Java CallableStatement Interface

CallableStatement interface is used to call the **stored procedures and functions**.

We can have business logic on the database by the use of stored procedures and functions that will make the performance better because these are precompiled.

Suppose you need the get the age of the employee based on the date of birth, you may create a function that receives date as the input and returns age of the employee as the output.

What is the difference between stored procedures and functions?

The differences between stored procedures and functions are given below:

Stored Procedure	Function
is used to perform business logic.	is used to perform calculation.
must not have the return type.	must have the return type.
may return 0 or more values.	may return only one values.
We can call functions from the procedure.	Procedure cannot be called from function.
Procedure supports input and output parameters.	Function supports only input parameter.
Exception handling using try/catch block can be used in stored procedures.	Exception handling using try/catch can't be used in user defined functions.

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:

```
[IN | OUT | INOUT] parameter_name datatype[(length)]
```

Code language: SQL (Structured Query Language) (sql)

In this syntax,

- 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.

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:

```
DELIMITER //

CREATE PROCEDURE GetOfficeByCountry(
	IN countryName VARCHAR(255)
)
BEGIN
	SELECT *
	FROM offices
	WHERE country = countryName;
END //

DELIMITER ;
```

Code language: SQL (Structured Query Language) (sql)

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:

```
CALL GetOfficeByCountry('USA');
```

Code language: SQL (Structured Query Language) (sql)

officeCode	city	phone	addressLine1	addressLine2	state	country	postalCode	territory
1	San Francisco	+1 650 219 4782	100 Market Street	Suite 300	CA	USA	94080	NA
2	Boston	+1 215 837 0825	1550 Court Place	Suite 102	MA	USA	02107	NA
3	NYC	+1 212 555 3000	523 East 53rd Street	apt. 5A	NY	USA	10022	NA

To find offices in France, you pass the literal string France to the Get Office By Country stored procedure as follows:

```
CALL GetOfficeByCountry('France')
```

Code language: SQL (Structured Query Language) (sql)

officeCode	city	phone	addressLine1	addressLine2	state	country	postalCode	territory
4	Paris	+33 14 723 4404	43 Rue Jouffroy D'abbans	NULL	NULL	France	75017	EMEA

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

```
CALL GetOfficeByCountry();
```

Code language: SQL (Structured Query Language) (sql)
Here is the error:

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

Code language: JavaScript (javascript)

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

```
DELIMITER $$

CREATE PROCEDURE GetOrderCountByStatus (
        IN  orderStatus VARCHAR(25),
        OUT total INT
)
BEGIN
        SELECT COUNT(orderNumber)
        INTO total
        FROM orders
        WHERE status = orderStatus;
END$$
```

DELIMITER ;
Code language: SQL (Structured Query Language) (sql)
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.

```
CALL GetOrderCountByStatus('Shipped',@total);

SELECT @total;
```

Code language: SQL (Structured Query Language) (sql)

	@total
▶	303

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

```
CALL GetOrderCountByStatus('in process',@total);

SELECT @total AS total_in_process;
```

Code language: SQL (Structured Query Language) (sql)

	total_in_process
▶	6

INOUT

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

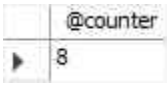
```
DELIMITER $$

CREATE PROCEDURE SetCounter(
        INOUT counter INT,
    IN inc INT
)
BEGIN
        SET counter = counter + inc;
END$$

DELIMITER ;
```

Code language: SQL (Structured Query Language) (sql)
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:

```
SET @counter = 1;
CALL SetCounter(@counter,1); -- 2
CALL SetCounter(@counter,1); -- 3
CALL SetCounter(@counter,5); -- 8
SELECT @counter; -- 8
```
Code language: SQL (Structured Query Language) (sql)
Here is the output:



	@counter
▶	8

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

How to get the instance of CallableStatement?

The prepareCall() method of Connection interface returns the instance of CallableStatement. Syntax is given below:

```
public CallableStatement prepareCall("{ call procedurename(?,?...?)}");
```

The example to get the instance of CallableStatement is given below:

```
CallableStatement stmt=con.prepareCall("{call myprocedure(?,?)}");
```

It calls the procedure myprocedure that receives 2 arguments.

