**Spring Data Core JPA & Hibernate**

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

Resources>Application.properties:

# Application Name

spring.application.name=EmployeeManagementSystem

# H2 Database Configuration

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

spring.datasource.driverClassName=org.h2.Driver

spring.datasource.username=sa

spring.datasource.password=yourPassword

# JPA Configuration

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

spring.jpa.show-sql=true

# H2 Console Configuration

spring.h2.console.enabled=true

spring.h2.console.path=/h2-console

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

Deparment.java:

package com.example.EmployeeManagementSystem.model;

import jakarta.persistence.\*;

import java.util.HashSet;

import java.util.Set;

@Entity

@Table(name = "departments")

public class Department {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@Column(name = "name", nullable = false, unique = true)

private String name;

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

private Set<Employee> employees = new HashSet<>();

// No-argument constructor

public Department() {}

// Constructor with department name

public Department(String name) {

this.name = name;

}

// Getter and Setter for id

public Long getId() {

return id;

}

public void setId(Long id) {

this.id = id;

}

// Getter and Setter for name

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

// Getter and Setter for employees set

public Set<Employee> getEmployees() {

return employees;

}

public void setEmployees(Set<Employee> employees) {

this.employees = employees;

}

// Method to add an employee

public void addEmployee(Employee employee) {

employees.add(employee);

employee.setDepartment(this);

}

// Method to remove an employee

public void removeEmployee(Employee employee) {

employees.remove(employee);

employee.setDepartment(null);

}

}

Employee.java

package com.example.EmployeeManagementSystem.model;

import jakarta.persistence.\*;

@Entity

@Table(name = "employees")

public class Employee {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@Column(name = "name", nullable = false)

private String name;

@Column(name = "email", nullable = false, unique = true)

private String email;

@ManyToOne(fetch = FetchType.LAZY)

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

private Department department;

// No-argument constructor

public Employee() {}

// Parameterized constructor

public Employee(String name, String email, Department department) {

this.name = name;

this.email = email;

this.department = department;

}

// Getter and Setter for id

public Long getId() {

return id;

}

public void setId(Long id) {

this.id = id;

}

// Getter and Setter for name

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

// Getter and Setter for email

public String getEmail() {

return email;

}

public void setEmail(String email) {

this.email = email;

}

// Getter and Setter for department

public Department getDepartment() {

return department;

}

public void setDepartment(Department department) {

this.department = department;

}

}

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

EmployeeRepository.java:

package com.example.EmployeeManagementSystem.repository;

import com.example.EmployeeManagementSystem.model.Employee;

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

import org.springframework.stereotype.Repository;

import java.util.List;

@Repository

public interface EmployeeRepository extends JpaRepository<Employee, Long> {

// Query to get a list of employees matching the specified name

List<Employee> findAllByName(String name);

// Query to retrieve an employee based on their email address

Employee findOneByEmail(String email);

// Query to find employees associated with a particular department ID

List<Employee> findAllByDepartmentId(Long departmentId);

}

DepartmentRepository.java:

package com.example.EmployeeManagementSystem.repository;

import com.example.EmployeeManagementSystem.model.Department;

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

import org.springframework.stereotype.Repository;

@Repository

public interface DepartmentRepository extends JpaRepository<Department, Long> {

// Query to locate a department using its name

Department findByDepartmentName(String name);

}

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

EmployeeController.java:

package com.example.EmployeeManagementSystem.controller;

import com.example.EmployeeManagementSystem.model.Employee;

import com.example.EmployeeManagementSystem.repository.EmployeeRepository;

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 EmployeeRepository employeeRepo;

@Autowired

public EmployeeController(EmployeeRepository employeeRepo) {

this.employeeRepo = employeeRepo;

}

// Endpoint to create a new employee

@PostMapping

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

Employee savedEmployee = employeeRepo.save(employee);

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

}

// Endpoint to retrieve a list of all employees

@GetMapping

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

List<Employee> employees = employeeRepo.findAll();

return ResponseEntity.ok(employees);

