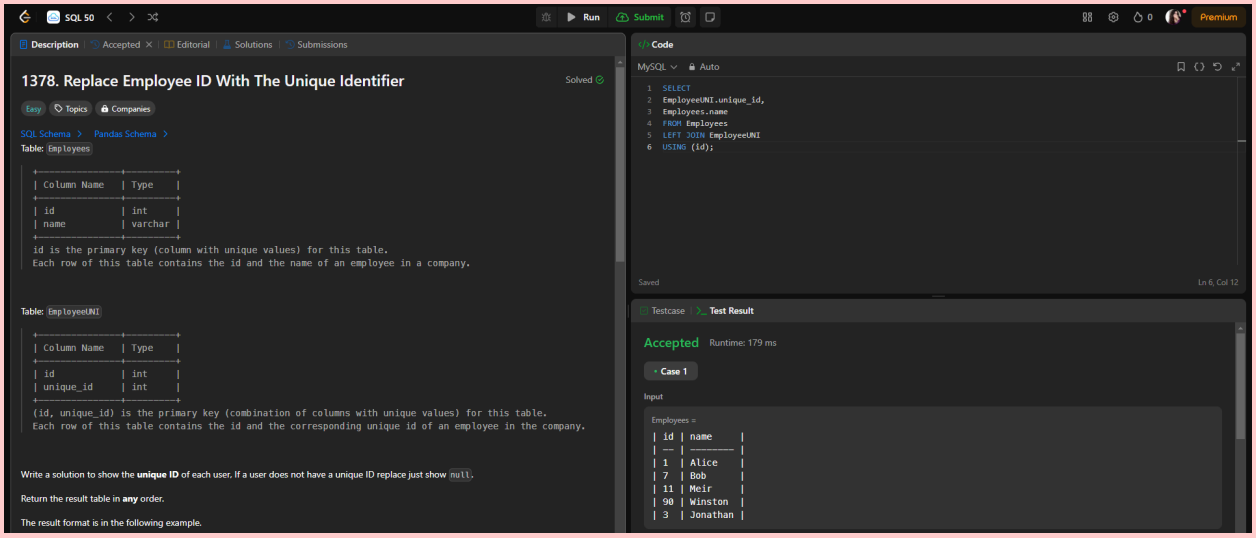
# Question 1

Complete the ‘Basic Joins’ section of SQL 50 Badge on leetcode.

## 1378. Replace Employee ID With The Unique Identifier



1378. Replace Employee ID With The Unique Identifier

Solved

Easy | Topics | Companies

SQL Schema | Pandas Schema

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 |
| --- | --- |
| id | int |
| unique_id | int |

(id, unique_id) is the primary key (combination of columns with unique values) for this table.
Each row of this table contains the id and the corresponding unique id of an employee in the company.

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.

The result format is in the following example.

```
SELECT
EmployeeUNI.unique_id,
Employees.name
FROM Employees
LEFT JOIN EmployeeUNI
USING (id);
```

Accepted Runtime: 179 ms

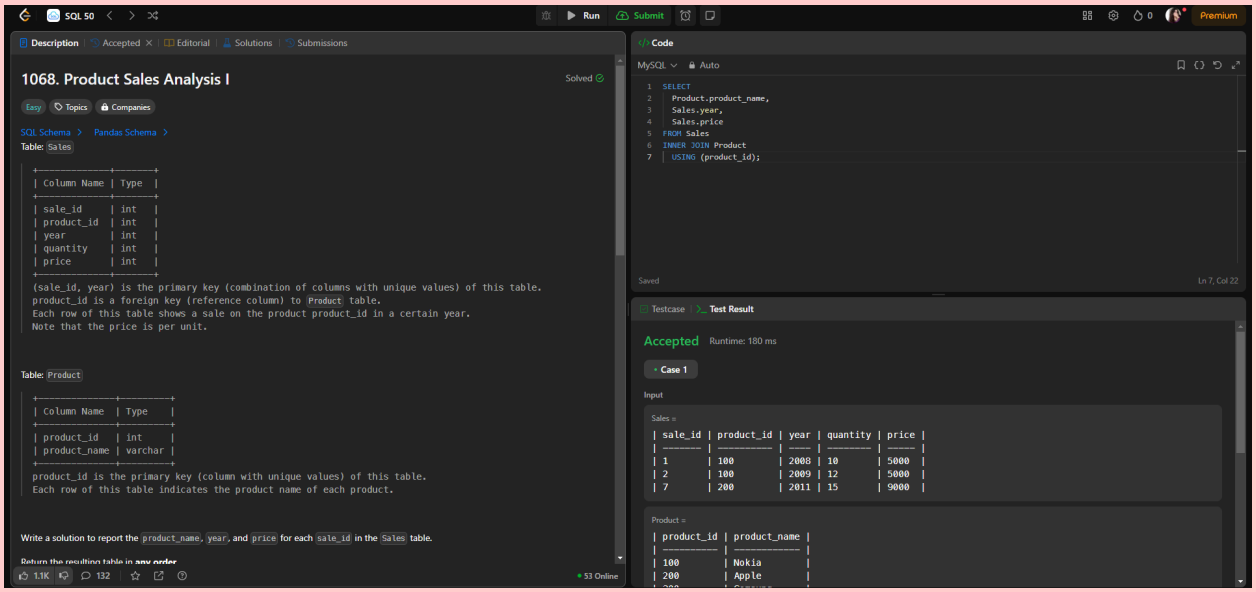
Case 1

Input

Employees =

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

## 1068. Product Sales Analysis I



1068. Product Sales Analysis I

Solved

Easy | Topics | Companies

SQL Schema | Pandas Schema

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.

