Q 1 Difference Between TRUNCATE, DELETE, And DROP In SQL

The difference between TRUNCATE, DELETE, and DROP is one of the most common interview questions. Here are some of the common differences between them.

TRUNCATE

TRUNCATE SQL query removes all rows from a table, without logging the individual row deletions.

The following example removes all data from the Customers table.

1. **TRUNCATE** **TABLE** Customers;

* TRUNCATE is a DDL command
* TRUNCATE is executed using a table lock and whole table is locked for remove all records.
* We cannot use WHERE clause with TRUNCATE.
* TRUNCATE removes all rows from a table.
* Minimal logging in transaction log, so it is faster performance wise.
* TRUNCATE TABLE removes the data by deallocating the data pages used to store the table data and records only the page deallocations in the transaction log.
* Identify column is reset to its seed value if table contains any identity column.
* To use Truncate on a table you need at least ALTER permission on the table.
* Truncate uses less transaction space than the Delete statement.
* Truncate cannot be used with indexed views.
* TRUNCATE is faster than DELETE.

DELETE

To execute a DELETE query, delete permissions are required on the target table. If you need to use a WHERE clause in a DELETE, select permissions are required as well.

The following query deletes all rows from the Customers table.

1. **DELETE** **FROM** Customers;
2. GO

The following SQL query deletes all rows from the Customers table where OrderID is greater than 1000.

1. **DELETE** **FROM** Customers **WHERE** OrderId > 1000;
2. GO

* DELETE is a DML command.
* DELETE is executed using a row lock, each row in the table is locked for deletion.
* We can use where clause with DELETE to filter & delete specific records.
* The DELETE command is used to remove rows from a table based on WHERE condition.
* It maintain the log, so it slower than TRUNCATE.
* The DELETE statement removes rows one at a time and records an entry in the transaction log for each deleted row.
* Identity of column keep DELETE retains the identity.
* To use Delete you need DELETE permission on the table.
* Delete uses the more transaction space than Truncate statement.
* Delete can be used with indexed views.

DROP

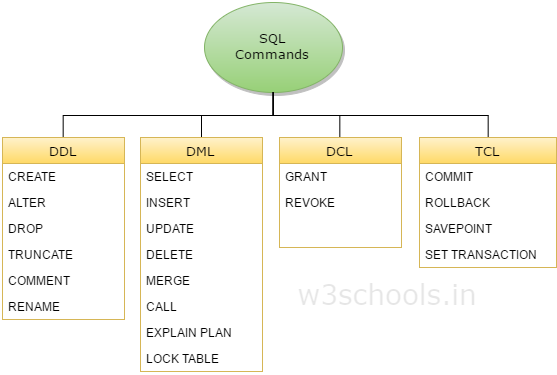
DROP table query removes one or more table definitions and all data, indexes, triggers, constraints, and permission specifications for those tables. DROP command requires ALTER permission on the schema to which the table belongs, CONTROL permission on the table, or membership in the db\_ddladmin fixed database role.

The following SQL query drops the Customers table and its data and indexes from the current database.

1. **DROP** **TABLE** Customers ;

* The DROP command removes a table from the database.
* All the tables' rows, indexes and privileges will also be removed.
* No DML triggers will be fired.
* The operation cannot be rolled back.
* DROP and TRUNCATE are DDL commands, whereas DELETE is a DML command.
* DELETE operations can be rolled back (undone), while DROP and TRUNCATE operations cannot be rolled back

Q 2 DDL, DML , DCL and TCL commands

[](https://i.stack.imgur.com/7uUaJ.png)

More information see here: [MySQL What is DDL, DML and DCL?](http://www.w3schools.in/mysql/ddl-dml-dcl/), the original is as follows:

**DDL**

DDL is short name of Data Definition Language, which deals with database schemas and descriptions, of how the data should reside in the database.

* CREATE – to create database and its objects like (table, index, views, store procedure, function and triggers)
* ALTER – alters the structure of the existing database
* DROP – delete objects from the database
* TRUNCATE – remove all records from a table, including all spaces allocated for the records are removed
* COMMENT – add comments to the data dictionary
* RENAME – rename an object

**DML**

DML is short name of Data Manipulation Language which deals with data manipulation, and includes most common SQL statements such SELECT, INSERT, UPDATE, DELETE etc, and it is used to store, modify, retrieve, delete and update data in database.

* SELECT – retrieve data from the a database
* INSERT – insert data into a table
* UPDATE – updates existing data within a table
* DELETE – Delete all records from a database table
* MERGE – UPSERT operation (insert or update)
* CALL – call a PL/SQL or Java subprogram
* EXPLAIN PLAN – interpretation of the data access path
* LOCK TABLE – concurrency Control

**DCL**

DCL is short name of Data Control Language which includes commands such as GRANT, and mostly concerned with rights, permissions and other controls of the database system.

