CallableStatement in Java Example

APRIL 2, 2018 BY PANKAJ — 13 COMMENTS

CallableStatement in java is used to call stored procedure from java program. **Stored Procedures** are group of statements that we compile in the database for some task. Stored procedures are beneficial when we are dealing with multiple tables with complex scenario and rather than sending multiple queries to the database, we can send required data to the stored procedure and have the logic executed in the database server itself.

Table of Contents [hide]

CallableStatement

JDBC API provides support to execute Stored Procedures through `CallableStatement` interface.

Stored Procedures requires to be written in the database specific syntax and for my tutorial, I will use Oracle database. We will look into standard features of **CallableStatement** with IN and OUT parameters.

Later on we will look into Oracle specific **STRUCT** and **Cursor** examples.

Let's first create a table for our CallableStatement example programs with below SQL query.

`create_employee.sql`

```
-- For Oracle DB
CREATE TABLE EMPLOYEE
  (
    "EMPID"   NUMBER NOT NULL ENABLE,
    "NAME"    VARCHAR2(10 BYTE) DEFAULT NULL,
    "ROLE"    VARCHAR2(10 BYTE) DEFAULT NULL,
    "CITY"    VARCHAR2(10 BYTE) DEFAULT NULL,
    "COUNTRY" VARCHAR2(10 BYTE) DEFAULT NULL,
    PRIMARY KEY ("EMPID")
  );
```

Let's first create a utility class to get the Oracle database Connection object. Make sure Oracle OJDBC jar is in the build path of the project.

`DBConnection.java`

```
package com.journaldev.jdbc.storedproc;
```

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class DBConnection {

	private static final String DB_DRIVER_CLASS =
"oracle.jdbc.driver.OracleDriver";
	private static final String DB_URL = "jdbc:oracle:thin:@localhost:1521:orcl";
	private static final String DB_USERNAME = "HR";
	private static final String DB_PASSWORD = "oracle";

	public static Connection getConnection() {
		Connection con = null;
		try {
			// load the Driver Class
			Class.forName(DB_DRIVER_CLASS);

			// create the connection now
			con
```

CallableStatement Example

Let's write a simple stored procedure to insert data into Employee table.

insertEmployee.sql

```
CREATE OR REPLACE PROCEDURE insertEmployee
(in_id IN EMPLOYEE.EMPID%TYPE,
 in_name IN EMPLOYEE.NAME%TYPE,
 in_role IN EMPLOYEE.ROLE%TYPE,
 in_city IN EMPLOYEE.CITY%TYPE,
 in_country IN EMPLOYEE.COUNTRY%TYPE,
 out_result OUT VARCHAR2)
AS
BEGIN
  INSERT INTO EMPLOYEE (EMPID, NAME, ROLE, CITY, COUNTRY)
  values (in_id,in_name,in_role,in_city,in_country);
  commit;

  out_result := 'TRUE';

EXCEPTION
  WHEN OTHERS THEN
```

```
    out_result := 'FALSE';
    ROLLBACK;
  END;
```

As you can see that insertEmployee procedure is expecting inputs from the caller that will be inserted into the Employee table.

If insert statement works fine, it's returning TRUE and incase of any exception it's returning FALSE.

Let's see how we can use `CallableStatement` to execute `insertEmployee` stored procedure to insert employee data.

`JDBCStoredProcedureWrite.java`

```
package com.journaldev.jdbc.storedproc;

import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.SQLException;
import java.util.Scanner;

public class JDBCStoredProcedureWrite {

        public static void main(String[] args) {
                Connection con = null;
                CallableStatement stmt = null;

                //Read User Inputs
                Scanner input = new Scanner(System.in);
                System.out.println("Enter Employee ID (int):");
                int id = Integer.parseInt(input.nextLine());
                System.out.println("Enter Employee Name:");
                String name = input.nextLine();
                System.out.println("Enter Employee Role:");
                String role = input.nextLine();
                System.out.println("Enter Employee City:");
```

We are reading user input to be stored in Employee table. The only thing different from `PreparedStatement` is the creation of CallableStatement through "`{call insertEmployee(?,?,?,?,?,?)}`" and setting OUT parameter with `CallableStatement registerOutParameter()` method.

We have to register the OUT parameter before executing the stored procedure. Once the stored procedure is executed, we can use `CallableStatement getXXX()` method to get the OUT object data. Notice that while registering the OUT parameter, we need to specify the type of OUT parameter through `java.sql.Types`.

The code is generic in nature, so if we have same stored procedure in other relational database like MySQL, we can execute them with this program too.

Below is the output when we are executing above CallableStatement example program multiple times.

