QUESTION:

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***

+----------+---------+

| Column Name | Type |

+----------+---------+

| product_id | int |

| low_fats | enum |

| recyclable | enum |

+----------+---------+

product_id is the primary key (column with unique values) for this table.

low_fats is an ENUM (category) of type ('Y', 'N') where 'Y' means this product is low fat and 'N' means it is not.

recyclable is an ENUM (category) of types ('Y', 'N') where 'Y' means this product is recyclable and 'N' means it is not.

Intuition:

We need to find where in the table the products are both low fat and recyclable.

Approach:

Use (‘ ‘) quotation Marks in the WHERE statement to specify which Value in the column we want to get. Use the GROUP BY statement to combine the product ids Into single row results.

ANSWER:

```
SELECT product_id
FROM Products
WHERE low_fats = 'Y' AND recyclable = 'Y'
GROUP BY product_id
```

Accepted

Products =

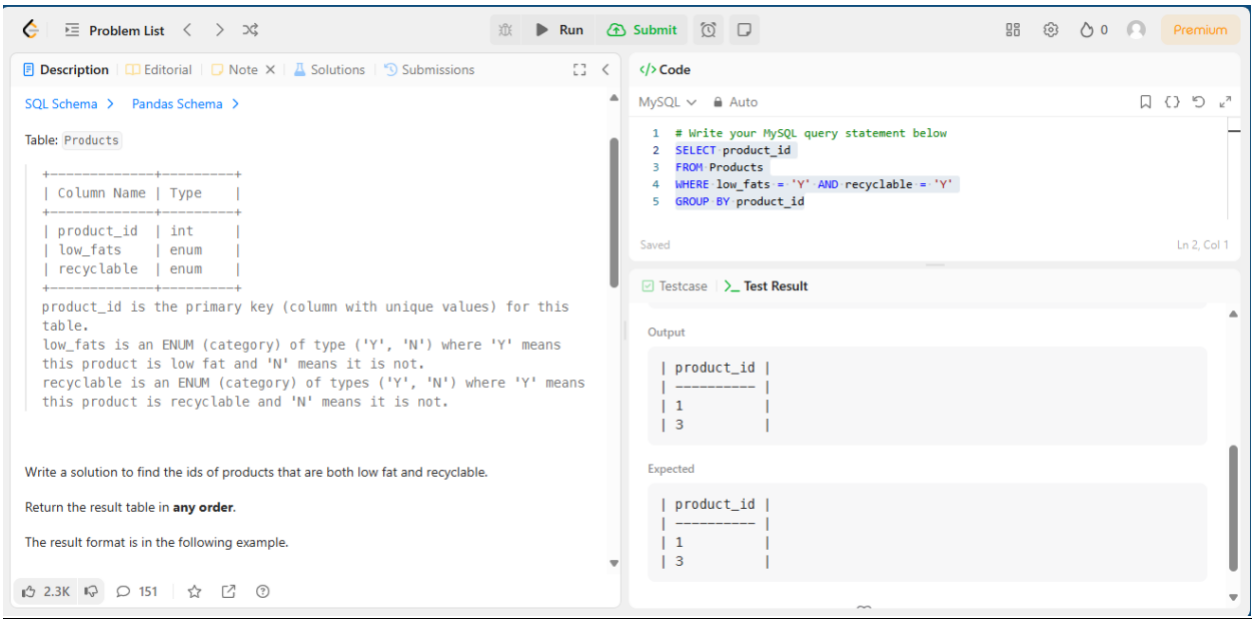
| product_id | low_fats | recyclable | | ---------- | -------- | ---------- | | 0 | Y | N | | 1 | Y | Y | | 2 | N | Y | | 3 | Y | Y | | 4 | N | N |

Output

| product_id | | ---------- | | 1 | | 3 |

Expected

| product_id | | ---------- | | 1 | | 3 |



QUESTION:

Find the names of the customer that are **not referred by** the customer with id = 2.

Return the result table in **any order**.

The result format is in the following example.

Table: ***Customer***

+------------+---------+

| Column Name | Type |

+------------+---------+

| id | int |

| name | varchar |

| referee_id | int |

+------------+---------+

In SQL, id is the primary key column for this table.

Each row of this table indicates the id of a customer, their name, and the id of the customer who referred them.

Intuition:

We want to find all customers that do not have a refree id with the value 2 although there are customers that have referee id values of null, so we have to change those to 0

Approach:

Use COALESCE to handle Null values and replace them with 0's. Use (<>) less than or greater then to get return all without 2. SELECT name because that is the only column we need.

