**Exercise 1: Employee Management System - Overview and Setup**

**Business Scenario:**

You are developing an employee management system that will manage employee data, departments, and their relationships.

**Instructions:**

1. **Creating a Spring Boot Project:**
   * Initialize a new Spring Boot project named **EmployeeManagementSystem**.
   * Add dependencies: **Spring Data JPA, H2 Database, Spring Web, Lombok**.
2. **Configuring Application Properties:**
   * Configure **application.properties** for H2 database connection.

*spring.datasource.url=jdbc:h2:mem:testdb*

*spring.datasource.driverClassName=org.h2.Driver*

*spring.datasource.username=sa*

*spring.datasource.password=password*

*spring.jpa.database-platform=org.hibernate.dialect.H2Dialect*

ANSWER:

**Application.properties**

spring.datasource.url=jdbc:h2:mem:testdb

spring.datasource.driverClassName=org.h2.Driver

spring.datasource.username=sa

spring.datasource.password=password

spring.jpa.database-platform=org.hibernate.dialect.H2Dialect

**Exercise 2: Employee Management System - Creating Entities**

**Business Scenario:**

Define JPA entities for Employee and Department with appropriate relationships.

**Instructions:**

1. **Creating JPA Entities:**
   * Define **Employee** entity with fields: **id, name, email, department**.
   * Define **Department** entity with fields: **id, name**.

1. **Mapping Entities to Database Tables:**
   * Use annotations like **@Entity, @Table, @Id, @GeneratedValue**, etc.
   * Define one-to-many relationship between **Department** and **Employee**.

ANSWER:

Employees.java:

import jakarta.persistence.\*;

import lombok.\*;

@Entity

@Table(name = "employees")

@Data

@NoArgsConstructor

@AllArgsConstructor

public class Employee {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@Column(nullable = false)

private String name;

@Column(nullable = false, unique = true)

private String email;

@ManyToOne

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

private Department department;

}

Department.java:

import jakarta.persistence.\*;

import lombok.\*;

import java.util.List;

@Entity

@Table(name = "departments")

@Data

@NoArgsConstructor

@AllArgsConstructor

public class Department {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@Column(nullable = false)

private String name;

@OneToMany(mappedBy = "department", cascade = CascadeType.ALL, orphanRemoval = true)

private List<Employee> employees;

}

**Exercise 3: Employee Management System - Creating Repositories**

**Business Scenario:**

Create repositories for Employee and Department entities to perform CRUD operations.

**Instructions:**

1. **Overview of Spring Data Repositories:**
   * Learn the benefits of using Spring Data repositories.
2. **Creating Repositories:**
   * Create **EmployeeRepository** and **DepartmentRepository** interfaces extending **JpaRepository**.
   * Define derived query methods in these repositories.

ANSWER:

**Create EmployeeRepository Interface**

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

import org.springframework.stereotype.Repository;

@Repository

public interface EmployeeRepository extends JpaRepository<Employee, Long> {

// Derived query method to find employees by department

List<Employee> findByDepartmentName(String departmentName);

// Derived query method to find an employee by email

Employee findByEmail(String email);

}

**Create DepartmentRepository Interface**

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

import org.springframework.stereotype.Repository;

@Repository

public interface DepartmentRepository extends JpaRepository<Department, Long> {

// Derived query method to find a department by name

Department findByName(String name);

}

**Exercise 4: Employee Management System - Implementing CRUD Operations**

**Business Scenario:**

Implement CRUD operations for managing employees and departments.

**Instructions:**

1. **Basic CRUD Operations:**
   * Use **JpaRepository** methods to create, read, update, and delete employees and departments.
   * Implement RESTful endpoints for these operations using **EmployeeController** and **DepartmentController**.

