01: Spring JDBC Introduction

Spring JDBC simplifies the use of JDBC (Java Database Connectivity) by providing an abstraction through the JDBC Template class.

👉 JDBC Steps Without Spring:

In JDBC, the following steps are required to interact with a database:

1. **Load the Driver:** Load the JDBC driver for the database being used.
2. **Define Connection URL:** Define the URL where the database is hosted along with credentials.
3. **Establish Connection:** Establish a connection to the database using DriverManager.
4. **Create Statement Object:** Create a Statement or PreparedStatement object to execute queries.
5. **Execute Query:** Execute the query and retrieve the result using ResultSet.
6. **Process the Result:** Process the ResultSet to fetch the required data.
7. **Close the Connection:** Close the connection, statement, and result set to avoid memory leaks.

👉 Spring JDBC Template:

The JDBC Template in Spring simplifies the process of working with databases by reducing boilerplate code. It internally handles:

- **Connection with database**: It manages connection to the database using a DataSource.
- **Query Execution:** SQL queries can be executed directly using provided methods (queryForObject, query, update, etc.).
- **Output Retrieval:** Retrieve results from the query in the form of objects, lists, or directly as primitive data types.

👉 Datasource in Spring JDBC:

- DataSource is used by Spring JDBC to provide a connection to the database. It acts as a factory for Connection objects.
- Spring can manage the DataSource using connection pooling, improving the application's performance by reusing connections.

👉 In-Memory Database (H2):

- Spring Boot provides built-in support for the H2 database, which is an in-memory database.
- By default in H2, the data is stored temporarily in memory, and it is lost when the application is stopped or restarted.
- H2 is ideal for development and testing environments where persistent storage is not required.

02 – Creating A Spring JDBC Project

Create a Spring Boot project using Spring Initializr or with any preferred IDE with the following steps:

1. Go to Spring Initializr.
2. Give the all project details.
3. Select the following dependencies:
    - **JDBC API**: It provides the necessary tools to interact with the database using Spring JDBC.
    - **H2 Database**: An in-memory database that stores data during runtime but clears the data once the application is stopped.
4. Generate the project and import it in the IDE.

Dependencies:

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-jdbc</artifactId>
</dependency>

<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <scope>runtime</scope>
</dependency>
```

In a Spring JDBC project, a class represents a table in the database, and each object of the class represents a row in that table.

Example:

Application.java

```
package com.example.SpringJDBC;

@SpringBootApplication
public class Application {

    public static void main(String[] args) {

        ApplicationContext context = SpringApplication.run(Application.class, args);

        Student s = context.getBean(Student.class);

        s.setRollNo(101);

        s.setName("Navin");

        s.setMarks(78);



        addStudent(s);

    }

    public static void addStudent(Student student){

    }

}
```

Student.java

```
package com.example.SpringJDBC.model;

@Component
@Scope("prototype")
public class Student {

    private int rollNo;

    private String name;

    private int marks;

    public int getRollNo() {

        return rollNo;

    }

    public void setRollNo(int rollNo) {

        this.rollNo = rollNo;

    }

    public String getName() {

        return name;

    }

    public void setName(String name) {

        this.name = name;

    }

    public int getMarks() {

        return marks;

    }

    public void setMarks(int marks) {
        this.marks = marks;
    }
}
```

Code Link:

https://github.com/navinreddy20/spring6-course/tree/c6690e4f2c70d8f530d70623f13d14ff0ffd7e7d/4%20Spring%20JDBC/4.2%20Creating%20A%20Spring%20Jdbc%20Project

03 - Student Service and Repository

In the Spring Boot project, we have implemented a basic structure following the MVC pattern, that utilizes Spring JDBC for database interaction. The application demonstrates:

1. **Model Layer**: Represents the entity that maps to a table in the database.
2. **Repository Layer**: Manages interactions with the database for saving and retrieving data.
3. **Service Layer**: Contains the business logic, interacting with the repository.
4. **Application Layer**: Acts as the entry point and coordinates the process of adding and retrieving students.

Example:

Student.java:

```
package com.telusko.SpringJDBCEx.model;

@Component
@Scope("prototype")
public class Student {
   private int rollNo;
   private String name;
   private int marks;
   // Getters and setters...
}
```

StudentRepo.java:

```
package com.telusko.SpringJDBCEx.repo;

@Repository
public class StudentRepo {

   public void save(Student s) {
      System.out.println("Student added to the database.");
   }

   public List<Student> findAll() {
      List<Student> students = new ArrayList<>();
      return students;
   }
}
```

StudentService.java:

```
package com.telusko.SpringJDBCEx.service;

@Service
public class StudentService {
   private StudentRepo repo;
   @Autowired
   public void setRepo(StudentRepo repo) {
      this.repo = repo;
   }

