Name:- Vaibhav kumar gupta

Date:- 10-02-2025

1. **Relational Database Management System (RDBMS)**

RDBMS is a type of database management system that stores data in a structured format using tables. It ensures data integrity, consistency, and supports relationships between tables.

2. Table and Database

- **Database**: A collection of organized data that can be easily accessed, managed, and updated.
- **Table**: A structured collection of related data in rows and columns within a database.

Commands:-

1 Creating a Database

```
CREATE DATABASE my_database;
```

Explanation:- This command creates a new database named my_database.

2. Creating a Table

```
CREATE TABLE employees (
   id INT PRIMARY KEY,
   name VARCHAR(100),
   age INT,
   department VARCHAR(50)
);
```

Explanation:- Defines an employees table with relevant columns.

3. Inserting Data

```
INSERT INTO employees (id, name, age, department)
VALUES (1, 'Alice', 30, 'HR');
```

Explanation:- Adds a record to the employees table.

4. Deleting Data

```
DELETE FROM employees WHERE id = 1;
```

Explanation:- Removes the record where id is 1.

5. Truncating a Table

```
TRUNCATE TABLE employees;
```

Explanation:- Clears all data from the employees table.

6. Renaming a Table

ALTER TABLE employees RENAME TO staff;

Explanation:- Changes table name from employees to staff.

7. Modifying a Column

ALTER TABLE employees RENAME COLUMN department TO dept;

Explanation:- Renames department column to dept.

8. Using Views

CREATE VIEW hr_employees AS

SELECT name, department FROM employees WHERE department = 'HR';

Explanation:- Creates a hr_employees view.

9. Ordering Data

SELECT * FROM employees ORDER BY age DESC;

Explanation:- Sorts results by age in descending order.

10. Filtering Data

SELECT * FROM employees WHERE age > 25;

Explanation:- Selects employees older than 25.

11. Deep vs Shallow Copy

- **Shallow Copy:**

CREATE TABLE employees_copy AS SELECT * FROM employees;

- **Deep Copy:**

CREATE TABLE employees_copy like employees;

1757. Recyclable and Low Fat Products

Solved

Easy | Topics | Companies

SQL Schema > Pandas Schema >

Table: `Products`

```
+-------------+---------+
| Column Name | Type    |
+-------------+---------+
| product_id  | int     |
| low_fats    | enum    |
| recyclable  | enum    |
+-------------+---------+
product_id is the primary key (column with unique values) for this table.
low_fats is an ENUM (category) of type ('Y', 'N') where 'Y' means this product is
low fat and 'N' means it is not.
recyclable is an ENUM (category) of types ('Y', 'N') where 'Y' means this product
is recyclable and 'N' means it is not.
```

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

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

Code

MySQL | Auto

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

```

Input

Products =

```
| 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          |
```

584. Find Customer Referee

Easy | Topics | Companies | Hint

SQL Schema > Pandas Schema >

Table: `Customer`

```
+-------------+---------+
| Column Name | Type    |
+-------------+---------+
| id          | int     |
| name        | varchar |
| referee_id  | int     |
+-------------+---------+
In SQL, id is the primary key column for this table.
Each row of this table indicates the id of a customer, their name, and the id of
the customer who referred them.
```

Find the names of the customer that are **not referred by** the customer with `id = 2`.

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

Code

MySQL | Auto

```
# Write your MySQL query statement below
Select name from customer where referee_id!=2 or referee_id is null ;
```

Testcase | Test Result

Input

Customer =

```
| 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 |
```

595. Big Countries

Easy | Topics | Companies

SQL Schema > Pandas Schema >

Table: `World`

```
+-------------+---------+
| Column Name | Type    |
+-------------+---------+
| name        | varchar |
| continent   | varchar |
| area        | int     |
| population  | int     |
| gdp         | bigint  |
+-------------+---------+
name is the primary key (column with unique values) for this table.
Each row of this table gives information about the name of a country, the
continent to which it belongs, its area, the population, and its GDP value.
```

A country is **big** if:

- it has an area of at least three million (i.e., `3000000 km²`), or
- it has a population of at least twenty-five million (i.e., `25000000`).

Code

MySQL Auto

```mysql
# Write your MySQL query statement below
Select name,population,area from world where area>=3000000 or population>=25000000;
```

Input

World =

```
| 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 |
```

1148. Article Views I

Easy Topics Companies

SQL Schema > Pandas Schema >

Table: `Views`

```
+---------------+---------+
| Column Name   | Type    |
+---------------+---------+
| article_id    | int     |
| author_id     | int     |
| viewer_id     | int     |
| view_date     | date    |
+---------------+---------+
There is no primary key (column with unique values) for this table, the table may
have duplicate rows.
Each row of this table indicates that some viewer viewed an article (written by
some author) on some date.
Note that equal author_id and viewer_id indicate the same person.
```

Write a solution to find all the authors that viewed at least one of their own articles.

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

The result format is in the following example.

Code

MySQL Auto

```mysql
# Write your MySQL query statement below
Select distinct author_id as id
from views
where author_id=viewer_id
order by author_id;
```

Input

Views =

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

View more

Output

id
4
7

SQL Schema > Pandas Schema >

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 characters on an American Keyboard, 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.

Code

MySQL Auto

```mysql
# Write your MySQL query statement below
select tweet_id from tweets where length(content) >15;
```

Testcase | Test Result

Input

Tweets =

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

Output

```
| tweet_id |
| -------- |
| 2        |
```

1378. Replace Employee ID With The Unique Identifier

Easy | Topics | Companies

SQL Schema > Pandas Schema >

Table: `Employees`

```
+----------------+----------+
| Column Name    | Type     |
+----------------+----------+
| id             | int      |
| name           | varchar  |
+----------------+----------+
id is the primary key (column with unique values) for this table.
Each row of this table contains the id and the name of an employee in a company.
```

Table: `EmployeeUNI`

```
+----------------+----------+
| Column Name    | Type     |
+----------------+----------+
| id             | int      |
| unique_id      | int      |
+----------------+----------+
(id, unique_id) is the primary key (combination of columns with unique values)
for this table.
```

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.

Code

MySQL | Auto

```mysql
# Write your MySQL query statement below
select unique_id , name from employees as ese
left join
EmployeeUni as euni
on ese.id = euni.id;
```

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

620. Not Boring Movies

Easy | Topics | Companies

SQL Schema > Pandas Schema >

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 so we do not include it in the answer.
```

620. Not Boring Movies

Attempted

Easy | Topics | Companies

SQL Schema > Pandas Schema >

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.

Code

MySQL | Auto

```mysql
# Write your MySQL query statement below
select * from cinema where id%2!=0 and description!="boring" order by rating  desc;
```

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 so we do not include it in the answer.

1251. Average Selling Price

Easy | Topics | Companies

SQL Schema > Pandas Schema >

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) for this 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 intersecting periods for the same product_id.
```

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**.

1251. Average Selling Price

Attempted

Easy | Topics | Companies

SQL Schema > Pandas Schema >

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) for this 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 intersecting periods for the same product_id.
```

Code

MySQL | Auto

```mysql
# Write your MySQL query statement below
select p.product_id,round(sum(p.price * uni.units)/ sum(uni.units),2) as average_price from
prices as p left join unitsSold as uni
on p.product_id= uni.product_id
where (uni.purchase_date between p.start_date and p.end_date)
group by p.product_id  ;
```