**WEEK-3**

**SPRING DATA JPA AND HIBERNATE HANDSON**

**Superset ID: 6419740**

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

Spring Data JPA allows us to define query methods directly in the repository interface by just following a naming convention. These methods are automatically implemented by Spring at runtime no need to write SQL or JPQL manually.

**Common Query Method Use Cases:**

1. **Search by Containing Text**

List<Employee> findByNameContaining(String keyword);

Fetches employees whose name contains the given keyword (case sensitive by default).

1. **Filter with Starting Text**

List<Employee> findByDepartmentStartingWith(String prefix);

Useful when searching names or departments that begin with a specific string.

1. **Sorting Results**

List<Employee> findByDepartmentOrderByNameAsc(String department);

Sorts employees by name within the given department.

1. **Fetch Between Dates**

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

Useful to fetch records within a specific range of time, e.g., monthly reports.

1. **Greater Than / Less Than**

List<Employee> findBySalaryGreaterThan(Double minSalary);

List<Employee> findBySalaryLessThanEqual(Double maxSalary);

Filters based on numeric comparisons.

1. **Top / First Records**

List<Employee> findTop3ByOrderBySalaryDesc();

Fetches top N records with the highest salary, useful for leaderboards or insights.

These methods can be combined and customized as needed, offering a clean way to query without writing boilerplate 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.