

In this article, we are going to learn about the stored procedures in [MySQL](#). In this article, I am covering the basics of the stored procedure that includes the following

1. Summary of MySQL Stored Procedure
2. Create a stored procedure using Query and MySQL workbench
3. Create a Parameterized stored procedure
4. Drop the Stored Procedure using query and MySQL workbench

The stored procedure is SQL statements wrapped within the **CREATE PROCEDURE** statement. The stored procedure may contain a conditional statement like IF or CASE or the Loops. The stored procedure can also execute another stored procedure or a function that modularizes the code.

Following are the benefits of a stored procedure:

1. **Reduce the Network Traffic:** Multiple SQL Statements are encapsulated in a stored procedure. When you execute it, instead of sending multiple queries, we are sending only the name and the parameters of the stored procedure
2. **Easy to maintain:** The stored procedure are reusable. We can implement the business logic within an SP, and it can be used by applications multiple times, or different modules of an application can use the same procedure. This way, a stored procedure makes the database more consistent. If any change is required, you need to make a change in the stored procedure only
3. **Secure:** The stored procedures are more secure than the AdHoc queries. The permission can be granted to the user to execute the stored procedure without giving permission to the tables used in the stored procedure. The stored procedure helps to prevent the database from SQL Injection

The syntax to create a MySQL Stored procedure is the following:

```
Create Procedure [Procedure Name] ([Parameter 1], [Parameter 2], [Parameter 3] )  
Begin  
SQL Queries..  
End
```

In the syntax:

1. The name of the procedure must be specified after the **Create Procedure** keyword
2. After the name of the procedure, the list of parameters must be specified in the parenthesis. The parameter list must be comma-separated
3. The SQL Queries and code must be written between **BEGIN** and **END** keywords

1. The name of the procedure must be specified after the **CREATE PROCEDURE** keyword
2. After the name of the procedure, the list of parameters must be specified in the parenthesis. The parameter list must be comma-separated
3. The SQL Queries and code must be written between **BEGIN** and **END** keywords

To execute the store procedure, you can use the CALL keyword. Below is syntax:

```
CALL [Procedure Name] ([Parameters]..)
```

In the syntax:

1. The procedure name must be specified after the CALL keyword
2. If the procedure has the parameters, then the parameter values must be specified in the parenthesis

Let us create a basic stored procedure. For demonstration, I am using the **sakila** database.

## Create a simple stored procedure

Suppose you want to populate the list of films. The output should contain film\_id, title, description, release year, and rating column. The code of the procedure is the following:

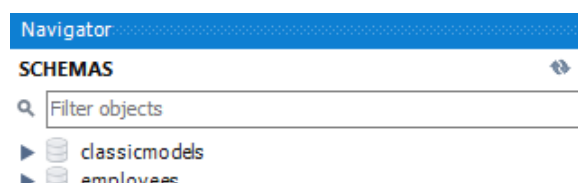
```
DELIMITER //
CREATE PROCEDURE sp_GetMovies()
BEGIN
    select title,description,release_year,rating from film;
END //
DELIMITER ;
```

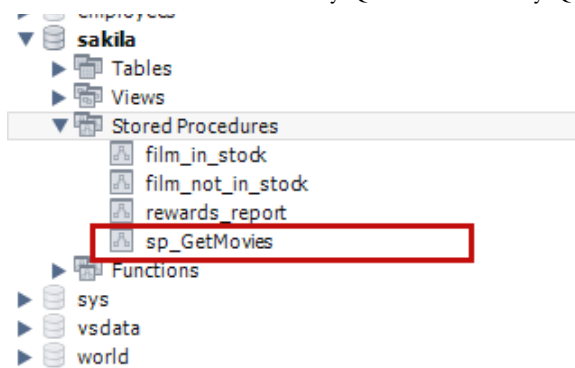
To create the MySQL Stored Procedure, open the **MySQL workbench** Connect to the **MySQL Database** copy-paste the code in the query editor window click on **Execute**.

