**Student Name:** KARTHYAYENI P

**Registration No:** 22CSR089

**Course/Batch:** KONGU ENGINEERING COLLEGE (B.E COMPUTER SCIENCE AND ENGINEERING)

**HANDS ON 1: SPRING DATA JPA – QUICK EXAMPLE**

**Introduction:**

This Spring Boot application provides a quick and practical demonstration of **Spring Data JPA** by performing basic CRUD operations on a Country entity mapped to a MySQL database. It shows how **Spring Boot**, **JPA**, and **Hibernate** work together to simplify ORM (Object-Relational Mapping) with minimal boilerplate code.

**Objective:**

* To configure and connect a Spring Boot application with a MySQL database using Spring Data JPA.
* To create a JPA-managed entity class (Country) and interact with the database using a JpaRepository interface.
* To retrieve all records from the country table using a service method and log the results with SLF4J for monitoring and debugging.

**Implementation Breakdown:**

**OrmLearnApplication.java:**

package com.cognizant.orm\_learn;

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);

LOGGER.info("Inside main");

countryService = context.getBean(CountryService.class);

testGetAllCountries();

}

private static void testGetAllCountries() {

LOGGER.info("Start testGetAllCountries");

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

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

LOGGER.info("End testGetAllCountries");

}

}

**Country.java:**

package com.cognizant.orm\_learn;

import jakarta.persistence.\*;

@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 + "]";

}

}

**CountryRepository.java:**

package com.cognizant.orm\_learn;

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;

import org.springframework.transaction.annotation.Transactional;

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

import org.springframework.stereotype.Service;

import java.util.List;

@Service

public class CountryService {

@Autowired

private CountryRepository countryRepository;

@Transactional

public List<Country> getAllCountries() {

return countryRepository.findAll();

}

}

**Application.properties:**

spring.application.name=orm-learn

logging.level.org.springframework=info

logging.level.com.cognizant=debug

logging.level.org.hibernate.SQL=trace

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

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

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.datasource.url=jdbc:mysql://localhost:3306/ormlearn

