

LEETCODE SQL (50) QUESTION & SOLUTION

➤ SELECT-

1) 1757. Recyclable and Low Fat Products

The screenshot shows the LeetCode SQL editor interface. The problem title is "1757. Recyclable and Low Fat Products". The code area contains the following query:

```
1 SELECT PRODUCT_ID
2 FROM PRODUCTS
3 WHERE low_fats = 'Y' && recyclable = 'Y';
```

The "Test Result" section shows the query was accepted with a runtime of 81 ms. The input data is a table named "Products" with columns product_id, low_fats, and recyclable. The output shows the product_id where both conditions are met.

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

PRODUCT_ID
1

2) 584. Find Customer Referee

The screenshot shows the LeetCode SQL editor interface. The problem title is "584. Find Customer Referee". The code area contains the following query:

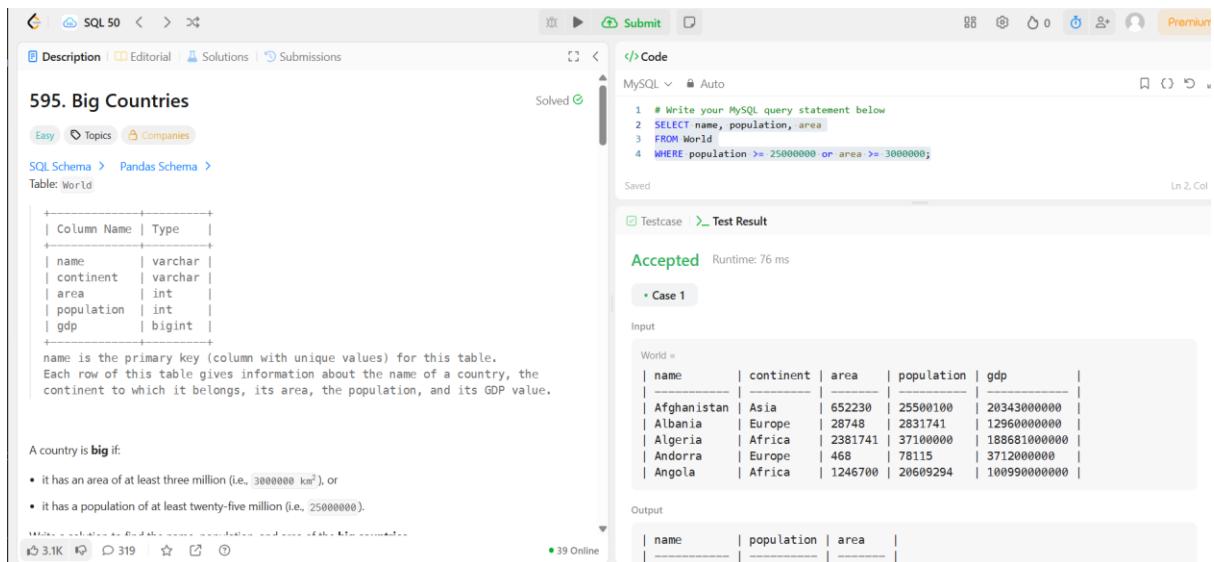
```
1 SELECT name
2 FROM Customer
3 WHERE referee_id != 2 OR referee_id IS NULL;
```

The "Test Result" section shows the query was accepted with a runtime of 75 ms. The input data is a table named "Customer" with columns id, name, and referee_id. The output shows the names of customers who were not referred by customer id 2 or were null.

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

name
Will
Jane
Bill
Zack

3) 595. Big Countries

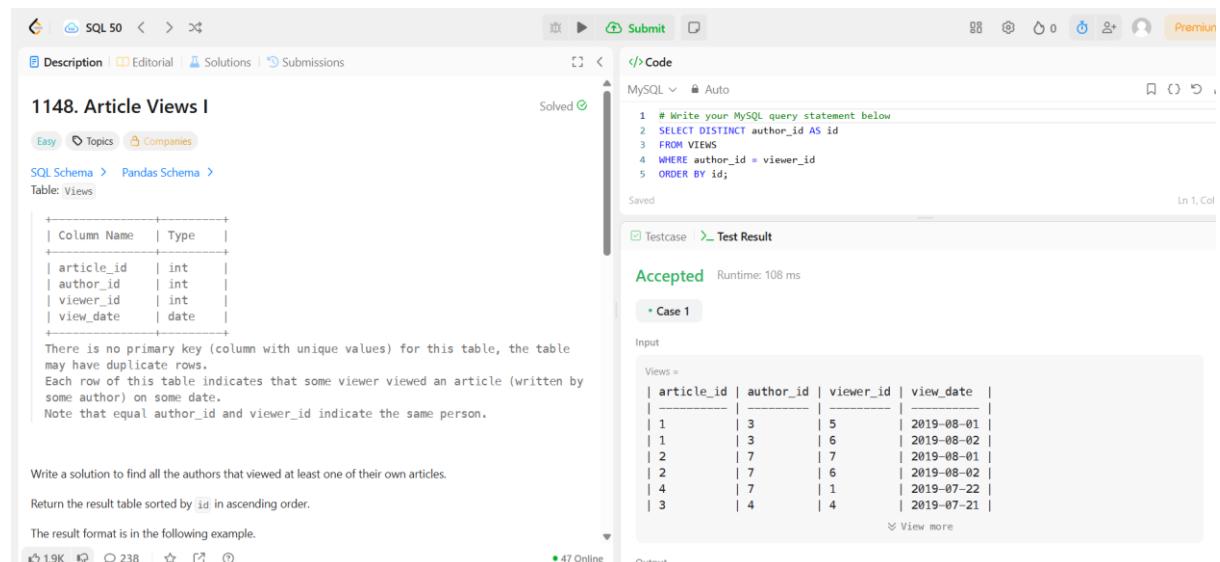


The screenshot shows a LeetCode SQL challenge titled "595. Big Countries". The challenge asks to select the names, populations, and areas of countries where the population is at least 25 million or the area is at least 3 million square kilometers. The table "World" has columns: name (varchar), continent (varchar), area (int), population (int), and gdp (bigint). The primary key is name. The challenge interface includes a code editor with the query, a test result section showing an accepted solution with runtime 76 ms, and an input/output pane.

