**Student Name:** Jayaanth S R

**SuperSet ID :** 6420871

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

**Introduction:**

This project is a hands-on implementation of **Spring Data JPA** using Hibernate to interact with a **MySQL database** in a Spring Boot application. The project demonstrates how to configure a Spring Boot application with JPA, define entity mappings, and fetch data from a relational database using the repository pattern.

**Objective:**

* To configure a Maven-based Spring Boot project with dependencies for Spring Data JPA and MySQL.
* To demonstrate Object-Relational Mapping (ORM) using the @Entity, @Table, and @Column annotations.
* To implement a Repository interface extending JpaRepository for performing database operations.

**Implementation Breakdown:**

**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;

@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);

LOGGER.info("Inside main");

testGetAllCountries();

}

private static void testGetAllCountries() {

LOGGER.info("Start");

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

for (Country country : countries) {

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

}

LOGGER.info("End");

}

}

**Country.java:**

package com.cognizant.ormlearn.model;

import javax.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.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> {

}

**CountryService.java:**

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();

}

}

**Application.properties:**

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

spring.datasource.username=root

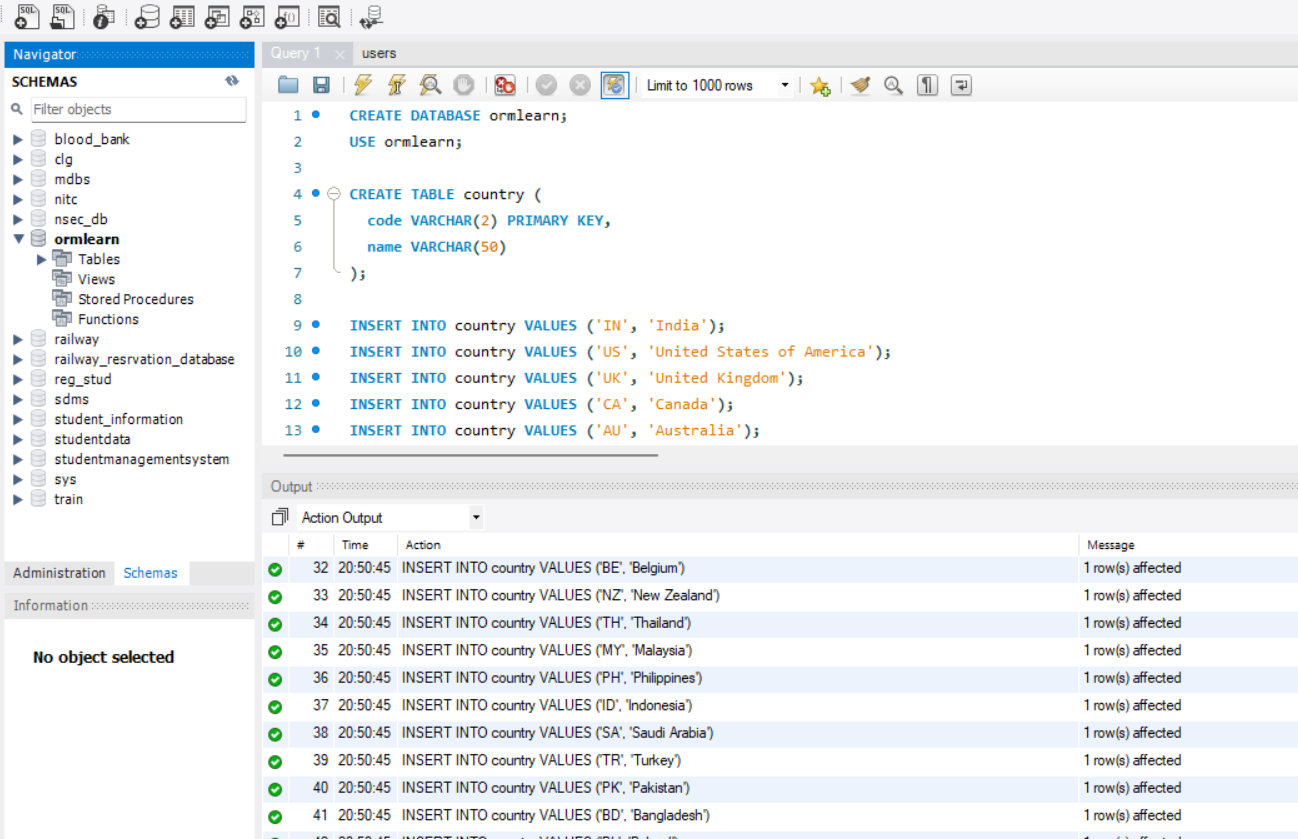
spring.datasource.password=root

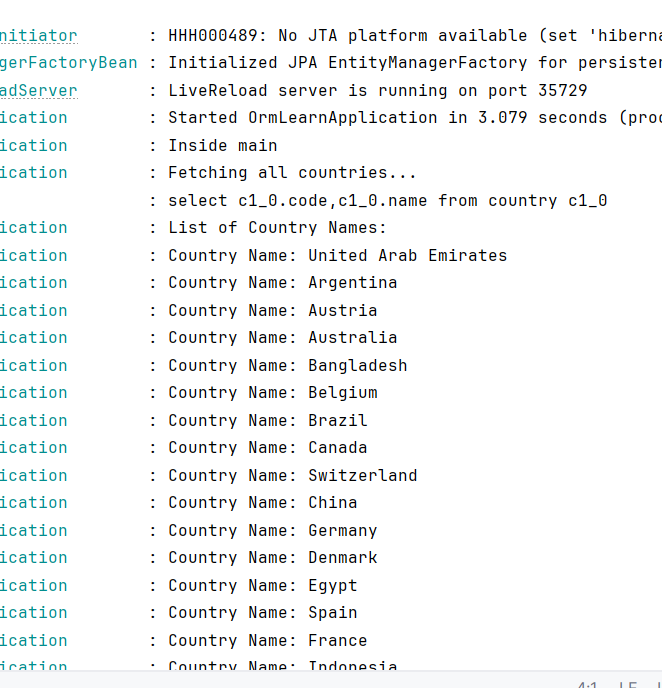
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.jpa.hibernate.ddl-auto=validate

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

**Output:**

****

****

****

**Conclusion:**

This exercise successfully demonstrates how to build a **Spring Boot JPA application** connected to a MySQL database. The use of **Spring Data JPA**, layered architecture (Model, Repository, Service), and proper logging ensures that the application is well-structured, modular, and ready for extension into more complex CRUD operations or queries.

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

**Introduction:**

In Java-based enterprise applications, persisting data efficiently is crucial. **Java Persistence API (JPA)** provides a standard specification for ORM, **Hibernate** is its widely-used implementation, and **Spring Data JPA** simplifies the development process by offering an abstraction layer over JPA providers like Hibernate. Understanding their roles helps in designing robust and maintainable backend systems.

