**STORED PROCEDURES**

**Stored Procedure**

* **A Stored procedure is a collection of pre-compiled SQL statements stored inside the database.**
* **It was first introduced in MySQL version 5. (My MySQL version is version 8)**
* **A procedure always contains a name, parameter lists (Optional), and SQL statements.**
* **We can invoke the procedures by using triggers, other procedures and applications such as Java, Python, PHP, etc.**
* **A procedure is called a recursive stored procedure when it calls itself. (It is not supported well in MySQL.**)

**Advantages**

* **Reduce network traffic**

Stored procedures reduce the network traffic between applications and MySQL Server, because the application has to send only the stored procedure's name and parameters instead of sending multiple SQL statements.

* **Centralize business logic in the database**

You can use the stored procedures to implement business logic that is reusable by multiple applications. The stored procedures reduce the efforts of duplicating the same logic in many applications and make your database more consistent.

* **Make database more secure**

A procedure is always secure. The database administrator can grant permissions to applications that access stored procedures in the database without giving any permission on the database tables.

* **Increases the performance**

Stored Procedure increases the performance of the applications. Once stored procedures are created, they are compiled and stored in the database.

**Disadvantages**

* **Resource usages**

If you use many stored procedures, the memory usage of every connection will increase. Overusing a large number of logical operations in the stored procedures will increase the CPU usage.

* **Troubleshooting**

It’s difficult to debug stored procedures. Unfortunately, MySQL does not provide any facilities to debug stored procedures like other enterprise database products such as Oracle and SQL Server.

* **Maintenance**

Developing and maintaining stored procedures often requires a specialized skill set that not all application developers possess. This may lead to problems in both application development and maintenance.

**Syntax**

**For creating a Stored Procedure -**

DELIMITER $$

CREATE PROCEDURE procedure\_name(parameter\_list)

BEGIN

Statements;

END $$

DELIMITER;

**Explanation**

* First, specify the name of the stored procedure that you want to create after the **CREATE PROCEDURE** keywords.
* Second, specify a list of comma-separated parameters for the stored procedure in parentheses after the procedure name.
* Third, write the code between the BEGIN and END blocks. After the END keyword, you place the delimiter character to end the procedure statement.

**Executing a stored procedure**

**CALL stored\_procedure\_name(argument\_list);**

**Explanation**

You specify the name of the stored procedure after the CALL keyword. If the stored procedure has parameters, you need to pass arguments inside parentheses

**Delimiter**

* MySQL program uses the delimiter (;) to separate statements and executes each statement separately.
* If you use a MySQL client program to define a stored procedure that contains semicolon characters, the MySQL client program will not treat the whole stored procedure as a single statement, but many statements.
* You must redefine the delimiter temporarily so that you can pass the whole stored procedure to the server as a single statement.
* “**DELIMITER delimiter\_character**” is the redefined delimiter
* The delimiter\_character may consist of **// or $$.** However, you should avoid using the backslash (\) because it’s the escape character in MySQL.

**For Example -**

DELIMITER $$

CREATE PROCEDURE procedure\_name(parameter\_list)

BEGIN

Statements;

END $$

DELIMITER;

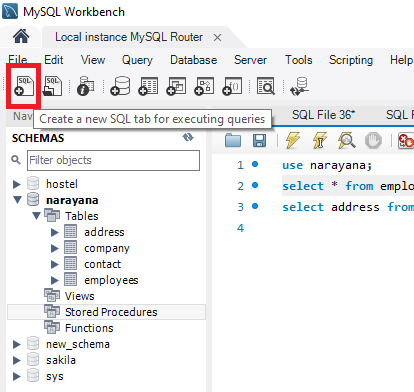
**Explanation**

* First, change the default delimiter to $$.
* Second, use (;) in the body of the stored procedure and $$ after the END keyword to end the stored procedure.
* Third, change the default delimiter back to a semicolon (;)