* GRANT – allow users access privileges to database
* REVOKE – withdraw users access privileges given by using the GRANT command

**TCL**

TCL is short name of Transaction Control Language which deals with transaction within a database.

* COMMIT – commits a Transaction
* ROLLBACK – rollback a transaction in case of any error occurs
* SAVEPOINT – to rollback the transaction making points within groups
* SET TRANSACTION – specify characteristics for the transaction

Q 3 Difference between Primary Key and Unique Key

**Primary Key:**

* There can only be one primary key in a table
* In some DBMS it cannot be NULL - e.g. MySQL adds NOT NULL
* Primary Key is a unique key identifier of the record

**Unique Key:**

* Can be more than one unique key in one table
* Unique key can have NULL values
* It can be a candidate key
* Unique key can be NULL ; multiple rows can have NULL values and therefore may not be considered "unique"

**Q4 IN , Out , INOUT parameter in stored procedure**

[Home](http://www.mysqltutorial.org/) / [MySQL Stored Procedures](http://www.mysqltutorial.org/mysql-stored-procedure-tutorial.aspx) / MySQL Stored Procedure Parameters

# **MySQL Stored Procedure Parameters**

**Summary**: in this tutorial, you will learn how to create stored procedures with parameters including IN, OUT, and INTOUT parameters.

## Introduction to MySQL stored procedure parameters

Almost stored procedures that you develop require parameters. The parameters make the stored procedure more flexible and useful.

In MySQL, a parameter has one of three modes: IN,OUT, or INOUT.

### **IN parameters**

IN is the default mode. When you define an IN parameter in a stored procedure, the calling program has to pass an argument to the stored procedure. In addition, the value of an IN parameter is protected. It means that even the value of the IN parameter is changed inside the stored procedure, its original value is retained after the stored procedure ends. In other words, the stored procedure only works on the copy of the IN parameter.

### **OUT parameters**

The value of an OUT parameter can be changed inside the stored procedure and its new value is passed back to the calling program. Notice that the stored procedure cannot access the initial value of the OUT parameter when it starts.

### **INOUT parameters**

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

### **Defining a parameter**

Here is the basic syntax of defining a parameter in stored procedures:

|  |  |
| --- | --- |
| 1 | [IN | OUT | INOUT] parameter\_name datatype[(length)] |

In this syntax,

* First, specify the parameter mode, which can be IN , OUTor INOUT , depending on the purpose of the parameter in the stored procedure.
* Second, specify the name of the parameter. The parameter name must follow the naming rules of the column name in MySQL.
* Third, specify the data type and maximum length of the parameter.

## MySQL stored procedure parameter examples

Let’s take some examples of using stored procedure parameters.

### **The IN parameter example**

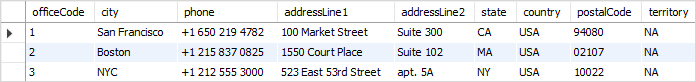
The following example creates a stored procedure that finds all offices that locate in a country specified by the input parameter countryName:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | DELIMITER //    CREATE PROCEDURE GetOfficeByCountry(      IN countryName VARCHAR(255)  )  BEGIN      SELECT \*      FROM offices      WHERE country = countryName;  END //    DELIMITER ; |

In this example, the countryName is the IN parameter of the stored procedure.

Suppose that you want to find offices locating in the USA, you need to pass an argument (USA) to the stored procedure as shown in the following query:

|  |  |
| --- | --- |
| 1 | CALL GetOfficeByCountry('USA'); |



To find offices in France, you pass the literal string France to the GetOfficeByCountry stored procedure as follows:

|  |  |
| --- | --- |
| 1 | CALL GetOfficeByCountry('France') |

http://www.mysqltutorial.org/wp-content/uploads/2009/12/MySQL-IN-parameter-offices-in-France.png

Because the countryName is the IN parameter, you must pass an argument. Fail to do so will result in an error:

|  |  |
| --- | --- |
| 1 | CALL GetOfficeByCountry(); |

Here is the error:

|  |  |
| --- | --- |
| 1 | Error Code: 1318. Incorrect number of arguments for PROCEDURE classicmodels.GetOfficeByCountry; expected 1, got 0 |

### **The OUT parameter example**

The following stored procedure returns the number of orders by order status.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | DELIMITER $$    CREATE PROCEDURE GetOrderCountByStatus (      IN  orderStatus VARCHAR(25),      OUT total INT  )  BEGIN      SELECT COUNT(orderNumber)      INTO total      FROM orders      WHERE status = orderStatus;  END$$    DELIMITER ; |

The stored procedure GetOrderCountByStatus() has two parameters:

* orderStatus : is the IN parameter specifies the status of orders to return.
* total : is the OUT parameter that stores the number of orders in a specific status.