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**.

```
SELECT
  Product.product_name,
  Sales.year,
  Sales.price
FROM Sales
INNER JOIN Product
  USING (product_id);
```

Accepted Runtime: 180 ms

Case 1

Input

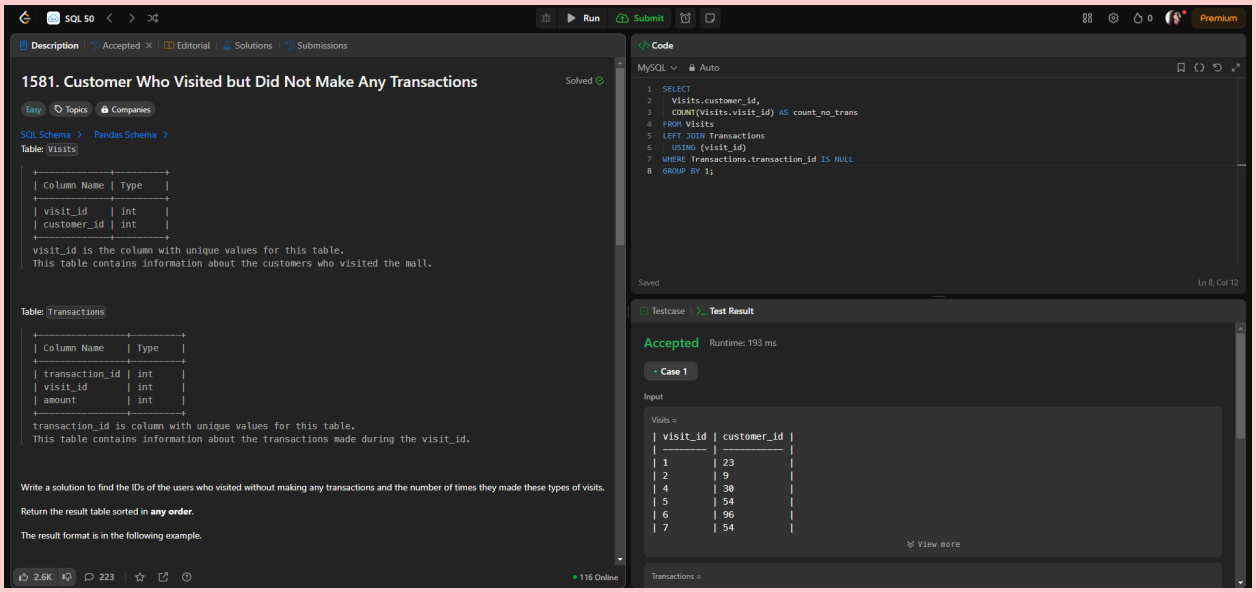
Sales =

| 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 |

## 1581. Customer Who Visited but Did Not Make Any Transactions



1581. Customer Who Visited but Did Not Make Any Transactions

Solved

Easy | Topics | Companies

SQL Schema | Pandas Schema

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.

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**.

The result format is in the following example.

```
SELECT
  Visits.customer_id,
  COUNT(Visits.visit_id) AS count_no_trans
FROM Visits
LEFT JOIN Transactions
  USING (visit_id)
WHERE Transactions.transaction_id IS NULL
GROUP BY 1;
```

Accepted Runtime: 193 ms

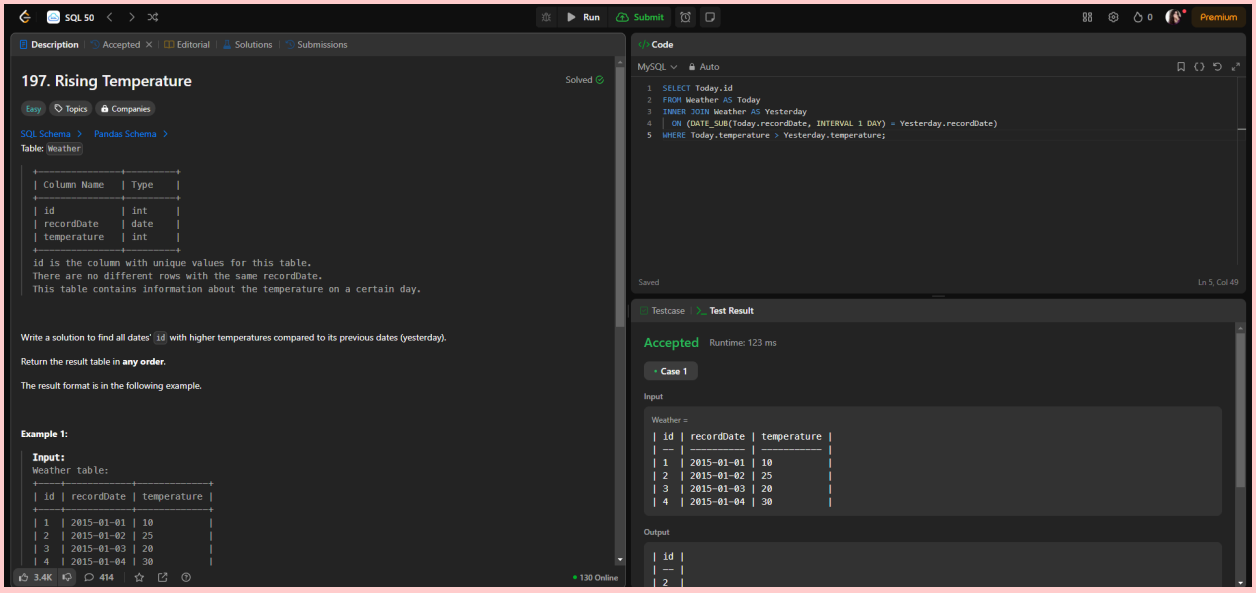
Case 1

Input

Visits =

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

## 197. Rising Temperature



197. Rising Temperature

Solved

Easy | Topics | Companies

SQL Schema | Pandas Schema

Table: Weather

| Column Name | Type |
| --- | --- |
| id | int |
| recordDate | date |
| temperature | int |

id is the column with unique values for this table.
There are no different rows with the same recordDate.
This table contains information about the temperature on a certain day.

