 ms | Beats 15.45%

```sql
# Write your MySQL query statement below
SELECT
  actor_id,
  director_id
FROM ActorDirector
GROUP BY actor_id, director_id
HAVING COUNT(*) >= 3;
```

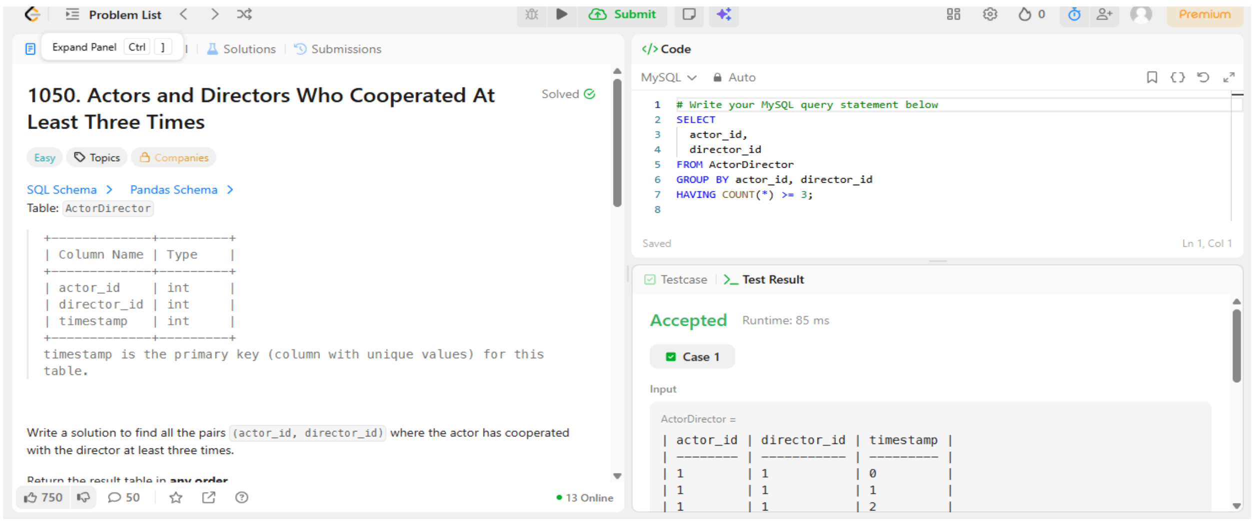
Accepted Runtime: 80 ms

Case 1

Input

ActorDirector =

```
| actor_id | director_id | timestamp |
| -------- | ----------- | --------- |
| 1        | 1           | 0         |
| 1        | 1           | 1         |
| 1        | 1           | 2         |
```



1050. Actors and Directors Who Cooperated At Least Three Times

Solved

Easy · Topics · Companies

SQL Schema > Pandas Schema >

Table: `ActorDirector`

```
+-------------+---------+
| Column Name | Type    |
+-------------+---------+
| actor_id    | int     |
| director_id | int     |
| timestamp   | int     |
+-------------+---------+
timestamp is the primary key (column with unique values) for this
table.
```

Write a solution to find all the pairs `(actor_id, director_id)` where the actor has cooperated with the director at least three times.

```sql
# Write your MySQL query statement below
SELECT
  actor_id,
  director_id
FROM ActorDirector
GROUP BY actor_id, director_id
HAVING COUNT(*) >= 3;
```

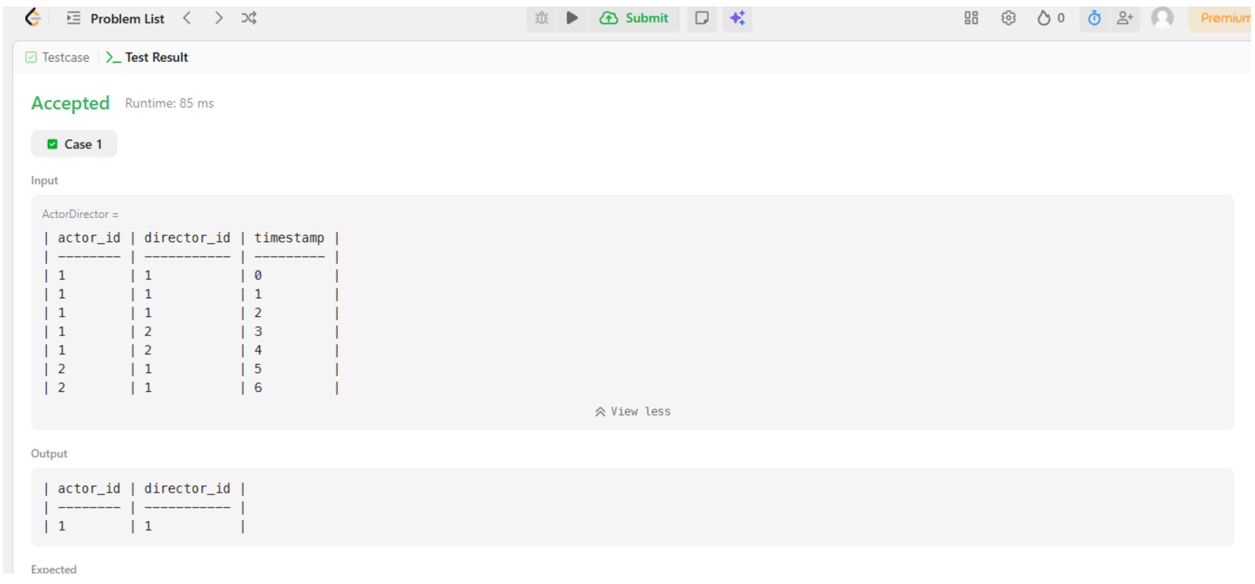
Accepted Runtime: 85 ms

Case 1

Input

ActorDirector =

```
| actor_id | director_id | timestamp |
| -------- | ----------- | --------- |
| 1        | 1           | 0         |
| 1        | 1           | 1         |
| 1        | 1           | 2         |
```



Testcase · Test Result

Accepted Runtime: 85 ms

Case 1

Input

ActorDirector =

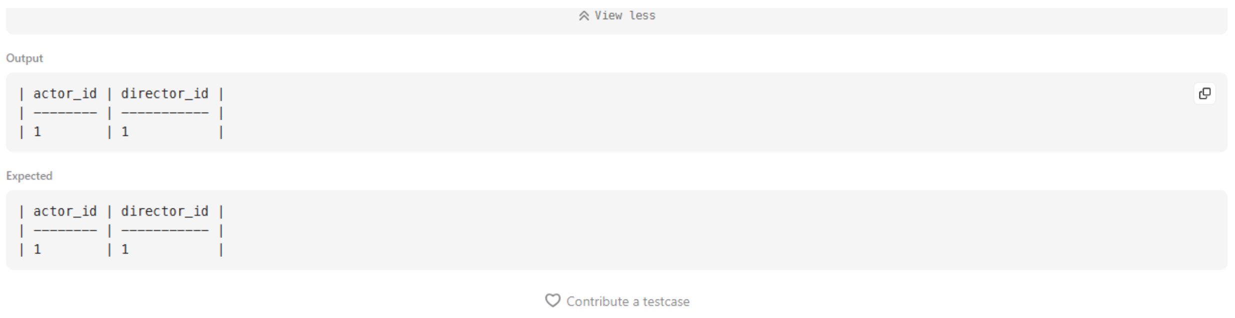
```
| actor_id | director_id | timestamp |
| -------- | ----------- | --------- |
| 1        | 1           | 0         |
| 1        | 1           | 1         |
| 1        | 1           | 2         |
| 1        | 2           | 3         |
| 1        | 2           | 4         |
| 2        | 1           | 5         |
| 2        | 1           | 6         |
```

View less

Output

```
| actor_id | director_id |
| -------- | ----------- |
| 1        | 1           |
```

Expected



View less

Output

```
| actor_id | director_id |
| -------- | ----------- |
| 1        | 1           |
```

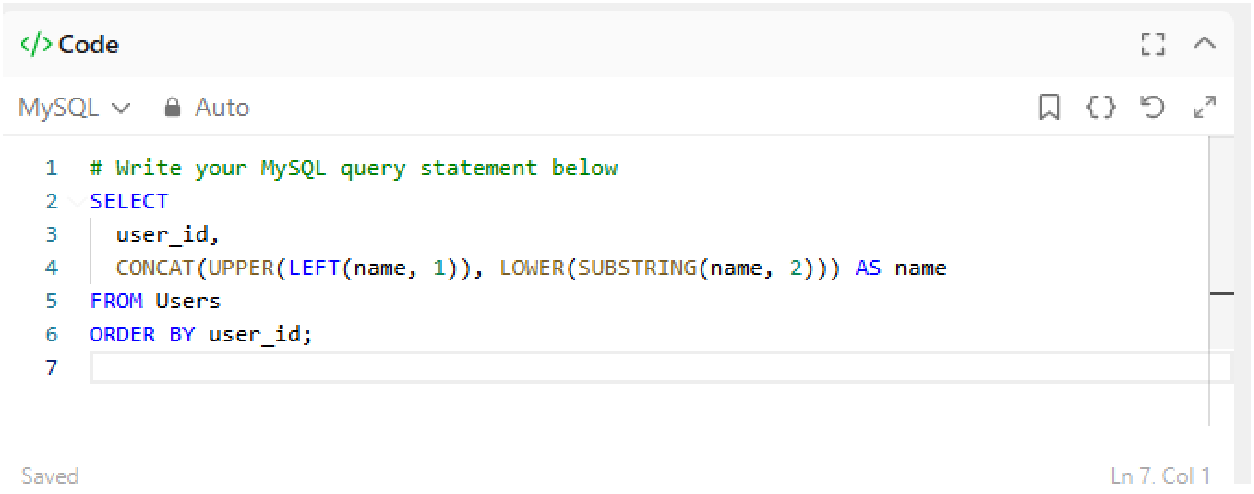
Expected

```
| actor_id | director_id |
| -------- | ----------- |
| 1        | 1           |
```

Contribute a testcase

Answer#2: 1667. Fix Names in a Table

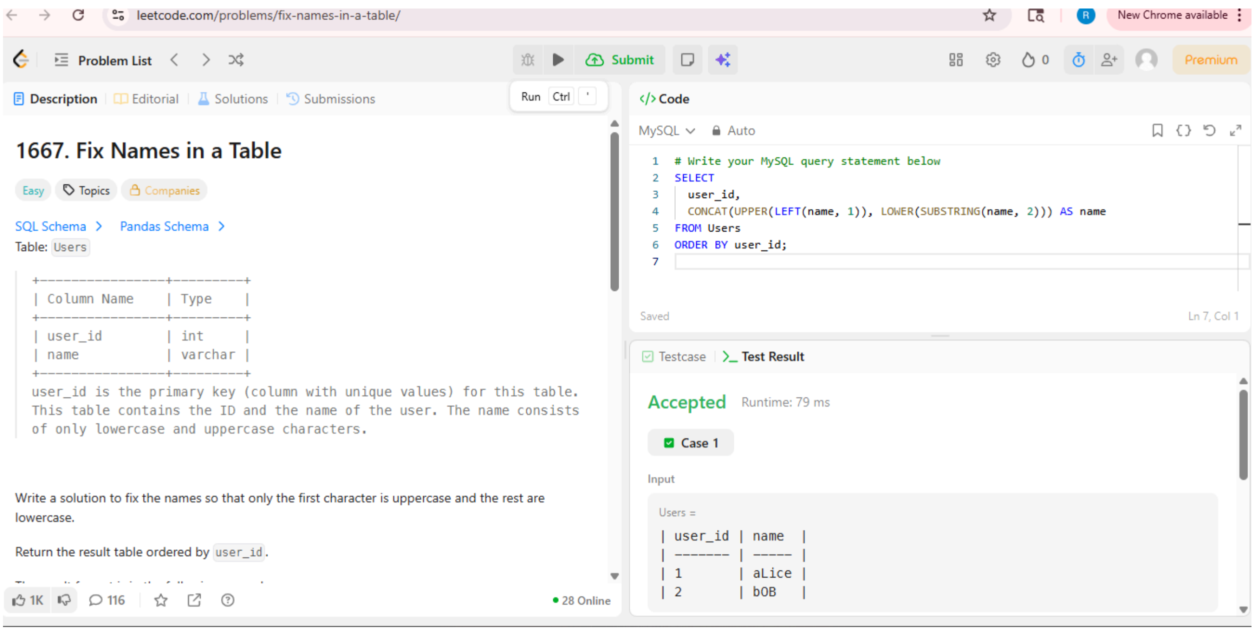
Code I used:



```sql
# Write your MySQL query statement below
SELECT
    user_id,
    CONCAT(UPPER(LEFT(name, 1)), LOWER(SUBSTRING(name, 2))) AS name
FROM Users
ORDER BY user_id;
```