ANSWER:

```
SELECT name
FROM Customer
WHERE COALESCE(referee_id,0) <> 2
```

Accepted

Customer =

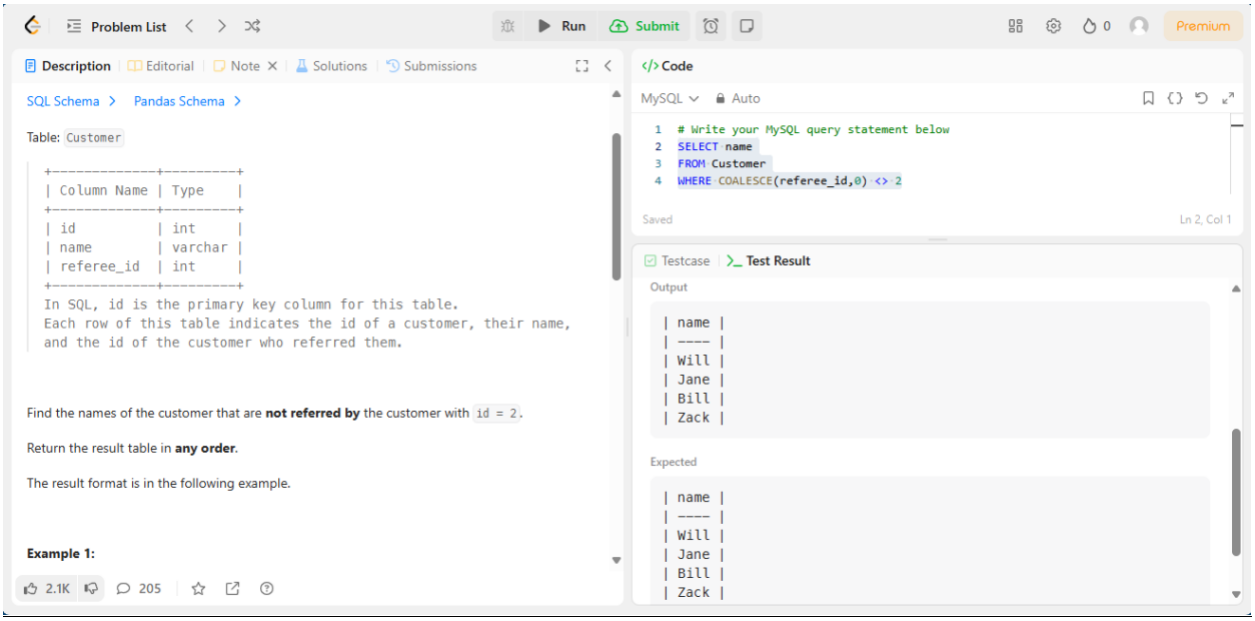
| id | name | referee_id | | -- | ---- | ---------- | | 1 | Will | null | | 2 | Jane | null | | 3 | Alex | 2 | | 4 | Bill | null | | 5 | Zack | 1 | | 6 | Mark | 2 |

Output

| name | | ---- | | Will | | Jane | | Bill | | Zack |

Expected

| name | | ---- | | Will | | Jane | | Bill | | Zack |



QUESTION:

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**.

Table: ***Sales***

+-------------+-------+

| Column Name | Type |

+-------------+-------+

| sale_id | int |

| product_id | int |

| year | int |

| quantity | int |

| price | int |

+-------------+-------+

(sale_id, year) is the primary key (combination of columns with unique values) of this table.

product_id is a foreign key (reference column) to Product table.

Each row of this table shows a sale on the product product_id in a certain year.

Note: that the price is per unit.

Table: ***Product***

+--------------+---------+

| Column Name | Type |

+--------------+---------+

| product_id | int |

| product_name | varchar |

+--------------+---------+

product_id is the primary key (column with unique values) of this table.

Each row of this table indicates the product name of each product.

Intuition:

We need to use LEFT JOIN to combine the two tables, then we can query results from there. Being that both Product_id columns are keys we will use the USING clause in our LEFT JOIN statement.

Approach:

Use p. As an indicator for The product table and s. As an indicator for the product table. Remember to clarify s and p when calling the tables.

ANSWER:

```
SELECT p.product_name, s.year, s.price
FROM Sales s
LEFT JOIN Product p USING (product_id)
```

Accepted

Sales =

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

***Product* =**

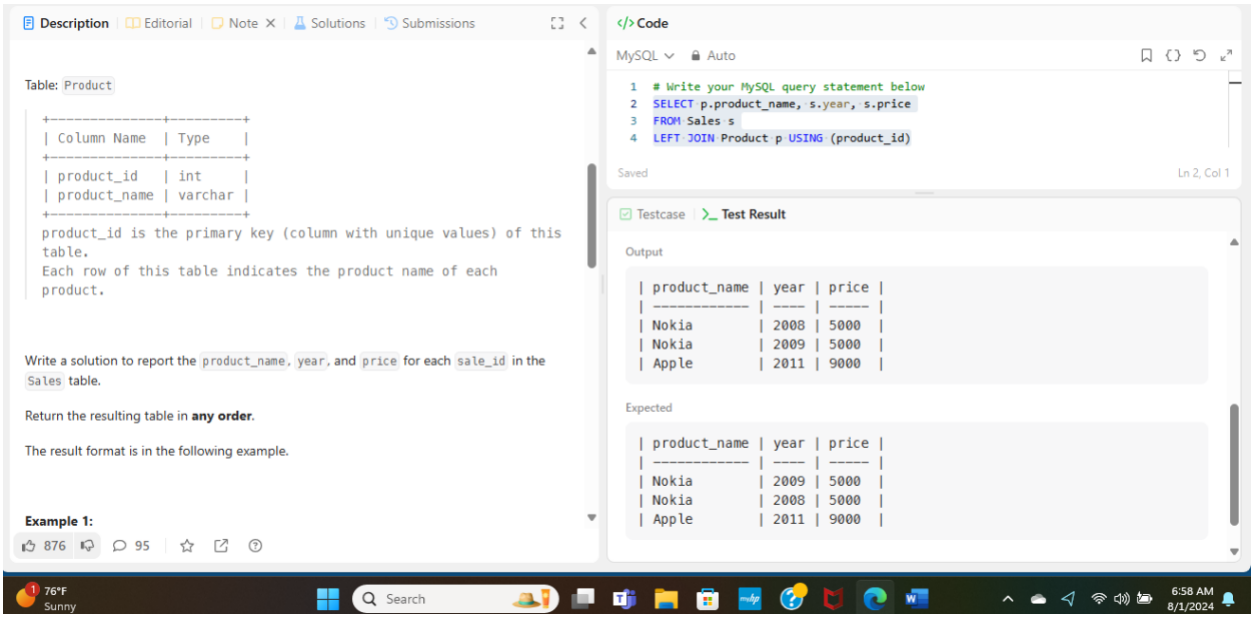
| product_id | product_name | | ---------- | ---------- | | 100 | Nokia | | 200 | Apple | | 300 | Samsung |

