RDBMS and MySQL

RDBMS stands for **Relational Database Management Systems.** A database is an organized collection of data stored in a computer system and usually controlled by a database management system (DBMS). The data in common databases is modeled in tables, making querying and processing efficient.

What is RDBMS?

RDBMS stands for Relational Database Management Systems. It is a program that allows us to create, delete, and update a relational database**. A Relational Database is a database system that stores and retrieves data in a tabular format organized in the form of rows and columns**.

What do you mean by SQL?

Introduction to SQL:

SQL is a standard language for accessing and manipulation of database.

★ What is SQL ?

- it stands for structured Query Language
- it lets us access and manipulate databases.
- it became a standard of American National standard Institute (ANSI) in 1986, and international Organization for standardization (ISO) in 1987.

Tables:

A Table is a collection of related data entries and it consists of columns and rows.

① Fields: it refers to column in a Table, it is designed to maintain specific information about every record in the Table. it contains all information associated with a specific field in a Table.

② Record: it refers to row in a Table, and it holds data about each individual data.

SQL Create Database:

The "Create Database" statement is used to create a new SQL database.

★ Syntax:

CREATE DATABASE database name;

★ Tip:

Make sure you have admin privilege before creating any database. Once a database is created, you can check it in the list of databases with the following SQL command:
SHOW DATABASES;

SQL DROP Database:

The "DROP Database" statement is used to drop an existing SQL Database.

★ Syntax:

DROP DataBase database name;

SQL BACKUP Database:

The "BACKUP DATABASE" statement is used in SQL server to create a full back up of an existing SQL database.

★ Syntax:

BACKUP DATABASE database name TO DISK = 'File Path';

Extension: SQL BACKUP with Differential statement:

A differential backup only backs up the parts of the database that have changed since last full database backup.

★ Syntax:

BACKUP DATABASE database name TO DISK = 'file Path' with DIFFERENTIAL;

SQL CREATE Table:

The CREATE TABLE statement is used to create a new table in a database.

★ Syntax:

CREATE TABLE table_name (column1, data type, column2 data type...);

* Example:

CREATE TABLE Persons (PersonID int, LastName varchar (255), First Name varchar (255), Address varchar (255));

* AND TO INSERT INTO Table:

INSERT INTO table-name (colum 1 , column 2, ...) Values (value 1, Value2, ...);

* Extension: To Create a Table using Another Table

Syntax:

CREATE TABLE new-table-name AS SELECT column 1, column2, ...
FROM Existing-table-name where condition;

[This helps us To create a copy of an existing table. The new Table gets The same column Definitions. (NOTE: All columns or specific columns can be selected.)].

SQL Select Statement:

The "SELECT" statement is used to select data from a database. The data returned is stored in a result Table, called The result-set.