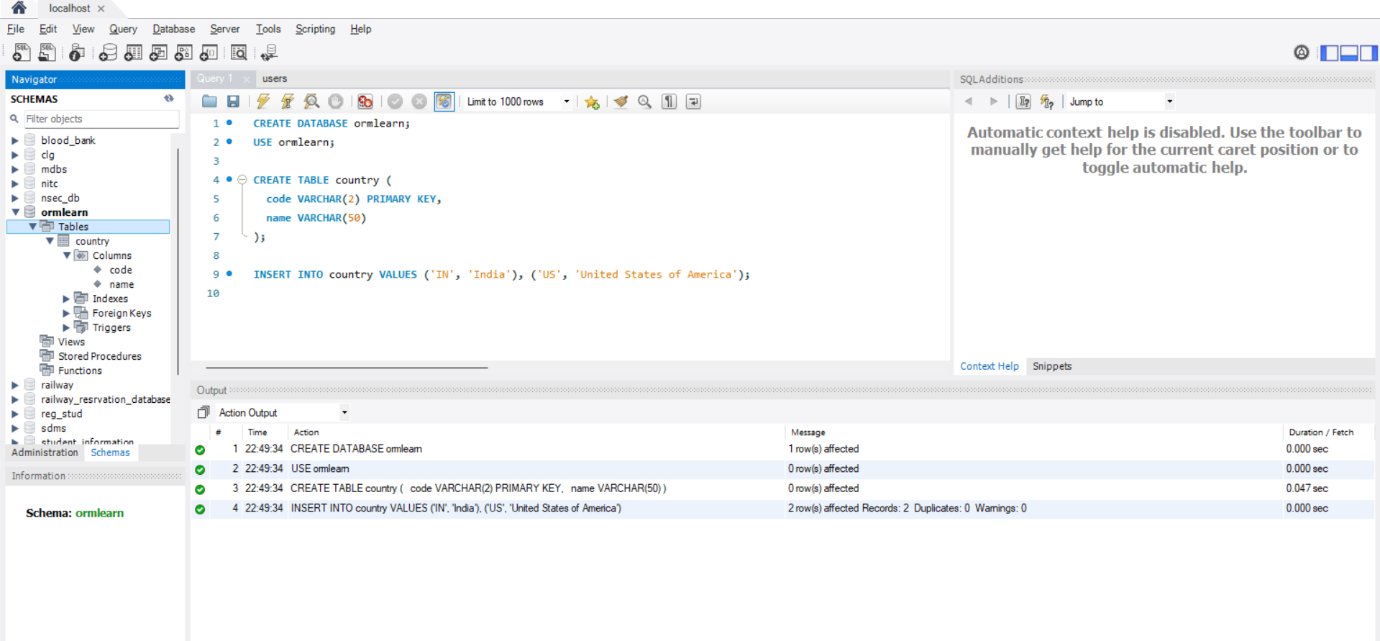
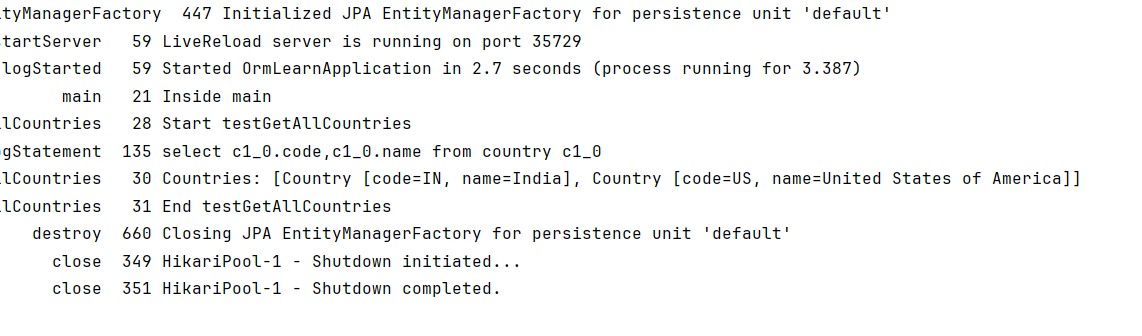
spring.datasource.username=root

spring.datasource.password=root

spring.jpa.hibernate.ddl-auto=validate

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

**Output:**

****

**Conclusion:**

This quick example effectively demonstrates how Spring Data JPA abstracts database interaction, allowing developers to focus on business logic without writing SQL. By leveraging Spring Boot’s auto-configuration and JPA’s ORM features, the application achieves seamless data access and maintainable code structure.

**HANDS ON 4: DIFFERENCE BETWEEN JPA, HIBERNATE AND SPRING DATA**

**Introduction:**

This Spring Boot application demonstrates how to build a basic **Employee Management System** using **Spring Data JPA**, which internally relies on **JPA** for persistence abstraction and **Hibernate** as the ORM provider. It allows seamless interaction with a MySQL database without writing complex SQL or boilerplate DAO code.

**Objective:**

* To create an Employee entity mapped to a MySQL table using **JPA annotations** like @Entity and @Id.
* To use **Spring Data JPA** (JpaRepository) to perform standard CRUD operations automatically.
* To leverage **Hibernate** as the JPA provider for managing entity persistence and schema generation (ddl-auto=update).

**Implementation Breakdown:**

**EmployeeDemoApplication.java:**

package com.example.employee\_demo;

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

import org.springframework.boot.CommandLineRunner;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class EmployeeDemoApplication implements CommandLineRunner {

@Autowired

private EmployeeService employeeService;

public static void main(String[] args) {

SpringApplication.run(EmployeeDemoApplication.class, args);

}

@Override

public void run(String... args) throws Exception {

Employee e1 = new Employee("Kavya", "IT");

Employee e2 = new Employee("Kavitha", "HR");

employeeService.addEmployee(e1);

employeeService.addEmployee(e2);

System.out.println("All Employees:");

employeeService.getAllEmployees().forEach(System.out::println);

