**Spring Data JPA with Spring Boot, Hibernate**

**Spring Data JPA - Quick Example**

**//country.java**

package com.cognizant.orm\_learn.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;

// Getters and Setters

@Override

public String toString() {

return "Country{" +

"code='" + code + '\'' +

", name='" + name + '\'' +

'}';

}

}

**//CountryRepository.java**

package com.cognizant.orm\_learn.repository;

import com.cognizant.orm\_learn.model.Country;

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

import org.springframework.stereotype.Repository;

@Repository

public interface CountryRepository extends JpaRepository<Country, String> {

}

**// CountryService.java**

package com.cognizant.orm\_learn.service;

import com.cognizant.orm\_learn.model.Country;

import com.cognizant.orm\_learn.repository.CountryRepository;

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

import org.springframework.stereotype.Service;

import org.springframework.transaction.annotation.Transactional;

import java.util.List;

@Service

public class CountryService {

@Autowired

private CountryRepository countryRepository;

@Transactional

public List<Country> getAllCountries() {

return countryRepository.findAll();

}

}

**//OmrLearnApplication.java**

package com.cognizant.orm\_learn;

import com.cognizant.orm\_learn.model.Country;

import com.cognizant.orm\_learn.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;

import java.util.List;

@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);

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");

}

}

**//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

# Oracle DB config

spring.datasource.driver-class-name=oracle.jdbc.OracleDriver

spring.datasource.url=jdbc:oracle:thin:@localhost:1521/XEPDB1

spring.datasource.username=ormlearn

spring.datasource.password=ormlearn

# Hibernate config

spring.jpa.hibernate.ddl-auto=validate

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

**//OmrLearnApplicationTest.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() {

}

}

**// schema.sql**

-- SQL Schema for Country table

-- This file demonstrates DDL operations and will be executed automatically by Spring Boot

-- Drop table if exists (for clean setup)

DROP TABLE IF EXISTS countries;

-- Create countries table

CREATE TABLE countries (

country\_id BIGINT AUTO\_INCREMENT PRIMARY KEY,

country\_code VARCHAR(3) NOT NULL UNIQUE,

iso\_code VARCHAR(2) NOT NULL,

country\_name VARCHAR(100) NOT NULL,

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

updated\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP

);

-- Create indexes for better query performance

CREATE INDEX idx\_country\_code ON countries(country\_code);

CREATE INDEX idx\_iso\_code ON countries(iso\_code);

CREATE INDEX idx\_country\_name ON countries(country\_name);

-- Insert some sample data (optional)

INSERT INTO countries (country\_code, iso\_code, country\_name) VALUES

('USA', 'US', 'United States'),

('GBR', 'GB', 'United Kingdom'),

('IND', 'IN', 'India'),

('DEU', 'DE', 'Germany'),

('FRA', 'FR', 'France'),

('JPN', 'JP', 'Japan'),

('CHN', 'CN', 'China'),

('CAN', 'CA', 'Canada'),

('AUS', 'AU', 'Australia'),

('BRA', 'BR', 'Brazil');

**//CountryServiceTest.java**

package com.cognizant.ormlearn.service;

import com.cognizant.ormlearn.model.Country;

import com.cognizant.ormlearn.repository.CountryRepository;

import org.junit.jupiter.api.Test;

import org.junit.jupiter.api.BeforeEach;

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

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

import org.springframework.test.context.ActiveProfiles;

import org.springframework.transaction.annotation.Transactional;

import java.util.List;

import static org.junit.jupiter.api.Assertions.\*;

/\*\*

\* Integration tests for CountryService

\*

\* This test class demonstrates:

\* - Spring Boot Test configuration

\* - Integration testing with H2 database

\* - JPA repository testing

\* - Service layer testing

\* - Transaction management in tests

\*/

@SpringBootTest

@ActiveProfiles("test")

@Transactional

public class CountryServiceTest {

@Autowired

private CountryService countryService;

@Autowired

private CountryRepository countryRepository;

@BeforeEach

void setUp() {

// Clean up database before each test

countryRepository.deleteAll();

}

@Test

void testCreateCountry() {

// Given

String countryCode = "TST";

String isoCode = "TS";

String countryName = "Test Country";

// When

Country createdCountry = countryService.createCountry(countryCode, isoCode, countryName);

// Then

assertNotNull(createdCountry);

assertNotNull(createdCountry.getId());

assertEquals(countryCode, createdCountry.getCountryCode());

assertEquals(isoCode, createdCountry.getIsoCode());

assertEquals(countryName, createdCountry.getCountryName());

assertNotNull(createdCountry.getCreatedAt());

assertNotNull(createdCountry.getUpdatedAt());

}

@Test

void testCreateDuplicateCountryCode() {

// Given

String countryCode = "DUP";

countryService.createCountry(countryCode, "DP", "Duplicate Country");

// When & Then

assertThrows(RuntimeException.class, () -> {

countryService.createCountry(countryCode, "D2", "Another Duplicate");

});

}

@Test

void testGetAllCountries() {

// Given

countryService.createCountry("US1", "U1", "Test Country 1");

countryService.createCountry("US2", "U2", "Test Country 2");

countryService.createCountry("US3", "U3", "Test Country 3");

// When

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

// Then

assertEquals(3, countries.size());

}

@Test

void testGetCountryById() {

// Given

Country createdCountry = countryService.createCountry("FND", "FN", "Find Country");

// When

Country foundCountry = countryService.getCountryById(createdCountry.getId());

