**A trigger** in MySQL is a set of SQL statements that reside in a system catalog. It is a special type of stored procedure that is invoked automatically in response to an event. Each trigger is associated with a table, which is activated on any DML statement such as INSERT, UPDATE, or DELETE.

A trigger is called a special procedure because it cannot be called directly like a stored procedure. The main difference between the trigger and procedure is that a trigger is called automatically when a data modification event is made against a table. In contrast, a stored procedure must be called explicitly.

Generally, triggers are of two types according to the SQL standard: row-level triggers and statement-level triggers.

**Row-Level Trigger:** It is a trigger, which is activated for each row by a triggering statement such as insert, update, or delete. For example, if a table has inserted, updated, or deleted multiple rows, the row trigger is fired automatically for each row affected by the insert, update, or delete statement.

**Why we need/use triggers in MySQL?**

**We need/use triggers in MySQL due to the following features:**

Triggers help us to enforce business rules.

Triggers help us to validate data even before they are inserted or updated.

Triggers help us to keep a log of records like maintaining audit trails in tables.

SQL triggers provide an alternative way to check the integrity of data.

Triggers provide an alternative way to run the scheduled task.

Triggers increases the performance of SQL queries because it does not need to compile each time the query is executed.

Triggers reduce the client-side code that saves time and effort.

Triggers help us to scale our application across different platforms.

Triggers are easy to maintain.

**Limitations of Using Triggers in MySQL**

MySQL triggers do not allow to use of all validations; they only provide extended validations. For example, we can use the NOT NULL, UNIQUE, CHECK and FOREIGN KEY constraints for simple validations.

Triggers are invoked and executed invisibly from the client application. Therefore, it isn't easy to troubleshoot what happens in the database layer.

Triggers may increase the overhead of the database server.

**Types of Triggers in MySQL?**

We can define the maximum six types of actions or events in the form of triggers:

**Before Insert:** It is activated before the insertion of data into the table.

**After Insert:** It is activated after the insertion of data into the table.

**Before Update:** It is activated before the update of data in the table.

**After Update:** It is activated after the update of the data in the table.

**Before Delete:** It is activated before the data is removed from the table.

**After Delete:** It is activated after the deletion of data from the table.

When we use a statement that does not use INSERT, UPDATE or DELETE query to change the data in a table, the triggers associated with the trigger will not be invoked.

**Trigger Activation Time: BEFORE | AFTER**

**Trigger Event: INSERT | UPDATE | DELETE**

**Syntax**

CREATE TRIGGER trigger\_name

(AFTER | BEFORE) (INSERT | UPDATE | DELETE)

ON table\_name FOR EACH ROW

BEGIN

--variable declarations

--trigger code

END;

CREATE TRIGGER trigger\_name trigger\_time trigger\_event

ON table\_name FOR EACH ROW

BEGIN

--variable declarations

--trigger code

END;

**Parameter Explanation**

The create trigger syntax contains the following parameters:

**trigger\_name:** It is the name of the trigger that we want to create. It must be written after the CREATE TRIGGER statement. It is to make sure that the trigger name should be unique within the schema.

**trigger\_time:** It is the trigger action time, which should be either BEFORE or AFTER. It is the required parameter while defining a trigger. It indicates that the trigger will be invoked before or after each row modification occurs on the table.

**trigger\_event:** It is the type of operation name that activates the trigger. It can be either INSERT, UPDATE, or DELETE operation. The trigger can invoke only one event at one time. If we want to define a trigger which is invoked by multiple events, it is required to define multiple triggers, and one for each event.

**table\_name:** It is the name of the table to which the trigger is associated. It must be written after the ON keyword. If we did not specify the table name, a trigger would not exist.

**BEGIN END Block:** Finally, we will specify the statement for execution when the trigger is activated. If we want to execute multiple statements, we will use the BEGIN END block that contains a set of queries to define the logic for the trigger.

The trigger body can access the column's values, which are affected by the DML statement. The NEW and OLD modifiers are used to distinguish the column values BEFORE and AFTER the execution of the DML statement. We can use the column name with NEW and OLD modifiers as OLD.col\_name and NEW.col\_name. The OLD.column\_name indicates the column of an existing row before the updation or deletion occurs. NEW.col\_name indicates the column of a new row that will be inserted or an existing row after it is updated.

For example, suppose we want to update the column name message\_info using the trigger. In the trigger body, we can access the column value before the update as OLD.message\_info and the new value NEW.message\_info.

Example:

CREATE TABLE employee(

name varchar(45) NOT NULL,

occupation varchar(35) NOT NULL,

working\_date date,

working\_hours varchar(10)

);

INSERT INTO employee VALUES

('Robin', 'Scientist', '2020-10-04', 12),

('Warner', 'Engineer', '2020-10-04', 10),

('Peter', 'Actor', '2020-10-04', 13),

('Marco', 'Doctor', '2020-10-04', 14),

('Brayden', 'Teacher', '2020-10-04', 12),

('Antonio', 'Business', '2020-10-04', 11);

mysql> DELIMITER //

mysql> Create Trigger before\_insert\_empworkinghours

BEFORE INSERT ON employee FOR EACH ROW

BEGIN

IF NEW.working\_hours < 0 THEN SET NEW.working\_hours = 0;

END IF;

END //

insert the nehative values

INSERT INTO employee VALUES

('Alexander', 'Actor', '2020-10-012', -13);

INSERT INTO employee VALUES

('Alexander', 'Actor', '2020-10-012', -1);

INSERT INTO sales\_info(product, quantity, fiscalYear)

VALUES

('2003 Maruti Suzuki',110, 2020),

('2015 Avenger', 120,2020),

('2018 Honda Shine', 150,2020),

('2014 Apache', 150,2020);

CREATE TRIGGER before\_update\_salesInfo

BEFORE UPDATE

ON sales\_info FOR EACH ROW

BEGIN

DECLARE error\_msg VARCHAR(255);

SET error\_msg = ('The new quantity cannot be greater than 2 times the current quantity');

IF new.quantity > old.quantity \* 2 THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = error\_msg;

END IF;

END $$

DELIMITER ;

**How to Create Index in MySQL**

An index is a data structure that allows us to add indexes in the existing table. It enables you to improve the faster retrieval of records on a database table. It creates an entry for each value of the indexed columns. We use it to quickly find the record without searching each row in a database table whenever the table is accessed. We can create an index by using one or more columns of the table for efficient access to the records.

