

# My LeetCode Submissions - @Suraj-01

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

## 596 Classes With at Least 5 Students ([link](#))

### Description

Table: Courses

Column Name	Type
student	varchar
class	varchar

(student, class) is the primary key (combination of columns with unique values) for this table. Each row of this table indicates the name of a student and the class in which they are en-



Write a solution to find all the classes that have **at least five students**.

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

The result format is in the following example.

### Example 1:

**Input:**

Courses table:

student	class
A	Math
B	English
C	Math
D	Biology
E	Math
F	Computer
G	Math
H	Math
I	Math

```
+-----+-----+
```

**Output:**

```
+-----+
```

```
| class |
```

```
+-----+
```

```
| Math |
```

```
+-----+
```

**Explanation:**

- Math has 6 students, so we include it.
- English has 1 student, so we do not include it.
- Biology has 1 student, so we do not include it.
- Computer has 1 student, so we do not include it.

(scroll down for solution)

# Solution

Language: mysql

Status: Accepted

```
# Write your MySQL query statement below
select class from Courses
group by class
having count(student) >= 5;
```

# 1155 Product Sales Analysis III (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.  
 Each row records a sale of a product in a given year.  
 A product may have multiple sales entries in the same year.  
 Note that the per-unit price.

Write a solution to find all sales that occurred in the **first year** each product was sold.

- For each `product_id`, identify the earliest year it appears in the `Sales` table.
- Return **all** sales entries for that product in that year.

Return a table with the following columns: **product\_id**, **first\_year**, **quantity**, and **price**.  
 Return the result in any order.

### 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

**Output:**

product_id	first_year	quantity	price
100	2008	10	5000
200	2011	15	9000

(scroll down for solution)

# Solution

Language: mysql

Status: Accepted

```
# Write your MySQL query statement below
SELECT
    s.product_id,
    s.year AS first_year,
    s.quantity,
    s.price
FROM Sales s
WHERE (s.product_id, s.year) IN (
    SELECT product_id, MIN(year)
    FROM Sales
    GROUP BY product_id
);
```

# 1245 User Activity for the Past 30 Days I (link)

## Description

Table: Activity

Column Name	Type
user_id	int
session_id	int
activity_date	date
activity_type	enum

This table may have duplicate rows.

The activity\_type column is an ENUM (category) of type ('open\_session', 'end\_session', 'scroll\_down', 'send\_message').

The table shows the user activities for a social media website.

Note that each session belongs to exactly one user.



Write a solution to find the daily active user count for a period of 30 days ending 2019-07-27 inclusively. A user was active on someday if they made at least one activity on that day.

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

The result format is in the following example.

Note: **Any** activity from ('open\_session', 'end\_session', 'scroll\_down', 'send\_message') will be considered valid activity for a user to be considered active on a day.

## Example 1:

**Input:**

Activity table:

user_id	session_id	activity_date	activity_type
1	1	2019-07-20	open_session
1	1	2019-07-20	scroll_down
1	1	2019-07-20	end_session
2	4	2019-07-20	open_session
2	4	2019-07-21	send_message
2	4	2019-07-21	end_session
3	2	2019-07-21	open_session
3	2	2019-07-21	send_message
3	2	2019-07-21	end_session
4	3	2019-06-25	open_session
4	3	2019-06-25	end_session

**Output:**

--

day	active_users
2019-07-20	2
2019-07-21	2

**Explanation:** Note that we do not care about days with zero active users.

(scroll down for solution)

# Solution

Language: mysql

Status: Accepted

```
# Write your MySQL query statement below
SELECT
    activity_date AS day,
    COUNT(DISTINCT user_id) AS active_users
FROM Activity
WHERE activity_date BETWEEN '2019-06-28' AND '2019-07-27'
GROUP BY activity_date
ORDER BY day;
```

# 2495 Number of Unique Subjects Taught by Each Teacher ([link](#))

## Description

Table: Teacher

Column Name	Type
teacher_id	int
subject_id	int
dept_id	int

(subject\_id, dept\_id) is the primary key (combinations of columns with unique values) of the table.

Each row in this table indicates that the teacher with teacher\_id teaches the subject sub

Write a solution to calculate the number of unique subjects each teacher teaches in the university.

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

The result format is shown in the following example.