// Then

assertNotNull(foundCountry);

assertEquals(createdCountry.getId(), foundCountry.getId());

assertEquals("FND", foundCountry.getCountryCode());

assertEquals("Find Country", foundCountry.getCountryName());

}

@Test

void testGetCountryByIdNotFound() {

// When & Then

assertThrows(RuntimeException.class, () -> {

countryService.getCountryById(999L);

});

}

@Test

void testUpdateCountry() {

// Given

Country country = countryService.createCountry("UPD", "UP", "Update Country");

country.setCountryName("Updated Country Name");

// When

Country updatedCountry = countryService.updateCountry(country);

// Then

assertEquals("Updated Country Name", updatedCountry.getCountryName());

assertEquals(country.getId(), updatedCountry.getId());

}

@Test

void testDeleteCountry() {

// Given

Country country = countryService.createCountry("DEL", "DL", "Delete Country");

Long countryId = country.getId();

// When

countryService.deleteCountry(countryId);

// Then

assertThrows(RuntimeException.class, () -> {

countryService.getCountryById(countryId);

});

}

@Test

void testFindByCountryCode() {

// Given

countryService.createCountry("SRC", "SR", "Search Country");

// When

Country foundCountry = countryService.findByCountryCode("SRC");

// Then

assertNotNull(foundCountry);

assertEquals("SRC", foundCountry.getCountryCode());

assertEquals("Search Country", foundCountry.getCountryName());

}

@Test

void testFindByCountryCodeNotFound() {

// When

Country foundCountry = countryService.findByCountryCode("NON");

// Then

assertNull(foundCountry);

}

@Test

void testFindByIsoCode() {

// Given

countryService.createCountry("ISO1", "IS", "ISO Country 1");

countryService.createCountry("ISO2", "IS", "ISO Country 2");

// When

List<Country> countries = countryService.findByIsoCode("IS");

// Then

assertEquals(2, countries.size());

}

@Test

void testFindByCountryNameContaining() {

// Given

countryService.createCountry("US1", "U1", "United States");

countryService.createCountry("UK1", "U2", "United Kingdom");

countryService.createCountry("FR1", "F1", "France");

// When

List<Country> countries = countryService.findByCountryNameContaining("United");

// Then

assertEquals(2, countries.size());

assertTrue(countries.stream().allMatch(c -> c.getCountryName().contains("United")));

}

@Test

void testGetCountryCount() {

// Given

countryService.createCountry("CNT1", "C1", "Count Country 1");

countryService.createCountry("CNT2", "C2", "Count Country 2");

// When

long count = countryService.getCountryCount();

// Then

assertEquals(2, count);

}

@Test

void testExistsByCountryCode() {

// Given

countryService.createCountry("EXT", "EX", "Exists Country");

// When & Then

assertTrue(countryService.existsByCountryCode("EXT"));

assertFalse(countryService.existsByCountryCode("NEX"));

}

@Test

void testFindAllCountryCodes() {

// Given

countryService.createCountry("CD1", "C1", "Code Country 1");

countryService.createCountry("CD2", "C2", "Code Country 2");

countryService.createCountry("CD3", "C3", "Code Country 3");

// When

List<String> codes = countryService.findAllCountryCodes();

// Then

assertEquals(3, codes.size());

assertTrue(codes.contains("CD1"));

assertTrue(codes.contains("CD2"));

assertTrue(codes.contains("CD3"));

}

@Test

void testFindCountriesWithLongNames() {

// Given

countryService.createCountry("SHT", "SH", "Short");

countryService.createCountry("LNG", "LG", "Very Long Country Name");

// When

List<Country> longNameCountries = countryService.findCountriesWithLongNames(10);

// Then

assertEquals(1, longNameCountries.size());

assertEquals("Very Long Country Name", longNameCountries.get(0).getCountryName());

}

}

**//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

http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.cognizant</groupId>

<artifactId>spring-data-jpa-example</artifactId>

<version>1.0.0</version>

<packaging>jar</packaging>

<name>Spring Data JPA Quick Example</name>

<description>Spring Data JPA with Spring Boot and Hibernate Example</description>

<parent>

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

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

<version>2.7.0</version>

<relativePath/>

</parent>

<properties>

<java.version>11</java.version>

<maven.compiler.source>11</maven.compiler.source>

<maven.compiler.target>11</maven.compiler.target>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

</properties>

<dependencies>

<!-- Spring Boot Starter Web -->

<dependency>

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

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

</dependency>

<!-- Spring Boot Starter Data JPA -->

<dependency>

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

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

</dependency>

<!-- H2 Database (for development and testing) -->

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

<scope>runtime</scope>

</dependency>

<!-- MySQL Connector (optional, for production) -->

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<scope>runtime</scope>

</dependency>

<!-- Spring Boot Starter Test -->

<dependency>

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

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

<scope>test</scope>

</dependency>

<!-- Spring Boot Starter Validation -->

<dependency>

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

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

</dependency>

<!-- Spring Boot DevTools (optional, for development) -->

<dependency>

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

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

<scope>runtime</scope>

<optional>true</optional>

</dependency>

</dependencies>

<build>

<plugins>

<!-- Spring Boot Maven Plugin -->

<plugin>

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

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

</plugin>

<!-- Maven Compiler Plugin -->

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.8.1</version>

<configuration>

<source>11</source>

<target>11</target>

</configuration>

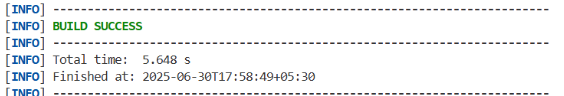
</plugin>

</plugins>

</build>

</project>

**OUTPUT:**

****