Write a solution to find all dates' `id` with higher temperatures compared to its previous dates (yesterday).

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

The result format is in the following example.

**Example 1:**

**Input:**
Weather table:

| id | recordDate | temperature |
| --- | --- | --- |
| 1 | 2015-01-01 | 10 |
| 2 | 2015-01-02 | 25 |
| 3 | 2015-01-03 | 20 |
| 4 | 2015-01-04 | 30 |

```
SELECT Today.id
FROM Weather AS Today
INNER JOIN Weather AS Yesterday
  ON (DATE_SUB(Today.recordDate, INTERVAL 1 DAY) = Yesterday.recordDate)
WHERE Today.temperature > Yesterday.temperature;
```

Accepted Runtime: 123 ms

Case 1

Input

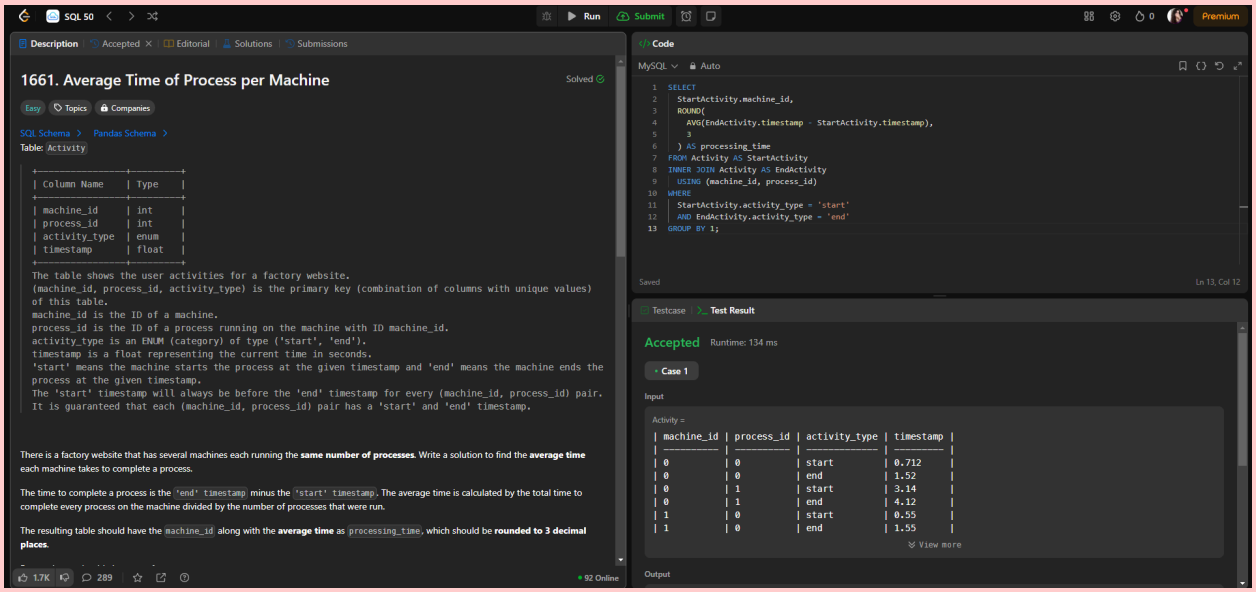
Weather =

| id | recordDate | temperature |
| --- | --- | --- |
| 1 | 2015-01-01 | 10 |
| 2 | 2015-01-02 | 25 |
| 3 | 2015-01-03 | 20 |
| 4 | 2015-01-04 | 30 |

Output

| id |
| --- |
| 2 |

## 1661. Average Time of Process per Machine



1661. Average Time of Process per Machine

Solved

Easy | Topics | Companies

SQL Schema | Pandas Schema

Table: Activity

| Column Name | Type |
| --- | --- |
| machine_id | int |
| process_id | int |
| activity_type | enum |
| timestamp | float |

The table shows the user activities for a factory website.
(machine_id, process_id, activity_type) is the primary key (combination of columns with unique values) of this table.
machine_id is the ID of a machine.
process_id is the ID of a process running on the machine with ID machine_id.
activity_type is an ENUM (category) of type ('start', 'end').
timestamp is a float representing the current time in seconds.
'start' means the machine starts the process at the given timestamp and 'end' means the machine ends the process at the given timestamp.
The 'start' timestamp will always be before the 'end' timestamp for every (machine_id, process_id) pair.
It is guaranteed that each (machine_id, process_id) pair has a 'start' and 'end' timestamp.

