### 1. Defining Query Methods

#### 1.1 Using Keywords in Method Names

Spring Data JPA allows you to define query methods by using specific keywords in method names. These methods are automatically implemented by Spring Data JPA.

**Example for EmployeeRepository:**

package com.example.employeemanagementsystem.repository;

import com.example.employeemanagementsystem.model.Employee;

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

import org.springframework.stereotype.Repository;

import java.util.List;

@Repository

public interface EmployeeRepository extends JpaRepository<Employee, Long> {

// Find employees by their name (case insensitive)

List<Employee> findByNameIgnoreCase(String name);

// Find employees by email (assumed unique)

Employee findByEmail(String email);

// Find employees by department's ID

List<Employee> findByDepartmentId(Long departmentId);

// Find employees whose names contain a given string

List<Employee> findByNameContaining(String namePart);

}

* findByNameIgnoreCase(String name): Finds employees by name in a case-insensitive manner.
* findByEmail(String email): Finds a single employee by email.
* findByDepartmentId(Long departmentId): Finds all employees in a specific department.
* findByNameContaining(String namePart): Finds employees whose names contain a specified substring.

#### 1.2 Implementing Custom Query Methods with @Query

For more complex queries, use the @Query annotation to define JPQL queries directly.

**Example for EmployeeRepository with** @Query**:**

package com.example.employeemanagementsystem.repository;

import com.example.employeemanagementsystem.model.Employee;

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

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

import org.springframework.data.repository.query.Param;

import org.springframework.stereotype.Repository;

import java.util.List;

@Repository

public interface EmployeeRepository extends JpaRepository<Employee, Long> {

// Custom query using @Query annotation

@Query("SELECT e FROM Employee e WHERE e.name = :name")

List<Employee> findEmployeesByName(@Param("name") String name);

@Query("SELECT e FROM Employee e WHERE e.email LIKE %:emailPart%")

List<Employee> findEmployeesByEmailPart(@Param("emailPart") String emailPart);

// Join query to find employees by department name

@Query("SELECT e FROM Employee e JOIN e.department d WHERE d.name = :departmentName")

List<Employee> findEmployeesByDepartmentName(@Param("departmentName") String departmentName);

}

* findEmployeesByName(@Param("name") String name): Finds employees by their exact name using a JPQL query.
* findEmployeesByEmailPart(@Param("emailPart") String emailPart): Finds employees where their email contains a specific part.
* findEmployeesByDepartmentName(@Param("departmentName") String departmentName): Finds employees based on the department's name using a join query.

### 2. Named Queries

#### 2.1 Defining Named Queries

Named queries are pre-defined queries stored in the entity class using the @NamedQuery or @NamedQueries annotations. They provide a way to define queries once and reuse them.

**Example of Named Queries in** Employee **Entity:**

package com.example.employeemanagementsystem.model;

import lombok.Data;

import lombok.NoArgsConstructor;

import javax.persistence.\*;

import java.util.Set;

@Entity

@Table(name = "employees")

@NamedQueries({

@NamedQuery(name = "Employee.findByName",

query = "SELECT e FROM Employee e WHERE e.name = :name"),

@NamedQuery(name = "Employee.findByEmailPart",

query = "SELECT e FROM Employee e WHERE e.email LIKE %:emailPart%"),

@NamedQuery(name = "Employee.findByDepartmentName",

query = "SELECT e FROM Employee e JOIN e.department d WHERE d.name = :departmentName")

})

@Data

@NoArgsConstructor

public class Employee {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

private String email;

@ManyToOne

@JoinColumn(name = "department\_id", nullable = false)

private Department department;

// Add constructors, getters, and setters if not using Lombok

}

* @NamedQuery: Defines a query that can be referenced by name. For instance, "Employee.findByName" can be used to find employees by name.

#### 2.2 Using Named Queries

To use named queries, you need to call them from the repository.

**Example for EmployeeRepository Using Named Queries:**

package com.example.employeemanagementsystem.repository;

import com.example.employeemanagementsystem.model.Employee;

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

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

import org.springframework.stereotype.Repository;

import java.util.List;

@Repository

public interface EmployeeRepository extends JpaRepository<Employee, Long> {

// Use the named queries defined in the Employee entity

@Query(name = "Employee.findByName")

List<Employee> findByName(String name);

@Query(name = "Employee.findByEmailPart")

List<Employee> findByEmailPart(String emailPart);

@Query(name = "Employee.findByDepartmentName")

List<Employee> findByDepartmentName(String departmentName);

}