ANSWER:  
**EmployeeService Class**

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

import org.springframework.stereotype.Service;

import java.util.List;

import java.util.Optional;

@Service

public class EmployeeService {

@Autowired

private EmployeeRepository employeeRepository;

// Create a new employee

public Employee createEmployee(Employee employee) {

return employeeRepository.save(employee);

}

// Get all employees

public List<Employee> getAllEmployees() {

return employeeRepository.findAll();

}

// Get an employee by ID

public Optional<Employee> getEmployeeById(Long id) {

return employeeRepository.findById(id);

}

// Update an existing employee

public Employee updateEmployee(Long id, Employee updatedEmployee) {

return employeeRepository.findById(id)

.map(employee -> {

employee.setName(updatedEmployee.getName());

employee.setEmail(updatedEmployee.getEmail());

employee.setDepartment(updatedEmployee.getDepartment());

return employeeRepository.save(employee);

})

.orElseThrow(() -> new ResourceNotFoundException("Employee not found with id " + id));

}

// Delete an employee

public void deleteEmployee(Long id) {

employeeRepository.deleteById(id);

}

}

**DepartmentService Class**

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

import org.springframework.stereotype.Service;

import java.util.List;

import java.util.Optional;

@Service

public class DepartmentService {

@Autowired

private DepartmentRepository departmentRepository;

// Create a new department

public Department createDepartment(Department department) {

return departmentRepository.save(department);

}

// Get all departments

public List<Department> getAllDepartments() {

return departmentRepository.findAll();

}

// Get a department by ID

public Optional<Department> getDepartmentById(Long id) {

return departmentRepository.findById(id);

}

// Update an existing department

public Department updateDepartment(Long id, Department updatedDepartment) {

return departmentRepository.findById(id)

.map(department -> {

department.setName(updatedDepartment.getName());

return departmentRepository.save(department);

})

.orElseThrow(() -> new ResourceNotFoundException("Department not found with id " + id));

}

// Delete a department

public void deleteDepartment(Long id) {

departmentRepository.deleteById(id);

}

}

**EmployeeController Class**

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

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.\*;

import java.util.List;

@RestController

@RequestMapping("/api/employees")

public class EmployeeController {

@Autowired

private EmployeeService employeeService;

// Create a new employee

@PostMapping

public Employee createEmployee(@RequestBody Employee employee) {

return employeeService.createEmployee(employee);

}

// Get all employees

@GetMapping

public List<Employee> getAllEmployees() {

return employeeService.getAllEmployees();

}

// Get an employee by ID

@GetMapping("/{id}")

public ResponseEntity<Employee> getEmployeeById(@PathVariable Long id) {

return employeeService.getEmployeeById(id)

.map(ResponseEntity::ok)

.orElse(ResponseEntity.notFound().build());

}

// Update an employee

@PutMapping("/{id}")

public ResponseEntity<Employee> updateEmployee(@PathVariable Long id, @RequestBody Employee updatedEmployee) {

return ResponseEntity.ok(employeeService.updateEmployee(id, updatedEmployee));

}

// Delete an employee

@DeleteMapping("/{id}")

public ResponseEntity<Void> deleteEmployee(@PathVariable Long id) {

employeeService.deleteEmployee(id);

return ResponseEntity.noContent().build();

}

}

**DepartmentController Class**

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

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.\*;

import java.util.List;

@RestController

@RequestMapping("/api/departments")

public class DepartmentController {

@Autowired

private DepartmentService departmentService;

// Create a new department

@PostMapping

public Department createDepartment(@RequestBody Department department) {

return departmentService.createDepartment(department);

}

// Get all departments

@GetMapping

public List<Department> getAllDepartments() {

return departmentService.getAllDepartments();

}

// Get a department by ID

@GetMapping("/{id}")

public ResponseEntity<Department> getDepartmentById(@PathVariable Long id) {

return departmentService.getDepartmentById(id)

.map(ResponseEntity::ok)

.orElse(ResponseEntity.notFound().build());

}

// Update a department

@PutMapping("/{id}")

public ResponseEntity<Department> updateDepartment(@PathVariable Long id, @RequestBody Department updatedDepartment) {

return ResponseEntity.ok(departmentService.updateDepartment(id, updatedDepartment));

}

// Delete a department

@DeleteMapping("/{id}")

public ResponseEntity<Void> deleteDepartment(@PathVariable Long id) {

departmentService.deleteDepartment(id);

return ResponseEntity.noContent().build();

}

}

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

**Business Scenario:**

Enhance your repository to support custom queries.

