**//NAME- SUJATA KUMARI**

**//SUPERSET\_ID- 5004852**

**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**.

Initialized a new Spring Boot project named **EmployeeManagementSystem** and

added dependencies: **Spring Data JPA, H2 Database, Spring Web, Lombok**.

1. **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*

Created a file named **application.properties** in the **src/main/resources** directory and added the following configuration:

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

This configuration sets up the H2 in-memory database for our application.

**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**.

Created two new Java classes, **Employee.java** and **Department.java**, in the **com.example.employeemanagementsystem.entity** package.

//Employee.java

package com.example.employeemanagementsystem.entity;

import javax.persistence.\*;

import lombok.Data;

@Entity

@Table(name = "employees")

@Data

public class Employee {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

private String email;

@ManyToOne

@JoinColumn(name = "department\_id")

private Department department;

}

//Department.java

package com.example.employeemanagementsystem.entity;

import javax.persistence.\*;

import java.util.List;

import lombok.Data;

@Entity

@Table(name = "departments")

@Data

public class Department {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

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

private List<Employee> employees;

}

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

The annotations used in the entity classes will map the entities to the corresponding database tables. The relationships between the entities will also be reflected in the database schema.

**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.

Spring Data repositories provide a convenient way to perform CRUD (Create, Read, Update, Delete) operations on JPA entities. The benefits of using Spring Data repositories include:

* Simplified data access: No need to write boilerplate code for CRUD operations.
* Type-safe queries: Derived query methods allow for type-safe querying.
* Reduced code duplication: Repositories can be reused across the application.

1. **Creating Repositories:**
   * Create **EmployeeRepository** and **DepartmentRepository** interfaces extending **JpaRepository**.
   * Define derived query methods in these repositories.

Create two new Java interfaces, **EmployeeRepository.java** and **DepartmentRepository.java**, in the **com.example.employeemanagementsystem.repository** package.

//EmployeeRepository.java

package com.example.employeemanagementsystem.repository;

import com.example.employeemanagementsystem.entity.Employee;

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

import org.springframework.stereotype.Repository;

@Repository

public interface EmployeeRepository extends JpaRepository<Employee, Long> {

List<Employee> findByName(String name);

List<Employee> findByDepartment(Department department);

}

//DepartmentRepository.java

package com.example.employeemanagementsystem.repository;

import com.example.employeemanagementsystem.entity.Department;

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

import org.springframework.stereotype.Repository;

@Repository

public interface DepartmentRepository extends JpaRepository<Department, Long> {

Department findByName(String name);

}

We've added derived query methods to each repository:

* + **EmployeeRepository**: **findByName** and **findByDepartment** methods allow for querying employees by name and department, respectively.
  + **DepartmentRepository**: **findByName** method allows for querying departments by 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.

Created two new Java classes, **EmployeeService.java** and **DepartmentService.java**, in the **com.example.employeemanagementsystem.service** package. These classes will encapsulate the CRUD operations using the **JpaRepository** methods.

//EmployeeService.java

package com.example.employeemanagementsystem.service;

import com.example.employeemanagementsystem.entity.Employee;

import com.example.employeemanagementsystem.repository.EmployeeRepository;

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

import org.springframework.stereotype.Service;

import java.util.List;

@Service

public class EmployeeService {

private final EmployeeRepository employeeRepository;

@Autowired

public EmployeeService(EmployeeRepository employeeRepository) {

this.employeeRepository = employeeRepository;

}

public Employee createEmployee(Employee employee) {

return employeeRepository.save(employee);

}

public List<Employee> getAllEmployees() {

return employeeRepository.findAll();

}

public Employee getEmployeeById(Long id) {

return employeeRepository.findById(id).orElse(null);

}

public Employee updateEmployee(Employee employee) {

return employeeRepository.save(employee);

}

public void deleteEmployee(Long id) {

employeeRepository.deleteById(id);

}

}

//DepartmentService.java

package com.example.employeemanagementsystem.service;

import com.example.employeemanagementsystem.entity.Department;

import com.example.employeemanagementsystem.repository.DepartmentRepository;

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

import org.springframework.stereotype.Service;

import java.util.List;

@Service

public class DepartmentService {

private final DepartmentRepository departmentRepository;

