**How to create Spring-Boot Application in Eclipse IDE**

## 1. Create Spring Boot Project with following Dependencies:

* Spring Web (For Embedded Tomcat)
* Spring JDBC ( For connecting Spring Boot Application to Database Server)
* Actuator (For Monitoring Health of the Spring Boot Application)
* MySql (For determining that the Database Server is of MySql)
* Thymeleaf (For view resolving)

Dependency Section of your pom.xml must look like:

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-actuator</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jdbc</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-thymeleaf</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>mysql</groupId>

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

<version>8.0.33</version>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

</dependencies>

## 2. Configure DataSource

Spring Boot auto-configures DataSource, but we can customize.

In application.properties:

#Database configuration mysql

spring.datasource.url=jdbc:mysql://localhost:3306/world?allowPublicKeyRetrieval=true&useSSL=false

useUnicode=true&characterEncoding=UTF-8&serverTimezone=UTC

spring.datasource.driverClassName=com.mysql.cj.jdbc.Driver

spring.datasource.username=abc

spring.datasource.password=xyz

## 3. Create Database Schema & Data

File: src/main/resources/schema.sql

CREATE TABLE student (

id INT AUTO\_INCREMENT PRIMARY KEY,

name VARCHAR(100),

age INT

);

File: src/main/resources/data.sql

INSERT INTO student (name, age) VALUES ('Alice', 21);

INSERT INTO student (name, age) VALUES ('Bob', 22);

## 4. JDBC Template Bean

Spring Boot automatically provides JdbcTemplate. You can **autowire** it:

StudentRepository.java

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

@Repository

public class StudentRepository {

private final JdbcTemplate jdbcTemplate;

public StudentRepository(JdbcTemplate jdbcTemplate) {

this.jdbcTemplate = jdbcTemplate;

}

// 1. Execute SQL Statement

public void createTable() {

jdbcTemplate.execute("CREATE TABLE IF NOT EXISTS test (id INT PRIMARY KEY, name VARCHAR(50))");

public void createTable() {

jdbcTemplate.execute("CREATE TABLE IF NOT EXISTS student (id INT PRIMARY KEY, name VARCHAR(50))");

}

// 2. Insert with SQL Parameters (positional)

public int addStudent(String name, int age) {

String sql = "INSERT INTO student (name, age) VALUES (?, ?)";

return jdbcTemplate.update(sql, name, age);

}

// 3. Query for object

public String findStudentNameById(int id) {

String sql = "SELECT name FROM student WHERE id = ?";

return jdbcTemplate.queryForObject(sql, new Object[]{id}, String.class);

}

}

## 5. Using Named Parameters

For **readability**, use NamedParameterJdbcTemplate.

import org.springframework.jdbc.core.namedparam.MapSqlParameterSource;

import org.springframework.jdbc.core.namedparam.NamedParameterJdbcTemplate;

import org.springframework.stereotype.Repository;

@Repository

public class StudentNamedRepository {

private final NamedParameterJdbcTemplate namedParameterJdbcTemplate;

public StudentNamedRepository(NamedParameterJdbcTemplate namedParameterJdbcTemplate) {

this.namedParameterJdbcTemplate = namedParameterJdbcTemplate;

}

public int addStudent(String name, int age) {

String sql = "INSERT INTO student (name, age) VALUES (:name, :age)";

MapSqlParameterSource params = new MapSqlParameterSource()

.addValue("name", name)

.addValue("age", age);

return namedParameterJdbcTemplate.update(sql, params);

}

}

## 6. Service & Controller Layer

StudentService.java

import org.springframework.stereotype.Service;

@Service

public class StudentService {

private final StudentRepository studentRepo;

public StudentService(StudentRepository studentRepo) {

this.studentRepo = studentRepo;

}

public void demo() {

studentRepo.addStudent("Charlie", 23);

System.out.println("Student Name with ID=1: " + studentRepo.findStudentNameById(1));

}

}

StudentController.java

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.RestController;

@Controller

public class StudentController {

private final StudentService service;

public StudentController(StudentService service) {

this.service = service;

}

@GetMapping("/test")

public String test() {

service.demo();

return "Check Console for Student Data!";

}

}

## 7. Run & Test

Start Spring Boot app

Visit → [http://localhost:8080/test](http://localhost:8080/test" \t "_new)