The screenshot shows the MySQL Workbench interface. The top toolbar has the 'Execute' button (a lightning bolt icon) highlighted with a red box. Below the toolbar, the query editor contains the SQL code for creating the stored procedure. The 'Output' tab is selected, showing a table with columns '#', 'Time', 'Action', and 'Message'. The first row shows a successful execution of the procedure.

#	Time	Action	Message
1	15:04:17	CREATE PROCEDURE sp_GetMovies() BEGIN select title,description,release_year,rating from film; END	0 row(s) affected

You can view the procedure under stored procedures. See the below screenshot.



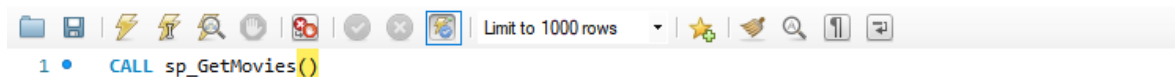


Administration Schemas

To execute the procedure, run the below command.

```
CALL sp_GetMovies
```

Below is the partial screenshot of the output:

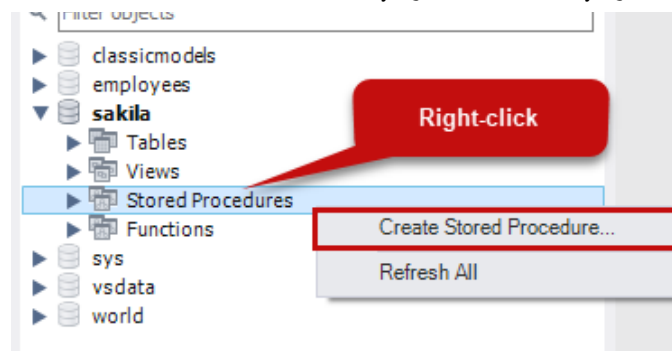


title	description	release_year	rating
ACADEMY DINOSAUR	A Epic Drama of a Feminist And a Mad Scientist who must Battle a Teacher in T...	2006	PG
ACE GOLDFINGER	A Astounding Epistle of a Database Administrator And a Explorer who must Fi...	2006	G
ADAPTATION HOLES	A Astounding Reflection of a Lumberjack And a Car who must Sink a Lumberja...	2006	NC-17
AFFAIR PREJUDICE	A Fandful Documentary of a Frisbee And a Lumberjack who must Chase a Mo...	2006	G
AFRICAN EGG	A Fast-Paced Documentary of a Pastry Chef And a Dentist who must Pursue a...	2006	G
AGENT TRUMAN	A Intrepid Panorama of a Robot And a Boy who must Escape a Sumo Wrestler...	2006	PG
AIRPLANE SIERRA	A Touching Saga of a Hunter And a Butler who must Discover a Butler in A Jet ...	2006	PG-13
AIRPORT POLLOCK	A Epic Tale of a Moose And a Girl who must Confront a Monkey in Ancient India	2006	R
ALABAMA DEVIL	A Thoughtful Panorama of a Database Administrator And a Mad Scientist who ...	2006	PG-13
ALADDIN CALENDAR	A Action-Packed Tale of a Man And a Lumberjack who must Reach a Feminist i...	2006	NC-17
ALAMO VIDEOTAPE	A Boring Epistle of a Butler And a Cat who must Fight a Pastry Chef in A MySQL...	2006	G
ALASKA PHANTOM	A Fandful Saga of a Hunter And a Pastry Chef who must Vanquish a Boy in Au...	2006	PG
ALI FOREVER	A Action-Packed Drama of a Dentist And a Crocodile who must Battle a Femini...	2006	PG

## Create procedure using MySQL workbench wizard

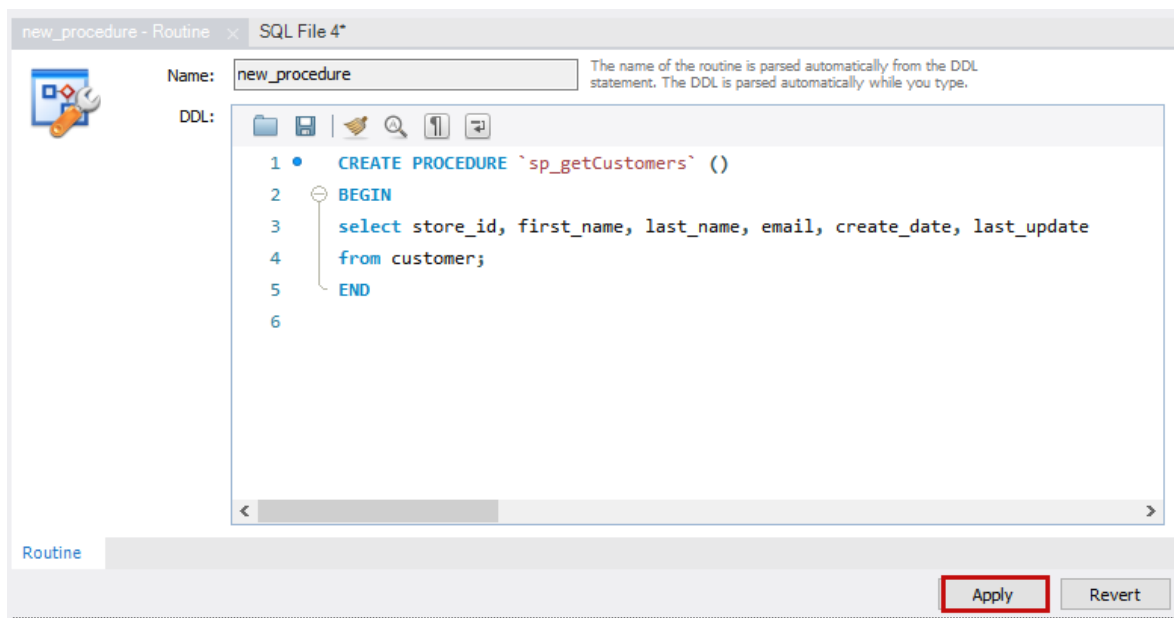
We can use the MySQL workbench wizard to create a stored procedure. Suppose you want to get the list of the customer from the sakila database. To do that, expand the **sakila** schema Right-click on **Stored Procedures** Select **Create a Stored procedure**.



