**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.

**Configuring Application Properties:**

Application.properties file:

# H2 Database 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

# H2 Console (Optional)

spring.h2.console.enabled=true

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

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

**Business Scenario:**

Define JPA entities for Employee and Department with appropriate relationships.

1. **Creating JPA Entities:**

package com.yourcompany.employeemanagementsystem.entity;

import jakarta.persistence.\*;

import lombok.Data;

import lombok.NoArgsConstructor;

import lombok.AllArgsConstructor;

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, unique = true)

private String name;

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

private List<Employee> employees;

}

package com.yourcompany.employeemanagementsystem.entity;

import jakarta.persistence.\*;

import lombok.Data;

import lombok.NoArgsConstructor;

import lombok.AllArgsConstructor;

@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;

}

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

**Business Scenario:**

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

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

**Spring Data Repositories** simplify data access layers by providing a consistent API for CRUD operations without the need to write custom queries or boilerplate code. Some key benefits include:

* **Automatic Implementation**: When you extend Spring Data JPA's JpaRepository, Spring automatically generates the implementation at runtime.
* **CRUD Operations**: Out-of-the-box support for standard CRUD operations (save, findAll, findById, delete, etc.).
* **Custom Queries**: Support for derived query methods and custom JPQL/SQL queries.
* **Pagination and Sorting**: Built-in support for paginated and sorted queries.

1. **Creating Repositories:**

package com.yourcompany.employeemanagementsystem.repository;

import com.yourcompany.employeemanagementsystem.entity.Employee;

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

import java.util.List;

public interface EmployeeRepository extends JpaRepository<Employee, Long> {

// Derived query method to find employees by department name

List<Employee> findByDepartmentName(String departmentName);

// Derived query method to find employees by email

Employee findByEmail(String email);

}

package com.yourcompany.employeemanagementsystem.repository;

import com.yourcompany.employeemanagementsystem.entity.Department;

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

public interface DepartmentRepository extends JpaRepository<Department, Long> {

// Derived query method to find a department by name

Department findByName(String name);

}

EXAMPLE USAGE:

package com.yourcompany.employeemanagementsystem.service;

import com.yourcompany.employeemanagementsystem.entity.Employee;

import com.yourcompany.employeemanagementsystem.repository.EmployeeRepository;

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

import org.springframework.stereotype.Service;

import java.util.List;

@Service

public class EmployeeService {

@Autowired

private EmployeeRepository employeeRepository;

public List<Employee> getEmployeesByDepartment(String departmentName) {

return employeeRepository.findByDepartmentName(departmentName);

}

public Employee getEmployeeByEmail(String email) {

return employeeRepository.findByEmail(email);

}

public Employee saveEmployee(Employee employee) {

return employeeRepository.save(employee);

}

public void deleteEmployee(Long id) {

employeeRepository.deleteById(id);

}

}

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

**Business Scenario:**

Implement CRUD operations for managing employees and departments.

1. **Basic CRUD Operations:**

package com.yourcompany.employeemanagementsystem.service;

import com.yourcompany.employeemanagementsystem.entity.Employee;

import com.yourcompany.employeemanagementsystem.repository.EmployeeRepository;

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 or Update an Employee

public Employee saveEmployee(Employee employee) {

return employeeRepository.save(employee);

}

// Read all Employees

public List<Employee> getAllEmployees() {

return employeeRepository.findAll();

}

// Read an Employee by ID

public Optional<Employee> getEmployeeById(Long id) {

return employeeRepository.findById(id);

}

// Delete an Employee by ID

public void deleteEmployee(Long id) {

employeeRepository.deleteById(id);

}

}

DEPARTMENT SERVICE:

package com.yourcompany.employeemanagementsystem.service;

import com.yourcompany.employeemanagementsystem.entity.Department;

import com.yourcompany.employeemanagementsystem.repository.DepartmentRepository;

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 or Update a Department

public Department saveDepartment(Department department) {

return departmentRepository.save(department);

}

// Read all Departments

public List<Department> getAllDepartments() {

return departmentRepository.findAll();

}

// Read a Department by ID

public Optional<Department> getDepartmentById(Long id) {

return departmentRepository.findById(id);

}

// Delete a Department by ID

public void deleteDepartment(Long id) {

departmentRepository.deleteById(id);

}

}

**EmployeeController**

package com.yourcompany.employeemanagementsystem.controller;

import com.yourcompany.employeemanagementsystem.entity.Employee;

import com.yourcompany.employeemanagementsystem.service.EmployeeService;

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

import org.springframework.http.ResponseEntity;

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

import java.util.List;

import java.util.Optional;

