**1] Spring Data JPA Example: Student Management System**

You are building a system to manage students. Each student has an **ID**, **name**, **email**, and **department**. You want to use Spring Boot, Spring Data JPA, and MySQL to:

* Save students to the database
* Retrieve all students

**Create Spring Boot Project**

Go to [https://start.spring.io](https://start.spring.io/):

* **Group**: com.example
* **Artifact**: student-demo
* **Description**: Spring Boot project for managing students
* **Dependencies**:
  + Spring Web
  + Spring Data JPA
  + MySQL Driver
  + Spring Boot DevTools
* Click **Generate**, extract ZIP, and import into Eclipse (File > Import > Maven > Existing Maven Projects)

**Create MySQL Schema**

CREATE DATABASE studentdb;

USE studentdb;

CREATE TABLE student (

id INT PRIMARY KEY,

name VARCHAR(50),

email VARCHAR(100),

department VARCHAR(50)

);

INSERT INTO student VALUES (1, 'Alice', 'alice@example.com', 'Computer Science');

INSERT INTO student VALUES (2, 'Bob', 'bob@example.com', 'Mechanical Engineering');

**Configure application.properties**

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

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

logging.level.org.springframework=info

logging.level.org.hibernate.SQL=debug

**Student.java**

package com.example.studentdemo.model;

import javax.persistence.\*;

@Entity

@Table(name = "student")

public class Student {

@Id

private int id;

private String name;

private String email;

private String department;

// Getters and setters

public int getId() { return id; }

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

public String getName() { return name; }

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

public String getEmail() { return email; }

public void setEmail(String email) { this.email = email; }

public String getDepartment() { return department; }

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

@Override

public String toString() {

return "Student [id=" + id + ", name=" + name + ", email=" + email + ", department=" + department + "]";

}

}

**StudentRepository.java**

package com.example.studentdemo.repository;

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

import org.springframework.stereotype.Repository;

import com.example.studentdemo.model.Student;

@Repository

public interface StudentRepository extends JpaRepository<Student, Integer> {

}

**StudentService.java**

package com.example.studentdemo.service;

import com.example.studentdemo.model.Student;

import com.example.studentdemo.repository.StudentRepository;

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

import org.springframework.stereotype.Service;

import javax.transaction.Transactional;

import java.util.List;

@Service

public class StudentService {

@Autowired

private StudentRepository studentRepository;

@Transactional

public List<Student> getAllStudents() {

return studentRepository.findAll();

}

}

**StudentDemoApplication.java**

package com.example.studentdemo;

import com.example.studentdemo.model.Student;

import com.example.studentdemo.service.StudentService;

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 StudentDemoApplication {

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

private static StudentService studentService;

public static void main(String[] args) {

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

studentService = context.getBean(StudentService.class);

testGetAllStudents();

}

private static void testGetAllStudents() {

LOGGER.info("Start");

List<Student> students = studentService.getAllStudents();

students.forEach(student -> LOGGER.debug("Student: {}", student));

LOGGER.info("End");

}

}

## 2] Difference between JPA, Hibernate, and Spring Data JPA

### Java Persistence API (JPA)

* Definition: JPA is a Java Specification (JSR 338) that defines how Java objects interact with a relational database.
* Nature: It is just an API, not an implementation.
* Role: It provides interfaces and annotations (@Entity, @Id, etc.) for object-relational mapping (ORM).
* Usage: Developers need a JPA provider (like Hibernate) to actually persist objects.
* Key Point: Think of JPA as a set of rules or standards.

### Hibernate

* Definition: Hibernate is a popular ORM tool and a JPA implementation.
* Role: It implements all of JPA's interfaces and provides additional features (e.g., lazy loading, caching).
* Usage: Hibernate can be used with or without JPA.
* Key Point: Hibernate provides the engine that works under the hood of JPA to perform operations like insert, update, delete, and query.

### Spring Data JPA

* Definition: Spring Data JPA is a part of the Spring Data project that simplifies JPA-based data access layers.
* Nature: It is not a JPA provider itself. It works on top of JPA and abstracts away the boilerplate code.
* Role:
  + Reduces the need to write DAO classes.
  + Automatically generates repository methods from method names.
  + Manages transactions, pagination, sorting, and query derivation

### Code Comparison

### Hibernate

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

// Repository Interface

public interface EmployeeRepository extends JpaRepository<Employee, Integer> {

}

// Service Layer

@Autowired

private EmployeeRepository employeeRepository;

@Transactional

public void addEmployee(Employee employee) {

employeeRepository.save(employee);

}