Output

| product_name | year | price | | ---------- | ---- | ----- | | Nokia | 2008 | 5000 | | Nokia | 2009 | 5000 | | Apple | 2011 | 9000 |

Expected

| product_name | year | price | | ---------- | ---- | ----- | | Nokia | 2009 | 5000 | | Nokia | 2008 | 5000 | | Apple | 2011 | 9000 |



QUESTION:

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**.

Table: ***Visits***

+-------------+---------+

| Column Name | Type |

+-------------+---------+

| visit_id | int |

| customer_id | int |

+-------------+---------+

visit_id is the column with unique values for this table.

This table contains information about the customers who visited the mall.

Table: ***Transactions***

+----------------+---------+

| Column Name | Type |

+----------------+---------+

| transaction_id | int |

| visit_id | int |

| amount | int |

+----------------+---------+

transaction_id is column with unique values for this table.

This table contains information about the transactions made during the visit_id.

Intuition:

We want to find the ids that are associated with the null values and group them together by the amount of times they visited.

Approach:

Once agian we are going to use LEFT JOIN, although this time we are going to add the WHERE statement and use the IS clause to find the NULL value, then GROUP BY to finish our query.

ANSWER:

```
SELECT customer_id, Count(v.visit_id) AS count_no_trans
FROM Visits v
LEFT JOIN Transactions t ON t.visit_id = v.visit_id
WHERE transaction_id IS NULL
GROUP BY customer_id
```

Accepted

Visits =

visit_id	customer_id
1	23
2	9
4	30
5	54
6	96
7	54

Transactions =

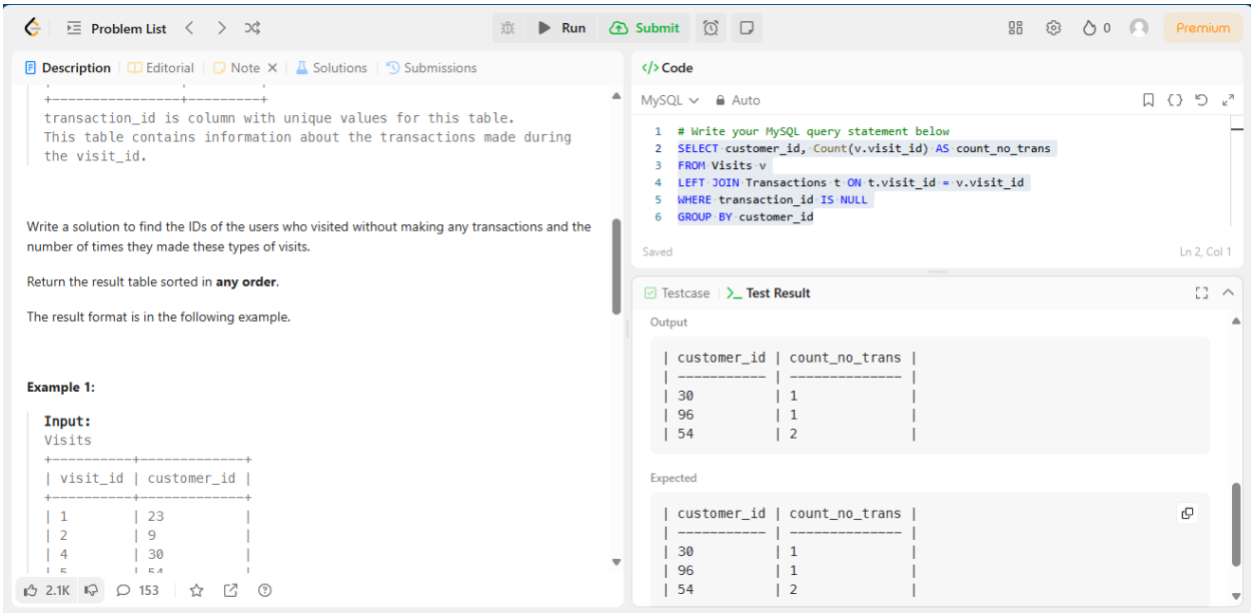
transaction_id	visit_id	amount
2	5	310
3	5	300
9	5	200
12	1	910
13	2	970

Output

customer_id	count_no_trans
30	1
96	1
54	2

Expected

customer_id	count_no_trans
30	1
96	1
54	2



QUESTION:

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.