   // Method to add a student to the repository
   public void addStudent(Student s) {
      repo.save(s);
   }

   // Method to retrieve all students from the repository
   public List<Student> getStudents() {
      return repo.findAll();
    }
}
```

Application.java:

```
package com.telusko.SpringJDBCEx;

@SpringBootApplication
public class Application {
   public static void main(String[] args) {
      ApplicationContext context = SpringApplication.run(Application.class, args);
      Student s = context.getBean(Student.class);
      StudentService service = context.getBean(StudentService.class);

      // Set the properties for the student
      s.setRollNo(101);
      s.setName("Navin");
      s.setMarks(78);

      // Add the student using the service
      service.addStudent(s);

      // Retrieve all students from the service
      List<Student> students = service.getStudents();
      System.out.println(students);
   }
}
```

Code Link:

https://github.com/navinreddy20/spring6-course/tree/c6690e4f2c70d8f530d70623f13d14ff0ffd7e7d/4%20Spring%20JDBC/4.3%20Student%20Service%20And%20Repo

04 - JdbcTemplate

JdbcTemplate is the core component that simplifies the interaction with the database in a Spring-based application. It provides various methods for executing SQL operations like insert, update, delete, and retrieving data.

👉 **update() method:**

- update() method is used to execute insert, delete, and update queries in the database.
- It accepts two parameters:
  - SQL Query: The SQL statement to execute
  - Values: The values to pass into the SQL query
- The return value of update() is an integer that indicates the number of rows affected by the operation.

Example:

StudentRepo.java:

```
package com.telusko.SpringJDBCEx.repo;

@Repository
public class StudentRepo {
   private JdbcTemplate jdbc;
   public JdbcTemplate getJdbc() {
      return jdbc;
   }

   @Autowired
   public void setJdbc(JdbcTemplate jdbc) {
      this.jdbc = jdbc;
   }

   // Save method using JdbcTemplate
   public void save(Student s) {

      String sql = "INSERT INTO student (rollno, name, marks) VALUES (?, ?, ?)";
      int rows = jdbc.update(sql, s.getRollNo(), s.getName(), s.getMarks());
      System.out.println(rows + " row's affected.");
  }
```

```
    // Method to retrieve all students
    public List<Student> findAll() {
        List<Student> students = new ArrayList<>();
        return students;
    }
}
```

- The JdbcTemplate object is injected into the StudentRepo class using Spring's @Autowired annotation. This provides the repository class with methods to interact with the database.
- In the save() method, the SQL INSERT query is defined, where the rollNo, name, and marks values are passed as query parameters.
- The jdbc.update() method is called with the query and student data that inserts the data into the database and returns the number of rows affected.

Code Link:

https://github.com/navinreddy20/spring6-course/tree/c6690e4f2c70d8f530d70623f13d14ff0ffd7e7d/4%20Spring%20JDBC/4.4%20Jdbctemplate

05 - Schema and Data Files

In Spring Boot, we can define the schema and seed data for the database using SQL files that Spring Boot automatically detects and executes during application startup. These files ensure that the database schema is created and populated with initial data when the application starts.

👉 **schema.sql**:

- This file is used to define the structure (schema) of database tables.
- It is executed first to create the table structure and contains SQL statements such as CREATE TABLE to define how the database will look.

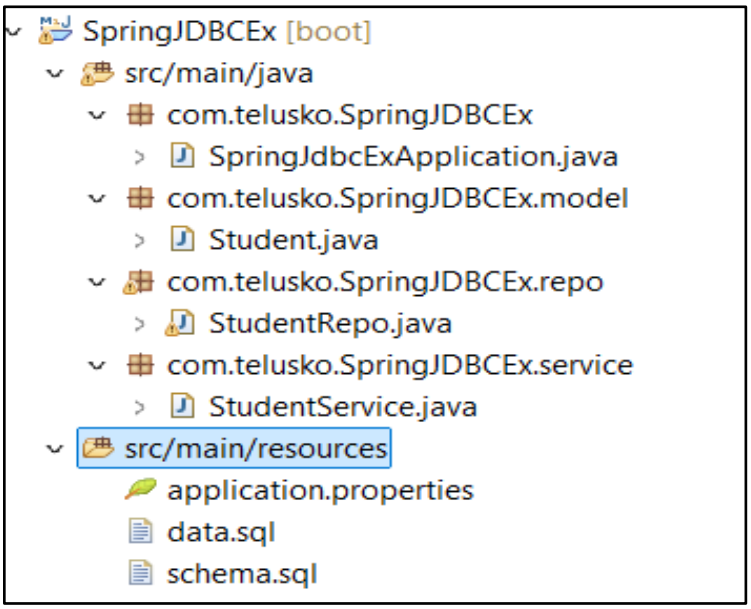
```
CREATE TABLE student (
    rollno INT PRIMARY KEY,
    name VARCHAR(50),
    marks INT
);
```

👉 **data.sql**:

- This file is used to insert initial data into the database once the schema is created as defined in the schema.sql.
- It contains INSERT SQL statements to populate the tables with default data.