```
Enter Employee ID (int):
1
Enter Employee Name:
Pankaj
Enter Employee Role:
Developer
Enter Employee City:
Bangalore
Enter Employee Country:
India
Employee Record Save Success::TRUE

-----
Enter Employee ID (int):
2
Enter Employee Name:
Pankaj Kumar
Enter Employee Role:
CEO
Enter Employee City:
San Jose
Enter Employee Country:
```

Notice that second execution failed because name passed is bigger than the column size. We are consuming the exception in the stored procedure and returning false in this case.

CallableStatement Example – Stored Procedure OUT Parameters

Now let's write a stored procedure to get the employee data by id. User will enter the employee id and program will display the employee information.

`getEmployee.sql`

```
create or replace
PROCEDURE getEmployee
(in_id IN EMPLOYEE.EMPID%TYPE,
 out_name OUT EMPLOYEE.NAME%TYPE,
 out_role OUT EMPLOYEE.ROLE%TYPE,
 out_city OUT EMPLOYEE.CITY%TYPE,
 out_country OUT EMPLOYEE.COUNTRY%TYPE
 )
AS
BEGIN
  SELECT NAME, ROLE, CITY, COUNTRY
  INTO out_name, out_role, out_city, out_country
  FROM EMPLOYEE
  WHERE EMPID = in_id;

END;
```

Java CallableStatement example program using getEmployee stored procedure to read the employee data is;

JDBCStoredProcedureRead.java

```
package com.journaldev.jdbc.storedproc;

import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.SQLException;
import java.util.Scanner;

public class JDBCStoredProcedureRead {

        public static void main(String[] args) {
                Connection con = null;
                CallableStatement stmt = null;

                //Read User Inputs
                Scanner input = new Scanner(System.in);
                System.out.println("Enter Employee ID (int):");
                int id = Integer.parseInt(input.nextLine());

                try{
                        con = DBConnection.getConnection();
```

```
stmt = con.prepareCall("{call getEmployee(?,?,?,?,?)}");
stmt.setInt(1, id);
```

Again the program is generic and works for any database having same stored procedure. Let's see what is the output when we execute the above CallableStatement example program.

```
Enter Employee ID (int):
1
Employee Name=Pankaj,Role=Developer,City=Bangalore,Country=India
```

CallableStatement Example – Stored Procedure Oracle CURSOR

Since we are reading the employee information through ID, we are getting single result and OUT parameters works well to read the data. But if we search by role or country, we might get multiple rows and in that case we can use Oracle CURSOR to read them like result set.

getEmployeeByRole.sql

```
create or replace
PROCEDURE getEmployeeByRole
(in_role IN EMPLOYEE.ROLE%TYPE,
 out_cursor_emps OUT SYS_REFCURSOR
 )
AS
BEGIN
  OPEN out_cursor_emps FOR
  SELECT EMPID, NAME, CITY, COUNTRY
  FROM EMPLOYEE
  WHERE ROLE = in_role;

END;
```

JDBCStoredProcedureCursor.java

```
package com.journaldev.jdbc.storedproc;

import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.Scanner;
```

```
import oracle.jdbc.OracleTypes;

public class JDBCStoredProcedureCursor {

        public static void main(String[] args) {

                Connection con = null;
                CallableStatement stmt = null;
                ResultSet rs = null;

                //Read User Inputs
                Scanner input = new Scanner(System.in);
                System.out.println("Enter Employee Role:");
                String role = input.nextLine();
```

This program is using Oracle OJDBC specific classes and won't work with other database. We are setting OUT parameter type as `OracleTypes.CURSOR` and then casting it to `ResultSet` object. Other part of the code is simple JDBC programming.

When we execute above CallableStatement example program, we get below output.