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

name	continent	area	population	gdp
Afghanistan	Asia	652230	25500100	20343000000
Albania	Europe	28748	2831741	12960000000
Algeria	Africa	2381741	37100000	188681000000
Andorra	Europe	468	7815	3712000000
Angola	Africa	1246700	20609294	100990000000

4) 1148. Article Views I



The screenshot shows a LeetCode SQL challenge titled "1148. Article Views I". The challenge asks to find authors who viewed at least one of their own articles. The table "Views" has columns: article_id (int), author_id (int), viewer_id (int), and view_date (date). There is no primary key. The challenge interface includes a code editor with the query, a test result section showing an accepted solution with runtime 108 ms, and an input/output pane.

```
1 # Write your MySQL query statement below
2 SELECT DISTINCT author_id AS id
3 FROM Views
4 WHERE author_id = viewer_id
5 ORDER BY id;
```

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

5) 1683. Invalid Tweets

1683. Invalid Tweets

MySQL Auto

```

1 # Write your MySQL query statement below
2 SELECT tweet_id
3 FROM TWEETS
4 WHERE CHAR_LENGTH(content) > 15;

```

Saved

Testcase Test Result

Accepted Runtime: 80 ms

Case 1

Input

Tweets =

tweet_id	content
1	Let us Code
2	More than fifteen chars are here!

Output

tweet_id
2

Expected

1.3K 226 52 Online

➤ Basic Join

6) [1378. Replace Employee ID With The Unique Identifier](#)

1378. Replace Employee ID With The Unique Identifier

MySQL Auto

```

1 # Write your MySQL query statement below
2 SELECT E.unique_id AS unique_id, E.name AS name
3 FROM Employees E
4 LEFT JOIN EmployeeUNI Enu
5 ON E.id = Enu.id;

```

Saved

Testcase Test Result

Accepted Runtime: 97 ms

Case 1

Input

Employees =

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

EmployeeUNI =

id	unique_id
----	-----------

1.7K 184 68 Online

7) [1068. Product Sales Analysis I](#)

1068. Product Sales Analysis I

MySQL v Auto

```

1 # Write your MySQL query statement below
2 SELECT P.PRODUCT_NAME, S.YEAR, S.PRICE
3 FROM SALES S
4 LEFT JOIN PRODUCT P
5 ON S.PRODUCT_ID = P.PRODUCT_ID;

```

Saved Ln 2, Col 1

Testcase > Test Result

Accepted Runtime: 93 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

1.2K 146 48 Online

8) 1581. Customer Who Visited but Did Not Make Any Transactions

1581. Customer Who Visited but Did Not Make Any Transactions

MySQL v Auto

```

1 # Write your MySQL query statement below
2 SELECT V.CUSTOMER_ID, COUNT(CUSTOMER_ID)
3 AS COUNT_NO_TRANS
4 FROM VISITS V LEFT JOIN TRANSACTIONS T
5 ON V.VISIT_ID = T.VISIT_ID
6 WHERE T.TRANSACTION_ID IS NULL
7 GROUP BY V.CUSTOMER_ID;

```

Saved Ln 2, Col 1

Testcase > Test Result

Accepted Runtime: 113 ms

Case 1

Input

Visits =

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

2.9K 259 72 Online

9) 197. Rising Temperature

197. Rising Temperature

MySQL v Auto

```

1 # Write your MySQL query statement below
2 SELECT w1.id
3 FROM Weather w1
4 INNER JOIN Weather w2
5 ON DATEDIFF(w1.recordDate, w2.recordDate) = 1
6 AND w1.temperature > w2.temperature;

```

Saved Ln 2, Col 1

Testcase Test Result Accepted Runtime: 71 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

3.7K 487 115 Online

10) 1661. Average Time of Process per Machine

1661. Average Time of Process per Machine

MySQL v Auto

```

2 SELECT a1.machine_id, ROUND(AVG(a2.timestamp-a1.timestamp), 3)
3 AS processing_time
4 FROM Activity a1
5 INNER JOIN Activity a2
6 ON a1.machine_id = a2.machine_id
7 AND a1.process_id = a2.process_id
8 AND a1.activity_type='start'
9 AND a2.activity_type='end'
10 GROUP BY a1.machine_id;

```

Saved Ln 9, Col 1

Testcase Test Result Accepted Runtime: 95 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

2K 340 98 Online

11) 577. Employee Bonus

The screenshot shows a MySQL query editor interface. At the top, there are navigation icons and a "Submit" button. Below the header, there are tabs for "Description", "Editorial", "Solutions", and "Submissions". The main area displays a problem titled "577. Employee Bonus". The problem description states: "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." Below this, there are two tables: "Employee" and "Bonus". The "Employee" table has columns: empId (int), name (varchar), supervisor (int), and salary (int). The "Bonus" table has columns: empId (int) and bonus (int). The code editor contains the following MySQL query:

```
1 # Write your MySQL query statement below
2 SELECT Employee.name,Bonus.bonus
3 FROM Employee
4 LEFT JOIN Bonus
5 ON Employee.empID = Bonus.empID
6 WHERE bonus < 1000 OR Bonus IS NULL;
```

The status bar at the bottom right shows "Accepted Runtime: 129 ms".

