**[1378. Replace Employee ID With The Unique Identifier](https://leetcode.com/problems/replace-employee-id-with-the-unique-identifier/)**

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.

**Example 1:**

**Input:**

Employees table:

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

| id | name |

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

| 1 | Alice |

| 7 | Bob |

| 11 | Meir |

| 90 | Winston |

| 3 | Jonathan |

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

EmployeeUNI table:

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

| id | unique\_id |

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

| 3 | 1 |

| 11 | 2 |

| 90 | 3 |

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

**Output:**

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

| unique\_id | name |

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

| null | Alice |

| null | Bob |

| 2 | Meir |

| 3 | Winston |

| 1 | Jonathan |

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

**Explanation:**

Alice and Bob do not have a unique ID, We will show null instead.

The unique ID of Meir is 2.

The unique ID of Winston is 3.

The unique ID of Jonathan is 1.

SELECT IFNULL(eu.unique\_id, NULL) AS unique\_id, e.name

FROM Employees e

LEFT JOIN EmployeeUNI eu ON e.id = eu.id;

[**1757. Recyclable and Low Fat Products**](https://leetcode.com/problems/recyclable-and-low-fat-products/)

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.

Write a solution to find the ids of products that are both low fat and recyclable.

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

The result format is in the following example.

**Example 1:**

**Input:**

Products table:

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

| 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 |

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

**Explanation:** Only products 1 and 3 are both low fat and recyclable.

select product\_id from Products p

where p.low\_fats="Y" and p.recyclable ="Y"

[**584. Find Customer Referee**](https://leetcode.com/problems/find-customer-referee/)

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.

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.

**Example 1:**

**Input:**

Customer table:

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

| 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 |

+------+

SELECT

name

FROM

Customer

WHERE referee\_id is null or referee\_id!=2

[**595. Big Countries**](https://leetcode.com/problems/big-countries/)

Table: World

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

| Column Name | Type |

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

| name | varchar |

| continent | varchar |

| area | int |

| population | int |

| gdp | bigint |

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

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

Each row of this table gives information about the name of a country, the continent to which it belongs, its area, the population, and its GDP value.

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**.

The result format is in the following example.

**Example 1:**

**Input:**

World table:

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

| name | continent | area | population | gdp |

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

| Afghanistan | Asia | 652230 | 25500100 | 20343000000 |

| Albania | Europe | 28748 | 2831741 | 12960000000 |

| Algeria | Africa | 2381741 | 37100000 | 188681000000 |

| Andorra | Europe | 468 | 78115 | 3712000000 |

| Angola | Africa | 1246700 | 20609294 | 100990000000 |

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

**Output:**

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

| name | population | area |

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

| Afghanistan | 25500100 | 652230 |

| Algeria | 37100000 | 2381741 |

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

# Write your MySQL query statement below

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

[**1148. Article Views I**](https://leetcode.com/problems/article-views-i/)

Table: Views

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

| Column Name | Type |

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

| article\_id | int |

| author\_id | int |

| viewer\_id | int |

| view\_date | date |

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

There is no primary key (column with unique values) for this table, the table may have duplicate rows.

Each row of this table indicates that some viewer viewed an article (written by some author) on some date.

Note that equal author\_id and viewer\_id indicate the same person.

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.

The result format is in the following example.

**Example 1:**

**Input:**

Views table:

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

| article\_id | author\_id | viewer\_id | view\_date |

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

| 1 | 3 | 5 | 2019-08-01 |

| 1 | 3 | 6 | 2019-08-02 |

| 2 | 7 | 7 | 2019-08-01 |

| 2 | 7 | 6 | 2019-08-02 |

| 4 | 7 | 1 | 2019-07-22 |

| 3 | 4 | 4 | 2019-07-21 |

| 3 | 4 | 4 | 2019-07-21 |

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

**Output:**

+------+

| id |

+------+

| 4 |

| 7 |

+------+

# Write your MySQL query statement below

select distinct author\_id as id from Views where author\_id=viewer\_id order by id;

[**1683. Invalid Tweets**](https://leetcode.com/problems/invalid-tweets/)

Table: Tweets

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

| Column Name | Type |

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

| tweet\_id | int |

| content | varchar |

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

tweet\_id is the primary key (column with unique values) for this table.

This table contains all the tweets in a social media app.

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**.

The result format is in the following example.

**Example 1:**

**Input:**

Tweets table:

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

| tweet\_id | content |

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

| 1 | Vote for Biden |

| 2 | Let us make America great again! |

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

**Output:**

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

| tweet\_id |

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

| 2 |

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

**Explanation:**

Tweet 1 has length = 14. It is a valid tweet.