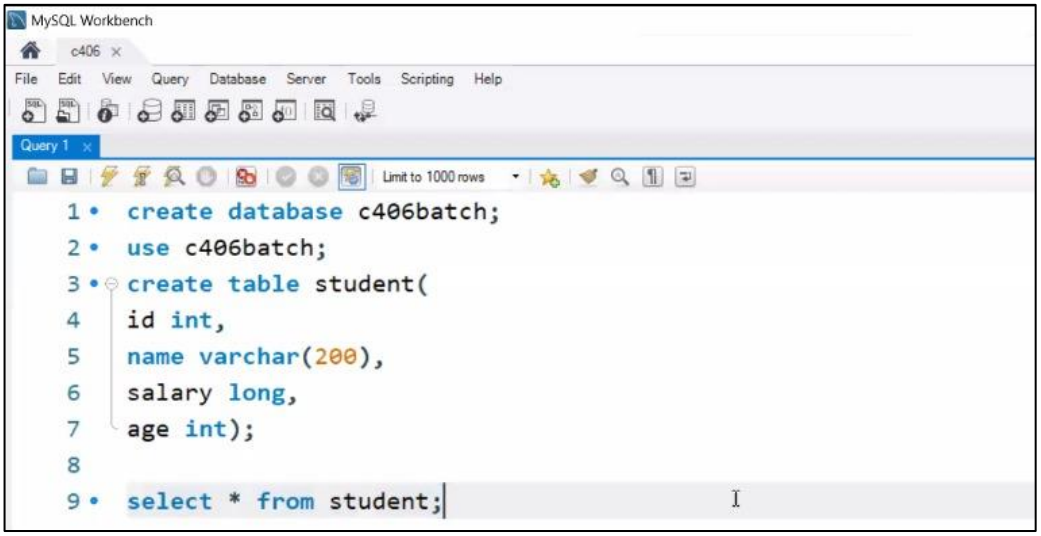
SYNTAX:-

SELECT COLUMN_NAME_1 , COLUMN_NAME_2 , COLUMN_NAME_3 FROM Table_Name;

(here column_Name_1, 2, 3 are The field names of The Table we want To select data from)

* If we want To SELECT All The fields Available in Table:

SELECT * FROM Table_name;



SQL SELECT DISTINCT Statement:

The "SELECT DISTINCT" statement is used to return only distinct (different values). Inside a table, a column often contains many duplicate values; and sometimes we need only the list of distinct (or different) values.

Syntax:-

```
SELECT DISTINCT COLUMN1, COLUMN2, ... FROM Table_name;
```

★ If we want to count the Distinct entries for a column:

```
SELECT COUNT (DISTINCT COUNTRY) FROM Tourist_Table_Name;
```

OR we can also write it as...

```
SELECT COUNT (*) AS DISTINCT column_name FROM (
    SELECT DISTINCT COUNTRY FROM Customers);
```

(or alias name)

The Most Important SQL Commands:

- SELECT - extracts data from a database.
- UPDATE - updates data in a database
- DELETE - deletes data from a database
- INSERT INTO - inserts new data into a database
- CREATE DATABASE - creates a new database
- ALTER DATABASE - modifies a database
- CREATE TABLE - creates a new Table
- ALTER TABLE - modifies a Table
- DROP TABLE - deletes a Table
- CREATE INDEX - creates an index (search Key)
- DROP INDEX - deletes an index

P.T.O...

SQL WHERE CLAUSE:

The "WHERE" CLAUSE is used to filter records.

It is used to extract only those records that fulfill a specific condition.

SYNTAX:-

SELECT column_name FROM table_name WHERE condition;

NOTE: The "WHERE" clause is not only used "SELECT" statements, it is also used in "UPDATE", "DELETE", etc.!

eg: SELECT * FROM CUSTOMERS WHERE COUNTRY = 'Mexico';

it would return all the records (rows) which have country as mexico.

</> Code

MySQL Auto

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

OPERATORS IN The WHERE CLAUSE:

OPERATOR	DESCRIPTION	EXAMPLE
'='	Equal	SELECT * FROM PRODUCTS WHERE PRICE = 18;
'>'	GREATER Than	SELECT * FROM PRODUCTS WHERE Price>
'<'	LESS Than	" " " " " Price < 30;
'>='	GREATER Than or equal	" " " " " Price ≥ 30;
'<=' OR	LESS Than or equal	" " " " " Price ≤ 30;
'<>' OR '!='	NOT equal	" " " " " Price <> 30;
'BETWEEN'	BETWEEN a Certain range	" " " " where Price BETWEEN 50 AND 60;
'LIKE'	Search for a Pattern	SELECT * FROM CUSTOMER WHERE city LIKE 's%'; (returns all cities starting with s)
'IN'	To specify Multiple Possible Values for a column	SELECT * FROM CUSTOMER WHERE City IN ('Paris', 'LONDON'); (returns customers who belong to city Paris, LONDON)

SQL AND, OR and NOT operators:

The "where" clause can be combined with AND, OR, and NOT operators.

The AND and OR operators are used to filter records based on more than one condition.

* The AND operator displays a record if all the conditions separated by AND are TRUE
* The OR operator displays a record if any of the conditions separated by OR is TRUE
* The NOT operator displays a record if the condition is NOT TRUE.

