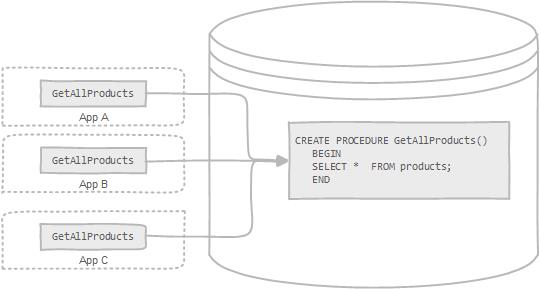
**MySQL PLSQL**

* MySQL Stored Procedures

A stored procedure is a segment of declarative SQL statements stored inside the database catalog. A stored procedure can be invoked by triggers, other stored procedures, and applications such as Java, Python, PHP.



## **Stored Procedures in MySQL**

MySQL is known as the most popular open source RDBMS which is widely used by both community and enterprise. However, during the first decade of its existence, it did not support stored procedures, [stored functions](http://mysqltutorial.org/mysql-stored-function/), [trigger](http://mysqltutorial.org/mysql-triggers.aspx)s, and [events](http://mysqltutorial.org/mysql-triggers/working-mysql-scheduled-event/). Since MySQL version 5.0, those features were added to MySQL database engine to make it more flexible and powerful.

## **MySQL stored procedures Advantages**

* Typically stored procedures help increase the performance of the applications. Once created, stored procedures are compiled and stored in the database. However, MySQL implements the stored procedures slightly different. MySQL stored procedures are compiled on demand. After compiling a stored procedure, MySQL puts it into a cache. And MySQL maintains its own stored procedure cache for every single connection. If an application uses a stored procedure multiple times in a single connection, the compiled version is used, otherwise, the stored procedure works like a query.
* Stored procedures help reduce the traffic between application and database server because instead of sending multiple lengthy SQL statements, the application has to send only name and parameters of the stored procedure.
* Stored procedures are reusable and transparent to any applications. Stored procedures expose the database interface to all applications so that developers don’t have to develop functions that are already supported in stored procedures.
* Stored procedures are secure. The database administrator can grant appropriate permissions to applications that access stored procedures in the database without giving any permissions on the underlying database tables.

Besides those advantages, stored procedures have their own disadvantages, which you should be aware of before using them in your databases.

## **MySQL stored procedures Disadvantages**

* If you use many stored procedures, the memory usage of every connection that is using those stored procedures will increase substantially. In addition, if you overuse a large number of logical operations inside store procedures, the CPU usage will also increase because the database server is not well-designed for logical operations.
* Stored procedure’s constructs are not designed for developing complex and flexible business logic.
* It is difficult to debug stored procedures. Only a few database management systems allow you to debug stored procedures. Unfortunately, MySQL does not provide facilities for debugging stored procedures.
* It is not easy to develop and maintain stored procedures. Developing and maintaining stored procedures are often required a specialized skill set that not all application developers possess. This may lead to problems in both application development and maintenance phases.
* **How To Create Stored Procedure :**

**DELIMITER //**

**CREATE PROCEDURE GetAllProducts()**

**BEGIN**

**SELECT \* FROM products;**

**END //**

**DELIMITER ;**

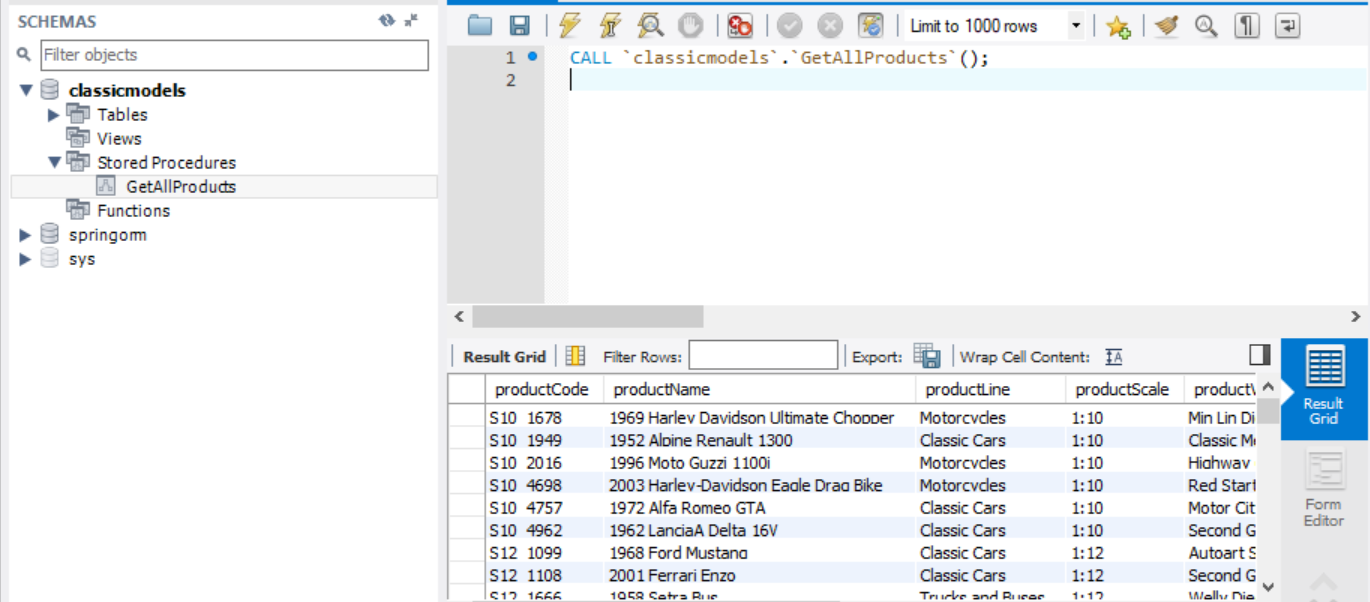
**Explanation :**

* The first command is DELIMITER // , which is not related to the stored procedure syntax. The DELIMITER statement changes the standard delimiter which is semicolon ( ; ) to another. In this case, the delimiter is changed from the semicolon( ; ) to double-slashes // Why do we have to change the delimiter? Because we want to pass the stored procedure to the server as a whole rather than letting mysql tool interpret each statement at a time.  Following the END keyword, we use the delimiter //  to indicate the end of the stored procedure. The last command ( DELIMITER; *)* changes the delimiter back to the semicolon (;).
* We use the CREATE PROCEDURE  statement to create a new stored procedure. We specify the name of stored procedure after the CREATE PROCEDURE  statement. In this case, the name of the stored procedure is GetAllProducts *.*We put the parentheses after the name of the stored procedure*.*
* The section between BEGIN and END  is called the body of the stored procedure. You put the declarative SQL statements in the body to handle business logic. In this stored procedure, we use a simple [SELECT](http://mysqltutorial.org/mysql-select-statement-query-data.aspx)statement to query data from the products table.

## **Calling stored procedures**

In order to call a stored procedure, you use the following SQL command:

CALL STORED\_PROCEDURE\_NAME();



## **Declaring variables**

To declare a variable inside a stored procedure, you use the DECLARE statement as follows:

DECLARE variable\_name datatype(size) DEFAULT default\_value;

* First, you specify the variable name after the DECLARE keyword. The variable name must follow the naming rules of MySQL table column names.
* Second, you specify the data type of the variable and its size. A variable can have any MySQL data types such as INT, VARCHAR , and DATETIME.
* Third, when you declare a variable, its initial value is NULL. You can assign the variable a default value using the DEFAULTkeyword.

**Example :**

DECLARE total\_sale INT DEFAULT 0;

MySQL allows you to declare two or more variables that share the same data type using a single DECLARE statement as following:

DECLARE x, y INT DEFAULT 0;

## **Assigning variables**

Once you declared a variable, you can start using it. To assign a variable another value, you use the SET statement, for example:

DECLARE total\_count INT DEFAULT 0;

SET total\_count = 10;

Besides the SET statement, you can use the SELECT INTO statement to assign the result of a query, which returns a scalar value, to a variable. See the following example:

DECLARE total\_products INT DEFAULT 0

SELECT COUNT(\*) INTO total\_products

FROM products

## **Variables scope**

A variable has its own scope that defines its lifetime. If you declare a variable inside a stored procedure, it will be out of scope when the END statement of stored procedure reached.