Tweet 2 has length = 32. It is an invalid tweet.

# Write your MySQL query statement below

select tweet\_id from Tweets where length(content)>15

[**1068. Product Sales Analysis I**](https://leetcode.com/problems/product-sales-analysis-i/)

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**.

The result format is in the following example.

**Example 1:**

**Input:**

Sales table:

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

| sale\_id | product\_id | year | quantity | price |

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

| 1 | 100 | 2008 | 10 | 5000 |

| 2 | 100 | 2009 | 12 | 5000 |

| 7 | 200 | 2011 | 15 | 9000 |

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

Product table:

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

| product\_id | product\_name |

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

| 100 | Nokia |

| 200 | Apple |

| 300 | Samsung |

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

**Output:**

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

| product\_name | year | price |

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

| Nokia | 2008 | 5000 |

| Nokia | 2009 | 5000 |

| Apple | 2011 | 9000 |

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

**Explanation:**

From sale\_id = 1, we can conclude that Nokia was sold for 5000 in the year 2008.

From sale\_id = 2, we can conclude that Nokia was sold for 5000 in the year 2009.

From sale\_id = 7, we can conclude that Apple was sold for 9000 in the year 2011.

# Write your MySQL query statement below

select

p.product\_name,s.year,s.price

from Sales s inner join product p on s.product\_id =p.product\_id;

[**1581. Customer Who Visited but Did Not Make Any Transactions**](https://leetcode.com/problems/customer-who-visited-but-did-not-make-any-transactions/)

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.

**Example 1:**

**Input:**

Visits

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

| visit\_id | customer\_id |

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

| 1 | 23 |

| 2 | 9 |

| 4 | 30 |

| 5 | 54 |

| 6 | 96 |

| 7 | 54 |

| 8 | 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 |

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

| 54 | 2 |

| 30 | 1 |

| 96 | 1 |

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

# Write your MySQL query statement below

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.visit\_id IS NULL

GROUP BY v.customer\_id;

[**197. Rising Temperature**](https://leetcode.com/problems/rising-temperature/)

Table: Weather

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

| Column Name | Type |

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

| id | int |

| recordDate | date |

| temperature | int |

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

id is the column with unique values for this table.

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 |

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

**Output:**

+----+

| id |

+----+

| 2 |

| 4 |

# Write your MySQL query statement below

select w2.id from Weather w2, Weather w1 where w2.temperature > w1. temperature and datediff(w2.recordDate, w1.recordDate)=1

DATEDIFF(interval, date1, date2)

Date1>date2

SELECT DATEDIFF(month, '2017/08/25', '2011/08/25') AS DateDiff;

SELECT DATEDIFF(hour, '2017/08/25 07:00', '2017/08/25 12:45') AS DateDiff;

[**1661. Average Time of Process per Machine**](https://leetcode.com/problems/average-time-of-process-per-machine/)

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.

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**.

The result format is in the following example.

**Example 1:**

**Input:**

Activity table:

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

| machine\_id | process\_id | activity\_type | timestamp |

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

| 0 | 0 | start | 0.712 |

| 0 | 0 | end | 1.520 |

| 0 | 1 | start | 3.140 |

| 0 | 1 | end | 4.120 |

| 1 | 0 | start | 0.550 |

| 1 | 0 | end | 1.550 |

| 1 | 1 | start | 0.430 |

| 1 | 1 | end | 1.420 |

| 2 | 0 | start | 4.100 |

| 2 | 0 | end | 4.512 |

| 2 | 1 | start | 2.500 |

| 2 | 1 | end | 5.000 |

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

**Output:**

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

| machine\_id | processing\_time |

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

| 0 | 0.894 |

| 1 | 0.995 |

| 2 | 1.456 |

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

**Explanation:**

There are 3 machines running 2 processes each.

Machine 0's average time is ((1.520 - 0.712) + (4.120 - 3.140)) / 2 = 0.894

Machine 1's average time is ((1.550 - 0.550) + (1.420 - 0.430)) / 2 = 0.995

Machine 2's average time is ((4.512 - 4.100) + (5.000 - 2.500)) / 2 = 1.456

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

select a.machine\_id,

round(( select avg(a1.timestamp) from activity a1 where a1.activity\_type = 'end' and a1.machine\_id= a.machine\_id)-

(select avg(a1.timestamp) from activity a1 where a1.activity\_type = 'start' and a1.machine\_id= a.machine\_id),3)as processing\_time

from Activity a, Activity a1

group by a.machine\_id

# Write your MySQL query statement below

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