Saved

Ln 7, Col 1



1667. Fix Names in a Table

Table: Users

```
+----------------+---------+
| Column Name    | Type    |
+----------------+---------+
| user_id        | int     |
| name           | varchar |
+----------------+---------+
user_id is the primary key (column with unique values) for this table.
This table contains the ID and the name of the user. The name consists
of only lowercase and uppercase characters.
```

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.

```sql
# Write your MySQL query statement below
SELECT
    user_id,
    CONCAT(UPPER(LEFT(name, 1)), LOWER(SUBSTRING(name, 2))) AS name
FROM Users
ORDER BY user_id;
```

Saved

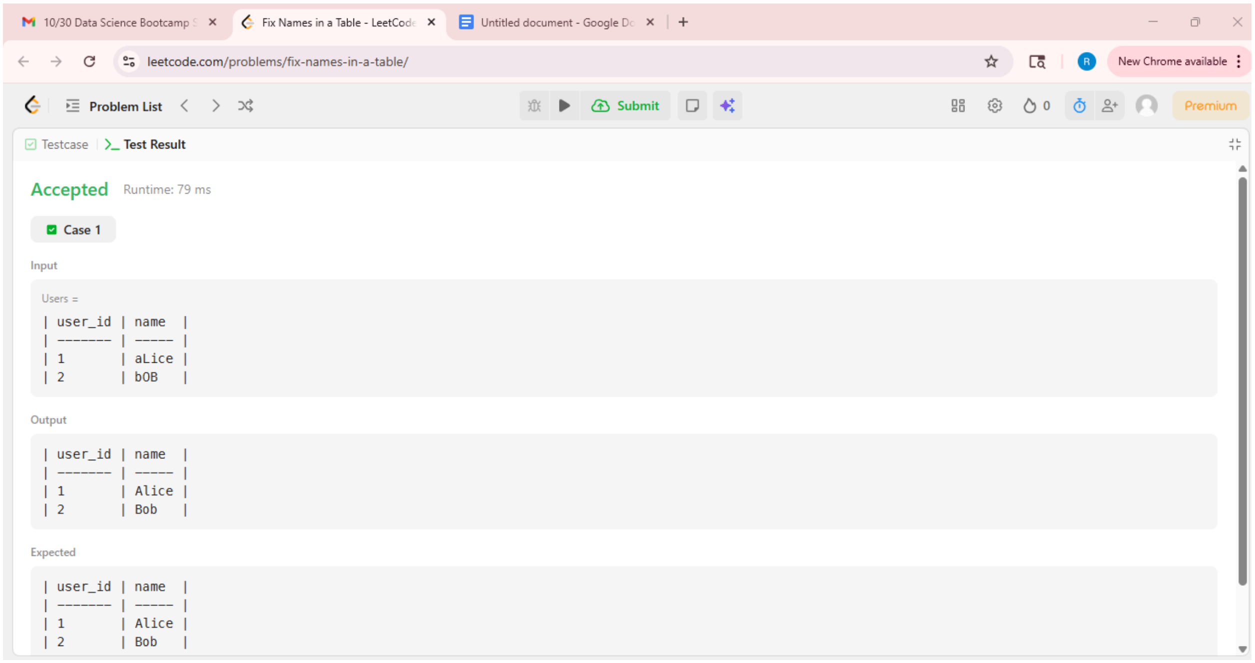
Ln 7, Col 1

Accepted Runtime: 79 ms

Case 1

Input

```
Users =
| user_id | name  |
| ------- | ----- |
| 1       | aLice |
| 2       | bOB   |
```


10/30 Data Science Bootcamp
Fix Names in a Table - LeetCode
Untitled document - Google Docs
leetcode.com/problems/fix-names-in-a-table/
New Chrome available
Problem List
Submit
Premium
Testcase
Test Result

Accepted Runtime: 79 ms

Case 1

Input

Users =

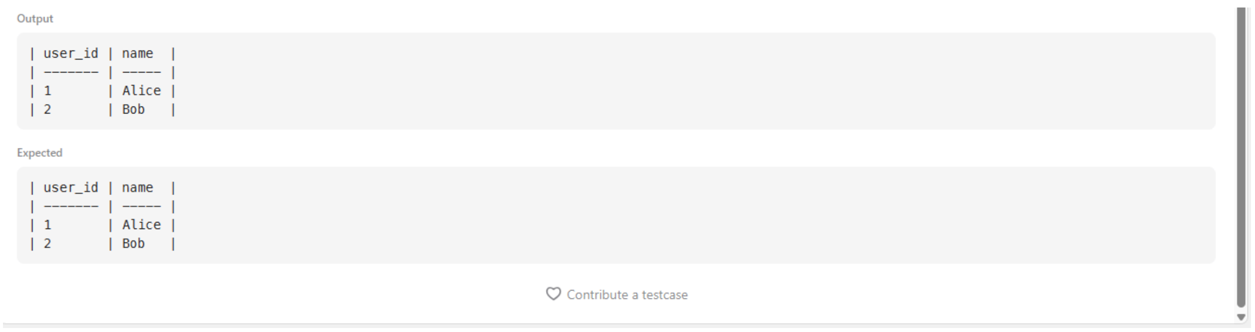
```
| user_id | name  |
| ------- | ----- |
| 1       | aLice |
| 2       | bOB   |
```

Output

```
| user_id | name  |
| ------- | ----- |
| 1       | Alice |
| 2       | Bob   |
```

Expected

```
| user_id | name  |
| ------- | ----- |
| 1       | Alice |
| 2       | Bob   |
```



Output

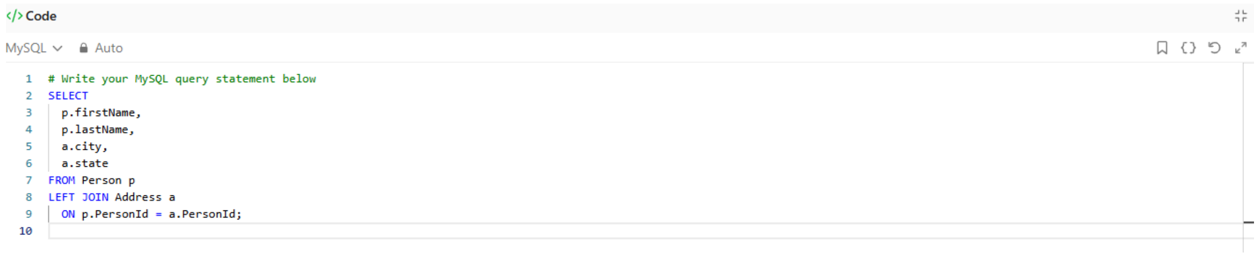
```
| user_id | name  |
| ------- | ----- |
| 1       | Alice |
| 2       | Bob   |
```

Expected

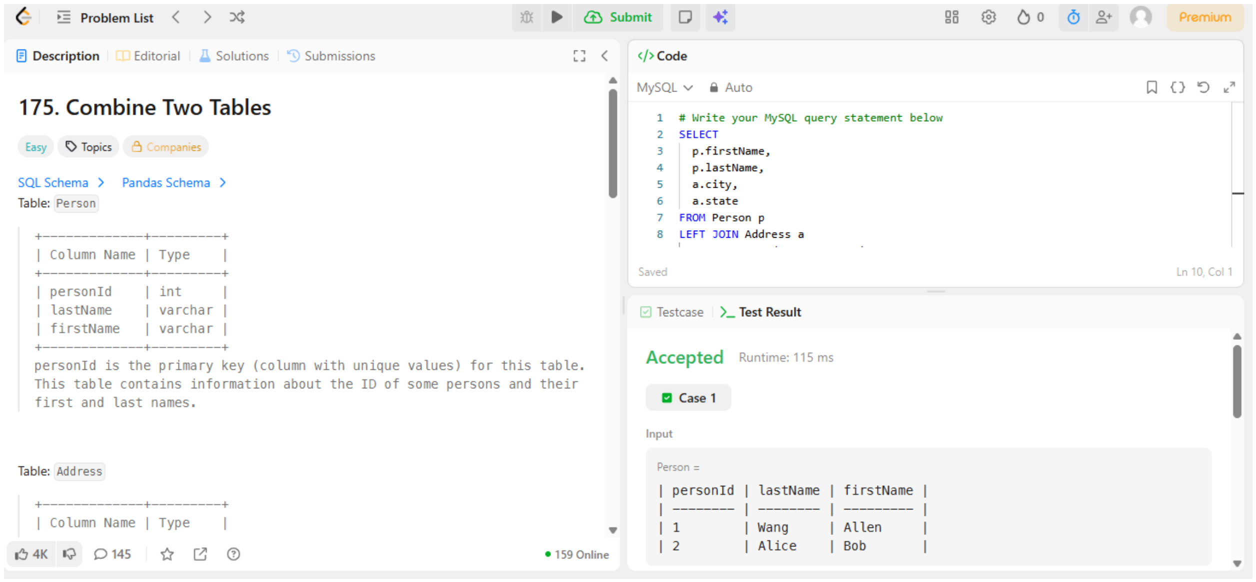
```
| user_id | name  |
| ------- | ----- |
| 1       | Alice |
| 2       | Bob   |
```

Answer#3 175. Combine Two Tables

Code I used:



```sql
# Write your MySQL query statement below
SELECT
  p.firstName,
  p.lastName,
  a.city,
  a.state
FROM Person p
LEFT JOIN Address a
  ON p.PersonId = a.PersonId;
```



175. Combine Two Tables

SQL Schema > Pandas Schema >

Table: Person

```
+-------------+---------+
| Column Name | Type    |
+-------------+---------+
| personId    | int     |
| lastName    | varchar |
| firstName   | varchar |
+-------------+---------+
personId is the primary key (column with unique values) for this table.
This table contains information about the ID of some persons and their
first and last names.
```

Table: Address

```
+-------------+---------+
| Column Name | Type    |
```

```sql
# Write your MySQL query statement below
SELECT
  p.firstName,
  p.lastName,
  a.city,
  a.state
FROM Person p
LEFT JOIN Address a
```

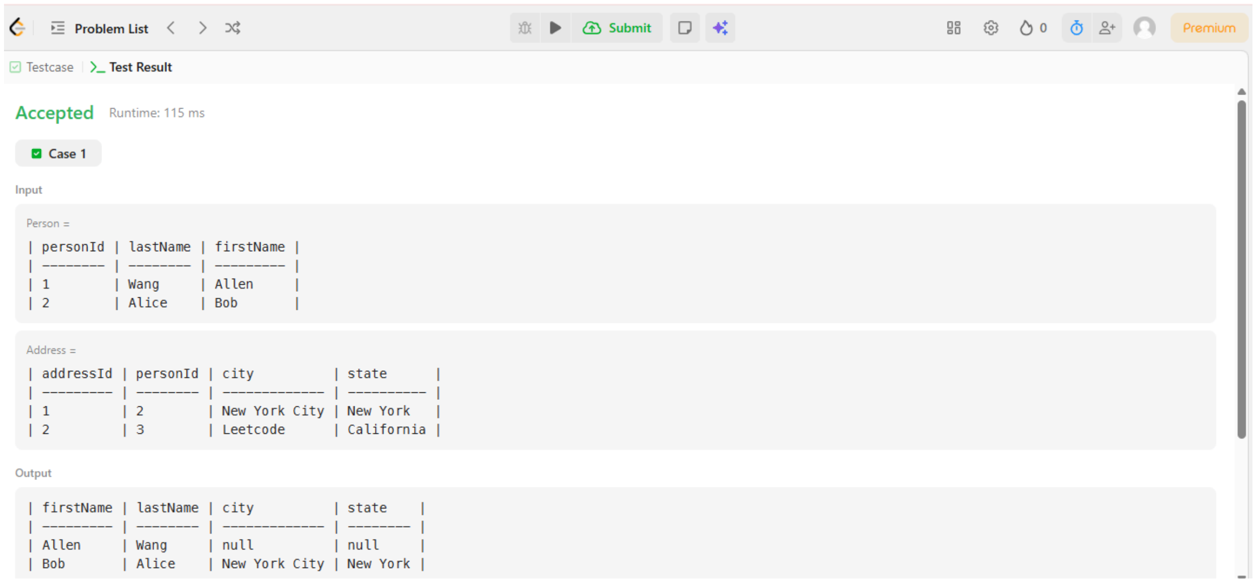
Accepted Runtime: 115 ms

Case 1

Input

Person =

```
| personId | lastName | firstName |
| -------- | -------- | --------- |
| 1        | Wang     | Allen     |
| 2        | Alice    | Bob       |
```



Accepted Runtime: 115 ms

Case 1

Input

Person =

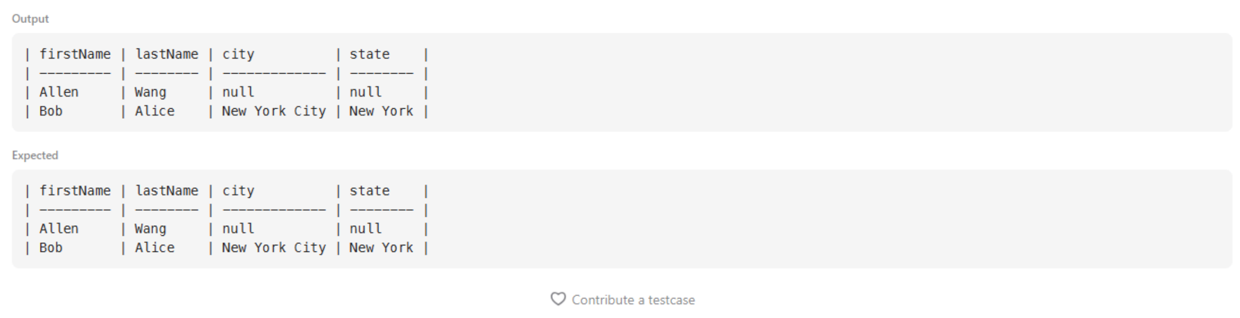
```
| personId | lastName | firstName |
| -------- | -------- | --------- |
| 1        | Wang     | Allen     |
| 2        | Alice    | Bob       |
```

Address =

```
| addressId | personId | city          | state      |
| --------- | -------- | ------------- | ---------- |
| 1         | 2        | New York City | New York   |
| 2         | 3        | Leetcode      | California |
```

Output

```
| firstName | lastName | city          | state    |
| --------- | -------- | ------------- | -------- |
| Allen     | Wang     | null          | null     |
| Bob       | Alice    | New York City | New York |
```



Output

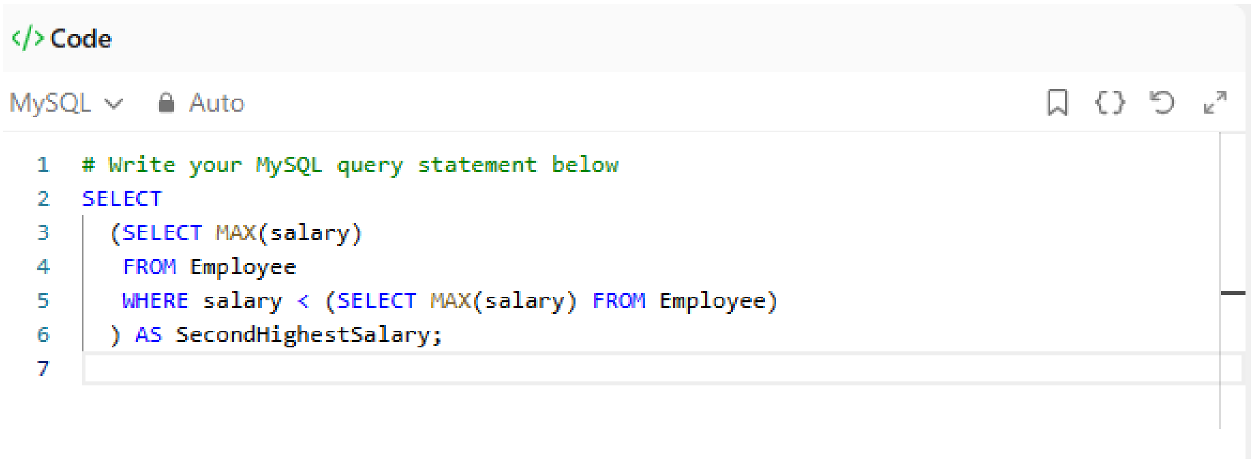
```
| firstName | lastName | city          | state    |
| --------- | -------- | ------------- | -------- |
| Allen     | Wang     | null          | null     |
| Bob       | Alice    | New York City | New York |
```

Expected

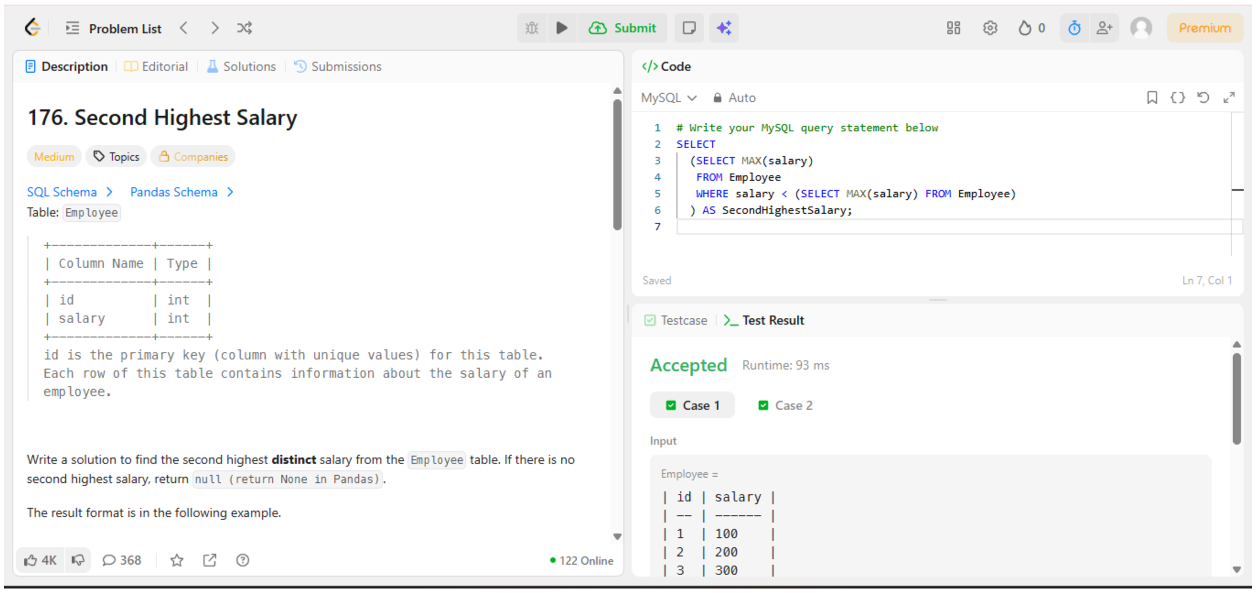
```
| firstName | lastName | city          | state    |
| --------- | -------- | ------------- | -------- |
| Allen     | Wang     | null          | null     |
| Bob       | Alice    | New York City | New York |
```

Answer# 4: [176. Second Highest Salary]

Code I used:



```sql
# Write your MySQL query statement below
SELECT
  (SELECT MAX(salary)
   FROM Employee
   WHERE salary < (SELECT MAX(salary) FROM Employee)
  ) AS SecondHighestSalary;
```



176. Second Highest Salary

Table: `Employee`

```
+-------------+------+
| Column Name | Type |
+-------------+------+
| id          | int  |
| salary      | int  |
+-------------+------+
id is the primary key (column with unique values) for this table.
Each row of this table contains information about the salary of an
employee.
```

Write a solution to find the second highest **distinct** salary from the `Employee` table. If there is no second highest salary, return `null (return None in Pandas)`.

The result format is in the following example.

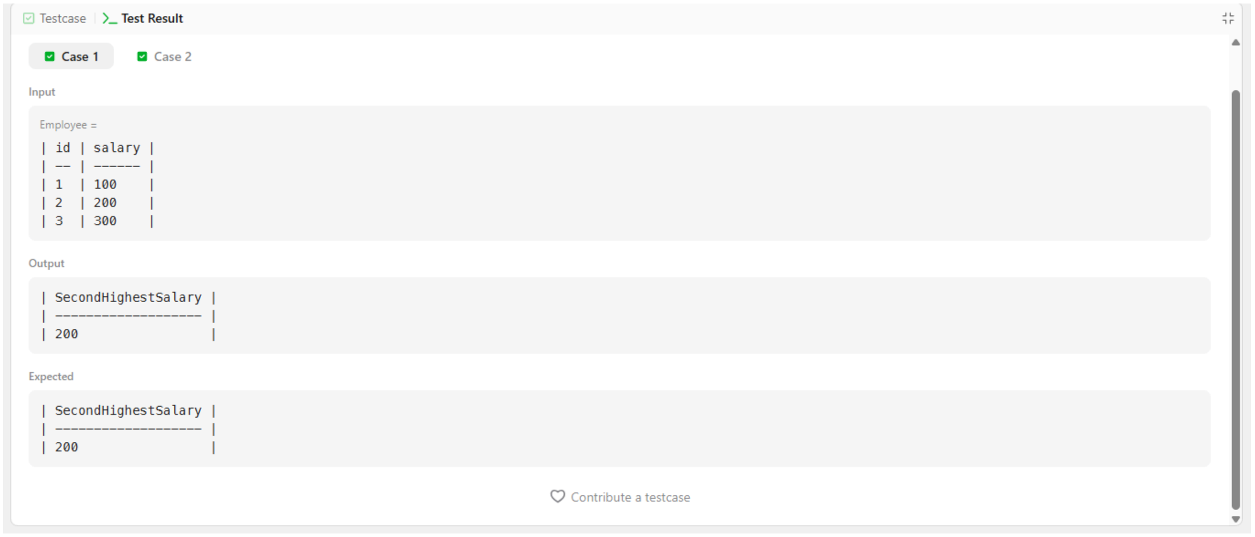
```sql
# Write your MySQL query statement below
SELECT
  (SELECT MAX(salary)
   FROM Employee
   WHERE salary < (SELECT MAX(salary) FROM Employee)
  ) AS SecondHighestSalary;
```

