**Spring Data JPA HandsOn-2  
1.** **Demonstrate implementation of Query Methods feature of Spring Data JPA**

**What Are Query Methods?**

Spring Data JPA can automatically generate SQL just from method names in the repository interface. You don’t need to write SQL or JPQL — just follow naming conventions.  
**Example Use Case**

We’ll create a table called Product with fields:

Id, name, category, price

Then we’ll define **query methods** like:

findByName(), findByCategory(), findByPriceGreaterThan(), findByNameAndCategory()

**Entity Class**

import jakarta.persistence.\*;   
@Entity   
public class Product {   
@Id   
@GeneratedValue(strategy = GenerationType.IDENTITY)  
 private Long id;  
 private String name;  
 private String category;  
 private double price;  }

**Repository Interface with Query Methods**

import org.springframework.data.jpa.repository.JpaRepository;  
 import java.util.List;   
public interface ProductRepository extends JpaRepository<Product, Long> {   
List<Product> findByName(String name);  
 List<Product> findByCategory(String category);   
List<Product> findByPriceGreaterThan(double price);  
 List<Product> findByNameAndCategory(String name, String category);  
 }

**Application Class to Test**

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 ProductApp implements CommandLineRunner {

@Autowired

private ProductRepository repo;

public static void main(String[] args) {

SpringApplication.run(ProductApp.class, args);

}

@Override

public void run(String... args) {

// Inserting sample data

repo.save(new Product(null, "iPhone", "Electronics", 75000));

repo.save(new Product(null, "TV", "Electronics", 40000));

repo.save(new Product(null, "T-shirt", "Clothing", 800));

repo.save(new Product(null, "Shoes", "Clothing", 1500));

// Query Method Examples

System.out.println("Find by name:");

repo.findByName("TV").forEach(p -> System.out.println(p.getName()));

System.out.println("Find by category:");

repo.findByCategory("Clothing").forEach(p -> System.out.println(p.getName()));

System.out.println("Find products with price > 1000:");

repo.findByPriceGreaterThan(1000).forEach(p -> System.out.println(p.getName()));

System.out.println("Find by name and category:");

repo.findByNameAndCategory("TV", "Electronics").forEach(p -> System.out.println(p.getName()));

}

}

**application.properties**

spring.datasource.url=jdbc:mysql://localhost:3306/testdb   
spring.datasource.username=root   
spring.datasource.password=1234   
spring.jpa.show-sql=true   
spring.jpa.hibernate.ddl-auto=update

**2. Demonstrate implementation of O/R Mapping**

**@ManyToOne, @JoinColumn, @OneToMany, FetchType.EAGER, FetchType.LAZY, @ManyToMany, @JoinTable, mappedBy  
  
Step 1: Entities**

**Department Entity – @OneToMany (EAGER)**import jakarta.persistence.\*;

import java.util.List;

@Entity

public class Department {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;  
 @OneToMany(mappedBy = "department", fetch = FetchType.EAGER, cascade = CascadeType.ALL  
 private List<Employee> employees;  
}  
  
**Employee Entity – @ManyToOne + @ManyToMany**import jakarta.persistence.\*;

import java.util.List;

@Entity

public class Employee {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

@ManyToOne(fetch = FetchType.LAZY)

@JoinColumn(name = "dept\_id") // Foreign key

private Department department;

@ManyToMany

@JoinTable(

name = "employee\_project",

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

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

)

private List<Project> projects;

}  
  
**Project Entity – @ManyToMany (mappedBy)**

import jakarta.persistence.\*;

import java.util.List;

@Entity

public class Project {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String projectName;

@ManyToMany(mappedBy = "projects")

private List<Employee> employees;

}  
**Step 2: Repositories**

import org.springframework.data.jpa.repository.JpaRepository;  
public interface DepartmentRepository extends JpaRepository<Department, Long> {}  
public interface EmployeeRepository extends JpaRepository<Employee, Long> {}  
public interfaceProjectRepository extends JpaRepository<Project, Long> {}  
**Step 3: Testing the Mapping**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.List;

@SpringBootApplication

public class OrmMappingApp implements CommandLineRunner {

@Autowired

DepartmentRepository deptRepo;

@Autowired

EmployeeRepository empRepo;

@Autowired

ProjectRepository projRepo;

public static void main(String[] args) {

SpringApplication.run(OrmMappingApp.class, args);

}

@Override

public void run(String... args) {

// Create projects

Project p1 = new Project();

p1.setProjectName("HR Portal");

Project p2 = new Project();

p2.setProjectName("ERP System");  
 projRepo.saveAll(List.of(p1, p2));

// Create department

Department d1 = new Department();

d1.setName("IT");

// Create employees

Employee e1 = new Employee();

e1.setName("Alice");

e1.setDepartment(d1);

e1.setProjects(List.of(p1, p2));  
 Employee e2 = new Employee();

e2.setName("Bob");

e2.setDepartment(d1);

e2.setProjects(List.of(p1));  
 d1.setEmployees(List.of(e1, e2));

deptRepo.save(d1);

}

}