To find the number of orders that already shipped, you call GetOrderCountByStatus  and pass the order status as of Shipped, and also pass a session variable ( @total ) to receive the return value.

|  |  |
| --- | --- |
| 1  2 | CALL GetOrderCountByStatus('Shipped',@total);  SELECT @total; |

http://www.mysqltutorial.org/wp-content/uploads/2009/12/MySQL-OUT-parameter-order-shipped.png

To get the number of orders that are in-process, you call the stored procedure GetOrderCountByStatus as follows:

|  |  |
| --- | --- |
| 1  2 | CALL GetOrderCountByStatus('in process',@total);  SELECT @total AS  total\_in\_process; |

http://www.mysqltutorial.org/wp-content/uploads/2009/12/MySQL-OUT-parameter-orders-in-process.png

### **The INOUT parameter example**

The following example demonstrates how to use an INOUT parameter in the stored procedure.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | DELIMITER $$    CREATE PROCEDURE SetCounter(      INOUT counter INT,      IN inc INT  )  BEGIN      SET counter = counter + inc;  END$$    DELIMITER ; |

In this example, the stored procedure SetCounter()  accepts one INOUT  parameter ( counter ) and one IN parameter ( inc ). It increases the counter ( counter ) by the value of specified by the inc parameter.

These statements illustrate how to call the SetSounter  stored procedure:

|  |  |
| --- | --- |
| 1  2  3  4  5 | SET @counter = 1;  CALL SetCounter(@counter,1); -- 2  CALL SetCounter(@counter,1); -- 3  CALL SetCounter(@counter,5); -- 8  SELECT @counter; -- 8 |

Here is the output:

http://www.mysqltutorial.org/wp-content/uploads/2019/09/MySQL-Stored-Procedure-Parameter-INOUT.png

In this tutorial, you have learned how create stored procedures with parameters including IN, OUT, and INOUT parameters.

[Home](http://www.mysqltutorial.org/) / [MySQL Stored Procedures](http://www.mysqltutorial.org/mysql-stored-procedure-tutorial.aspx) / MySQL Stored Procedures That Return Multiple Values

# **MySQL Stored Procedures That Return Multiple Values**

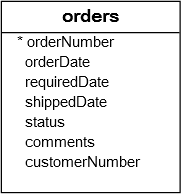
**Summary**: in this tutorial, you will learn how to develop stored procedures that return multiple values.

[MySQL stored function](http://www.mysqltutorial.org/mysql-stored-function/) returns only one value. To develop stored programs that return multiple values, you need to use stored procedures with INOUT or OUT parameters.

If you are not familiar with INOUT or OUT parameters, check it out the [stored procedure’s parameters tutorial](http://www.mysqltutorial.org/stored-procedures-parameters.aspx) for the detailed information.

## Stored procedures that return multiple values example

Let’s take a look at the orders table in the [sample database](http://wwww.mysqltutorial.org/mysql-sample-database.aspx).



The following stored procedure accepts customer number and returns the total number of orders that were shipped, canceled, resolved, and disputed.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46 | DELIMITER $$    CREATE PROCEDURE get\_order\_by\_cust(      IN cust\_no INT,      OUT shipped INT,      OUT canceled INT,      OUT resolved INT,      OUT disputed INT)  BEGIN          -- shipped          SELECT              count(\*) INTO shipped          FROM              orders          WHERE              customerNumber = cust\_no                  AND status = 'Shipped';            -- canceled          SELECT              count(\*) INTO canceled          FROM              orders          WHERE              customerNumber = cust\_no                  AND status = 'Canceled';            -- resolved          SELECT              count(\*) INTO resolved          FROM              orders          WHERE              customerNumber = cust\_no                  AND status = 'Resolved';            -- disputed          SELECT              count(\*) INTO disputed          FROM              orders          WHERE              customerNumber = cust\_no                  AND status = 'Disputed';    END |

In addition to the IN parameter, the stored procedure takes four additional OUT parameters: shipped, canceled, resolved, and disputed. Inside the stored procedure, you use a [SELECT](http://www.mysqltutorial.org/mysql-select-statement-query-data.aspx) statement with the [COUNT](http://www.mysqltutorial.org/mysql-count/) function to get the corresponding total of orders based on the order’s status and assign it to the respective parameter.

To use the get\_order\_by\_cust stored procedure, you pass customer number and four user-defined variables to get the out values.

After executing the stored procedure, you use the SELECT statement to output the variable values.

|  |  |
| --- | --- |
| 1  2 | CALL get\_order\_by\_cust(141,@shipped,@canceled,@resolved,@disputed);  SELECT @shipped,@canceled,@resolved,@disputed; |

MySQL Stored Procedures That Return Multiple Values

**Q5 Stored function in the sql**

[Home](http://www.mysqltutorial.org/) / [MySQL Stored Procedures](http://www.mysqltutorial.org/mysql-stored-procedure-tutorial.aspx) / MySQL Stored Function

# **MySQL Stored Function**

**Summary**: in this tutorial, you will learn how to create stored functions using the CREATE FUNCTION statement.

A stored function is a special kind stored program that returns a single value. Typically, you use stored functions to encapsulate common formulas or business rules that are reusable among SQL statements or stored programs.