12) 1280. Students and Examinations

1280. Students and Examinations

MySQL

```

1 # Write your MySQL query statement below
2 SELECT s.student_id, s.student_name, sub.subject_name , count(e.student_id)
3 FROM students s
4 AS attended_exams
5 CROSS JOIN subjects sub
6 LEFT JOIN examinations e
7 ON s.student_id = e.student_id
8 AND sub.subject_name = e.subject_name
9 GROUP BY 1,2,3
10 ORDER BY 1,2,3

```

Accepted Runtime: 133 ms

Case 1

Input

student_id	student_name
1	Alice
2	Bob
13	John
6	Alex

13) 570. Managers with at Least 5 Direct Reports

570. Managers with at Least 5 Direct Reports

MySQL

```

1 # Write your MySQL query statement below
2 SELECT a.name
3 FROM Employee a
4 JOIN Employee b
5 ON a.id = b.managerId
6 GROUP BY b.managerId
7 HAVING COUNT(*) >= 5

```

Accepted Runtime: 69 ms

Case 1

Input

id	name	department	managerId
101	John	A	null
102	Dan	A	101
103	James	A	101
104	Amy	A	101
105	Anne	A	101

14) 1934. Confirmation Rate

1934. Confirmation Rate

MySQL Query:

```

1 # Write your MySQL query statement below
2 SELECT s.user_id, round(avg(if(c.action="confirmed",1,0)),2)
3 AS confirmation_rate
4 FROM Signups AS s
5 LEFT JOIN Confirmations AS c
6 ON s.user_id=c.user_id
7 GROUP BY user_id;

```

Test Result

Accepted Runtime: 124 ms

Case 1

Input

Signups =

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

Confirmations =

user_id	time_stamp	action
---------	------------	--------

➤ Basic Aggregate Functions

15) 620. Not Boring Movies

620. Not Boring Movies

MySQL Query:

```

1 # Write your MySQL query statement below
2 SELECT *
3 FROM Cinema
4 WHERE MOD(id, 2) = 1
5 AND description <> "boring"
6 ORDER BY rating DESC

```

Test Result

Accepted Runtime: 73 ms

Case 1

Input

Cinema =

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

Output

id	movie	description	rating
----	-------	-------------	--------

16) 1251. Average Selling Price

1251. Average Selling Price

MySQL v Auto

```

2 SELECT p.product_id ,Coalesce(round(sum(p.price * u.units) / sum(u.units),2),0)
3 AS average_price
4 FROM prices p
5 LEFT JOIN unitssold u
6 ON p.product_id = u.product_id
7 AND u.purchase_date BETWEEN p.start_date AND p.end_date
8 GROUP BY p.product_id;

```

Saved

Testcase > Test Result

Accepted Runtime: 121 ms

Case 1

Input

Prices =

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

UnitsSold =

product_id	purchase_date	units
------------	---------------	-------

1.6K 351 100 Online

17) 1075. Project Employees I

1075. Project Employees I

MySQL v Auto

```

1 /* Write your MySQL query statement below
2 SELECT project_id, round(sum(experience_years)/count(project_id), 2) average_years
3 FROM Project P
4 LEFT JOIN Employee E
5 ON P.employee_id = E.employee_id
6 GROUP BY project_id;

```

Saved

Testcase > Test Result

Accepted Runtime: 113 ms

Case 1

Input

Project =

project_id	employee_id
1	1
1	2
1	3
2	1
2	4

Employee =

employee_id	name	experience_years
-------------	------	------------------

828 119 100 Online

18) 1633. Percentage of Users Attended a Contest

1633. Percentage of Users Attended a Contest

MySQL v Auto

```

1 # Write your MySQL query statement below
2 SELECT contest_id, round(count(distinct user_id) * 100 / (select count(user_id) from Users) ,2)
AS percentage
3 FROM Register
4 GROUP BY contest_id
5 ORDER BY percentage DESC, contest_id;

```

Saved Ln 2, Col 1

Testcase Test Result

Accepted Runtime: 108 ms

Case 1

Input

Users =

user_id	user_name
6	Alice
2	Bob
7	Alex

Register =

contest_id	user_id
215	6

955 163 100 Online

19) 1211. Queries Quality and Percentage

1211. Queries Quality and Percentage

MySQL v Auto

```

1 # Write your MySQL query statement below
2 SELECT DISTINCT query_name , round(avg(rating/position) over(partition by query_name) ,2)
AS quality,
3 round(avg(case when rating<3 then 1 else 0 end) over(partition by query_name)*100,2)
AS poor_query_percentage
4 FROM queries
5 WHERE query_name IS NOT NULL

```

Saved Ln 2, Col 1

Testcase Test Result

Accepted Runtime: 70 ms

Case 1

Input

Queries =

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

889 174 100 Online

20) 1193. Monthly Transactions I

1193. Monthly Transactions I

Medium Topics Companies

SQL Schema > Pandas Schema >

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.