@Autowired

public DepartmentService(DepartmentRepository departmentRepository) {

this.departmentRepository = departmentRepository;

}

public Department createDepartment(Department department) {

return departmentRepository.save(department);

}

public List<Department> getAllDepartments() {

return departmentRepository.findAll();

}

public Department getDepartmentById(Long id) {

return departmentRepository.findById(id).orElse(null);

}

public Department updateDepartment(Department department) {

return departmentRepository.save(department);

}

public void deleteDepartment(Long id) {

departmentRepository.deleteById(id);

}

}

* + Implement RESTful endpoints for these operations using **EmployeeController** and **DepartmentController**.

Created two new Java classes, **EmployeeController.java** and **DepartmentController.java**, in the **com.example.employeemanagementsystem.controller** package. These classes will expose RESTful endpoints for the CRUD operations.

//EmployeeController.java

package com.example.employeemanagementsystem.controller;

import com.example.employeemanagementsystem.entity.Employee;

import com.example.employeemanagementsystem.service.EmployeeService;

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

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

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

import java.util.List;

@RestController

@RequestMapping("/api/employees")

public class EmployeeController {

private final EmployeeService employeeService;

@Autowired

public EmployeeController(EmployeeService employeeService) {

this.employeeService = employeeService;

}

@PostMapping

public ResponseEntity<Employee> createEmployee(@RequestBody Employee employee) {

Employee createdEmployee = employeeService.createEmployee(employee);

return new ResponseEntity<>(createdEmployee, HttpStatus.CREATED);

}

@GetMapping

public ResponseEntity<List<Employee>> getAllEmployees() {

List<Employee> employees = employeeService.getAllEmployees();

return new ResponseEntity<>(employees, HttpStatus.OK);

}

@GetMapping("/{id}")

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

Employee employee = employeeService.getEmployeeById(id);

return new ResponseEntity<>(employee, HttpStatus.OK);

}

@PutMapping("/{id}")

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

Employee updatedEmployee = employeeService.updateEmployee(employee);

return new ResponseEntity<>(updatedEmployee, HttpStatus.OK);

}

@DeleteMapping("/{id}")

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

employeeService.deleteEmployee(id);

return new ResponseEntity<>(HttpStatus.NO\_CONTENT);

}

}

//DepartmentController.java

package com.example.employeemanagementsystem.controller;

import com.example.employeemanagementsystem.entity.Department;

import com.example.employeemanagementsystem.service.DepartmentService;

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

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

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

import java.util.List;

@RestController

@RequestMapping("/api/departments")

public class DepartmentController {

private final DepartmentService departmentService;

@Autowired

public DepartmentController(DepartmentService departmentService) {

this.departmentService = departmentService;

}

@PostMapping

public ResponseEntity<Department> createDepartment(@RequestBody Department department) {

Department createdDepartment = departmentService.createDepartment(department);

return new ResponseEntity<>(createdDepartment, HttpStatus.CREATED);

}

@GetMapping

public ResponseEntity<List<Department>> getAllDepartments() {

List<Department> departments = departmentService.getAllDepartments();

return new ResponseEntity<>(departments, HttpStatus.OK);

}

@GetMapping("/{id}")

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

Department department = departmentService.getDepartmentById(id);

return new ResponseEntity<>(department, HttpStatus.OK);

}

@PutMapping("/{id}")

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

Department updatedDepartment = departmentService.updateDepartment(department);

return new ResponseEntity<>(updatedDepartment, HttpStatus.OK);

}

@DeleteMapping("/{id}")

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

departmentService.deleteDepartment(id);

return new ResponseEntity<>(HttpStatus.NO\_CONTENT);

}

}

**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.

Added custom query methods to our **EmployeeRepository** and **DepartmentRepository** interfaces using keywords in method names.

//EmployeeRepository.java

package com.example.employeemanagementsystem.repository;

import com.example.employeemanagementsystem.entity.Employee;

import org.springframework.data.repository.CrudRepository;

import java.util.List;

public interface EmployeeRepository extends CrudRepository<Employee, Long> {

List<Employee> findByFirstName(String firstName);

List<Employee> findByLastName(String lastName);

List<Employee> findByDepartment\_DepartmentName(String departmentName);

List<Employee> findBySalaryGreaterThan(double salary);

}

//DepartmentRepository.java

package com.example.employeemanagementsystem.repository;

import com.example.employeemanagementsystem.entity.Department;

import org.springframework.data.repository.CrudRepository;

import java.util.List;

public interface DepartmentRepository extends CrudRepository<Department, Long> {

List<Department> findByDepartmentName(String departmentName);

List<Department> findByLocation(String location);

}

* + Implement custom query methods using the **@Query** annotation.

Added custom query methods to our **EmployeeRepository** and **DepartmentRepository** interfaces using the **@Query** annotation.

//EmployeeRepository.java (updated)

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 java.util.List;

public interface EmployeeRepository extends JpaRepository<Employee, Long> {

@Query("SELECT e FROM Employee e WHERE e.firstName = ?1")

List<Employee> findByFirstName(String firstName);

@Query("SELECT e FROM Employee e WHERE e.lastName = ?1")

List<Employee> findByLastName(String lastName);

@Query("SELECT e FROM Employee e WHERE e.department.departmentName = ?1")

List<Employee> findByDepartment\_DepartmentName(String departmentName);

@Query("SELECT e FROM Employee e WHERE e.salary > ?1")

List<Employee> findBySalaryGreaterThan(double salary);

}

//DepartmentRepository.java (updated)

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 java.util.List;

public interface DepartmentRepository extends JpaRepository<Department, Long> {

@Query("SELECT d FROM Department d WHERE d.departmentName = ?1")

List<Department> findByDepartmentName(String departmentName);

@Query("SELECT d FROM Department d WHERE d.location = ?1")

List<Department> findByLocation(String location);

}

1. **Named Queries:**
   * Define and execute named queries with **@NamedQuery** and **@NamedQueries**.

Defined named queries using the **@NamedQuery** and **@NamedQueries** annotations.

//Employee.java (updated)

package com.example.employeemanagementsystem.entity;

import javax.persistence.Entity;

import javax.persistence.NamedQueries;

import javax.persistence.NamedQuery;

import javax.persistence.Table;

@Entity

@Table(name = "employees")

@NamedQueries({

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

@NamedQuery(name = "Employee.findBySalaryRange", query = "SELECT e FROM Employee e WHERE e.salary BETWEEN :minSalary AND :maxSalary")

})

public class Employee {

// ...

}

//Department.java (updated)

package com.example.employeemanagementsystem.entity;

import javax.persistence.Entity;

import javax.persistence.NamedQueries;

import javax.persistence.NamedQuery;

import javax.persistence.Table;

@Entity

@Table(name = "departments")

@NamedQueries({

@NamedQuery(name = "Department.findByLocation", query = "SELECT d FROM Department d WHERE d.location = :location")

})

public class Department {

// ...

}

We can execute these named queries using the **createNamedQuery** method of the **EntityManager**.

**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**.

//EmployeeRepository.java (updated)

package com.example.employeemanagementsystem.repository;

import com.example.employeemanagementsystem.entity.Employee;

import org.springframework.data.domain.Page;

import org.springframework.data.domain.Pageable;

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

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

import java.util.List;

public interface EmployeeRepository extends JpaRepository<Employee, Long> {

@Query("SELECT e FROM Employee e WHERE e.firstName = ?1")

List<Employee> findByFirstName(String firstName);

@Query("SELECT e FROM Employee e WHERE e.lastName = ?1")

List<Employee> findByLastName(String lastName);

@Query("SELECT e FROM Employee e WHERE e.department.departmentName = ?1")

List<Employee> findByDepartment\_DepartmentName(String departmentName);

@Query("SELECT e FROM Employee e WHERE e.salary > ?1")

List<Employee> findBySalaryGreaterThan(double salary);

Page<Employee> findAll(Pageable pageable);

}

1. **Sorting:**
   * Add sorting functionality to your queries.