**Objective:**

* To understand the individual roles of JPA, Hibernate, and Spring Data JPA in Java persistence.
* To demonstrate how Hibernate implements JPA specifications for ORM.
* To show how Spring Data JPA simplifies database access by reducing boilerplate code.
* To compare traditional Hibernate code with modern Spring Data JPA approaches using code snippets.

**Implementation Breakdown:**

**EmployeeReviewApplication.java:**

package com.example.employee\_review;

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

import org.springframework.boot.\*;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import java.time.LocalDate;

@SpringBootApplication

public class EmployeeReviewApplication implements CommandLineRunner {

@Autowired

private EmployeeService employeeService;

public static void main(String[] args) {

SpringApplication.run(EmployeeReviewApplication.class, args);

}

@Override

public void run(String... args) {

Employee e1 = new Employee("Amit", "Engineering");

employeeService.addEmployee(e1);

Skill s1 = new Skill("Java", "Advanced", e1);

Skill s2 = new Skill("Spring Boot", "Intermediate", e1);

employeeService.addSkill(s1);

employeeService.addSkill(s2);

PerformanceReview r1 = new PerformanceReview("Excellent project delivery", LocalDate.now(), 9, e1);

employeeService.addPerformanceReview(r1);

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

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

}

}

**Employee.java:**

package com.example.employee\_review;

import jakarta.persistence.\*;

import java.util.List;

@Entity

public class Employee {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Integer id;

private String name;

private String department;

@OneToMany(mappedBy = "employee", cascade = CascadeType.ALL)

private List<Skill> skills;

@OneToMany(mappedBy = "employee", cascade = CascadeType.ALL)

private List<PerformanceReview> reviews;

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; }

public List<Skill> getSkills() { return skills; }

public void setSkills(List<Skill> skills) { this.skills = skills; }

public List<PerformanceReview> getReviews() { return reviews; }

public void setReviews(List<PerformanceReview> reviews) { this.reviews = reviews; }

@Override

public String toString() {

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

}

}

**Skill.java:**

package com.example.employee\_review;

import jakarta.persistence.\*;

@Entity

public class Skill {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Integer id;

private String skillName;

private String proficiencyLevel;

@ManyToOne

@JoinColumn(name = "employee\_id")

private Employee employee;

public Skill() {}

public Skill(String skillName, String proficiencyLevel, Employee employee) {

this.skillName = skillName;

this.proficiencyLevel = proficiencyLevel;

this.employee = employee;