1.1K 174 100 Online

MySQL < Auto

```
1 # Write your MySQL query statement below
2 SELECT SUBSTR(trans_date,1,7) AS month, country, COUNT(id) AS trans_count, SUM(CASE WHEN state
3 = 'approved' THEN 1 ELSE 0 END) AS approved_count, SUM(amount) AS trans_total_amount, SUM(CASE
4 WHEN state = 'approved' THEN amount ELSE 0 END) AS approved_total_amount
5 FROM Transactions
6 GROUP BY month, country;
```

Saved Ln 2, Col

Testcase > Test Result

Accepted Runtime: 81 ms

Case 1

Input

Transactions
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_amount
2018-12 US 2 1 3000 1000
2019-01 US 1 1 2000 2000
2019-01 DE 1 1 2000 2000

21) 1174. Immediate Food Delivery II

1174. Immediate Food Delivery II

Medium Topics Companies

SQL Schema > Pandas Schema >

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 date and specify a preferred delivery date (on the same order date or after it).

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.

995 213 100 Online

MySQL < Auto

```
1 # Write your MySQL query statement below
2 SELECT round(avg(order_date = customer_pref_delivery_date)*100, 2)
3 AS immediate_percentage
4 FROM Delivery
5 WHERE (customer_id, order_date) IN (SELECT customer_id, min(order_date)
6 FROM Delivery
7 GROUP BY customer_id)
```

Saved Ln 2, Col

Testcase > Test Result

Accepted Runtime: 89 ms

Case 1

Input

Delivery
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

View more

22) 550. Game Play Analysis IV

550. Game Play Analysis IV

MySQL

```

1 # Write your MySQL query statement below
2 SELECT ROUND(COUNT(DISTINCT player_id) / (SELECT COUNT(DISTINCT player_id) FROM Activity), 2)
   AS fraction
3 FROM Activity
4 WHERE (player_id, DATE_SUB(event_date, INTERVAL 1 DAY))
5 IN (SELECT player_id, MIN(event_date) AS first_login FROM Activity GROUP BY player_id);

```

Accepted Runtime: 74 ms

Case 1

Input

Activity
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

1.2K 250 100 Online

➤ Sorting and Grouping

23) 2356. Number of Unique Subjects Taught by Each Teacher

2356. Number of Unique Subjects Taught by Each Teacher

MySQL

```

1 # Write your MySQL query statement below
2 SELECT teacher_id , count(DISTINCT subject_id)
3 AS cnt
4 FROM teacher
5 GROUP BY teacher_id;

```

Accepted Runtime: 84 ms

Case 1

Input

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

606 72 100 Online

24) 1141. User Activity for the Past 30 Days I

1141. User Activity for the Past 30 Days I

MySQL Schema > Pandas Schema >

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**.

877 Submissions 176 Testcases 100 Online

Code

```

1 # Write your MySQL query statement below
2 SELECT activity_date AS day, COUNT(DISTINCT user_id) AS active_users
3 FROM Activity
4 WHERE (activity_date > "2019-06-27" AND activity_date <= "2019-07-27")
5 GROUP BY activity_date;

```

Testcase **Test Result**

Accepted Runtime: 68 ms

Case 1

Input

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

Output

25) 1070. Product Sales Analysis III

1070. Product Sales Analysis III

MySQL 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 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.

613 Submissions 269 Testcases 100 Online

Code

```

1 # Write your MySQL query statement below
2 WITH CTE AS (
3   SELECT product_id, MIN(year) AS minyear FROM Sales
4   GROUP BY product_id
5 )
6
7 SELECT s.product_id, s.year AS first_year, s.quantity, s.price
8 FROM Sales s
9 INNER JOIN CTE ON cte.product_id = s.product_id AND s.year = cte.minyear;

```

Testcase **Test Result**

Accepted Runtime: 87 ms

Case 1

Input

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

Output

26) 596. Classes With at Least 5 Students

596. Classes With at Least 5 Students

Easy Topics Companies

SQL Schema > Pandas Schema >

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 enrolled.

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.

• 12K • 104 • 100 Online

Code

```
MySQL ▾ Auto
1 # Write your MySQL query statement below
2 SELECT class
3 FROM Courses
4 GROUP BY class
5 HAVING COUNT(student) >= 5;
```

Test Result

Accepted Runtime: 81 ms

Case 1

Input

Courses =

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

Output

View more

27) 1729. Find Followers Count

1729. Find Followers Count

Easy Topics Companies

SQL Schema > Pandas Schema >

Table: Followers

Column Name	Type
user_id	int
follower_id	int

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

This table contains the IDs of a user and a follower in a social media app where the follower follows the user.

Write a solution that will, for each user, return the number of followers.

Return the result table ordered by `user_id` in ascending order.

The result format is in the following example.

• 673 • 62 • 100 Online

Code

```
MySQL ▾ Auto
1 # Write your MySQL query statement below
2 SELECT user_id, count(follower_id) AS followers_count
3 FROM Followers
4 GROUP BY user_id
5 ORDER BY user_id ASC;
```

Test Result

Accepted Runtime: 150 ms

Case 1

Input

Followers =

user_id	follower_id
0	1
1	0
2	0
2	1

Output

user_id	followers_count
0	1

28) 619. Biggest Single Number

619. Biggest Single Number

MySQL ✓ Auto

```

2 SELECT MAX(num) AS num
3 FROM (
4   SELECT num
5   FROM MyNumbers
6   GROUP BY num
7   HAVING COUNT(*) = 1
8 ) AS single_numbers
9 ORDER BY num DESC
10 LIMIT 1;

```

Saved Ln 10, Col 9

Testcase Test Result

Accepted Runtime: 108 ms

Case 1 Case 2

Input

MyNumbers =

num
—
8
8
3
3
1
4

• 100 Online

29) 1045. Customers Who Bought All Products

1045. Customers Who Bought All Products

MySQL ✓ Auto

```

1 # Write your MySQL query statement below
2 SELECT customer_id
3 FROM Customer
4 GROUP BY customer_id
5 HAVING COUNT(distinct product_key) = (SELECT COUNT(product_key) FROM Product);

```

Saved Ln 2, Col 1

Testcase Test Result

Accepted Runtime: 105 ms

Case 1

Input

Customer =

customer_id	product_key
1	5
2	6
3	5
3	6
1	6

Product =

product_key

• 100 Online

➤ Advanced Select and Joins

30) 1731. The Number of Employees Which Report to Each Employee

1731. The Number of Employees Which Report to Each Employee

MySQL v Auto

```

1 # Write your MySQL query statement below
2 SELECT e1.employee_id, e1.name, count(e2.employee_id)
3 AS reports_count, round(avg(e2.age)) AS average_age
4 FROM employees e1
5 JOIN employees e2
6 ON e1.employee_id = e2.reports_to
7 GROUP BY e1.employee_id, e1.name
8 ORDER BY e1.employee_id;

```

Saved Ln 2, Col 1

Testcase Test Result

Accepted Runtime: 101 ms

Case 1 Case 2

Input

Employees =

employee_id	name	reports_to	age
9	Hercy	null	43
6	Alice	9	41
4	Bob	9	36
2	Winston	null	37

Output

719 137 100 Online

31) 1789. Primary Department for Each Employee

1789. Primary Department for Each Employee

MySQL v Auto

```

1 # Write your MySQL query statement below
2 SELECT employee_id, department_id
3 FROM Employee
4 WHERE primary_flag='Y' OR
5 employee_id IN
6 (SELECT employee_id
7 FROM Employee
8 Group by employee_id
9 having count(employee_id)=1);

```

Saved Ln 9, Col 3

Testcase Test Result

Accepted Runtime: 124 ms

Case 1

Input

Employee =

employee_id	department_id	primary_flag
1	1	N
2	1	Y
2	2	N
3	3	N
4	2	N
4	3	Y

685 173 100 Online

32) 610. Triangle Judgement

610. Triangle Judgement

MySQL v Auto

```
1 # Write your MySQL query statement below
2 SELECT *, IF(x+y>z AND y+z>x AND z+x>y, "Yes", "No") AS triangle
3 FROM Triangle;
```

Table: Triangle

Column Name	Type
x	int
y	int
z	int

In SQL, (x, y, z) is the primary key column for this table.
Each row of this table contains the lengths of three line segments.

Report for every three line segments whether they can form a triangle.

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

The result format is in the following example.

Example 1:

Input

x	y	z
—	—	—
13	15	30
10	20	15

Output

x	y	z	triangle
—	—	—	—
13	15	30	No
10	20	15	Yes

Expected

Accepted Runtime: 71 ms

Case 1

720 96 100 Online

33) 180. Consecutive Numbers

180. Consecutive Numbers

MySQL v Auto

```
1 # Write your MySQL query statement below
2 SELECT DISTINCT l1.num AS ConsecutiveNums
3 FROM logs l1, logs l2, logs l3
4 WHERE l1.id=l2.id+1 AND l1.num=l2.num
5 AND l1.id=l3.id+2 AND l1.num= l3.num;
```

Table: Logs

Column Name	Type
id	int
num	varchar

In SQL, id is the primary key for this table.
id is an autoincrement column starting from 1.

Find all numbers that appear at least three times consecutively.

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

The result format is in the following example.

Example 1:

Input

id	num
—	—
1	1
2	1
3	1
4	2
5	1
6	2

Output

Logs =

id	num
—	—
1	1
2	1
3	1
4	2
5	1
6	2

View more

Accepted Runtime: 277 ms

Case 1

2.4K 235 100 Online

34) 1164. Product Price at a Given Date

1164. Product Price at a Given Date

Medium Topics Companies

SQL Schema > Pandas Schema >

Table: Products

```
+-----+-----+
| Column Name | Type |
+-----+-----+
| product_id | int  |
| new_price  | int  |
| change_date | date |
+-----+-----+
```

(product_id, change_date) is the primary key (combination of columns with unique values) of this table.
Each row of this table indicates that the price of some product was changed to a new price at some date.

Initially, all products have price 10.

Write a solution to find the prices of all products on the date 2019-08-16.

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

The result format is in the following example.

1.2K 169 • 100 Online

Code MySQL Auto

```
1 # Write your MySQL query statement below
2 SELECT product_id ,10 AS price
3 FROM products GROUP BY product_id HAVING min(change_date) > '2019-08-16'
4 UNION ALL
5 SELECT product_id , new_price AS price FROM products
6 WHERE( product_id , change_date) IN (SELECT product_id , max(change_date)
7 AS price FROM products WHERE change_date <= '2019-08-16' GROUP BY product_id);
```

Saved Ln 7, Col

Testcase Test Result Accepted Runtime: 66 ms Case 1 Input

Products =

product_id	new_price	change_date
1	20	2019-08-14
2	50	2019-08-14
1	30	2019-08-15
1	35	2019-08-16
2	65	2019-08-17
3	20	2019-08-18

35) 1204. Last Person to Fit in the Bus

1204. Last Person to Fit in the Bus

Medium Topics Companies

SQL Schema > Pandas Schema >

Table: Queue

```
+-----+-----+
| Column Name | Type |
+-----+-----+
| person_id | int  |
| person_name | varchar |
| weight | int  |
| turn | int  |
+-----+-----+
```

person_id column contains unique values.
This table has the information about all people waiting for a bus.
The person_id and turn columns will contain all numbers from 1 to n, where n is the number of rows in the table.
turn determines the order of which the people will board the bus, where turn=1 denotes the first person to board and turn=n denotes the last person to board.
weight is the weight of the person in kilograms.

