**Module 6 - Spring Data JPA with Spring Boot, Hibernate**

**Hands on 1: Spring Data JPA - Quick Example**

**Objective:**

**To implement a basic Spring Boot application using Spring Data JPA and Hibernate to retrieve all country records from a MySQL database and display them via logs.**

**Steps Followed:**

**1. Project Setup**

* Created a Spring Boot Maven project named orm-learn.
* Added dependencies in pom.xml:
  + Spring Boot Starter Data JPA
  + MySQL Driver
  + Spring Boot DevTools

**POM.XML CODE:**

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<parent>

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

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

<version>3.5.3</version>

<relativePath/> <!-- lookup parent from repository -->

</parent>

<groupId>com.cognizant</groupId>

<artifactId>orm-learn</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>orm-learn</name>

<description>Demo project for Spring Data JPA and Hibernate</description>

<url/>

<licenses>

<license/>

</licenses>

<developers>

<developer/>

</developers>

<scm>

<connection/>

<developerConnection/>

<tag/>

<url/>

</scm>

<properties>

<java.version>17</java.version>

</properties>

<dependencies>

<dependency>

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

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

</dependency>

<dependency>

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

<artifactId>spring-boot-devtools</artifactId>

<scope>runtime</scope>

<optional>true</optional>

</dependency>

<dependency>

<groupId>com.mysql</groupId>

<artifactId>mysql-connector-j</artifactId>

<scope>runtime</scope>

</dependency>

<dependency>

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

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

<scope>test</scope>

</dependency>

<dependency>

<groupId>jakarta.persistence</groupId>

<artifactId>jakarta.persistence-api</artifactId>

<version>3.1.0</version>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

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

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

</project>

**2. MySQL Table Setup**

* Opened MySQL Workbench.
* Created a database and table:

CREATE TABLE country (

co\_code VARCHAR(2) PRIMARY KEY,

co\_name VARCHAR(50) NOT NULL

);

INSERT INTO country (co\_code, co\_name) VALUES ('IN', 'India'), ('US', 'United States of America');

A screenshot of a computer

AI-generated content may be incorrect.

**3. application.properties Configuration**

Configured DB connection details:

# Spring Framework and application logs

logging.level.org.springframework=info

logging.level.com.cognizant=debug

# Hibernate logs (to see SQLs)

logging.level.org.hibernate.SQL=trace

logging.level.org.hibernate.type.descriptor.sql=trace

# Console log format

logging.pattern.console=%d{dd-MM-yy} %d{HH:mm:ss.SSS} %-20.20thread %5p %-25.25logger{25} %25M %4L %m%n

# Database configuration

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.datasource.url=jdbc:mysql://localhost:3306/ormlearn

spring.datasource.username=root

spring.datasource.password=bhavyainjam

# Hibernate

spring.jpa.hibernate.ddl-auto=validate

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect

**4. Created Country Entity Class**

package com.cognizant.ormlearn.model;

import jakarta.persistence.Column;

import jakarta.persistence.Entity;

import jakarta.persistence.Id;

import jakarta.persistence.Table;

@Entity

@Table(name = "country")

public class Country {

@Id

@Column(name = "co\_code")

private String code;

@Column(name = "co\_name")

private String name;

public String getCode() {

return code;

}

public void setCode(String code) {

this.code = code;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

@Override

public String toString() {

return "Country [code=" + code + ", name=" + name + "]";

}

}

**5. Created CountryRepository Interface**

package com.cognizant.ormlearn.repository;

import com.cognizant.ormlearn.model.Country;

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.stereotype.Repository;

@Repository

public interface CountryRepository extends JpaRepository<Country, String> { }

**6. Created CountryService Interface**

package com.cognizant.ormlearn.service;

import com.cognizant.ormlearn.model.Country;

import java.util.List;

public interface CountryService {

List<Country> getAllCountries();

}

**7. Created CountryServiceImpl Class**

package com.cognizant.ormlearn.service;

import com.cognizant.ormlearn.model.Country;

import com.cognizant.ormlearn.repository.CountryRepository;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import java.util.List;

@Service

public class CountryServiceImpl implements CountryService {

@Autowired

private CountryRepository countryRepository;

@Override

public List<Country> getAllCountries() {

return countryRepository.findAll();

}

}

**8. Modified OrmLearnApplication.java to Test**

@SpringBootApplication

public class OrmLearnApplication implements CommandLineRunner {

private static final Logger LOGGER = LoggerFactory.getLogger(OrmLearnApplication.class);

@Autowired

private CountryService countryService;

public static void main(String[] args) {

SpringApplication.run(OrmLearnApplication.class, args);

}

@Override

public void run(String... args) throws Exception {

LOGGER.info("Inside run method");

testGetAllCountries();

}

public void testGetAllCountries() {

LOGGER.info("Start");

List<Country> countries = countryService.getAllCountries();

LOGGER.debug("countries={}", countries);

LOGGER.info("End");

}

}

**OUTPUT SCREENSHOTS:**

**Inside run method**

**Start**

**countries=[Country [code=IN, name=India], Country [code=US, name=United States of America]]**

**End**

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

**Result:**

Successfully retrieved all countries from the MySQL database using Spring Data JPA and displayed them in the console using logs.

**Summary:**

* Spring Boot Project Setup
* Country Table Created in MySQL
* JPA Entity and Repository Created
* Service Layer Added
* Connected MySQL DB and Retrieved Data
* Console Output Verified

**Hands-On 4:**

**Difference Between JPA, Hibernate, and Spring Data JPA**

**Java Persistence API (JPA)**

* JPA is a specification (JSR 338) for persisting, reading, and managing data between Java objects and a relational database.
* It does not contain implementation; instead, vendors (like Hibernate) provide concrete implementations.
* It helps define standard annotations and APIs for ORM without locking developers into one particular tool.
* Example providers: Hibernate, EclipseLink, OpenJPA.

**Hibernate**

* Hibernate is an Object Relational Mapping (ORM) tool and the most popular implementation of JPA.
* It allows us to map Java classes to database tables using XML or annotations.
* Provides caching, lazy loading, automatic dirty checking, and transaction management.
* Requires boilerplate code for opening/closing sessions and managing transactions.

**Spring Data JPA**

* Spring Data JPA is not an implementation of JPA; instead, it is a framework built on top of JPA and an implementation provider (like Hibernate).
* It provides a higher-level abstraction, reducing boilerplate code drastically.
* It manages the entity lifecycle and transaction management using Spring’s features.
* Includes powerful repository support by extending interfaces like JpaRepository.

**Practical Demonstration**

**Using Hibernate (Manual session handling)**

public Integer addEmployee(Employee employee){

Session session = factory.openSession();

Transaction tx = null;

Integer employeeID = null;

try {

tx = session.beginTransaction();

employeeID = (Integer) session.save(employee);

tx.commit();

} catch (HibernateException e) {

if (tx != null) tx.rollback();

e.printStackTrace();

} finally {

session.close();

}

return employeeID; }

* Requires explicit session management and transaction handling.
* More boilerplate code and risk of forgetting cleanup.

**Using Spring Data JPA (Simple and Declarative)**

public interface EmployeeRepository extends JpaRepository<Employee, Integer> {

// No method body required, Spring provides implementation

}

@Service

public class EmployeeService {

@Autowired

private EmployeeRepository employeeRepository;

@Transactional

public void addEmployee(Employee employee) {

employeeRepository.save(employee);

}

}

* Just extend JpaRepository, and Spring auto-generates methods like save(), delete(), findById(), etc.
* Uses Spring's transaction management (@Transactional).
* Cleaner, testable, and maintainable code.

**Conclusion**

* JPA gives standardization for ORM.
* Hibernate is the most popular and feature-rich JPA implementation.
* Spring Data JPA abstracts all the complexities and lets developers focus on business logic instead of data access boilerplate.
* Spring Data JPA is highly recommended for rapid and cleaner development in Spring Boot projects.

**Real-Time Example Comparison**

Imagine you're building an HR system and need to save employee records.

* With Hibernate, you manually write the logic to open sessions, handle transactions, and save the employee.
* With Spring Data JPA, you simply call employeeRepository.save(employee)—everything else (session, transaction, flushing) is taken care of automatically.

**This means:**  
Less code  
Fewer chances of errors  
Easier testing  
Faster development

**Final Notes**

* In industry-standard Spring Boot projects, Spring Data JPA is preferred for its elegance and minimal code.
* Hibernate is still very powerful and used when custom behavior, performance tuning, or deep control over SQL is required.
* Both Hibernate and Spring Data JPA can coexist—Spring Data JPA internally uses Hibernate by default!