**SUPERSET ID : 6375627**

**Week 3: Spring Data JPA with Spring Boot, Hibernate**

**Hands on 2**

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

**What Is Java Persistence API?**

The Java Persistence API provides a specification for persisting, reading, and managing data from your Java object to relational tables in the database.

**What Is Hibernate Framework?**

Hibernate is an object-relational mapping solution for Java environments. Object-relational mapping or ORM is the programming technique to map application domain model objects to the relational database tables.

Hibernate provides a reference implementation of the Java Persistence API that makes it a great choice as an ORM tool with the benefits of loose coupling.

Example: Below diagram shows an Object Relational Mapping between the Student Java class and the students table in the database.

**What Is Spring Data JPA?**

Spring Data is a part of the Spring Framework. The goal of Spring Data repository abstraction is to significantly reduce the amount of boilerplate code required to implement data access layers for various persistence stores.

Spring Data JPA is not a JPA provider. It is a library/framework that adds an extra layer of abstraction on the top of our JPA provider (like Hibernate).

**What Is the Difference Between Hibernate and Spring Data JPA?**

Hibernate is a JPA implementation, while Spring Data JPA is a JPA Data Access Abstraction.

Spring Data offers a solution to GenericDao custom implementations. It can also generate JPA queries on your behalf through method name conventions.

With Spring Data, you may use Hibernate, Eclipse Link, or any other JPA provider. A very interesting benefit is that you can control transaction boundaries declaratively using the @Transactional annotation.

Spring Data JPA is not an implementation or JPA provider, it's just an abstraction used to significantly reduce the amount of boilerplate code required to implement data access layers for various persistence stores.

Hibernate provides a reference implementation of the Java Persistence API that makes it a great choice as an ORM tool with the benefits of loose coupling**.**

**Hands on :**

Implementation of Query Methods in Spring Data JPA

Introduction

Spring Data JPA is a powerful module in the Spring ecosystem that provides a high-level abstraction for accessing relational databases using JPA. One of its most valuable features is the Query Methods, which allow developers to define database queries just by method names, without writing any JPQL or SQL.

These methods are based on a method naming convention, where Spring interprets the method name and auto-generates the appropriate query.

Why Use Query Methods?

* Removes boilerplate code
* Reduces the need for manually written queries
* Ensures faster development and cleaner code
* Integrates seamlessly with Spring Boot

Common Keywords Used in Query Methods

| Keyword | Description |
| --- | --- |
| findBy | Selects by a field |
| Containing | SQL LIKE %value% |
| StartingWith | SQL LIKE value% |
| GreaterThan | SQL > |
| LessThan | SQL < |
| Between | SQL BETWEEN x AND y |
| OrderBy | Sorts the result |
| Top | Fetches limited results (e.g. top 3) |

**1. spring-data-jpa-handson: Demonstrate implementation of Query Methods feature of Spring Data JPA**

**OrmLearnApplication.java**

testCountriesByPartialName();

testCountriesByPartialNameOrdered();

testCountriesByAlphabet();   
private static void testCountriesByPartialName() {

LOGGER.info("Start testCountriesByPartialName");

List<Country> countries = countryService.getCountriesByNameContaining("ou");

LOGGER.debug("Countries containing 'ou': {}", countries);

LOGGER.info("End testCountriesByPartialName");

}

private static void testCountriesByPartialNameOrdered() {

LOGGER.info("Start testCountriesByPartialNameOrdered");

List<Country> countries = countryService.getCountriesByNameContainingOrdered("ou");

LOGGER.debug("Countries containing 'ou' ordered: {}", countries);

LOGGER.info("End testCountriesByPartialNameOrdered");

}

private static void testCountriesByAlphabet() {

LOGGER.info("Start testCountriesByAlphabet");

List<Country> countries = countryService.getCountriesByNameStartingWith("Z");

LOGGER.debug("Countries starting with 'Z': {}", countries);

LOGGER.info("End testCountriesByAlphabet");

}

**CountryService.java**

@Transactional

public List<Country> getCountriesByNameContaining(String fragment) {

return countryRepository.findByNameContaining(fragment);

}

@Transactional

public List<Country> getCountriesByNameContainingOrdered(String fragment) {

return countryRepository.findByNameContainingOrderByNameAsc(fragment);

}

@Transactional

public List<Country> getCountriesByNameStartingWith(String alphabet) {

return countryRepository.findByNameStartingWith(alphabet);

}

**CountryRepository.java**

package com.cognizant.ormlearn.repository;

import java.util.List;

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

import org.springframework.stereotype.Repository;

import com.cognizant.ormlearn.Country;

@Repository

public interface CountryRepository extends JpaRepository<Country, String> {

List<Country> findByNameContaining(String nameFragment);

List<Country> findByNameContainingOrderByNameAsc(String nameFragment);

List<Country> findByNameStartingWith(String alphabet);

}

**OUTPUT:**