```
INSERT INTO student (rollno, name, marks) VALUES (101, 'Kiran', 79);
INSERT INTO student (rollno, name, marks) VALUES (102, 'Harsh', 68);
INSERT INTO student (rollno, name, marks) VALUES (103, 'Sushil', 82);
```

★ Both files (schema.sql and data.sql) should be placed in the **src/main/resources** directory of the Spring Boot project.



We can also use custom names like my_custom_schema.sql and my_custom_data.sql for the SQL files, but then we need to modify the Spring Boot configuration in the **application.properties** file to point to those files.

application.properties:

```
spring.sql.init.schema-locations=classpath:my_custom_schema.sql

spring.sql.init.data-locations=classpath:my_custom_data.sql
```

Code Link:

https://github.com/navinreddy20/spring6-course/tree/c6690e4f2c70d8f530d70623f13d14ff0ffd7e7d/4%20Spring%20JDBC/4.5%20Schema%20And%20Data%20Files

06 - RowMapper

The RowMapper is a functional interface in Spring JDBC used to map rows of the ResultSet to objects.

- It simplifies the handling of SELECT queries, allowing data from the database to be seamlessly converted into objects.
- It provides the method mapRow() that needs to be implemented to map each row.

👉 **mapRow(ResultSet rs, int rowNum):**

- This method is executed for each row in the ResultSet.
- It contains two parameters:
    - **ResultSet rs:** The result of the SQL query containing the data.
    - **int rowNum:** The current row number in the ResultSet.
- It returns a Java object representing the current row, populated with data from the ResultSet.

👉 **jdbcTemplate.query():**

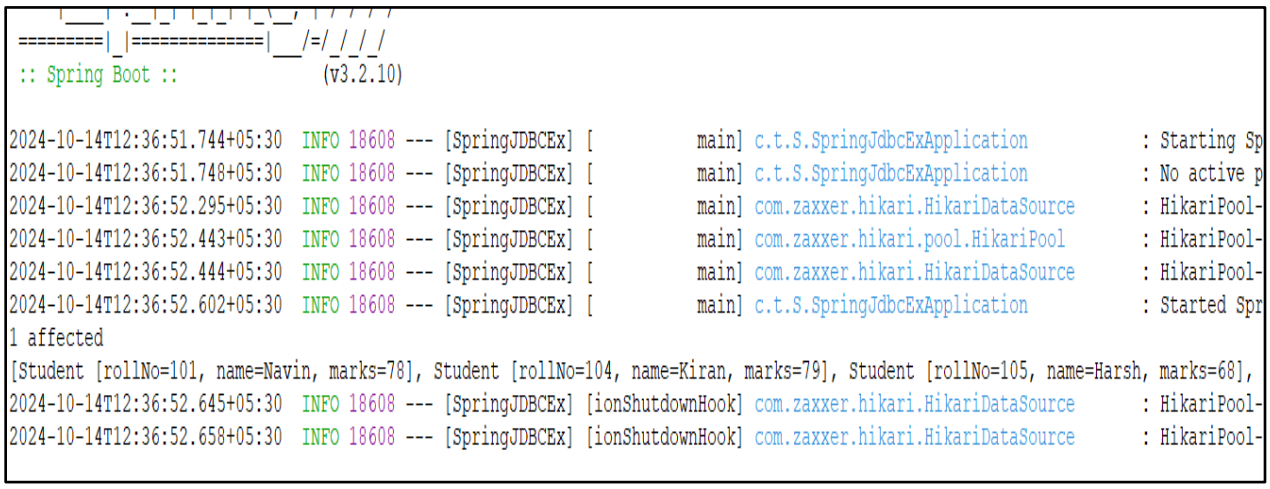
- This method is used to perform SELECT operations in Spring JDBC.
- It requires two parameters:
    1. **SQL Query:** The SQL query that need to be executed.
    2. **RowMapper:** The object responsible for mapping each row of the ResultSet to a Java object.

Example:

StudentRepo.java

```
package com.telusko.SpringJDBCEx.repo;

@Repository
public class StudentRepo {
   private JdbcTemplate jdbc;
   @Autowired
   public void setJdbc(JdbcTemplate jdbc) {
      this.jdbc = jdbc;
   }

   public void save(Student s) {
      String sql = "INSERT INTO student(rollno, name, marks) VALUES(?, ?, ?)";
      int rows = jdbc.update(sql, s.getRollNo(), s.getName(), s.getMarks());
      System.out.println(rows + " rows affected");
   }

