**Spring JDBC**

1. **Jdbc**, registering drivers and loading drivers, making connections, preparing statements, executing statements has to be done explicitly, due to this reason performance of the applications will fall when we are creating enterprise applications.  
   **Spring JDBC**, it is only responsibility for executing queries, making connections and loading drivers will be done implicitly.
2. Limited Support for Transactions in JDBC, in Spring JDBC, it supports full transactions with the help of its relevant classes and interface.
3. In JDBC all the exceptions have to be handled Explicitly because every exception is a checked exception, whereas Spring JDBC will handle all exceptions implicitly.

To prepare DAO Layer, Spring JDBC has provided the complete predefined library in the form of class and interfaces, these all are available in following package “org.springframework.jdbc”

JdbcTemplate

NamedParameterJdbcTemplate

SimpleJdbcTemplate -- it will work on spring jdbc 1.0 and 2.0, on later versions it has been removed

SimpleJdbcInsert :- it is only for Insert

SimpleJdbcCall :- it is only for calling procedures

SQL Mapping through SQLUpdate and SQL Insert

Steps :-

1. Create DAO interface with the required methods
2. Create DAO implementation class with implementation for DAO interface methods
3. In Config file, provide configuration for DataSource class, JdbcTemplate class and DAO Implementation class.
4. Prepare Test Application

Connection con = DriverManager.getConnection(“string”, “username”, “password”)

Spring Configuration File, we have to configure DataSource with following Properties

driverClassName :- “com.mysql.jdbc.Driver”

url :- “jdbc://http://localhost:3306/dbname”

username :- “username”

password:- “password”

To configure DataSource Spring provided a separate predefined DataSource class in the form of “org.springframework.jdbc.datasource.DriverManagerDataSource”.

JdbcTemplate class providing basic environment to interact with Database like Loading Driver class, getting connection between java app and DB. Creating Statement, PreparedStatement and CallableStatement and closing connection with the help of the provided DataSource and JdbcTemplate class has provided the following methods to execute SQL Queries

<beans xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:context=*"http://www.springframework.org/schema/context"*

xmlns:mvc=*"http://www.springframework.org/schema/mvc"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans*

*http://www.springframework.org/schema/beans/spring-beans-3.0.xsd*

*http://www.springframework.org/schema/context*

*http://www.springframework.org/schema/context/spring-context-3.0.xsd"*>

<bean id=*"dataSource"* class=*"org.springframework.jdbc.datasource.DriverManagerDataSource"*>

<property name=*"driverClassName"* value=*"com.mysql.jdbc.Driver"* />

<property name=*"url"* value=*"jdbc:mysql://localhost:3306/MorningDb?useSSL=false"* />

<property name=*"username"* value=*"root"* />

<property name=*"password"* value=*"root"* />

</bean>

<bean id=*"EmpServObj"* class=*"com.jdbc.service.EmpServiceImpl"*>

<property name=*"dataSource"* ref=*"dataSource"* />

</bean>

**JdbcTemplate <obj> = new JdbcTemplate(dataSourceObj);**

1. For Non Select SQL Queries and DML queries(Insert/Update/Delete)
   * **public int update(sql query string)**
     1. Insert into student values(?,?,?);
2. for DDL Sql Queries
   * public void execute(sql query string) 🡺 **Create, Alter, Drop SQL Commands**
3. for SELECT sql queries
   * public int queryForInt(query)
     1. select count(\*) from emp
   * public long queryForLong(query)
     1. select max(sal) from emp
   * public String queryForString(query)
     1. select ename from emp where empno = 1001
   * public Object queryForObject(query) = single row
     1. select \* from emp where empno = 101
   * public List query(query) = multiple rows
   * public List queryForList(query)
   * public Map queryForMap(query)
   * public RowSet queryForRowSet(query)

While performing retrieval operations to convert data from ResultSet object[records] to Bean objects Spring Framework has provided a predefined interface in the form of “org.springframework.jdbc.core.RowMapper” which contains the following method.

public Object mapRow(ResultSet rs, int rowCount)

public class StudentMapper implements RowMapper<Student>{

public Student mapRow(ResultSet rs, int rowCount) throws SQLException{

Stduent std = new Student();

std.setSid(rs.getString(“sid”));

std.setSname(rs.getString(“sname”));

std.setSaddr(rs.getString(“saddr”));

return std;

}

}

In Spring JDBC applications, we will use positional parameters(?) also in sql queries which we are providing along with JdbcTemplate class provided query execution methods.

If we provide positional parameters in sql queries the JdbcTemplate class will use “PreparedStatement” internally to execute sql query instead of Statement.

To provide values to the Positional parameters in SQL Queries we have to use Object[] with values as parameter to all JdbcTemplate class provided query execution methods.

public int update(String query, Object[] param\_values)

public int querForInt(String query, Object[] param\_values)

public long querForLong(String query, Object[] param\_values)

public Object querForObject(String query, Object[] param\_values, RowMapper rm)

**JdbcTemplate <obj> = new JdbcTemplate(datasourceObj)**

**String Query = “insert into student values(?,?,?)”;**

**Int rowCount = jdbctemplate.update(query, Object[]{“S-111”, “Sahasra”, “Hyd”});**

Pom.xml

<properties>

<spring-framework.version>4.0.2.RELEASE</spring-framework.version>

<!-- Logging -->

<logback.version>1.0.13</logback.version>

<slf4j.version>1.7.5</slf4j.version>

</properties>

<dependencies>

<!-- Spring and Transactions -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>${spring-framework.version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-tx</artifactId>

<version>${spring-framework.version}</version>

</dependency>

<!-- Spring JDBC Support -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-jdbc</artifactId>

<version>${spring-framework.version}</version>

</dependency>

<!-- MySQL Driver -->

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>8.0.15</version>

</dependency>

</dependencies>

NamedParameterJdbcTemplate :- it is as same as JdbcTemplate, it will accept arguments or parameters with name.

String inscmd = “Insert into student values(:p1, :p2, :p3)”

Here :p1, :p2, :p3 are the named parameters, to pass values these parameter, we have two ways

1. By using Map Directly
2. By using SqlParameterSource Interface

**By using Map:-**

String inscmd = “Insert into student values(:p1, :p2, :p3)”

Map map = new HashMap();

//map.put(parametername, value)

map.put(“p1”, 1001)

map.put(“p2”, “Naresh”)

map.put(“p3”, “Hyd”)

NamedParameterJdbcTemplate npjt = new NamedParameterJdbcTemplate(dataSource);

npjt.update(inscmd, map);

**By using SqlParameterSource Interface**

With the help this interface, we can pass parameters, but it has two implementations,

1. MapSqlParameterSource
2. BeanPropertySqlParameterSource

String inscmd = “Insert into student values(:p1, :p2, :p3)”

syntax

SqlParameterSource <Obj> = new MapSqlParameterSoruce(“parameter”, value);

Example

MapSqlParameterSoruce sps = new MapSqlParameterSoruce(“p1”, 1001);

sps = sps.addValue(“p2”, “Murali”);

sps = sps.addValue(“p3”, “Hyd”);

NamedParameterJdbcTemplate npjt = new NamedParameterJdbcTemplate(dataSource);

npjt.update(inscmd, sps);

refer following link for different select queries:-

[Spring JdbcTemplate Querying Examples - Mkyong.com](https://mkyong.com/spring/spring-jdbctemplate-querying-examples/)