AND Syntax:

```
SELECT column1, column2... FROM Table_name WHERE CONDITION1
AND CONDITION2 AND CONDITION 3... ;
```

OR Syntax:

```
SELECT column1 FROM Table_name WHERE CONDITION1 OR CONDITION2;
```

NOT Syntax:

```
SELECT column1, column2... FROM table_name WHERE NOT CONDITION;
```

Examples:

* `SELECT * FROM CUSTOMER WHERE COUNTRY='GERMANY' AND CITY='MEXICO';`
* `SELECT * FROM CUSTOMER WHERE COUNTRY='Germany' OR COUNTRY='BERLIN';`
* `SELECT * FROM CUSTOMER WHERE NOT COUNTRY='GERMANY';`

Code

MySQL Auto

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

COMBINING AND, OR and NOT:

eg: SELECT * FROM CUSTOMER WHERE COUNTRY = 'GERMANY' AND (CITY = 'BERLIN' OR CITY = 'MUNCHEN');

eg: SELECT * FROM CUSTOMER WHERE NOT COUNTRY = 'GERMANY' AND NOT COUNTRY = 'USA';

SQL ORDER BY KEYWORD:

The 'ORDER BY' Keyword is used to sort the result set in ascending or descending order.

The 'ORDER BY' Keyword sorts the records in ascending order by default. To sort the records in Descending order, we use the DESC Keyword.

★ SYNTAX:

SELECT column1, column2,... FROM table_name ORDER By column1, column2,... ASC/DESC;

① eg: To Sort in Ascending Order:

SELECT * FROM CUSTOMER ORDER By COUNTRY;

② eg: To Sort in Descending Order:

SELECT * FROM CUSTOMER ORDER By COUNTRY DESC;

③ eg: To use ORDER BY on SEVERAL columns:

SELECT * FROM CUSTOMER ORDER By COUNTRY, CUSTOMER NAME;

(Explanation: This means it orders By COUNTRY, But if some rows have same COUNTRY, it then orders them by Customer names.)

④ eg: To USE ORDER By on SEVERAL columns But in Different ORDERS:

SELECT * FROM CUSTOMER ORDER BY COUNTRY ASC, CUSTOMER NAME DESC;

(Explanation: The logic remains same as previous query just order of sorting changes.)

PAGE-6

Code

MySQL Auto

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

SQL INSERT INTO STATEMENT:

The "INSERT INTO" statement is used to insert new records in the table.

SYNTAX:

① INSERT INTO table_name (column1, column2, ...) Values (value1, value2, value3 ....);

(here we specified both the column names and the values to be inserted.)

②. INSERT INTO table_name VALUES (value1, value2, value3 ...);

(if we are adding values for all the columns of table, then we do not need to specify column name in the query. However, we have to make sure the order of the values is in the same order as the columns in the table)

★ Examples:

INSERT INTO CUSTOMERS (CUSTOMER NAME, CONTACT NAME, ADDRESS, CITY, POSTAL CODE, COUNTRY) VALUES ('CARDINAL', 'TOM B. Erichsen', 'SKAGEN 21', 'STAVANGER', '4006', 'NORWAY');

★ Example: TO INSERT DATA ONLY IN SPECIFIED COLUMNS

INSERT INTO CUSTOMERS (CUSTOMER NAME, CITY, COUNTRY) Values ('Cardinal', 'Stavanger', 'Norway');

// The other columns would be filled with null value

Example

```
INSERT INTO Customers (CustomerName, ContactName, Address, City, PostalCode, Country)
VALUES
('Cardinal', 'Tom B. Erichsen', 'Skagen 21', 'Stavanger', '4006', 'Norway'),
('Greasy Burger', 'Per Olsen', 'Gateveien 15', 'Sandnes', '4306', 'Norway'),
('Tasty Tee', 'Finn Egan', 'Streetroad 19B', 'Liverpool', 'L1 0AA', 'UK');
```

SQL NULL VALUES:

A field with a "NULL VALUE" is a field with no value. if a field in a Table is optional, it is possible to insert a new record or update a record without adding a value To That field. Then The field will be saved with a NULL Value.