Different from a [stored procedure](http://www.mysqltutorial.org/mysql-stored-procedure-tutorial.aspx), you can use a stored function in SQL statements wherever an expression is used. This helps improve the readability and maintainability of the procedural code.

To create a stored function, you use the CREATE FUNCTION statement.

## MySQL CREATE FUNCTION syntax

The following illustrates the basic syntax for creating a new stored function:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | DELIMITER $$    CREATE FUNCTION function\_name(      param1,      param2,…  )  RETURNS datatype  [NOT] DETERMINISTIC  BEGIN  -- statements  END $$    DELIMITER ; |

In this syntax:

First, specify the name of the stored function that you want to create after CREATE FUNCTION  keywords.

Second, list all [parameters](http://www.mysqltutorial.org/stored-procedures-parameters.aspx) of the stored function inside the parentheses followed by the function name. By default, all parameters are the IN parameters. You cannot specify IN , OUT or INOUT modifiers to parameters

Third, specify the data type of the return value in the RETURNS statement, which can be any valid [MySQL data types](http://www.mysqltutorial.org/mysql-data-types.aspx).

Fourth, specify if a function is deterministic or not using the DETERMINISTIC keyword.

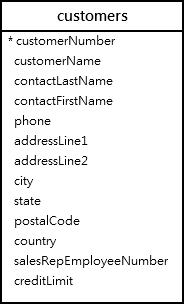
A deterministic function always returns the same result for the same input parameters whereas a non-deterministic function returns different results for the same input parameters.

If you don’t use DETERMINISTIC or NOT DETERMINISTIC, MySQL uses the NOT DETERMINISTIC option by default.

Fifth, write the code in the body of the stored function in the BEGIN END block. Inside the body section, you need to specify at least one RETURN statement. The RETURN statement returns a value to the calling programs. Whenever the RETURN statement is reached, the execution of the stored function is terminated immediately.

## MySQL CREATE FUNCTION example

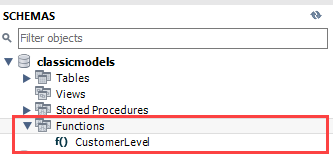
Let’s take the example of creating a stored function. We will use the customers table in the [sample database](http://www.mysqltutorial.org/mysql-sample-database.aspx) for the demonstration.



The following CREATE FUNCTION statement creates a function that returns the customer level based on credit:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | DELIMITER $$    CREATE FUNCTION CustomerLevel(      credit DECIMAL(10,2)  )  RETURNS VARCHAR(20)  DETERMINISTIC  BEGIN      DECLARE customerLevel VARCHAR(20);        IF credit > 50000 THEN          SET customerLevel = 'PLATINUM';      ELSEIF (credit >= 50000 AND              credit <= 10000) THEN          SET customerLevel = 'GOLD';      ELSEIF credit < 10000 THEN          SET customerLevel = 'SILVER';      END IF;      -- return the customer level      RETURN (customerLevel);  END$$  DELIMITER ; |

Once the function is created, you can view it in MySQL Workbench under the **Functions** section:



Or you can view all stored functions in the current classicmodels database by using the SHOW FUNCTION STATUS as follows:

|  |  |
| --- | --- |
| 1  2 | SHOW FUNCTION STATUS  WHERE db = 'classicmodels'; |

mysql stored function - show function status

### **Calling a stored function in an SQL statement**

The following statement uses the CustomerLevel stored function:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | SELECT      customerName,      CustomerLevel(creditLimit)  FROM      customers  ORDER BY      customerName; |



### **Calling a stored function in a stored procedure**

The following statement [creates a new stored procedure](http://www.mysqltutorial.org/getting-started-with-mysql-stored-procedures.aspx) that calls the CustomerLevel() stored function:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | DELIMITER $$    CREATE PROCEDURE GetCustomerLevel(      IN  customerNo INT,      OUT customerLevel VARCHAR(20)  )  BEGIN        DECLARE credit DEC(10,2) DEFAULT 0;        -- get credit limit of a customer      SELECT          creditLimit      INTO credit      FROM customers      WHERE          customerNumber = customerNo;        -- call the function      SET customerLevel = CustomerLevel(credit);  END$$    DELIMITER ; |

The following illustrates how to call the GetCustomerLevel() stored procedure:

|  |  |
| --- | --- |
| 1  2 | CALL GetCustomerLevel(-131,@customerLevel);  SELECT @customerLevel; |

It’s important to notice that if a stored function contains SQL statements that query data from tables, then you should not use it in other SQL statements; otherwise, the stored function will slow down the speed of the query.