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**.

```
SELECT
  StartActivity.machine_id,
  ROUND(
    AVG(EndActivity.timestamp - StartActivity.timestamp),
    3
  ) AS processing_time
FROM Activity AS StartActivity
INNER JOIN Activity AS EndActivity
  USING (machine_id, process_id)
WHERE
  StartActivity.activity_type = 'start'
  AND EndActivity.activity_type = 'end'
GROUP BY 1;
```

Accepted Runtime: 134 ms

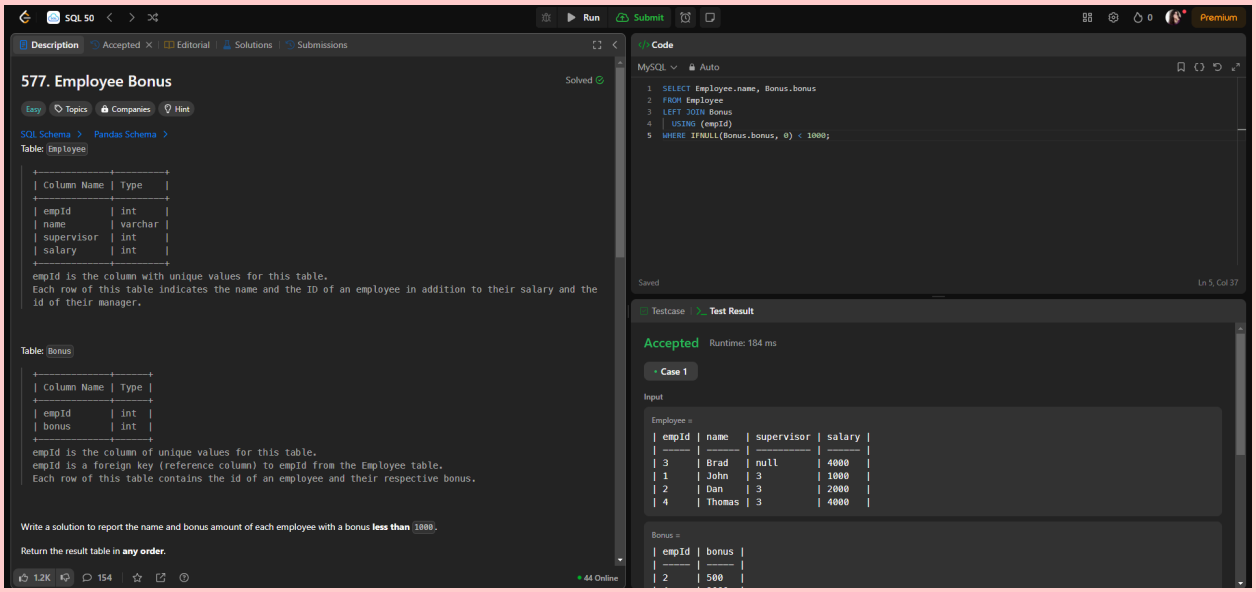
Case 1

Input

Activity =

| machine_id | process_id | activity_type | timestamp |
| --- | --- | --- | --- |
| 0 | 0 | start | 0.712 |
| 0 | 0 | end | 1.52 |
| 0 | 1 | start | 3.14 |
| 0 | 1 | end | 4.12 |
| 1 | 0 | start | 0.55 |
| 1 | 0 | end | 1.55 |

## 577. Employee Bonus



577. Employee Bonus

Solved

Easy | Topics | Companies | Hint

SQL Schema | Pandas Schema

Table: Employee

| Column Name | Type |
| --- | --- |
| empId | int |
| name | varchar |
| supervisor | int |
| salary | int |

empId is the column with unique values for this table.
Each row of this table indicates the name and the ID of an employee in addition to their salary and the id of their manager.

Table: Bonus

| Column Name | Type |
| --- | --- |
| empId | int |
| bonus | int |

empId is the column of unique values for this table.
empId is a foreign key (reference column) to empId from the Employee table.
Each row of this table contains the id of an employee and their respective bonus.

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**.

```
SELECT Employee.name, Bonus.bonus
FROM Employee
LEFT JOIN Bonus
  USING (empId)
WHERE IFNULL(Bonus.bonus, 0) < 1000;
```