}

// Endpoint to retrieve an employee by their ID

@GetMapping("/{id}")

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

return employeeRepo.findById(id)

.map(ResponseEntity::ok)

.orElseGet(() -> ResponseEntity.notFound().build());

}

// Endpoint to update details of an existing employee

@PutMapping("/{id}")

public ResponseEntity<Employee> modifyEmployee(@PathVariable Long id, @RequestBody Employee updatedDetails) {

return employeeRepo.findById(id)

.map(existingEmployee -> {

existingEmployee.setName(updatedDetails.getName());

existingEmployee.setEmail(updatedDetails.getEmail());

existingEmployee.setDepartment(updatedDetails.getDepartment());

employeeRepo.save(existingEmployee);

return ResponseEntity.ok(existingEmployee);

})

.orElseGet(() -> ResponseEntity.notFound().build());

}

// Endpoint to remove an employee by their ID

@DeleteMapping("/{id}")

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

return employeeRepo.findById(id)

.map(employee -> {

employeeRepo.delete(employee);

return ResponseEntity.noContent().build();

})

.orElseGet(() -> ResponseEntity.notFound().build());

}

}

DepartmentController.java

package com.example.EmployeeManagementSystem.controller;

import com.example.EmployeeManagementSystem.model.Department;

import com.example.EmployeeManagementSystem.repository.DepartmentRepository;

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 DepartmentRepository departmentRepo;

@Autowired

public DepartmentController(DepartmentRepository departmentRepo) {

this.departmentRepo = departmentRepo;

}

// Endpoint to add a new department

@PostMapping

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

Department savedDepartment = departmentRepo.save(department);

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

}

// Endpoint to retrieve all departments

@GetMapping

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

List<Department> departments = departmentRepo.findAll();

return ResponseEntity.ok(departments);

}

// Endpoint to retrieve a department by its ID

@GetMapping("/{id}")

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

return departmentRepo.findById(id)

.map(ResponseEntity::ok)

.orElseGet(() -> ResponseEntity.notFound().build());

}

// Endpoint to update an existing department

@PutMapping("/{id}")

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

return departmentRepo.findById(id)

.map(existingDepartment -> {

existingDepartment.setName(updatedDepartment.getName());

departmentRepo.save(existingDepartment);

return ResponseEntity.ok(existingDepartment);

})

.orElseGet(() -> ResponseEntity.notFound().build());

}

// Endpoint to remove a department by its ID

@DeleteMapping("/{id}")

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

return departmentRepo.findById(id)

.map(department -> {

departmentRepo.delete(department);

return ResponseEntity.noContent().build();

})

.orElseGet(() -> ResponseEntity.notFound().build());

}

}

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

Employee.java:

package com.example.EmployeeManagementSystem.model;

import jakarta.persistence.\*;

import lombok.Data;

@Data

@Entity

@Table(name = "employees")

@NamedQueries({

@NamedQuery(

name = "Employee.findByDeptNameNamed",

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

),

@NamedQuery(

name = "Employee.findByEmailNamed",

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

)

})

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(fetch = FetchType.LAZY)

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

private Department department;

}

EmployeeController.java:

package com.example.EmployeeManagementSystem.controller;

import com.example.EmployeeManagementSystem.model.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("/api/employees")

public class EmployeeController {

private final EmployeeRepository empRepo;

@Autowired

public EmployeeController(EmployeeRepository empRepo) {

this.empRepo = empRepo;

}

@PostMapping

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

Employee savedEmployee = empRepo.save(employee);

return ResponseEntity.status(201).body(savedEmployee);

}

@GetMapping

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

List<Employee> employees = empRepo.findAll();

return ResponseEntity.ok(employees);

}

@GetMapping("/{id}")

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

return empRepo.findById(id)

.map(ResponseEntity::ok)

.orElseGet(() -> ResponseEntity.notFound().build());

}

@PutMapping("/{id}")

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

return empRepo.findById(id)