Table***: Students***

+---------------+---------+

| Column Name | Type |

+---------------+---------+

| student_id | int |

| student_name | varchar |

+---------------+---------+

student_id is the primary key (column with unique values) for this table.

Each row of this table contains the ID and the name of one student in the school.

Table***: Subjects***

+--------------+---------+

| Column Name | Type |

+--------------+---------+

| subject_name | varchar |

+--------------+---------+

subject_name is the primary key (column with unique values) for this table.

Each row of this table contains the name of one subject in the school.

Table: ***Examinations***

+--------------+---------+

| Column Name | Type |

+--------------+---------+

| student_id | int |

| subject_name | varchar |

+--------------+---------+

There is no primary key (column with unique values) for this table. It may contain duplicates.

Each student from the Students table takes every course from the Subjects table.

Each row of this table indicates that a student with ID student_id attended the exam of subject_name.

Intuition:

We need to join all three tables then count the number of of times each student has taken the test indicated by the subject Id.

Approach:

Not all columns match and the Examinations table doesn't have a primary key so we are going to combine CROSS JOIN AND LEFT JOIN so we can specify which columns match which from the Examinations table.

ANSWER:

```
SELECT st.student_id, st.student_name, su.subject_name, COUNT(e.subject_name) AS
attended_exams
FROM Students st
CROSS JOIN Subjects su
LEFT JOIN Examinations e ON su.subject_name=e.subject_name AND
e.student_id=st.student_id
GROUP BY st.student_id, st.student_name, su.subject_name
ORDER BY st.student_id, su.subject_name
```

Accepted

Students =

| student_id | student_name | | ---------- | ---------- | | 1 | Alice | | 2 | Bob | | 13 | John | | 6 | Alex |

Subjects =

| subject_name | | ---------- | | Math | | Physics | | Programming |

Examinations =

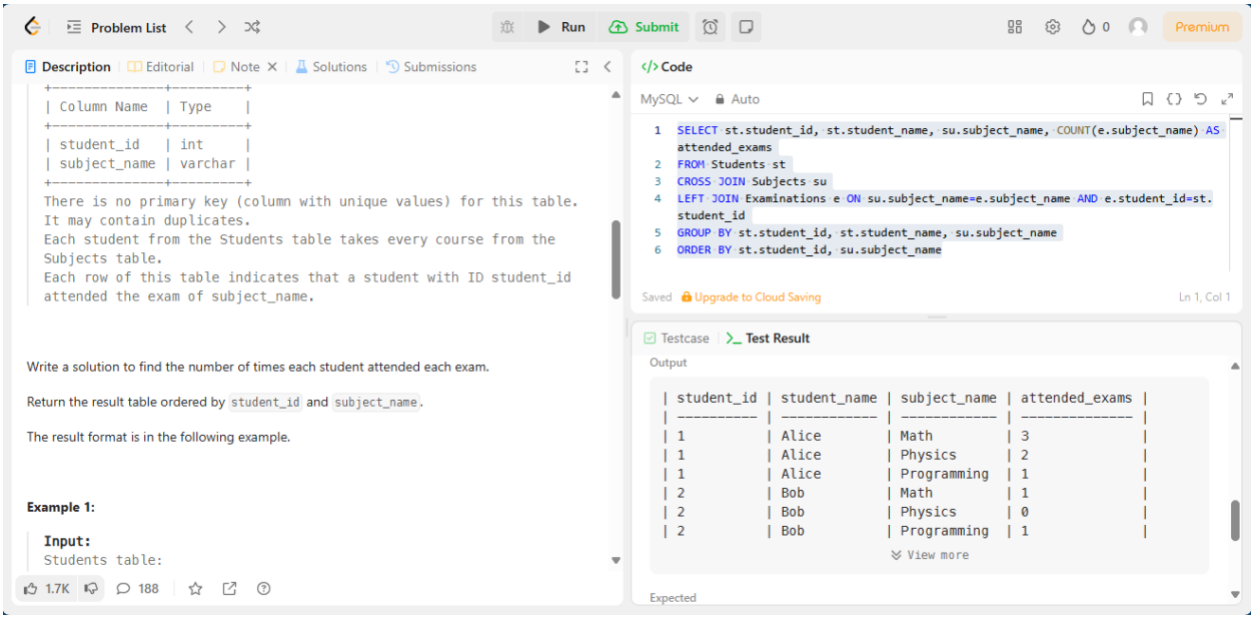
| student_id | subject_name | | ---------- | ---------- | | 1 | Math | | 1 | Physics | | 1 | Programming | | 2 | Programming | | 1 | Physics | | 1 | Math | | 13 | Math | | 13 | Programming | | 13 | Physics | | 2 | Math | | 1 | Math |