In this tutorial, you have learned how to create a stored function to encapsulate the common formula or business rules.

**Q6 Triggers in SQL.**

[Home](http://www.mysqltutorial.org/) / MySQL Triggers

MySQL Triggers

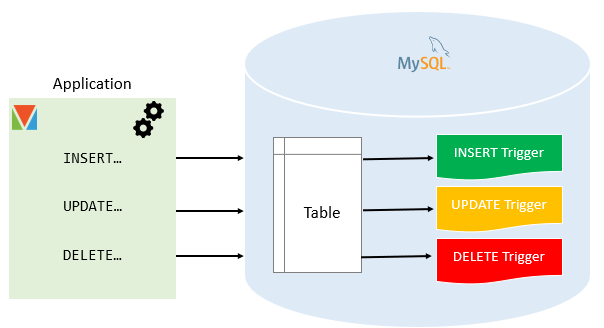
In MySQL, a trigger is a stored program invoked automatically in response to an event such as [insert](http://www.mysqltutorial.org/mysql-insert-statement.aspx), [update](http://www.mysqltutorial.org/mysql-update-data.aspx), or [delete](http://www.mysqltutorial.org/mysql-delete-statement.aspx) that occurs in the associated table. For example, you can define a trigger that is invoked automatically before a new row is inserted into a table.

MySQL supports triggers that are invoked in response to the [INSERT](http://www.mysqltutorial.org/mysql-insert-statement.aspx), [UPDATE](http://www.mysqltutorial.org/mysql-update-data.aspx) or [DELETE](http://www.mysqltutorial.org/mysql-delete-statement.aspx) event.

The SQL standard defines two types of triggers: row-level triggers and statement-level triggers.

* A row-level trigger is activated for each row that is inserted, updated, or deleted.  For example, if a table has 100 rows inserted, updated, or deleted, the trigger is automatically invoked 100 times for the 100 rows affected.
* A statement-level trigger is executed once for each transaction regardless of how many rows are inserted, updated, or deleted.

MySQL supports only row-level triggers. It doesn’t support statement-level triggers.



Advantages of triggers

* Triggers provide another way to check the integrity of data.
* Triggers handle errors from the database layer.
* Triggers give an alternative way to [run scheduled tasks](http://www.mysqltutorial.org/mysql-triggers/working-mysql-scheduled-event/). By using triggers, you don’t have to wait for the [scheduled events](http://www.mysqltutorial.org/mysql-triggers/working-mysql-scheduled-event/) to run because the triggers are invoked automatically *before* or *after* a change is made to the data in a table.
* Triggers can be useful for auditing the data changes in tables.

Disadvantages of triggers

* Triggers can only provide extended validations, not all validations. For simple validations, you can use the [NOT NULL](http://www.mysqltutorial.org/mysql-not-null-constraint/), [UNIQUE](http://www.mysqltutorial.org/mysql-unique-constraint/), [CHECK](http://www.mysqltutorial.org/mysql-check-constraint/) and [FOREIGN KEY](http://www.mysqltutorial.org/mysql-foreign-key/) constraints.
* Triggers can be difficult to troubleshoot because they execute automatically in the database, which may not invisible to the client applications.
* Triggers may increase the overhead of the MySQL Server.

Managing MySQL triggers

* [Create triggers](http://www.mysqltutorial.org/create-the-first-trigger-in-mysql.aspx)  – describe steps of how to create a trigger in MySQL.
* [Drop triggers](http://www.mysqltutorial.org/mysql-triggers/mysql-drop-trigger/) – show you how to drop a trigger.
* [Create a BEFORE INSERT trigger](http://www.mysqltutorial.org/mysql-triggers/mysql-before-insert-trigger/) – show you how to create a BEFORE INSERT trigger to maintain a summary table from another table.
* [Create an AFTER INSERT trigger](http://www.mysqltutorial.org/mysql-triggers/mysql-after-insert-trigger/) – describe how to create an AFTER INSERT trigger to insert data into a table after inserting data into another table.
* [Create a BEFORE UPDATE trigger](http://www.mysqltutorial.org/mysql-triggers/mysql-before-update-trigger/) – learn how to create a BEFORE UPDATE trigger that validates data before it is updated to the table.
* [Create an AFTER UPDATE trigger](http://www.mysqltutorial.org/mysql-triggers/mysql-after-update-trigger/) – show you how to create an AFTER UPDATE trigger to log the changes of data in a table.
* [Create a BEFORE DELETE trigger](http://www.mysqltutorial.org/mysql-triggers/mysql-before-delete-trigger/) – show how to create a BEFORE DELETE trigger.
* [Create an AFTER DELETE trigger](http://www.mysqltutorial.org/mysql-triggers/mysql-after-delete-trigger/) – describe how to create an AFTER DELETE trigger.
* [Create multiple triggers for a table that have the same trigger event and time](http://www.mysqltutorial.org/mysql-triggers/create-multiple-triggers-for-the-same-trigger-event-and-action-time/) – MySQL 8.0 allows you to define multiple triggers for a table that have the same trigger event and time.
* [Show triggers](http://www.mysqltutorial.org/mysql-triggers/mysql-show-triggers/) – list triggers in a database, table by specific patterns.

