

My LeetCode Submissions - @Suraj-01

[Download PDF](#)[Follow @TheShubham99 on GitHub](#)[Star on GitHub](#)[View Source Code](#)

[577 Employee Bonus \(link\)](#)

Description

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

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 who satisfies either of the following:

- The employee has a bonus **less than** 1000.
- The employee did not get any bonus.

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

The result format is in the following example.

Example 1:

Input:

Employee table:

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

Bonus table:

empId	bonus
2	500
4	2000

Output:

name	bonus
Brad	null
John	null
Dan	500

(scroll down for solution)

Solution

Language: mysql

Status: Accepted

```
# Write your MySQL query statement below
SELECT
    e.name,
    b.bonus
FROM
    Employee e
LEFT JOIN
    Bonus b
    ON e.empId = b.empId
WHERE
    b.bonus < 1000
    OR b.bonus IS NULL;
```

[1801 Average Time of Process per Machine \(link\)](#)

Description

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). 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 process ends 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**.

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 $((4.512 - 4.100) + (5.000 - 2.500)) / 2 = 1.456$

(scroll down for solution)

Solution

Language: mysql

Status: Accepted

```
# Write your MySQL query statement below
Select e.machine_id, Round(Avg(t.timestamp - e.timestamp),3) as processing_time from Activity
join activity t
  on e.machine_id = t.machine_id
  and e.process_id = t.process_id
  and e.activity_type = 'start'
  and t.activity_type = 'end'
group by e.machine_id
order by e.machine_id;
```



[197 Rising Temperature \(link\)](#)

Description

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

Output:

id
2
4

Explanation:

In 2015-01-02, the temperature was higher than the previous day (10 -> 25).
In 2015-01-04, the temperature was higher than the previous day (20 -> 30).

(scroll down for solution)

Solution

Language: mysql

Status: Accepted

```
# Write your MySQL query statement below
SELECT w1.id
FROM Weather w1
INNER JOIN Weather w2
ON DATEDIFF(w1.recordDate, w2.recordDate) = 1
WHERE w1.temperature > w2.temperature;
```

1724 Customer Who Visited but Did Not Make Any Transactions (link)

Description

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

Explanation:

Customer with id = 23 visited the mall once and made one transaction during the visit with amount = 910.
Customer with id = 9 visited the mall once and made one transaction during the visit with amount = 200.
Customer with id = 30 visited the mall once and did not make any transactions.
Customer with id = 54 visited the mall three times. During 2 visits they did not make any transactions and during one visit they made one transaction with amount = 310.
Customer with id = 96 visited the mall once and did not make any transactions.
As we can see, users with IDs 30 and 96 visited the mall one time without making any transactions.

(scroll down for solution)



Solution

Language: mysql

Status: Accepted

```
# Write your MySQL query statement below
select v.customer_id, count(*) count_no_trans from visits v
left join Transactions T on v.visit_id = T.visit_id
where transaction_id is null
group by v.customer_id
order by v.customer_id;
```

1153 Product Sales Analysis I (link)

Description

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.

(scroll down for solution)

Solution

Language: mysql

Status: Accepted

```
# Write your MySQL query statement below
select s.year , p.product_name, s.price  from sales s
left join product p on s.product_Id = p.product_id;
```

1509 Replace Employee ID With The Unique Identifier (link)

Description

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.

(scroll down for solution)

Solution

Language: mysql

Status: Accepted

```
# Write your MySQL query statement below
select e.name, eu.unique_id
from Employees e
left join EmployeeUNI eu on e.id = eu.id
order by unique_id asc;
```

[1827 Invalid Tweets \(link\)](#)

Description

Table: Tweets

Column Name	Type
tweet_id	int
content	varchar

tweet_id is the primary key (column with unique values) for this table.
content consists of alphanumeric characters, '!', or ' ' and no other special characters.
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	Let us Code
2	More than fifteen chars are here!

Output:

tweet_id
2

Explanation:

Tweet 1 has length = 11. It is a valid tweet.

Tweet 2 has length = 33. It is an invalid tweet.

(scroll down for solution)

Solution

Language: mysql

Status: Accepted

Write your MySQL query statement below

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

[1258 Article Views I \(link\)](#)

Description

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 a specific 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

(scroll down for solution)

Solution

Language: mysql

Status: Accepted

```
# Write your MySQL query statement below
SELECT DISTINCT author_id AS id
FROM Views
WHERE author_id = viewer_id
ORDER BY id ASC;
```

[1182 Game Play Analysis IV \(link\)](#)

Description

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)

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 number of total players.

The result format is in the following example.

Example 1:

Input:

Activity table:

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

Explanation:

Only the player with id 1 logged back in after the first day he had logged in so the answer is 1/3 = 0.33

(scroll down for solution)

Solution

Language: mysql

Status: Accepted

```
# Write your MySQL query statement below
SELECT
    ROUND(
        COUNT(DISTINCT a.player_id) /
        (SELECT COUNT(DISTINCT player_id) FROM Activity),
        2
    ) AS fraction
FROM Activity a
JOIN (
    SELECT player_id, MIN(event_date) AS first_login
    FROM Activity
    GROUP BY player_id
) f
ON a.player_id = f.player_id
AND a.event_date = DATE_ADD(f.first_login, INTERVAL 1 DAY);
```

595 Big Countries (link)

Description

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, and other country statistics.

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

(scroll down for solution)

Solution

Language: mysql

Status: Accepted

```
# Write your MySQL query statement below
SELECT name, population, area
FROM World
WHERE area >= 3000000
      OR population >= 25000000;
```

[584 Find Customer Referee \(link\)](#)

Description

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 they referred.



Find the names of the customer that are either:

1. **referred by** any customer with id != 2.
2. **not referred by** any customer.

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

(scroll down for solution)

Solution

Language: mysql

Status: Accepted

```
# Write your MySQL query statement below
SELECT name
FROM Customer
WHERE referee_id <> 2
       OR referee_id IS NULL;
```

[1908 Recyclable and Low Fat Products \(link\)](#)

Description

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

recyclable is an ENUM (category) of types ('Y', 'N') where 'Y' means this product is recy

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.

(scroll down for solution)

Solution

Language: mysql

Status: Accepted

```
# Write your MySQL query statement below
select product_id from products
where low_fats = 'Y'
      AND recyclable = 'Y';
```