Output

| student_id | student_name | subject_name | attended_exams | | ---------- | ---------- | ---------- | ---------- | | 1 | Alice | Math | 3 | | 1 | Alice | Physics | 2 | | 1 | Alice | Programming | 1 | | 2 | Bob | Math | 1 | | 2 | Bob | Physics | 0 | | 2 | Bob | Programming | 1 | | 6 | Alex | Math | 0 | | 6 | Alex | Physics | 0 | | 6 | Alex | Programming | 0 | | 13 | John | Math | 1 | | 13 | John | Physics | 1 | | 13 | John | Programming | 1 |

Expected

| student_id | student_name | subject_name | attended_exams | | ---------- | ---------- | ---------- | ---------- | | 1 | Alice | Math | 3 | | 1 | Alice | Physics | 2 | | 1 | Alice | Programming | 1 | | 2 | Bob | Math | 1 | | 2 | Bob | Physics | 0 | | 2 | Bob | Programming | 1 | | 6 | Alex | Math | 0 | | 6 | Alex | Physics | 0 | | 6 | Alex | Programming | 0 | | 13 | John | Math | 1 | | 13 | John | Physics | 1 | | 13 | John | Programming | 1 |



QUESTION:

Write a solution to find the ***confirmation rate*** *of each user.*

Return the result table in ***any order***

Table: ***Signups***

+----------------+----------+

| Column Name | Type |

+----------------+----------+

| user_id | int |

| time_stamp | datetime |

+----------------+----------+

Each row contains information about the signup time for the user with ID user_id.

Table: ***Confirmations***

+----------------+----------+

| Column Name | Type |

+----------------+----------+

| user_id | int |

| time_stamp | datetime |

| action | ENUM |

+----------------+----------+

(user_id, time_stamp) is the primary key (combination of columns with unique values) for this table.

user_id is a foreign key (reference column) to the Signups table.

action is an ENUM (category) of the type ('confirmed', 'timeout')

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.

Intuition:

We need to find the confirmation rate which will be the average of confirmed values in the action column from the confirmations table.

Approach:

Using AVG(if(...)) we can find the average if the value if it returns a 1 signifying the value is confirmed. 0 returned is not confirmed. Then ROUND those values by 2 decimal places.

ANSWER:

```
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 1
```

Accepted

Signups =

| user_id | time_stamp | | ------- | ------------------- | | 3 | 2020-03-21 10:16:13 | | 7 | 2020-01-04 13:57:59 | | 2 | 2020-07-29 23:09:44 | | 6 | 2020-12-09 10:39:37 |

Confirmations =

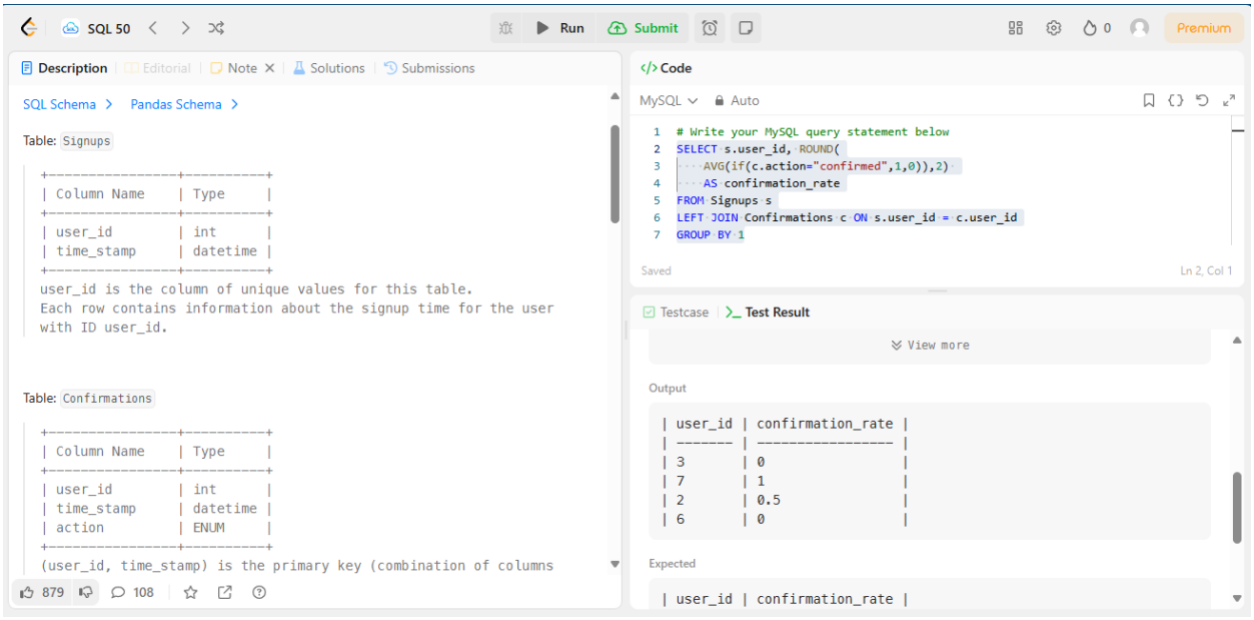
| user_id | time_stamp | action | | ------- | ------------------- | --------- | | 3 | 2021-01-06 03:30:46 | timeout | | 3 | 2021-07-14 14:00:00 | timeout | | 7 | 2021-06-12 11:57:29 | confirmed | | 7 | 2021-06-13 12:58:28 | confirmed | | 7 | 2021-06-14 13:59:27 | confirmed | | 2 | 2021-01-22 00:00:00 | confirmed |