Accepted Runtime: 184 ms

Case 1

Input

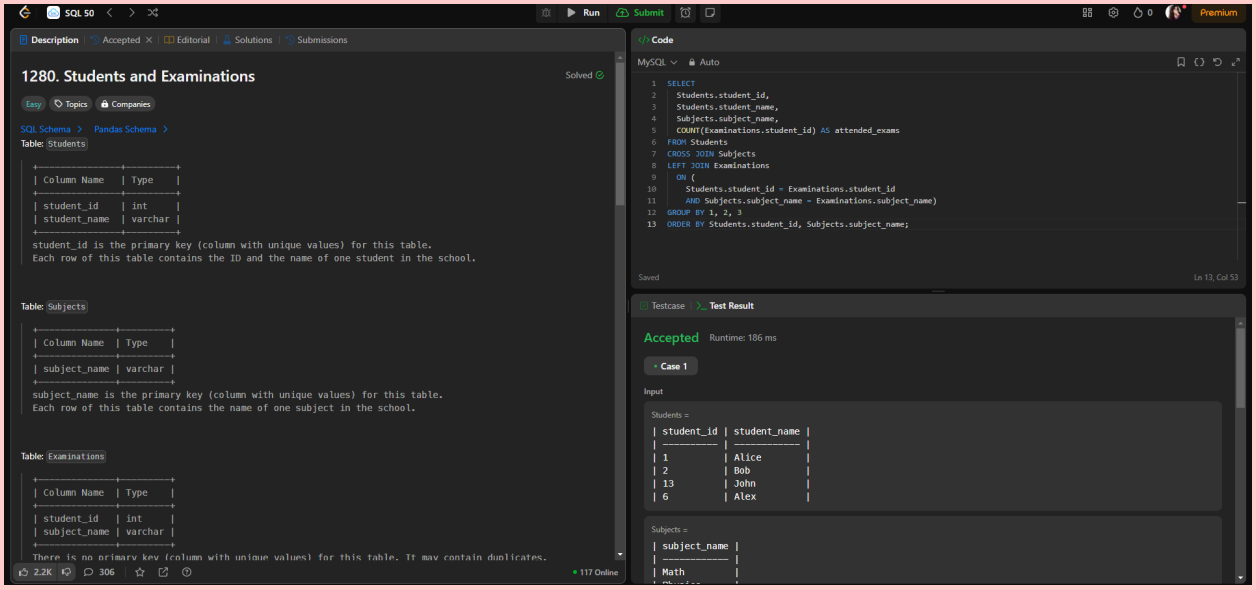
Employee =

| empId | name | supervisor | salary |
| --- | --- | --- | --- |
| 3 | Brad | null | 4000 |
| 1 | John | 3 | 1000 |
| 2 | Dan | 3 | 2000 |
| 4 | Thomas | 3 | 4000 |

Bonus =

| empId | bonus |
| --- | --- |
| 2 | 500 |

## 1280. Students and Examinations



SQL 50 | Run | Submit | Premium

Description | Accepted | Editorial | Solutions | Submissions

### 1280. Students and Examinations

Solved

Easy | Topics | Companies

SQL Schema | Pandas Schema

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.

2.2K | 306 | 117 Online

Code

MySQL | Auto

```sql
SELECT
  Students.student_id,
  Students.student_name,
  Subjects.subject_name,
  COUNT(Examinations.student_id) AS attended_exams
FROM Students
CROSS JOIN Subjects
LEFT JOIN Examinations
  ON (
    Students.student_id = Examinations.student_id
    AND Subjects.subject_name = Examinations.subject_name)
GROUP BY 1, 2, 3
ORDER BY Students.student_id, Subjects.subject_name;
```

Saved | Ln 13, Col 53

Testcase | Test Result

Accepted Runtime: 186 ms

Case 1

Input

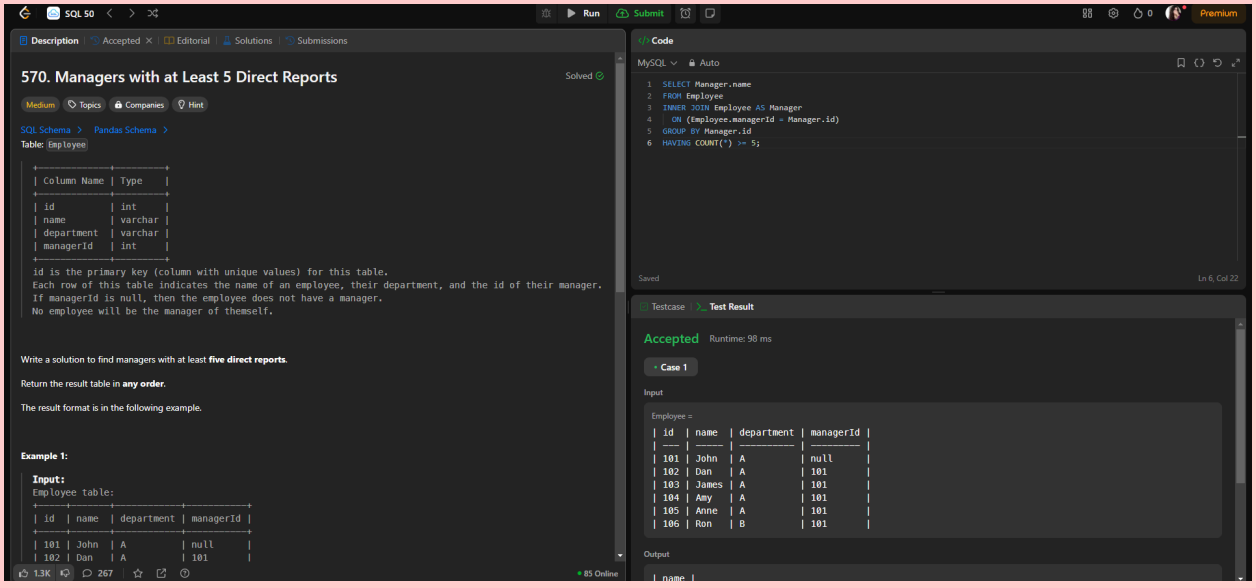
Students =

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

Subjects =

| subject_name |
| --- |
| Math |

## 570. Managers with at Least 5 Direct Reports



SQL 50 | Run | Submit | Premium

Description | Accepted | Editorial | Solutions | Submissions

### 570. Managers with at Least 5 Direct Reports

Solved

Medium | Topics | Companies | Hint

SQL Schema | Pandas Schema

Table: Employee

| Column Name | Type |
| --- | --- |
| id | int |
| name | varchar |
| department | varchar |
| managerId | int |

id is the primary key (column with unique values) for this table.
Each row of this table indicates the name of an employee, their department, and the id of their manager.
If managerId is null, then the employee does not have a manager.
No employee will be the manager of themself.

Write a solution to find managers with at least **five direct reports**.

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

The result format is in the following example.

