**Exercise 8: Employee Management System – Creating Projections**

In the Employee Management System, projections allow you to fetch specific subsets of data from your entities rather than retrieving entire entities. This is especially useful when you want to optimize performance by limiting the amount of data retrieved from the database.

1. **Interface-Based Projections**

Interface-based projections are a simple way to define a subset of fields you want to retrieve from an entity.

* 1. **Define Interface-Based Projection for Employee**

Create an interface that defines the fields you want to project.

public interface EmployeeNameAndDepartment {

String getName();

String getDepartment();

}

* 1. **Define Interface-Based Projection for Department** Similarly, define a projection for the Department entity: java

Copy code

public interface DepartmentNameAndId {

Long getId();

String getName();

}

* 1. **Use Projections in Repository**

In your repository interface, add methods that return these projections: java

Copy code

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> {

// Projection with a derived query method

List<EmployeeNameAndDepartment> findAllByDepartment(String department);

// Projection using a custom query

@Query("SELECT e.name AS name, e.department AS department FROM Employee e") List<EmployeeNameAndDepartment> findEmployeeNamesAndDepartments();

}

@Repository

public interface DepartmentRepository extends JpaRepository<Department, Long> {

List<DepartmentNameAndId> findBy();

}

1. **Class-Based Projections**

Class-based projections use a DTO (Data Transfer Object) class to represent the subset of data. You can use constructor expressions to achieve this.

* 1. **Define DTO for Employee**

Create a DTO class that will hold the projected data. java

Copy code

public class EmployeeDTO {

private String name;

private String department;

// Constructor for the projection

public EmployeeDTO(String name, String department) {

this.name = name;

this.department = department;

}

// Getters and Setters

}

* 1. **Use DTO in Repository**

In the repository interface, create a query that returns the DTO:

java

Copy code

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("SELECT new com.example.demo.EmployeeDTO(e.name, e.department) FROM Employee e")

List<EmployeeDTO> findEmployeeDTOs();

}

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

Sometimes, you might want to create more complex projections or derive values on the fly. You can do this using the @Value annotation and SpEL (Spring Expression Language).

* 1. **Create a Projection with Derived Fields**

Suppose you want to include a full name (firstName + lastName) in your projection. You can achieve this using @Value.

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

public interface EmployeeFullNameProjection {

@Value("#{target.firstName + ' ' + target.lastName}") String getFullName();

String getDepartment();

}

* 1. **Use the Projection in Repository**

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

import org.springframework.stereotype.Repository;

import java.util.List; @Repository

public interface EmployeeRepository extends JpaRepository<Employee, Long> {

List<EmployeeFullNameProjection> findByDepartment(String department);

}

1. **Fetching and Using Projections**

You can fetch and use projections in your service or controller layer like this:

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

import org.springframework.stereotype.Service;

import java.util.List; @Service

public class EmployeeService {

@Autowired

private EmployeeRepository employeeRepository;

public List<EmployeeNameAndDepartment> getEmployeeNameAndDepartment(String department) {

return employeeRepository.findAllByDepartment(department);

}

public List<EmployeeDTO> getEmployeeDTOs() { return employeeRepository.findEmployeeDTOs();

}

public List<EmployeeFullNameProjection> getEmployeeFullNames(String department) { return employeeRepository.findByDepartment(department);

}

}