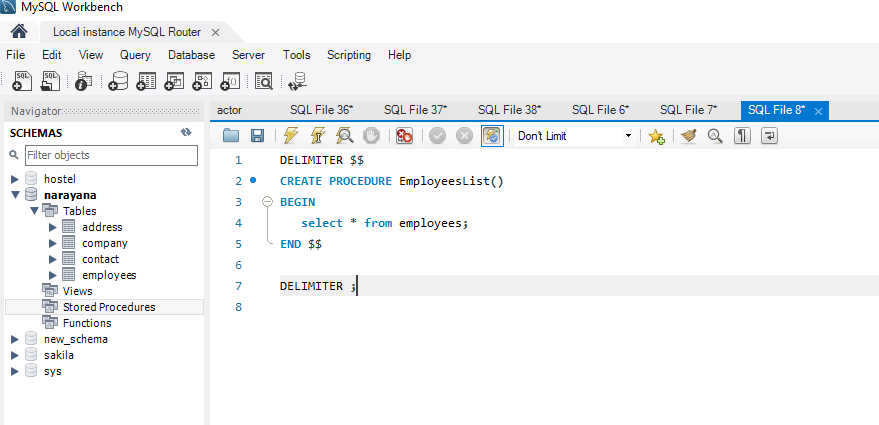
**How to create a Stored Procedure**

**Method 1**

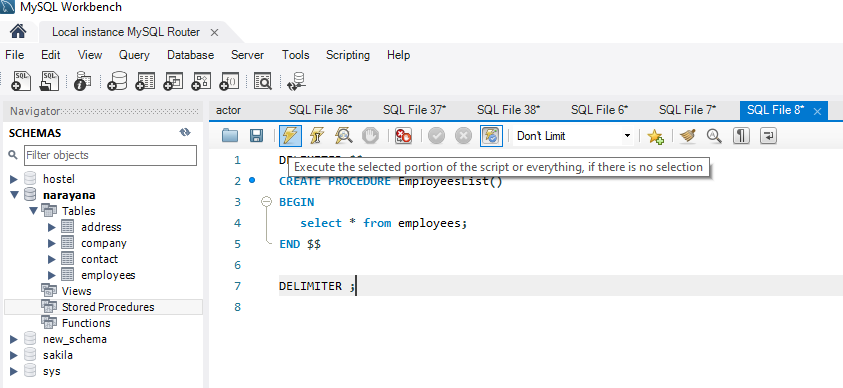
* launch MySQL Workbench and click on **create a new SQL tab for executing queries**

****

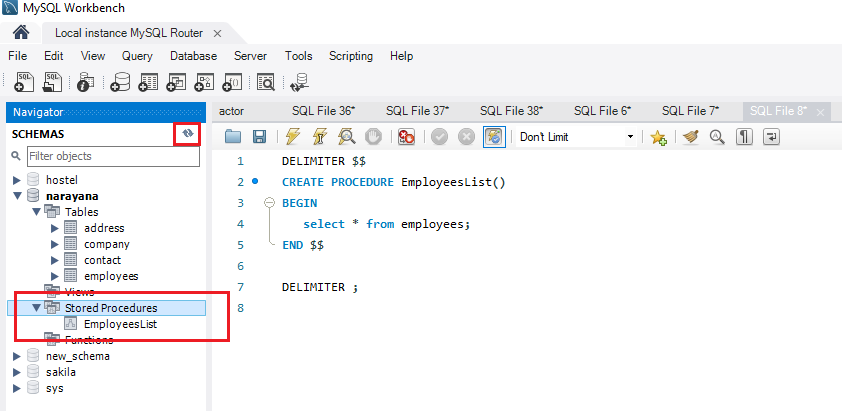
* Enter the statements in the SQL tab



* Note that you can select all statements in the SQL tab (or nothing) and click the **Execute** button. MySQL will create the stored procedure and save it in the server.

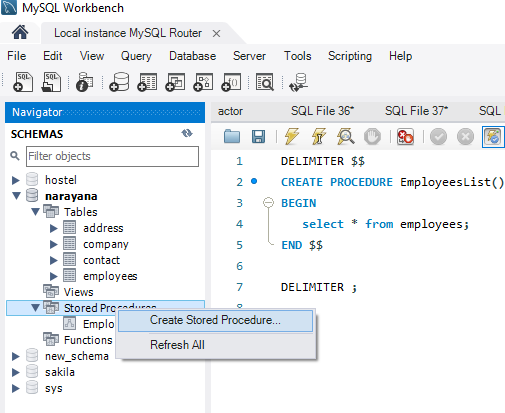


* Check the stored procedure by opening the **Stored Procedures node**. If you don’t see the stored procedure, you can click the **Refresh button** next to the **SCHEMAS** title

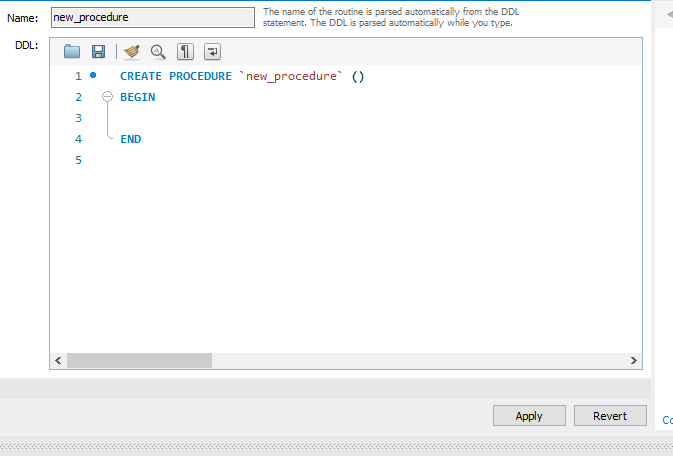


**Method 2**

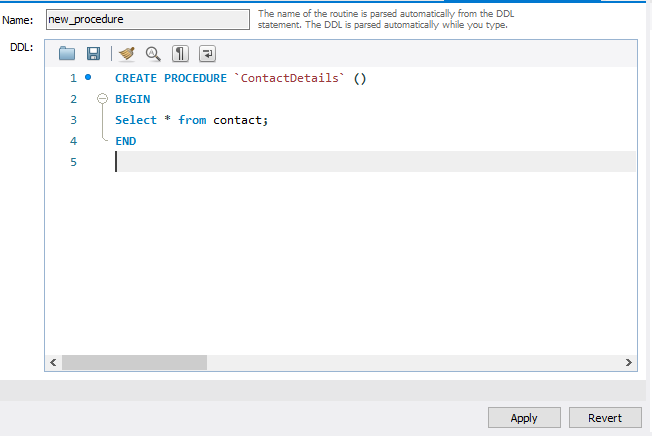
* Right-click on the **Stored Procedures** from the **Navigator** and select the **Create Stored Procedure** (menu item).



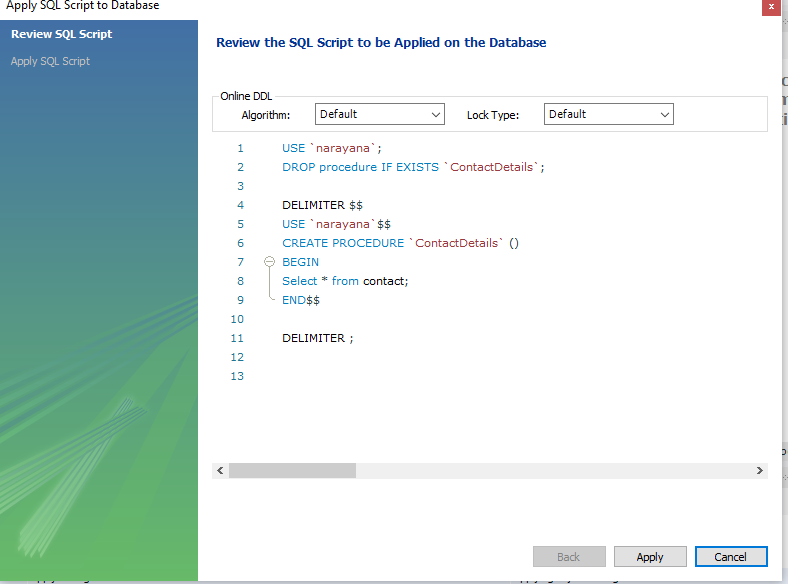
* The following tab will open



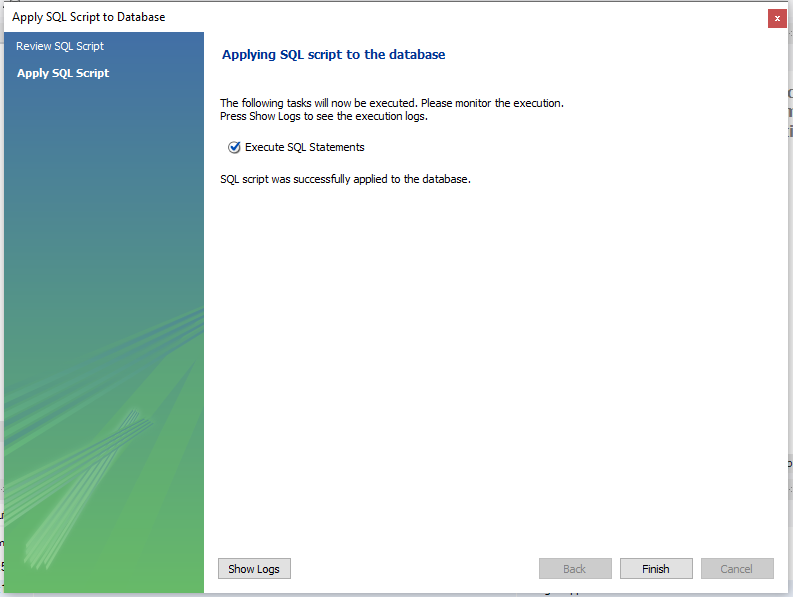
* Change the stored procedure’s name and add the code between the BEGIN and END block



* Click the **Apply** button, MySQL Workbench will open a new window for reviewing SQL script before applying it on the database



* Click the **Apply** button to confirm. MySQL Workbench will create the stored procedure



* Click the **Finish** button to close the window.

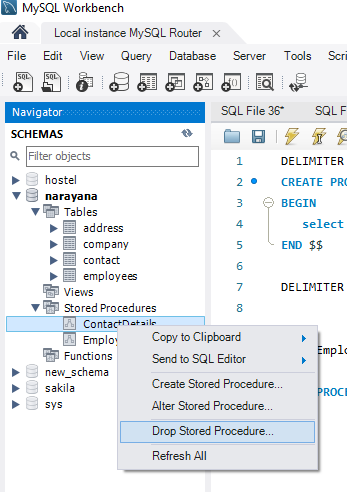
**How to delete the stored procedure**

**Method 1**

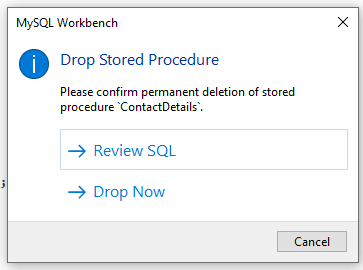
* The **DROP PROCEDURE** statement deletes a stored procedure created by the **CREATE PROCEDURE** statement.
* Syntax : **DROP PROCEDURE IF EXISTS stored\_procedure\_name;**
* Specify the name of the stored procedure that you want to remove after the DROP PROCEDURE keywords and **IF EXISTS** is optional
* When you drop a procedure that does not exist without using the IF EXISTS option, MySQL issues an error. In this case, if you use the IF EXISTS option, MySQL issues a warning instead.
* MySQL error: **Error Code: 1305. PROCEDURE narayana.ContactDetails does not exist**
* MySQL warning: **0 row(s) affected, 1 warning(s): 1305 PROCEDURE narayana.ContactDetails does not exist**

**Method 2**

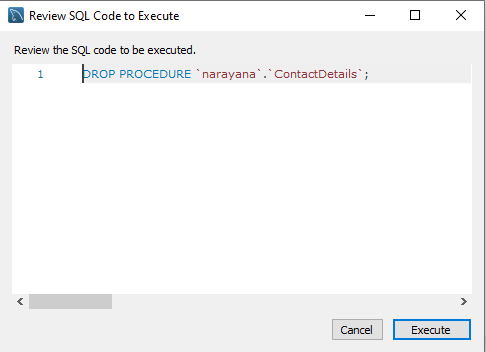
* Right-click the name of the stored procedure that you want to remove and choose **Drop Stored Procedure** (option).



* Click on **Review SQL** to review the SQL statement, or click on **Drop Now** if you want to immediately remove the stored procedure.



* Review the SQL code and click on **Execute** button to drop the stored procedure.

