Spring Data JPA with Spring Boot, Hibernate

1. Spring Data JPA - Quick Example:-

Creation of Project:-

* Group: com.cognizant
* Artifact: orm-learn
* Add Dependencies: Spring Data JPA, MySQL Driver, Spring Boot DevTools

Configuration of application.properties:-

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

spring.jpa.hibernate.ddl-auto=validate

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

logging.level.org.hibernate.SQL=trace

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

MySQL code:-

CREATE DATABASE ormlearn;

CREATE TABLE country (

co\_code VARCHAR(2) PRIMARY KEY,

co\_name VARCHAR(50)

);

INSERT INTO country VALUES ('IN', 'India'), ('US', 'United States');

Creation of Entity class(Country.java):-

@Entity

@Table(name = "country")

public class Country {

@Id @Column(name = "co\_code") private String code;

@Column(name = "co\_name") private String name;

}

Create Repository Interface:-

@Repository

public interface CountryRepository extends JpaRepository<Country, String> {

}

Create Service Class:-

@Service

public class CountryService {

@Autowired private CountryRepository repo;

@Transactional public List<Country> getAllCountries() { return repo.findAll(); }

@Transactional public void addCountry(Country c) { repo.save(c); }

@Transactional public void updateCountry(String code, String name) {

Country c = repo.findById(code).get(); c.setName(name); repo.save(c);

}

@Transactional public void deleteCountry(String code) { repo.deleteById(code); }

}

Test in OrmLearnApplication.java:-

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

countryService = context.getBean(CountryService.class);

testGetAllCountries();

1. Difference between JPA, Hibernate, and Spring Data JPA:-

JPA (Java Persistence API) is a specification defined by the Java Community Process (JSR 338) that outlines how Java applications should interact with relational databases. It is only a set of rules and interfaces and does not provide an actual implementation. JPA standardizes the way developers can persist, retrieve, and manage data in Java applications.

Hibernate is one of the most popular implementations of JPA. It is an Object-Relational Mapping (ORM) tool that follows the JPA specification and adds its own advanced features. Hibernate allows developers to map Java classes to database tables and provides capabilities like caching, lazy loading, and HQL (Hibernate Query Language). However, working directly with Hibernate or plain JPA still requires writing repetitive data access code such as entity manager handling and CRUD operations.

Spring Data JPA is a higher-level abstraction built on top of JPA and an implementation like Hibernate. It is part of the Spring ecosystem and integrates smoothly with Spring Boot. The main advantage of Spring Data JPA is that it eliminates the need for writing boilerplate DAO implementations. By simply defining a repository interface that extends JpaRepository, developers can automatically get access to common CRUD operations and even custom queries based on method naming conventions.