**Example 1:**

**Input:**
Employee table:

| id | name | department | managerId |
| --- | --- | --- | --- |
| 101 | John | A | null |
| 102 | Dan | A | 101 |

1.3K | 267 | 85 Online

Code

MySQL | Auto

```sql
SELECT Manager.name
FROM Employee
INNER JOIN Employee AS Manager
  ON (Employee.managerId = Manager.id)
GROUP BY Manager.id
HAVING COUNT(*) >= 5;
```

Saved | Ln 6, Col 22

Testcase | Test Result

Accepted Runtime: 98 ms

Case 1

Input

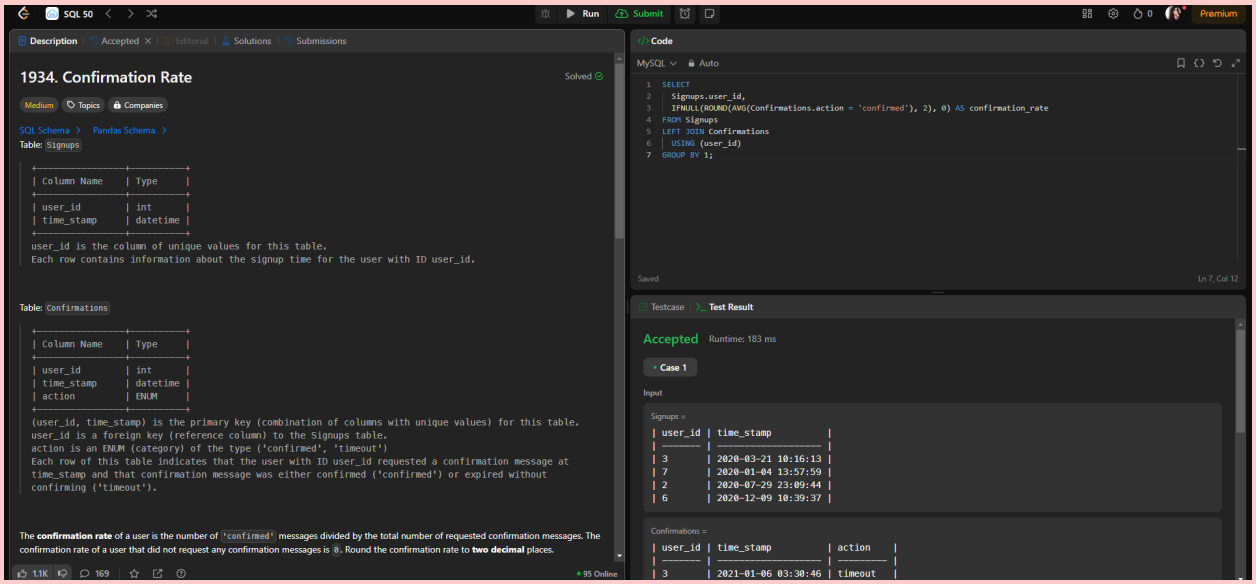
Employee =

| id | name | department | managerId |
| --- | --- | --- | --- |
| 101 | John | A | null |
| 102 | Dan | A | 101 |
| 103 | James | A | 101 |
| 104 | Amy | A | 101 |
| 105 | Anne | A | 101 |
| 106 | Ron | B | 101 |

Output

| name |
| --- |

## 1934. Confirmation Rate




# Question 2

Study Self Join and solve the following question:

Display the employee last name and employee id along with their manager's last name and manager id. Label the columns as Employee_ID, Emp_LN , Manager_ID, and Mgr_LN.

# Query

```
SELECT e.EMPLOYEE_ID AS Employee_ID,
    e.LAST_NAME AS Emp_LN,
    m.EMPLOYEE_ID AS Manager_ID,
    m.LAST_NAME AS Mgr_LN
FROM HR.EMPLOYEES e
LEFT JOIN HR.EMPLOYEES m ON e.MANAGER_ID = m.EMPLOYEE_ID;
```

## Output

| EMPLOYEE_ID | EMP_LN | MANAGER_ID | MGR_LN |
|---|---|---|---|
| 168 | Ozer | 148 | Cambrault |
| 169 | Bloom | 148 | Cambrault |
| 170 | Fox | 148 | Cambrault |
| 171 | Smith | 148 | Cambrault |
| 172 | Bates | 148 | Cambrault |
| 173 | Kumar | 148 | Cambrault |
| 103 | Hunold | 102 | De Haan |
| 162 | Vishney | 147 | Errazuriz |
| 163 | Greene | 147 | Errazuriz |
| 164 | Marvins | 147 | Errazuriz |
| 165 | Lee | 147 | Errazuriz |
| 166 | Ande | 147 | Errazuriz |
| 167 | Banda | 147 | Errazuriz |

| | | | |
|---|---|---|---|
| 129 | Bissot | 121 | Fripp |
| 130 | Atkinson | 121 | Fripp |
| 131 | Marlow | 121 | Fripp |
| 132 | Olson | 121 | Fripp |
| 184 | Sarchand | 121 | Fripp |
| 185 | Bull | 121 | Fripp |
| 186 | Dellinger | 121 | Fripp |
| 187 | Cabrio | 121 | Fripp |
| 109 | Faviet | 108 | Greenberg |
| 110 | Chen | 108 | Greenberg |
| 111 | Sciarra | 108 | Greenberg |
| 112 | Urman | 108 | Greenberg |
| 113 | Popp | 108 | Greenberg |
| 202 | Fay | 201 | Hartstein |
| 206 | Gietz | 205 | Higgins |