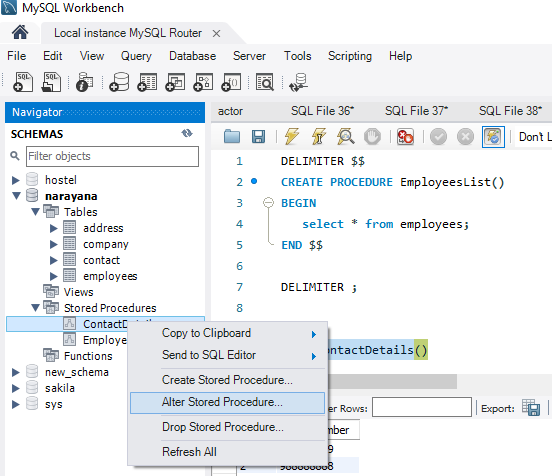


**How to modify (alter) the stored procedure**

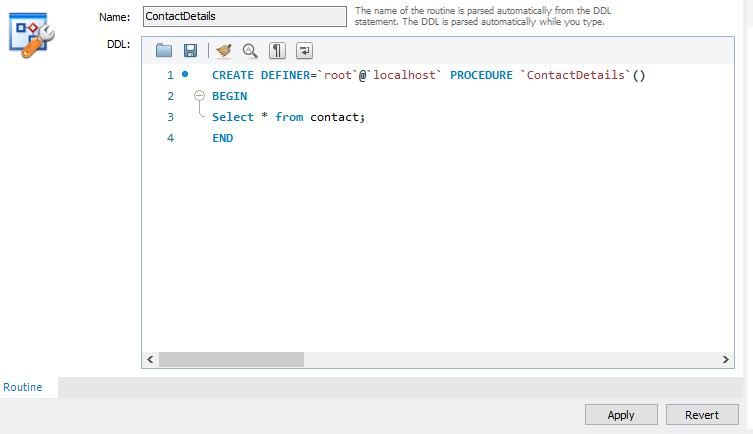
* MySQL does not have any statement that allows you to directly modify (adding or removing) the parameters and body of the stored procedure.
* To make such changes, you must drop and re-create the stored procedure using the **DROP PROCEDURE** and **CREATE PROCEDURE** statements.
* Using MySQL Workbench, you can modify (adding or removing) the parameters and body of the stored procedure.

**Steps**

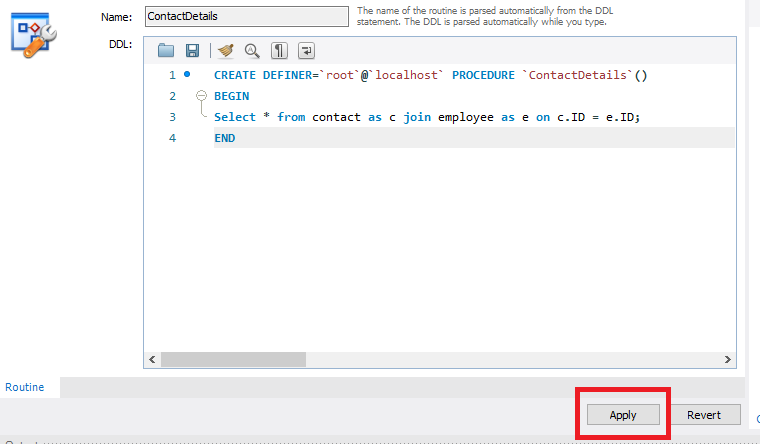
* Right-click the stored procedure that you want to change and select **Alter Stored Procedure**



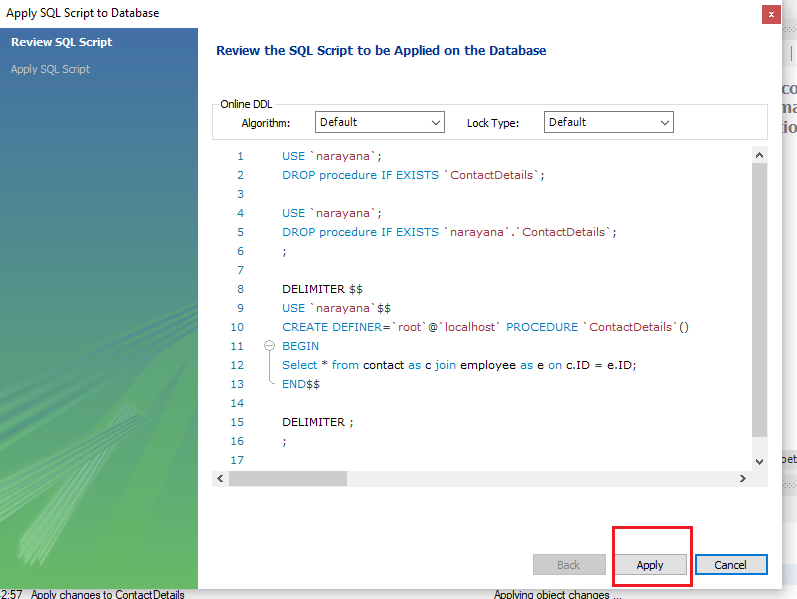
* MySQL Workbench will open a new tab that contains the definition of the stored procedure.