//EmployeeRepository.java (updated)

package com.example.employeemanagementsystem.repository;

import com.example.employeemanagementsystem.entity.Employee;

import org.springframework.data.domain.Page;

import org.springframework.data.domain.Pageable;

import org.springframework.data.domain.Sort;

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

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

import java.util.List;

public interface EmployeeRepository extends JpaRepository<Employee, Long> {

@Query("SELECT e FROM Employee e WHERE e.firstName = ?1")

List<Employee> findByFirstName(String firstName);

@Query("SELECT e FROM Employee e WHERE e.lastName = ?1")

List<Employee> findByLastName(String lastName);

@Query("SELECT e FROM Employee e WHERE e.department.departmentName ?1")

List<Employee> findByDepartment\_DepartmentName(String departmentName);

@Query("SELECT e FROM Employee e WHERE e.salary > ?1")

List<Employee> findBySalaryGreaterThan(double salary);

Page<Employee> findAll(Pageable pageable);

List<Employee> findAll(Sort sort);

Page<Employee> findAll(Pageable pageable, Sort sort);

}

* + Combine pagination and sorting in your search endpoint.

//EmployeeService.java (updated)

package com.example.employeemanagementsystem.service;

import com.example.employeemanagementsystem.entity.Employee;

import com.example.employeemanagementsystem.repository.EmployeeRepository;

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

import org.springframework.data.domain.Page;

import org.springframework.data.domain.Pageable;

import org.springframework.data.domain.Sort;

import org.springframework.stereotype.Service;

import java.util.List;

@Service

public class EmployeeService {

@Autowired

private EmployeeRepository employeeRepository;

public Page<Employee> searchEmployees(String firstName, String lastName, String departmentName, double salary, Pageable pageable, Sort sort) {

if (firstName != null && !firstName.isEmpty()) {

return employeeRepository.findByFirstName(firstName, pageable);

} else if (lastName != null && !lastName.isEmpty()) {

return employeeRepository.findByLastName(lastName, pageable);

} else if (departmentName != null && !departmentName.isEmpty()) {

return employeeRepository.findByDepartment\_DepartmentName(departmentName, pageable);

} else if (salary > 0) {

return employeeRepository.findBySalaryGreaterThan(salary, pageable);

} else {

return employeeRepository.findAll(pageable, sort);

} }}

**Exercise 7: Employee Management System - Enabling Entity Auditing**

**Business Scenario:**

Implement auditing to track the creation and modification of employees and departments.

**Instructions:**

1. **Entity Auditing:**
   * Enable auditing in your application by configuring auditing properties.

//application.properties

spring.jpa.properties.hibernate.ejb.use\_class\_enhancer=true

spring.jpa.properties.javax.persistence.provider=org.hibernate.jpa.HibernatePersistenceProvider

spring.jpa.properties.hibernate.enhancer. enabled=true

spring.jpa.properties.hibernate.ejb.event.post-insert=true

spring.jpa.properties.hibernate.ejb.event.post-update=true

spring.jpa.properties.hibernate.ejb.event.post-delete=true

* + Use annotations like **@CreatedBy, @LastModifiedBy, @CreatedDate**, and **@LastModifiedDate**.

//Employee.java (updated)

package com.example.employeemanagementsystem.entity;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

import javax.persistence.JoinColumn;

import javax.persistence.ManyToOne;

import javax.persistence.NamedQueries;

import javax.persistence.NamedQuery;

import javax.persistence.Table;

import org.springframework.data.annotation.CreatedBy;

import org.springframework.data.annotation.CreatedDate;

import org.springframework.data.annotation.LastModifiedBy;

import org.springframework.data.annotation.LastModifiedDate;

@Entity

@Table(name = "employees")

@NamedQueries({

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

@NamedQuery(name = "Employee.findBySalaryRange", query = "SELECT e FROM Employee e WHERE e.salary BETWEEN :minSalary AND :maxSalary")

})

public class Employee {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String firstName;

private String lastName;

private double salary;

@ManyToOne

@JoinColumn(name = "department\_id")

private Department department;

@CreatedBy

private String createdBy;

@CreatedDate

private java.util.Date createdDate;

@LastModifiedBy

private String lastModifiedBy;

@LastModifiedDate

private java.util.Date lastModifiedDate;