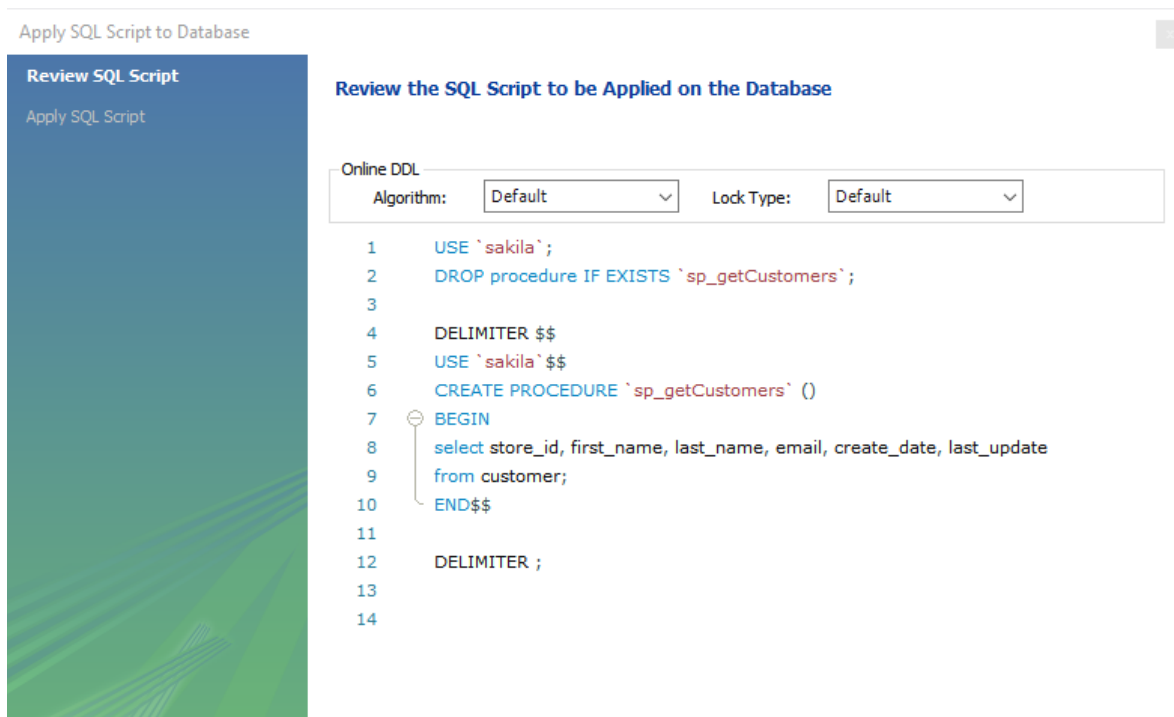


In the New procedure window, a create template has been created. In the template, replace the procedure name with ***sp\_getCustomers***. In the code block, enter the following query

```
select store_id, first_name, last_name, email, create_date, last_update from customer
```



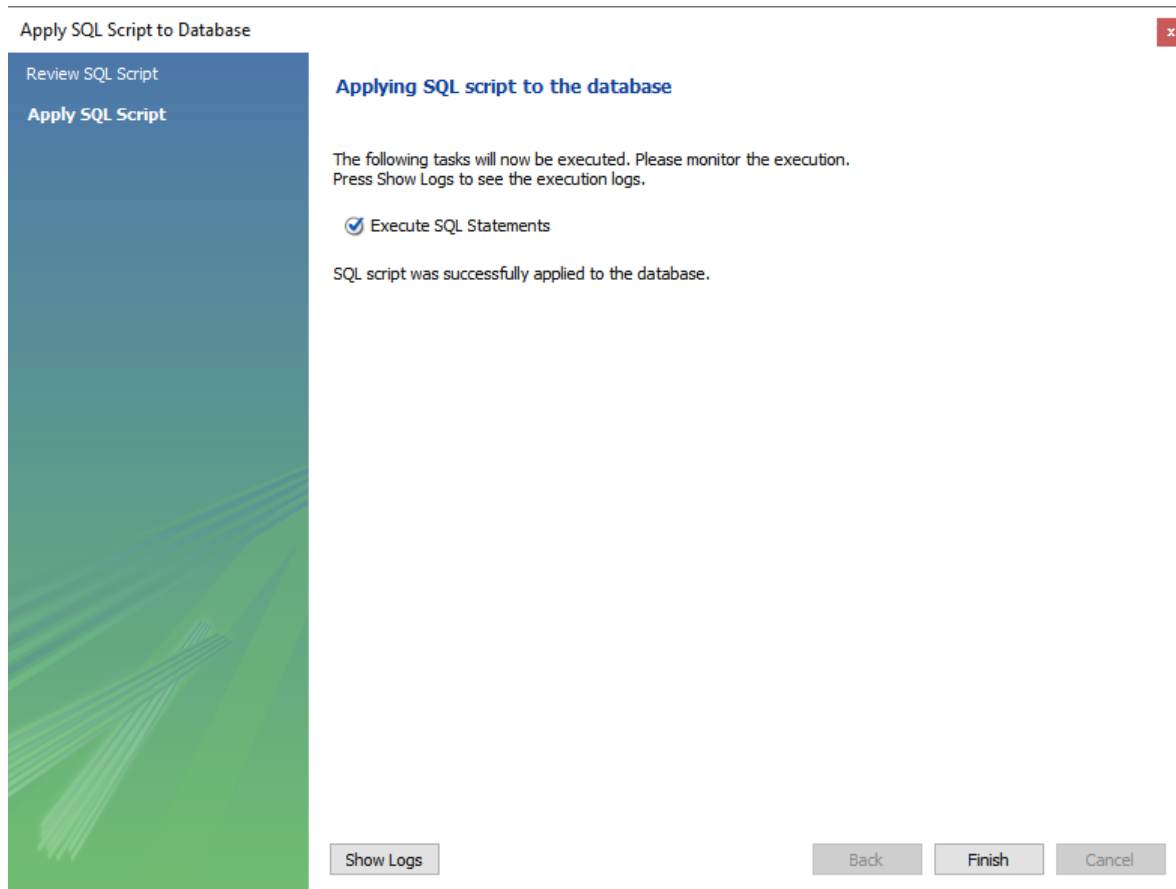
Click on Apply. A dialog box, Apply script to database opens. On the Review the script screen, you can view the code of the stored procedure. Click on Apply.



