**WEEK -3**

**SPRING DATA JPA WITH SPRING BOOT, HIBERNATE**

1. **Spring Data JPA - Quick Example**

**Package name: com.cognizant.ormlearn.model**

**File name: Country.java**

package com.cognizant.ormlearn.model;

import jakarta.persistence.\*;

@Entity

@Table(name = "country")

public class Country {

@Id

@Column(name = "co\_code")

private String code;

@Column(name = "co\_name")

private String name;

// Getters and setters

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;

}

// toString for logging/debugging

@Override

public String toString() {

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

}

}

**Package name: com.cognizant.ormlearn.service**

**File name: CountryService.java**

package com.cognizant.ormlearn.service;

import java.util.List;

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

import org.springframework.stereotype.Service;

import org.springframework.transaction.annotation.Transactional;

import com.cognizant.ormlearn.model.Country;

import com.cognizant.ormlearn.repository.CountryRepository;

@Service

public class CountryService {

@Autowired

private CountryRepository countryRepository;

@Transactional

public List<Country> getAllCountries() {

return countryRepository.findAll();

}

}

**Package name: com.cognizant.ormlearn.repository**

**File name: CountryRepository.java**

package com.cognizant.ormlearn.repository;

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

import org.springframework.stereotype.Repository;

import com.cognizant.ormlearn.model.Country;

@Repository

public interface CountryRepository extends JpaRepository<Country, String> {

}

**Package name:com.cognizant.resources**

**File name:application.properties**

spring.application.name=orm-learn

# Logging

logging.level.org.springframework=info

logging.level.com.cognizant=debug

logging.level.org.hibernate.SQL=trace

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

logging.level.com.cognizant.orm\_learn=DEBUG

# Database

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=2004

# Hibernate

spring.jpa.database-platform=org.hibernate.dialect.MySQLDialect

**File name: OrmLearnApplicationTests.java**

package com.cognizant.orm\_learn;

import org.junit.jupiter.api.Test;

import org.springframework.boot.test.context.SpringBootTest;

@SpringBootTest

class OrmLearnApplicationTests {

@Test

void contextLoads() {

}

}

**File name:pom.xml**

<?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>

</dependencies>

<build>

<plugins>

<plugin>

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

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

</plugin>

</plugins>

</build>

</project>

**Package name: com.cognizant.orm\_learn**

**File name: OrmLearnApplication.java**

package com.cognizant.orm\_learn;

import java.util.List;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.boot.autoconfigure.domain.EntityScan;

import org.springframework.context.ApplicationContext;

import org.springframework.context.annotation.ComponentScan;

import org.springframework.data.jpa.repository.config.EnableJpaRepositories;

import com.cognizant.ormlearn.model.Country;

import com.cognizant.ormlearn.service.CountryService;

@SpringBootApplication

@ComponentScan(basePackages = "com.cognizant.ormlearn")

@EnableJpaRepositories(basePackages = "com.cognizant.ormlearn.repository")

@EntityScan(basePackages = "com.cognizant.ormlearn.model")

public class OrmLearnApplication {

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

private static CountryService countryService;

public static void main(String[] args) {

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

countryService = context.getBean(CountryService.class);

testGetAllCountries();

}

private static void testGetAllCountries() {

LOGGER.info("Start");

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

for (Country c : countries) {

System.out.println("Country Code: " + c.getCode() + ", Name: " + c.getName());

}

LOGGER.info("End");

}

}

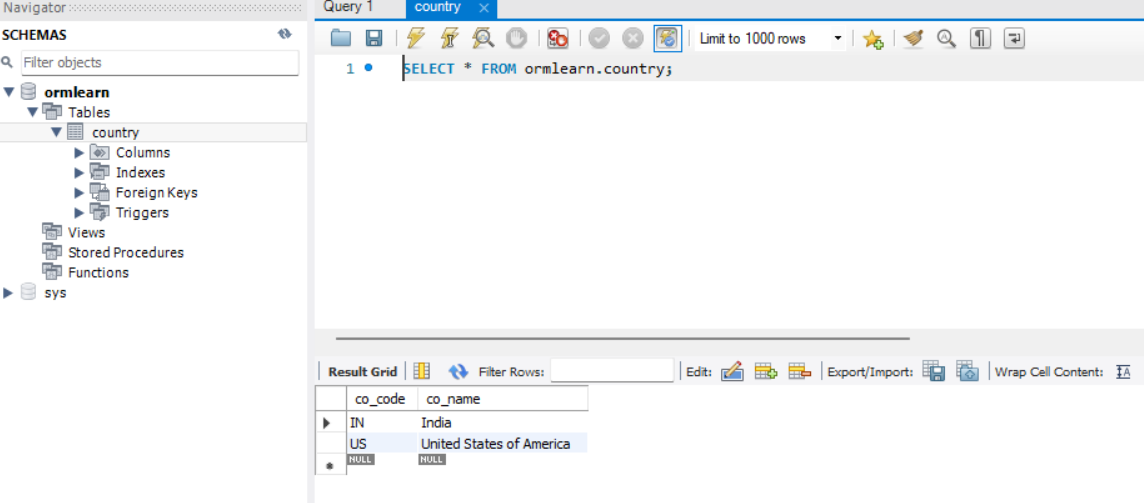
**Output:**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

****

1. **Understanding JPA, Hibernate, and Spring Data JPA**

**What is JPA?**

JPA stands for Java Persistence API, and it's essentially a blueprint or specification created by the Java community. Think of it as a contract that defines how Java applications should interact with databases. JPA itself doesn't actually do any work - it just tells developers what methods and annotations they should use when working with databases.

JPA provides you with standardized ways to map your Java objects to database tables using annotations like @Entity for marking classes as database entities, @Id for primary keys, and @OneToMany for relationships between tables. The key thing to remember is that JPA is just the rulebook, not the actual implementation.

**What is Hibernate?**

Hibernate is like a skilled craftsman who knows how to follow the JPA blueprint perfectly. It's a concrete implementation of the JPA specification, meaning it actually does the heavy lifting of converting your Java objects into database operations and vice versa.

But Hibernate goes beyond just implementing JPA. It adds its own extra features that make database operations more efficient and powerful. For example, it includes advanced caching mechanisms to speed up your applications, sophisticated lazy loading strategies to avoid unnecessary database calls, and its own query language called HQL (Hibernate Query Language) that's more object-oriented than regular SQL.

When you use Hibernate, you can choose to stick with standard JPA methods or tap into Hibernate's special features when you need more control over your database operations.

**What is Spring Data JPA?**

Spring Data JPA is like having a personal assistant who handles all the tedious paperwork for you. It's a layer built on top of JPA that dramatically reduces the amount of code you need to write when working with databases.

Instead of manually creating Data Access Objects (DAOs) and writing repetitive database queries, Spring Data JPA lets you simply define interfaces with method names that describe what you want to do. For instance, if you want to find a user by their username, you just declare a method called findByUsername and Spring Data JPA automatically generates the necessary database query for you.

The beauty of Spring Data JPA is that it works with any JPA implementation, not just Hibernate. Whether you're using Hibernate, EclipseLink, or another JPA provider underneath, Spring Data JPA will work seamlessly with it.

**How They Work Together**

Imagine you're building a house. JPA is the architectural blueprint that shows what the house should look like. Hibernate is the construction company that actually builds the house according to those blueprints, but they also add some custom features like a better foundation or upgraded electrical systems. Spring Data JPA is like a project manager who streamlines the entire construction process, handling all the paperwork and coordination so you don't have to worry about the details.

**Real-World Example: Finding Countries by Code**

Let's say you're building an application that needs to look up countries by their country codes like "US" for United States or "IN" for India.

First, you'd create a Country entity class that represents your database table:

import jakarta.persistence.Entity;

import jakarta.persistence.Id;

@Entity

public class Country {

@Id

private String code;

private String name;

// Standard getters and setters would go here

}

Next, instead of writing complex database access code, you'd simply create a repository interface:

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

import java.util.Optional;

public interface CountryRepository extends JpaRepository<Country, String> {

Optional<Country> findByCode(String code);

}

Finally, you'd use this repository in your service layer:

@Service

public class CountryService {

@Autowired

private CountryRepository countryRepository;

public Country getCountryByCode(String code) {

return countryRepository.findByCode(code)

.orElseThrow(() -> new RuntimeException("Country not found"));

}

}

Behind the scenes, Spring Data JPA automatically generates the SQL query SELECT \* FROM country WHERE code = ? when you call the findByCode method. You don't have to write any SQL or worry about database connections - it's all handled for you.

**The Bottom Line**

JPA provides the standards, Hibernate implements those standards with additional features, and Spring Data JPA makes it incredibly easy to use JPA in your applications. Together, they form a powerful trio that lets you focus on your business logic rather than getting bogged down in database plumbing code.