.map(existingEmployee -> {

existingEmployee.setName(updatedEmployee.getName());

existingEmployee.setEmail(updatedEmployee.getEmail());

existingEmployee.setDepartment(updatedEmployee.getDepartment());

empRepo.save(existingEmployee);

return ResponseEntity.ok(existingEmployee);

})

.orElseGet(() -> ResponseEntity.notFound().build());

}

@DeleteMapping("/{id}")

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

return empRepo.findById(id)

.map(employee -> {

empRepo.delete(employee);

return ResponseEntity.noContent().build();

})

.orElseGet(() -> ResponseEntity.notFound().build());

}

@GetMapping("/search/by-name")

public ResponseEntity<List<Employee>> searchByName(@RequestParam String name) {

List<Employee> employees = empRepo.findByNameContainingIgnoreCase(name);

return ResponseEntity.ok(employees);

}

@GetMapping("/search/by-department")

public ResponseEntity<List<Employee>> searchByDeptName(@RequestParam String deptName) {

List<Employee> employees = empRepo.findByDeptName(deptName);

return ResponseEntity.ok(employees);

}

@GetMapping("/search/by-department-jpql")

public ResponseEntity<List<Employee>> searchByDeptNameJPQL(@RequestParam String deptName) {

List<Employee> employees = empRepo.findByDeptNameJPQL(deptName);

return ResponseEntity.ok(employees);

}

@GetMapping("/search/by-name-native")

public ResponseEntity<List<Employee>> searchByNameNative(@RequestParam String name) {

List<Employee> employees = empRepo.findByNameNativeSQL(name);

return ResponseEntity.ok(employees);

}

}

DepartmentController.java:

package com.example.EmployeeManagementSystem.controller;

import com.example.EmployeeManagementSystem.model.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("/api/departments")

public class DepartmentController {

private final DepartmentRepository deptRepo;

@Autowired

public DepartmentController(DepartmentRepository deptRepo) {

this.deptRepo = deptRepo;

}

@PostMapping

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

Department savedDepartment = deptRepo.save(department);

return ResponseEntity.status(201).body(savedDepartment);

}

@GetMapping

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

List<Department> departments = deptRepo.findAll();

return ResponseEntity.ok(departments);

}

@GetMapping("/{id}")

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

return deptRepo.findById(id)

.map(ResponseEntity::ok)

.orElseGet(() -> ResponseEntity.notFound().build());

}

@PutMapping("/{id}")

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

return deptRepo.findById(id)

.map(existingDepartment -> {

existingDepartment.setName(updatedDepartment.getName());

deptRepo.save(existingDepartment);

return ResponseEntity.ok(existingDepartment);

})

.orElseGet(() -> ResponseEntity.notFound().build());

}

@DeleteMapping("/{id}")

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

return deptRepo.findById(id)

.map(department -> {

deptRepo.delete(department);

return ResponseEntity.noContent().build();

})

.orElseGet(() -> ResponseEntity.notFound().build());

}

}

DepartmentRepository.java

package com.example.EmployeeManagementSystem.repository;

import com.example.EmployeeManagementSystem.model.Department;

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

import org.springframework.stereotype.Repository;

@Repository

public interface DepartmentRepository extends JpaRepository<Department, Long> {}

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

Employee.java

package com.example.EmployeeManagementSystem.model;

import jakarta.persistence.\*;

import lombok.Data;

@Data

@Entity

@Table(name = "employees")

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(fetch = FetchType.LAZY)

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

private Department department;

}

EmployeeRepository.java

package com.example.EmployeeManagementSystem.repository;

import com.example.EmployeeManagementSystem.model.Employee;

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 and sorting

Page<Employee> findAll(Pageable pageable);

// Custom query with pagination and sorting to find employees by department name

Page<Employee> findByDepartment\_Name(String deptName, Pageable pageable);

}

EmployeeController.java

package com.example.EmployeeManagementSystem.controller;

import com.example.EmployeeManagementSystem.model.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.http.ResponseEntity;

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

import java.util.List;