```
Enter Employee Role:
Developer
Employee ID=5,Name=Kumar,Role=Developer,City=San Jose,Country=USA
Employee ID=1,Name=Pankaj,Role=Developer,City=Bangalore,Country=India
```

Your output may vary depending on the data in your Employee table.

CallableStatement Example – Oracle DB Object and STRUCT

If you look at the `insertEmployee` and `getEmployee` stored procedures, I am having all the parameters of the Employee table in the procedure. When number of column grows, this can lead to confusion and more error prone. Oracle database provides option to create database Object and we can use Oracle STRUCT to work with them.

Let's first define Oracle DB object for Employee table columns.

`EMPLOYEE_OBJ.sql`

```
create or replace TYPE EMPLOYEE_OBJ AS OBJECT
(
  EMPID NUMBER,
  NAME VARCHAR2(10),
```

```
    ROLE VARCHAR2(10),
    CITY  VARCHAR2(10),
    COUNTRY  VARCHAR2(10)

    );
```

Now let's rewrite the insertEmployee stored procedure using EMPLOYEE_OBJ.

insertEmployeeObject.sql

```
  CREATE OR REPLACE PROCEDURE insertEmployeeObject
  (IN_EMPLOYEE_OBJ IN EMPLOYEE_OBJ,
   out_result OUT VARCHAR2)
  AS
  BEGIN
    INSERT INTO EMPLOYEE (EMPID, NAME, ROLE, CITY, COUNTRY) values
    (IN_EMPLOYEE_OBJ.EMPID, IN_EMPLOYEE_OBJ.NAME, IN_EMPLOYEE_OBJ.ROLE,
  IN_EMPLOYEE_OBJ.CITY, IN_EMPLOYEE_OBJ.COUNTRY);
    commit;

    out_result := 'TRUE';

  EXCEPTION
    WHEN OTHERS THEN
    out_result := 'FALSE';
    ROLLBACK;
  END;
```

Let's see how we can call `insertEmployeeObject` stored procedure in java program.

JDBCStoredProcedureOracleStruct.java

```
  package com.journaldev.jdbc.storedproc;

  import java.sql.Connection;
  import java.sql.SQLException;
  import java.util.Scanner;

  import oracle.jdbc.OracleCallableStatement;
  import oracle.sql.STRUCT;
  import oracle.sql.StructDescriptor;
```

```
public class JDBCStoredProcedureOracleStruct {

	public static void main(String[] args) {
		Connection con = null;
		OracleCallableStatement stmt = null;

		//Create Object Array for Stored Procedure call
		Object[] empObjArray = new Object[5];
		//Read User Inputs
		Scanner input = new Scanner(System.in);
		System.out.println("Enter Employee ID (int):");
		empObjArray[0] = Integer.parseInt(input.nextLine());
```

First of all we are creating an Object array of same length as the EMPLOYEE_OBJ database object. Then we are setting values according to the EMPLOYEE_OBJ object variables. This is very important otherwise the data will get inserted into wrong columns.

Then we are creating `oracle.sql.STRUCT` object with the help of `oracle.sql.StructDescriptor` and our Object array. Once the STRUCT object is created, we are setting it as IN parameter for the stored procedure, register the OUT parameter and executing it. This code is tightly couple with OJDBC API and will not work for other databases.

Here is the output when we are executing this program.