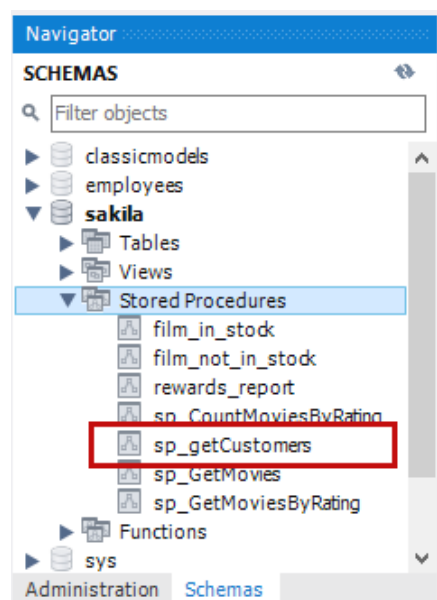


Back Apply Cancel

The script is applied successfully, and a stored procedure ***sp\_GetCustomer*** has been created successfully.



In MySQL Workbench, You can view the stored procedure under the ***Stored Procedures*** folder of the sakila schema.



## Create a parameterized stored procedure

The MySQL Stored procedure parameter has three modes: IN, OUT, and INOUT. When we declare an IN type parameter, the application must pass an argument to the stored procedure. It is a default mode. The OUT type parameter, the stored procedure returns a final output generated by SQL Statements. When we declare the IN-OUT type parameter, the application has to pass an argument, and based on the input argument; the procedure returns the output to the application.

When we create a stored procedure, the parameters must be specified within the parenthesis. The syntax is following:

```
(IN | OUT | INOUT) (Parameter Name [datatype(length)])
```

In the syntax:

1. Specify the type of the parameter. It can be IN, OUT or INOUT
2. Specify the name and data type of the parameter

## Example of IN parameter

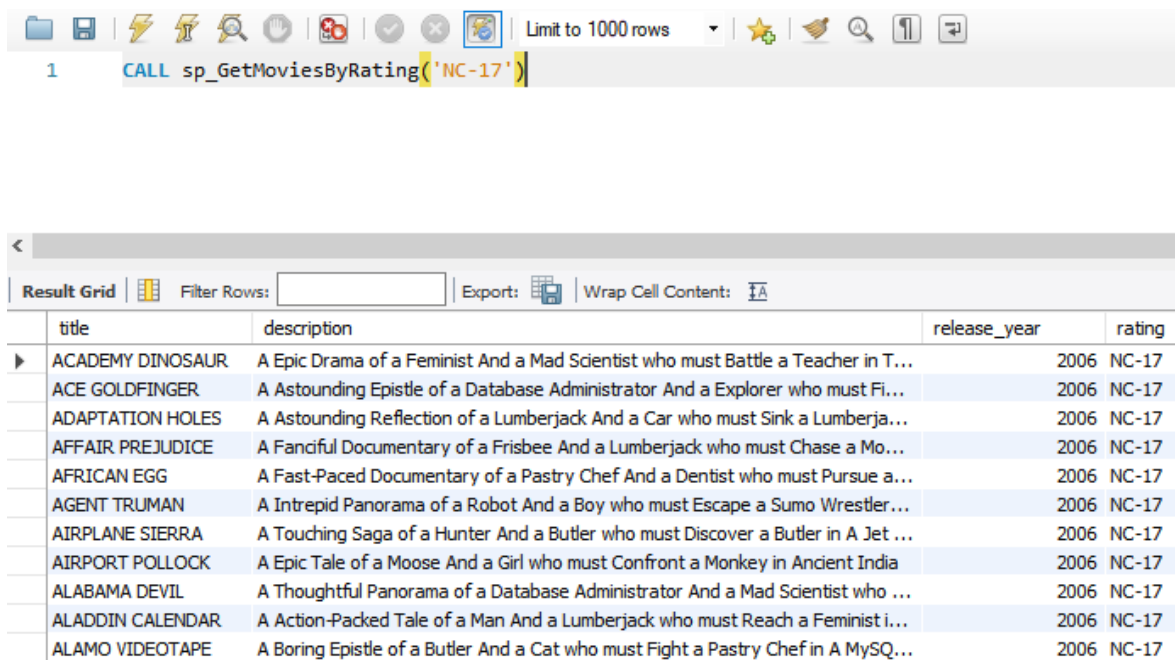
Suppose we want to get the list of films based on the rating. The **param\_rating** is an input parameter, and the data type is varchar. The code of the procedure is the following:

```
DELIMITER //
CREATE PROCEDURE sp_GetMoviesByRating(IN rating varchar(50))
BEGIN
    select title,description,release_year,rating from film where rating=rating;
END //
DELIMITER ;
```