If you declare a variable inside BEGIN END block, it will be out of scope if the END is reached. You can declare two or more variables with the same name in different scopes because a variable is only effective in its own scope. However, declaring variables with the same name in different scopes is not good programming practice.

A variable that begins with the @ sign is session variable. It is available and accessible until the session ends.

## **Passing Parameters**

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

* **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** – 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** – an INOUT parameter is the 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.

The syntax of defining a parameter in the stored procedures is as follows:

* The MODE could be IN , OUT,,or INOUT , depending on the purpose of the parameter in the stored procedure.
* The param\_name is the name of the parameter. The name of the parameter must follow the naming rules of the column name in MySQL.
* Followed the parameter name is its data type and size. Like a variable, the data type of the parameter can be any valid MySQL data type.

Each parameter is separated by a comma (,) if the stored procedure has more than one parameter

### **IN parameter example**

DELIMITER //

CREATE PROCEDURE GetOfficeByCountry(IN countryName VARCHAR(255))

BEGIN

SELECT \*

FROM offices

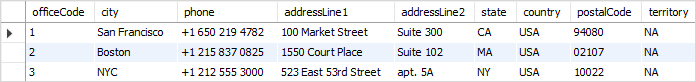
WHERE country = countryName;

END //

DELIMITER ;

The countryName is the IN parameter of the stored procedure. Inside the stored procedure, we select all offices that locate in the country specified by the countryName parameter

CALL GetOfficeByCountry('USA');



### **OUT parameter example**

DELIMITER $$

CREATE PROCEDURE CountOrderByStatus(

IN orderStatus VARCHAR(25),

OUT total INT)

BEGIN

SELECT count(orderNumber)

INTO total

FROM orders

WHERE status = orderStatus;

END$$

DELIMITER ;

To get the number of shipped orders, we call the CountOrderByStatus stored procedure and pass the order status as Shipped, and also pass an argument ( @total ) to get the return value.

CALL CountOrderByStatus('Shipped',@total);

SELECT @total;

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

* MySQL IF Statement

The MySQL IF statement allows you to execute a set of SQL statements based on a certain condition or value of an expression. To form an expression in MySQL, you can combine literals, [variables](http://mysqltutorial.org/variables-in-stored-procedures.aspx), operators, and even [functions](http://mysqltutorial.org/mysql-functions.aspx). An expression can return one of three values TRUE FALSE, or NULL.

**Syntax:**

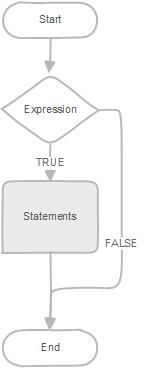
IF expression THEN

   statements;

END IF;

If the expression evaluates to TRUE , then the statements will be executed, otherwise, the control is passed to the next statement following the END IF.

The following flowchart demonstrates the IF statement:



* MySQL IF ELSE statement

In case you want to execute statements when the expression evaluates to FALSE , you use the IF ELSEstatement as follows:

**Syntax :**

IF expression THEN

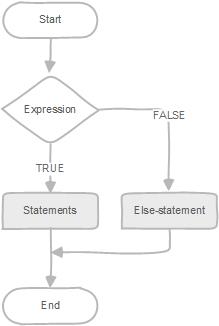
   statements;

ELSE

   else-statements;

END IF;

The following flowchart illustrates the IF ELSE statement:



* MySQL IF ELSEIF ELSE statement

If you want to execute statements conditionally based on multiple expressions, you use the IF ELSEIF ELSE statement as follows:

**Syntax:**

IF expression THEN

   statements;

ELSEIF elseif-expression THEN

   elseif-statements;

...