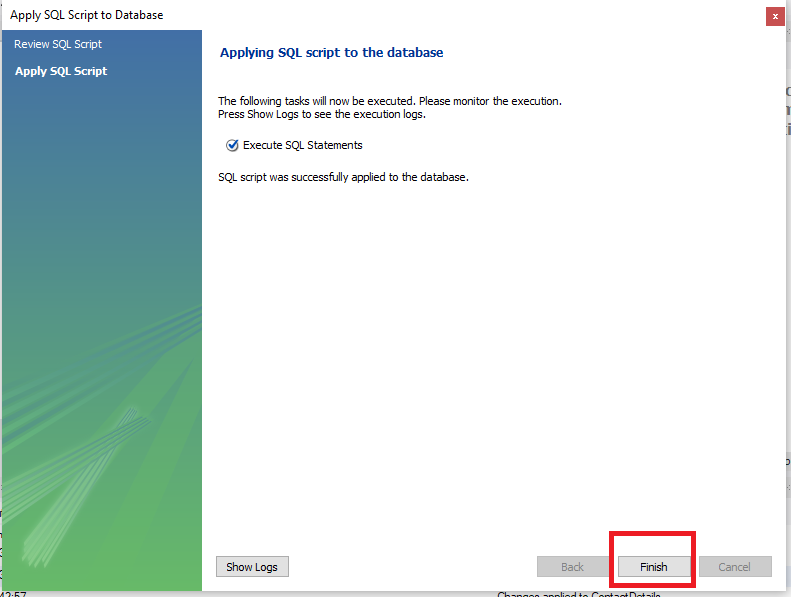
* Make the changes and click the **Apply** button.



* MySQL Workbench will display a SQL Script review window and click on **Apply**.



* Click the Finish button to complete the change.



**Different types of Procedures**

There are **four** different types of MySQL procedures:

**1. Procedure with no parameters:**

A procedure without parameters does not take any input. It is simply called with its procedure name followed by () (without any parameters). It is used for simple queries.

**2. Procedure with IN parameter:**

An IN parameter is used to take a parameter as input. When we define an IN parameter in a 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 if the value of the IN parameter is changed inside the procedure, its original value is retained after the procedure ends (like pass by value). In other words, the procedure only works on the copy of the IN parameter.

**3. Procedure with OUT parameter:**

An OUT parameter is used to pass a parameter as output. The value of an OUT parameter can be changed inside the procedure and its new value is passed back to the calling program.

**4. Procedure with IN-OUT parameter:**

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**

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

* First, specify the parameter mode, which can be IN , OUT or 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.

**Example for Procedure with no parameters**

DELIMITER $$

CREATE PROCEDURE WithoutParameters()

BEGIN

select \* from employees;

insert into employees(name,lastname) values("Kohli","Virat");

select \* from employees;

update employees set name = "Rohit",lastname = "Sharma" where ID = 40;

select \* from employees;

delete from employees where ID = 40;

select \* from employees;

END $$

DELIMITER ;

Call WithoutParameters();

**Example for Procedure with In parameters**

DELIMITER $$

CREATE PROCEDURE InParameters(in name1 varchar(100),in lastname1 varchar(100),

in name2 varchar(100),in lastname2 varchar(100),in ID1 int)

BEGIN

select \* from employees;

insert into employees(name,lastname) values(name1,lastname1);

select \* from employees;

update employees set name = name2,lastname = lastname2 where ID = ID1;

select \* from employees;

delete from employees where ID = ID1;

select \* from employees;

END $$

DELIMITER ;

Call InParameters("Kohli","Virat","Rohit","Sharma",41);

**Example for Procedure with Out parameters**

DELIMITER $$

CREATE PROCEDURE OutParameters(out NoOfFilms int,out NoOfFilmCategories int)

BEGIN

select count(film\_id) into NoOfFilms from sakila.film;

select count(distinct(category\_id)) into NoOfFilmCategories from sakila.film\_category;

END$$

DELIMITER ;

Call OutParameters(@NoOfFilms,@NoOfFilmCategories);

select @NoOfFilms, @NoOfFilmCategories;

**Example for Procedure with InOut parameters**

DELIMITER $$

CREATE PROCEDURE InOutParameters(inout Number int)

BEGIN

select Phonenumber into Number from contact where ID = Number;

END$$

DELIMITER ;

set @Number = 1 ;

Call InOutParameters(@Number);

select @Number;