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

**1. Creating a Spring Boot Project:**

* Set the project metadata:
  + Group: com.example
  + Artifact: EmployeeManagementSystem
  + Name: EmployeeManagementSystem
  + Description: Employee Management System
  + Package Name: com.example.employeemanagementsystem
  + Packaging: Jar
  + Java: 17
  + Add dependencies:
  + Spring Data JPA
  + H2 Database
  + Spring Web
  + Lombok

**2. Configuring Application Properties:**

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**

**1. Creating JPA Entities:**

* **Employee Entity:**

package com.example.employeemanagementsystem.entity;

import lombok.Data;

import javax.persistence.\*;

@Data

@Entity

@Table(name = "employees")

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 Entity:**

package com.example.employeemanagementsystem.entity;

import lombok.Data;

import javax.persistence.\*;

import java.util.List;

@Data

@Entity

@Table(name = "departments")

public class Department {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

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

private List<Employee> employees;

}

**2. Mapping Entities to Database Tables:**

* The @Entity annotation specifies that the class is a JPA entity.
* @Table maps the entity to a specific table in the database.
* @Id and @GeneratedValue specify the primary key and the generation strategy.
* @ManyToOne and @OneToMany define the relationships between Employee and Department.

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

**1. Overview of Spring Data Repositories:**

Spring Data repositories simplify data access by providing common CRUD operations out of the box. By extending JpaRepository, you inherit several methods such as save, findAll, findById, and delete.

**2. Creating Repositories:**

* **EmployeeRepository:**

package com.example.employeemanagementsystem.repository;

import com.example.employeemanagementsystem.entity.Employee;

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

public interface EmployeeRepository extends JpaRepository<Employee, Long> {

}

* **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> {

}

These repositories enable basic CRUD operations for Employee and Department.

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

**1. Basic CRUD Operations:**

* **EmployeeController:**

package com.example.employeemanagementsystem.controller;

import com.example.employeemanagementsystem.entity.Employee;

import com.example.employeemanagementsystem.repository.EmployeeRepository;

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

import org.springframework.http.ResponseEntity;

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

import java.util.List;

@RestController

@RequestMapping("/employees")

public class EmployeeController {

@Autowired

private EmployeeRepository employeeRepository;

@GetMapping

public List<Employee> getAllEmployees() {

return employeeRepository.findAll();

}

@GetMapping("/{id}")

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

return employeeRepository.findById(id)

.map(employee -> ResponseEntity.ok().body(employee))

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

}

@PostMapping

public Employee createEmployee(@RequestBody Employee employee) {

return employeeRepository.save(employee);

}

@PutMapping("/{id}")

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

return employeeRepository.findById(id)

.map(employee -> {

employee.setName(employeeDetails.getName());

employee.setEmail(employeeDetails.getEmail());

employee.setDepartment(employeeDetails.getDepartment());

return ResponseEntity.ok(employeeRepository.save(employee));

})

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

}

@DeleteMapping("/{id}")

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

return employeeRepository.findById(id)

.map(employee -> {

employeeRepository.delete(employee);

return ResponseEntity.ok().build();

})

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

}

}

* **DepartmentController:**

package com.example.employeemanagementsystem.controller;

import com.example.employeemanagementsystem.entity.Department;

import com.example.employeemanagementsystem.repository.DepartmentRepository;

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

import org.springframework.http.ResponseEntity;

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

import java.util.List;

@RestController

@RequestMapping("/departments")

public class DepartmentController {

@Autowired

private DepartmentRepository departmentRepository;

@GetMapping

public List<Department> getAllDepartments() {

return departmentRepository.findAll();

}

@GetMapping("/{id}")

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

return departmentRepository.findById(id)

.map(department -> ResponseEntity.ok().body(department))

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

}

@PostMapping

public Department createDepartment(@RequestBody Department department) {

return departmentRepository.save(department);

}

@PutMapping("/{id}")

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

return departmentRepository.findById(id)

.map(department -> {

department.setName(departmentDetails.getName());

return ResponseEntity.ok(departmentRepository.save(department));

})

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

}

@DeleteMapping("/{id}")

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

return departmentRepository.findById(id)

.map(department -> {

departmentRepository.delete(department);

return ResponseEntity.ok().build();

})

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

}

}

**2. Implementing RESTful Endpoints:**

These controllers expose RESTful endpoints for performing CRUD operations on Employee and Department entities.