Accepted Runtime: 93 ms

Input

Employee =

```
| id | salary |
| -- | ------ |
| 1  | 100    |
| 2  | 200    |
| 3  | 300    |
```



Input

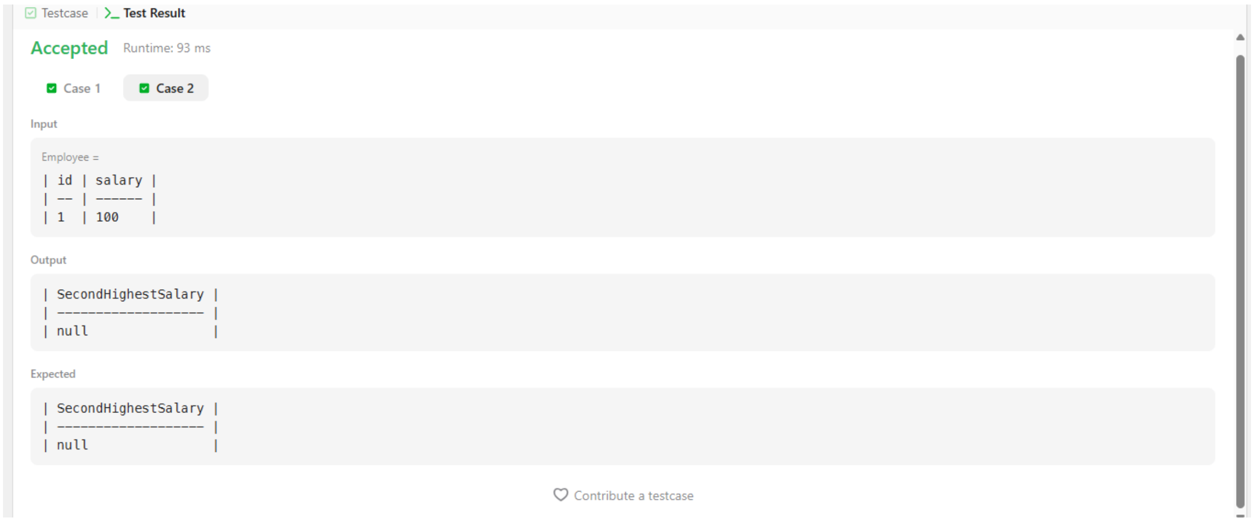
```
Employee =
| id | salary |
| -- | ------ |
| 1  | 100    |
| 2  | 200    |
| 3  | 300    |
```

Output

```
| SecondHighestSalary |
| ------------------- |
| 200                 |
```

Expected

```
| SecondHighestSalary |
| ------------------- |
| 200                 |
```



Accepted Runtime: 93 ms

Input

```
Employee =
| id | salary |
| -- | ------ |
| 1  | 100    |
```

Output

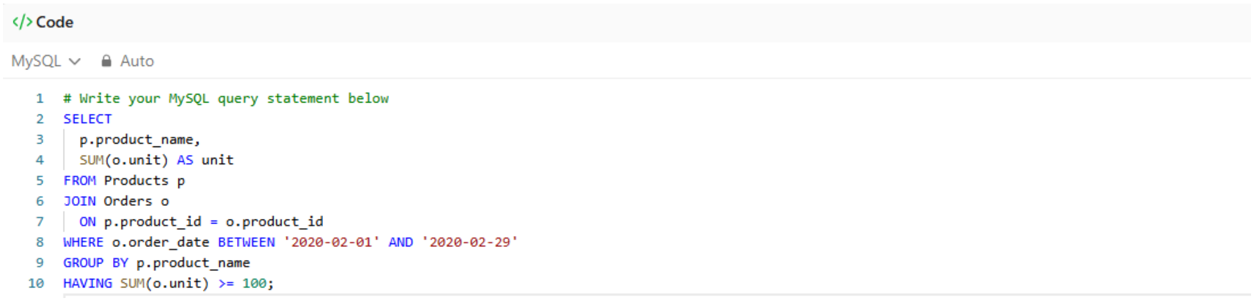
```
| SecondHighestSalary |
| ------------------- |
| null                |
```

Expected

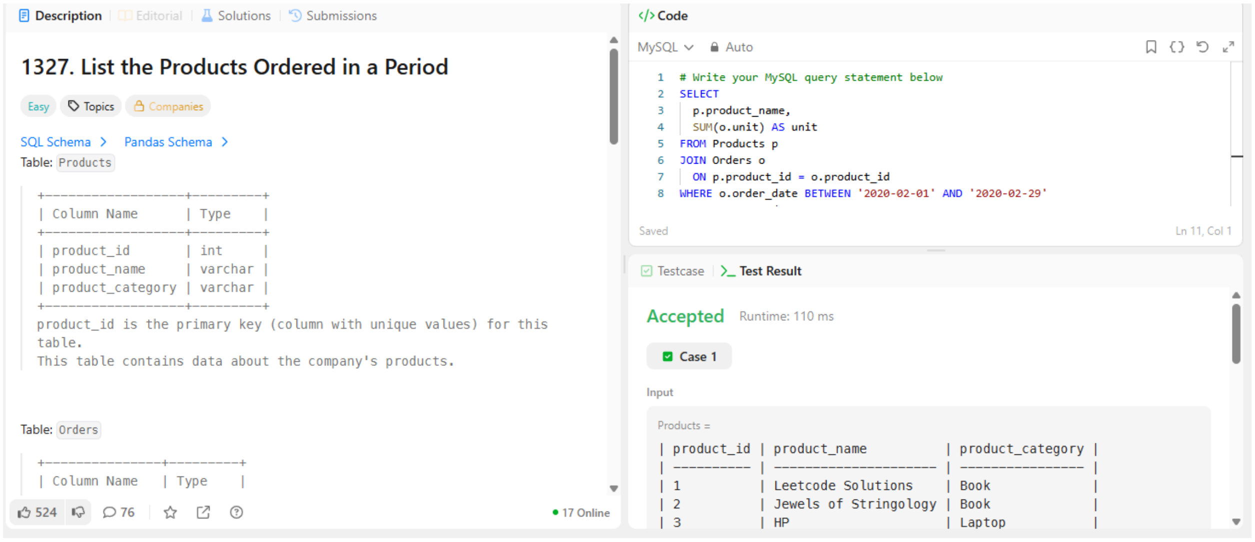
```
| SecondHighestSalary |
| ------------------- |
| null                |
```

Answer#5: 1327. List the Products Ordered in a Period

Code I used:



```sql
# Write your MySQL query statement below
SELECT
  p.product_name,
  SUM(o.unit) AS unit
FROM Products p
JOIN Orders o
  ON p.product_id = o.product_id
WHERE o.order_date BETWEEN '2020-02-01' AND '2020-02-29'
GROUP BY p.product_name
HAVING SUM(o.unit) >= 100;
```


Description | Editorial | Solutions | Submissions

1327. List the Products Ordered in a Period

Easy | Topics | Companies

SQL Schema > Pandas Schema >

Table: `Products`

```
+------------------+---------+
| Column Name      | Type    |
+------------------+---------+
| product_id       | int     |
| product_name     | varchar |
| product_category | varchar |
+------------------+---------+
product_id is the primary key (column with unique values) for this
table.
This table contains data about the company's products.
```

Table: `Orders`

```
+---------------+---------+
| Column Name   | Type    |
```

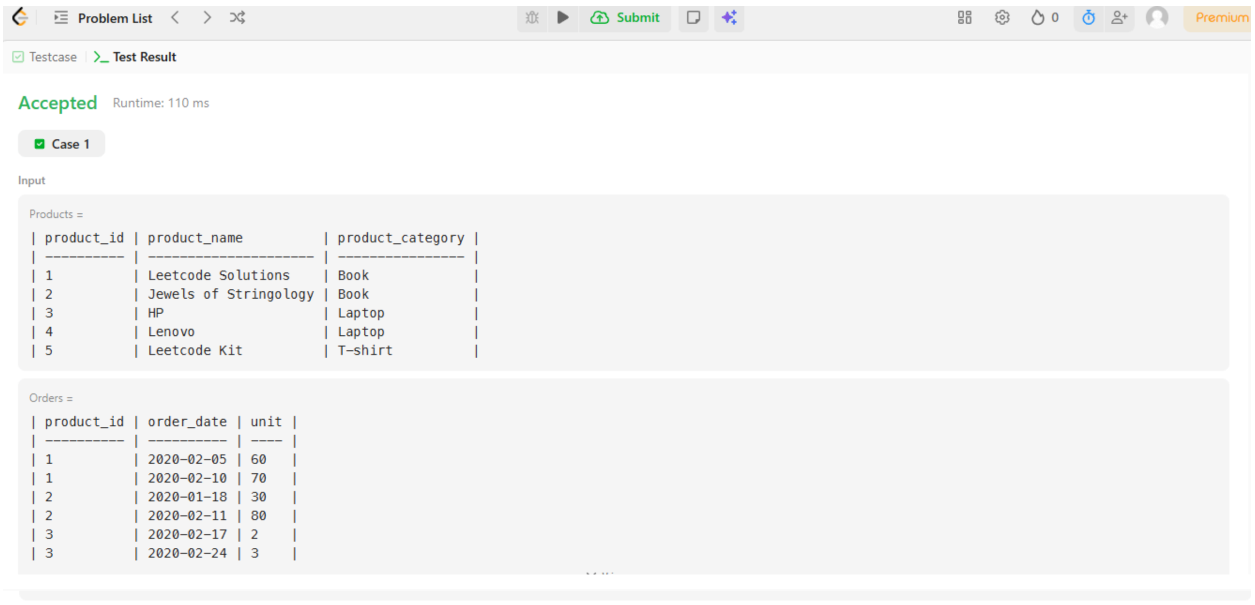
```sql
# Write your MySQL query statement below
SELECT
  p.product_name,
  SUM(o.unit) AS unit
FROM Products p
JOIN Orders o
  ON p.product_id = o.product_id
WHERE o.order_date BETWEEN '2020-02-01' AND '2020-02-29'
```

Accepted Runtime: 110 ms

Input

Products =

```
| product_id | product_name          | product_category |
| ---------- | --------------------- | ---------------- |
| 1          | Leetcode Solutions    | Book             |
| 2          | Jewels of Stringology | Book             |
| 3          | HP                    | Laptop           |
```