// getters and setters

}

//Department.java

package com.example.employeemanagementsystem.entity;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

import javax.persistence.NamedQueries;

import javax.persistence.NamedQuery;

import javax.persistence.OneToMany;

import javax.persistence.Table;

import org.springframework.data.annotation.CreatedBy;

import org.springframework.data.annotation.CreatedDate;

import org.springframework.data.annotation.LastModifiedBy;

import org.springframework.data.annotation.LastModifiedDate;

import java.util.ArrayList;

import java.util.List;

@Entity

@Table(name = "departments")

@NamedQueries({

@NamedQuery(name = "Department.findByLocation", query = "SELECT d FROM Department d WHERE d.location = :location")

})

public class Department {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String departmentName;

private String location;

@OneToMany(mappedBy = "department")

private List<Employee> employees;

@CreatedBy

private String createdBy;

@CreatedDate

private java.util.Date createdDate;

@LastModifiedBy

private String lastModifiedBy;

@LastModifiedDate

private java.util.Date lastModifiedDate;

// getters and setters

}

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

**Business Scenario:**

Create projections to fetch specific data subsets from the employee and department entities.

**Instructions:**

1. **Projections:**
   * Define interface-based and class-based projections.

//EmployeeSummary.java (interface-based projection)

package com.example.employeemanagementsystem.projection;

public interface EmployeeSummary {

Long getId();

String getFirstName();

String getLastName();

double getSalary();

}

//DepartmentSummary.java (interface-based projection)

package com.example.employeemanagementsystem.projection;

public interface DepartmentSummary {

Long getId();

String getDepartmentName();

String getLocation();

}

//EmployeeDetails.java (class-based projection)

package com.example.employeemanagementsystem.projection;

public class EmployeeDetails {

private Long id;

private String firstName;

private String lastName;

private double salary;

private DepartmentSummary department;

public EmployeeDetails(Long id, String firstName, String lastName, double salary, DepartmentSummary department) {

this.id = id;

this.firstName = firstName;

this.lastName = lastName;

this.salary = salary;

this.department = department;

}

// getters and setters

}

//DepartmentDetails.java (class-based projection)

package com.example.employeemanagementsystem.projection;

public class DepartmentDetails {

private Long id;

private String departmentName;

private String location;

public DepartmentDetails(Long id, String departmentName, String location) {

this.id = id;

this.departmentName = departmentName;

this.location = location;

}

// getters and setters

}

* + Use **@Value** and constructor expressions to control the fetched data.

//EmployeeRepository.java (updated)

package com.example.employeemanagementsystem.repository;

import com.example.employeemanagementsystem.entity.Employee;

import com.example.employeemanagementsystem.projection.EmployeeSummary;

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

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

import java.util.List;

public interface EmployeeRepository extends JpaRepository<Employee, Long> {

@Query("SELECT new com.example.employeemanagementsystem.projection.EmployeeSummary(e.id, e.firstName, e.lastName, e.salary) FROM Employee e")

List<EmployeeSummary> findEmployeeSummaries();

@Query("SELECT new com.example.employeemanagementsystem.projection.EmployeeDetails(e.id, e.firstName, e.lastName, e.salary, new com.example.employeemanagementsystem.projection.DepartmentSummary(e.department.id, e.department.departmentName, e.department.location)) FROM Employee e")

List<EmployeeDetails> findEmployeeDetails();

}

//DepartmentRepository.java (updated)

package com.example.employeemanagementsystem.repository;

import com.example.employeemanagementsystem.entity.Department;

import com.example.employeemanagementsystem.projection.DepartmentSummary;

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

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

import java.util.List;

public interface DepartmentRepository extends JpaRepository<Department, Long> {

@Query("SELECT new com.example.employeemanagementsystem.projection.DepartmentSummary(d.id, d.departmentName, d.location) FROM Department d")

List<DepartmentSummary> findDepartmentSummaries();

@Query("SELECT new com.example.employeemanagementsystem.projection.DepartmentDetails(d.id, d.departmentName, d.location) FROM Department d")

List<DepartmentDetails> findDepartmentDetails();

}

**Exercise 9: Employee Management System - Customizing Data Source Configuration**

**Business Scenario:**

Customize your data source configuration and manage multiple data sources.

**Instructions:**

1. **Spring Boot Auto-Configuration:**
   * Leverage Spring Boot auto-configuration for data sources.