@RestController

@RequestMapping("/api/employees")

public class EmployeeController {

@Autowired

private EmployeeService employeeService;

// Create or Update an Employee

@PostMapping

public Employee createOrUpdateEmployee(@RequestBody Employee employee) {

return employeeService.saveEmployee(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) {

Optional<Employee> employee = employeeService.getEmployeeById(id);

return employee.map(ResponseEntity::ok).orElseGet(() -> ResponseEntity.notFound().build());

}

// Delete an Employee by ID

@DeleteMapping("/{id}")

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

employeeService.deleteEmployee(id);

return ResponseEntity.noContent().build();

}

}

**DepartmentController**

package com.yourcompany.employeemanagementsystem.controller;

import com.yourcompany.employeemanagementsystem.entity.Department;

import com.yourcompany.employeemanagementsystem.service.DepartmentService;

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

import org.springframework.http.ResponseEntity;

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

import java.util.List;

import java.util.Optional;

@RestController

@RequestMapping("/api/departments")

public class DepartmentController {

@Autowired

private DepartmentService departmentService;

// Create or Update a Department

@PostMapping

public Department createOrUpdateDepartment(@RequestBody Department department) {

return departmentService.saveDepartment(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) {

Optional<Department> department = departmentService.getDepartmentById(id);

return department.map(ResponseEntity::ok).orElseGet(() -> ResponseEntity.notFound().build());

}

// Delete a Department by ID

@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.

1. **Defining Query Methods:**

package com.yourcompany.employeemanagementsystem.repository;

import com.yourcompany.employeemanagementsystem.entity.Employee;

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

import java.util.List;

public interface EmployeeRepository extends JpaRepository<Employee, Long> {

// Find employees by their 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 department name and sort them by name

List<Employee> findByDepartmentNameOrderByNameAsc(String departmentName);

}

package com.yourcompany.employeemanagementsystem.repository;

import com.yourcompany.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> {

// Custom JPQL query to find employees by part of their name

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

List<Employee> findEmployeesByNameContains(@Param("namePart") String namePart);

// Custom native query to find employees by department name using native SQL

@Query(value = "SELECT \* FROM employees e INNER JOIN departments d ON e.department\_id = d.id WHERE d.name = :departmentName", nativeQuery = true)

List<Employee> findEmployeesByDepartmentNameNative(@Param("departmentName") String departmentName);

}

1. **Named Queries:**

package com.yourcompany.employeemanagementsystem.entity;

import jakarta.persistence.\*;

import java.util.List;

@Entity

@Table(name = "employees")

@NamedQueries({

@NamedQuery(name = "Employee.findByDepartmentId", query = "SELECT e FROM Employee e WHERE e.department.id = :departmentId"),

@NamedQuery(name = "Employee.findByEmailDomain", query = "SELECT e FROM Employee e WHERE e.email LIKE :domain")

})

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

}

package com.yourcompany.employeemanagementsystem.repository;

import com.yourcompany.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> {

// Execute the named query "Employee.findByDepartmentId"

@Query(name = "Employee.findByDepartmentId")

List<Employee> findEmployeesByDepartmentId(@Param("departmentId") Long departmentId);

// Execute the named query "Employee.findByEmailDomain"

@Query(name = "Employee.findByEmailDomain")

List<Employee> findEmployeesByEmailDomain(@Param("domain") String domain);

}

EXAMPLE USAGE:

package com.yourcompany.employeemanagementsystem.service;

import com.yourcompany.employeemanagementsystem.entity.Employee;

import com.yourcompany.employeemanagementsystem.repository.EmployeeRepository;

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

import org.springframework.stereotype.Service;

import java.util.List;

@Service

public class EmployeeService {

@Autowired

private EmployeeRepository employeeRepository;

public List<Employee> findEmployeesByDepartmentName(String departmentName) {

return employeeRepository.findByDepartmentName(departmentName);

}

public List<Employee> searchEmployeesByNamePart(String namePart) {

return employeeRepository.findEmployeesByNameContains(namePart);

}

public List<Employee> findEmployeesByEmailDomain(String domain) {

return employeeRepository.findEmployeesByEmailDomain(domain);

}

}

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

**Business Scenario:**

Add pagination and sorting capabilities to your employee search functionality.