Accepted Runtime: 110 ms

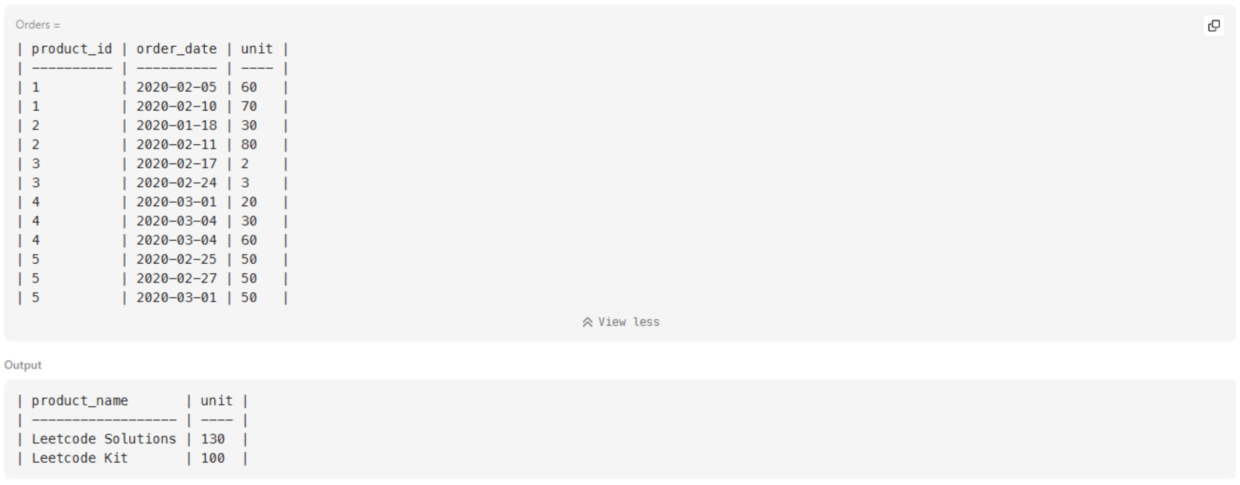
Input

Products =

```
| product_id | product_name          | product_category |
| ---------- | --------------------- | ---------------- |
| 1          | Leetcode Solutions    | Book             |
| 2          | Jewels of Stringology | Book             |
| 3          | HP                    | Laptop           |
| 4          | Lenovo                | Laptop           |
| 5          | Leetcode Kit          | T-shirt          |
```

Orders =

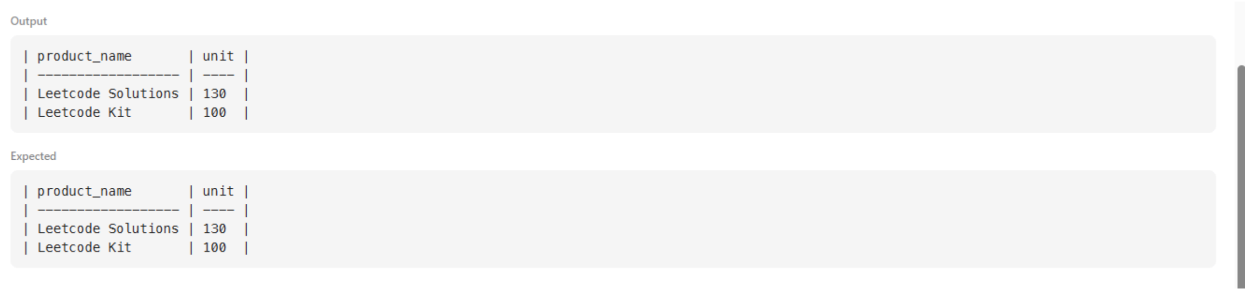
```
| product_id | order_date | unit |
| ---------- | ---------- | ---- |
| 1          | 2020-02-05 | 60   |
| 1          | 2020-02-10 | 70   |
| 2          | 2020-01-18 | 30   |
| 2          | 2020-02-11 | 80   |
| 3          | 2020-02-17 | 2    |
| 3          | 2020-02-24 | 3    |
```


Orders =

```
| product_id | order_date | unit |
| ---------- | ---------- | ---- |
| 1          | 2020-02-05 | 60   |
| 1          | 2020-02-10 | 70   |
| 2          | 2020-01-18 | 30   |
| 2          | 2020-02-11 | 80   |
| 3          | 2020-02-17 | 2    |
| 3          | 2020-02-24 | 3    |
| 4          | 2020-03-01 | 20   |
| 4          | 2020-03-04 | 30   |
| 4          | 2020-03-04 | 60   |
| 5          | 2020-02-25 | 50   |
| 5          | 2020-02-27 | 50   |
| 5          | 2020-03-01 | 50   |
```

Output

```
| product_name       | unit |
| ------------------ | ---- |
| Leetcode Solutions | 130  |
| Leetcode Kit       | 100  |
```



Output

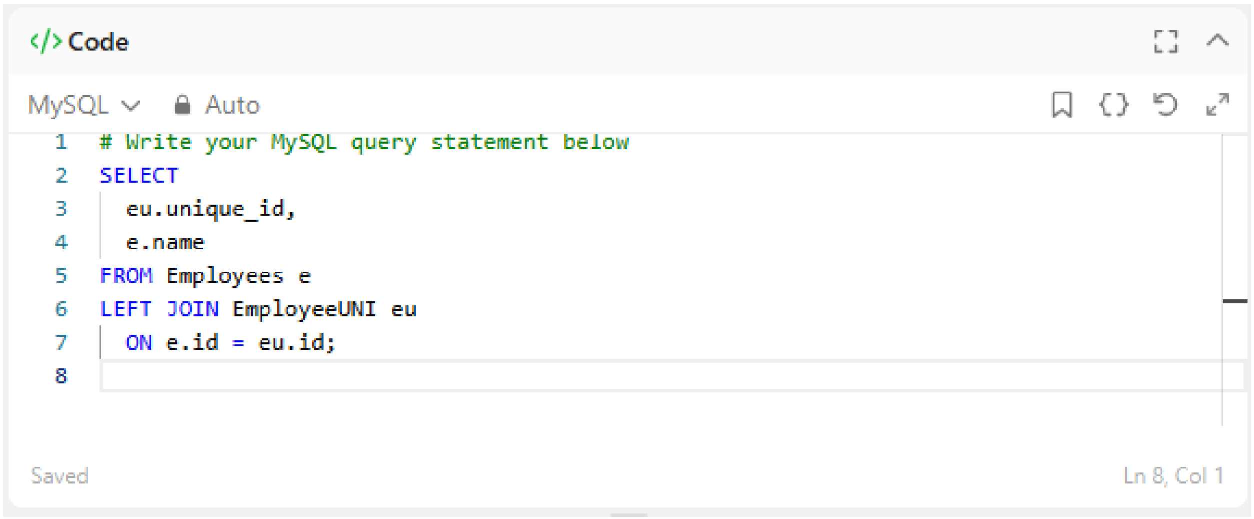
```
| product_name      | unit |
| ----------------- | ---- |
| Leetcode Solutions | 130  |
| Leetcode Kit      | 100  |
```

Expected

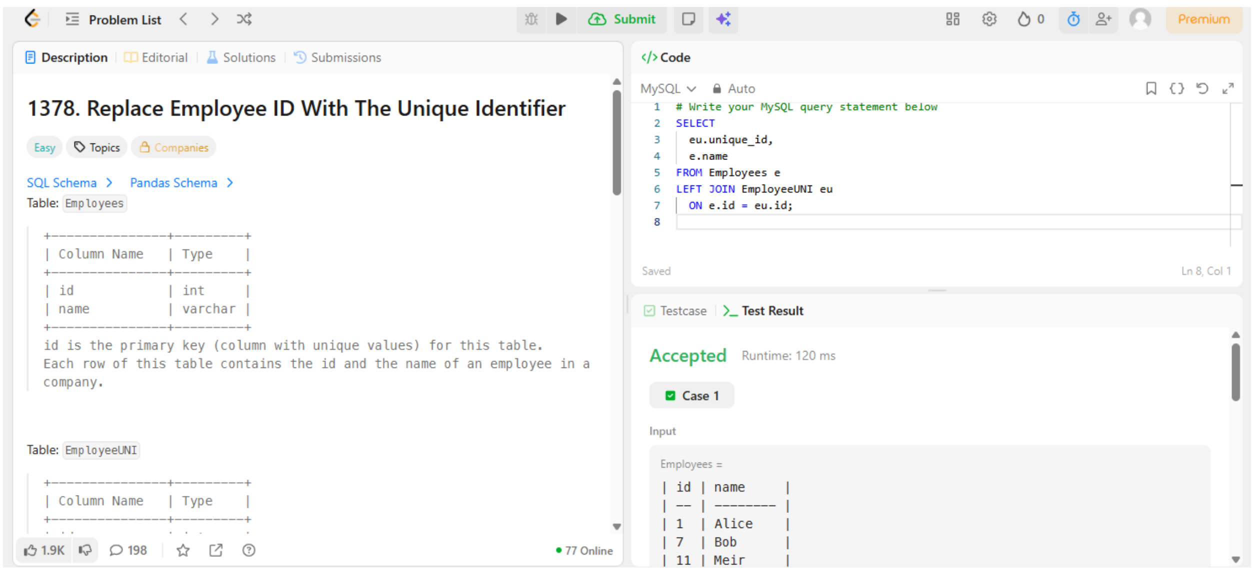
```
| product_name      | unit |
| ----------------- | ---- |
| Leetcode Solutions | 130  |
| Leetcode Kit      | 100  |
```

Answer#6: 1378. Replace Employee ID With The Unique Identifier

Code I used:



```sql
# Write your MySQL query statement below
SELECT
  eu.unique_id,
  e.name
FROM Employees e
LEFT JOIN EmployeeUNI eu
  ON e.id = eu.id;
```



1378. Replace Employee ID With The Unique Identifier

Table: Employees

```
+---------------+---------+
| Column Name   | Type    |
+---------------+---------+
| id            | int     |
| name          | varchar |
+---------------+---------+
id is the primary key (column with unique values) for this table.
Each row of this table contains the id and the name of an employee in a
company.
```

Table: EmployeeUNI

```
+---------------+---------+
| Column Name   | Type    |
+---------------+---------+
```

```sql
# Write your MySQL query statement below
SELECT
  eu.unique_id,
  e.name
FROM Employees e
LEFT JOIN EmployeeUNI eu
  ON e.id = eu.id;
```

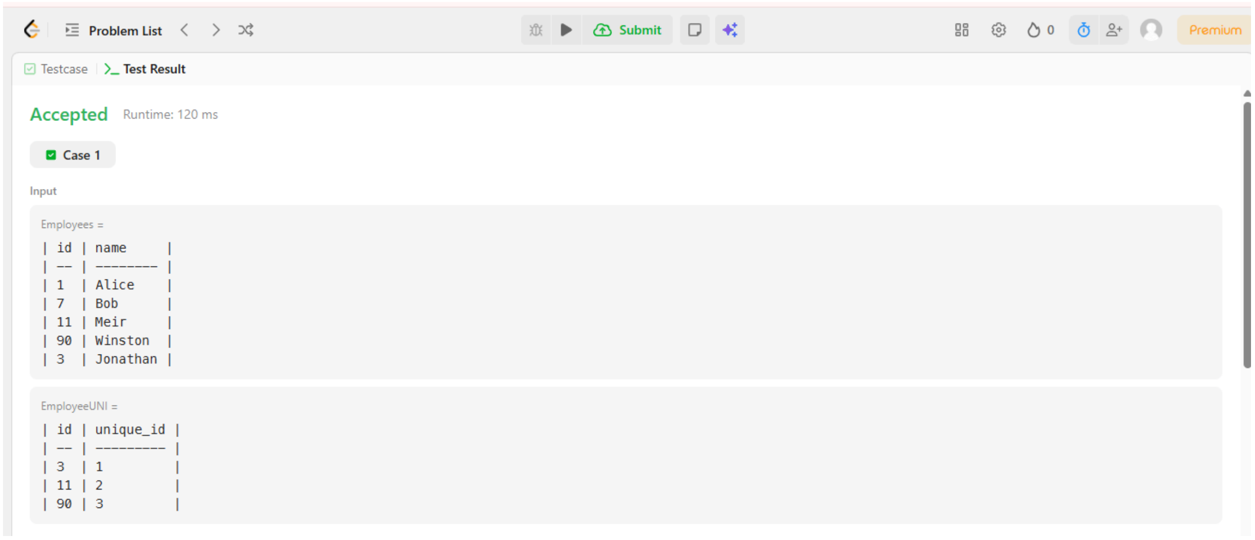
Accepted Runtime: 120 ms

Case 1

Input

Employees =

```
| id | name     |
| -- | -------- |
| 1  | Alice    |
| 7  | Bob      |
| 11 | Meir     |
```