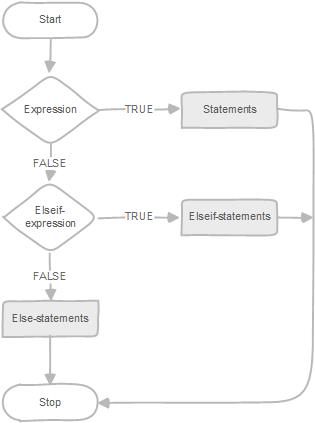
ELSE

   else-statements;

END IF;

If the expression evaluates to TRUE , the statements in the IF branch executes. If the expressionevaluates to FALSE , MySQL will check the *elseif-expression* and execute the elseif-statements in the ELSEIF branch if the *elseif\_expression* evaluates to TRUE .

The IF statement may have multiple ELSEIF branches to check multiple expressions. If no expression evaluates to TRUE, the else-statements in the ELSE branch will execute.



## **IF statement Example**

DELIMITER $$

CREATE PROCEDURE GetCustomerLevel(

in p\_customerNumber int(11),

out p\_customerLevel varchar(10))

BEGIN

DECLARE creditlim double;

SELECT creditlimit INTO creditlim

FROM customers

WHERE customerNumber = p\_customerNumber;

IF creditlim > 50000 THEN

SET p\_customerLevel = 'PLATINUM';

ELSEIF (creditlim <= 50000 AND creditlim >= 10000) THEN

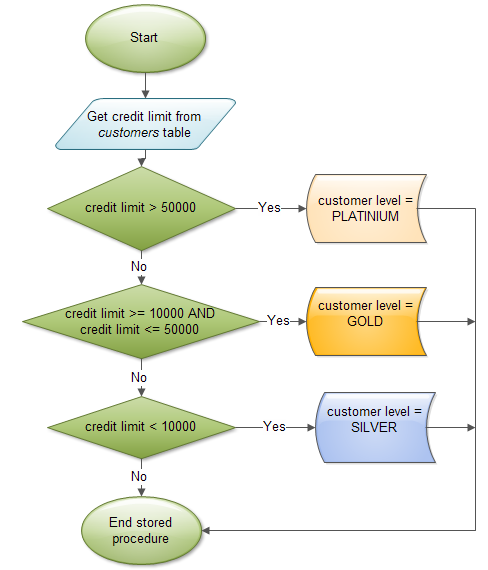
SET p\_customerLevel = 'GOLD';

ELSEIF creditlim < 10000 THEN

SET p\_customerLevel = 'SILVER';

END IF;

END$$



* MySQL Functions

A stored function is a special kind stored program that returns a single value. 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://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.

**Syntax:**

**CREATE FUNCTION function name(param1,param2,…)**

**RETURNS datatype**

**[NOT] DETERMINISTIC**

**statements**

First, you specify the name of the stored function after CREATE FUNCTION clause.

Second, you list all parameters of the stored function inside the parentheses. By default, all parameters are IN parameters. You cannot specify IN , OUT or INOUT modifiers to the parameters.

Third, you must specify the data type of the return value in the RETURNS statement. It can be any valid MySQL data types.

Fourth, for the same input parameters, if the stored function returns the same result, it is considered deterministic and otherwise the stored function is not deterministic. You have to decide whether a stored function is deterministic or not. If you declare it incorrectly, the stored function may produce an unexpected result, or the available optimization is not used which degrades the performance.

Fifth, you write the code in the body of the stored function. It can be a single statement or a compound statement. Inside the body section, you have to specify at least one RETURN statement. The RETURN statement returns a value to the caller. Whenever the RETURN statement is reached, the stored function’s execution is terminated immediately.

Example:

**DELIMITER $$**

**CREATE FUNCTION CustomerLevel(p\_creditLimit double) RETURNS VARCHAR(10)**

**DETERMINISTIC**

**BEGIN**

**DECLARE lvl varchar(10);**

**IF p\_creditLimit > 50000 THEN**

**SET lvl = 'PLATINUM';**

**ELSEIF (p\_creditLimit <= 50000 AND p\_creditLimit >= 10000) THEN**

**SET lvl = 'GOLD';**

**ELSEIF p\_creditLimit < 10000 THEN**

**SET lvl = 'SILVER';**

**END IF;**

**RETURN (lvl);**

**END**

Now, we can call the Customer Level() in an [SQL SELECT statement](http://mysqltutorial.org/mysql-select-statement-query-data.aspx) as follows:

**SELECT customerName, CustomerLevel(creditLimit)**

**FROM customers ORDER BY customerName;**

As you can see, the  GetCustomerLevel() stored procedure is much more readable when using the  CustomerLevel() stored function.