   // Find all students (using lambda expression)
   public List<Student> findAll() {
      String sql = "SELECT * FROM student";
      RowMapper<Student> mapper = (rs, rowNum) -> {
         Student s = new Student();

         s.setRollNo(rs.getInt("rollno"));
         s.setName(rs.getString("name"));
         s.setMarks(rs.getInt("marks"));
         return s;
      };
      return jdbc.query(sql, mapper);
   }

   // Find all students (without lambda expression)
   /*
   public List<Student> findAll() {
      String sql = "SELECT * FROM student";
      RowMapper<Student> mapper = new RowMapper<Student>() {
        @Override
         public Student mapRow(ResultSet rs, int rowNum) throws SQLException {
            Student s = new Student();
            s.setRollNo(rs.getInt("rollno"));
            s.setName(rs.getString("name"));
            s.setMarks(rs.getInt("marks"));
            return s;
         }
      };
      return jdbc.query(sql, mapper);
   }   */
}
```

Output:



```
 :: Spring Boot ::                (v3.2.10)

2024-10-14T12:36:51.744+05:30  INFO 18608 --- [SpringJDBCEx] [           main] c.t.S.SpringJdbcExApplication           : Starting Sp
2024-10-14T12:36:51.748+05:30  INFO 18608 --- [SpringJDBCEx] [           main] c.t.S.SpringJdbcExApplication           : No active p
2024-10-14T12:36:52.295+05:30  INFO 18608 --- [SpringJDBCEx] [           main] com.zaxxer.hikari.HikariDataSource      : HikariPool-
2024-10-14T12:36:52.443+05:30  INFO 18608 --- [SpringJDBCEx] [           main] com.zaxxer.hikari.pool.HikariPool       : HikariPool-
2024-10-14T12:36:52.444+05:30  INFO 18608 --- [SpringJDBCEx] [           main] com.zaxxer.hikari.HikariDataSource      : HikariPool-
2024-10-14T12:36:52.602+05:30  INFO 18608 --- [SpringJDBCEx] [           main] c.t.S.SpringJdbcExApplication           : Started Spr
1 affected
[Student [rollNo=101, name=Navin, marks=78], Student [rollNo=104, name=Kiran, marks=79], Student [rollNo=105, name=Harsh, marks=68],
2024-10-14T12:36:52.645+05:30  INFO 18608 --- [SpringJDBCEx] [ionShutdownHook] com.zaxxer.hikari.HikariDataSource      : HikariPool-
2024-10-14T12:36:52.658+05:30  INFO 18608 --- [SpringJDBCEx] [ionShutdownHook] com.zaxxer.hikari.HikariDataSource      : HikariPool-
```

Code Link:

https://github.com/navinreddy20/spring6-course/tree/c6690e4f2c70d8f530d70623f13d14ff0ffd7e7d/4%20Spring%20JDBC/4.6%20Rowmapper

07 - Spring JDBC Postgres

PostgreSQL is an open-source, powerful, and highly extensible relational database management system (RDBMS).

- It supports SQL (Structured Query Language) for querying and interacting with the database and is known for its reliability, robustness, and performance.
- In addition to using H2 for in-memory database operations, we can use PostgreSQL to store and access persistent data in the Spring Boot application.
- **Link to download Postresql:** https://www.postgresql.org/download/

👉 **Steps to Use PostgreSQL:**

1. **Add PostgreSQL Dependency:**

   To use PostgreSQL we need to replace the H2 dependency with PostgreSQL in the pom.xml file.

```
<dependency>
    <groupId>org.postgresql</groupId>
    <artifactId>postgresql</artifactId>
    <scope>runtime</scope>
</dependency>
```

2. **Configure PostgreSQL in application.properties:**

- We have to configure the connection settings in the application.properties file that is located in the src/main/resources directory.
- The default port for PostgreSQL is 5432.

Application.properties:

```
# PostgreSQL database connection URL

spring.datasource.url=jdbc:postgresql://localhost:5432/telusko

# Database username

spring.datasource.username=postgres

# Database password

spring.datasource.password=root

# PostgreSQL JDBC Driver

spring.datasource.driver-class-name=org.postgresql.Driver
```

- ***spring.datasource.url:*** Specifies the JDBC URL for connecting to the PostgreSQL database. Replace localhost with the database server's IP if hosted remotely. The telusko is the name of your database.
- ***spring.datasource.username:*** The username for accessing the PostgreSQL database. The default username is Postgres.
- ***spring.datasource.password:*** The password associated with the username.
- ***spring.datasource.driver-class-name:*** Specifies the PostgreSQL JDBC driver class.

Code Link:

https://github.com/navinreddy20/spring6-course/tree/c6690e4f2c70d8f530d70623f13d14ff0ffd7e7d/4%20Spring%20JDBC/4.7%20Spring%20Jdbc%20Postgres