Accepted Runtime: 120 ms

Case 1

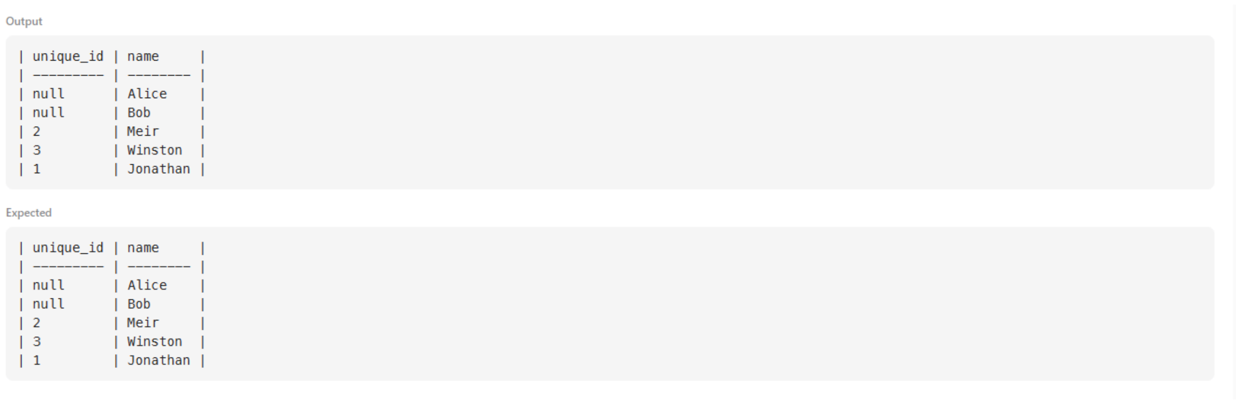
Input

Employees =

```
| id | name     |
| -- | -------- |
| 1  | Alice    |
| 7  | Bob      |
| 11 | Meir     |
| 90 | Winston  |
| 3  | Jonathan |
```

EmployeeUNI =

```
| id | unique_id |
| -- | --------- |
| 3  | 1         |
| 11 | 2         |
| 90 | 3         |
```



Output

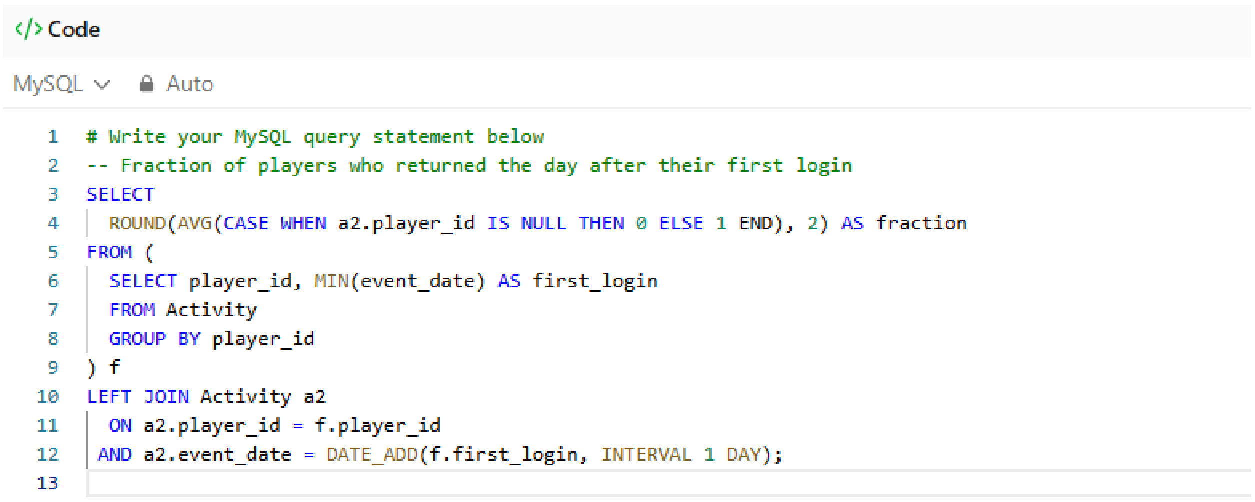
```
| unique_id | name     |
| --------- | -------- |
| null      | Alice    |
| null      | Bob      |
| 2         | Meir     |
| 3         | Winston  |
| 1         | Jonathan |
```

Expected

```
| unique_id | name     |
| --------- | -------- |
| null      | Alice    |
| null      | Bob      |
| 2         | Meir     |
| 3         | Winston  |
| 1         | Jonathan |
```

Answer#7 : 550. Game Play Analysis IV

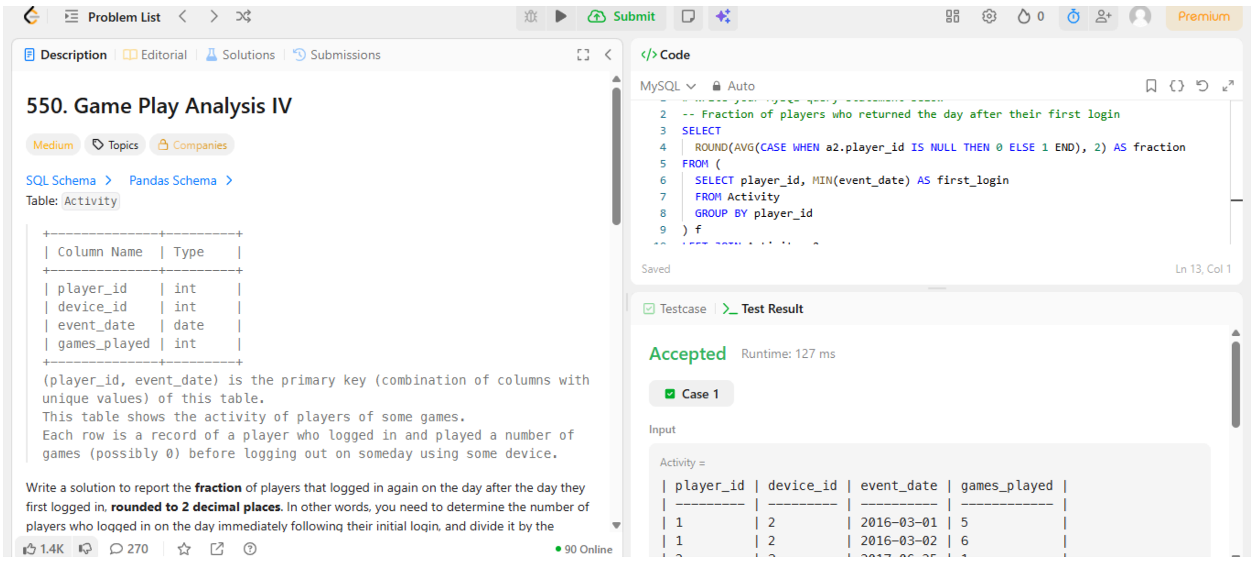
Code I used:



Code

MySQL Auto

```sql
# Write your MySQL query statement below
-- Fraction of players who returned the day after their first login
SELECT
  ROUND(AVG(CASE WHEN a2.player_id IS NULL THEN 0 ELSE 1 END), 2) AS fraction
FROM (
  SELECT player_id, MIN(event_date) AS first_login
  FROM Activity
  GROUP BY player_id
) f
LEFT JOIN Activity a2
  ON a2.player_id = f.player_id
 AND a2.event_date = DATE_ADD(f.first_login, INTERVAL 1 DAY);
```



550. Game Play Analysis IV

Medium · Topics · Companies

SQL Schema > Pandas Schema >

Table: `Activity`

```
+--------------+---------+
| Column Name  | Type    |
+--------------+---------+
| player_id    | int     |
| device_id    | int     |
| event_date   | date    |
| games_played | int     |
+--------------+---------+
(player_id, event_date) is the primary key (combination of columns with
unique values) of this table.
This table shows the activity of players of some games.
Each row is a record of a player who logged in and played a number of
games (possibly 0) before logging out on someday using some device.
```

Write 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 determine the number of players who logged in on the day immediately following their initial login, and divide it by the

```sql
-- Fraction of players who returned the day after their first login
SELECT
  ROUND(AVG(CASE WHEN a2.player_id IS NULL THEN 0 ELSE 1 END), 2) AS fraction
FROM (
  SELECT player_id, MIN(event_date) AS first_login
  FROM Activity
  GROUP BY player_id
) f
```

Saved · Ln 13, Col 1

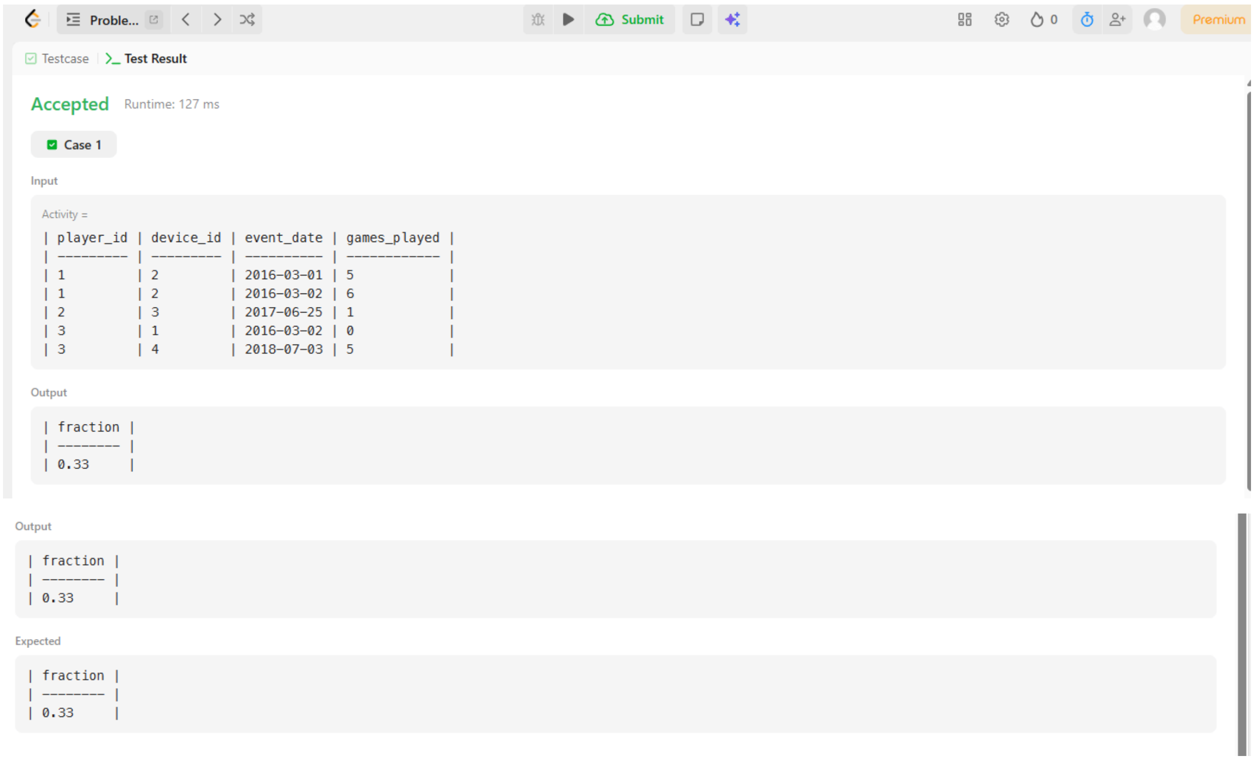
Testcase | Test Result

Accepted Runtime: 127 ms

Case 1

Input

```
Activity =
| player_id | device_id | event_date | games_played |
| --------- | --------- | ---------- | ------------ |
| 1         | 2         | 2016-03-01 | 5            |
| 1         | 2         | 2016-03-02 | 6            |
```