@RestController

@RequestMapping("/api/employees")

public class EmployeeController {

private final EmployeeRepository empRepo;

@Autowired

public EmployeeController(EmployeeRepository empRepo) {

this.empRepo = empRepo;

}

@PostMapping

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

Employee savedEmployee = empRepo.save(employee);

return ResponseEntity.status(201).body(savedEmployee);

}

@GetMapping

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

List<Employee> employees = empRepo.findAll();

return ResponseEntity.ok(employees);

}

@GetMapping("/{id}")

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

return empRepo.findById(id)

.map(ResponseEntity::ok)

.orElseGet(() -> ResponseEntity.notFound().build());

}

@PutMapping("/{id}")

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

return empRepo.findById(id)

.map(existingEmployee -> {

existingEmployee.setName(updatedEmployee.getName());

existingEmployee.setEmail(updatedEmployee.getEmail());

existingEmployee.setDepartment(updatedEmployee.getDepartment());

empRepo.save(existingEmployee);

return ResponseEntity.ok(existingEmployee);

})

.orElseGet(() -> ResponseEntity.notFound().build());

}

@DeleteMapping("/{id}")

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

return empRepo.findById(id)

.map(employee -> {

empRepo.delete(employee);

return ResponseEntity.noContent().build();

})

.orElseGet(() -> ResponseEntity.notFound().build());

}

// Endpoint to get employees with pagination and sorting

@GetMapping("/paginated")

public ResponseEntity<Page<Employee>> getEmployeesPaginated(

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

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

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

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

Page<Employee> employeesPage = empRepo.findAll(pageable);

return ResponseEntity.ok(employeesPage);

}

// Endpoint to get employees by department name with pagination and sorting

@GetMapping("/search/by-department-paginated")

public ResponseEntity<Page<Employee>> searchByDeptNamePaginated(

@RequestParam String deptName,

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

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

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

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

Page<Employee> employeesPage = empRepo.findByDepartment\_Name(deptName, pageable);

return ResponseEntity.ok(employeesPage);

}

}

DepartmentRepository.java

package com.example.EmployeeManagementSystem.repository;

import com.example.EmployeeManagementSystem.model.Department;

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

import org.springframework.stereotype.Repository;

@Repository

public interface DepartmentRepository extends JpaRepository<Department, Long> {}

DepartmentController.java

package com.example.EmployeeManagementSystem.controller;

import com.example.EmployeeManagementSystem.model.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("/api/departments")

public class DepartmentController {

private final DepartmentRepository deptRepo;

@Autowired

public DepartmentController(DepartmentRepository deptRepo) {

this.deptRepo = deptRepo;

}

@PostMapping

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

Department savedDepartment = deptRepo.save(department);

return ResponseEntity.status(201).body(savedDepartment);

}

@GetMapping

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

List<Department> departments = deptRepo.findAll();

return ResponseEntity.ok(departments);

}

@GetMapping("/{id}")

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

return deptRepo.findById(id)

.map(ResponseEntity::ok)

.orElseGet(() -> ResponseEntity.notFound().build());

}

@PutMapping("/{id}")

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

return deptRepo.findById(id)

.map(existingDepartment -> {

existingDepartment.setName(updatedDepartment.getName());

deptRepo.save(existingDepartment);

return ResponseEntity.ok(existingDepartment);

})

.orElseGet(() -> ResponseEntity.notFound().build());

}

@DeleteMapping("/{id}")

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

return deptRepo.findById(id)

.map(department -> {

deptRepo.delete(department);

return ResponseEntity.noContent().build();

})

.orElseGet(() -> ResponseEntity.notFound().build());

}

}

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

pom.xml

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

ApplicationConfig.java

package com.example.EmployeeManagementSystem.config;

import org.springframework.context.annotation.Configuration;

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

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

@Configuration

@EnableJpaRepositories(basePackages = "com.example.EmployeeManagementSystem.repository")