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

**1. Defining Query Methods:**

You can define custom query methods by using method names or the @Query annotation.

* **Derived Query Example:**

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> {

List<Employee> findByDepartmentName(String departmentName);

}

* **Custom Query Example:**

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> {

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

List<Employee> findByNameLike(@Param("name") String name);

}

**2. Named Queries:**

* **Named Query Example:**

package com.example.employeemanagementsystem.entity;

import lombok.Data;

import javax.persistence.\*;

@Data

@Entity

@Table(name = "employees")

@NamedQuery(name = "Employee.findByDepartmentName",

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

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;

}

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

**1. Pagination:**

To implement pagination, use Page and Pageable interfaces in the repository methods.

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;

public interface EmployeeRepository extends JpaRepository<Employee, Long> {

Page<Employee> findAll(Pageable pageable);

}

* **Controller Example:**

package com.example.employeemanagementsystem.controller;

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.PageRequest;

import org.springframework.data.domain.Pageable;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RequestParam;

import org.springframework.web.bind.annotation.RestController;

@RestController

@RequestMapping("/employees")

public class EmployeeController {

@Autowired

private EmployeeRepository employeeRepository;

@GetMapping("/paged")

public Page<Employee> getAllEmployeesPaged(

@RequestParam(defaultValue = "0") int page,

@RequestParam(defaultValue = "10") int size) {

Pageable pageable = PageRequest.of(page, size);

return employeeRepository.findAll(pageable);

}

}

**2. Sorting:**

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;

public interface EmployeeRepository extends JpaRepository<Employee, Long> {

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

}

* **Controller Example:**

package com.example.employeemanagementsystem.controller;

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.PageRequest;

import org.springframework.data.domain.Pageable;

import org.springframework.data.domain.Sort;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RequestParam;

import org.springframework.web.bind.annotation.RestController;

@RestController

@RequestMapping("/employees")

public class EmployeeController {

@Autowired

private EmployeeRepository employeeRepository;

@GetMapping("/pagedSorted")

public Page<Employee> getAllEmployeesPagedSorted(

@RequestParam(defaultValue = "0") int page,

@RequestParam(defaultValue = "10") int size,

@RequestParam(defaultValue = "name") String sortBy) {

Pageable pageable = PageRequest.of(page, size, Sort.by(sortBy));

return employeeRepository.findAll(pageable);

}

}

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

**1. Entity Auditing:**

**Step 1: Enable Auditing**

* **Application Class:**

package com.example.employeemanagementsystem;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.data.jpa.repository.config.EnableJpaAuditing;

@SpringBootApplication

@EnableJpaAuditing

public class EmployeeManagementSystemApplication {

public static void main(String[] args) {

SpringApplication.run(EmployeeManagementSystemApplication.class, args);

}

}

**Step 2: Add Auditing Fields to Entities**

* **Employee Entity:**

package com.example.employeemanagementsystem.entity;

import lombok.Data;

import org.springframework.data.annotation.CreatedDate;

import org.springframework.data.annotation.LastModifiedDate;

import org.springframework.data.jpa.domain.support.AuditingEntityListener;

import javax.persistence.\*;

import java.time.LocalDateTime;

@Data

@Entity

@Table(name = "employees")

@EntityListeners(AuditingEntityListener.class)

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;

@CreatedDate

private LocalDateTime createdDate;

@LastModifiedDate

private LocalDateTime lastModifiedDate;

}

* **Department Entity:**

package com.example.employeemanagementsystem.entity;

import lombok.Data;

import org.springframework.data.annotation.CreatedDate;

import org.springframework.data.annotation.LastModifiedDate;

import org.springframework.data.jpa.domain.support.AuditingEntityListener;

import javax.persistence.\*;

import java.time.LocalDateTime;

import java.util.List;

@Data

@Entity

@Table(name = "departments")

@EntityListeners(AuditingEntityListener.class)

public class Department {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

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

private List<Employee> employees;

@CreatedDate

private LocalDateTime createdDate;

@LastModifiedDate

private LocalDateTime lastModifiedDate;

}

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

**1. Projections:**

Projections allow you to fetch specific fields from entities, either as an interface or a class.