Notice that a stored function returns a single value only. If you include a SELECT statement without the INTO clause, you will get an error.

In addition, if a stored function contains SQL statements, you should not use it inside other SQL statements; otherwise, the stored function will slow down the speed of the query.

* MySQL Cursor:

To handle a result set inside a [stored procedure](http://mysqltutorial.org/mysql-stored-procedure-tutorial.aspx), you use a cursor. A cursor allows you to [iterate](http://mysqltutorial.org/stored-procedures-loop.aspx)a set of rows returned by a query and process each row accordingly.

MySQL cursor is read-only, non-scrollable and asensitive.

* **Read only**: you cannot update data in the underlying table through the cursor.
* **Non-scrollable**: you can only fetch rows in the order determined by the [SELECT statement](http://mysqltutorial.org/mysql-select-statement-query-data.aspx). You cannot fetch rows in the reversed order. In addition, you cannot skip rows or jump to a specific row in the result set.
* **Asensitive**: there are two kinds of cursors: asensitive cursor and insensitive cursor. An asensitive cursor points to the actual data, whereas an insensitive cursor uses a temporary copy of the data. An asensitive cursor performs faster than an insensitive cursor because it does not have to make a temporary copy of data. However, any change that made to the data from other connections will affect the data that is being used by an asensitive cursor, therefore, it is safer if you don’t update the data that is being used by an asensitive cursor. MySQL cursor is asensitive.

You can use MySQL cursors in [stored procedures](http://mysqltutorial.org/mysql-stored-procedure-tutorial.aspx), [stored functions](http://mysqltutorial.org/mysql-stored-function/), and [triggers](http://mysqltutorial.org/mysql-triggers.aspx).

* **Working with MySQL cursor**

First, you have to declare a cursor by using the DECLARE statement:

DECLARE cursor\_name CURSOR FOR SELECT\_statement;

The cursor declaration must be after any [variable](http://mysqltutorial.org/variables-in-stored-procedures.aspx)declaration. If you declare a cursor before variables declaration, MySQL will issue an error. A cursor must always be associated with a SELECT statement.

Next, you open the cursor by using the OPEN statement. The OPEN statement initializes the result set for the cursor, therefore, you must call the OPEN statement before fetching rows from the result set.

OPEN cursor\_name;

Then, you use the FETCH statement to retrieve the next row pointed by the cursor and move the cursor to the next row in the result set.

FETCH cursor\_name INTO variables list;

After that, you can check to see if there is any row available before fetching it.

Finally, you call the CLOSE statement to deactivate the cursor and release the memory associated with it as follows:

CLOSE cursor\_name;

When the cursor is no longer used, you should close it.

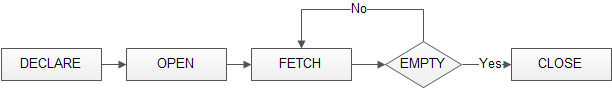
When working with MySQL cursor, you must also declare a NOT FOUND handler to handle the situation when the cursor could not find any row. Because each time you call the FETCH statement, the cursor attempts to read the next row in the result set. When the cursor reaches the end of the result set, it will not be able to get the data, and a condition is raised. The handler is used to handle this condition.

To declare a NOT FOUND handler, you use the following syntax:

DECLARE CONTINUE HANDLER FOR NOT FOUND SET finished = 1;

Where finished is a variable to indicate that the cursor has reached the end of the result set. Notice that the handler declaration must appear after variable and cursor declaration inside the stored procedures.

The following diagram illustrates how MySQL cursor works.