To Test for NULL VALUES:

Here we use "IS NULL" and "IS NOT NULL" OPERATORS.

* IS NULL SYNTAX:

SELECT column_names FROM Table_name WHERE column_name IS NULL;

* IS NOT NULL SYNTAX:

SELECT column_names FROM Table_name WHERE column_name IS NOT NULL;

Examples:

① SELECT CUSTOMER NAME, Contact Name, Address FROM CUSTOMERS WHERE Address IS NULL;

② SELECT CUSTOMER NAME, CONTACT NAME, ADDRESS FROM CUSTOMER WHERE Address IS NOT NULL;

IS NULL Syntax

```
SELECT column_names
FROM table_name
WHERE column_name IS NULL;
```

IS NOT NULL Syntax

```
SELECT column_names
FROM table_name
WHERE column_name IS NOT NULL;
```

SQL UPDATE STATEMENT:

The "UPDATE STATEMENT" is used to modify existing records in a table.

SYNTAX:

UPDATE table_name SET column1 = value1, column2 = value2, ...
where condition;

[NOTE: Here the Where clause specifies which records should be updated.
if we omit the where clause, all records in the table will be updated]

* Example:

UPDATE CUSTOMERS SET CONTACT NAME = 'Alfred Schmidt', CITY = 'FRANKFURT'
Where CUSTOMER ID = 1;

UPDATE MULTIPLE RECORDS:

It is "where" clause that determines how many records will be updated.

* eg: UPDATE CUSTOMER SET CONTACT NAME = 'JUAN' WHERE COUNTRY = 'Mexico';

[NOTE: Here if we omit the Where clause, All the records of ATTRIBUTE (CONTACT NAME) would be updated to "JUAN"]

Example

```
UPDATE Customers
SET ContactName = 'Alfred Schmidt', City= 'Frankfurt'
WHERE CustomerID = 1;
```

Example

```
UPDATE Customers
SET ContactName='Juan'
WHERE Country='Mexico';
```

SQL DELETE STATEMENT:

The "DELETE" STATEMENT is used to DELETE EXISTING RECORDS IN A TABLE.

* SYNTAX:

DELETE FROM table-Name WHERE condition;

[NOTE: Here the where clause specifies which records should be deleted. if we "omit the Where clause", all records in the Table will be DELETED.]

* Example:

DELETE FROM CUSTOMERS WHERE CUSTOMERNAME = 'Alfred';

DELETING ALL RECORDS:

it is POSSIBLE to delete all rows in a Table without deleting the Table.

This means that the Table structure, attributes, and indexes will be intact.

* SYNTAX:

DELETE FROM table_name;

Example

```
DELETE FROM Customers WHERE CustomerName='Alfreds Futterkiste';
```

SQL MIN() and MAX() FUNCTIONS:

The "Min()" function returns the smallest value of selected column.

The "Max()" function returns the largest value of the selected column.

* MIN() SYNTAX:

SELECT MIN (column_name) FROM table_name where condition;

eg: SELECT MIN(PRICE) AS smallest price FROM Products;
(Alias)

* MAX() SYNTAX:

SELECT MAX (column_name) FROM table_name where condition;

eg: SELECT MAX (PRICE) AS LARGEST PRICE FROM PRODUCTS;

SQL TOP, LIMIT, FETCH FIRST OR ROWNUM CLAUSE:

The "SELECT TOP" clause is used to specify the number of records to return.

The SELECT TOP clause is USEFUL ON LARGE TABLES with thousands of records.

[NOTE: Not all database systems support the SELECT TOP clause. My SQL SUPPORTS the LIMIT CLAUSE to select a limited number of records while ORACLE USES FETCH FIRST n ROWS ONLY and ROWNUM.]