* **Interface-Based Projection:**

package com.example.employeemanagementsystem.projection;

public interface EmployeeNameProjection {

String getName();

}

* **Repository Method:**

package com.example.employeemanagementsystem.repository;

import com.example.employeemanagementsystem.entity.Employee;

import com.example.employeemanagementsystem.projection.EmployeeNameProjection;

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

import java.util.List;

public interface EmployeeRepository extends JpaRepository<Employee, Long> {

List<EmployeeNameProjection> findByDepartmentName(String departmentName);

}

* **Class-Based Projection:**

package com.example.employeemanagementsystem.dto;

public class EmployeeDTO {

private String name;

private String email;

public EmployeeDTO(String name, String email) {

this.name = name;

this.email = email;

}

// Getters and Setters

}

* **Repository Method with Class-Based Projection:**

package com.example.employeemanagementsystem.repository;

import com.example.employeemanagementsystem.dto.EmployeeDTO;

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 new com.example.employeemanagementsystem.dto.EmployeeDTO(e.name, e.email) FROM Employee e WHERE e.department.name = :departmentName")

List<EmployeeDTO> findEmployeeDTOByDepartmentName(String departmentName);

}

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

**1. Customizing Data Source Configuration:**

* **Example Configuration:**

spring.datasource.url=jdbc:mysql://localhost:3306/employeedb

spring.datasource.username=root

spring.datasource.password=secret

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.jpa.database-platform=org.hibernate.dialect.MySQL5Dialect

**2. Managing Multiple Data Sources:**

To configure multiple data sources, you can define multiple DataSource beans.

* **Primary DataSource Configuration:**

package com.example.employeemanagementsystem.config;

import org.springframework.boot.autoconfigure.jdbc.DataSourceProperties;

import org.springframework.boot.context.properties.ConfigurationProperties;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.context.annotation.Primary;

import javax.sql.DataSource;

@Configuration

public class DataSourceConfig {

@Primary

@Bean(name = "primaryDataSource")

@ConfigurationProperties("spring.datasource.primary")

public DataSource primaryDataSource(DataSourceProperties properties) {

return properties.initializeDataSourceBuilder().build();

}

@Bean(name = "secondaryDataSource")

@ConfigurationProperties("spring.datasource.secondary")

public DataSource secondaryDataSource(DataSourceProperties properties) {

return properties.initializeDataSourceBuilder().build();

}

}

**3. Example Properties for Multiple Data Sources:**

spring.datasource.primary.url=jdbc:mysql://localhost:3306/employeedb

spring.datasource.primary.username=root

spring.datasource.primary.password=secret

spring.datasource.primary.driver-class-name=com.mysql.cj.jdbc.Driver

spring.datasource.secondary.url=jdbc:h2:mem:secondarydb

spring.datasource.secondary.driver-class-name=org.h2.Driver

spring.datasource.secondary.username=sa

spring.datasource.secondary.password=password

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

**1. Hibernate-Specific Annotations:**

Hibernate provides additional annotations for customizing mappings.

* **Example:**

package com.example.employeemanagementsystem.entity;

import lombok.Data;

import org.hibernate.annotations.DynamicUpdate;

import org.hibernate.annotations.SelectBeforeUpdate;

import javax.persistence.\*;

@Data

@Entity

@Table(name = "employees")

@DynamicUpdate

@SelectBeforeUpdate

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;

}

**2. Configuring Hibernate Dialect and Properties:**

To optimize performance, you can configure Hibernate properties in application.properties.

* **Example Configuration:**

spring.jpa.properties.hibernate.jdbc.batch\_size=50

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

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

spring.jpa.properties.hibernate.cache.use\_second\_level\_cache=true

spring.jpa.properties.hibernate.cache.use\_query\_cache=true

spring.jpa.properties.hibernate.cache.region.factory\_class=org.hibernate.cache.ehcache.EhCacheRegionFactory

**3. Batch Processing with Hibernate:**

Hibernate supports batch processing, which can be configured to improve performance when performing bulk operations.

* **Batch Processing Example:**

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 EmployeeBatchRepository {

@PersistenceContext

private EntityManager entityManager;

@Transactional

public void saveEmployeesBatch(List<Employee> employees) {

int batchSize = 50;

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

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

if (i % batchSize == 0 && i > 0) {

entityManager.flush();

entityManager.clear();

}

}

}

}