To populate the list of the films with an **NC-17** rating, we pass the **NC-17** value to the **sp\_getMoviesByRating()** procedure.

```
CALL sp_GetMoviesByRating('NC-17')
```

Output:



The screenshot shows a MySQL IDE interface. At the top, there's a toolbar with various icons. Below it, a text area contains the command: `CALL sp_GetMoviesByRating('NC-17')`. Below the text area, there's a "Result Grid" section. It includes a "Filter Rows:" input field, an "Export:" button, and a "Wrap Cell Content:" checkbox. The main part of the result grid is a table with the following data:

	title	description	release_year	rating
▶	ACADEMY DINOSAUR	A Epic Drama of a Feminist And a Mad Scientist who must Battle a Teacher in T...	2006	NC-17
	ACE GOLDFINGER	A Astounding Epistle of a Database Administrator And a Explorer who must Fi...	2006	NC-17
	ADAPTATION HOLES	A Astounding Reflection of a Lumberjack And a Car who must Sink a Lumberja...	2006	NC-17
	AFFAIR PREJUDICE	A Fanciful Documentary of a Frisbee And a Lumberjack who must Chase a Mo...	2006	NC-17
	AFRICAN EGG	A Fast-Paced Documentary of a Pastry Chef And a Dentist who must Pursue a...	2006	NC-17
	AGENT TRUMAN	A Intrepid Panorama of a Robot And a Boy who must Escape a Sumo Wrestler...	2006	NC-17
	AIRPLANE SIERRA	A Touching Saga of a Hunter And a Butler who must Discover a Butler in A Jet ...	2006	NC-17
	AIRPORT POLLOCK	A Epic Tale of a Moose And a Girl who must Confront a Monkey in Ancient India	2006	NC-17
	ALABAMA DEVIL	A Thoughtful Panorama of a Database Administrator And a Mad Scientist who ...	2006	NC-17
	ALADDIN CALENDAR	A Action-Packed Tale of a Man And a Lumberjack who must Reach a Feminist i...	2006	NC-17
	ALAMO VIDEOTAPE	A Boring Epistle of a Butler And a Cat who must Fight a Pastry Chef in A MySQL...	2006	NC-17

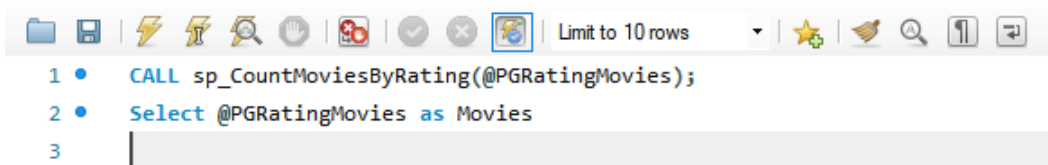
## Example of OUT parameter

Suppose we want to get the count of the films that have a **PG-13** rating. The **Total\_Movies** is an output parameter, and the data type is an integer. The count of the movies is assigned to the **OUT** variable (**Total\_Movies**) using the INTO keyword. The code of the procedure is the following:

```
DELIMITER //
CREATE PROCEDURE sp_CountMoviesByRating(OUT Total_Movies int)
BEGIN
    select count(title) INTO Total_Movies from film where rating='PG-13';
END //
DELIMITER ;
```

