**Triggers----**

MySQL Trigger

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](https://www.javatpoint.com/sql-tutorial) 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](https://www.javatpoint.com/mysql-insert), [update](https://www.javatpoint.com/mysql-update), or [delete statement](https://www.javatpoint.com/mysql-delete).

**tatement-Level Trigger:** It is a trigger, which is fired once for each event that occurs on a table regardless of how many rows are inserted, updated, or deleted.

#### **NOTE: We should know that MySQL doesn't support statement-level triggers. It provides supports for row-level triggers only.**

### **Types of Triggers in MySQL?**

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

1. [**Before Insert**](https://www.javatpoint.com/mysql-before-insert-trigger)**:** It is activated before the insertion of data into the table.
2. [**After Insert**](https://www.javatpoint.com/mysql-after-insert-trigger)**:** It is activated after the insertion of data into the table.
3. [**Before Update**](https://www.javatpoint.com/mysql-before-update-trigger)**:** It is activated before the update of data in the table.
4. [**After Update**](https://www.javatpoint.com/mysql-after-update-trigger)**:** It is activated after the update of the data in the table.
5. [**Before Delete**](https://www.javatpoint.com/mysql-before-delete-trigger)**:** It is activated before the data is removed from the table.
6. [**After Delete**](https://www.javatpoint.com/mysql-after-delete-trigger)**:** It is activated after the deletion of data from the table.
7. **CREATE** **TRIGGER** trigger\_name
8. (**AFTER** | BEFORE) (**INSERT** | **UPDATE** | **DELETE**)
9. **ON** table\_name **FOR** EACH ROW
10. **BEGIN**
11. --variable declarations
12. --trigger code
13. **END**;

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.

We can understand the availability of OLD and NEW modifiers with the below table:

|  |  |  |
| --- | --- | --- |
| **Trigger Event** | **OLD** | **NEW** |
| INSERT | No | Yes |
| UPDATE | Yes | Yes |
| DELETE | Yes |  |

1. mysql> DELIMITER //
2. mysql> **Create** **Trigger** before\_insert\_empworkinghours
3. BEFORE **INSERT** **ON** employee **FOR** EACH ROW
4. **BEGIN**
5. IF NEW.working\_hours < 0 **THEN** **SET** NEW.working\_hours = 0;
6. **END** IF;
7. **END** //

**Show Triggers;**

**Drop trigger name;**

# MySQL Stored Procedure

A procedure (often called a stored procedure) is a **collection of pre-compiled SQL statements** stored inside the database. It is a subroutine or a subprogram in the regular computing language. **A procedure always contains a name, parameter lists, and SQL statements**.

### **How to create a procedure?**

1. DELIMITER &&
2. **CREATE** **PROCEDURE** procedure\_name [[IN | **OUT** | INOUT] parameter\_name datatype [, parameter datatype]) ]
3. **BEGIN**
4. Declaration\_section
5. Executable\_section
6. **END** &&
7. DELIMITER ;

### **Parameter Explanations**

The procedure syntax has the following parameters:

|  |  |
| --- | --- |
| **Parameter Name** | **Descriptions** |
| procedure\_name | It represents the name of the stored procedure. |
| parameter | It represents the number of parameters. It can be one or more than one. |
| Declaration\_section | It represents the declarations of all variables. |
| Executable\_section | It represents the code for the function execution. |

**MySQL procedure parameter has one of three modes:**

**IN parameter**

It is the default mode. It takes a parameter as input, such as an attribute. When we define it, the calling program has to pass an argument to the stored procedure. This parameter's value is always protected.

**OUT parameters**

It is used to pass a parameter as output. Its value can be changed inside the stored procedure, and the changed (new) value is passed back to the calling program. It is noted that a procedure cannot access the OUT parameter's initial value when it starts.

**INOUT parameters**

It is a combination of IN and OUT parameters. It means the calling program can pass the argument, and the procedure can modify the INOUT parameter, and then passes the new value back to the calling program.

### **How to call a stored procedure?**

We can use the **CALL statement** to call a stored procedure. This statement returns the values to its caller through its parameters (IN, OUT, or INOUT). The following syntax is used to call the stored procedure in MySQL:

1. CALL procedure\_name ( parameter(s))

### **Procedure without Parameter**

1. DELIMITER &&
2. **CREATE** **PROCEDURE** get\_merit\_student ()
3. **BEGIN**
4. **SELECT** \* **FROM** student\_info **WHERE** marks > 70;
5. **SELECT** COUNT(stud\_code) **AS** Total\_Student **FROM** student\_info;
6. **END** &&
7. DELIMITER ;

### **Procedures with IN Parameter**

1. DELIMITER &&
2. **CREATE** **PROCEDURE** get\_student (IN var1 **INT**)
3. **BEGIN**
4. **SELECT** \* **FROM** student\_info LIMIT var1;
5. **SELECT** COUNT(stud\_code) **AS** Total\_Student **FROM** student\_info;
6. **END** &&
7. DELIMITER ;

### **Procedures with OUT Parameter**

In this procedure, we have used the OUT parameter as the **'highestmark'** of integer type. Its body part fetches the maximum marks from the table using a **MAX() function**. See the procedure code:

1. DELIMITER &&
2. **CREATE** **PROCEDURE** display\_max\_mark (**OUT** highestmark **INT**)
3. **BEGIN**
4. **SELECT** **MAX**(marks) **INTO** highestmark **FROM** student\_info;
5. **END** &&
6. DELIMITER ;

mysql> CALL display\_max\_mark(@M);

mysql> **SELECT** @M;

### **Procedures with INOUT Parameter**

In this procedure, we have used the INOUT parameter as **'var1'** of integer type. Its body part first fetches the marks from the table with the specified **id** and then stores it into the same variable var1. The var1 first acts as the IN parameter and then OUT parameter. Therefore, we can call it the INOUT parameter mode. See the procedure code:

1. DELIMITER &&
2. **CREATE** **PROCEDURE** display\_marks (INOUT var1 **INT**)
3. **BEGIN**
4. **SELECT** marks **INTO** var1 **FROM** student\_info **WHERE** stud\_id = var1;
5. **END** &&
6. DELIMITER ;

After successful execution, we can call the procedure as follows:

1. mysql> **SET** @M = '3';
2. mysql> CALL display\_marks(@M);
3. mysql> **SELECT** @M;

### **How to show or list stored procedures in MySQL?**

1. mysql> SHOW **PROCEDURE** STATUS **WHERE** db = 'mystudentdb';

### **How to delete/drop stored procedures in MySQL?**

1. **DROP** **PROCEDURE** [ IF EXISTS ] procedure\_name;

# MySQL Stored Function

A stored function in MySQL is a set of SQL statements that perform some task/operation and return a single value. It is one of the types of stored programs in MySQL

he stored function is almost similar to the procedure in [MySQL](https://www.javatpoint.com/mysql-tutorial), but it has some differences that are as follows:

* The function parameter may contain only the **IN parameter** but can't allow specifying this parameter, while the procedure can allow **IN, OUT, INOUT parameters**.
* The stored function can return only a single value defined in the function header.
* The stored function may also be called within SQL statements.
* It may not produce a result set.
* DELIMITER $$
* **CREATE** **FUNCTION** fun\_name(fun\_parameter(s))
* **RETURNS** datatype
* [NOT] {Characteristics}
* fun\_body;

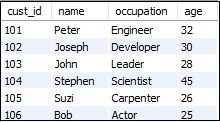
### **Parameter Used**

The stored function syntax uses the following parameters which are discussed below:

|  |  |
| --- | --- |
| **Parameter Name** | **Descriptions** |
| fun\_name | It is the name of the stored function that we want to create in a database. It should not be the same as the built-in function name of MySQL. |
| fun\_parameter | It contains the list of parameters used by the function body. It does not allow to specify IN, OUT, INOUT parameters. |
| datatype | It is a data type of return value of the function. It should any valid MySQL data type. |
| characteristics | The CREATE FUNCTION statement only accepted when the characteristics (DETERMINISTIC, NO SQL, or READS SQL DATA) are defined in the declaration. |
| fun\_body | This parameter has a set of SQL statements to perform the operations. It requires at least one RETURN statement. When the return statement is executed, the function will be terminated automatically. The function body is given below: BEGIN -- SQL statements END $$ DELIMITER |

## **MySQL Stored Function Example**

Let us understand how stored function works in MySQL through the example. Suppose our database has a table named **"customer"** that contains the following data:



Now, we will create a function that returns the **customer occupation** based on the **age** using the below statement.

1. DELIMITER $$
2. **CREATE** **FUNCTION** Customer\_Occupation(
3. age **int**
4. )
5. **RETURNS** **VARCHAR**(20)
6. DETERMINISTIC
7. **BEGIN**
8. **DECLARE** customer\_occupation **VARCHAR**(20);
9. IF age > 35 **THEN**
10. **SET** customer\_occupation = 'Scientist';
11. ELSEIF (age <= 35 AND
12. age >= 30) **THEN**
13. **SET** customer\_occupation = 'Engineer';
14. ELSEIF age < 30 **THEN**
15. **SET** customer\_occupation = 'Actor';
16. **END** IF;
17. -- return the customer occupation
18. **RETURN** (customer\_occupation);
19. **END**$$
20. DELIMITER;
21. SHOW **FUNCTION** STATUS **WHERE** db = 'mysqltestdb';

### **Stored Function Call**

Now, we are going to see how stored function is called with the SQL statement. The following statement uses **customer\_occupation** stored function to get the result:

1. **SELECT** **name**, age, Customer\_Occupation(age)
2. **FROM** customer **ORDER** **BY** age;

### **Stored Function Call in Procedure**

Here, we are going to see how this function can be called in a stored procedure. This statement creates a procedure in a database that uses **Customer\_Occupation()** stored function.

1. DELIMITER $$
2. **CREATE** **PROCEDURE** GetCustomerDetail()
3. **BEGIN**
4. **SELECT** **name**, age, Customer\_Occupation(age) **FROM** customer **ORDER** **BY** age;
5. **END**$$
6. DELIMITER ;

he below statement can be used to call the stored procedure:

1. CALL GetCustomerDetail();