//DataSourceConfig.java

package com.example.employeemanagementsystem.config;

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

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.jdbc.datasource.DriverManagerDataSource;

import javax.sql.DataSource;

@Configuration

public class DataSourceConfig {

@Value("${spring.datasource.url}")

private String url;

@Value("${spring.datasource.username}")

private String username;

@Value("${spring.datasource.password}")

private String password;

@Bean

public DataSource dataSource() {

DriverManagerDataSource dataSource = new DriverManagerDataSource();

dataSource.setUrl(url);

dataSource.setUsername(username);

dataSource.setPassword(password);

return dataSource;

}

}

1. **Externalizing Configuration:**
   * Externalize configuration with application.properties.

//application.properties

spring.datasource.url=jdbc:mysql://localhost:3306/employee\_management

spring.datasource.username=root

spring.datasource.password=password

* + Manage multiple data sources within your application.

//DataSourceConfig.java (updated)

package com.example.employeemanagementsystem.config;

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

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.jdbc.datasource.DriverManagerDataSource;

import javax.sql.DataSource;

@Configuration

public class DataSourceConfig {

@Value("${spring.datasource.url}")

private String url;

@Value("${spring.datasource.username}")

private String username;

@Value("${spring.datasource.password}")

private String password;

@Value("${spring.datasource2.url}")

private String url2;

@Value("${spring.datasource2.username}")

private String username2;

@Value("${spring.datasource2.password}")

private String password2;

@Bean

public DataSource dataSource() {

DriverManagerDataSource dataSource = new DriverManagerDataSource();

dataSource.setUrl(url);

dataSource.setUsername(username);

dataSource.setPassword(password);

return dataSource;

}

@Bean

public DataSource dataSource2() {

DriverManagerDataSource dataSource = new DriverManagerDataSource();

dataSource.setUrl(url2);

dataSource.setUsername(username2);

dataSource.setPassword(password2);

return dataSource;

}

}

//application.properties (updated)

spring.datasource.url=jdbc:mysql://localhost:3306/employee\_management

spring.datasource.username=root

spring.datasource.password=password

spring.datasource2.url=jdbc:mysql://localhost:3306/department\_management

spring.datasource2.username=root

spring.datasource2.password=password

**Exercise 10: Employee Management System - Hibernate-Specific Features**

**Business Scenario:**

Leverage Hibernate-specific features to enhance your application's performance and capabilities.

**Instructions:**

1. **Hibernate-Specific Annotations:**
   * Use Hibernate-specific annotations to customize entity mappings.

//Employee.java (updated)

package com.example.employeemanagementsystem.entity;

import org.hibernate.annotations.DynamicInsert;

import org.hibernate.annotations.DynamicUpdate;

import org.hibernate.annotations.SelectBeforeUpdate;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

@Entity

@DynamicInsert

@DynamicUpdate

@SelectBeforeUpdate

public class Employee {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String firstName;

private String lastName;

private double salary;

private Department department;

// getters and setters

}

1. **Configuring Hibernate Dialect and Properties:**
   * Configure Hibernate dialect and properties for optimal performance.

//application.properties (updated)

spring.jpa.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect

spring.jpa.hibernate.ddl-auto=update

spring.jpa.properties.hibernate.jdbc.batch\_size=20

spring.jpa.properties.hibernate.order\_inserts=true

spring.jpa.properties.hibernate.order\_updates=true

1. **Batch Processing:**
   * Implement batch processing with Hibernate for bulk operations.

//EmployeeRepository.java (updated)

package com.example.employeemanagementsystem.repository;

import com.example.employeemanagementsystem.entity.Employee;

import org.springframework.stereotype.Repository;

import org.springframework.transaction.annotation.Transactional;

import javax.persistence.EntityManager;

import javax.persistence.PersistenceContext;

import java.util.List;

@Repository

public class EmployeeRepository {

@PersistenceContext

private EntityManager entityManager;

@Transactional

public void batchInsertEmployees(List<Employee> employees) {

int batchSize = 20;

for (int i = 0; i < employees.size(); i++) {

entityManager.persist(employees.get(i));

if ((i + 1) % batchSize == 0) {

entityManager.flush();

entityManager.clear();

}

}

}

}