# **Create Trigger in MySQL**

**Summary**: in this tutorial, you will learn how to use the MySQL CREATE TRIGGER statement to create a trigger in the database**.**

## Introduction to MySQL CREATE TRIGGER statement

The CREATE TRIGGER statement creates a new trigger. Here is the basic syntax of the CREATE TRIGGER statement:

|  |  |
| --- | --- |
| 1  2  3  4 | CREATE TRIGGER trigger\_name  {BEFORE | AFTER} {INSERT | UPDATE| DELETE }  ON table\_name FOR EACH ROW  trigger\_body; |

In this syntax:

* First, specify the name of the trigger that you want to create after the CREATE TRIGGER keywords. Note that the trigger name must be unique within a database.
* Next, specify the trigger action time which can be either BEFORE or AFTER which indicates that the trigger is invoked before or after each row is modified.
* Then, specify the operation that activates the trigger, which can be [INSERT](http://www.mysqltutorial.org/mysql-insert-statement.aspx), [UPDATE](http://www.mysqltutorial.org/mysql-update-data.aspx), or [DELETE](http://www.mysqltutorial.org/mysql-delete-statement.aspx).
* After that, specify the name of the table to which the trigger belongs after the ON keyword.
* Finally, specify the statement to execute when the trigger activates. If you want to execute multiple statements, you use the BEGIN END compound statement.

The trigger body can access the values of the column being affected by the DML statement.

To distinguish between the value of the columns BEFORE and AFTER the DML has fired, you use the NEW and OLD modifiers.

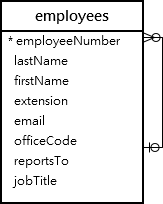
For example, if you update the column description, in the trigger body, you can access the value of the description before the update OLD.description and the new value NEW.description.

The following table illustrates the availability of the OLD and NEW modifiers:

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

## MySQL trigger examples

Let’s start creating a trigger in MySQL to log the changes of the employees table.



First, [create a new table](http://www.mysqltutorial.org/mysql-create-table/) named employees\_audit to keep the changes to the employees table:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | CREATE TABLE employees\_audit (      id INT AUTO\_INCREMENT PRIMARY KEY,      employeeNumber INT NOT NULL,      lastname VARCHAR(50) NOT NULL,      changedat DATETIME DEFAULT NULL,      action VARCHAR(50) DEFAULT NULL  ); |

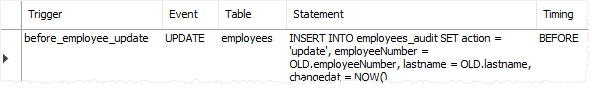
Next, create a BEFORE UPDATE trigger that is invoked before a change is made to the employees table.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | CREATE TRIGGER before\_employee\_update      BEFORE UPDATE ON employees      FOR EACH ROW  INSERT INTO employees\_audit  SET action = 'update',       employeeNumber = OLD.employeeNumber,       lastname = OLD.lastname,       changedat = NOW(); |

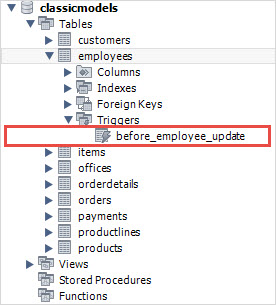
Inside the body of the trigger, we used the OLD keyword to access values of the columns employeeNumber and lastname of the row affected by the trigger.

Then, show all triggers in the current database by using the SHOW TRIGGERS statement:

|  |  |
| --- | --- |
| 1 | SHOW TRIGGERS; |



In addition, if you look at the schema using MySQL Workbench under the **employees > triggers**, you will see the before\_employee\_update trigger as shown in the screenshot below:



After that, update a row in the employees table:

|  |  |
| --- | --- |
| 1  2  3  4  5 | UPDATE employees  SET      lastName = 'Phan'  WHERE      employeeNumber = 1056; |

Finally, query the employees\_audit table to check if the trigger was fired by the UPDATE statement:

|  |  |
| --- | --- |
| 1 | SELECT \* FROM employees\_audit; |

The following shows the output of the query:

MySQL CREATE TRIGGER example

As you see clearly from the output, the trigger was automatically invoked and inserted a new row into the employees\_audit table.