| | | | |
|---|---|---|---|
| 206 | Gietz | 205 | Higgins |
| 104 | Ernst | 103 | Hunold |
| 105 | Austin | 103 | Hunold |
| 106 | Pataballa | 103 | Hunold |
| 107 | Lorentz | 103 | Hunold |
| 133 | Mallin | 122 | Kaufling |
| 134 | Rogers | 122 | Kaufling |
| 135 | Gee | 122 | Kaufling |
| 136 | Philtanker | 122 | Kaufling |
| 188 | Chung | 122 | Kaufling |
| 189 | Dilly | 122 | Kaufling |
| 190 | Gates | 122 | Kaufling |
| 191 | Perkins | 122 | Kaufling |
| 101 | Kochhar | 100 | King |

| | | | |
|---|---|---|---|
| 190 | Gates | 122 | Kaufling |
| 191 | Perkins | 122 | Kaufling |
| 101 | Kochhar | 100 | King |
| 102 | De Haan | 100 | King |
| 114 | Raphaely | 100 | King |
| 120 | Weiss | 100 | King |
| 121 | Fripp | 100 | King |
| 122 | Kaufling | 100 | King |
| 123 | Vollman | 100 | King |
| 124 | Mourgos | 100 | King |
| 145 | Russell | 100 | King |
| 146 | Partners | 100 | King |

Download CSV

Rows 1 - 50. More rows exist.

## Question 3

Write a query to display the last name, job, department number, and department name for all employees who work in Toronto.

## Query

```
SELECT e.LAST_NAME,
    e.JOB_ID,
    e.DEPARTMENT_ID,
    d.DEPARTMENT_NAME
FROM HR.EMPLOYEES e
JOIN HR.DEPARTMENTS d ON e.DEPARTMENT_ID = d.DEPARTMENT_ID
JOIN HR.LOCATIONS l ON d.LOCATION_ID = l.LOCATION_ID
WHERE l.CITY = 'Toronto';
```

## Output

| LAST_NAME | JOB_ID | DEPARTMENT_ID | DEPARTMENT_NAME |
|---|---|---|---|
| Hartstein | MK_MAN | 20 | Marketing |
| Fay | MK_REP | 20 | Marketing |

Download CSV

2 rows selected.

## Question 4

Display the employee full name and department name for all employees who have an

i (lowercase) in their full names.

## Query

```
SELECT e.FIRST_NAME || ' ' || e.LAST_NAME AS Full_Name,
    d.DEPARTMENT_NAME
FROM HR.EMPLOYEES e
JOIN HR.DEPARTMENTS d ON e.DEPARTMENT_ID = d.DEPARTMENT_ID
WHERE LOWER(e.FIRST_NAME || ' ' || e.LAST_NAME) LIKE '%i%';
```

# Output

| FULL_NAME | DEPARTMENT_NAME |
|---|---|
| Jennifer Whalen | Administration |
| Michael Hartstein | Marketing |
| Guy Himuro | Purchasing |
| Sigal Tobias | Purchasing |
| Shelli Baida | Purchasing |
| Susan Mavris | Human Resources |
| Jennifer Dilly | Shipping |
| Julia Dellinger | Shipping |
| Curtis Davies | Shipping |
| Anthony Cabrio | Shipping |
| Alexis Bull | Shipping |
| Laura Bissot | Shipping |
| Mozhe Atkinson | Shipping |
| Renske Ladwig | Shipping |
| Matthew Weiss | Shipping |
| Winston Taylor | Shipping |

| | |
|---|---|
| Martha Sullivan | Shipping |
| Stephen Stiles | Shipping |
| Nandita Sarchand | Shipping |
| Michael Rogers | Shipping |
| Hazel Philtanker | Shipping |
| Randall Perkins | Shipping |
| Julia Nayer | Shipping |
| Kevin Mourgos | Shipping |
| Irene Mikkilineni | Shipping |
| Samuel McCain | Shipping |
| Jason Mallin | Shipping |
| Payam Kaufling | Shipping |
| Girard Geoni | Shipping |
| Ki Gee | Shipping |
| Timothy Gates | Shipping |
| Adam Fripp | Shipping |
| Kevin Feeney | Shipping |

| | |
|---|---|
| David Austin | IT |
| Diana Lorentz | IT |
| Lindsey Smith | Sales |
| William Smith | Sales |
| Oliver Tuvault | Sales |
| Jack Livingston | Sales |
| David Lee | Sales |
| Eleni Zlotkey | Sales |
| Sundita Kumar | Sales |
| Janette King | Sales |
| Elizabeth Bates | Sales |
| Mattea Marvins | Sales |
| Louise Doran | Sales |
| Amit Banda | Sales |
| Alberto Errazuriz | Sales |

Download CSV

Rows 1 - 50. More rows exist.

## Question 5

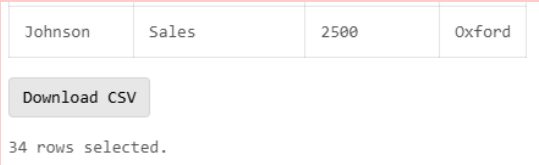
Write a query to display the employee last name, department name, location ID, and city of all employees who earn a commission.