}

public Integer getId() { return id; }

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

public String getSkillName() { return skillName; }

public void setSkillName(String skillName) { this.skillName = skillName; }

public String getProficiencyLevel() { return proficiencyLevel; }

public void setProficiencyLevel(String proficiencyLevel) { this.proficiencyLevel = proficiencyLevel; }

public Employee getEmployee() { return employee; }

public void setEmployee(Employee employee) { this.employee = employee; }

@Override

public String toString() {

return "Skill [id=" + id + ", skillName=" + skillName + ", proficiencyLevel=" + proficiencyLevel + "]";

}

}

**PerfomanceReview.java:**

package com.example.employee\_review;

import jakarta.persistence.\*;

import java.time.LocalDate;

@Entity

public class PerformanceReview {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Integer id;

private LocalDate reviewDate;

private String reviewer;

private String comments;

private int rating;

@ManyToOne

@JoinColumn(name = "employee\_id")

private Employee employee;

public PerformanceReview() {}

public PerformanceReview(String comments, LocalDate reviewDate, int rating, Employee employee) {

this.reviewDate = reviewDate;

this.reviewer = "Manager";

this.comments = comments;

this.rating = rating;

this.employee = employee;

}

public Integer getId() { return id; }

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

public LocalDate getReviewDate() { return reviewDate; }

public void setReviewDate(LocalDate reviewDate) { this.reviewDate = reviewDate; }

public String getReviewer() { return reviewer; }

public void setReviewer(String reviewer) { this.reviewer = reviewer; }

public String getComments() { return comments; }

public void setComments(String comments) { this.comments = comments; }

public int getRating() { return rating; }

public void setRating(int rating) { this.rating = rating; }

public Employee getEmployee() { return employee; }

public void setEmployee(Employee employee) { this.employee = employee; }

@Override

public String toString() {

return "PerformanceReview [id=" + id + ", reviewer=" + reviewer + ", rating=" + rating + ", comments=" + comments + "]";

}

}

**EmployeeRepository.java:**

package com.example.employee\_review;

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

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

**SkillRepository.java:**

package com.example.employee\_review;

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

public interface SkillRepository extends JpaRepository<Skill, Integer> {}

**PerformanceReviewRepository.java:**

package com.example.employee\_review;

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

public interface PerformanceReviewRepository extends JpaRepository<PerformanceReview, Integer> {}

**EmployeeService.java:**

package com.example.employee\_review;

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 employeeRepo;

@Autowired

private SkillRepository skillRepo;

@Autowired

private PerformanceReviewRepository reviewRepo;

@Transactional

public void addEmployee(Employee e) {

employeeRepo.save(e);

}

public List<Employee> getAllEmployees() {

return employeeRepo.findAll();

}

@Transactional

public void addSkill(Skill skill) {

skillRepo.save(skill);

}

@Transactional

public void addPerformanceReview(PerformanceReview review) {

reviewRepo.save(review);