@EnableJpaAuditing(auditorAwareRef = "auditorProvider")

public class ApplicationConfig {

}

AuditorAwareImpl.java

package com.example.EmployeeManagementSystem.config;

import org.springframework.data.domain.AuditorAware;

import org.springframework.stereotype.Component;

import java.util.Optional;

@Component

public class AuditorAwareImpl implements AuditorAware<String> {

@Override

public Optional<String> getCurrentAuditor() {

// Replace with actual user details fetching logic

return Optional.of("system");

}

}

Employee.java

package com.example.EmployeeManagementSystem.model;

import jakarta.persistence.\*;

import lombok.Data;

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

import java.time.LocalDateTime;

@Data

@Entity

@Table(name = "employees")

@EntityListeners(AuditingEntityListener.class)

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(fetch = FetchType.LAZY)

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

private Department department;

@CreatedDate

@Column(nullable = false, updatable = false)

private LocalDateTime createdDate;

@LastModifiedDate

private LocalDateTime lastModifiedDate;

@CreatedBy

@Column(nullable = false, updatable = false)

private String createdBy;

@LastModifiedBy

private String lastModifiedBy;

}

Department.java

package com.example.EmployeeManagementSystem.model;

import jakarta.persistence.\*;

import lombok.Data;

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

import java.time.LocalDateTime;

@Data

@Entity

@Table(name = "departments")

@EntityListeners(AuditingEntityListener.class)

public class Department {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@Column(nullable = false, unique = true)

private String name;

@CreatedDate

@Column(nullable = false, updatable = false)

private LocalDateTime createdDate;

@LastModifiedDate

private LocalDateTime lastModifiedDate;

@CreatedBy

@Column(nullable = false, updatable = false)

private String createdBy;

@LastModifiedBy

private String lastModifiedBy;

}

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

EmployeeProjection.java

package com.example.EmployeeManagementSystem.projection;

public interface EmployeeProjection {

Long getId();

String getName();

String getEmail();

String getDepartmentName(); // Custom method to get the department name

}

EmployeeDTO.java

package com.example.EmployeeManagementSystem.dto;

import lombok.AllArgsConstructor;

import lombok.Getter;

@Getter

@AllArgsConstructor

public class EmployeeDTO {

private Long id;

private String name;

private String email;

private String departmentName;

}

EmployeeRepository.java

package com.example.EmployeeManagementSystem.repository;

import com.example.EmployeeManagementSystem.model.Employee;

import com.example.EmployeeManagementSystem.projection.EmployeeProjection;

import com.example.EmployeeManagementSystem.dto.EmployeeDTO;

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> {

// Interface-based projection

@Query("SELECT e.id AS id, e.name AS name, e.email AS email, e.department.name AS departmentName FROM Employee e")

List<EmployeeProjection> findAllEmployeeProjections();

// Class-based projection using constructor expression

@Query("SELECT new com.example.EmployeeManagementSystem.dto.EmployeeDTO(e.id, e.name, e.email, e.department.name) FROM Employee e")

List<EmployeeDTO> findAllEmployeeDTOs();

}

EmployeeController.java

package com.example.EmployeeManagementSystem.controller;

import com.example.EmployeeManagementSystem.dto.EmployeeDTO;

import com.example.EmployeeManagementSystem.projection.EmployeeProjection;

import com.example.EmployeeManagementSystem.model.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("/api/employees")

public class EmployeeController {

private final EmployeeRepository empRepo;

@Autowired

public EmployeeController(EmployeeRepository empRepo) {

this.empRepo = empRepo;

}

// Existing methods...

