**Exercise 5: Employee Management System - Defining Query Methods EmployeeRepository**

package com.example.employeemanagementsystem.repository;

import com.example.employeemanagementsystem.entity.Employee; import org.springframework.data.jpa.repository.JpaRepository;

import java.util.List;

public interface EmployeeRepository extends JpaRepository<Employee, Long> {

// Find employees by their name

List<Employee> findByName(String name);

// Find employees by their email

Employee findByEmail(String email);

// Find employees by their department's name

List<Employee> findByDepartmentName(String departmentName);

// Find employees by their department ID and name

List<Employee> findByDepartmentIdAndName(Long departmentId, String name);

// Find employees whose name contains a specific string List<Employee> findByNameContaining(String keyword);

}

DepartmentRepository

package com.example.employeemanagementsystem.repository;

import com.example.employeemanagementsystem.entity.Department;

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

public interface DepartmentRepository extends JpaRepository<Department, Long> {

// Find a department by its name

Department findByName(String name);

// Find departments whose name starts with a specific prefix List<Department> findByNameStartingWith(String prefix);

// Find departments whose name ends with a specific suffix List<Department> findByNameEndingWith(String suffix);

// Find departments whose name contains a specific string List<Department> findByNameContaining(String keyword);

}

Implementing Custom Query Methods Using the @Query Annotation EmployeeRepository

package com.example.employeemanagementsystem.repository;

import com.example.employeemanagementsystem.entity.Employee;

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

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

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

import java.util.List;

public interface EmployeeRepository extends JpaRepository<Employee, Long> {

// Custom query to find employees by department's name

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

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

// Custom query to find employees by partial email match

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

List<Employee> findEmployeesByEmailFragment(@Param("emailFragment") String emailFragment);

}

DepartmentRepository

package com.example.employeemanagementsystem.repository;

import com.example.employeemanagementsystem.entity.Department;

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

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

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

import java.util.List;

public interface DepartmentRepository extends JpaRepository<Department, Long> {

// Custom query to find departments with a specific number of employees

@Query("SELECT d FROM Department d WHERE SIZE(d.employees) = :employeeCount")

List<Department> findDepartmentsByEmployeeCount(@Param("employeeCount") int employeeCount);

}

**Named Queries**

Named queries are pre-defined, reusable queries that can be declared at the entity level. They are typically used for more complex or frequently used queries.

Employee Entity with Named Query

package com.example.employeemanagementsystem.entity;

import jakarta.persistence.Entity;

import jakarta.persistence.GeneratedValue;

import jakarta.persistence.GenerationType;

import jakarta.persistence.Id;

import jakarta.persistence.ManyToOne;

import jakarta.persistence.NamedQueries;

import jakarta.persistence.NamedQuery;

import jakarta.persistence.Table;

import lombok.AllArgsConstructor;

import lombok.Data;

import lombok.NoArgsConstructor;

@Entity

@Table(name = "employees") @Data

@NoArgsConstructor @AllArgsConstructor @NamedQueries({

@NamedQuery(name = "Employee.findByDepartmentName",

query = "SELECT e FROM Employee e WHERE e.department.name = :departmentName"), @NamedQuery(name = "Employee.findByEmailFragment",

query = "SELECT e FROM Employee e WHERE e.email LIKE :emailFragment")

})

public class Employee {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY) private Long id;

private String name; private String email;

@ManyToOne

private Department department;

}

Use Named Queries in Repository

package com.example.employeemanagementsystem.repository;

import com.example.employeemanagementsystem.entity.Employee;

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

import org.springframework.data.repository.query.Param; import java.util.List;

public interface EmployeeRepository extends JpaRepository<Employee, Long> {

// Named query method to find employees by department name

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

// Named query method to find employees by partial email match

List<Employee> findByEmailFragment(@Param("emailFragment") String emailFragment);

}

**Explanation of Named Queries**

* **@NamedQuery**: Defines a single named query that can be reused in the repository. The name attribute is the identifier, and the query attribute is the JPQL query.
* **@NamedQueries**: Container for multiple @NamedQuery annotations.
* **Usage**: You can call named queries directly from your repository interface using their names.

**Final Steps**

1. **Test Custom Queries**: Use integration tests or tools like Postman to verify that your custom queries work as expected.
2. **Optimize Queries**: Review and optimize queries for performance, particularly if you're working with large datasets.
3. **Expand Query Methods**: Add more query methods to handle specific business logic or reporting needs as your application grows.

With these custom query methods and named queries in place, your Employee Management System is now more flexible and powerful, allowing you to handle complex data retrieval scenarios.