When a table is created with a primary key or unique key, it automatically creates a special index named PRIMARY. We called this index as a clustered index. All indexes other than PRIMARY indexes are known as a non-clustered index or secondary index.

**Need for Indexing in MySQL**

Suppose we have a contact book that contains names and mobile numbers of the user. In this contact book, we want to find the mobile number of Martin Williamson. If the contact book is an unordered format means the name of the contact book is not sorted alphabetically, we need to go over all pages and read every name until we will not find the desired name that we are looking for. This type of searching name is known as sequential searching.

To find the name and contact of the user from table contactbooks, generally, we used to execute the following query:

mysql> **SELECT mobile\_number FROM contactbooks WHERE first\_name = 'Martin' AND last\_name = '**Williamson**';**

This query is very simple and easy. Although it finds the phone number and name of the user fast, the database searches entire rows of the table until it will not find the rows that you want. Assume, the contactbooks table contains millions of rows, then, without an index, the data retrieval takes a lot of time to find the result. In that case, the database indexing plays an important role in returning the desired result and improves the overall performance of the query.

**MySQL CREATE INDEX Statement**

Generally, we create an index at the time of table creation in the database. The following statement creates a table with an index that contains two columns col2 and col3.

mysql> CREATE TABLE t\_index(

col1 INT PRIMARY KEY,

col2 INT NOT NULL,

col3 INT NOT NULL,

col4 VARCHAR(20),

INDEX (col2,col3)

);

**If we want to add index in table, we will use the CREATE INDEX statement as follows:**

mysql> CREATE INDEX [index\_name] ON [table\_name] (column names)

In this statement, index\_name is the name of the index, table\_name is the name of the table to which the index belongs, and the column\_names is the list of columns.

**Let us add the new index for the column col4, we use the following statement:**

mysql> CREATE INDEX ind\_1 ON t\_index(col4);

Create one Student table :

Create Table Student(

studentid int NOT NULL,

firstname varchar(20) NOT NULL,

lastname varchar(20),

class varchar(20),

age int);

insert into student values(101,"Preety","Singh","CS",34);

insert into student values(102,"Mark","Mark","IT",37);

SELECT studentid, firstname, lastname FROM student WHERE class = 'CS';

**MySQL SEQUENCE**

A sequence in MySQL is an arrangement of integers generated in the ascending order (1, 2, 3, and so on) on specific demand. Sequences are used in the databases to generate unique numbers. Many applications require each row of a table to contain a distinct value, such as student roll number in student\_table, employee numbers in HR, customer ID in CRM, etc. To fulfill this type of arrangement, we use sequences that provide an easy way to generate them.

MySQL does not provide any built-in function to create a sequence for a table's rows or columns. But we can generate it via SQL query. In this article, we are going to describe how to create a sequence in MySQL using SQL query.

**Create Sequence Using AUTO\_INCREMENT**

The simplest way for creating a sequence in MySQL is by defining the column as AUTO\_INCREMENT during table creation, which should be a primary key column.

The following are the rules which should be considered when we use the AUTO\_INCREMENT attribute for the column:

We can create only one AUTO\_INCREMENT column in each table, and the data type of this column is an integer.

The AUTO\_INCREMENT column should also have either PRIMARY or UNIQUE KEY indexing.

The AUTO\_INCREMENT column must contain NOT NULL However, MySQL automatically adds the NOT NULL constraint to the column implicitly when we set the column as an AUTO\_INCREMENT attribute.

**Example:**

Let us understand it with the help of the following example. First, we need to create a new table and make sure that there is one column with the AUTO\_INCREMENT attribute and that too, as PRIMARY KEY.

Execute the below query to create a table: .

mysql> CREATE TABLE Insects (

Id INT UNSIGNED NOT NULL AUTO\_INCREMENT,

PRIMARY KEY (id),

Name VARCHAR(30) NOT NULL,

Type VARCHAR(30) NOT NULL,

Origin VARCHAR(30) NOT NULL

);

Next, we will insert a few rows into this table where no need to provide the id for each row because it is auto-incremented by MySQL.

mysql> INSERT INTO Insects (Name, Type, Origin) VALUES

('Cockroach', 'Crawling', 'Kitchen'),

('Mosquito', 'Flying', 'Driveway'),

('Spider' ,'Crawling', 'Court yard'),

('Grasshopper', 'Flying', 'Front yard');

**Now execute the SELECT statement to verify the records:**

mysql> SELECT \* FROM Insects;

INSERT INTO Insects (Id, Name, Type, Origin) VALUES

(6, 'Bee', 'Flying', 'Court yard'),

(7, 'Ant', 'Crawling', 'Front yard');

**Creating the user and granting the permission to the user:**

create user preety@localhost identified by 'password';

GRANT ALL PRIVILEGES ON \* . \* TO preety@localhost;

GRANT CREATE, SELECT, INSERT ON \* . \* TO preety@localhost;

**To View All the users in Mysql**

select user from mysql.user;