Spring JDBC Example

Last Updated : 23 Jul, 2025

Spring JDBC (Java Database Connectivity) is a powerful module in the Spring Framework that simplifies database interaction by eliminating boilerplate code required for raw JDBC. It provides an abstraction over JDBC, making database operations more efficient, less error-prone, and easier to manage. This article will guide you through **setting up a Spring JDBC application**, demonstrating **how to connect and interact with a MySQL database using the Spring Framework.**

Prerequisite:

- Basic understanding of JDBC
- Installed MySQL database
- Java IDE (IntelliJ IDEA or Eclipse)
- Spring and MySQL Connector JAR files

Step-by-Step Implementation

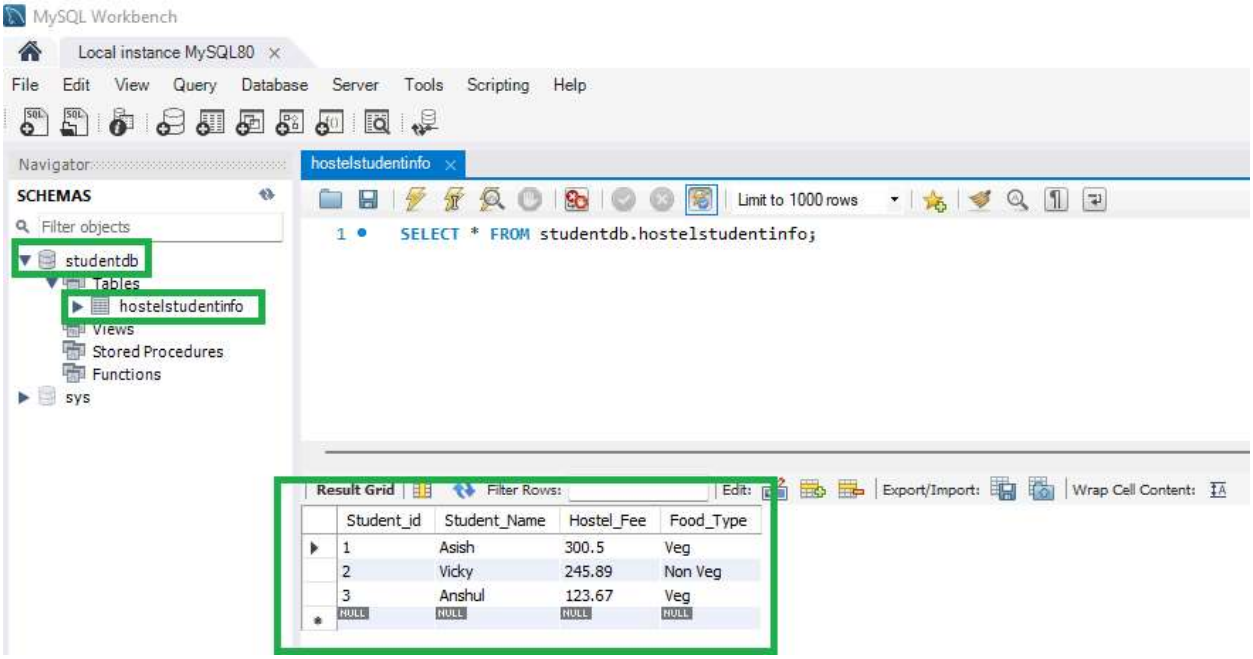
Step 1: Create a Java Project

Create a simple Java project in your preferred IDE (IntelliJ IDEA or Eclipse). You may refer to these articles:

- Creating First Java Application in IntelliJ IDEA
- How to Install Eclipse IDE For Java?

Step 2: Create a Database and Tables

Create the required tables inside your **MySQL** database. In this article, we have used the MySQL database. And the following data has been present inside our Database.



So here, **studentdb** is our schema name and **a** is the table name. Similarly, you can create your own schema and table and put some data inside that table manually. You may refer to these articles:

- How to Install MySQL on Windows?
- How to Install SQL Workbench For MySQL on Windows?

Step 3: Create StudentDAO Class

Go to the Java project and create one class named **StudentDAO** and inside the class, we are going to create a single method **selectAllRows()** to fetch all the data present inside our MySQL database. We are also going to declare our four most important attributes to connect our Java program with the MySQL server.

- Driver
- URL
- User
- Password

Example:

```
// Java Program to fetch All Data Present Inside MySQL DB

// Importing required classes
import java.sql.*;

// Main class
public class StudentDAO {

    // Class data members
    private String driver;
    private String url;
    private String userName;
    private String password;

    // Setter methods for dependency injection
    public void setDriver(String driver) {
        this.driver = driver;
    }
    public void setUrl(String url) {
        this.url = url;
    }
    public void setUserName(String userName) {
        this.userName = userName;
    }
    public void setPassword(String password) {
        this.password = password;
    }

    // Method to fetch all student records
    public void selectAllRows() throws ClassNotFoundException, SQLException
{
        System.out.println("Retrieving all student data..");

        // Load driver
        Class.forName(driver);

        // Establish connection
        Connection con = DriverManager.getConnection(url, userName,
password);

        // Execute query
        Statement stmt = con.createStatement();
        ResultSet rs = stmt.executeQuery("SELECT * FROM
studentdb.hostelstudentinfo");

        while (rs.next()) {
            int studentId = rs.getInt(1);
            String studentName = rs.getString(2);
            double hostelFees = rs.getDouble(3);
            String foodType = rs.getString(4);

            System.out.println(studentId + " " + studentName + " " +
hostelFees + " " + foodType);
```

```
        }

        // Close connection
        con.close();
    }
}
```

Step 4: Add External JAR Files

Now we have to Add the External JAR Files to an IntelliJ IDEA Project. A JAR (Java Archive) is a package file format typically used to aggregate many Java class files and associated metadata and resources (text, images, etc.) into one file to distribute application software or libraries on the Java platform. In simple words, a JAR file is a file that contains a compressed version of .class files, audio files, image files, or directories. We have to add the following external jar files to our Java project

- Spring
- MySQL Connector

You may refer to this article How to Add External JAR File to an IntelliJ IDEA Project?.

Step 5: Configure Spring Beans (beans.xml)

Let's create the bean of StudentDAO class inside the beans.xml file and inject the values of the properties by setter injection. You may refer to this article Spring – Injecting Literal Values By Setter Injection. Below is the code for the beans.xml file.

Spring Configuration (beans.xml):

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

    <bean id="studentDAO" class="StudentDAO">
```

```
        <property name="driver" value="com.mysql.cj.jdbc.Driver"/>
        <property name="url" value="jdbc:mysql://localhost:3306/studentdb"/>
        <property name="userName" value="root"/>
        <property name="password" value="your_password"/>
    </bean>

</beans>
```

Step 6: Create the Main Class to Run the Application

Create the Main class and let's test our application is running fine or not. Below is the code for the **Main.java** file.

```
// Importing required classes
import java.sql.SQLException;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

// Main class
public class Main {
    public static void main(String[] args) throws SQLException,
ClassNotFoundException {

        // Initialize Spring Application Context
        ApplicationContext context = new
ClassPathXmlApplicationContext("beans.xml");

        // Retrieve bean
        StudentDAO studentDAO = context.getBean("studentDAO",
StudentDAO.class);

        // Call method to fetch student records
        studentDAO.selectAllRows();
    }
}
```

Output: After running the program, the following output will be displayed:

```
Retrieving all student data..
1 Asish 300.5 Veg
2 Vicky 245.89 Non Veg
3 Anshul 123.67 Veg
```