Output

| user_id | confirmation_rate | | ------- | ----------------- | | 3 | 0 | | 7 | 1 | | 2 | 0.5 | | 6 | 0 |

Expected

| user_id | confirmation_rate | | ------- | ----------------- | | 6 | 0 | | 3 | 0 | | 7 | 1 | | 2 | 0.5 |



QUESTION:

Write a solution to report the movies with an odd-numbered ID and a description that is not "boring".

Table: ***Cinema***

+----------------+----------+

| Column Name | Type |

+----------------+----------+

| id | int |

| movie | varchar |

| description | varchar |

| rating | float |

+----------------+----------+

id is the primary key (column with unique values) for this table.

Each row contains information about the name of a movie, its genre, and its rating.

rating is a 2 decimal places float in the range [0, 10]

Return the result table ordered by rating **in descending order**.

Intuition:

The main part we need to focus on in this query is returning the odd number IDs. If you divide even numbers by 2 you will always have a reminder of 0, odd numbers will have a non zero reminder.

Approach:

In the WHERE clause, in addition to the description (!=) not equal to 'boring', (id % 2) <>0 IDs divided by 2 with a reminder less than or greater than 0

ANSWER:

```
SELECT *
FROM Cinema
WHERE (id % 2) <> 0 AND description != 'boring'
ORDER BY rating DESC
```

Accepted

cinema =

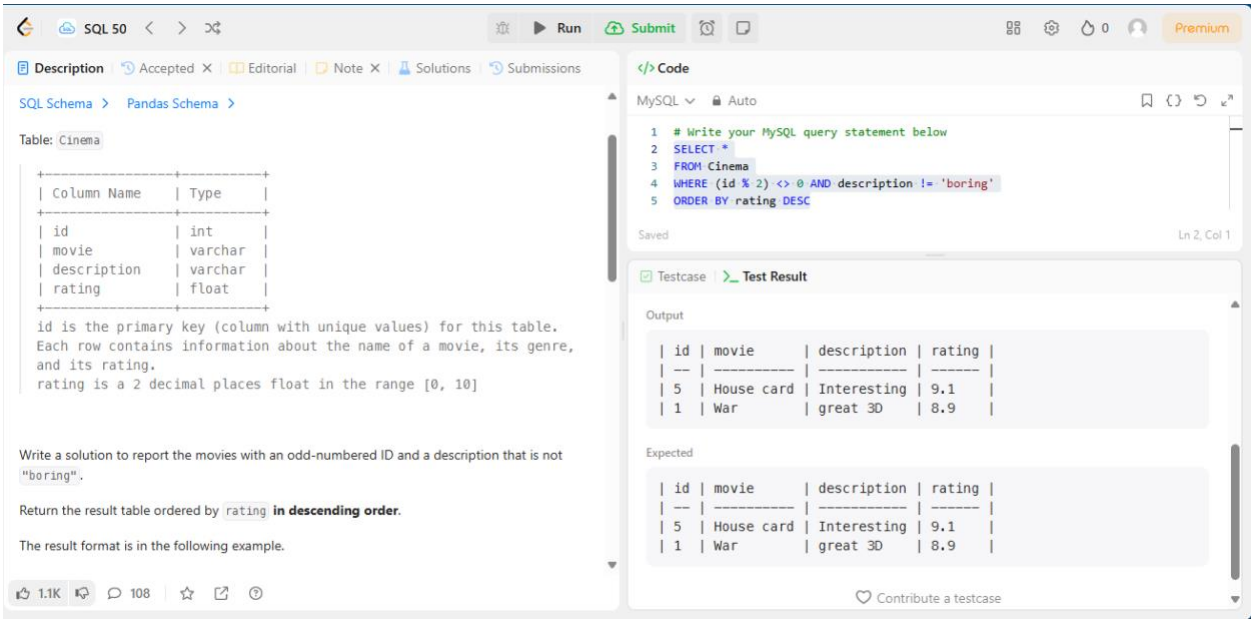
| id | movie | description | rating | | -- | ---------- | ----------- | ------ | | 1 | War | great 3D | 8.9 | | 2 | Science | fiction | 8.5 | | 3 | irish | boring | 6.2 | | 4 | Ice song | Fantacy | 8.6 | | 5 | House card | Interesting | 9.1 |

Output

| id | movie | description | rating | | -- | ---------- | ----------- | ------ | | 5 | House card | Interesting | 9.1 | | 1 | War | great 3D | 8.9 |

Expected

| id | movie | description | rating | | -- | ---------- | ----------- | ------ | | 5 | House card | Interesting | 9.1 | | 1 | War | great 3D | 8.9 |



QUESTION:

Write an SQL query that reports the **average** experience years of all the employees for each project, **rounded to 2 digits**.