1. **Pagination:**

package com.yourcompany.employeemanagementsystem.repository;

import com.yourcompany.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> {

// Find all employees with pagination

Page<Employee> findAll(Pageable pageable);

// Find employees by department name with pagination

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

}

**Implementation:**

package com.yourcompany.employeemanagementsystem.service;

import com.yourcompany.employeemanagementsystem.entity.Employee;

import com.yourcompany.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.stereotype.Service;

@Service

public class EmployeeService {

@Autowired

private EmployeeRepository employeeRepository;

// Get paginated list of employees

public Page<Employee> getPaginatedEmployees(int page, int size) {

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

return employeeRepository.findAll(pageable);

}

// Get paginated list of employees by department name

public Page<Employee> getPaginatedEmployeesByDepartment(String departmentName, int page, int size) {

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

return employeeRepository.findByDepartmentName(departmentName, pageable);

}

}

1. **Sorting:**

package com.yourcompany.employeemanagementsystem.repository;

import com.yourcompany.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 java.util.List;

public interface EmployeeRepository extends JpaRepository<Employee, Long> {

// Find all employees with pagination and sorting

Page<Employee> findAll(Pageable pageable);

// Find employees by department name with pagination and sorting

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

// Find employees and sort them by name

List<Employee> findByNameContaining(String name, Sort sort);

}

package com.yourcompany.employeemanagementsystem.service;

import com.yourcompany.employeemanagementsystem.entity.Employee;

import com.yourcompany.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.stereotype.Service;

@Service

public class EmployeeService {

@Autowired

private EmployeeRepository employeeRepository;

// Get paginated and sorted list of employees

public Page<Employee> getPaginatedAndSortedEmployees(int page, int size, String sortBy, boolean ascending) {

Sort sort = ascending ? Sort.by(sortBy).ascending() : Sort.by(sortBy).descending();

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

return employeeRepository.findAll(pageable);

}

// Get paginated and sorted list of employees by department name

public Page<Employee> getPaginatedAndSortedEmployeesByDepartment(String departmentName, int page, int size, String sortBy, boolean ascending) {

Sort sort = ascending ? Sort.by(sortBy).ascending() : Sort.by(sortBy).descending();

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

return employeeRepository.findByDepartmentName(departmentName, pageable);

}

}

EmployeeController:

package com.yourcompany.employeemanagementsystem.controller;

import com.yourcompany.employeemanagementsystem.entity.Employee;

import com.yourcompany.employeemanagementsystem.service.EmployeeService;

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

import org.springframework.data.domain.Page;

import org.springframework.http.ResponseEntity;

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

@RestController

@RequestMapping("/api/employees")

public class EmployeeController {

@Autowired

private EmployeeService employeeService;

// Get paginated and sorted list of employees

@GetMapping

public Page<Employee> getPaginatedAndSortedEmployees(

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

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

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

@RequestParam(defaultValue = "true") boolean ascending) {

return employeeService.getPaginatedAndSortedEmployees(page, size, sortBy, ascending);

}

// Get paginated and sorted list of employees by department name

@GetMapping("/department")

public Page<Employee> getPaginatedAndSortedEmployeesByDepartment(

@RequestParam String departmentName,

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

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

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

@RequestParam(defaultValue = "true") boolean ascending) {

return employeeService.getPaginatedAndSortedEmployeesByDepartment(departmentName, page, size, sortBy, ascending);

}

}

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

**Business Scenario:**

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

* + 1. **Entity Auditing:**

package com.yourcompany.employeemanagementsystem;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

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

@SpringBootApplication

@EnableJpaAuditing // Enabling JPA Auditing

public class EmployeeManagementSystemApplication {

public static void main(String[] args) {

SpringApplication.run(EmployeeManagementSystemApplication.class, args);

}

}

Employee Entity with Auditing:

package com.yourcompany.employeemanagementsystem.entity;

import org.springframework.data.annotation.CreatedDate;

import org.springframework.data.annotation.LastModifiedDate;

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

import jakarta.persistence.\*;

import java.time.LocalDateTime;

@Entity

@Table(name = "employees")

@EntityListeners(AuditingEntityListener.class) // Enable auditing for this entity

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;

@CreatedDate

@Column(nullable = false, updatable = false)

private LocalDateTime createdDate;

@LastModifiedDate

@Column(nullable = false)

private LocalDateTime lastModifiedDate;

// Other fields, getters, setters, etc.

}

Department Entity with Auditing:

package com.yourcompany.employeemanagementsystem.entity;

import org.springframework.data.annotation.CreatedDate;

import org.springframework.data.annotation.LastModifiedDate;

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

import jakarta.persistence.\*;

import java.time.LocalDateTime;

@Entity

@Table(name = "departments")

@EntityListeners(AuditingEntityListener.class) // Enable auditing for this entity

public class Department {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@Column(nullable = false)

private String name;

@CreatedDate

@Column(nullable = false, updatable = false)

private LocalDateTime createdDate;

@LastModifiedDate

@Column(nullable = false)

private LocalDateTime lastModifiedDate;

// Other fields, getters, setters, etc.