**Example :**

DELIMITER $$

 CREATE PROCEDURE build\_email\_list (INOUT email\_list varchar(4000))

BEGIN

DECLARE v\_finished INTEGER DEFAULT 0;

        DECLARE v\_email varchar(100) DEFAULT "";

 -- declare cursor for employee email

DEClARE email\_cursor CURSOR FOR

SELECT email FROM employees;

 -- declare NOT FOUND handler

DECLARE CONTINUE HANDLER

        FOR NOT FOUND SET v\_finished = 1;

 OPEN email\_cursor;

 get\_email: LOOP

 FETCH email\_cursor INTO v\_email;

 IF v\_finished = 1 THEN

LEAVE get\_email;

END IF;

 -- build email list

SET email\_list = CONCAT(v\_email,";",email\_list);

 END LOOP get\_email;

 CLOSE email\_cursor;

END$$

DELIMITER ;

You can test the build\_email\_list stored procedure using the following script:

**SET @email\_list = "";**

**CALL build\_email\_list(@email\_list);**

**SELECT @email\_list;**

* MySQL Trigger:

A SQL trigger is a set of SQL statements stored in the database catalog. A SQL trigger is executed or fired whenever an event associated with a table occurs e.g., insert, update or delete.

A SQL trigger is a special type of [stored procedure](http://mysqltutorial.org/mysql-stored-procedure-tutorial.aspx). It is special because it is not called directly like a stored procedure. The main difference between a trigger and a stored procedure is that a trigger is called automatically when a data modification event is made against a table whereas a stored procedure must be called explicitly.

It is important to understand SQL trigger’s advantages and disadvantages so that you can use it appropriately. In the following sections, we will discuss the advantages and disadvantages of using SQL triggers.

**Advantages of using SQL triggers**

* SQL triggers provide an alternative way to check the integrity of data.
* SQL triggers can catch errors in business logic in the database layer.
* SQL triggers provide an alternative way to [run scheduled tasks](http://mysqltutorial.org/mysql-triggers/working-mysql-scheduled-event/). By using SQL triggers, you don’t have to wait to run the scheduled tasks because the triggers are invoked  automatically *before* or *after* a change  is made to the data in the tables.
* SQL triggers are very useful to audit the changes of data in tables.

**Disadvantages of using SQL triggers**

* SQL triggers only can provide an extended validation and they cannot replace all the validations. Some simple validations have to be done in the application layer. For example, you can validate user’s inputs in the client side by using JavaScript or in the server side using server-side scripting languages such as JSP, PHP, ASP.NET, [Perl](http://www.zentut.com/perl-tutorial/).
* SQL triggers are invoked and executed invisible from the client applications, therefore, it is difficult to figure out what happen in the database layer.
* SQL triggers may increase the overhead of the database server.

In MySQL, a trigger is a set of SQL statements that is invoked automatically when a change is made to the data on the associated table. A trigger can be defined to be invoked either before or after the data is changed by INSERT, UPDATE or DELETE statement. Before MySQL version 5.7.2, you can to define maximum six triggers for each table.

* BEFORE INSERT – activated before data is inserted into the table.
* AFTER INSERT – activated after data is inserted into the table.
* BEFORE UPDATE – activated before data in the table is updated.
* AFTER UPDATE – activated after data in the table is updated.
* BEFORE DELETE – activated before data is removed from the table.
* AFTER DELETE – activated after data is removed from the table.

However, from MySQL version 5.7.2+, you can define multiple triggers for the same trigger event and action time.

When you use a statement that does not use INSERT, DELETE or UPDATE statement to change data in a table, the triggers associated with the table are not invoked. For example, the TRUNCATE statement removes all data of a table but does not invoke the trigger associated with that table.

There are some statements that use the INSERT statement behind the scenes such as REPLACE statement or LOAD DATA statement. If you use these statements, the corresponding triggers associated with the table are invoked.

You must use a unique name for each trigger associated with a table. However, you can have the same trigger name defined for different tables though it is a good practice.

You should name the triggers using the following naming convention:

(BEFORE | AFTER)\_tableName\_(INSERT| UPDATE | DELETE)

For example, before\_order\_update is a trigger invoked before a row in the order table is updated.

