**PROCEDURE & FUNCTION**

**PROCEDURE :**

A procedure (often called a stored procedure) is a collection of pre-compiled SQL statements stored inside the database**.**

 A procedure always contains a name, parameter lists, and SQL statements.

**If we consider the enterprise application, we always need to perform specific tasks such as database cleanup, processing payroll, and many more on the database regularly. Such tasks involve multiple**[**SQL**](https://www.javatpoint.com/sql-tutorial)**statements for executing each task. This process might easy if we group these tasks into a single task. We can fulfill this requirement in**[**MySQL**](https://www.javatpoint.com/mysql-tutorial)**by creating a stored procedure in our database.**

**recursive stored procedure is not supported well in MySQL.**

**STORED PROCEDURE FEATURES**

* Stored procedures boost application performance by being precompiled and stored in the database.
* They reduce server traffic by requiring only the procedure's name and parameters, not multiple SQL queries.
* They're reusable, secure, and allow admins to grant access without exposing database tables.

**SYNTAX :**

**CREATE PROCEDURE**procedure\_name**[[IN | OUT | INOUT]**parameter\_name datatype**[,**parameter datatype**]) ]**

**BEGIN**

Declaration\_section

    Executable\_section

**END**

**-------------------------------------------------------------------------------------------------------------------**

**CALL procedure\_name ( parameter(s))  ;**

**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.

**CREATE** **PROCEDURE** display\_max\_mark (**OUT** highestmark **INT**)

**BEGIN**

**SELECT** **MAX**(marks) **INTO** highestmark **FROM** student\_info;

**END**

**---------------------------------------------------------------------------------------**

CALL display\_max\_mark(@M);

**---------------------------------------------------------------------------------------**

**SELECT** @M;

**---------------------------------------------------------------------------------------**

**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.

**CREATE PROCEDURE display\_marks (INOUT var1 INT)**

**BEGIN**

**SELECT marks INTO var1 FROM student\_info WHERE stud\_id = var1;**

**END**

**------------------------------------------------------------------------------------------------------------------**

**SET @M = '3';**

**------------------------------------------------------------------------------------------------------------------**

**CALL display\_marks(@M);**

**------------------------------------------------------------------------------------------------------------------**

**SELECT @M;**

**SHOW STORED PROCEDURES :**

To list all stored procedures in your database, you can use:

**SHOW PROCEDURE STATUS WHERE Db = 'your\_database\_name';**

Or, to see the code of a specific procedure**:**

**SHOW CREATE PROCEDURE your\_procedure\_name;**

**DROP (DELETE) A STORED PROCEDURE :**

**DROP PROCEDURE IF EXISTS your\_procedure\_name;**

**DRAWBACKS OF USING PROCEDURE :**

* **Increased Memory and CPU Usage:** Using stored procedures can significantly increase memory and CPU usage, especially with complex logic.
* **Limited Debugging Support:** Debugging stored procedures is difficult, and MySQL lacks proper debugging tools.
* **Complex Development:** Developing and maintaining stored procedures requires specialized skills, which can complicate the process.
* **Not Ideal for Complex Logic:** Stored procedures are not well-suited for developing complex and flexible business logic.

**FUNCTIONS**

In MySQL, Function can also be created. A function always returns a value using the return statement. The function can be used in SQL queries.

**SYNTAX :**

**CREATE FUNCTION function\_name [**(parameter datatype [, parameter datatype]) **]**

**RETURNS**return\_datatype

**BEGIN**

Declaration\_section

Executable\_section

**END;**

**DROP A FUNCTION :**

In MySQL Function can also be dropped. When A function id dropped, it is removed from the database.

**DROP** **FUNCTION** [ IF EXISTS ] FUNCTION\_NAME;

**CALL FUNCTION :**

**SELECT function\_name(parameters);**

OR

**DECLARE** variable\_name **DATATYPE**;

**SET** result = function\_name(parameters);

**ADVANTAGES OF FUNCTIONS :**

**Code Reusability:** Functions allow you to reuse code across multiple queries, reducing redundancy and errors.

**Modularity:** Functions break down complex operations into smaller, manageable parts, making code easier to understand and maintain.

**Performance:** Precompiled functions can improve performance by executing faster than equivalent SQL code written inline.

**Consistency:** Using functions ensures consistent calculations and business logic across your application.

**Ease of Use:** Once defined, functions can be easily called with different inputs, simplifying complex operations.

**DRAWBACKS OF FUNCTIONS :**

**Limited Functionality:** Functions typically can't perform tasks like data modification, which limits their use to specific scenarios.

**Performance Overhead:** While functions can improve performance, overuse or poorly designed functions may cause additional overhead.

**Debugging Challenges:** Debugging functions can be difficult, especially if the database lacks robust debugging tools.

**Complex Development:** Writing and maintaining complex functions can require advanced SQL skills, making it challenging for some developers.

**Dependency Issues:** Over-reliance on functions can lead to tight coupling, where changes to the function require changes across multiple queries or procedures.