SELECT name From Customer

where referee\_id Is Null Or referee\_id !=2;

## What is a NULL Value?

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 this field. Then, the field will be saved with a NULL value.

**Note:** A NULL value is different from a zero value or a field that contains spaces. A field with a NULL value is one that has been left blank during record creation!

## How to Test for NULL Values?

It is not possible to test for NULL values with comparison operators, such as =, <, or <>.

We will have to use the IS NULL and IS NOT NULL operators instead.

### 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;

------------------------------------------------------------------------------------------------------------------------

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

Write a solution to find the name, population, and area of the **big countries**.

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

The result format is in the following example.

SELECT name, population,area From World

where area >= 3000000 or population >= 25000000;

------------------------------------------------------------------------------------------------------------------------

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.

**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 |

+------+

# Write your MySQL query statement below

Select Distinct author\_id as id From Views

where author\_id = viewer\_id order by id;

------------------------------------------------------------------------------------------------------------------------

Table: Tweets

+----------------+---------+

| Column Name | Type |

+----------------+---------+

| tweet\_id | int |

| content | varchar |

+----------------+---------+

tweet\_id is the primary key (column with unique values) for this table.

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 | Vote for Biden |

| 2 | Let us make America great again! |

+----------+----------------------------------+

**Output:**

+----------+

| tweet\_id |

+----------+

| 2 |

+----------+

**Explanation:**

Tweet 1 has length = 14. It is a valid tweet.

Tweet 2 has length = 32. It is an invalid tweet.

Select tweet\_id From Tweets

where Length(content) > 15;

# Write your MySQL query statement below

SELECT tweet\_id From Tweets

where Length(content) >15 ;

------------------------------------------------------------------------------------------------------------------------

Joins in SQL are used to combine rows from two or more tables based on a related column between them. There are several types of joins:

1. **Inner Join**: Returns rows when there is a match in both tables.
2. **Left (Outer) Join**: Returns all rows from the left table, and the matched rows from the right table. If there is no match, NULLs are returned for columns of the right table.
3. **Right (Outer) Join**: Returns all rows from the right table, and the matched rows from the left table. If there is no match, NULLs are returned for columns of the left table.
4. **Full (Outer) Join**: Returns rows when there is a match in one of the tables.
5. **Cross Join**: Returns the Cartesian product of rows from tables involved in the join. It pairs each row from the first table with every row from the second table.

Imagine we have two tables: Students and Courses.

* **Students** table:

| StudentID | StudentName |
| --- | --- |
| 1 | Alice |
| 2 | Bob |
| 3 | Carol |

* **Courses** table:

| CourseID | CourseName | StudentID |
| --- | --- | --- |
| 101 | Mathematics | 1 |
| 102 | English | 2 |
| 103 | Science | 1 |
| 104 | History | 4 |

### Inner Join Example:

We want to find out which students are enrolled in which courses. Since we're looking for matches between the two tables based on the StudentID, we can use an Inner Join.

sqlCopy code

SELECT Students.StudentName, Courses.CourseName

FROM Students

INNER JOIN Courses ON Students.StudentID = Courses.StudentID;

This SQL query will join the Students and Courses tables on the StudentID column and return rows where there is a match in both tables.

### Expected Result:

The result of this query will be:

| StudentName | CourseName |
| --- | --- |
| Alice | Mathematics |
| Bob | English |
| Alice | Science |

### Explanation:

* Alice (StudentID 1) is matched with Mathematics and Science, as she is enrolled in both these courses.
* Bob (StudentID 2) is matched with English.
* Carol is not listed in the result because she is not enrolled in any course listed in the Courses table.
* The History course is not in the result because there's no student with StudentID 4 in the Students table.

This example demonstrates how Inner Join works by only returning rows that have corresponding matches in both tables based on the specified condition.

### Example Scenario:

Let's say we have two tables: Employees and Departments.