The following naming convention is as good as the one above.

tablename\_(BEFORE | AFTER)\_(INSERT| UPDATE | DELETE)

* **MySQL triggers Storage**

MySQL stores triggers in a data directory e.g., /data/classicmodels/ with the files named tablename.TRG and triggername.TRN :

* The tablename.TRG file maps the trigger to the corresponding table.
* the triggername.TRN file contains the trigger definition.

You can back up the MySQL triggers by copying the trigger files to the backup folder. You can also [backup the triggers using the mysqldump tool](http://mysqltutorial.org/how-to-backup-database-using-mysqldump.aspx)*.*

* **MySQL trigger limitations**

MySQL triggers cover all features defined in the standard SQL. However, there are some limitations that you should know before using them in your applications.

MySQL triggers cannot:

* Use SHOW, LOADDATA, LOADTABLE, [BACKUP DATABASE](http://mysqltutorial.org/how-to-backup-database-using-mysqldump.aspx), RESTORE, FLUSH and RETURNstatements.
* Use statements that commit or rollback implicitly or explicitly such as [COMMIT , ROLLBACK , START TRANSACTION](http://mysqltutorial.org/mysql-transaction.aspx) , [LOCK/UNLOCK TABLES](http://mysqltutorial.org/mysql-table-locking/) , ALTER , CREATE , DROP ,  [RENAME](http://mysqltutorial.org/mysql-rename-table/).
* Use [prepared statements](http://mysqltutorial.org/mysql-prepared-statement.aspx) such as PREPAREand EXECUTE.
* Use dynamic SQL statements.

# Create Trigger in MySQL:

# CREATE TRIGGER trigger\_name trigger\_time trigger\_event

# ON table\_name

# FOR EACH ROW

# BEGIN

# ...

# END;

# You put the trigger name after the CREATE TRIGGER statement. The trigger name should follow the naming convention [trigger time]\_[table name]\_[trigger event], for example before\_employees\_update.

# Trigger activation time can be BEFORE or AFTER. You must specify the activation time when you define a trigger. You use the BEFORE keyword if you want to process action prior to the change is made on the table and AFTER if you need to process action after the change is made.

# The trigger event can be INSERT, UPDATE or DELETE. This event causes the trigger to be invoked. A trigger only can be invoked by one event. To define a trigger that is invoked by multiple events, you have to define multiple triggers, one for each event.

# A trigger must be associated with a specific table. Without a table trigger would not exist therefore you have to specify the table name after the ON keyword.

# You place the SQL statements between BEGIN and END block. This is where you define the logic for the trigger.

# Create a table employees

# Employee Table

# Insert some records into it

# 

1. Create table employee\_audit

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.

**DELIMITER $$**

**CREATE TRIGGER before\_employee\_update**

**BEFORE UPDATE ON employees**

**FOR EACH ROW**

**BEGIN**

**INSERT INTO employees\_audit**

**SET action = 'update',**

**employeeNumber = OLD.employeeNumber,**

**lastname = OLD.lastname,**

**changedat = NOW();**

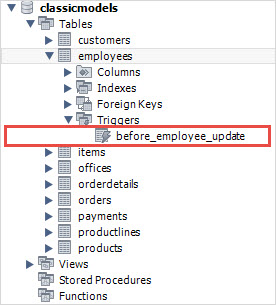
**END$$**

**DELIMITER ;**

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

Notice that in a trigger defined for [INSERT](http://mysqltutorial.org/mysql-insert-statement.aspx), you can use NEW keyword only. You cannot use the OLDkeyword. However, in the trigger defined for [DELETE](http://mysqltutorial.org/mysql-delete-statement.aspx), there is no new row so you can use the OLD keyword only. In the [UPDATE](http://mysqltutorial.org/mysql-update-data.aspx) trigger, OLD refers to the row before it is updated and NEW refers to the row after it is updated.

**How To Check Trigger :**



1. After that, update the employees table to check whether the trigger is invoked.

**UPDATE employees**

**SET lastName = 'Phan'**

**WHERE employeeNumber = 1056;**

1. Finally, to check if the trigger was invoked by the UPDATE statement, you can query the employees\_audit table using the following query:

Select \* from employees\_audit;

MySQL Trigger log