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 |

+------------------+---------+

| employee_id | int |

| name | varchar |

| experience_years | int |

+------------------+---------+

employee_id is the primary key of this table. It's guaranteed that experience_years is not NULL.

Each row of this table contains information about one employee.

Return the result table in **any order**.

Intuition:

We will use LEFT JOIN in this query because we want to show the project_id from one table and the aggregated experience_years from the other table

Approach:

Simple ROUND and AVG functions that we've used before, make sure to group the results by the project_id numbers.

ANSWER:

```
SELECT p.project_id, ROUND(
    AVG(e.experience_years),2) AS average_years
FROM Project p
LEFT JOIN Employee e ON p.employee_id=e.employee_id
GROUP BY p.project_id
```

Accepted

Project =

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

Employee =

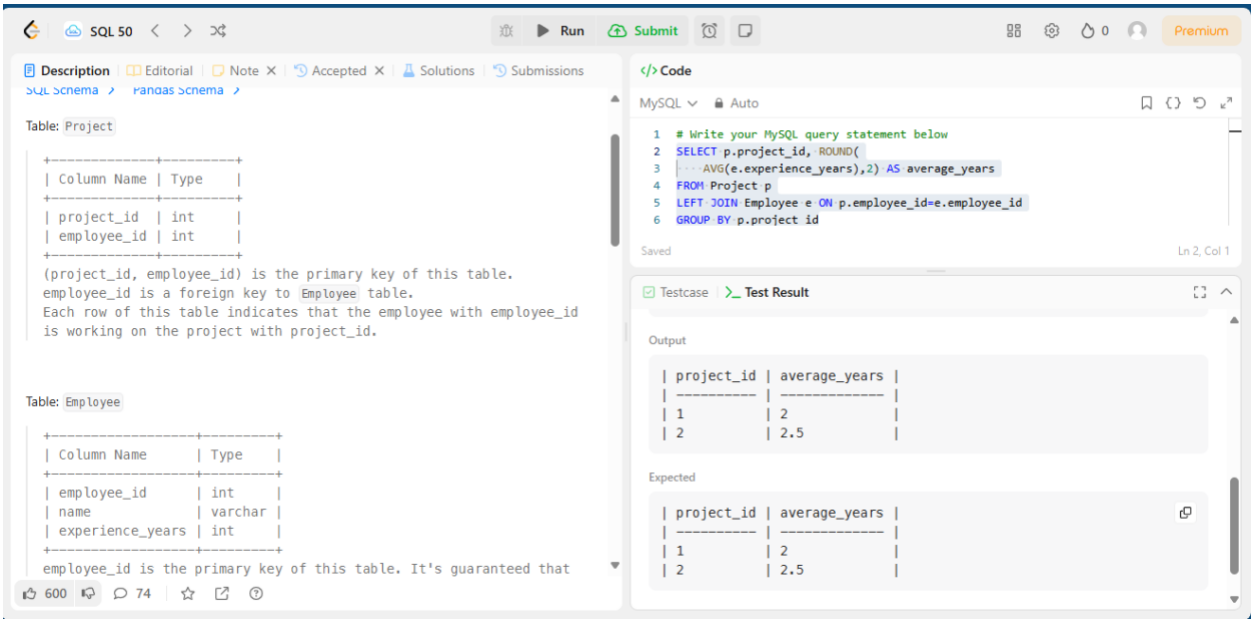
| employee_id | name | experience_years | | ---------- | ------ | ---------- | | 1 | Khaled | 3 | | 2 | Ali | 2 | | 3 | John | 1 | | 4 | Doe | 2 |

Output

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

Expected

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



QUESTION:

Write a solution to find the average selling price for each product. average_price should be **rounded to 2 decimal places**.

Table: ***Prices***

+---------------+---------+

| Column Name | Type |

+---------------+---------+

| product_id | int |

| start_date | date |

| end_date | date |

| price | int |

+---------------+---------+

(product_id, start_date, end_date) is the primary key (combination of columns with unique values) for this table.

Each row of this table indicates the price of the product_id in the period from start_date to end_date.

For each product_id there will be no two overlapping periods. That means there will be no two intersecting periods for the same product_id.

Table: ***UnitsSold***

+---------------+---------+

| Column Name | Type |

+---------------+---------+

| product_id | int |

| purchase_date | date |

| units | int |

+---------------+---------+

Each row of this table indicates the date, units, and product_id of each product sold.

Return the result table in **any order**.

Intuition:

We need to add in a formula to calculate the average price of each product. The prices change based on when they were purchased an some purchase have multiple of the same products.

Approach:

The formula we are going to use is the total amount of units at their current price divided by the total amount of units. We also need to specify that the purchase dates are in between the start_date and end_date of our Prices table. This will let the system know the current price at time of purchase.