In this tutorial, you have learned how to use the MySQL CREATE TRIGGER statement to create a new trigger in the database.

[Home](http://www.mysqltutorial.org/) / [MySQL Triggers](http://www.mysqltutorial.org/mysql-triggers.aspx) / MySQL DROP TRIGGER

# **MySQL DROP TRIGGER**

**Summary**: in this tutorial, you will learn how to use the MySQL DROP TRIGGER statement to drop a trigger from the database.

## Introduction to MySQL DROP TRIGGER statement

The DROP TRIGGER statement deletes a trigger from the database.

Here is the basic syntax of the DROP TRIGGER statement:

|  |  |
| --- | --- |
| 1 | DROP TRIGGER [IF EXISTS] [schema\_name.]trigger\_name; |

In this syntax:

* First, specify the name of the trigger that you want to drop after the DROP TRIGGER keywords.
* Second, specify the name of the schema to which the trigger belongs. If you skip the schema name, the statement will drop the trigger in the current database.
* Third, use IF EXISTS option to conditionally drops the trigger if the trigger exists. The IF EXISTS clause is optional.

If you drop a trigger that does not exist without using the IF EXISTS clause, MySQL issues an error. However, if you use the IF EXISTS clause, MySQL issues a NOTE instead.

The DROP TRIGGER requires the TRIGGER privilege for the table associated with the trigger.

Note that if you [drop a table](http://www.mysqltutorial.org/mysql-drop-table), MySQL will automatically drop all triggers associated with the table.

## MySQL DROP TRIGGER example

First, [create a table](http://www.mysqltutorial.org/mysql-create-table/) called billings for demonstration:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | CREATE TABLE billings (      billingNo INT AUTO\_INCREMENT,      customerNo INT,      billingDate DATE,      amount DEC(10 , 2 ),      PRIMARY KEY (billingNo)  ); |

Second, [create a new trigger](http://www.mysqltutorial.org/create-the-first-trigger-in-mysql.aspx) called BEFORE UPDATE that is associated with the billings table:

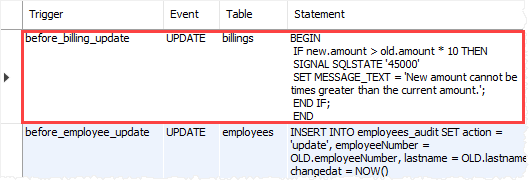
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | DELIMITER $$  CREATE TRIGGER before\_billing\_update      BEFORE UPDATE      ON billings FOR EACH ROW  BEGIN      IF new.amount > old.amount \* 10 THEN          SIGNAL SQLSTATE '45000'              SET MESSAGE\_TEXT = 'New amount cannot be 10 times greater than the current amount.';      END IF;  END$$  DELIMITER ; |

If you are not familiar with the DELIMITER statement, check it out here in the [stored procedure statement](http://www.mysqltutorial.org/getting-started-with-mysql-stored-procedures.aspx).

The trigger activates before any update. If the new amount is 10 times greater than the current amount, the trigger raises an error.

Third, show the triggers:

|  |  |
| --- | --- |
| 1 | SHOW TRIGGERS; |

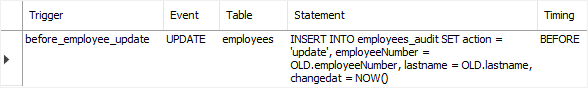


Fourth, drop the before\_billing\_update trigger:

|  |  |
| --- | --- |
| 1 | DROP TRIGGER before\_billing\_update; |

Finally, show the triggers again to verify the removal:

|  |  |
| --- | --- |
| 1 | SHOW TRIGGERS; |



[Home](http://www.mysqltutorial.org/) / [MySQL Triggers](http://www.mysqltutorial.org/mysql-triggers.aspx) / MySQL BEFORE INSERT Trigger

# **MySQL BEFORE INSERT Trigger**

**Summary**: in this tutorial, you will learn how to create a MySQL BEFORE INSERT trigger to maintain a summary table of another table.

## Introduction to MySQL BEFORE INSERT triggers

MySQL BEFORE INSERT [triggers](http://www.mysqltutorial.org/mysql-triggers.aspx) are automatically fired before an [insert](http://www.mysqltutorial.org/mysql-insert-statement.aspx) event occurs on the table.

The following illustrates the basic syntax of creating a MySQL BEFORE INSERT trigger:

|  |  |
| --- | --- |
| 1  2  3  4 | CREATE TRIGGER trigger\_name      BEFORE INSERT      ON table\_name FOR EACH ROW  trigger\_body; |

In this syntax:

First, specify the name of the trigger that you want to create in the [CREATE TRIGGER](http://www.mysqltutorial.org/create-the-first-trigger-in-mysql.aspx) clause.

Second, use BEFORE INSERT clause to specify the time to invoke the trigger.

