Spring - ResultSetExtractor

Last Updated : 23 Jul, 2025

Spring Framework is a powerful and widely used tool for building **Java applications**. With the evolution of **Spring Boot, JDBC**, and modern Java features, working with databases has become easier. In this article, we will discuss how to use the **ResultSetExtractor interface with Spring JDBC** to fetch records from a database.

What is ResultSetExtractor?

ResultSetExtractor is an interface that is used to fetch the records from the database. It is particularly useful when we need to map an entire ResultSet (e.g., multiple rows or complex relationships) into a single object or collection.

Note: The **query()** method of JdbcTemplate accepts a ResultSetExtractor implementation to fetch and transform data.

Syntax of query():

```
public T query(String sqlQuery, ResultSetExtractor<T>
resultSetExtractor);
```

To fetch the data using ResultSetExtractor, we need to implement the **ResultSetExtractor interface** and provide the definition for its method. It has only one method. i.e., **extractData()** which takes an instance of **ResultSet** as an argument and returns the list.

Syntax of extractData():

public T extractData(ResultSet resultSet) throws SQL Exception, DataAccessException

Prerequisites:

- **Java 21 or later:** Modern Java features like records, sealed classes, and pattern matching are used.
- **Spring Boot:** The latest version of Spring Boot simplifies configuration and dependency management.
- **A Database:** We'll use a simple Student table in a MySQL database.
- **Build Tool:** Maven or Gradle (we'll use Maven in this example).

Step-by-Step Implementation

Step 1: Define the Database Schema

Let's create a Student table with the following schema:

```
CREATE TABLE Student (
    id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(45) NOT NULL,
    department VARCHAR(45) NOT NULL
);
```

Insert the sample data:

```
INSERT INTO Student (name, department) VALUES
('geek', 'computer science');
```

Step 2: Add Dependencies

Add the following dependencies in the **pom.xml file.**

pom.xml:

```
<project xmlns="https://maven.apache.org/POM/4.0.0"
         xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="https://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.example</groupId>
    <artifactId>resultset-extractor-demo</artifactId>
    <version>1.0.0</version>

    <properties>
        <java.version>21</java.version>
        <spring-boot.version>3.2.0</spring-boot.version>
    </properties>

    <dependencies>
        <!-- Spring Boot Starter JDBC -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-jdbc</artifactId>
            <version>${spring-boot.version}</version>
        </dependency>

        <!-- MySQL Connector -->
        <dependency>
            <groupId>com.mysql</groupId>
            <artifactId>mysql-connector-j</artifactId>
            <version>8.0.33</version>
            <scope>runtime</scope>
        </dependency>

        <!-- Spring Boot Starter Test -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <version>${spring-boot.version}</version>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <!-- Spring Boot Maven Plugin -->
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
                <version>${spring-boot.version}</version>
            </plugin>
        </plugins>
```

```
    </build>
</project>
```

Step 3: Create a Model Class

Now, we will create a model class for our students.

Student.java:

```
public record Student(int id, String name, String department) {}
```

Step 4: Create a DAO Interface

Now, we will create an interface and name it is **StudentDao**, which we will use to access data from the database of data stoarage. We need to define **getAllStudentDetails()** method which will return all the details of the student.

```
import java.util.List;

import com.geeksforgeeks.model;

public interface StudentDao {
    // This method will return all
    // the details of the students
    List<Student> getAllStudents();

}
```

Step 5: Implement the DAO with ResultSetExtractor

In this step, we will create an implementation class **StudentDaoImpl.java**. This class implements the **StudentDao interface** and provides the definition to the **getAllStudentDetails()** method of the StudentDao

interface. In this class, we will also implement the **ResultSetExtractor** interface and provide the definition of its **extractData()** method.

StudentDaoImpl.java:

```
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.core.ResultSetExtractor;
import org.springframework.stereotype.Repository;

import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;

@Repository
public class StudentDaoImpl implements StudentDao {

    private final JdbcTemplate jdbcTemplate;

    public StudentDaoImpl(JdbcTemplate jdbcTemplate) {
        this.jdbcTemplate = jdbcTemplate;
    }

    @Override
    public List<Student> getAllStudents() {
        String sql = "SELECT id, name, department FROM Student";

        return jdbcTemplate.query(sql, new StudentResultSetExtractor());
    }

    private static class StudentResultSetExtractor implements
ResultSetExtractor<List<Student>> {
        @Override
        public List<Student> extractData(ResultSet rs) throws SQLException {
            List<Student> students = new ArrayList<>();
            while (rs.next()) {
                int id = rs.getInt("id");
                String name = rs.getString("name");
                String department = rs.getString("department");
                students.add(new Student(id, name, department));
            }
            return students;
        }
    }
}
```

Step 6: Configure the Application

Add the database configuration in application.yml file.

application.yml:

```
spring:
  datasource:
    url: ${DATABASE_URL}
    username: ${DATABASE_USERNAME}
    password: ${DATABASE_PASSWORD}
    driver-class-name: com.mysql.cj.jdbc.Driver
```

Step 7: Create the Main Application Class

Create the **SpringBootApplication class** to bootstrap the application.

ResultSetExtractorApplication.java:

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class ResultSetExtractorApplication {
    public static void main(String[] args) {
        SpringApplication.run(ResultSetExtractorApplication.class, args);
    }
}
```

Step 8: Test the Application

Write an integration test using **JUnit 6**.

StudentDaoTest.java:

```
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;

import java.util.List;

import static org.junit.jupiter.api.Assertions.assertFalse;
```

```
@SpringBootTest
public class StudentDaoTest {

    @Autowired
    private StudentDao studentDao;

    @Test
    public void testGetAllStudents() {
        List<Student> students = studentDao.getAllStudents();
        assertFalse(students.isEmpty());
        students.forEach(System.out::println);
    }
}
```

Step 9: Run the Application

Run the application using the following command:

mvn spring-boot:run

Output: