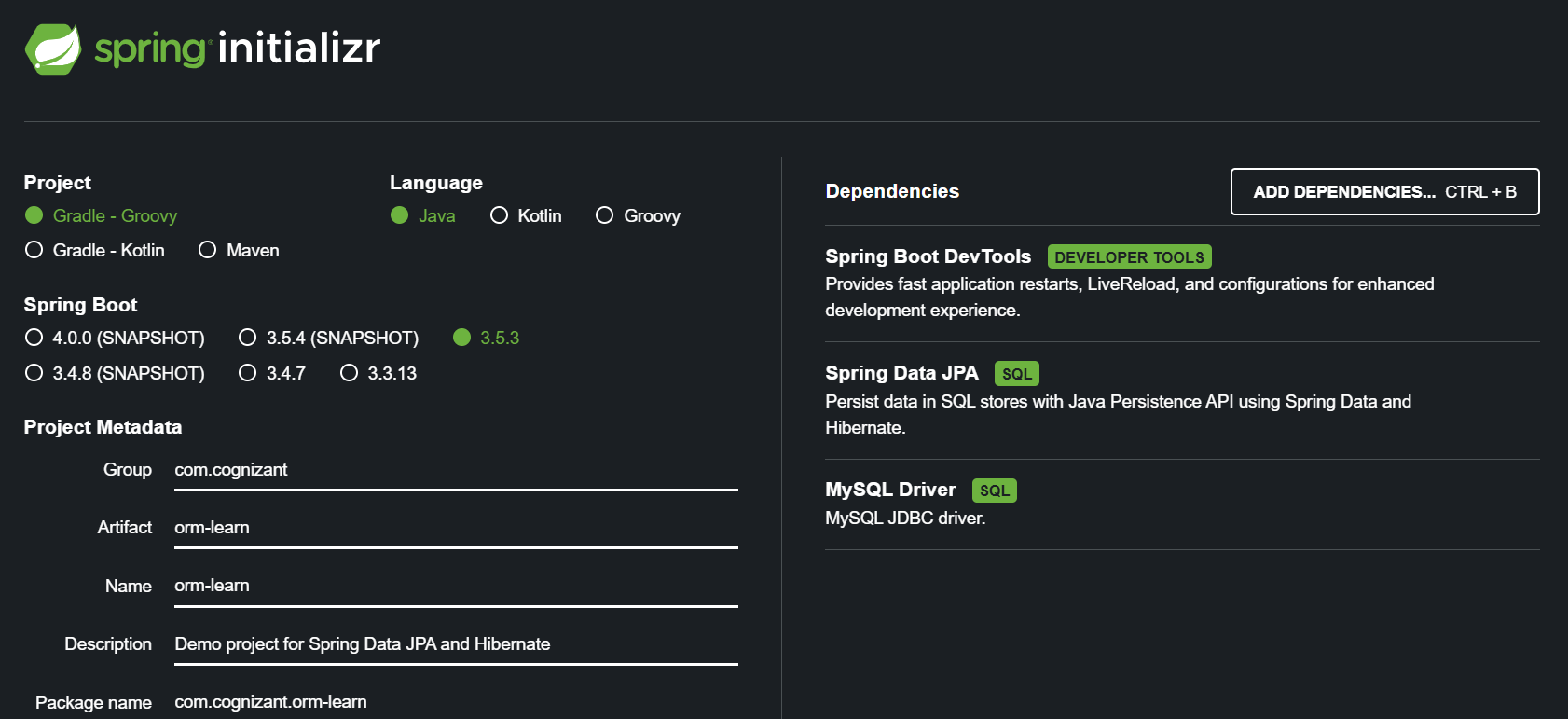
**1. Spring Data JPA - Quick Example**

### Software Prerequisites

Make sure the following are installed:

* **MySQL Server 8.0**
* **MySQL Workbench 8**
* **Eclipse IDE for Enterprise Java Developers (2019-03 R)**
* **Maven 3.6.2**

**1. Create Spring Boot Project using Spring Initializr**



* Click **Generate** → download .zip
* **Extract** the .zip into your Eclipse workspace

2. **Import into Eclipse**

* File > Import > Maven > Existing Maven Projects
* Browse to extracted folder → click **Finish**

### 3. **Setup MySQL Database**

* Open MySQL command line or Workbench:

create schema ormlearn;

### 4. **Configure application.properties**

Edit src/main/resources/application.properties:

# Spring & app logging

logging.level.org.springframework=info

logging.level.com.cognizant=debug

# Hibernate logs

logging.level.org.hibernate.SQL=trace

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

# Console log format

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

# DB connection

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 settings

spring.jpa.hibernate.ddl-auto=validate

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

### 5. **Build Project**

Open terminal (in project directory):

mvn clean package -Dhttp.proxyHost=proxy.cognizant.com -Dhttp.proxyPort=6050 -Dhttps.proxyHost=proxy.cognizant.com -Dhttps.proxyPort=6050 -Dhttp.proxyUser=123456

### 6. **Log Setup in Main Class**

Modify OrmLearnApplication.java:

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

@SpringBootApplication

public class OrmLearnApplication {

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

public static void main(String[] args) {

SpringApplication.run(OrmLearnApplication.class, args);

LOGGER.info("Inside main");

}

}

Run the application → check console for "Inside main" log.

**Country Table in MySQL**

create table country (

co\_code varchar(2) primary key,

co\_name varchar(50)

);

insert into country values ('IN', 'India');

insert into country values ('US', 'United States of America');

## **Entity Class: Country.java**

**Path**: com.cognizant.ormlearn.model.Country

package com.cognizant.ormlearn.model;

import javax.persistence.\*;

@Entity

@Table(name = "country")

public class Country {

@Id

@Column(name = "co\_code")

private String code;

@Column(name = "co\_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 Interface: CountryRepository.java**

**Path**: com.cognizant.ormlearn.repository

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> {

}

## **Service Class: CountryService.java**

**Path**: com.cognizant.ormlearn.service

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();

}

}

## **Testing in OrmLearnApplication.java**

Add:

import com.cognizant.ormlearn.service.CountryService;

import com.cognizant.ormlearn.model.Country;

import org.springframework.context.ApplicationContext;

import java.util.List;

private static CountryService countryService;

private static void testGetAllCountries() {

LOGGER.info("Start");

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

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

LOGGER.info("End");

}

public static void main(String[] args) {

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

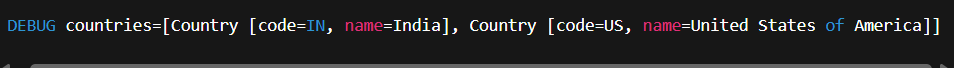
countryService = context.getBean(CountryService.class);

testGetAllCountries();

}

## **Final Output:**

Run OrmLearnApplication.java → console should show SQL logs and a list of countries like:

****

**2. Explain the difference between Java Persistence API, Hibernate and Spring Data JPA**

**1. JPA (Java Persistence API)**  
JPA is a specification, not a library.  
It defines a set of rules and interfaces to handle data persistence in Java applications.  
It allows you to map Java objects to database tables (ORM - Object Relational Mapping).

What it provides:  
Standard annotations and APIs, such as:

* @Entity – Marks a class as a table in the database
* @Id, @GeneratedValue – Primary key mapping
* @OneToMany, @ManyToOne – Relationship mappings
* EntityManager – Interface to manage database operations

What it doesn't do:  
It does not do anything by itself — there’s no actual code behind the interfaces.  
You need a JPA provider (like Hibernate) to use JPA in real projects.

**2. Hibernate**  
Hibernate is an ORM framework that provides the actual implementation of JPA.  
It handles the conversion between Java objects and database tables.

You can use Hibernate in two ways:

* As a native ORM tool using its own APIs like Session, SessionFactory
* As a JPA provider by implementing the JPA interfaces (most common in modern applications)

Hibernate adds extra features:

* Caching: Improves performance by storing frequently accessed data
* HQL (Hibernate Query Language): A query language similar to SQL but for Java objects
* Lazy and Eager Fetching
* Auto DDL: Can auto-create tables based on entities

**3. Spring Data JPA:**   
A Spring abstraction layer built on top of JPA and Hibernate.  
Makes it easy to create repository classes for database access.  
Reduces boilerplate code — no need to write queries or even implement classes.

Key Features:

* Interfaces like JpaRepository and CrudRepository
* You just define method names and Spring generates the queries

Method naming conventions:

List<User> findByEmail(String email);

List<User> findByAgeGreaterThan(int age);

**The major Differences are:-**

| **Feature / Aspect** | **JPA (Java Persistence API)** | **Hibernate** | **Spring Data JPA** |
| --- | --- | --- | --- |
| **Type** | Specification (API) | Framework (JPA Implementation + Native ORM) | Spring Abstraction over JPA and Hibernate |
| **Purpose** | Defines standard for ORM in Java | Implements JPA and provides ORM functionality | Simplifies JPA/Hibernate usage in Spring applications |
| **Who provides it** | Java EE / Jakarta EE (JSR 338) | Red Hat | Spring Framework (Pivotal/VMware) |
| **Implements functionality?** | No (just interfaces & annotations) | Yes | Yes (by wrapping Hibernate/JPA provider) |
| **Can run standalone?** | No | Yes | No (needs Spring ecosystem) |
| **Used APIs** | EntityManager, JPA annotations | Session, SessionFactory, HQL | JpaRepository, CrudRepository, PagingAndSortingRepository |
| **Query language** | JPQL | HQL (extends JPQL) | JPQL / method name queries / native SQL via @Query |
| **Extra features** | None | Caching, Interceptors, Custom ID generators | Method query generation, pagination, sorting, auditing |

**5. Implement services for managing Country**

### 1. Populate Country Table

In MySQL Workbench or CLI:

DELETE FROM country;

-- Paste full insert statements you were given

INSERT INTO country (co\_code, co\_name) VALUES ("AF", "Afghanistan");

INSERT INTO country (co\_code, co\_name) VALUES ("AL", "Albania");

### 2. Update Repository for Custom Query

CountryRepository.java

@Repository

public interface CountryRepository extends JpaRepository<Country, String> {

List<Country> findByNameContainingIgnoreCase(String name);

}

3. Update CountryService

@Service

public class CountryService {

@Autowired

private CountryRepository countryRepository;

@Transactional

public List<Country> getAllCountries() {

return countryRepository.findAll();

}

@Transactional

public Country findCountryByCode(String code) {

return countryRepository.findById(code)

.orElseThrow(() -> new RuntimeException("Country not found"));

}

@Transactional

public void addCountry(Country country) {

if (countryRepository.existsById(country.getCode())) {

throw new RuntimeException("Country already exists");

}

countryRepository.save(country);

}

@Transactional

public void updateCountry(String code, String newName) {

Country country = findCountryByCode(code);

country.setName(newName);

countryRepository.save(country);

}

@Transactional

public void deleteCountry(String code) {

countryRepository.deleteById(code);

}

@Transactional

public List<Country> findCountriesByNameContaining(String name) {

return countryRepository.findByNameContainingIgnoreCase(name);

}

}

4. Test Each Method in OrmLearnApplication.java

Test Methods

private static void testFindCountryByCode() {

LOGGER.info("Start testFindCountryByCode");

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

LOGGER.debug("Country: {}", c);

LOGGER.info("End testFindCountryByCode");

}

private static void testAddCountry() {

LOGGER.info("Start testAddCountry");

Country c = new Country();

c.setCode("ZZ");

c.setName("Zootopia");

countryService.addCountry(c);

LOGGER.debug("Added: {}", countryService.findCountryByCode("ZZ"));

LOGGER.info("End testAddCountry");

}

private static void testUpdateCountry() {

LOGGER.info("Start testUpdateCountry");

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

LOGGER.debug("Updated: {}", countryService.findCountryByCode("ZZ"));

LOGGER.info("End testUpdateCountry");

}

private static void testDeleteCountry() {

LOGGER.info("Start testDeleteCountry");

countryService.deleteCountry("ZZ");

try {

countryService.findCountryByCode("ZZ");

} catch (Exception e) {

LOGGER.debug("Deleted successfully: {}", e.getMessage());

}

LOGGER.info("End testDeleteCountry");

}

private static void testFindByPartialName() {

LOGGER.info("Start testFindByPartialName");

List<Country> countries = countryService.findCountriesByNameContaining("an");

LOGGER.debug("Found: {}", countries);

LOGGER.info("End testFindByPartialName");

}

**6. Find a country based on country code**

### 1. Create Custom Exception

**Path:** com.cognizant.ormlearn.service.exception

**File:** CountryNotFoundException.java

package com.cognizant.ormlearn.service.exception;

public class CountryNotFoundException extends Exception {

public CountryNotFoundException(String message) {

super(message);

}

}

### 2. Update CountryService

**Add method to** CountryService.java**:**

@Transactional

public Country findCountryByCode(String countryCode) throws CountryNotFoundException {

Optional<Country> result = countryRepository.findById(countryCode);

if (!result.isPresent()) {

throw new CountryNotFoundException("Country with code " + countryCode + " not found");

}

return result.get();

}

### 3. Add Test Method in OrmLearnApplication.java

**Add this test method:**

private static void getCountryByCodeTest() {

LOGGER.info("Start");

try {

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

LOGGER.debug("Country: {}", country);

} catch (CountryNotFoundException e) {

LOGGER.error("Exception: {}", e.getMessage());

}

LOGGER.info("End");

}

4. Update main() Method:

OrmLearnApplication.java

package com.cognizant.ormlearn;

import java.util.List;

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 com.cognizant.ormlearn.model.Country;

import com.cognizant.ormlearn.service.CountryService;

import com.cognizant.ormlearn.service.exception.CountryNotFoundException;

@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");

// Inject service

countryService = context.getBean(CountryService.class);

// Run the test

getCountryByCodeTest();

}

private static void getCountryByCodeTest() {

LOGGER.info("Start");

try {

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

LOGGER.debug("Country: {}", country);

} catch (CountryNotFoundException e) {

LOGGER.error("Exception: {}", e.getMessage());

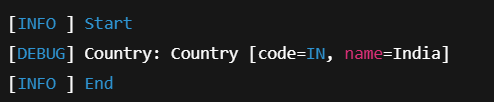
}

LOGGER.info("End");

}

}

**Output:**



**7. Add a new country**

1. Update CountryService.java:

@Transactional

public void addCountry(Country country) {

countryRepository.save(country);

}

### 2. Add Test Method in OrmLearnApplication.java

**Create a new method** testAddCountry() **in the class:**

private static void testAddCountry() {

LOGGER.info("Start");

Country newCountry = new Country();

newCountry.setCode("ZZ");

newCountry.setName("Zootopia");

countryService.addCountry(newCountry);

try {

Country country = countryService.findCountryByCode("ZZ");

LOGGER.debug("Country: {}", country);

} catch (CountryNotFoundException e) {

LOGGER.error("Country not found after insert: {}", e.getMessage());

}

LOGGER.info("End");

}

### 3. Call the Test Method from main()

**Inside the** main() **method of** OrmLearnApplication**:**

package com.cognizant.ormlearn;

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 com.cognizant.ormlearn.model.Country;

import com.cognizant.ormlearn.service.CountryService;

import com.cognizant.ormlearn.service.exception.CountryNotFoundException;

@SpringBootApplication

public class OrmLearnApplication {

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

private static CountryService countryService;

public static void main(String[] args) {

// Start Spring context and fetch CountryService bean

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

LOGGER.info("Inside main");

countryService = context.getBean(CountryService.class);

// Call the test method to add a country

testAddCountry();

}

private static void testAddCountry() {

LOGGER.info("Start");

// Create new country object

Country newCountry = new Country();

newCountry.setCode("ZZ");

newCountry.setName("Zootopia");

// Save it to the database

countryService.addCountry(newCountry);

// Retrieve to verify

try {

Country country = countryService.findCountryByCode("ZZ");

LOGGER.debug("Country: {}", country);

} catch (CountryNotFoundException e) {

LOGGER.error("Country not found after insert: {}", e.getMessage());

}

LOGGER.info("End");

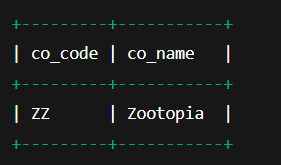
}

}

### Check in MySQL DB

Run this query in MySQL Workbench or CLI to confirm:

SELECT \* FROM country WHERE co\_code = 'ZZ';



**2nd Hands-on**

**Demonstrate implementation of Query Methods feature of Spring Data JPA**

## Step 1: Update Your Entity (if needed)

Let’s assume a Country entity like this:

@Entity

@Table(name = "country")

public class Country {

@Id

@Column(name = "co\_code")

private String code;

@Column(name = "co\_name")

private String name;

// getters, setters, toString()

}

## Step 2: Add Query Methods in Repository

Update 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.model.Country;

@Repository

public interface CountryRepository extends JpaRepository<Country, String> {

// 1. Search by name containing text

List<Country> findByNameContaining(String keyword);

// 2. Search by name starting with text

List<Country> findByNameStartingWith(String prefix);

// 3. Sort by name (ascending)

List<Country> findAllByOrderByNameAsc();

// 4. Top N by name length (use limit via Pageable or Top keyword)

List<Country> findTop3ByOrderByNameDesc();

// 5. Custom with ignore case

List<Country> findByNameContainingIgnoreCase(String keyword);

}

Step 3: Call from CountryService

public List<Country> searchByName(String keyword) {

return countryRepository.findByNameContaining(keyword);

}

public List<Country> getCountriesStartingWith(String prefix) {

return countryRepository.findByNameStartingWith(prefix);

}

public List<Country> getSortedCountries() {

return countryRepository.findAllByOrderByNameAsc();

}

public List<Country> getTop3CountriesByName() {

return countryRepository.findTop3ByOrderByNameDesc();

}

Step 4: Add Test in OrmLearnApplication

private static void testQueryMethods() {

LOGGER.info("Start");

LOGGER.debug("Containing 'an': {}", countryService.searchByName("an"));

LOGGER.debug("Starting with 'A': {}", countryService.getCountriesStartingWith("A"));

LOGGER.debug("Sorted by name: {}", countryService.getSortedCountries());

LOGGER.debug("Top 3 by name desc: {}", countryService.getTop3CountriesByName());

LOGGER.info("End");

}

Step 5: Call the test from main()

testQueryMethods();

**Demonstrate implementation of O/R Mapping**

## Step 1: Create Entities

## **Department.java**

@Entity

@Table(name = "department")

public class Department {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

// One department has many employees

@OneToMany(mappedBy = "department", fetch = FetchType.LAZY, cascade = CascadeType.ALL)

private List<Employee> employees = new ArrayList<>();

// getters, setters, toString()

}

**Employee.java**

@Entity

@Table(name = "employee")

public class Employee {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

// Many employees belong to one department

@ManyToOne(fetch = FetchType.EAGER)

@JoinColumn(name = "department\_id") // foreign key column

private Department department;

// Many employees can be assigned to many projects

@ManyToMany(fetch = FetchType.LAZY)

@JoinTable(

name = "employee\_project",

joinColumns = @JoinColumn(name = "employee\_id"),

inverseJoinColumns = @JoinColumn(name = "project\_id")

)

private Set<Project> projects = new HashSet<>();

// getters, setters, toString()

}

**Project.java**

@Entity

@Table(name = "project")

public class Project {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

// Many projects can have many employees

@ManyToMany(mappedBy = "projects")

private Set<Employee> employees = new HashSet<>();

// getters, setters, toString()

}

Step 2: Repositories

@Repository

public interface DepartmentRepository extends JpaRepository<Department, Long> {}

@Repository

public interface EmployeeRepository extends JpaRepository<Employee, Long> {}

@Repository

public interface ProjectRepository extends JpaRepository<Project, Long> {}

Step 3: Saving Data (Test)

private static void testORMapping() {

LOGGER.info("Start");

Department dept = new Department();

dept.setName("Engineering");

Employee emp1 = new Employee();

emp1.setName("Alice");

emp1.setDepartment(dept);

Employee emp2 = new Employee();

emp2.setName("Bob");

emp2.setDepartment(dept);

dept.setEmployees(Arrays.asList(emp1, emp2));

departmentRepository.save(dept); // cascade saves employees too

Project project = new Project();

project.setName("Apollo");

emp1.setProjects(Set.of(project));

emp2.setProjects(Set.of(project));

project.setEmployees(Set.of(emp1, emp2));

projectRepository.save(project);

LOGGER.debug("Saved Department with Employees and Projects");

LOGGER.info("End");

}

Step 3: Saving Data (Test)

private static void testORMapping() {

LOGGER.info("Start");

Department dept = new Department();

dept.setName("Engineering");

Employee emp1 = new Employee();

emp1.setName("Alice");

emp1.setDepartment(dept);

Employee emp2 = new Employee();

emp2.setName("Bob");

emp2.setDepartment(dept);

dept.setEmployees(Arrays.asList(emp1, emp2));

departmentRepository.save(dept); // cascade saves employees too

Project project = new Project();

project.setName("Apollo");

emp1.setProjects(Set.of(project));

emp2.setProjects(Set.of(project));

project.setEmployees(Set.of(emp1, emp2));

projectRepository.save(project);

LOGGER.debug("Saved Department with Employees and Projects");

LOGGER.info("End");

}

**3rd Hands On**

**Demonstrate writing Hibernate Query Language and Native Query**

## 1. HQL (Hibernate Query Language)

### Create a method using @Query in CountryRepository:

@Repository

public interface CountryRepository extends JpaRepository<Country, String> {

// HQL - search country by name using JPQL (HQL)

@Query("SELECT c FROM Country c WHERE c.name = :name")

Country findCountryByNameHQL(@Param("name") String name);

}

## 2. Native SQL Query

## Native query example in the same repository:

@Repository

public interface CountryRepository extends JpaRepository<Country, String> {

// Native SQL query - search using real table/column names

@Query(value = "SELECT \* FROM country WHERE co\_name = :name", nativeQuery = true)

Country findCountryByNameNative(@Param("name") String name);

}

## 3. Example Use in Service

public Country getCountryByNameHQL(String name) {

return countryRepository.findCountryByNameHQL(name);

}

public Country getCountryByNameNative(String name) {

return countryRepository.findCountryByNameNative(name);

}

4. Example Test in OrmLearnApplication.java

private static void testHQLandNativeQueries() {

LOGGER.info("Start");

Country country1 = countryService.getCountryByNameHQL("India");

LOGGER.debug("HQL Result: {}", country1);

Country country2 = countryService.getCountryByNameNative("India");

LOGGER.debug("Native SQL Result: {}", country2);

LOGGER.info("End");

}

And invoke it in your main():

testHQLandNativeQueries();