ANSWER:

```
SELECT p.product_id, COALESCE(ROUND(
    SUM(u.units * p.price)/SUM(u.units),2),0)
    AS average_price
FROM Prices p
LEFT JOIN UnitsSold 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
```

Accepted

Prices =

| product_id | start_date | end_date | price | | ---------- | ---------- | ---------- | ----- | | 1 | 2019-02-17 | 2019-02-28 | 5 | | 1 | 2019-03-01 | 2019-03-22 | 20 | | 2 | 2019-02-01 | 2019-02-20 | 15 | | 2 | 2019-02-21 | 2019-03-31 | 30 |

***UnitsSold* =**

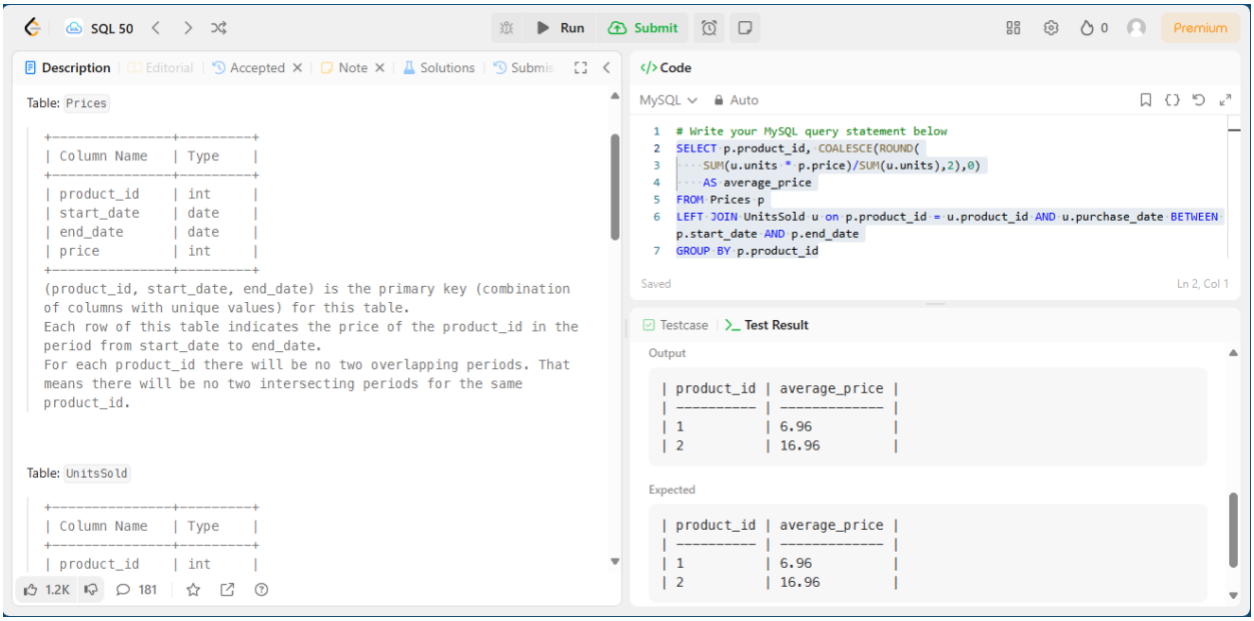
| product_id | purchase_date | units | | ---------- | ------------- | ----- | | 1 | 2019-02-25 | 100 | | 1 | 2019-03-01 | 15 | | 2 | 2019-02-10 | 200 | | 2 | 2019-03-22 | 30 |

Output

| product_id | average_price | | ---------- | ------------- | | 1 | 6.96 | | 2 | 16.96 |

Expected

| product_id | average_price | | ---------- | ------------- | | 1 | 6.96 | | 2 | 16.96 |



QUESTION:

Write a solution to find the percentage of the users registered in each contest rounded to **two decimals**.

Table: ***Users***

+-------------+---------+

| Column Name | Type |

+-------------+---------+

| user_id | int |

| user_name | varchar |

+-------------+---------+

user_id is the primary key (column with unique values) for this table.

Each row of this table contains the name and the id of a user.

Table: ***Register***

+-------------+---------+

| Column Name | Type |

+-------------+---------+

| contest_id | int |

| user_id | int |

+-------------+---------+

(contest_id, user_id) is the primary key (combination of columns with unique values) for this table.

Each row of this table contains the id of a user and the contest they registered into.

Return the result table ordered by percentage in **descending order**. In case of a tie, order it by contest_id in **ascending order**.

Intuition:

We don't need to show columns from both tables, we are just comparing the ids to their aggregated percentage results. We can use a sub query.

Approach:

We need to divide the COUNT OF unique Id's in the Register table, multiplied by 100. By the COUNT of all Id's in the Users table and call that the percentage column.

ANSWER:

```
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
```

Accepted

Users =

| user_id | user_name | | ------- | --------- | | 6 | Alice | | 2 | Bob | | 7 | Alex |

Register =

| contest_id | user_id | | ---------- | ------- | | 215 | 6 | | 209 | 2 | | 208 | 2 | | 210 | 6 | | 208 | 6 | | 209 | 7 |

Output

| contest_id | percentage | | ---------- | ---------- | | 208 | 100 | | 209 | 100 | | 210 | 100 | | 215 | 66.67 | | 207 | 33.33 |

Expected

| contest_id | percentage | | ---------- | ---------- | | 208 | 100 | | 209 | 100 | | 210 | 100 | | 215 | 66.67 | | 207 | 33.33 |