Accepted Runtime: 127 ms

Case 1

Input

Activity =

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

Output

```
| fraction |
| -------- |
| 0.33     |
```

Output

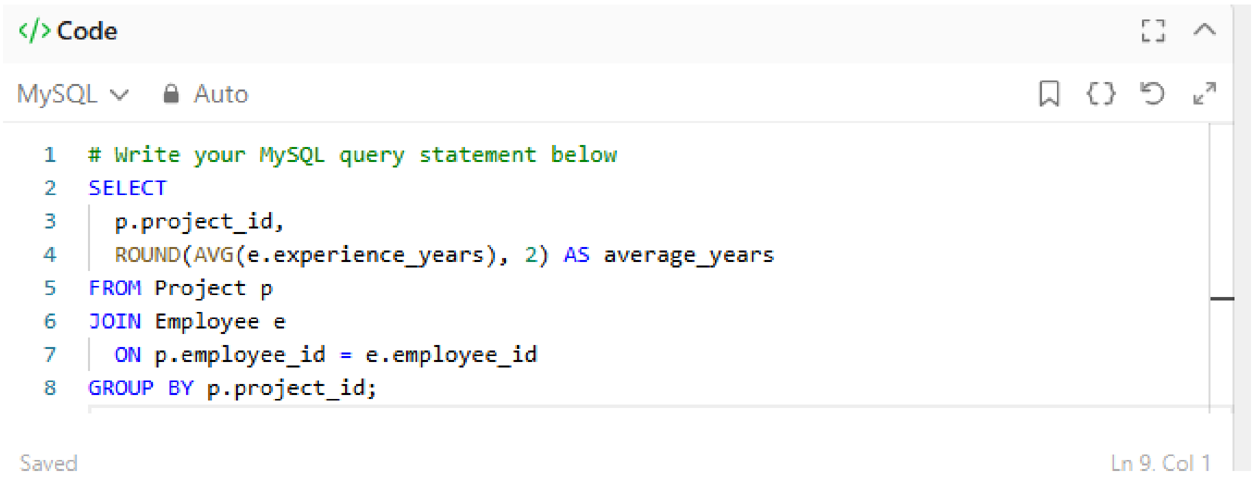
```
| fraction |
| -------- |
| 0.33     |
```

Expected

```
| fraction |
| -------- |
| 0.33     |
```

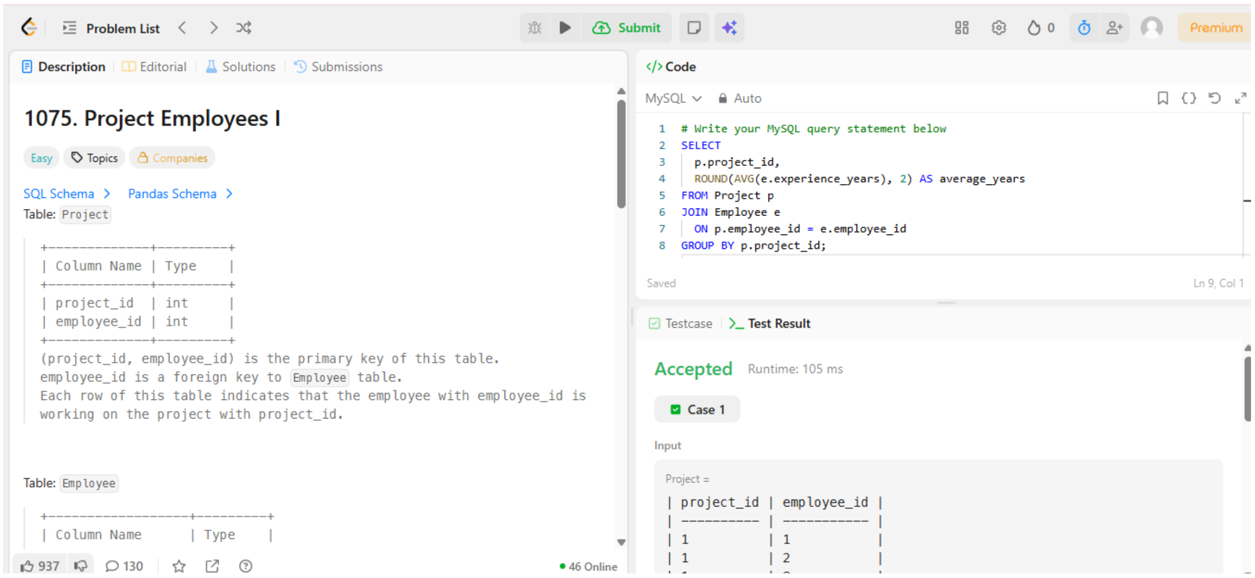
Answer#8 :1075. Project Employees I

Code I used:



```sql
# Write your MySQL query statement below
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;
```

Saved Ln 9, Col 1



Problem List | Submit | Premium

Description | Editorial | Solutions | Submissions

1075. Project Employees I

Easy | Topics | Companies

SQL Schema > Pandas Schema >

Table: `Project`

```
+-------------+---------+
| Column Name | Type    |
+-------------+---------+
| project_id  | int     |
| employee_id | int     |
+-------------+---------+
(project_id, employee_id) is the primary key of this table.
employee_id is a foreign key to Employee table.
Each row of this table indicates that the employee with employee_id is
working on the project with project_id.
```

Table: `Employee`

```
+------------------+---------+
| Column Name      | Type    |
```

937 | 130 | 46 Online

Code

MySQL | Auto

```sql
# Write your MySQL query statement below
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;
```

Saved | Ln 9, Col 1

Testcase | Test Result

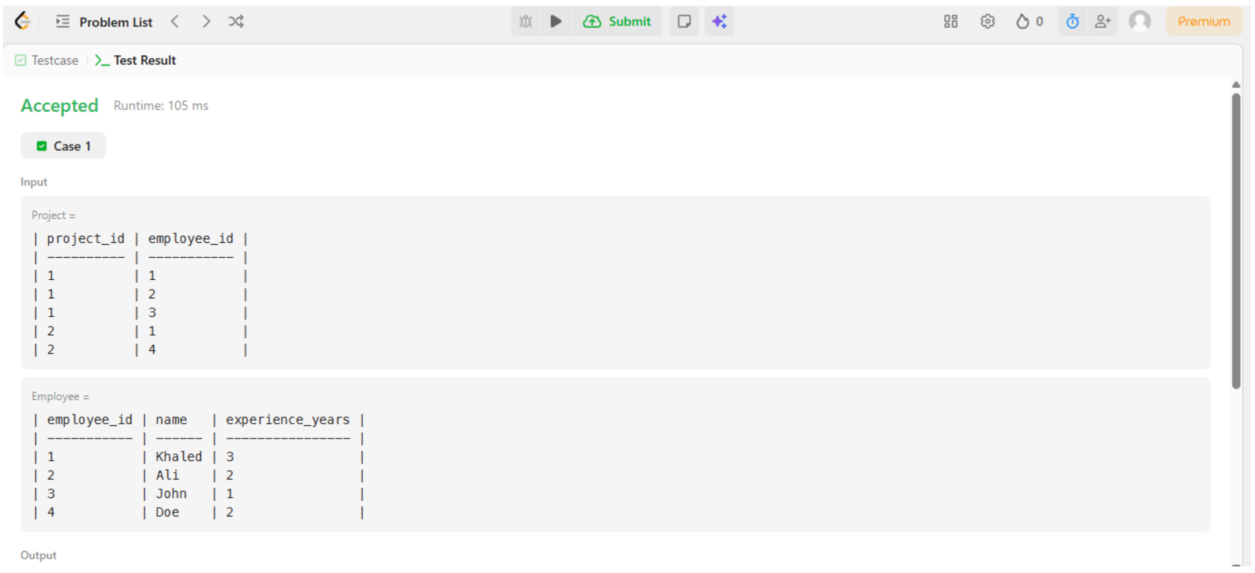
Accepted Runtime: 105 ms

Case 1

Input

Project =

```
| project_id | employee_id |
| ---------- | ----------- |
| 1          | 1           |
| 1          | 2           |
```



Problem List | Submit | Premium

Testcase | Test Result

Accepted Runtime: 105 ms

Case 1

Input

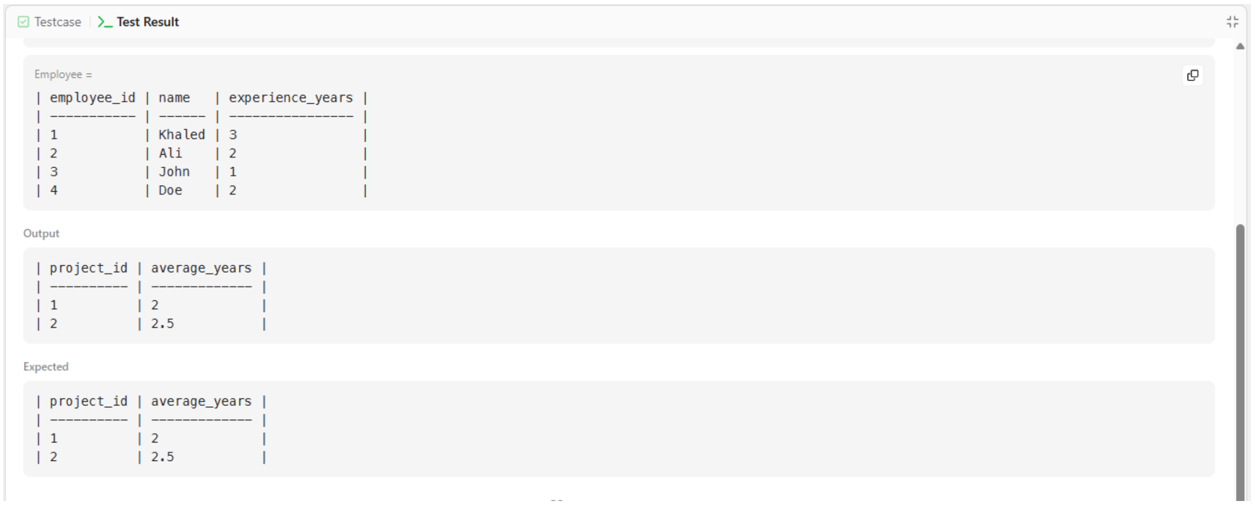
Project =

```
| project_id | employee_id |
| ---------- | ----------- |
| 1          | 1           |
| 1          | 2           |
| 1          | 3           |
| 2          | 1           |
| 2          | 4           |
```

Employee =

```
| employee_id | name   | experience_years |
| ----------- | ------ | ---------------- |
| 1           | Khaled | 3                |
| 2           | Ali    | 2                |
| 3           | John   | 1                |
| 4           | Doe    | 2                |
```

Output



Testcase | Test Result

Employee =

```
| employee_id | name   | experience_years |
| ----------- | ------ | ---------------- |
| 1           | Khaled | 3                |
| 2           | Ali    | 2                |
| 3           | John   | 1                |
| 4           | Doe    | 2                |
```

