**Demonstrate implementation of O/R Mapping**

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

Vaishnavi

6/7/2025

Object-Relational Mapping (O/R Mapping or ORM) is the mechanism by which Java objects are persisted to relational database tables. JPA (Java Persistence API), with Spring Data JPA, provides annotations to define how Java classes relate to database tables and how their properties relate to columns and relationships.

In this demonstration, we’ll map real-world entities such as **Employee**, **Department**, and **Skill** using:

One-to-Many (@OneToMany)

Many-to-One (@ManyToOne)

Many-to-Many (@ManyToMany)

Fetch strategies (FetchType.LAZY and FetchType.EAGER)

Join specifications (@JoinColumn, @JoinTable, and mappedBy)

**Objective:**

To implement O/R Mapping between entities using JPA annotations.

Demonstrate unidirectional and bidirectional relationships.

Explore fetch types and understand their role in lazy/eager data loading.

Configure joining strategies using @JoinColumn and @JoinTable.

**Implementation:**

**Entity Class:**

**Department.java**

@Entity

@Table(name = "department")

public class Department {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private int id;

@Column(name = "dp\_name")

private String name;

// One-to-Many with Employee

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

private Set<Employee> employeeList;

// Getters, Setters, toString

}

**Employee.java**

@Entity

@Table(name = "employee")

public class Employee {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private int id;

@Column(name = "em\_name")

private String name;

@Column(name = "em\_salary")

private double salary;

@Column(name = "em\_permanent")

private boolean permanent;

@Column(name = "em\_date\_of\_birth")

private Date dateOfBirth;

// Many-to-One with Department

@ManyToOne

@JoinColumn(name = "em\_dp\_id")

private Department department;

// Many-to-Many with Skill

@ManyToMany(fetch = FetchType.EAGER)

@JoinTable(name = "employee\_skill",

joinColumns = @JoinColumn(name = "es\_em\_id"),

inverseJoinColumns = @JoinColumn(name = "es\_sk\_id"))

private Set<Skill> skillList;

// Getters, Setters, toString

}

**Skill.java**

@Entity

@Table(name = "skill")

public class Skill {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private int id;

@Column(name = "sk\_name")

private String name;

// Many-to-Many mappedBy

@ManyToMany(mappedBy = "skillList")

private Set<Employee> employeeList;

// Getters, Setters, toString

}

**Add Employee with Department**

private static void testAddEmployee() {

Employee emp = new Employee();

emp.setName("Vaishnavi");

emp.setSalary(65000);

emp.setPermanent(true);

emp.setDateOfBirth(new Date());

Department dept = departmentService.get(1);

emp.setDepartment(dept);

employeeService.save(emp);

LOGGER.debug("Saved employee: {}", emp);

}

**Fetch Employee with Department and Skills**

private static void testGetEmployee() {

Employee emp = employeeService.get(1);

LOGGER.debug("Employee: {}", emp);

LOGGER.debug("Department: {}", emp.getDepartment());

LOGGER.debug("Skills: {}", emp.getSkillList());

}

**Add Skill to Employee**

private static void testAddSkillToEmployee() {

Employee emp = employeeService.get(1);

Skill skill = skillService.get(2); // e.g., Java

emp.getSkillList().add(skill);

employeeService.save(emp);

LOGGER.debug("Updated employee skills: {}", emp.getSkillList());

}

**Fetch Department with Employees**

private static void testGetDepartment() {

Department dept = departmentService.get(1);

LOGGER.debug("Department: {}", dept);

LOGGER.debug("Employees: {}", dept.getEmployeeList());

}