**SPRING DATA JPA**

**Hands on 1**

**Spring Data JPA - Quick Example**

**MODEL**

package com.example.country.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 = "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 + "]";

}

}

**REPOSITORY**

package com.example.country.repository;

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

import org.springframework.stereotype.Repository;

import com.example.country.model.country;

@Repository

public interface countryRepo extends JpaRepository<country, String> {

}

**SERVICE**

package com.example.country.service;

import java.util.List;

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

import org.springframework.stereotype.Service;

import com.example.country.model.country;

import com.example.country.repository.countryRepo;

import jakarta.transaction.Transactional;

@Service

public class countryService {

@Autowired

private countryRepo countryRepository;

@Transactional

public List<country> getAllCountries() {

return countryRepository.findAll();

}

}

**MAIN CLASS:**

package com.example.country;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import com.example.country.model.country;

import com.example.country.service.countryService;

import java.util.List;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.context.ApplicationContext;

@SpringBootApplication

public class CountryApplication {

private static countryService countryService;

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

public static void main(String[] args) {

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

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");

}

}

**Hands on 2**

**Hibernate XML Config implementation walk-through**

Object to Relational Database Mapping (ORM) in Hibernate XML

**1. Entity Class – Java POJO**

// Employee.java

public class Employee {

private int id;

private String firstName;

private String lastName;

private int salary;

// Getters and Setters

}

**2. Hibernate Mapping XML**

// Employee.hbm.xml

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

<!DOCTYPE hibernate-mapping PUBLIC

"-//Hibernate/Hibernate Mapping DTD 3.0//EN"

"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping>

<class name="Employee" table="EMPLOYEE">

<id name="id" type="int">

<column name="id"/>

<generator class="native"/>

</id>

<property name="firstName" column="first\_name" type="string"/>

<property name="lastName" column="last\_name" type="string"/>

<property name="salary" column="salary" type="int"/>

</class>

</hibernate-mapping>

SessionFactory

• Built once during application startup.

• It holds metadata and manages sessions.

• Created using hibernate.cfg.xml.

SessionFactory factory = new Configuration()

.configure("hibernate.cfg.xml")

.addResource("Employee.hbm.xml")

.buildSessionFactory();

Session

• Think of it as a connection with the database.

• Created by SessionFactory.

Session session = factory.openSession();

Transaction

• Used to group DB operations as a single unit (ACID).

• Needed for save, delete, etc.

Transaction tx = session.beginTransaction();

beginTransaction()

• Starts a new transaction.

• Required before making changes to the DB.

Transaction tx = session.beginTransaction();

commit()

• Saves all DB changes permanently.

tx.commit();

rollback()

• Cancels the current transaction (undoes changes).

tx.rollback();

session.save(Object)

• Saves an object into the DB.

Employee emp = new Employee();

emp.setFirstName("John");

emp.setLastName("Doe");

emp.setSalary(5000);

session.save(emp);

session.createQuery().list()

• Used for fetching multiple records.

List<Employee> list = session.createQuery("FROM Employee").list();

for (Employee e : list) {

System.out.println(e.getFirstName());

}

session.get(Class, id)

• Fetches a record by primary key.

• Returns null if not found.

Employee emp = session.get(Employee.class, 1);

session.delete(Object)

• Deletes the given entity object.

Employee emp = session.get(Employee.class, 1);

session.delete(emp);

**Hands on 3**

**Hibernate Annotation Config implementation walk-through**

Object to Relational Mapping (ORM) in Persistence Class Employee

Hibernate uses Java annotations to map Java class fields to database columns, eliminating the need for separate .hbm.xml files.

**// Employee.java – Persistence Class (POJO)**

import jakarta.persistence.\*;

@Entity

@Table(name = "EMPLOYEE")

public class Employee {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

@Column(name = "id")

private int id;

@Column(name = "first\_name")

private String firstName;

@Column(name = "last\_name")

private String lastName;

@Column(name = "salary")

private int salary;

// Getters and setters...

}

**Annotation Purpose**

@Entity Marks the class as a Hibernate entity (i.e., table mapped to a class).

@Table Specifies the DB table name for the entity.

@Id Identifies the primary key field.

@GeneratedValue Auto-generates primary key values (based on strategy).

@Column Maps a field to a DB column and allows specifying the column name.

Hibernate Configuration File **(hibernate.cfg.xml)**

This XML file tells Hibernate how to connect to the database and which annotated classes to use.

<?xml version='1.0' encoding='utf-8'?>

<!DOCTYPE hibernate-configuration PUBLIC

"-//Hibernate/Hibernate Configuration DTD 3.0//EN"

"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

<session-factory>

<!-- Database connection settings -->

<property name="hibernate.connection.driver\_class">com.mysql.cj.jdbc.Driver</property>

<property name="hibernate.connection.url">jdbc:mysql://localhost:3306/testdb</property>

<property name="hibernate.connection.username">root</property>

<property name="hibernate.connection.password">root</property>

<!-- SQL dialect -->

<property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>

<!-- Show executed SQL -->

<property name="hibernate.show\_sql">true</property>

<!-- Update the schema automatically -->

<property name="hibernate.hbm2ddl.auto">update</property>

<!-- Register annotated class -->

<mapping class="com.example.Employee"/>

</session-factory>

</hibernate-configuration>

hibernate.connection.driver\_class JDBC driver class (for MySQL it's com.mysql.cj.jdbc.Driver)

hibernate.connection.url JDBC URL pointing to your database

hibernate.connection.username DB username

hibernate.connection.password DB password

hibernate.dialect Tells Hibernate the SQL dialect to use (MySQLDialect for MySQL)

hibernate.show\_sql Prints SQL statements in console

hibernate.hbm2ddl.auto Auto DDL tool: update, create, validate, or none

<mapping class="..."/> Registers the annotated entity class

**Hands on 4**

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

1. Java Persistence API (JPA)

* What it is - A specification (JSR 338) for managing relational data in Java applications.
* Type - Only defines interfaces and rules – no actual code or implementation.
* Key Features - Annotations (@Entity, @Id, @OneToMany, etc.), EntityManager, JPQL (Java Persistence Query Language).
* Example Providers - Hibernate, EclipseLink, OpenJPA, etc. implement the JPA specification.

A blueprint for ORM in Java. It tells you what to do, but not how to do it.

1. Hibernate

* What it is - A concrete implementation of the JPA specification.
* Type - ORM framework and JPA provider.
* Key Features - Supports both JPA and its own native APIs (Session, Query, HQL).
* Extra Features - Lazy loading, caching, custom dialects, batch processing, etc.

Hibernate implements JPA and also provides advanced ORM features beyond the spec.

1. Spring Data JPA

* What it is - A part of Spring Data that provides abstraction over JPA (e.g., Hibernate).
* Type - Helper library that uses JPA provider (like Hibernate) underneath.
* Key Benefits
  + Removes boilerplate code
  + Auto-generates queries (findByName, etc.)
  + Integrates seamlessly with Spring Boot
  + Supports CrudRepository, JpaRepository, and more

| Transaction Management | Spring handles transactions behind the scenes with @Transactional |

Spring Data JPA simplifies JPA/Hibernate usage in Spring apps, like writing save(), findById() with no custom code.

**Hands on 5**

**Implement services for managing Country**

**MODEL**

package com.example.country.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;

// 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; }

@Override

public String toString() {

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

}

}

**REPOSITORY**

package com.example.country.repository;

import java.util.List;

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

import org.springframework.stereotype.Repository;

import com.example.country.model.country;

@Repository

public interface countryRepo extends JpaRepository<country, String> {

List<country> findByNameContainingIgnoreCase(String name);

}

**SERVICE**

package com.example.country.service;

import java.util.List;

import java.util.Optional;

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

import org.springframework.stereotype.Service;

import com.example.country.model.country;

import com.example.country.repository.countryRepo;

import jakarta.transaction.Transactional;

@Service

public class countryService {

@Autowired

private countryRepo countryRepository;

@Transactional

public List<country> getAllCountries() {

return countryRepository.findAll();

}

@Transactional

public country findCountryByCode(String code) throws Exception {

Optional<country> result = countryRepository.findById(code);

if (result.isPresent()) return result.get();

else throw new Exception("Country not found for code: " + code);

}

@Transactional

public void addCountry(country country) {

countryRepository.save(country);

}

@Transactional

public void updateCountry(String code, String newName) throws Exception {

country country = findCountryByCode(code);

country.setName(newName);

countryRepository.save(country);

}

@Transactional

public void deleteCountry(String code) {

countryRepository.deleteById(code);

}

@Transactional

public List<country> searchCountriesByName(String name) {

return countryRepository.findByNameContainingIgnoreCase(name);

}

}

**MAIN\_CLASS**

package com.example.country;

import java.util.List;

import org.slf4j.LoggerFactory;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.context.ConfigurableApplicationContext;

import com.example.country.model.country;

import com.example.country.service.countryService;

@SpringBootApplication

public class CountryApplication {

private static countryService countryService;

private static final org.slf4j.Logger LOGGER = LoggerFactory.getLogger(CountryApplication.class);

public static void main(String[] args) throws Exception {

ConfigurableApplicationContext context = SpringApplication.run(CountryApplication.class, args);

LOGGER.info("Inside main");

countryService = context.getBean(countryService.class);

testGetAllCountries();

testFindCountryByCode();

testAddCountry();

testUpdateCountry();

testDeleteCountry();

testSearchCountriesByName();

}

private static void testGetAllCountries() {

LOGGER.info("Start getAll");

List<country> list = countryService.getAllCountries();

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

LOGGER.info("End getAll");

}

private static void testFindCountryByCode() throws Exception {

LOGGER.info("Start findByCode");

country c = countryService.findCountryByCode("IN");

LOGGER.debug("country={}", c);

LOGGER.info("End findByCode");

}

private static void testAddCountry() {

LOGGER.info("Start add");

country c = new country();

c.setCode("ZZ");

c.setName("Zootopia");

countryService.addCountry(c);

LOGGER.info("End add");

}

private static void testUpdateCountry() throws Exception {

LOGGER.info("Start update");

countryService.updateCountry("ZZ", "Zootopia Updated");

LOGGER.info("End update");

}

private static void testDeleteCountry() {

LOGGER.info("Start delete");

countryService.deleteCountry("ZZ");

LOGGER.info("End delete");

}

private static void testSearchCountriesByName() {

LOGGER.info("Start search");

List<country> results = countryService.searchCountriesByName("land");

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

LOGGER.info("End search");

}

}

**Hands on 6**

**Find a country based on country code**

**MODEL**

package com.example.country.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;

// 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; }

@Override

public String toString() {

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

}

}

**REPOSITORY**

package com.example.country.repository;

import java.util.List;

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

import org.springframework.stereotype.Repository;

import com.example.country.model.country;

@Repository

public interface countryRepo extends JpaRepository<country, String> {

List<country> findByNameContainingIgnoreCase(String name);

}

**SERVICE**

package com.example.country.service;

import java.util.List;

import java.util.Optional;

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

import org.springframework.stereotype.Service;

import com.example.country.model.country;

import com.example.country.repository.countryRepo;

import jakarta.transaction.Transactional;

@Service

public class countryService {

@Autowired

private countryRepo countryRepository;

@Transactional

public List<country> getAllCountries() {

return countryRepository.findAll();

}

@Transactional

public country findCountryByCode(String code) throws Exception {

Optional<country> result = countryRepository.findById(code);

if (result.isPresent()) return result.get();

else throw new Exception("Country not found for code: " + code);

}

@Transactional

public void addCountry(country country) {

countryRepository.save(country);

}

@Transactional

public void updateCountry(String code, String newName) throws Exception {

country country = findCountryByCode(code);

country.setName(newName);

countryRepository.save(country);

}

@Transactional

public void deleteCountry(String code) {

countryRepository.deleteById(code);

}

@Transactional

public List<country> searchCountriesByName(String name) {

return countryRepository.findByNameContainingIgnoreCase(name);

}

}

**MAIN\_CLASS**

package com.example.country;

import java.util.List;

import org.slf4j.LoggerFactory;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.context.ConfigurableApplicationContext;

import com.example.country.model.country;

import com.example.country.service.countryService;

@SpringBootApplication

public class CountryApplication {

private static countryService countryService;

private static final org.slf4j.Logger LOGGER = LoggerFactory.getLogger(CountryApplication.class);

public static void main(String[] args) throws Exception {

ConfigurableApplicationContext context = SpringApplication.run(CountryApplication.class, args);

LOGGER.info("Inside main");

countryService = context.getBean(countryService.class);

testGetAllCountries();

testFindCountryByCode();

testAddCountry();

testUpdateCountry();

testDeleteCountry();

testSearchCountriesByName();

}

private static void testGetAllCountries() {

LOGGER.info("Start getAll");

List<country> list = countryService.getAllCountries();

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

LOGGER.info("End getAll");

}

private static void testFindCountryByCode() throws Exception {

LOGGER.info("Start findByCode");

country c = countryService.findCountryByCode("IN");

LOGGER.debug("country={}", c);

LOGGER.info("End findByCode");

}

private static void testAddCountry() {

LOGGER.info("Start add");

country c = new country();

c.setCode("ZZ");

c.setName("Zootopia");

countryService.addCountry(c);

LOGGER.info("End add");

}

private static void testUpdateCountry() throws Exception {

LOGGER.info("Start update");

countryService.updateCountry("ZZ", "Zootopia Updated");

LOGGER.info("End update");

}

private static void testDeleteCountry() {

LOGGER.info("Start delete");

countryService.deleteCountry("ZZ");

LOGGER.info("End delete");

}

private static void testSearchCountriesByName() {

LOGGER.info("Start search");

List<country> results = countryService.searchCountriesByName("land");

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

LOGGER.info("End search");

}

}