Practical Example

Make one table with the name JDBCStoreTable

```
create Table JDBCStoreTable(id varchar (20), name varchar(200));
```

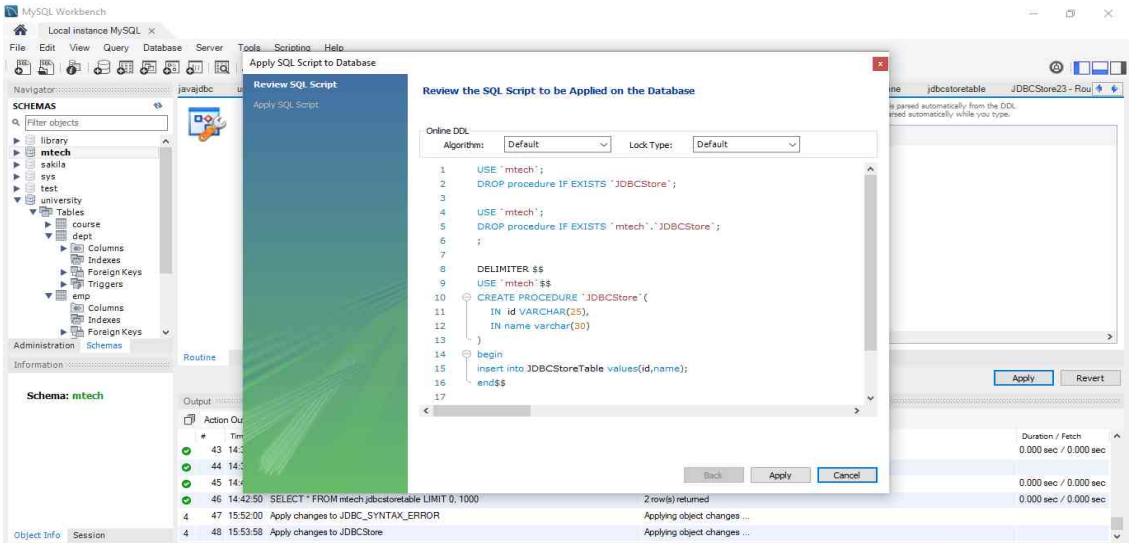
Make Store procedure:

```
CREATE PROCEDURE `JDBCStore23` (  /* stored procedure with two parameters id and name)
        IN  id VARCHAR(25),
        IN name varchar(30)
)
begin
```

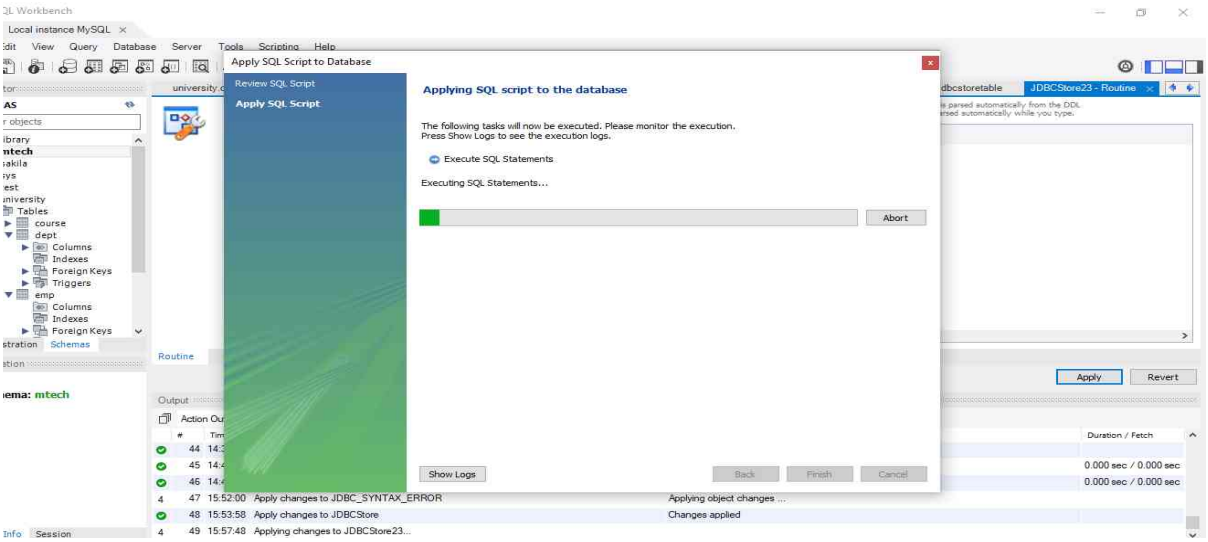
insert into JDBCStoreTable values(id,name); /* insert two values in the table*/

