**WEEK-3**

**SPRING DATA JPA AND HIBERNATE HANDSON**

**Objective 1: Demonstrate Implementation of Query Methods in Spring Data JPA**

**Spring Data JPA Query Methods Using Naming Conventions**

Spring Data JPA enables us to define query methods directly in the repository interface by simply following a specific naming pattern. At runtime, Spring automatically provides the implementation, eliminating the need to manually write SQL or JPQL queries.

**Typical Use Cases for Query Methods:**

1. **Search Using Partial Text Match**

List<Employee> findByNameContaining(String keyword);

1. **Filter Records by Starting Characters**

List<Employee> findByDepartmentStartingWith(String prefix);

1. **Sorting Query Results**

List<Employee> findByDepartmentOrderByNameAsc(String department);

1. **Retrieve Data Between Dates**

List<Employee> findByJoiningDateBetween(LocalDate start, LocalDate end);

1. **Numeric Comparisons (Greater/Less Than)**

List<Employee> findBySalaryGreaterThan(Double minSalary);

List<Employee> findBySalaryLessThanEqual(Double maxSalary);

1. **Fetch Top or First Records**

List<Employee> findTop3ByOrderBySalaryDesc();

Obtains the top three employees with the highest salary—commonly used for rankings or analytical summaries.

These query method patterns can be tailored and combined flexibly, offering a concise and readable way to access data without redundant code.

**Objective 2: Demonstrate Implementation of Object-Relational Mapping (O/R Mapping)**

Spring Data JPA uses annotations to define how Java objects map to relational database tables. These relationships are especially useful in multi-entity models like Employee, Department, and Project.

**Common Relationship Annotations:**

1. **@ManyToOne**

@ManyToOne(fetch = FetchType.LAZY)

@JoinColumn(name = "department\_id")

private Department department;

Many employees can belong to one department.

@JoinColumn specifies the foreign key column.

FetchType.LAZY means the department will be loaded only when accessed.

1. **@OneToMany**

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

private List<Employee> employees;

A department has many employees.

mappedBy refers to the field in the Employee entity that owns the relationship.

1. **@ManyToMany**

@ManyToMany

@JoinTable(

name = "employee\_project",

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

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

)

private Set<Project> projects;

An employee can be assigned to multiple projects, and a project can have multiple employees.

@JoinTable defines the linking (junction) table and its foreign keys.

1. **EAGER vs LAZY Fetching**

FetchType.LAZY: Data is loaded only when explicitly accessed. Better performance.

FetchType.EAGER: Data is loaded immediately with the entity. May cause overhead if not used carefully.