### Example 1:

**Input:**

Teacher table:

teacher_id	subject_id	dept_id
1	2	3
1	2	4
1	3	3
2	1	1
2	2	1
2	3	1
2	4	1

**Output:**

teacher_id	cnt
1	2
2	4

**Explanation:**

Teacher 1:

- They teach subject 2 in departments 3 and 4.
- They teach subject 3 in department 3.

Teacher 2:

- They teach subject 1 in department 1.
- They teach subject 2 in department 1.
- They teach subject 3 in department 1.
- They teach subject 4 in department 1.

(scroll down for solution)

# Solution

Language: mysql

Status: Accepted

```
# Write your MySQL query statement below
SELECT
    teacher_id,
    COUNT(DISTINCT subject_id) AS cnt
FROM Teacher
GROUP BY teacher_id
ORDER BY teacher_id;
```

## 1292 Immediate Food Delivery II (link)

### Description

Table: Delivery

Column Name	Type
delivery_id	int
customer_id	int
order_date	date
customer_pref_delivery_date	date

delivery\_id is the column of unique values of this table.

The table holds information about food delivery to customers that make orders at some day.

If the customer's preferred delivery date is the same as the order date, then the order is called **immediate**; otherwise, it is called **scheduled**.

The **first order** of a customer is the order with the earliest order date that the customer made. It is guaranteed that a customer has precisely one first order.

Write a solution to find the percentage of immediate orders in the first orders of all customers, **rounded to 2 decimal places**.

The result format is in the following example.

### Example 1:

#### Input:

Delivery table:

delivery_id	customer_id	order_date	customer_pref_delivery_date
1	1	2019-08-01	2019-08-02
2	2	2019-08-02	2019-08-02
3	1	2019-08-11	2019-08-12
4	3	2019-08-24	2019-08-24
5	3	2019-08-21	2019-08-22
6	2	2019-08-11	2019-08-13
7	4	2019-08-09	2019-08-09

#### Output:

immediate_percentage
50.00

#### Explanation:

The customer id 1 has a first order with delivery id 1 and it is scheduled.  
The customer id 2 has a first order with delivery id 2 and it is immediate.  
The customer id 3 has a first order with delivery id 5 and it is scheduled.  
The customer id 4 has a first order with delivery id 7 and it is immediate.  
Hence, half the customers have immediate first orders.

(scroll down for solution)

# Solution

Language: mysql

Status: Accepted

```
# Write your MySQL query statement below
SELECT
    ROUND(
        SUM(CASE WHEN order_date = customer_pref_delivery_date THEN 1 ELSE 0 END) * 100.0
        / COUNT(*)
    ) AS immediate_percentage
FROM Delivery
WHERE (customer_id, order_date) IN (
    SELECT customer_id, MIN(order_date)
    FROM Delivery
    GROUP BY customer_id
);
```



# 1317 Monthly Transactions I (link)

## Description

Table: Transactions

Column Name	Type
id	int
country	varchar
state	enum
amount	int
trans_date	date

id is the primary key of this table.

The table has information about incoming transactions.

The state column is an enum of type ["approved", "declined"].

Write an SQL query to find for each month and country, the number of transactions and their total amount, the number of approved transactions and their total amount.

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

The query result format is in the following example.

### Example 1:

#### Input:

Transactions table:

id	country	state	amount	trans_date
121	US	approved	1000	2018-12-18
122	US	declined	2000	2018-12-19
123	US	approved	2000	2019-01-01
124	DE	approved	2000	2019-01-07

#### Output:

month	country	trans_count	approved_count	trans_total_amount	approved_total
2018-12	US	2	1	3000	1000
2019-01	US	1	1	2000	2000
2019-01	DE	1	1	2000	2000

(scroll down for solution)

# Solution

Language: mysql

Status: Accepted

```
# Write your MySQL query statement below
select
    date_format(trans_date, '%Y-%m') as month,
    country,
    count(*) as trans_count,
    sum(case when state = "approved" then 1 else 0 end) as approved_count,
    sum(amount) as trans_total_amount,
    sum(case when state = "approved" then amount else 0 end) as approved_total_amount
from Transactions
GROUP BY DATE_FORMAT(trans_date, '%Y-%m'), country
ORDER BY month, country;
```

## 1338 Queries Quality and Percentage ([link](#))

### Description

Table: Queries

Column Name	Type
query_name	varchar
result	varchar
position	int
rating	int

This table may have duplicate rows.

This table contains information collected from some queries on a database.

The position column has a value from 1 to 500.

The rating column has a value from 1 to 5. Query with rating less than 3 is a poor query

We define query quality as:

The average of the ratio between query rating and its position.

We also define poor query percentage as:

The percentage of all queries with rating less than 3.

Write a solution to find each `query_name`, the `quality` and `poor_query_percentage`.

Both `quality` and `poor_query_percentage` should be **rounded to 2 decimal places**.

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

The result format is in the following example.

### Example 1:

**Input:**

Queries table:

query_name	result	position	rating
Dog	Golden Retriever	1	5
Dog	German Shepherd	2	5
Dog	Mule	200	1
Cat	Shirazi	5	2
Cat	Siamese	3	3
Cat	Sphynx	7	4

**Output:**

query_name	quality	poor_query_percentage
Dog	2.50	33.33
Cat	0.66	33.33

**Explanation:**

Dog queries quality is  $((5 / 1) + (5 / 2) + (1 / 200)) / 3 = 2.50$

Dog queries poor\_query\_percentage is  $(1 / 3) * 100 = 33.33$

Cat queries quality equals  $((2 / 5) + (3 / 3) + (4 / 7)) / 3 = 0.66$

Cat queries poor\_query\_percentage is  $(1 / 3) * 100 = 33.33$

(scroll down for solution)

# Solution

Language: mysql

Status: Accepted

```
# Write your MySQL query statement below
SELECT
    query_name,
    ROUND(
        AVG(rating * 1.0 / position),
        2
    ) AS quality,
    ROUND(
        SUM(CASE WHEN rating < 3 THEN 1 ELSE 0 END) * 100.0 / COUNT(*),
        2
    ) AS poor_query_percentage
FROM Queries
WHERE query_name IS NOT NULL
GROUP BY query_name;
```

# 1773 Percentage of Users Attended a Contest [\(link\)](#)

## Description

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.

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

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

The result format is in the following example.

### Example 1:

Input:

Users table:

user_id	user_name
6	Alice
2	Bob
7	Alex

Register table:

contest_id	user_id
215	6
209	2
208	2
210	6
208	6
209	7
209	6
215	7
208	7
210	2
207	2
210	7

Output:

contest_id	percentage
208	100.0
209	100.0
210	100.0
215	66.67
207	33.33

Explanation:

All the users registered in contests 208, 209, and 210. The percentage is 100% and we sort them. Alice and Alex registered in contest 215 and the percentage is  $((2/3) * 100) = 66.67\%$

(scroll down for solution)



# Solution

Language: mysql

Status: Accepted

```
SELECT
    contest_id,
    ROUND(
        COUNT(user_id) * 100.0 / (SELECT COUNT(*) FROM Users),
        2
    ) AS percentage
FROM Register
GROUP BY contest_id
ORDER BY percentage DESC, contest_id ASC;
```

# 1161 Project Employees I ([link](#))

## Description

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 non-negative.

Each row of this table contains information about one employee.

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

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

The query result format is in the following example.

### Example 1:

Input:  
Project table:

project_id	employee_id
1	1
1	2
1	3

2	1	
2	4	
<hr/>		
Employee table:		
employee_id	name	experience_years
1	Khaled	3
2	Ali	2
3	John	1
4	Doe	2
<hr/>		

  

Output:	
project_id	average_years
1	2.00
2	2.50
<hr/>	

(scroll down for solution)

# Solution

Language: mysql

Status: Accepted

```
# 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;
```

# 1390 Average Selling Price (link)

## Description

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 that can identify each row of a database 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 non-empty periods that overlap.

Table: UnitsSold

Column Name	Type
product_id	int
purchase_date	date
units	int

This table may contain duplicate rows.

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

Write a solution to find the average selling price for each product. `average_price` should be **rounded to 2 decimal places**. If a product does not have any sold units, its average selling price is assumed to be 0.

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

The result format is in the following example.

### Example 1:

**Input:**

Prices table:

product_id	start_date	end_date	price
1	2020-01-01	2020-01-31	10

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 table:

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

Explanation:

Average selling price = Total Price of Product / Number of products sold.