}

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

**Business Scenario:**

Customize your data source configuration and manage multiple data sources.

1. **Spring Boot Auto-Configuration:**

package com.yourcompany.employeemanagementsystem.projection;

public interface EmployeeProjection {

String getName();

String getEmail();

}

package com.yourcompany.employeemanagementsystem.repository;

import com.yourcompany.employeemanagementsystem.entity.Employee;

import com.yourcompany.employeemanagementsystem.projection.EmployeeProjection;

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> {

// Fetch data using projection

@Query("SELECT e FROM Employee e")

List<EmployeeProjection> findAllEmployeeProjections();

}

DTO CLASS

package com.yourcompany.employeemanagementsystem.dto;

public class EmployeeDTO {

private String name;

private String email;

// Constructor

public EmployeeDTO(String name, String email) {

this.name = name;

this.email = email;

}

// Getters and Setters

}

package com.yourcompany.employeemanagementsystem.repository;

import com.yourcompany.employeemanagementsystem.dto.EmployeeDTO;

import com.yourcompany.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> {

// Fetch data using DTO class

@Query("SELECT new com.yourcompany.employeemanagementsystem.dto.EmployeeDTO(e.name, e.email) FROM Employee e")

List<EmployeeDTO> findAllEmployeeDTOs();

}

Combining Projections in Queries:

package com.yourcompany.employeemanagementsystem.projection;

public interface DepartmentProjection {

String getName();

int getEmployeeCount();

}

package com.yourcompany.employeemanagementsystem.repository;

import com.yourcompany.employeemanagementsystem.projection.DepartmentProjection;

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> {

// Fetch department data with projection

@Query("SELECT new com.yourcompany.employeemanagementsystem.dto.DepartmentDTO(d.name, COUNT(e.id)) FROM Department d LEFT JOIN d.employees e GROUP BY d.name")

List<DepartmentProjection> findDepartmentProjections();

}

EXAMPLE USAGE:

package com.yourcompany.employeemanagementsystem.service;

import com.yourcompany.employeemanagementsystem.dto.EmployeeDTO;

import com.yourcompany.employeemanagementsystem.projection.EmployeeProjection;

import com.yourcompany.employeemanagementsystem.repository.EmployeeRepository;

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

import org.springframework.stereotype.Service;

import java.util.List;

@Service

public class EmployeeService {

@Autowired

private EmployeeRepository employeeRepository;

// Fetch employee projections

public List<EmployeeProjection> getEmployeeProjections() {

return employeeRepository.findEmployeeFullNameWithEmail();

}

// Fetch employee DTOs

public List<EmployeeDTO> getEmployeeDTOs() {

return employeeRepository.findAllEmployeeDTOs();

}

}

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

**Business Scenario:**

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

1. **Hibernate-Specific Annotations:**

package com.yourcompany.employeemanagementsystem.entity;

import org.hibernate.annotations.BatchSize;

import org.hibernate.annotations.DynamicInsert;

import org.hibernate.annotations.DynamicUpdate;

import jakarta.persistence.\*;

import java.time.LocalDateTime;

@Entity

@Table(name = "employees")

@DynamicInsert

@DynamicUpdate

@BatchSize(size = 10)

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;

@CreatedDate

@Column(nullable = false, updatable = false)

private LocalDateTime createdDate;

@LastModifiedDate

@Column(nullable = false)

private LocalDateTime lastModifiedDate;

// Other fields, getters, setters, etc.

}

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

# Hibernate Dialect

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.H2Dialect

# Enable Hibernate SQL logging

spring.jpa.show-sql=true

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

# Other Hibernate properties for performance tuning

spring.jpa.properties.hibernate.default\_batch\_fetch\_size=16

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:**

package com.yourcompany.employeemanagementsystem.repository;

import com.yourcompany.employeemanagementsystem.entity.Employee;

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

import org.springframework.stereotype.Repository;

@Repository

public interface EmployeeRepository extends JpaRepository<Employee, Long> {

// Batch processing can be implemented here if needed

}

Service Method for Batch Processing:

package com.yourcompany.employeemanagementsystem.service;

import com.yourcompany.employeemanagementsystem.entity.Employee;

import com.yourcompany.employeemanagementsystem.repository.EmployeeRepository;

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 EmployeeService {

@Autowired

private EmployeeRepository employeeRepository;

@Transactional

public void batchSaveEmployees(List<Employee> employees) {

int batchSize = 20; // Configure your batch size

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

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

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

// Flush and clear the persistence context

employeeRepository.flush();

employeeRepository.clear();

}

}

}

}