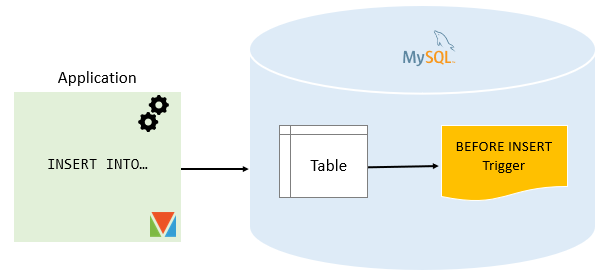
Third, specify the name of the table that the trigger is associated with after the ON keyword.

Finally, specify the trigger body which contains one or more SQL statements that execute when the trigger is invoked.

If you have multiple statements in the trigger\_body, you have to use the BEGIN END block and change the default [delimiter](http://www.mysqltutorial.org/mysql-stored-procedure/mysql-delimiter/):

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | DELIMITER $$    CREATE TRIGGER trigger\_name      BEFORE INSERT      ON table\_name FOR EACH ROW  BEGIN      -- statements  END$$    DELIMITER ; |

Note that in a BEFORE INSERT trigger, you can access and change the NEW values. However, you cannot access the OLD values because OLD values obviously do not exist.



## MySQL BEFORE INSERT trigger example

We will create a BEFORE INSERT trigger to maintain a summary table from another table.

### **Setting up a sample table**

First, [create a new table](http://www.mysqltutorial.org/mysql-create-table/) called WorkCenters:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | DROP TABLE IF EXISTS WorkCenters;    CREATE TABLE WorkCenters (      id INT AUTO\_INCREMENT PRIMARY KEY,      name VARCHAR(100) NOT NULL,      capacity INT NOT NULL  ); |

Second, create another table called WorkCenterStats that stores the summary of the capacity of the work centers:

|  |  |
| --- | --- |
| 1  2  3  4  5 | DROP TABLE IF EXISTS WorkCenterStats;    CREATE TABLE WorkCenterStats(      totalCapacity INT NOT NULL  ); |

### **Creating BEFORE INSERT trigger example**

The following trigger updates the total capacity in the WorkCenterStats table before a new work center is inserted into the WorkCenter table:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | DELIMITER $$    CREATE TRIGGER before\_workcenters\_insert  BEFORE INSERT  ON WorkCenters FOR EACH ROW  BEGIN      DECLARE rowcount INT;        SELECT COUNT(\*)      INTO rowcount      FROM WorkCenterStats;        IF rowcount > 0 THEN          UPDATE WorkCenterStats          SET totalCapacity = totalCapacity + new.capacity;      ELSE          INSERT INTO WorkCenterStats(totalCapacity)          VALUES(new.capacity);      END IF;    END $$    DELIMITER ; |

In this trigger:

First, the name of the trigger is before\_workcenters\_insert specified in the CREATE TRIGGER clause:

|  |  |
| --- | --- |
| 1 | CREATE TRIGGER before\_workcenters\_insert |

Second, the triggering event is:

|  |  |
| --- | --- |
| 1 | BEFORE INSERT |

Third, the table that the trigger associated with is WorkCenters table:

|  |  |
| --- | --- |
| 1 | ON WorkCenters FOR EACH ROW |

Finally, inside the trigger body, we check if there is any row in the WorkCenterStats table.

If the table WorkCenterStats has a row, the trigger adds the capacity to the totalCapacity column. Otherwise, it inserts a new row into the WorkCenterStats table.

### **Testing the MySQL BEFORE INSERT trigger**

First, [insert a new row](http://www.mysqltutorial.org/mysql-insert-statement.aspx) into the WorkCenter table:

|  |  |
| --- | --- |
| 1  2 | INSERT INTO WorkCenters(name, capacity)  VALUES('Mold Machine',100); |

Second, [query data](http://www.mysqltutorial.org/mysql-select-statement-query-data.aspx) from the WorkCenterStats table:

|  |  |
| --- | --- |
| 1 | SELECT \* FROM WorkCenterStats; |

http://www.mysqltutorial.org/wp-content/uploads/2019/09/MySQL-BEFORE-INSERT-Trigger-Example.png

The trigger has been invoked and inserted a new row into the WorkCenterStats table.

Third, insert a new work center:

|  |  |
| --- | --- |
| 1  2 | INSERT INTO WorkCenters(name, capacity)  VALUES('Packing',200); |

Finally, query data from the WorkCenterStats:

|  |  |
| --- | --- |
| 1 | SELECT \* FROM WorkCenterStats; |

MySQL BEFORE INSERT Trigger Example 2

The trigger has updated the total capacity from 100 to 200 as expected.

Note that to properly maintain the summary table WorkCenterStats, you should also create triggers to handle update and delete events on the WorkCenters table.

**Q7 Difference between stored procedure and function**

