SQL STORED PROCEDURE IN SPRING

The **SimpleJdbcCall** class can be used to call a stored procedure with IN and OUT parameters. You can use this approach while working with either of the RDBMS like Apache Derby, DB2, MySQL, Microsoft SQL Server, Oracle, and Sybase.

To understand the approach let us take our Student table which can be created in MySQL TEST database with the following DDL:

```
CREATE TABLE Student(
   ID   INT NOT NULL AUTO_INCREMENT,
   NAME VARCHAR(20) NOT NULL,
   AGE  INT NOT NULL,
   PRIMARY KEY (ID)
);
```

Next, consider the following MySQL stored procedure which takes student Id and returns corresponding student's name and age using OUT parameters. So let us create this stored procedure in your TEST database using MySQL command prompt:

```
DELIMITER $$

DROP PROCEDURE IF EXISTS `TEST`.`getRecord` $$
CREATE PROCEDURE `TEST`.`getRecord` (
IN in_id INTEGER,
OUT out_name VARCHAR(20),
OUT out_age  INTEGER)
BEGIN
   SELECT name, age
   INTO out_name, out_age
   FROM Student where id = in_id;
END $$

DELIMITER ;
```

Now let us write our Spring JDBC application which will implement simple Create and Read operations on our Student table. Let us have working Eclipse IDE in place and follow the following steps to create a Spring application:

Step	Description
1	Create a project with a name *SpringExample* and create a package *com.tutorialspoint* under the **src** folder in the created project.
2	Add required Spring libraries using *Add External JARs* option as explained in the *Spring Hello World Example* chapter.
3	Add Spring JDBC specific latest libraries **mysql-connector-java.jar**, **org.springframework.jdbc.jar** and **org.springframework.transaction.jar** in the project. You can download required libraries if you do not have them already.
4	Create DAO interface *StudentDAO* and list down all the required methods. Though it is not required and you can directly write *StudentJDBCTemplate* class, but as a good practice, let's do it.
5	Create other required Java classes *Student*, *StudentMapper*, *StudentJDBCTemplate* and *MainApp* under the *com.tutorialspoint* package.
6	Make sure you already created **Student** table in TEST database. Also make sure your MySQL server is working fine and you have read/write access on the database using the given username and password.

7	Create Beans configuration file *Beans.xml* under the **src** folder.
8	The final step is to create the content of all the Java files and Bean Configuration file and run the application as explained below.

Following is the content of the Data Access Object interface file **StudentDAO.java**:

```
package com.tutorialspoint;

import java.util.List;
import javax.sql.DataSource;

public interface StudentDAO {
   /**
    * This is the method to be used to initialize
    * database resources ie. connection.
    */
   public void setDataSource(DataSource ds);
   /**
    * This is the method to be used to create
    * a record in the Student table.
    */
   public void create(String name, Integer age);
   /**
    * This is the method to be used to list down
    * a record from the Student table corresponding
    * to a passed student id.
    */
   public Student getStudent(Integer id);
   /**
    * This is the method to be used to list down
    * all the records from the Student table.
    */
   public List<Student> listStudents();

}
```

Following is the content of the **Student.java** file:

```
package com.tutorialspoint;

public class Student {
   private Integer age;
   private String name;
   private Integer id;

   public void setAge(Integer age) {
      this.age = age;
   }
   public Integer getAge() {
      return age;
   }

   public void setName(String name) {
      this.name = name;
   }
   public String getName() {
      return name;
   }

   public void setId(Integer id) {
      this.id = id;
   }
   public Integer getId() {
      return id;
   }
}
```

Following is the content of the **StudentMapper.java** file:

```
package com.tutorialspoint;

import java.sql.ResultSet;
import java.sql.SQLException;
import org.springframework.jdbc.core.RowMapper;

public class StudentMapper implements RowMapper<Student> {
   public Student mapRow(ResultSet rs, int rowNum) throws SQLException {
      Student student = new Student();
      student.setId(rs.getInt("id"));
      student.setName(rs.getString("name"));
      student.setAge(rs.getInt("age"));
      return student;
   }
}
```