Output

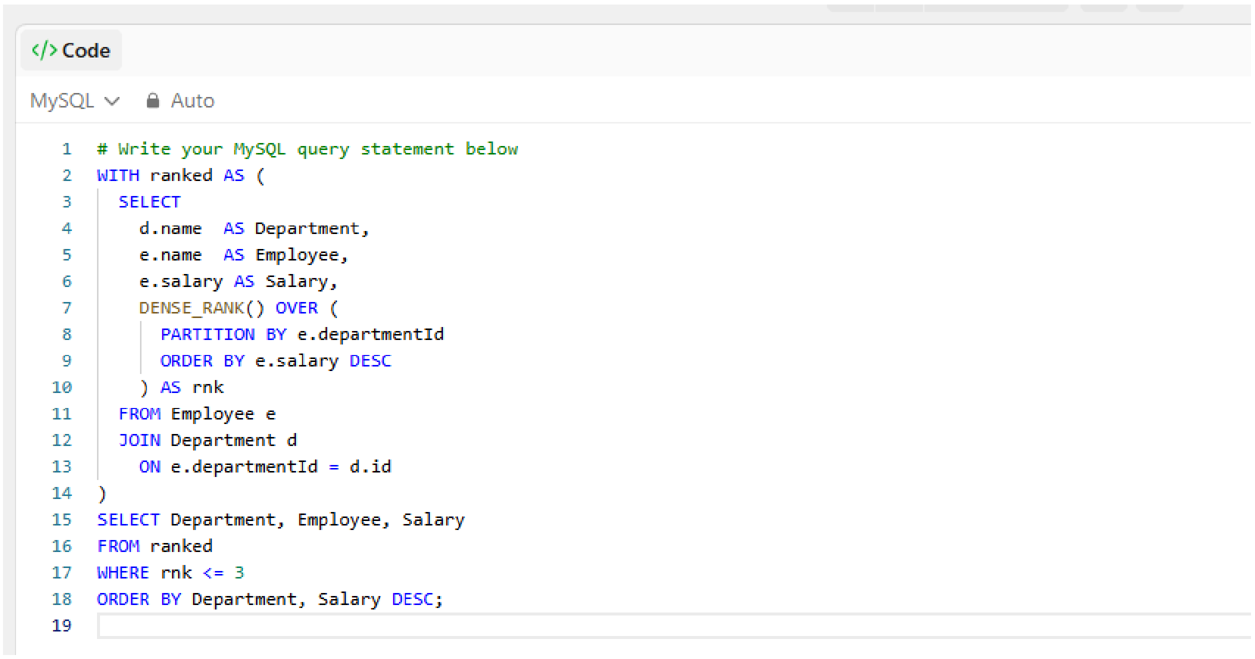
```
| project_id | average_years |
| ---------- | ------------- |
| 1          | 2             |
| 2          | 2.5           |
```

Expected

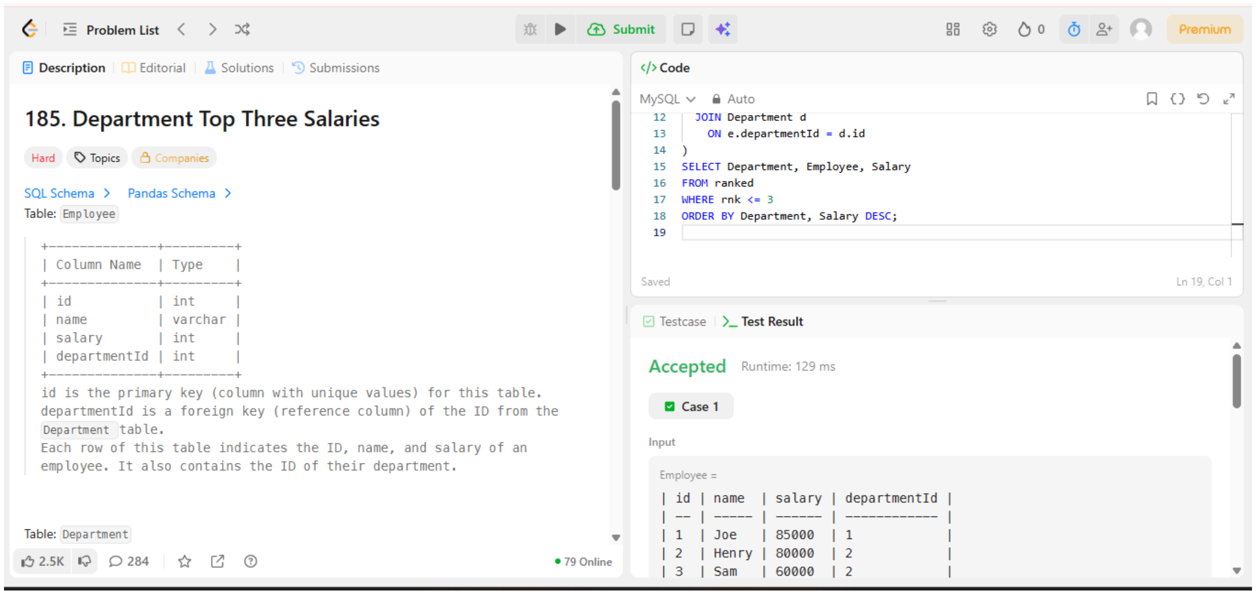
```
| project_id | average_years |
| ---------- | ------------- |
| 1          | 2             |
| 2          | 2.5           |
```

Answer#9: 185. Department Top Three Salaries

Code I used:



```sql
# Write your MySQL query statement below
WITH ranked AS (
  SELECT
    d.name  AS Department,
    e.name  AS Employee,
    e.salary AS Salary,
    DENSE_RANK() OVER (
      PARTITION BY e.departmentId
      ORDER BY e.salary DESC
    ) AS rnk
  FROM Employee e
  JOIN Department d
    ON e.departmentId = d.id
)
SELECT Department, Employee, Salary
FROM ranked
WHERE rnk <= 3
ORDER BY Department, Salary DESC;
```



185. Department Top Three Salaries

Table: `Employee`

```
+--------------+---------+
| Column Name  | Type    |
+--------------+---------+
| id           | int     |
| name         | varchar |
| salary       | int     |
| departmentId | int     |
+--------------+---------+
id is the primary key (column with unique values) for this table.
departmentId is a foreign key (reference column) of the ID from the
Department table.
Each row of this table indicates the ID, name, and salary of an
employee. It also contains the ID of their department.
```

Table: `Department`

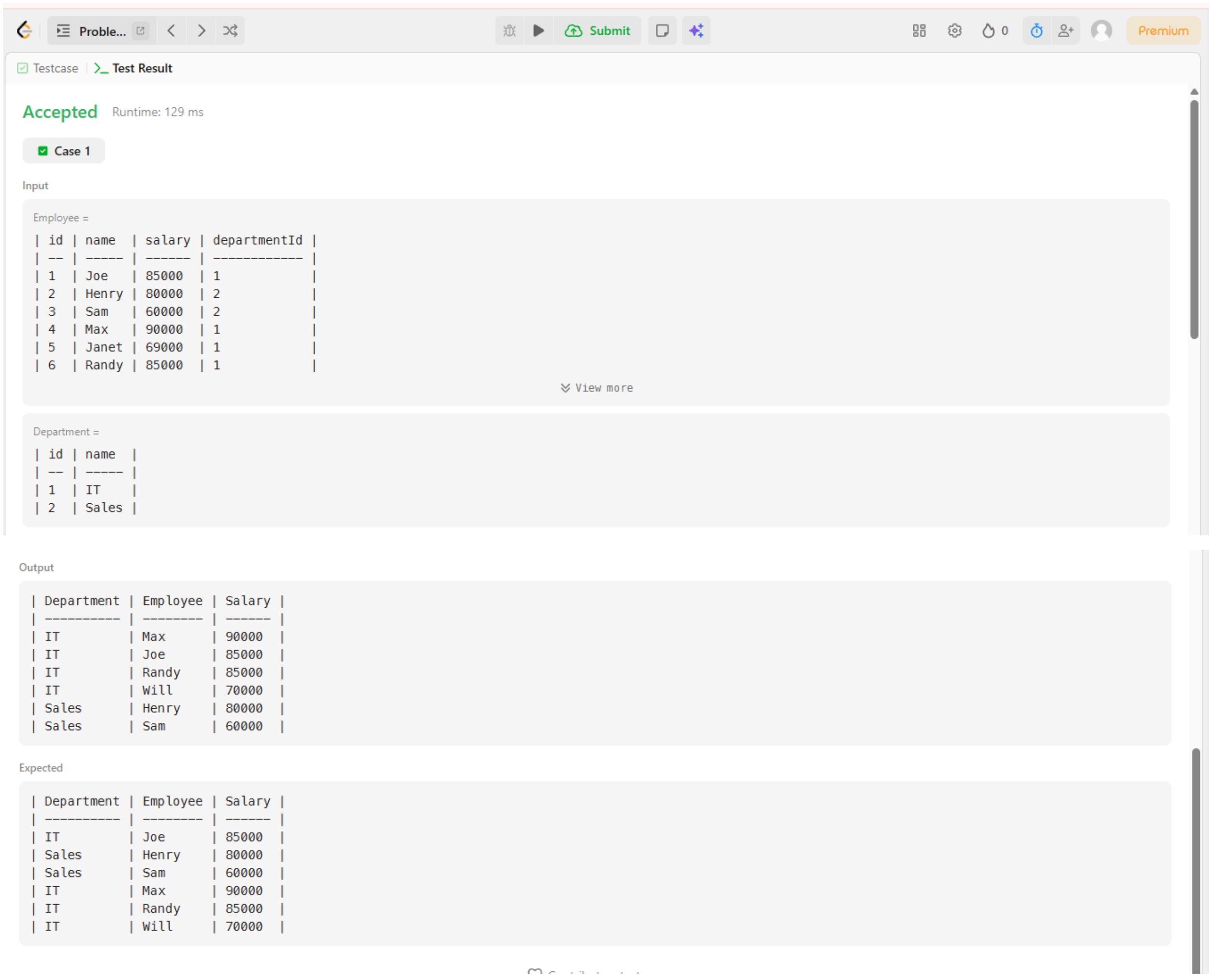
```sql
  JOIN Department d
    ON e.departmentId = d.id
)
SELECT Department, Employee, Salary
FROM ranked
WHERE rnk <= 3
ORDER BY Department, Salary DESC;
```

Accepted Runtime: 129 ms

Case 1

Input

```
Employee =
| id | name  | salary | departmentId |
| -- | ----- | ------ | ------------ |
| 1  | Joe   | 85000  | 1            |
| 2  | Henry | 80000  | 2            |
| 3  | Sam   | 60000  | 2            |
```



Accepted Runtime: 129 ms

Case 1

Input

Employee =

```
| id | name  | salary | departmentId |
| -- | ----- | ------ | ------------ |
| 1  | Joe   | 85000  | 1            |
| 2  | Henry | 80000  | 2            |
| 3  | Sam   | 60000  | 2            |
| 4  | Max   | 90000  | 1            |
| 5  | Janet | 69000  | 1            |
| 6  | Randy | 85000  | 1            |
```

View more

Department =

```
| id | name  |
| -- | ----- |
| 1  | IT    |
| 2  | Sales |
```

Output

```
| Department | Employee | Salary |
| ---------- | -------- | ------ |
| IT         | Max      | 90000  |
| IT         | Joe      | 85000  |
| IT         | Randy    | 85000  |
| IT         | Will     | 70000  |
| Sales      | Henry    | 80000  |
| Sales      | Sam      | 60000  |
```

Expected

```
| Department | Employee | Salary |
| ---------- | -------- | ------ |
| IT         | Joe      | 85000  |
| Sales      | Henry    | 80000  |
| Sales      | Sam      | 60000  |
| IT         | Max      | 90000  |
| IT         | Randy    | 85000  |
| IT         | Will     | 70000  |
```