}

}

**Employee.java:**

package com.example.employee\_demo;

import jakarta.persistence.\*;

@Entity

public class Employee {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Integer id;

private String name;

private String department;

public Employee() {}

public Employee(String name, String department) {

this.name = name;

this.department = department;

}

public Integer getId() { return id; }

public void setId(Integer id) { this.id = id; }

public String getName() { return name; }

public void setName(String name) { this.name = name; }

public String getDepartment() { return department; }

public void setDepartment(String department) { this.department = department; }

@Override

public String toString() {

return "Employee [id=" + id + ", name=" + name + ", department=" + department + "]";

}

}

**EmployeeRepository.java:**

package com.example.employee\_demo;

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

import org.springframework.stereotype.Repository;

@Repository

public interface EmployeeRepository extends JpaRepository<Employee, Integer> {

}

**EmployeeService.java:**

package com.example.employee\_demo;

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

import org.springframework.stereotype.Service;

import jakarta.transaction.Transactional;

import java.util.List;

@Service

public class EmployeeService {

@Autowired

private EmployeeRepository employeeRepository;

@Transactional

public void addEmployee(Employee emp) {

employeeRepository.save(emp);

}

public List<Employee> getAllEmployees() {

return employeeRepository.findAll();

}

}

**Application.properties:**

spring.application.name=employee-demo

spring.datasource.url=jdbc:mysql://localhost:3306/employeedb

spring.datasource.username=root

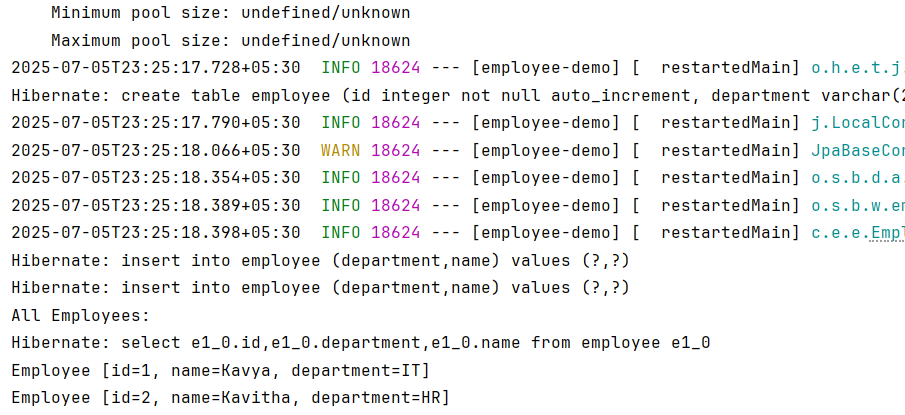
spring.datasource.password=root

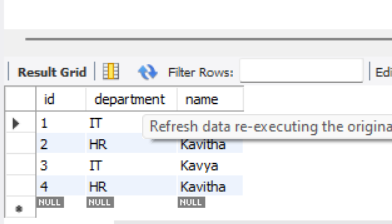
spring.jpa.hibernate.ddl-auto=update

spring.jpa.show-sql=true

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

**Output:**

****

****

**Difference:**

| **Feature** | **JPA** | **Hibernate** | **Spring Data JPA** |
| --- | --- | --- | --- |
| Type | Specification (API) | Implementation (ORM provider) | Framework/Abstraction |
| Role | Defines standard annotations & interfaces for persistence | Implements JPA & adds extra features (e.g., caching, lazy loading) | Simplifies JPA-based development with built-in CRUD, queries |
| Exampel in Code | @Entity, @Id, EntityManager | hibernate.dialect, @GeneratedValue(strategy=...) | extends JpaRepository<Employee, Integer> |
| Who provides it? | Oracle (Java EE spec) | Red Hat | Spring Framework |

**Conclusion:**

The project effectively combines the power of **JPA**, **Hibernate**, and **Spring Data JPA** to reduce boilerplate, enhance maintainability, and abstract database interaction. It shows how developers can build scalable data-driven applications with minimal configuration and maximum productivity using Spring Boot.