**Spring Data JPA with Spring Boot, Hibernate**

### **Query Methods in Spring Data JPA**

### **Project Setup**

1. **Spring Boot Dependencies (in pom.xml or Spring Initializr):**
   * Spring Web
   * Spring Data JPA
   * H2 Database (or MySQL)
   * Lombok (optional)

### **application.properties (if using H2):**

spring.datasource.url=jdbc:h2:mem:testdb

spring.datasource.driverClassName=org.h2.Driver

spring.datasource.username=sa

spring.datasource.password=

spring.jpa.database-platform=org.hibernate.dialect.H2Dialect

spring.h2.console.enabled=true

spring.jpa.show-sql=true

spring.jpa.hibernate.ddl-auto=create

### **Step 1: Create Entity Class**

import jakarta.persistence.\*;

import java.util.Date;

@Entity

public class Product {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

private Double price;

@Temporal(TemporalType.DATE)

private Date createdAt;

// Constructors

public Product() {}

public Product(String name, Double price, Date createdAt) {

this.name = name;

this.price = price;

this.createdAt = createdAt;

}

// Getters and setters

// (Omit or use Lombok's @Data if preferred)

}

### **Step 2: Repository Interface using Query Methods**

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

import java.util.Date;

import java.util.List;

public interface ProductRepository extends JpaRepository<Product, Long> {

// Contains (search by partial text)

List<Product> findByNameContaining(String name);

// Starts with

List<Product> findByNameStartingWith(String prefix);

// Greater than price

List<Product> findByPriceGreaterThan(Double price);

// Less than price

List<Product> findByPriceLessThan(Double price);

// Between two dates

List<Product> findByCreatedAtBetween(Date startDate, Date endDate);

// Top 3 cheapest products

List<Product> findTop3ByOrderByPriceAsc();

// Top 3 most expensive products

List<Product> findTop3ByOrderByPriceDesc();

// Sort by name

List<Product> findAllByOrderByNameAsc();

}

### **Step 3: Add Service Class (Optional)**

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

import org.springframework.stereotype.Service;

import java.util.Date;

import java.util.List;

@Service

public class ProductService {

@Autowired

private ProductRepository productRepository;

public List<Product> getProductsByNameContaining(String keyword) {

return productRepository.findByNameContaining(keyword);

}

public List<Product> getProductsStartingWith(String prefix) {

return productRepository.findByNameStartingWith(prefix);

}

public List<Product> getProductsPriceGreaterThan(Double price) {

return productRepository.findByPriceGreaterThan(price);

}

public List<Product> getProductsPriceLessThan(Double price) {

return productRepository.findByPriceLessThan(price);

}

public List<Product> getProductsBetweenDates(Date start, Date end) {

return productRepository.findByCreatedAtBetween(start, end);

}

public List<Product> getTop3CheapestProducts() {

return productRepository.findTop3ByOrderByPriceAsc();

}

public List<Product> getTop3ExpensiveProducts() {

return productRepository.findTop3ByOrderByPriceDesc();

}

public List<Product> getAllSortedByName() {

return productRepository.findAllByOrderByNameAsc();

}

}

### **Step 4: Add Some Sample Data in Main Class**

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

import org.springframework.boot.CommandLineRunner;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import java.text.SimpleDateFormat;

@SpringBootApplication

public class JpaQueryMethodDemoApplication implements CommandLineRunner {

@Autowired

private ProductRepository productRepository;

public static void main(String[] args) {

SpringApplication.run(JpaQueryMethodDemoApplication.class, args);

}

@Override

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

SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");

productRepository.save(new Product("Laptop", 50000.0, sdf.parse("2024-06-01")));

productRepository.save(new Product("Mobile", 20000.0, sdf.parse("2024-06-15")));

productRepository.save(new Product("Mouse", 500.0, sdf.parse("2024-07-01")));

productRepository.save(new Product("Monitor", 8000.0, sdf.parse("2024-07-02")));

productRepository.save(new Product("MacBook", 100000.0, sdf.parse("2024-07-03")));

}

}

### **Sample Calls in Controller (Optional)**

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

import org.springframework.web.bind.annotation.\*;

import java.text.SimpleDateFormat;

import java.util.Date;

import java.util.List;

@RestController

@RequestMapping("/products")

public class ProductController {

@Autowired

private ProductService productService;

@GetMapping("/search")

public List<Product> searchByName(@RequestParam String keyword) {

return productService.getProductsByNameContaining(keyword);

}

@GetMapping("/prefix")

public List<Product> searchByPrefix(@RequestParam String prefix) {

return productService.getProductsStartingWith(prefix);

}

@GetMapping("/greater")

public List<Product> getByGreaterThan(@RequestParam Double price) {

return productService.getProductsPriceGreaterThan(price);

}

@GetMapping("/less")

public List<Product> getByLessThan(@RequestParam Double price) {

return productService.getProductsPriceLessThan(price);

}

@GetMapping("/between")

public List<Product> getBetweenDates(@RequestParam String start, @RequestParam String end) throws Exception {

SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");

Date startDate = sdf.parse(start);

Date endDate = sdf.parse(end);

return productService.getProductsBetweenDates(startDate, endDate);

}

@GetMapping("/top3cheap")

public List<Product> getTop3Cheapest() {

return productService.getTop3CheapestProducts();

}

@GetMapping("/top3expensive")

public List<Product> getTop3Expensive() {

return productService.getTop3ExpensiveProducts();

}

@GetMapping("/sorted")

public List<Product> getSortedByName() {

return productService.getAllSortedByName();

}

}

### **Summary of Methods:**

| **Method Type** | **Query Method Used** |
| --- | --- |
| Search by partial text | findByNameContaining(String name) |
| Filter by prefix | findByNameStartingWith(String prefix) |
| Greater than | findByPriceGreaterThan(Double price) |
| Less than | findByPriceLessThan(Double price) |
| Between dates | findByCreatedAtBetween(Date start, Date end) |
| Top N records | findTop3ByOrderByPriceAsc() |
| Sorting | findAllByOrderByNameAsc() |

This implementation **covers all the required query method types** as asked in the assignment. Let me know your next question.

Here is a **complete and detailed implementation** of **Object-Relational Mapping (O/R Mapping)** using **Spring Data JPA** covering:

* @ManyToOne, @JoinColumn
* @OneToMany, mappedBy
* FetchType.EAGER, FetchType.LAZY
* @ManyToMany, @JoinTable

You can directly **copy-paste** this code into your Cognizant assignment project.

## **Scenario: A Student can enroll in multiple Courses, and each Course can have multiple Students.**

Additionally, each Course belongs to a single Department.

## **Step 1: Create Department Entity – (Many Courses to One Department)**

import jakarta.persistence.\*;

import java.util.List;

@Entity

public class Department {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

// One Department has many Courses (mappedBy used here)

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

private List<Course> courses;

public Department() {}

public Department(String name) {

this.name = name;

}

// Getters and Setters

}

## **Step 2: Create Course Entity – (Many Courses belong to One Department, Many-to-Many with Students)**

import jakarta.persistence.\*;

import java.util.List;

@Entity

public class Course {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String title;

// Many courses belong to one department

@ManyToOne(fetch = FetchType.EAGER)

@JoinColumn(name = "department\_id")

private Department department;

// Many students can enroll in many courses

@ManyToMany(mappedBy = "courses", fetch = FetchType.LAZY)

private List<Student> students;

public Course() {}

public Course(String title, Department department) {

this.title = title;

this.department = department;

}

// Getters and Setters

}

## **Step 3: Create Student Entity – (Many-to-Many with Courses)**

import jakarta.persistence.\*;

import java.util.List;

@Entity

public class Student {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

// Many students can enroll in many courses

@ManyToMany(fetch = FetchType.LAZY)

@JoinTable(

name = "student\_course",

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

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

)

private List<Course> courses;

public Student() {}

public Student(String name) {

this.name = name;

}

// Getters and Setters

}

## **Step 4: Repositories**

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

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

public interface CourseRepository extends JpaRepository<Course, Long> {}

public interface StudentRepository extends JpaRepository<Student, Long> {}

## **Step 5: Add Sample Data in Main Class**

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

import org.springframework.boot.CommandLineRunner;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import java.util.Arrays;

@SpringBootApplication

public class OrmMappingDemoApplication implements CommandLineRunner {

@Autowired

private DepartmentRepository departmentRepo;

@Autowired

private CourseRepository courseRepo;

@Autowired

private StudentRepository studentRepo;

public static void main(String[] args) {

SpringApplication.run(OrmMappingDemoApplication.class, args);

}

@Override

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

// Create department

Department dept = new Department("Computer Science");

departmentRepo.save(dept);

// Create courses

Course c1 = new Course("Data Structures", dept);

Course c2 = new Course("Algorithms", dept);

courseRepo.saveAll(Arrays.asList(c1, c2));

// Create students and enroll them in courses

Student s1 = new Student("Aakanksha");

Student s2 = new Student("Rahul");

s1.setCourses(Arrays.asList(c1, c2));

s2.setCourses(Arrays.asList(c2));

studentRepo.saveAll(Arrays.asList(s1, s2));

}

}

## **Summary of Annotations Used:**

| **Annotation** | **Purpose** |
| --- | --- |
| @ManyToOne | Many courses can belong to one department |
| @JoinColumn | Specifies the foreign key column name (department\_id) |
| @OneToMany | One department has many courses (mappedBy = "department") |
| FetchType.LAZY | Data is fetched lazily (when accessed) |
| FetchType.EAGER | Data is fetched immediately with the entity |
| @ManyToMany | Many students can enroll in many courses |
| @JoinTable | Creates a join table student\_course with student and course IDs |
| mappedBy | Used in bidirectional mapping to indicate owning side (in Course) |

This implementation **completely demonstrates O/R Mapping** using **all required annotations** as per the assignment. Let me know the next question.