}

}

**Application.properties:**

spring.application.name=EmployeeReview

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

spring.datasource.username=root

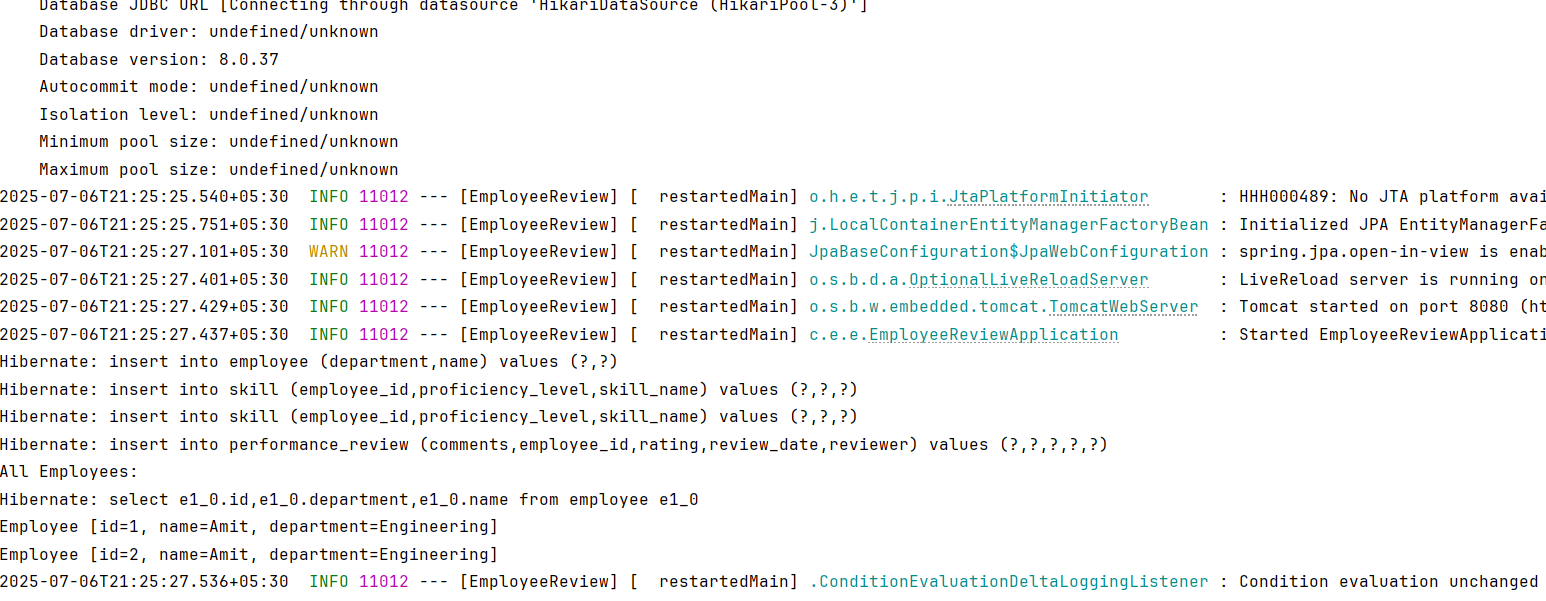
spring.datasource.password=Kavya@#0906

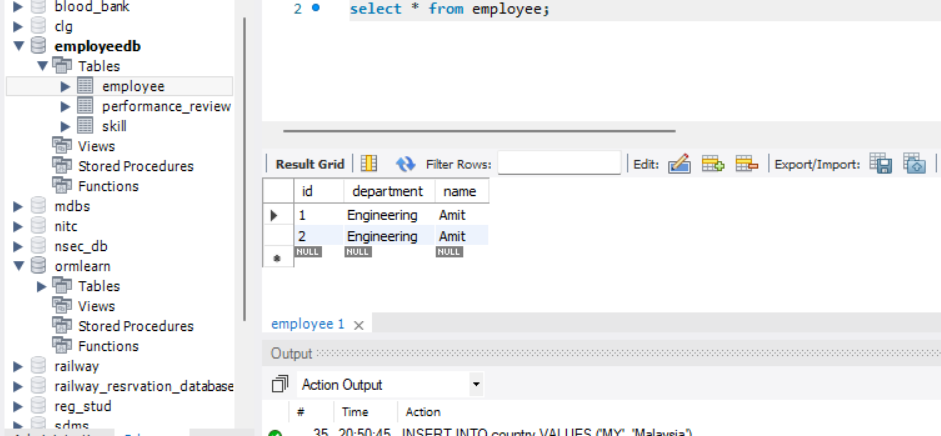
spring.jpa.hibernate.ddl-auto=update

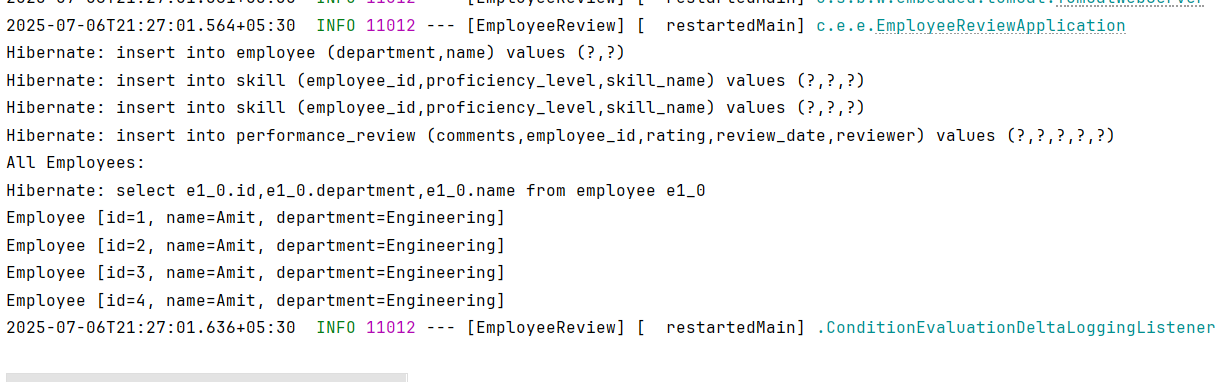
spring.jpa.show-sql=true

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:**

JPA defines the *what*, Hibernate implements the *how*, and Spring Data JPA simplifies the *usage*. By using **Spring Data JPA** in combination with **Hibernate**, developers can focus more on business logic and less on repetitive boilerplate code, while still maintaining full control over database interactions through JPA standards.