```
Enter Employee ID (int):
5
Enter Employee Name:
Kumar
Enter Employee Role:
Developer
Enter Employee City:
San Jose
Enter Employee Country:
USA
Employee Record Save Success::TRUE
```

We can use the Database object as OUT parameter also and read it to get the values from database.

That's all for CallableStatement in java example to execute Stored Procedures, I hope you learned something from it.

« PREVIOUS
JDBC Batch insert update MySQL Oracle

NEXT »
Java DataSource, JDBC DataSource Example

About Pankaj

If you have come this far, it means that you liked what you are reading. Why not reach little more and connect with me directly on **Google Plus**, **Facebook** or **Twitter**. I would love to hear your thoughts and opinions on my articles directly.

Recently I started creating video tutorials too, so do check out my videos on **Youtube**.

FILED UNDER: DATABASE, JAVA

Comments

Renaud91 says
JULY 24, 2018 AT 10:25 PM

Nice sample,
you should code your finally ;
}finally{
try {
if (rs != null) {rs.close();}
} catch (SQLException e) {
e.printStackTrace();
}
try {
if (stmt != null) {stmt.close();}
} catch (SQLException e) {
e.printStackTrace();
}
try {
if (con != null) {con.close();}
} catch (SQLException e) {
e.printStackTrace();
}
try {
input.close();
} catch (SQLException e) {
e.printStackTrace();
}
}
Or use ressource writing
(https://docs.oracle.com/javase/tutorial/essential/exceptions/tryResourceClose.html).
Reply

Osiel says
SEPTEMBER 8, 2016 AT 7:46 AM
Hi!
The example of struct is equals for %ROWTYPE? when you declare in parameter of IN;
CREATE OR REPLACE PROCEDURE insertEmployeeObject
(IN_EMPLOYEE_OBJ IN EMPLOYEE%ROWTYPE,
out_result OUT VARCHAR2)
AS
…..
Reply

Ankit says

OCTOBER 1, 2015 AT 2:45 AM

Hi Pankaj,

Is there a way to get the name of the cursor at the java end, like here u have used out_cursor_emps as the cursor name.

I have used resulset.getCursorName(), but it does not help.

Reply

suresh says

SEPTEMBER 22, 2015 AT 11:01 AM

best tutorial

Reply

Damla says

APRIL 14, 2015 AT 4:14 AM

Thank you so much, it really really helps me a lot !

Reply

Maruthi says

FEBRUARY 11, 2015 AT 2:37 AM

Excellent job….thanks a lot!! You really helped me understand IN, OUT parameters of Stored Procedures.

Reply

Sam says

OCTOBER 19, 2014 AT 8:53 PM

Thanks!! It helped me very much!

Reply

Jayasimha says

JULY 9, 2014 AT 1:50 AM

I did not see any explanation till now which is so simple and clear as the above. Thanks Pankaj…

Reply

Sreedhar Shastry says

MAY 30, 2014 AT 10:48 AM

Dear Pankaj,

Its awesome explanation. I was training somebody, I found this tutorial so interesting and alluring.

Though I have added many more explanatory comments to my trainee, this was a very good starter.

Thanx from bottom of my heart.

Reply

Akram Khan says

MAY 20, 2014 AT 12:25 AM

Pankaj, first all thanks for such a brief and helpful tutorial !!

I have a situation where my stored procedure is receiving 2 string arrays as input, and 1 output as array of objects. Great if you can help with an example where the output is array of object.

Below is my sample code..

Procedure on SQL part :

```
create or replace
REPORT_ROWS(sls_array in SLS_ARRAY ,sp_array in SP_ARRAY,report_tabtype out
LATEST_REPORT_TABTYPE)
```

// JDBC part

```
CallableStatement callStmt = conn.prepareCall("{ call REPORT_ROWS(?,?,?)}");
ArrayDescriptor des_aray1 = ArrayDescriptor.createDescriptor("schemaName.SLS_ARRAY", conn);
ARRAY sls_array = new ARRAY(des_aray1,conn,SLS_ARRAY_J);
ArrayDescriptor des_aray2 = ArrayDescriptor.createDescriptor("schemaName.SP_ARRAY", conn);
ARRAY sp_array = new ARRAY(des_aray2,conn,SP_ARRAY_J);
callStmt.setArray(1, sls_array);
callStmt.setArray(2, sp_array);
callStmt.registerOutParameter(3, Types.ARRAY, "schemaName.LATEST_REPORT_TABTYPE");
System.out.println(callStmt.execute());
```

OUTPUT : false as no value is returned.

Reply

Akram Khan says

MAY 20, 2014 AT 2:14 AM

I feels i am doing something wrong at :

```
callStmt.registerOutParameter(3, Types.ARRAY, "schemaName.LATEST_REPORT_TABTYPE");
```

The Types.ARRAY or OracleTypes.ARRAY are of int type, but i think it should be of Object type array.

May be i need to create a class with member variable exactly as the stored procedure's out object.

And then if possible use it in place of Types.ARRAY. what do you suggest ?

Reply

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

Name *

Email *

Save my name, email, and website in this browser for the next time I comment.

POST COMMENT

Search for tutorials...

DOWNLOAD ANDROID APP

CORE JAVA TUTORIAL

Java 10 Tutorials Java 9 Tutorials Java 8 Tutorials Java 7 Tutorials Core Java Basics OOPS Concepts Data

Types and Operators String Manipulation
Java Arrays Annotation and Enum
Java Collections Java IO Operations
Java Exception Handling
MultiThreading and Concurrency
Regular Expressions Advanced Java Concepts

RECOMMENDED TUTORIALS

Java Tutorials

- Java IO
- Java Regular Expressions
- Multithreading in Java
- Java Logging
- Java Annotations
- Java XML
- Collections in Java
- Java Generics
- Exception Handling in Java
- Java Reflection
- Java Design Patterns
- JDBC Tutorial

Java EE Tutorials

- Servlet JSP Tutorial
- Struts2 Tutorial
- Spring Tutorial
- Hibernate Tutorial
- Primefaces Tutorial
- Apache Axis 2
- JAX-RS
- Memcached Tutorial