**Instructions:**

1. **Defining Query Methods:**
   * Use keywords in method names to create custom query methods.
   * Implement custom query methods using the **@Query** annotation.
2. **Named Queries:**
   * Define and execute named queries with **@NamedQuery** and **@NamedQueries**.

ANSWER:  
**EmployeeRepository**

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 name

List<Employee> findByName(String name);

// Find employees by department name

List<Employee> findByDepartmentName(String departmentName);

// Find employees whose names start with a specific prefix

List<Employee> findByNameStartingWith(String prefix);

// Find employees by email domain

List<Employee> findByEmailEndingWith(String domain);

}

**DepartmentRepository:**

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;

@Repository

public interface DepartmentRepository extends JpaRepository<Department, Long> {

// Custom query to find department by name (case-insensitive)

@Query("SELECT d FROM Department d WHERE LOWER(d.name) = LOWER(:name)")

Department findDepartmentByNameIgnoreCase(@Param("name") String name);

// Custom native query to find department by ID

@Query(value = "SELECT \* FROM departments WHERE id = :id", nativeQuery = true)

Department findDepartmentByIdNative(@Param("id") Long id);

}

**Employee Entity**

import jakarta.persistence.\*;

@Entity

@Table(name = "employees")

@NamedQueries({

@NamedQuery(

name = "Employee.findByName",

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

),

@NamedQuery(

name = "Employee.findByDepartment",

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

)

})

public class Employee {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@Column(nullable = false)

private String name;

@Column(nullable = false, unique = true)

private String email;

@ManyToOne

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

private Department department;

// getters and setters

}

**employeeRepository**

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

import org.springframework.stereotype.Repository;

import java.util.List;

@Repository

public interface EmployeeRepository extends JpaRepository<Employee, Long> {

// Execute named query to find employees by name

List<Employee> findByName(String name);

// Execute named query to find employees by department name

List<Employee> findByDepartmentName(String departmentName);

}

**Exercise 6: Employee Management System - Implementing Pagination and Sorting**

**Business Scenario:**

Add pagination and sorting capabilities to your employee search functionality.

**Instructions:**

1. **Pagination:**
   * Implement pagination for the employee list using **Page** and **Pageable**.
2. **Sorting:**
   * Add sorting functionality to your queries.
   * Combine pagination and sorting in your search endpoint.

ANSWER:

**EmployeeRepository**

import org.springframework.data.domain.Page;

import org.springframework.data.domain.Pageable;

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

import org.springframework.stereotype.Repository;

@Repository

public interface EmployeeRepository extends JpaRepository<Employee, Long> {

// Method to find all employees with pagination support

Page<Employee> findAll(Pageable pageable);

// Method to find employees by department with pagination support

Page<Employee> findByDepartmentName(String departmentName, Pageable pageable);

}

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

import org.springframework.data.domain.Page;

import org.springframework.data.domain.Pageable;

import org.springframework.stereotype.Service;

@Service

public class EmployeeService {

@Autowired

private EmployeeRepository employeeRepository;

// Get all employees with pagination

public Page<Employee> getAllEmployees(Pageable pageable) {

return employeeRepository.findAll(pageable);

}

// Get employees by department with pagination

public Page<Employee> getEmployeesByDepartment(String departmentName, Pageable pageable) {

return employeeRepository.findByDepartmentName(departmentName, pageable);

}

}

**EmployeeController**

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

import org.springframework.data.domain.Page;

import org.springframework.data.domain.Pageable;

import org.springframework.web.bind.annotation.\*;

@RestController

@RequestMapping("/api/employees")

public class EmployeeController {

@Autowired

private EmployeeService employeeService;

// Get all employees with pagination

@GetMapping

public Page<Employee> getAllEmployees(Pageable pageable) {

return employeeService.getAllEmployees(pageable);

}

// Get employees by department with pagination

@GetMapping("/department/{departmentName}")

public Page<Employee> getEmployeesByDepartment(@PathVariable String departmentName, Pageable pageable) {

return employeeService.getEmployeesByDepartment(departmentName, pageable);

}

}