// Endpoint to get employee projections

@GetMapping("/projections")

public ResponseEntity<List<EmployeeProjection>> getEmployeeProjections() {

List<EmployeeProjection> projections = empRepo.findAllEmployeeProjections();

return ResponseEntity.ok(projections);

}

// Endpoint to get employee DTOs

@GetMapping("/dtos")

public ResponseEntity<List<EmployeeDTO>> getEmployeeDTOs() {

List<EmployeeDTO> dtos = empRepo.findAllEmployeeDTOs();

return ResponseEntity.ok(dtos);

}

}

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

application.properties

# Primary Data Source Configuration

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

spring.datasource.primary.username=root

spring.datasource.primary.password=password

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

# Secondary Data Source Configuration

spring.datasource.secondary.url=jdbc:mysql://localhost:3306/secondarydb

spring.datasource.secondary.username=root

spring.datasource.secondary.password=password

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

PrimaryDataSourceConfig.java

package com.example.EmployeeManagementSystem.config;

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

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

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

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

import org.springframework.orm.jpa.JpaTransactionManager;

import org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean;

import org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter;

import javax.sql.DataSource;

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

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

import org.springframework.transaction.PlatformTransactionManager;

@Configuration

@EnableConfigurationProperties

public class PrimaryDataSourceConfig {

@Bean

@ConfigurationProperties("spring.datasource.primary")

public DataSourceProperties primaryDataSourceProperties() {

return new DataSourceProperties();

}

@Bean

@ConfigurationProperties("spring.datasource.primary")

public DataSource primaryDataSource() {

return primaryDataSourceProperties().initializeDataSourceBuilder().build();

}

@Bean

public LocalContainerEntityManagerFactoryBean primaryEntityManagerFactory(

@Qualifier("primaryDataSource") DataSource dataSource) {

LocalContainerEntityManagerFactoryBean em = new LocalContainerEntityManagerFactoryBean();

em.setDataSource(dataSource);

em.setPackagesToScan("com.example.EmployeeManagementSystem.model");

em.setJpaVendorAdapter(new HibernateJpaVendorAdapter());

return em;

}

@Bean

public PlatformTransactionManager primaryTransactionManager(

@Qualifier("primaryEntityManagerFactory") LocalContainerEntityManagerFactoryBean primaryEntityManagerFactory) {

return new JpaTransactionManager(primaryEntityManagerFactory.getObject());

}

}

SecondaryDataSourceConfig.java

package com.example.EmployeeManagementSystem.config;

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

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

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

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

import org.springframework.orm.jpa.JpaTransactionManager;

import org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean;

import org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter;

import javax.sql.DataSource;

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

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

import org.springframework.transaction.PlatformTransactionManager;

@Configuration

@EnableConfigurationProperties

public class SecondaryDataSourceConfig {

@Bean

@ConfigurationProperties("spring.datasource.secondary")

public DataSourceProperties secondaryDataSourceProperties() {

return new DataSourceProperties();

}

@Bean

@ConfigurationProperties("spring.datasource.secondary")

public DataSource secondaryDataSource() {

return secondaryDataSourceProperties().initializeDataSourceBuilder().build();

}

@Bean

public LocalContainerEntityManagerFactoryBean secondaryEntityManagerFactory(

@Qualifier("secondaryDataSource") DataSource dataSource) {

LocalContainerEntityManagerFactoryBean em = new LocalContainerEntityManagerFactoryBean();

em.setDataSource(dataSource);

em.setPackagesToScan("com.example.EmployeeManagementSystem.model");

em.setJpaVendorAdapter(new HibernateJpaVendorAdapter());

return em;

}

@Bean

public PlatformTransactionManager secondaryTransactionManager(

@Qualifier("secondaryEntityManagerFactory") LocalContainerEntityManagerFactoryBean secondaryEntityManagerFactory) {

return new JpaTransactionManager(secondaryEntityManagerFactory.getObject());

}

}

PrimaryEmployeeRepository.java

package com.example.EmployeeManagementSystem.repository.primary;

import com.example.EmployeeManagementSystem.model.Employee;

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

import org.springframework.stereotype.Repository;

@Repository

public interface PrimaryEmployeeRepository extends JpaRepository<Employee, Long> {

}

SecondaryDepartmentRepository.java

package com.example.EmployeeManagementSystem.repository.secondary;

import com.example.EmployeeManagementSystem.model.Department;

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

import org.springframework.stereotype.Repository;