## Query

```
SELECT e.LAST_NAME,
    d.DEPARTMENT_NAME,
    d.LOCATION_ID,
    l.CITY
FROM HR.EMPLOYEES e
JOIN HR.DEPARTMENTS d ON e.DEPARTMENT_ID = d.DEPARTMENT_ID
JOIN HR.LOCATIONS l ON d.LOCATION_ID = l.LOCATION_ID
WHERE e.COMMISSION_PCT IS NOT NULL;
```

## Output

| LAST_NAME | DEPARTMENT_NAME | LOCATION_ID | CITY |
|---|---|---|---|
| Russell | Sales | 2500 | Oxford |
| Partners | Sales | 2500 | Oxford |
| Errazuriz | Sales | 2500 | Oxford |
| Cambrault | Sales | 2500 | Oxford |
| Zlotkey | Sales | 2500 | Oxford |
| Tucker | Sales | 2500 | Oxford |
| Bernstein | Sales | 2500 | Oxford |
| Hall | Sales | 2500 | Oxford |
| Olsen | Sales | 2500 | Oxford |
| Cambrault | Sales | 2500 | Oxford |
| Tuvault | Sales | 2500 | Oxford |
| King | Sales | 2500 | Oxford |
| Sully | Sales | 2500 | Oxford |
| McEwen | Sales | 2500 | Oxford |
| Smith | Sales | 2500 | Oxford |
| Doran | Sales | 2500 | Oxford |
| Sewall | Sales | 2500 | Oxford |
| Vishney | Sales | 2500 | Oxford |
| Greene | Sales | 2500 | Oxford |
| Marvins | Sales | 2500 | Oxford |
| Lee | Sales | 2500 | Oxford |
| Ande | Sales | 2500 | Oxford |
| Banda | Sales | 2500 | Oxford |
| Ozer | Sales | 2500 | Oxford |
| Bloom | Sales | 2500 | Oxford |
| Fox | Sales | 2500 | Oxford |
| Smith | Sales | 2500 | Oxford |
| Bates | Sales | 2500 | Oxford |
| Kumar | Sales | 2500 | Oxford |
| Abel | Sales | 2500 | Oxford |
| Hutton | Sales | 2500 | Oxford |
| Taylor | Sales | 2500 | Oxford |
| Livingston | Sales | 2500 | Oxford |



| | | | |
|---|---|---|---|
| Johnson | Sales | 2500 | Oxford |

Download CSV

34 rows selected.

## Question 6

Display the names and hire dates for all employees who were hired before their managers, along with their manager's names and hire dates. Label the columns Employee, Emp_Hired, Manager, and Mgr_Hired. Order the results in descending by the employee number.

## Query

```
SELECT e.LAST_NAME AS Employee,
    e.HIRE_DATE AS Emp_Hired,
    m.LAST_NAME AS Manager,
    m.HIRE_DATE AS Mgr_Hired
FROM HR.EMPLOYEES e
JOIN HR.EMPLOYEES m ON e.MANAGER_ID = m.EMPLOYEE_ID
WHERE e.HIRE_DATE < m.HIRE_DATE
ORDER BY e.EMPLOYEE_ID DESC;
```

# Output

| EMPLOYEE | EMP_HIRED | MANAGER | MGR_HIRED |
|---|---|---|---|
| Higgins | 07-JUN-02 | Kochhar | 21-SEP-05 |
| Baer | 07-JUN-02 | Kochhar | 21-SEP-05 |
| Mavris | 07-JUN-02 | Kochhar | 21-SEP-05 |
| Whalen | 17-SEP-03 | Kochhar | 21-SEP-05 |
| OConnell | 21-JUN-07 | Mourgos | 16-NOV-07 |
| Feeney | 23-MAY-06 | Mourgos | 16-NOV-07 |
| Walsh | 24-APR-06 | Mourgos | 16-NOV-07 |
| Everett | 03-MAR-05 | Vollman | 10-OCT-05 |
| Bell | 04-FEB-04 | Vollman | 10-OCT-05 |
| Bull | 20-FEB-05 | Fripp | 10-APR-05 |
| Sarchand | 27-JAN-04 | Fripp | 10-APR-05 |
| Johnson | 04-JAN-08 | Zlotkey | 29-JAN-08 |
| Grant | 24-MAY-07 | Zlotkey | 29-JAN-08 |
| Livingston | 23-APR-06 | Zlotkey | 29-JAN-08 |
| Taylor | 24-MAR-06 | Zlotkey | 29-JAN-08 |
| Hutton | 19-MAR-05 | Zlotkey | 29-JAN-08 |
| Abel | 11-MAY-04 | Zlotkey | 29-JAN-08 |
| Bates | 24-MAR-07 | Cambrault | 15-OCT-07 |
| Smith | 23-FEB-07 | Cambrault | 15-OCT-07 |
| Fox | 24-JAN-06 | Cambrault | 15-OCT-07 |
| Bloom | 23-MAR-06 | Cambrault | 15-OCT-07 |
| Ozer | 11-MAR-05 | Cambrault | 15-OCT-07 |
| McEwen | 01-AUG-04 | Partners | 05-JAN-05 |
| Sully | 04-MAR-04 | Partners | 05-JAN-05 |
| King | 30-JAN-04 | Partners | 05-JAN-05 |
| Vargas | 09-JUL-06 | Mourgos | 16-NOV-07 |
| Matos | 15-MAR-06 | Mourgos | 16-NOV-07 |
| Davies | 29-JAN-05 | Mourgos | 16-NOV-07 |
| Rajs | 17-OCT-03 | Mourgos | 16-NOV-07 |
| Ladwig | 14-JUL-03 | Vollman | 10-OCT-05 |
| Marlow | 16-FEB-05 | Fripp | 10-APR-05 |
| Kaufling | 01-MAY-03 | King | 17-JUN-03 |
| Raphaely | 07-DEC-02 | King | 17-JUN-03 |
| Faviet | 16-AUG-02 | Greenberg | 17-AUG-02 |
| Greenberg | 17-AUG-02 | Kochhar | 21-SEP-05 |
| Austin | 25-JUN-05 | Hunold | 03-JAN-06 |
| De Haan | 13-JAN-01 | King | 17-JUN-03 |

Download CSV

37 rows selected.