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

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

**2. Configuring Application Properties:**

In application.properties

# H2 Database Configuration

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

spring.datasource.driverClassName=org.h2.Driver

spring.datasource.username=sa

spring.datasource.password=password

# Hibernate Dialect for H2

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

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

**1. Creating JPA Entities:**

**Creating a new class named Employee under a package like com.example.employee management system.entity**

package com.example.employeemanagementsystem.entity;

import javax.persistence.\*;

import lombok.Data;

import lombok.NoArgsConstructor;

import lombok.AllArgsConstructor;

@Data

@NoArgsConstructor

@AllArgsConstructor

@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;

// Define the many-to-one relationship with Department

@ManyToOne(fetch = FetchType.LAZY)

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

private Department department;

}

**Creating a Department class**

package com.example.employeemanagementsystem.entity;

import javax.persistence.\*;

import lombok.Data;

import lombok.NoArgsConstructor;

import lombok.AllArgsConstructor;

import java.util.List;

@Data

@NoArgsConstructor

@AllArgsConstructor

@Entity

@Table(name = "departments")

public class Department {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@Column(nullable = false, unique = true)

private String name;

// Define the one-to-many relationship with Employee

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

private List<Employee> employees;

}

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

**Creating the EmployeeRepository Interface**

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> {

// Derived query method to find employees by department

List<Employee> findByDepartmentId(Long departmentId);

// Derived query method to find employees by name

List<Employee> findByNameContainingIgnoreCase(String name);

// Derived query method to find an employee by email

Employee findByEmail(String email);

}

**Create the DepartmentRepository Interface**

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> {

// Derived query method to find a department by name

Department findByName(String name);

}

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

**Creating a class EmployeeController in the controller package**

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("/api/employees")

public class EmployeeController {

@Autowired

private EmployeeRepository employeeRepository;

// Create a new employee

@PostMapping

public Employee createEmployee(@RequestBody Employee employee) {

return employeeRepository.save(employee);

}

// Get all employees

@GetMapping

public List<Employee> getAllEmployees() {

return employeeRepository.findAll();

}

// Get an employee by ID

@GetMapping("/{id}")

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

return employeeRepository.findById(id)

.map(ResponseEntity::ok)

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

}

// Update an existing employee

@PutMapping("/{id}")

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

return employeeRepository.findById(id)

.map(employee -> {

employee.setName(updatedEmployee.getName());

employee.setEmail(updatedEmployee.getEmail());

employee.setDepartment(updatedEmployee.getDepartment());

Employee savedEmployee = employeeRepository.save(employee);

return ResponseEntity.ok(savedEmployee);

})

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

}

// Delete an employee

@DeleteMapping("/{id}")

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

return employeeRepository.findById(id)

.map(employee -> {

employeeRepository.delete(employee);

return ResponseEntity.noContent().build();

})

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

}

}

**Similarly, create a 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("/api/departments")

public class DepartmentController {

@Autowired

private DepartmentRepository departmentRepository;

// Create a new department

@PostMapping

public Department createDepartment(@RequestBody Department department) {

return departmentRepository.save(department);

}

// Get all departments

@GetMapping

public List<Department> getAllDepartments() {

return departmentRepository.findAll();

}

// Get a department by ID

@GetMapping("/{id}")

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

return departmentRepository.findById(id)

.map(ResponseEntity::ok)

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

}

// Update an existing department

@PutMapping("/{id}")

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

return departmentRepository.findById(id)

.map(department -> {

department.setName(updatedDepartment.getName());

Department savedDepartment = departmentRepository.save(department);

return ResponseEntity.ok(savedDepartment);

})

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

}

// Delete a department

@DeleteMapping("/{id}")

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

return departmentRepository.findById(id)

.map(department -> {

departmentRepository.delete(department);

return ResponseEntity.noContent().build();

})

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

}

}

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

**Step1: Defining Query Methods**

**Using Keywords in Method Names**

package com.example.employeemanagementsystem.repository;

import com.example.employeemanagementsystem.entity.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> {

// Find employees by department name

List<Employee> findByDepartmentName(String departmentName);

// Find employees by email domain (e.g., all employees with emails from "example.com")

List<Employee> findByEmailEndingWith(String domain);

// Find employees whose name starts with a specific letter

List<Employee> findByNameStartingWith(String prefix);

}

**Step 2: Implementing Custom Query Methods with @Query**

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 org.springframework.stereotype.Repository;

import java.util.List;

@Repository

public interface EmployeeRepository extends JpaRepository<Employee, Long> {

// Custom query to find employees by department name

@Query("SELECT e FROM Employee e WHERE e.department.name = :departmentName")

