**Spring JDBC:**

Which helps to connect our Java Application with our database and to perform applications.

**Problems using JDBC (Normal):**

1. we need to write lot of code before executing the query and after executing the query.
2. Every time we need to write statements for open and close the statements
3. Exception Handling Problem: Checked exception for every operation we have to write the try and catch block (Sql Exception)
4. Repeating of all these code from one to another database logic is a time consuming task.

**Solutions:**

Spring JDBC -🡪 provides class JDBC Template which has all important methods to perform operation with database.

Steps to involve

1. Create Maven project

2. Add dependencies Spring JDBC and mysql connector along with spring core and spring context.

Note: to avoid error make sure (try) to use same versions for all dependencies

Once you have added all dependencies you can go with the next step

(check buildpath for jre14 or maven depencies) added or not

3. Database Setup, Spring JDBC Configuration, JDBC Template and DataSource Setup

First you have to create database in mysql (local host)

Create table – student

4.After that create student.java file add attribute student id, name, address

Generate,getter and setter ,default constructor, super constructor and toString()

1. After that create xml file add injecting beans for jdbc template for that app.java click ctrl+shift+t add package search it for jdbctemplate.

Note: If you want to configure JDBCTemplate configuration in config.xml file , first we have to inject jdbctemplate object . To get that object we have a implementation class called DriverManagerDataScource.

Can we for app.jav and press ctr+shft+t 🡪 search DriverMangerDataSource

Then that class has opened and to create a bean for that class, for that we have to set 4 properties those are

Mysql driver class name

Url

Username

Password

by setting up these properties we have created the bean.

Next we need inject this name (object) into the JDBCTemplate bean

The property of JDBCTemplate is DataSource ref bean=””

Once we have done with this we have completed JDBCTemplate configuration in xml file

**Inserting data into database Using Spring JDBC**

**Update data into database Using Spring JDBC**

**Delete data from database using springJDBC**

**Fetching data using springJDBC – (Row Mapper)**

To fetch a single object using a RowMapper in Spring JDBC, you can use the queryForObject method provided by the JdbcTemplate class. The queryForObject method is used when you expect a single row as the result of the query.

Here's an example of how to fetch a single object using a RowMapper:

JdbcTemplate jdbcTemplate = new JdbcTemplate(dataSource);

String sql = "SELECT id, username, email FROM users WHERE id = ?";

Object[] params = {userId};

User user = jdbcTemplate.queryForObject(sql, params, new UserRowMapper());

In the example above, the SQL query includes a parameter placeholder (?) for the user ID. The params array contains the actual parameter value(s) that will be used in the query. This is done to avoid SQL injection and provide a secure way to pass parameters to the query.

The queryForObject method takes three arguments:

The SQL query string.

An array of parameters, if any, to be bound to the query.

An instance of the RowMapper that will map the row to a domain object.

The queryForObject method will execute the query and attempt to map the result to a single object using the provided RowMapper. If the query returns zero rows, it will throw an exception (such as EmptyResultDataAccessException). If the query returns more than one row, it will also throw an exception (such as IncorrectResultSizeDataAccessException).

Make sure that the query you provide in sql returns only one row when using queryForObject. Otherwise, you should consider using query method with a RowMapper and handle the list of objects appropriately.

To fetch a single object using a RowMapper in Spring JDBC, you can use the queryForObject method provided by the JdbcTemplate class. The queryForObject method is used when you expect a single row as the result of the query.

Here's an example of how to fetch a single object using a RowMapper:

JdbcTemplate jdbcTemplate = new JdbcTemplate(dataSource);

String sql = "SELECT id, username, email FROM users WHERE id = ?";

Object[] params = {userId};

User user = jdbcTemplate.queryForObject(sql, params, new UserRowMapper());

In the example above, the SQL query includes a parameter placeholder (?) for the user ID. The params array contains the actual parameter value(s) that will be used in the query. This is done to avoid SQL injection and provide a secure way to pass parameters to the query.

The queryForObject method takes three arguments:

The SQL query string.

An array of parameters, if any, to be bound to the query.

An instance of the RowMapper that will map the row to a domain object.

The queryForObject method will execute the query and attempt to map the result to a single object using the provided RowMapper. If the query returns zero rows, it will throw an exception (such as EmptyResultDataAccessException). If the query returns more than one row, it will also throw an exception (such as IncorrectResultSizeDataAccessException).

Make sure that the query you provide in sql returns only one row when using queryForObject. Otherwise, you should consider using query method with a RowMapper and handle the list of objects appropriately.