There is a queue of people waiting to board a bus. However, the bus has a weight limit of 1000 kilograms, so there may be some people who cannot board.

924 134 • 100 Online

Code MySQL Auto

```
1 # Write your MySQL query statement below
2 SELECT queue1.person_name
3 FROM queue queue1 JOIN queue queue2
4 ON queue1.turn >= queue2.turn
5 GROUP BY queue1.turn HAVING sum(queue2.weight) <= 1000
6 ORDER BY sum(queue2.weight) desc
7 LIMIT 1;
```

Saved Ln 2, Col

Testcase Test Result Accepted Runtime: 89 ms Case 1 Input

Queue =

person_id	person_name	weight	turn
5	Alice	250	1
4	Bob	175	5
3	Alex	350	2
6	John Cena	400	3
1	Winston	500	6
2	Marie	200	4

36) 1907. Count Salary Categories

```

MySQL < Auto
1 # Write your MySQL query statement below
2 (SELECT "Low Salary" AS category, COUNT(*) AS accounts_count FROM accounts WHERE income < 20000)
3 UNION
4 (SELECT "Average Salary" AS category, COUNT(*) AS accounts_count FROM accounts WHERE income BETWEEN 20000 AND 50000)
5 UNION
6 (SELECT "High Salary" AS category, COUNT(*) AS accounts_count FROM accounts WHERE income > 50000)

```

Accepted Runtime: 72 ms

Case 1

Input

account_id	income
3	108939
2	12747
8	87709
6	91796

Output

➤ Subqueries

37) 1978. Employees Whose Manager Left the Company

```

MySQL < Auto
1 # Write your MySQL query statement below
2 SELECT DISTINCT e2.employee_id
3 FROM Employees e1, Employees e2
4 WHERE e2.Salary<30000 AND
5 e2.manager_id NOT IN(SELECT employee_id FROM Employees) ORDER BY employee_id;

```

Accepted Runtime: 82 ms

Case 1

Input

employee_id	name	manager_id	salary
3	Mila	9	60301
12	Antonella	null	31000
13	Emery	null	67084
1	Kaleel	11	21241
9	Mikaela	null	50937
11	Joziah	6	28485

38) 626. Exchange Seats

626. Exchange Seats

Medium Topics Companies

SQL Schema > Pandas Schema >

Table: Seat

Column Name	Type
<code>id</code>	<code>int</code>
<code>student</code>	<code>varchar</code>

`id` is the primary key (unique value) column for this table.
Each row of this table indicates the name and the ID of a student.
The ID sequence always starts from 1 and increments continuously.

Write a solution to swap the seat id of every two consecutive students. If the number of students is odd, the id of the last student is not swapped.

Return the result table ordered by `id` in ascending order.

The result format is in the following example.

• 1.7K • 134 • 100 Online

Code

```

</> Code
MySQL v Auto
1 = SELECT YOUR MySQL QUERY STATEMENT BELOW
2 SELECT CASE
3 WHEN s.id % 2 <> 0 AND s.id = (SELECT COUNT(*) FROM Seat) THEN s.id
4 WHEN s.id % 2 = 0 THEN s.id - 1
5 ELSE
6     s.id + 1
7 END AS id,
8 student
9 FROM Seat AS s
10 ORDER BY id

```

Saved Ln 10, Col 12

Testcase **Test Result**

Accepted Runtime: 73 ms

Case 1

Input

<code>id</code>	<code>student</code>
1	Abbot
2	Doris
3	Emerson
4	Green
5	Jeames

39) 1341. Movie Rating

1341. Movie Rating

Medium Topics Companies

SQL Schema > Pandas Schema >

Table: Movies

Column Name	Type
<code>movie_id</code>	<code>int</code>
<code>title</code>	<code>varchar</code>

`movie_id` is the primary key (column with unique values) for this table.
`title` is the name of the movie.

Table: Users

Column Name	Type
<code>user_id</code>	<code>int</code>
<code>name</code>	<code>varchar</code>

`user_id` is the primary key (column with unique values) for this table.
The column 'name' has unique values.

• 781 • 191 • 100 Online

Code

```

</> Code
MySQL v Auto
1 WITH
2 TheMostActiveUser AS (
3     SELECT name
4     FROM
5         Users
6     NATURAL JOIN MovieRating
7     GROUP BY user_id
8     ORDER BY COUNT(*) DESC, name
9     LIMIT 1

```

Saved Ln 27, Col 21

Testcase **Test Result**

Accepted Runtime: 135 ms

Case 1

Input

<code>movie_id</code>	<code>title</code>
1	Avengers
2	Frozen 2
3	Joker

<code>user_id</code>	<code>name</code>
1	John
2	Mary
3	Mike

40) 1321. Restaurant Growth

1321. Restaurant Growth

Medium Topics Companies Hint

SQL Schema > Pandas Schema >

Table: Customer

```

| Column Name | Type |
| customer_id | int |
| name         | varchar |
| visited_on   | date |
| amount       | int |

```

In SQL, `(customer_id, visited_on)` is the primary key for this table.
This table contains data about customer transactions in a restaurant.
`visited_on` is the date on which the customer with ID `(customer_id)` has visited the restaurant.
`amount` is the total paid by a customer.

You are the restaurant owner and you want to analyze a possible expansion (there will be at least one customer every day).
Compute the moving average of how much the customer paid in a seven days window (i.e., current day + 6 days before). `average_amount` should be **rounded to two decimal places**.

984 144 100 Online

MySQL < Auto

```

1 # Write your MySQL query statement below
2 SELECT visited_on,
3        sum(amount) over (rows between 6 preceding and current row) AS amount,
4        round(avg(amount)) OVER (rows between 6 preceding and current row),2) AS average_amount
5 FROM Customer
6 GROUP BY visited_on
7 ORDER BY visited_on
8 LIMIT 999999 offset 6;

```

Saved Ln 2, Col 1

Testcase Test Result Accepted Runtime: 67 ms Case 1

Input

customer_id	name	visited_on	amount
1	Jhon	2019-01-01	100
2	Daniel	2019-01-02	110
3	Jade	2019-01-03	120
4	Khaled	2019-01-04	130
5	Winston	2019-01-05	110
6	Elvis	2019-01-06	140

100 Online

41) 602. Friend Requests II: Who Has the Most Friends

602. Friend Requests II: Who Has the Most Friends

Medium Topics Companies Hint

SQL Schema > Pandas Schema >

Table: RequestAccepted

```

| Column Name | Type |
| requester_id | int |
| accepter_id | int |
| accept_date  | date |

```

`(requester_id, accepter_id)` is the primary key (combination of columns with unique values) for this table.
This table contains the ID of the user who sent the request, the ID of the user who received the request, and the date when the request was accepted.

Write a solution to find the people who have the most friends and the most friends number.
The test cases are generated so that only one person has the most friends.
The result format is in the following example.

808 102 100 Online

MySQL < Auto

```

1 # Write your MySQL query statement below
2 WITH base AS(SELECT requester_id id FROM RequestAccepted
3 UNION ALL
4 SELECT accepter_id id FROM RequestAccepted)
5
6 SELECT id, count(*) num FROM base GROUP BY 1 ORDER BY 2 DESC LIMIT 1;

```

Saved Ln 2, Col 1

Testcase Test Result Accepted Runtime: 82 ms Case 1

Input

requester_id	accepter_id	accept_date
1	2	2016/06/03
1	3	2016/06/08
2	3	2016/06/08
3	4	2016/06/09

Output

id	num
----	-----

42) 585. Investments in 2016

585. Investments in 2016

MySQL v Auto

```

1 -- Write your MySQL query statement below
2 SELECT ROUND(SUM(tiv_2016),2) AS tiv_2016
3 FROM Insurance
4 WHERE tiv_2015 IN (
5   SELECT tiv_2015
6   FROM Insurance
7   GROUP BY tiv_2015
8   HAVING COUNT(*) > 1
9 ) AND (lat, lon) IN (
10  SELECT lat, lon
11  FROM Insurance
12  GROUP BY lat, lon
13 )

```

Saved Ln 14, Col 1

Testcase **Test Result**

Accepted Runtime: 93 ms

Case 1

Input

pid	tiv_2015	tiv_2016	lat	lon
1	10	5	10	10
2	20	20	20	20
3	10	30	20	20
4	10	40	40	40

• 100 Online

43) 185. Department Top Three Salaries

185. Department Top Three Salaries

MySQL v Auto

```

1 SELECT Department, Employee, salary
2 FROM (
3   SELECT
4     dept.name AS Department,
5     emp.name AS Employee,
6     emp.salary,
7     DENSE_RANK() OVER (PARTITION BY dept.name ORDER BY emp.salary DESC) AS rate
8   FROM
9     Employee emp
10    JOIN
11      Department dept
12    ON
13      emp.departmentId = dept.id

```

Saved Ln 17, Col 1!

Testcase **Test Result**

Accepted Runtime: 97 ms

Case 1

Input

id	name	salary	departmentId
1	Joe	85000	1
2	Henry	80000	2
3	Sam	60000	2

• 100 Online

➤ Advanced String Functions / Regex / Clause

44) 1667. Fix Names in a Table

1667. Fix Names in a Table

MySQL Schema > Pandas Schema >

Table: Users

```

+-----+-----+
| Column Name | Type   |
+-----+-----+
| user_id     | int    |
| name        | varchar|
+-----+-----+

```

user_id is the primary key (column with unique values) for this table.
This table contains the ID and the name of the user. The name consists of only lowercase and uppercase characters.

Write a solution to fix the names so that only the first character is uppercase and the rest are lowercase.
Return the result table ordered by user_id.
The result format is in the following example.

Example 1:

MySQL code:

```

1 # Write your MySQL query statement below
2 SELECT user_id,
3       CONCAT(UPPER(LEFT(name, 1)),
4              LOWER(RIGHT(name, LENGTH(name) - 1)))
5  AS name
6 FROM Users
7 ORDER BY user_id;

```

Test Result: Accepted Runtime: 89 ms

Case 1

Input

user_id	name
1	aLice
2	bOB

Output

user_id	name
1	aLice
2	bOB

45) 1527. Patients With a Condition

1527. Patients With a Condition

MySQL Schema > Pandas Schema >

Table: Patients

```

+-----+-----+
| Column Name | Type   |
+-----+-----+
| patient_id  | int    |
| patient_name| varchar|
| conditions   | varchar|
+-----+-----+

```

patient_id is the primary key (column with unique values) for this table.
'conditions' contains 0 or more code separated by spaces.
This table contains information of the patients in the hospital.

Write a solution to find the patient_id, patient_name, and conditions of the patients who have Type I Diabetes.
Type I Diabetes always starts with DIAB1 prefix.
Return the result table in any order.
The result format is in the following example.

MySQL code:

```

1 # Write your MySQL query statement below
2 SELECT * FROM patients WHERE conditions REGEXP '^\\bDIAB1'

```

Test Result: Accepted Runtime: 76 ms

Case 1

Input

patient_id	patient_name	conditions
1	Daniel	YFEV COUGH
2	Alice	
3	Bob	DIAB100 MYOP
4	George	ACNE DIAB100
5	Alain	DIAB201

Output

patient_id	patient_name	conditions
3	Bob	DIAB100 MYOP

46) 196. Delete Duplicate Emails

196. Delete Duplicate Emails

MySQL v Auto

```
1 # Write your MySQL query statement below
2 DELETE p1 FROM Person p1,
3      Person p2
4 WHERE
5     p1.Email = p2.Email AND p1.Id > p2.Id
```

Saved Ln 5, Col 1

Testcase Test Result Accepted Runtime: 111 ms

Case 1

Input

Person =

id	email
1	john@example.com
2	bob@example.com
3	john@example.com

Output

id	email
----	-------

1.8K 287 48 Online

47) 176. Second Highest Salary

176. Second Highest Salary

MySQL v Auto

```
1 # Write your MySQL query statement below
2 select
3 (select distinct Salary
4 from Employee order by salary desc
5 limit 1 offset 1)
6 as SecondHighestSalary;
```

Saved Ln 6, Col 2

Testcase Test Result Accepted Runtime: 577 ms

Case 1 Case 2

Input

Employee =

id	salary
1	100
2	200
3	300

Output

SecondHighestSalary

3.9K 324 98 Online

48) 1484. Group Sold Products By The Date

1484. Group Sold Products By The Date

MySQL Schema > Pandas Schema >

Table Activities:

```
+-----+-----+
| Column Name | Type   |
+-----+-----+
| sell_date   | date   |
| product     | varchar|
+-----+-----+
```

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

Each row of this table contains the product name and the date it was sold in a market.

Write a solution to find for each date the number of different products sold and their names.

The sold products names for each date should be sorted lexicographically.

Return the result table ordered by `sell_date`.

The result format is in the following example.

Code:

```
1 # Write your MySQL query statement below
2 select sell_date, count(DISTINCT product) as num_sold,
3 GROUP_CONCAT(DISTINCT product order by product ASC separator ',') as products
4 FROM Activities GROUP BY sell_date order by sell_date ASC;
```

Testcase | Test Result

Accepted Runtime: 75 ms

Case 1

Input

Activities =

sell_date	product
2020-05-30	Headphone
2020-06-01	Pencil
2020-06-02	Mask
2020-05-30	Basketball
2020-06-01	Bible
2020-06-02	Mask

View more

Output

49) 1327. List the Products Ordered in a Period

1327. List the Products Ordered in a Period

MySQL Schema > Pandas Schema >

Table: Products

```
+-----+-----+
| Column Name | Type   |
+-----+-----+
| product_id  | int    |
| product_name| varchar|
| product_category| varchar|
+-----+-----+
```

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

This table contains data about the company's products.

Table: Orders

```
+-----+-----+
| Column Name | Type   |
+-----+-----+
| product_id  | int    |
| order_date  | date   |
| unit        | int    |
+-----+-----+
```

Code:

```
1 # Write your MySQL query statement below
2 SELECT p.product_name, sum(o.unit) AS unit
3 FROM Products p
4 JOIN Orders o
5 ON p.product_id=o.product_id
6 WHERE o.order_date>'2020-02-01' AND o.order_date<='2020-02-29'
7 GROUP BY o.product_id HAVING unit>=100 ;
```

Testcase | Test Result

Accepted Runtime: 109 ms

Case 1

Input

Products =

product_id	product_name	product_category
1	Leetcode Solutions	Book
2	Jewels of Stringology	Book
3	HP	Laptop
4	Lenovo	Laptop
5	Leetcode Kit	T-shirt

Orders =

50) 1517. Find Users With Valid E-Mails

SQL 50 < > Premium

Description | Editorial | Solutions | Submissions

1517. Find Users With Valid E-Mails

Easy Topics Companies

SQL Schema > Pandas Schema >

Table: Users

Column Name	Type
user_id	int
name	varchar
mail	varchar

user_id is the primary key (column with unique values) for this table.
This table contains information of the users signed up in a website. Some e-mails are invalid.

Write a solution to find the users who have valid emails.

A valid e-mail has a prefix name and a domain where:

- The prefix name is a string that may contain letters (upper or lower case), digits, underscore '_', period '.', and/or dash '-'. The prefix name **must** start with a letter.
- The domain is '@leetcode.com'.

604 111 25 Online

Code MySQL Auto

```
1 # Write your MySQL query statement below
2 SELECT * FROM Users
3 WHERE regexp_like(mail, '^[A-Za-z]+[A-Za-z0-9_\\.-]*@leetcode\\.com$')
```

Testcase Test Result

Accepted Runtime: 90 ms

Case 1

Input

user_id	name	mail
1	Winston	winston@leetcode.com
2	Jonathan	jonathanisgreat
3	Annabelle	bella@leetcode.com
4	Sally	sally.com@leetcode.com
5	Marwan	quarz#2020@leetcode.com
6	David	david69@gmail.com

Output

user_id	name	mail
1	Winston	winston@leetcode.com
2	Jonathan	jonathanisgreat
3	Annabelle	bella@leetcode.com
4	Sally	sally.com@leetcode.com
5	Marwan	quarz#2020@leetcode.com
6	David	david69@gmail.com