To store the value returned by the procedure, pass a session variable named **@PGRatingMovies**.

```
CALL sp_CountMoviesByRating(@PGRatingMovies)
Select @PGRatingMovies as Movies
```



Result Grid	
	Movies
▶	223

## Example of an INOUT parameter

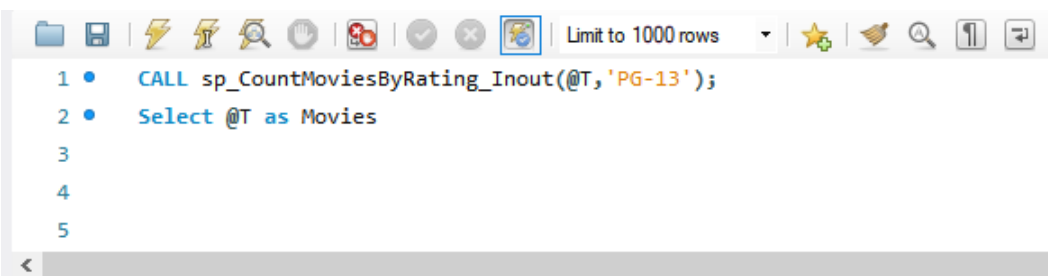
Suppose we want to get the total count of movies based on the rating. The input parameter is param\_rating in the procedure, and the data type is **varchar(10)**. The output parameter is **Movies\_count**, and the data type is an **integer**.

Code of procedure:

```
DELIMITER //
CREATE PROCEDURE sp_CountMoviesByRating_Inout(inout Movies_count int, In param_rating varchar(10))
BEGIN
    select count(title) INTO Movies_count from film where rating=param_rating ;
END //
DELIMITER ;
```

Execute the procedure using **CALL** keyword and save the output in session variable named **@MoviesCount**

```
CALL sp_CountMoviesByRating_Inout (@T, 'PG-13');
Select @T as Movies
```





Result Grid	Filter Rows:	Export:	Wrap Cell Content:
Movies			
223			

## View the list of stored procedure in a database using a query

To view the list of the stored procedure, you can query the `information_schema.routines` table. It contains the list of the stored procedure and stored functions created on the database. To view the list of the **stored procedure** created in a sakila database, run the following query. Moreover, it also provides the **owner**, **created date**, **security type**, and **SQL data access** to the stored procedures.

```
select routine_name, routine_type, definer, created, security_type, SQL_Data_Access from information_schema.routines where routine_type='PROCEDURE' and routine_schema='sakila';
```

Limit to 1000 rows	
1 • use sakila;	
2 • select routine_name, routine_type, definer, created, security_type, SQL_Data_Access	
3 from information_schema.routines where routine_type='PROCEDURE' and routine_schema='sakila';	
4	
5	

ROUTINE_NAME	ROUTINE_TYPE	DEFINER	CREATED	SECURITY_TYPE	SQL_DATA_ACCESS
film_in_stock	PROCEDURE	root@localhost	2020-11-24 14:59:01	DEFINER	READS SQL DATA
film_not_in_stock	PROCEDURE	root@localhost	2020-11-24 14:59:01	DEFINER	READS SQL DATA
rewards_report	PROCEDURE	root@localhost	2020-11-24 14:59:01	DEFINER	READS SQL DATA
sp_CountMoviesByRating	PROCEDURE	root@localhost	2020-11-24 17:55:35	DEFINER	CONTAINS SQL
sp_CountMoviesByRating_Inout	PROCEDURE	root@localhost	2020-11-25 23:12:21	DEFINER	CONTAINS SQL
sp_getCustomers	PROCEDURE	root@localhost	2020-11-25 23:41:40	DEFINER	CONTAINS SQL
sp_GetMovies	PROCEDURE	root@localhost	2020-11-24 15:04:17	DEFINER	CONTAINS SQL
sp_GetMoviesByRating	PROCEDURE	root@localhost	2020-11-24 17:16:58	DEFINER	CONTAINS SQL

## Drop a Stored Procedure

To drop the stored procedure, you can use the drop procedure command. The syntax is following

```
Drop procedure [IF EXISTS] <Procedure Name>
```