* **Employees** table:

| EmployeeID | EmployeeName | DepartmentID |
| --- | --- | --- |
| 1 | John Doe | 101 |
| 2 | Jane Smith | 102 |
| 3 | Anne Brown | 101 |

* **Departments** table:

| DepartmentID | DepartmentName |
| --- | --- |
| 101 | HR |
| 102 | IT |
| 103 | Marketing |

### Examples of Different Joins:

1. **Inner Join**:

sqlCopy code

SELECT Employees.EmployeeName, Departments.DepartmentName

FROM Employees

INNER JOIN Departments ON Employees.DepartmentID = Departments.DepartmentID;

This will return rows where there is a match in both Employees and Departments tables:

| EmployeeName | DepartmentName |
| --- | --- |
| John Doe | HR |
| Jane Smith | IT |
| Anne Brown | HR |

1. **Left Join**:

sqlCopy code

SELECT Employees.EmployeeName, Departments.DepartmentName

FROM Employees

LEFT JOIN Departments ON Employees.DepartmentID = Departments.DepartmentID;

This includes all employees, even if their department isn't listed:

| EmployeeName | DepartmentName |
| --- | --- |
| John Doe | HR |
| Jane Smith | IT |
| Anne Brown | HR |

1. **Right Join**:

sqlCopy code

SELECT Employees.EmployeeName, Departments.DepartmentName

FROM Employees

RIGHT JOIN Departments ON Employees.DepartmentID = Departments.DepartmentID;

This includes all departments, even if there are no employees in that department:

| EmployeeName | DepartmentName |
| --- | --- |
| John Doe | HR |
| Jane Smith | IT |
| Anne Brown | HR |
| NULL | Marketing |

1. **Full Join**:

sqlCopy code

SELECT Employees.EmployeeName, Departments.DepartmentName

FROM Employees

FULL JOIN Departments ON Employees.DepartmentID = Departments.DepartmentID;

This will show all employees and all departments, with NULLs where there is no match.

### Performing a Cross Join:

When we perform a Cross Join between these two tables, it combines each row from the Employees table with each row from the Departments table.

The SQL for a Cross Join is:

sqlCopy code

SELECT Employees.EmployeeName, Departments.DepartmentName

FROM Employees

CROSS JOIN Departments;

### Result of the Cross Join:

For our tables, the result would look like this:

| EmployeeName | DepartmentName |
| --- | --- |
| John Doe | HR |
| John Doe | IT |
| John Doe | Marketing |
| Jane Smith | HR |
| Jane Smith | IT |
| Jane Smith | Marketing |
| Anne Brown | HR |
| Anne Brown | IT |
| Anne Brown | Marketing |

### Explanation:

* **John Doe** is paired with **HR**, **IT**, and **Marketing** (3 rows).
* **Jane Smith** is also paired with **HR**, **IT**, and **Marketing** (another 3 rows).
* **Anne Brown** follows the same pattern (another 3 rows).

In total, there are 9 rows (3 employees × 3 departments). The result is a comprehensive combination of every possible pair of rows from the two tables.

### Key Points:

* Cross Join does not require a condition (like ON clause in other joins).
* It can result in a very large number of rows if the tables are large.
* It's not commonly used in standard database operations as it often produces more data than needed. However, it's useful in certain scenarios where you need to combine every possible pair of rows between two sets of data.

ChatGPT

The Cartesian product, in the context of set theory and mathematics, refers to the product of multiple sets. It's a way of combining all elements from one set with all elements of another set, resulting in pairs (or tuples if there are more than two sets) that represent all possible combinations of elements from the sets.

### Basic Concept:

If you have two sets, Set A and Set B, the Cartesian product of these two sets is the set of all possible ordered pairs where the first element is from Set A and the second element is from Set B.

### Example:

Let's take two simple sets to illustrate:

* Set A = {1, 2}
* Set B = {a, b}

The Cartesian product of Set A and Set B (denoted as A × B) is:

* A × B = {(1, a), (1, b), (2, a), (2, b)}

Here, each element of Set A is paired with each element of Set B, resulting in 4 ordered pairs.