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

// Custom query to find employees by email domain

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

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

}

**Defining Named Queries**

package com.example.employeemanagementsystem.entity;

import javax.persistence.\*;

import lombok.Data;

import lombok.NoArgsConstructor;

import lombok.AllArgsConstructor;

import java.util.List;

@Data

@NoArgsConstructor

@AllArgsConstructor

@Entity

@Table(name = "employees")

@NamedQueries({

@NamedQuery(

name = "Employee.findByDepartmentNameNamedQuery",

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

),

@NamedQuery(

name = "Employee.findByEmailDomainNamedQuery",

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

)

})

public class Employee {

}

**Executing Named Queries**

package com.example.employeemanagementsystem.repository;

import com.example.employeemanagementsystem.entity.Employee;

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

import org.springframework.data.repository.query.Param;

import org.springframework.stereotype.Repository;

import java.util.List;

@Repository

public interface EmployeeRepository extends JpaRepository<Employee, Long> {

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

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

}

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

**Step 1: Add Pagination to EmployeeRepository**

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.stereotype.Repository;

@Repository

public interface EmployeeRepository extends JpaRepository<Employee, Long> {

Page<Employee> findAll(Pageable pageable);

}

**Step 2: Modify EmployeeController to Include Pagination**

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.\*;

@RestController

@RequestMapping("/api/employees")

public class EmployeeController {

@Autowired

private EmployeeRepository employeeRepository;

@GetMapping

public Page<Employee> getAllEmployees(

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

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

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

return employeeRepository.findAll(pageable);

}

}

**Step 3: Add Sorting to the Repository**

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.stereotype.Repository;

import java.util.List;

@Repository

public interface EmployeeRepository extends JpaRepository<Employee, Long> {

Page<Employee> findAll(Pageable pageable);

List<Employee> findAll(Sort sort);

}

**Step 4: Modify EmployeeController to Include Sorting**

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.\*;

@RestController

@RequestMapping("/api/employees")

public class EmployeeController {

@Autowired

private EmployeeRepository employeeRepository;

@GetMapping

public Page<Employee> getAllEmployees(

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

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

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

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

return employeeRepository.findAll(pageable);

}

}

**Step:3 Combining Pagination and Sorting**

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.\*;

@RestController

@RequestMapping("/api/employees")

public class EmployeeController {

@Autowired

private EmployeeRepository employeeRepository;

@GetMapping

public Page<Employee> getAllEmployees(

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

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

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

@RequestParam(defaultValue = "asc") String direction) {

Sort sort = direction.equalsIgnoreCase("asc") ? Sort.by(sortBy).ascending() : Sort.by(sortBy).descending();

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

return employeeRepository.findAll(pageable);

}

}

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

**Enabling JPA Auditing**

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);

}

}

**Audit Configuration:**

package com.example.employeemanagementsystem.config;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.data.domain.AuditorAware;

import java.util.Optional;

@Configuration

public class AuditConfig {

@Bean

public AuditorAware<String> auditorProvider() {

return () -> Optional.of("system"); // Replace "system" with actual user info from your security context

}

}

**Modify the Employee and Department Entities**

**Employee entity:**

package com.example.employeemanagementsystem.entity;

import lombok.Data;

import lombok.NoArgsConstructor;

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 javax.persistence.\*;

import java.time.LocalDateTime;

@Data

@NoArgsConstructor

@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;

@CreatedBy

@Column(updatable = false)

private String createdBy;

@LastModifiedBy

private String lastModifiedBy;

@CreatedDate

@Column(updatable = false)

private LocalDateTime createdDate;

@LastModifiedDate

private LocalDateTime lastModifiedDate;

}

**Department Entity:**

package com.example.employeemanagementsystem.entity;

import lombok.Data;

import lombok.NoArgsConstructor;

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 javax.persistence.\*;

import java.time.LocalDateTime;

@Data

@NoArgsConstructor

@Entity

@Table(name = "departments")

@EntityListeners(AuditingEntityListener.class)

public class Department {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

@CreatedBy

@Column(updatable = false)

private String createdBy;

@LastModifiedBy

private String lastModifiedBy;

@CreatedDate

@Column(updatable = false)

private LocalDateTime createdDate;

@LastModifiedDate

private LocalDateTime lastModifiedDate;

}

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

**Step 1: Define Interface-Based Projections**

**Employee Projection:**

package com.example.employeemanagementsystem.projection;

public interface EmployeeProjection {

Long getId();