Following is the implementation class file **StudentJDBCTemplate.java** for the defined DAO interface StudentDAO:

```
package com.tutorialspoint;

import java.util.Map;

import javax.sql.DataSource;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.core.namedparam.MapSqlParameterSource;
import org.springframework.jdbc.core.namedparam.SqlParameterSource;
import org.springframework.jdbc.core.simple.SimpleJdbcCall;

public class StudentJDBCTemplate implements StudentDAO {
   private DataSource dataSource;
   private SimpleJdbcCall jdbcCall;

   public void setDataSource(DataSource dataSource) {
      this.dataSource = dataSource;
      this.jdbcCall =  new SimpleJdbcCall(dataSource).
                       withProcedureName("getRecord");
   }

   public void create(String name, Integer age) {
      JdbcTemplate jdbcTemplateObject = new JdbcTemplate(dataSource);
      String SQL = "insert into Student (name, age) values (?, ?)";

      jdbcTemplateObject.update( SQL, name, age);
      System.out.println("Created Record Name = " + name + " Age = " + age);
      return;
   }

   public Student getStudent(Integer id) {
      SqlParameterSource in = new MapSqlParameterSource().
                              addValue("in_id", id);
      Map<String, Object> out = jdbcCall.execute(in);

      Student student = new Student();
      student.setId(id);
      student.setName((String) out.get("out_name"));
      student.setAge((Integer) out.get("out_age"));

      return student;
   }

   public List<Student> listStudents() {
      String SQL = "select * from Student";

      List <Student> students = jdbcTemplateObject.query(SQL,
```

```
                                        new StudentMapper());
      return students;
   }

}
```

Few words about above program: The code you write for the execution of the call involves creating an *SqlParameterSource* containing the IN parameter. It's important to match the name provided for the input value with that of the parameter name declared in the stored procedure. The *execute* method takes the IN parameters and returns a Map containing any out parameters keyed by the name as specified in the stored procedure. Now let us move with the main application file **MainApp.java**, which is as follows:

```
package com.tutorialspoint;
import java.util.List;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import com.tutorialspoint.StudentJDBCTemplate;

public class MainApp {
   public static void main(String[] args) {
      ApplicationContext context =
             new ClassPathXmlApplicationContext("Beans.xml");

      StudentJDBCTemplate studentJDBCTemplate =
      (StudentJDBCTemplate)context.getBean("studentJDBCTemplate");

      System.out.println("------Records Creation--------" );
      studentJDBCTemplate.create("Zara", 11);
      studentJDBCTemplate.create("Nuha", 2);
      studentJDBCTemplate.create("Ayan", 15);

      System.out.println("------Listing Multiple Records--------" );
      List<Student> students = studentJDBCTemplate.listStudents();
      for (Student record : students) {
         System.out.print("ID : " + record.getId() );
         System.out.print(", Name : " + record.getName() );
         System.out.println(", Age : " + record.getAge());
      }

      System.out.println("----Listing Record with ID = 2 -----" );
      Student student = studentJDBCTemplate.getStudent(2);
      System.out.print("ID : " + student.getId() );
      System.out.print(", Name : " + student.getName() );
      System.out.println(", Age : " + student.getAge());

   }
}
```

Following is the configuration file **Beans.xml**:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd ">

   <!-- Initialization for data source -->
   <bean
      >
      <property name="driverClassName" value="com.mysql.jdbc.Driver"/>
      <property name="url" value="jdbc:mysql://localhost:3306/TEST"/>
      <property name="username" value="root"/>
      <property name="password" value="password"/>
   </bean>

   <!-- Definition for studentJDBCTemplate bean -->
   <bean
```

```
      >
      <property name="dataSource"  ref="dataSource" />
   </bean>

</beans>
```

Once you are done with creating source and bean configuration files, let us run the application. If everything is fine with your application, this will print the following message:

```
------Records Creation--------
Created Record Name = Zara Age = 11
Created Record Name = Nuha Age = 2
Created Record Name = Ayan Age = 15
------Listing Multiple Records--------
ID : 1, Name : Zara, Age : 11
ID : 2, Name : Nuha, Age : 2
ID : 3, Name : Ayan, Age : 15
----Listing Record with ID = 2 -----
ID : 2, Name : Nuha, Age : 2
```