★ SYNTAX FOR SQL SERVER/ MS ACCESS Syntax:

SELECT TOP NUMBER\PERCENT COLUMN_Name FROM Table_name WHERE CONDITION;

eg: SELECT TOP 3 * FROM CUSTOMERS;

★ SYNTAX FOR My SQL:

SELECT Column_name FROM table_name WHERE condition LIMIT number;

eg: SELECT * FROM CUSTOMERS LIMIT 3;

★ ~~ORACLE 12 SYNTAX:~~

~~SELECT column_name FROM table_name WHERE ROWNUM <= number;~~

~~eg: SELECT * FROM CUSTOMER WHERE ROWNUM <= 3;~~

★ OLDER ORACLE SYNTAX:

SELECT column_name FROM table_name WHERE ROWNUM ≤ number;

★ ORACLE 12 SYNTAX:

SELECT column_name FROM table_name ORDER BY column_name FETCH FIRST NUMBER ROWS ONLY;

eg: SELECT * FROM CUSTOMER FETCH FIRST 3 ROWS ONLY;

SQL DROP TABLES:

The "DROP TABLE" statement is used to drop an existing table in a database.

★ Syntax:

DROP Table table_name;

SQL TRUNCATE TABLES:

The "TRUNCATE TABLE" statement is used to delete the data inside a table, but not table itself.

★ Syntax:

TRUNCATE TABLE table-name:

Example

```
DROP TABLE Shippers;
```

Syntax

```
TRUNCATE TABLE table_name;
```

SQL ALTER TABLE:

The "ALTER TABLE" statement is used to add, delete or modify columns in an existing table. The Alter table statement is also used to add and drop various constraints on an existing table.

★ The Various Alter table Commands Syntax:

- ALTER TABLE - ADD Column: To add a column

  Syntax:

  ALTER TABLE table_name ADD Column_name data type;

- ALTER TABLE - DROP Column: To Delete a column.

  Syntax:

  ALTER TABLE table_name DROP COLUMN Column_name;

- ALTER TABLE - RENAME Columns: To rename a column:

  Syntax:

  ALTER TABLE table_name RENAME COLUMN old_name to new_name;

- ALTER TABLE - ALTER/ Modify Datatypes: To change the datatype of column.

  Syntax:

  ALTER TABLE table_name MODIFY COLUMN Column_name datatype;

  Or

  ALTER TABLE table_name MODIFY column_name Data type;

P.T.O...

ALTER TABLE - ADD Column

Example

```
ALTER TABLE Customers
ADD Email varchar(255);
```

ALTER TABLE - DROP COLUMN

Example

```
ALTER TABLE Customers
DROP COLUMN Email;
```

ALTER TABLE - RENAME COLUMN

To rename a column in a table, use the following syntax:

```
ALTER TABLE table_name
RENAME COLUMN old_name to new_name;
```

EXAMPLE:

```
EXEC sp_rename 'table_name.old_name',  'new_name', 'COLUMN';
```

ALTER TABLE - ALTER/MODIFY DATATYPE

To change the data type of a column in a table, use the following syntax:

SQL Server / MS Access:

```
ALTER TABLE table_name
ALTER COLUMN column_name datatype;
```

SQL DATA TYPES FOR MY SQL, SQL SERVER & MS ACCESS:

★ STRING DATA TYPES:

DATA TYPE	DESCRIPTION
• CHAR (size)	A fixed length string (can contain letters, number and special characters). The size parameter specifies column length in characters - can be from 0 to 255. Default is 1
• Varchar (size)	(Can contain letter, number & special characters) Max length → 0 to 65535
• BINARY (size)	Equal to CHAR(), but stores binary byte string
• VARBINARY (size)	used to store binary byte strings
• TINYBLOB	for BLOBs (Binary Large Objects). Max Length 255 Bytes
• ENUM (val1, val2...)	A string object that can have only one value, choosen from a list of possible values. if a value is inserted that is not in the list, a blank value will be inserted
• SET (val1, val2...)	A string object that can have 0 or more values, chosen from a list of possible values

★ Numeric Data Types:

DATA TYPE	DESCRIPTION
• INT (size)	used to store INTEGER VALUE
• Float (size, d)	The total number of digits is specified in size. The number of digits after the decimal point is specified in the d parameter

Leet-code important SQL questions:

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

```sql
# 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

```mysql
# 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.
```

Code

MySQL | Auto

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

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**.

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  ;
```