**WEEK 3**

**EXERCISE 8**

**Employee Management System - Creating Projections**

**1. Projections:**

**a. Defining Interface-Based Projections**

* **Purpose:** Create lightweight views of your entities by defining interface-based projections that allow you to fetch only specific fields rather than entire entities.

**Example: Interface-Based Projection for Employee Entity**

package com.example.employeemanagementsystem.projection;

public interface EmployeeBasicInfo {

// Fetch the employee's name and email

String getName();

String getEmail();

}

**Example: Using Interface-Based Projection in Repository**

package com.example.employeemanagementsystem.repository;

import com.example.employeemanagementsystem.entity.Employee;

import com.example.employeemanagementsystem.projection.EmployeeBasicInfo;

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

import org.springframework.stereotype.Repository;

import java.util.List;

@Repository

public interface EmployeeRepository extends JpaRepository<Employee, Long> {

// Fetch basic info of all employees

List<EmployeeBasicInfo> findAllProjectedBy();

}

**b. Defining Class-Based Projections**

* **Purpose:** Create custom objects that are populated directly from the database query using constructor expressions.

**Example: Class-Based Projection for Employee Entity**

package com.example.employeemanagementsystem.projection;

public class EmployeeNameAndDepartment {

private final String name;

private final String departmentName;

public EmployeeNameAndDepartment(String name, String departmentName) {

this.name = name;

this.departmentName = departmentName;

}

public String getName() {

return name;

}

public String getDepartmentName() {

return departmentName;

}

}

**Example: Using Class-Based Projection in Repository**

package com.example.employeemanagementsystem.repository;

import com.example.employeemanagementsystem.entity.Employee;

import com.example.employeemanagementsystem.projection.EmployeeNameAndDepartment;

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> {

// Fetch employee name and department name using class-based projection

@Query("SELECT new com.example.employeemanagementsystem.projection.EmployeeNameAndDepartment(e.name, d.name) " +

"FROM Employee e JOIN e.department d")

List<EmployeeNameAndDepartment> findEmployeeNamesWithDepartment();

}

**2. Using @Value and Constructor Expressions**

**a. Using @Value Annotation in Interface-Based Projections**

* **Purpose:** Use @Value to customize the fields returned by interface-based projections, allowing you to concatenate or transform values.

**Example: Customized Interface-Based Projection**

package com.example.employeemanagementsystem.projection;

import org.springframework.beans.factory.annotation.Value;

public interface CustomizedEmployeeInfo {

// Concatenate name and email in a single string

@Value("#{target.name + ' (' + target.email + ')'}")

String getFullNameWithEmail();

}

**b. Constructor Expressions in Queries**

* **Purpose:** Use constructor expressions in JPQL queries to directly create custom objects from query results.

**Example: Query with Constructor Expression**

package com.example.employeemanagementsystem.repository;

import com.example.employeemanagementsystem.projection.EmployeeNameAndDepartment;

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> {

// Query using constructor expression

@Query("SELECT new com.example.employeemanagementsystem.projection.EmployeeNameAndDepartment(e.name, d.name) " +

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

List<EmployeeNameAndDepartment> findEmployeesByDepartment(String departmentName);

}