end

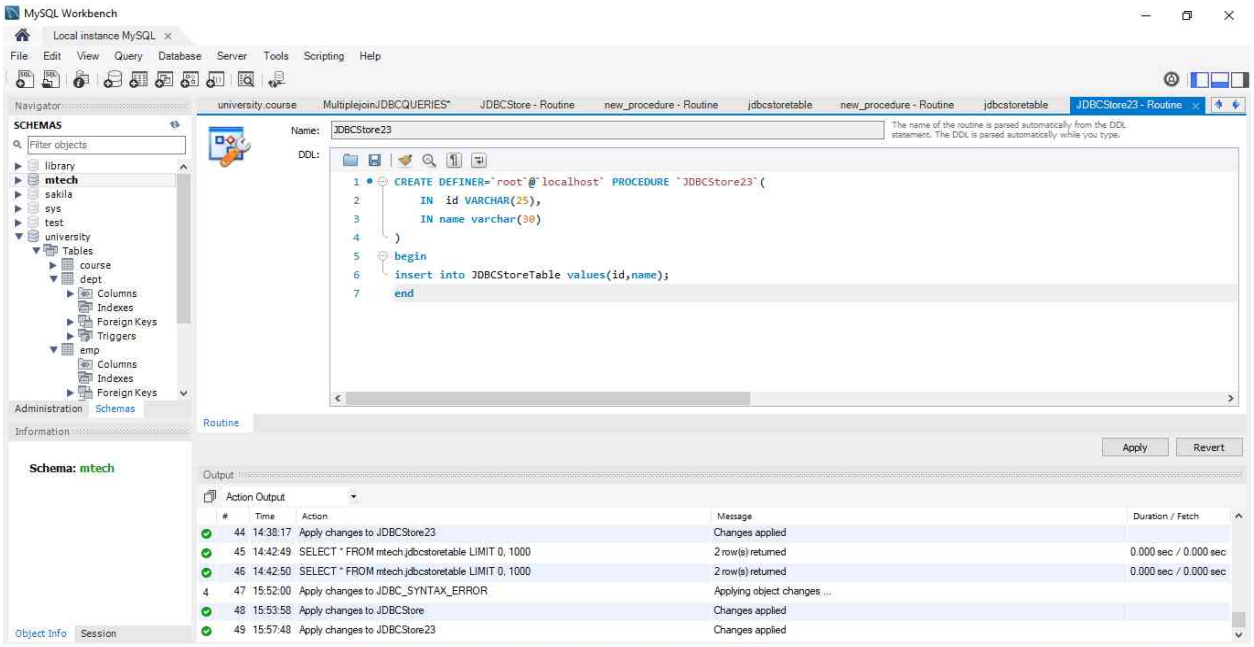
Execute the stored procedure with command ‘Apply’ then you find the screen like



Then again click on the ‘Apply’ but to execute it, the screen will appear like this



And executed procedure will be like this:



Now we will make a program to call this stored procedure with the help of callable interface:

```
import java.sql.*;

public class MYSQLProc {

public static void main(String[] args) throws Exception{

        Connection conn = null;

            //Statement stmt = null;

//Class.forName("oracle.jdbc.driver.OracleDriver");

//Connection con=DriverManager.getConnection(

//"jdbc:oracle:thin:@localhost:1521:xe","system","oracle");

                String userName = "root";

               String password = "Rakesh@84";
```

```
String url = "jdbc:mysql://localhost:3306/mtech";

Class.forName ("com.mysql.cj.jdbc.Driver").newInstance();

conn = DriverManager.getConnection (url, userName, password);

CallableStatement stmt=conn.prepareCall("{call JDBCStore23(?,?)}");

stmt.setInt(1,1013);

stmt.setString(2,"Ani");

stmt.execute();


System.out.println("success");

}

}
```

Output will be like this:
The value will in the database: