**SPRING DATA JPA - QUICK EXAMPLE**

**PROGRAM:**

**application.properties**

# 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

# Console log pattern

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

# Database config

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=root

# Hibernate config

spring.jpa.hibernate.ddl-auto=validate

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5Dialect

**com.cognizant.ormlearn.model.Country**

package com.cognizant.ormlearn.model;

import javax.persistence.\*;

@Entity

@Table(name = "country")

public class Country {

@Id

@Column(name = "code")

private String code;

@Column(name = "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 + "]";

}

}

**com.cognizant.ormlearn.repository.CountryRepository**

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> {

}

**com.cognizant.ormlearn.service.CountryService**

package com.cognizant.ormlearn.service;

import java.util.List;

import javax.transaction.Transactional;

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

import org.springframework.stereotype.Service;

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();

}

}

**com.cognizant.ormlearn.OrmLearnApplication**

package com.cognizant.ormlearn;

import java.util.List;

import com.cognizant.ormlearn.model.Country;

import com.cognizant.ormlearn.service.CountryService;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.context.ApplicationContext;

@SpringBootApplication

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);

LOGGER.info("Inside main");

countryService = context.getBean(CountryService.class);

testGetAllCountries();

}

private static void testGetAllCountries() {

LOGGER.info("Start");

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

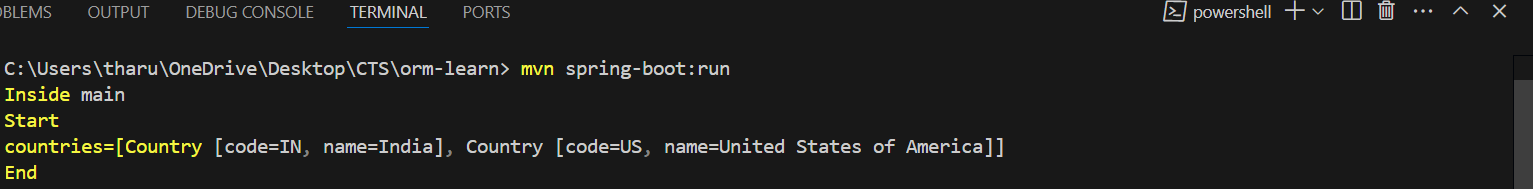
LOGGER.debug("countries={}", countries);

LOGGER.info("End");

}

}

**OUTPUT:**

****

**DIFFERENCE BETWEEN JPA, HIBERNATE AND SPRING DATA JPA**

**Java Persistence API (JPA)**

* JPA is a Java Specification (JSR 338) that defines how Java objects are persisted in relational databases.
* It is not a framework; it does not provide any actual implementation.
* It provides annotations (@Entity, @Id, @Column, etc.) and an interface (EntityManager) for ORM.
* It is a standard that can be implemented by frameworks like Hibernate, EclipseLink, etc.
* JPA is just a set of rules, not a tool.
* It defines how Java classes map to database tables.
* Requires a provider/implementation like Hibernate to actually work**.**

**Hibernate**

* Hibernate is an ORM framework that implements the JPA specification.
* It provides actual working classes to persist Java objects to the database.
* It supports features beyond JPA, such as HQL (Hibernate Query Language), lazy loading, caching, etc.
* Requires the developer to handle Session creation and transaction management manually (or with some abstraction).
* Hibernate is a tool that implements JPA.
* It handles the actual database interaction.
* You need to write more boilerplate code (open sessions, manage transactions).
* Supports both JPA-based and native Hibernate APIs.

**Hibernate Code Example:**

/\* Method to CREATE an employee in the database \*/

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;

}

**Spring Data JPA**

* Spring Data JPA is a Spring module that sits on top of JPA (and Hibernate).
* It simplifies development by eliminating boilerplate code.
* You only need to define a Repository interface — Spring generates the implementation at runtime.
* It supports CRUD operations, query derivation, pagination, and more.
* Uses @Transactional to automatically manage transactions.
* Spring Data JPA is an abstraction over JPA/Hibernate.
* It removes the need for writing DAO/Repository implementations.
* Automatically manages transactions and generates query methods.
* Common in Spring Boot applications.

**Spring Data JPA Code Example:**

// EmployeeRepository.java

public interface EmployeeRepository extends JpaRepository<Employee, Integer> {

}

// EmployeeService.java

@Autowired

private EmployeeRepository employeeRepository;

@Transactional

public void addEmployee(Employee employee) {

employeeRepository.save(employee);

}