@Repository

public interface SecondaryDepartmentRepository extends JpaRepository<Department, Long> {

}

EmployeeService.java

package com.example.EmployeeManagementSystem.service;

import com.example.EmployeeManagementSystem.model.Employee;

import com.example.EmployeeManagementSystem.repository.primary.PrimaryEmployeeRepository;

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

import org.springframework.stereotype.Service;

import java.util.List;

@Service

public class EmployeeService {

private final PrimaryEmployeeRepository primaryEmployeeRepository;

@Autowired

public EmployeeService(PrimaryEmployeeRepository primaryEmployeeRepository) {

this.primaryEmployeeRepository = primaryEmployeeRepository;

}

public List<Employee> findAllEmployees() {

return primaryEmployeeRepository.findAll();

}

public Employee saveEmployee(Employee employee) {

return primaryEmployeeRepository.save(employee);

}

// Additional methods as needed

}

DepartmentService.java

package com.example.EmployeeManagementSystem.service;

import com.example.EmployeeManagementSystem.model.Department;

import com.example.EmployeeManagementSystem.repository.secondary.SecondaryDepartmentRepository;

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

import org.springframework.stereotype.Service;

import java.util.List;

@Service

public class DepartmentService {

private final SecondaryDepartmentRepository secondaryDepartmentRepository;

@Autowired

public DepartmentService(SecondaryDepartmentRepository secondaryDepartmentRepository) {

this.secondaryDepartmentRepository = secondaryDepartmentRepository;

}

public List<Department> findAllDepartments() {

return secondaryDepartmentRepository.findAll();

}

public Department saveDepartment(Department department) {

return secondaryDepartmentRepository.save(department);

}

// Additional methods as needed

}

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

Employee.java

package com.example.EmployeeManagementSystem.model;

import jakarta.persistence.\*;

import lombok.Data;

import org.hibernate.annotations.BatchSize;

import org.hibernate.annotations.DynamicInsert;

import org.hibernate.annotations.DynamicUpdate;

import org.hibernate.annotations.Type;

@Data

@Entity

@Table(name = "employees")

@DynamicInsert

@DynamicUpdate

@BatchSize(size = 50) // Batch size for fetching collections

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(fetch = FetchType.LAZY)

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

private Department department;

// Example of using @Type

@Type(type = "org.hibernate.type.StringClobType")

@Column(columnDefinition = "TEXT")

private String description;

}

application.properties

# Hibernate Dialect

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

# Hibernate Show SQL

spring.jpa.show-sql=true

# Hibernate Format SQL

spring.jpa.properties.hibernate.format\_sql=true

# Hibernate DDL Auto

spring.jpa.hibernate.ddl-auto=update

# Batch Size Configuration

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

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

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

package com.example.EmployeeManagementSystem.service;

import com.example.EmployeeManagementSystem.model.Employee;

import com.example.EmployeeManagementSystem.repository.PrimaryEmployeeRepository;

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

import org.springframework.stereotype.Service;

import org.springframework.transaction.annotation.Transactional;

import java.util.List;

@Service

public class EmployeeBatchService {

private final PrimaryEmployeeRepository primaryEmployeeRepository;

@Autowired

public EmployeeBatchService(PrimaryEmployeeRepository primaryEmployeeRepository) {

this.primaryEmployeeRepository = primaryEmployeeRepository;

}

@Transactional

public void saveEmployeesInBatch(List<Employee> employees) {

int batchSize = 50;

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

primaryEmployeeRepository.save(employees.get(i));

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

// Flush and clear to manage memory

primaryEmployeeRepository.flush();

primaryEmployeeRepository.clear();

}

}

}

}

PrimaryEmployeeRepository.java

package com.example.EmployeeManagementSystem.repository.primary;

import com.example.EmployeeManagementSystem.model.Employee;

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

import org.springframework.stereotype.Repository;

@Repository

public interface PrimaryEmployeeRepository extends JpaRepository<Employee, Long> {

// Define custom methods if needed

}