In the syntax, the name of the stored procedure must be followed by the **Drop Procedure** keyword. If you want to drop the **sp\_getCustomers** procedure from the sakila database, you can run the following query.

```
Drop procedure sp_getCustomers
```

When you try to drop the procedure that does not exist on a database, the query shows an error:

```
ERROR 1305 (42000): PROCEDURE sakila.getCustomer does not exist
```

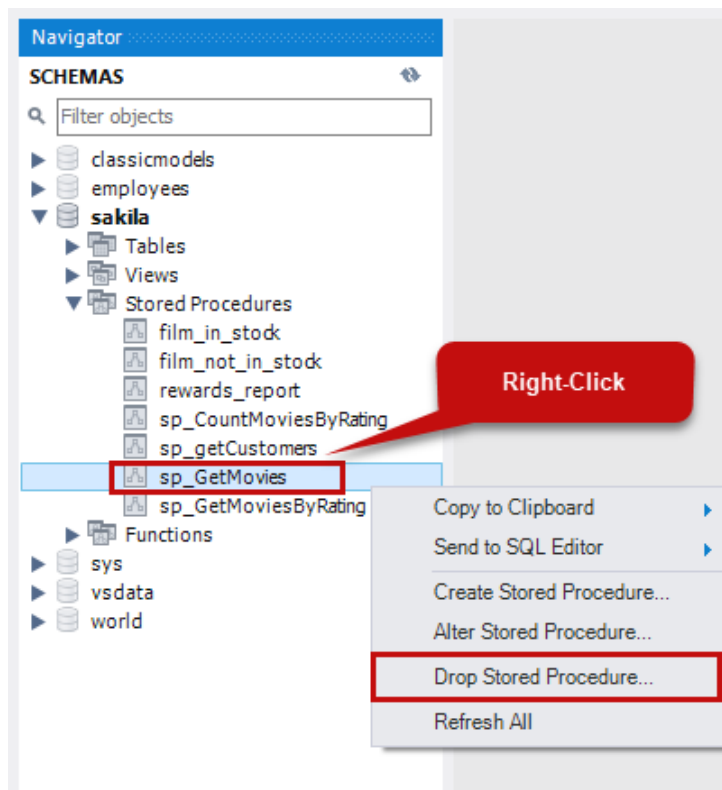
To avoid this, you can include the `[IF EXISTS]` option in the drop procedure command. When you include the `IF EXISTS` keyword, instead of an error, the query returns a warning:

```
Query OK, 0 rows affected, 1 warning (0.01 sec) 1305 PROCEDURE sakila.getCustomer does not exist
```

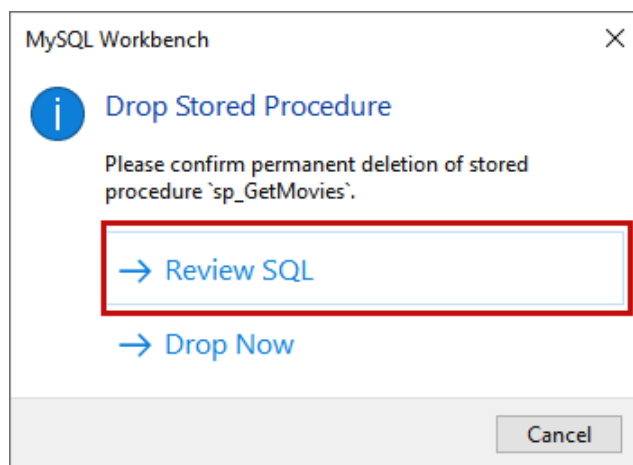


## Drop a Stored Procedure using MySQL workbench wizard

You can use the MySQL workbench wizard to drop the procedure. To drop any procedure, expand **sakila** schema Expand **Stored Procedures** Right-click on **sp\_GetMovies** Click on **Drop Stored Procedure**.



A dialog box opens. You can choose to review the procedure before dropping it, or you can drop it without reviewing it. It is good practice to review the database object before dropping it, so choose Review SQL.



In **Review SQL Code to Execute** dialog box, you can review the drop statement and the object name.