Average selling price for product 1 =  $((100 * 5) + (15 * 20)) / 115 = 6.96$

Average selling price for product 2 =  $((200 * 15) + (30 * 30)) / 230 = 16.96$

(scroll down for solution)

# Solution

Language: mysql

Status: Accepted

```
# Write your MySQL query statement below
select
    p.product_id,
    ifnull (Round(Sum(p.price*u.units)/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;
```



## 620 Not Boring Movies ([link](#))

### Description

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]

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

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

The result format is in the following example.

### Example 1:

**Input:**  
 Cinema table:

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

**Explanation:**

We have three movies with odd-numbered IDs: 1, 3, and 5. The movie with ID = 3 is boring

(scroll down for solution)

# Solution

Language: mysql

Status: Accepted

```
# Write your MySQL query statement below
select id, movie, description, rating  from Cinema
where id % 2 = 1
    and description != "boring"
order by rating desc;
```

# 2087 Confirmation Rate (link)

## Description

Table: Signups

Column Name	Type
user_id	int
time_stamp	datetime

user\_id is the column of unique values for this table.

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 user\_id is a foreign key (reference column) to the Signups table.

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

Each row of this table indicates that the user with ID user\_id requested a confirmation r



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**.

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

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

The result format is in the following example.

### Example 1:

**Input:**

Signups table:

user_id	time_stamp
1	2023-10-01 08:00:00

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 table:

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
2	2021-02-28 23:59:59	timeout

Output:

user_id	confirmation_rate
6	0.00
3	0.00
7	1.00
2	0.50

Explanation:

User 6 did not request any confirmation messages. The confirmation rate is 0.

User 3 made 2 requests and both timed out. The confirmation rate is 0.

User 7 made 3 requests and all were confirmed. The confirmation rate is 1.

(scroll down for solution)



# Solution

Language: mysql

Status: Accepted

```
# Write your MySQL query statement below
SELECT
    s.user_id,
    ROUND(
        COALESCE(AVG(CASE WHEN c.action = 'confirmed' THEN 1.0 ELSE 0 END), 0),
        2
    ) AS confirmation_rate
FROM
    Signups s
LEFT JOIN
    Confirmations c ON s.user_id = c.user_id
GROUP BY
    s.user_id;
```

# 570 Managers with at Least 5 Direct Reports [\(link\)](#)

## Description

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 themselves.

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
103	James	A	101
104	Amy	A	101
105	Anne	A	101
106	Ron	B	101

#### Output:

name
John

(scroll down for solution)

# Solution

Language: mysql

Status: Accepted

```
# Write your MySQL query statement below
select e.name from employee e
join (
    select managerid, count(*)as report_count
    from employee
    where managerid is not null
    group by managerid
    having count(*) >= 5
) e2 on e.id = e2.managerid
```

# 1415 Students and Examinations ([link](#))

## Description

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.



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.

The result format is in the following example.

### Example 1:

**Input:**

Students table:

student_id	student_name
1	Alice
2	Bob
13	John
6	Alex

Subjects table:

subject_name
Math
Physics
Programming

Examinations table:

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

**Explanation:**

The result table should contain all students and all subjects.

Alice attended the Math exam 3 times, the Physics exam 2 times, and the Programming exam 1 time.

Bob attended the Math exam 1 time, the Programming exam 1 time, and did not attend the Physics exam.

Alex did not attend any exams.

John attended the Math exam 1 time, the Physics exam 1 time, and the Programming exam 1 time.

(scroll down for solution)

# Solution

Language: mysql

Status: Accepted

```
# Write your MySQL query statement below
SELECT
    s.student_id,
    s.student_name,
    sub.subject_name,
    COUNT(e.subject_name) AS attended_exams
FROM
    Students s
CROSS JOIN
    Subjects sub
LEFT JOIN
    Examinations e
ON s.student_id = e.student_id
    AND sub.subject_name = e.subject_name
GROUP BY
    s.student_id,
    s.student_name,
    sub.subject_name
ORDER BY
    s.student_id,
    sub.subject_name;
```

# 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
1	A	2	1000

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 that can identify each row). 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 stops 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**.

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 e
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 without making any purchases.

Customer with id = 9 visited the mall once and made one transaction during the visit without making any purchases.

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 purchases.

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

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).  
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 = 0.3333333333333333 = 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, its area, its population, and its GDP value.



A country is **big** if:

- it has an area of at least three million (i.e.,  $3000000 \text{ km}^2$ ), 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 who referred them.

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 recyclable

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';
```