Spring Data JPA - Hands-on 4 Completion Report

# Overview

This document provides a detailed explanation and implementation steps for Hands-on 4: Understanding the differences between JPA, Hibernate, and Spring Data JPA with practical code examples for persisting employee data in a MySQL database.

# 1. Difference Between JPA, Hibernate, and Spring Data JPA

## Java Persistence API (JPA)

- JPA is a specification (JSR 338) for managing relational data in Java applications.  
- It defines a set of interfaces and annotations but does not provide implementation.  
- Hibernate is a popular JPA implementation.

## Hibernate

- Hibernate is an ORM (Object Relational Mapping) framework that provides a concrete implementation of the JPA specification.  
- Handles all database interaction, object lifecycle, and mapping Java objects to database tables.

## Spring Data JPA

- A part of the Spring ecosystem that builds on JPA and Hibernate.  
- Provides repository abstraction to reduce boilerplate code.  
- Simplifies CRUD operations, queries, and transaction management.  
- Does not provide its own implementation but uses Hibernate or any other JPA provider.

# 2. Code Comparison

## Hibernate Example

/\* Method to CREATE an employee in the database \*/  
public Integer addEmployee(Employee employee){  
 Session session = factory.openSession();  
 Transaction tx = null;  
 Integer employeeID = null;  
   
 try {  
 tx = session.beginTransaction();  
 employeeID = (Integer) session.save(employee);   
 tx.commit();  
 } catch (HibernateException e) {  
 if (tx != null) tx.rollback();  
 e.printStackTrace();   
 } finally {  
 session.close();   
 }  
 return employeeID;  
}

## Spring Data JPA Example

EmployeeRepository.java

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

EmployeeService.java

@Autowired  
private EmployeeRepository employeeRepository;  
  
@Transactional  
public void addEmployee(Employee employee) {  
 employeeRepository.save(employee);  
}

# 3. Implementation Guide

## Step 1: Create a Spring Boot Maven Project

- Go to https://start.spring.io/  
- Group: com.cognizant  
- Artifact: orm-learn  
- Add Dependencies: Spring Data JPA, MySQL Driver  
- Generate the project and import into Eclipse IDE.

## Step 2: Configure Database

application.properties

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=yourpassword  
  
spring.jpa.hibernate.ddl-auto=update  
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect

## Step 3: Create Entity Class

Employee.java

@Entity  
@Table(name = "employee")  
public class Employee {  
  
 @Id  
 @GeneratedValue(strategy = GenerationType.IDENTITY)  
 private int id;  
  
 private String name;  
 private BigDecimal salary;  
  
 // Getters and Setters  
}

## Step 4: Create Repository

EmployeeRepository.java

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

## Step 5: Create Service

EmployeeService.java

@Service  
public class EmployeeService {  
  
 @Autowired  
 private EmployeeRepository employeeRepository;  
  
 @Transactional  
 public void addEmployee(Employee employee) {  
 employeeRepository.save(employee);  
 }  
  
 public List<Employee> getAllEmployees() {  
 return employeeRepository.findAll();  
 }  
}

## Step 6: Test Using Main Class

OrmLearnApplication.java

@SpringBootApplication  
public class OrmLearnApplication implements CommandLineRunner {  
  
 @Autowired  
 private EmployeeService employeeService;  
  
 public static void main(String[] args) {  
 SpringApplication.run(OrmLearnApplication.class, args);  
 }  
  
 @Override  
 public void run(String... args) throws Exception {  
 Employee emp = new Employee();  
 emp.setName("John");  
 emp.setSalary(new BigDecimal("10000"));  
  
 employeeService.addEmployee(emp);  
  
 employeeService.getAllEmployees().forEach(System.out::println);  
 }  
}

# Conclusion

This hands-on demonstrates how Spring Data JPA simplifies interaction with the database by abstracting away the boilerplate code required by Hibernate and JPA. The implementation illustrates an Employee use case showing repository and service integration.