String getName();

String getEmail();

String getDepartmentName();

}

**Department Projection:**

package com.example.employeemanagementsystem.projection;

public interface DepartmentProjection {

Long getId();

String getName();

}

**Step 2: Define Class-Based Projections**

Employee DTO:

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;

}

**Department DTO:**

package com.example.employeemanagementsystem.dto;

import lombok.AllArgsConstructor;

import lombok.Getter;

@Getter

@AllArgsConstructor

public class DepartmentDTO {

private Long id;

private String name;

}

**Step 3: Use @Value and Constructor Expressions for Projections**

**Employee Projection with @Value:**

package com.example.employeemanagementsystem.repository;

import com.example.employeemanagementsystem.entity.Employee;

import com.example.employeemanagementsystem.projection.EmployeeProjection;

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

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

import org.springframework.data.repository.query.Param;

import org.springframework.stereotype.Repository;

import java.util.List;

@Repository

public interface EmployeeRepository extends JpaRepository<Employee, Long> {

@Query("SELECT e.id AS id, e.name AS name, e.email AS email, e.department.name AS departmentName " +

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

List<EmployeeProjection> findEmployeesByDepartmentName(@Param("departmentName") String departmentName);

}

**Class-Based Projections with Constructor Expressions:**

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 org.springframework.data.repository.query.Param;

import org.springframework.stereotype.Repository;

import java.util.List;

@Repository

public interface EmployeeRepository extends JpaRepository<Employee, Long> {

@Query("SELECT new com.example.employeemanagementsystem.dto.EmployeeDTO(e.id, e.name, e.email, e.department.name) " +

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

List<EmployeeDTO> findEmployeesByDepartmentName(@Param("departmentName") String departmentName);

}

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

**Step 1: Basic Configuration with application.properties**

# application.properties

# Primary Data Source (H2 Database)

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

**Step 2: Customize Data Source Configuration**

package com.example.employeemanagementsystem.config;

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 {

@Bean

public DataSource dataSource() {

DriverManagerDataSource dataSource = new DriverManagerDataSource();

dataSource.setDriverClassName("org.h2.Driver");

dataSource.setUrl("jdbc:h2:mem:testdb");

dataSource.setUsername("sa");

dataSource.setPassword("password");

return dataSource;

}

}

**Step 3: Externalize Configuration in application.properties**

**Primary Data Source Configuration:**

# Primary Data Source (H2 Database)

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

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

spring.datasource.primary.username=sa

spring.datasource.primary.password=password

**Secondary Data Source Configuration:**

# Secondary Data Source (MySQL)

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

spring.datasource.secondary.driverClassName=com.mysql.cj.jdbc.Driver

spring.datasource.secondary.username=root

spring.datasource.secondary.password=password

**Step 4: Configure Multiple Data Sources**

package com.example.employeemanagementsystem.config;

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

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

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

import org.springframework.jdbc.datasource.DriverManagerDataSource;

import javax.sql.DataSource;

@Configuration

public class DataSourceConfig {

@Bean(name = "primaryDataSource")

@ConfigurationProperties(prefix = "spring.datasource.primary")

public DataSource primaryDataSource() {

return new DriverManagerDataSource();

}

@Bean(name = "secondaryDataSource")

@ConfigurationProperties(prefix = "spring.datasource.secondary")

public DataSource secondaryDataSource() {

return new DriverManagerDataSource();

}

}

**Step 5: Create Data Source Configuration Classes**

**Primary Data Source Configuration:**

package com.example.employeemanagementsystem.config;

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

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.orm.jpa.JpaTransactionManager;

import org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean;

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

import javax.persistence.EntityManagerFactory;

import javax.sql.DataSource;

@Configuration

public class PrimaryDataSourceConfig {

@Bean(name = "primaryEntityManagerFactory")

public LocalContainerEntityManagerFactoryBean primaryEntityManagerFactory(

@Qualifier("primaryDataSource") DataSource dataSource) {

LocalContainerEntityManagerFactoryBean em = new LocalContainerEntityManagerFactoryBean();

em.setDataSource(dataSource);

em.setPackagesToScan("com.example.employeemanagementsystem.entity");

em.setJpaVendorAdapter(new HibernateJpaVendorAdapter());

return em;

}

@Bean(name = "primaryTransactionManager")

public JpaTransactionManager primaryTransactionManager(

@Qualifier("primaryEntityManagerFactory") EntityManagerFactory entityManagerFactory) {

return new JpaTransactionManager(entityManagerFactory);

}

}

**Secondary Data Source Configuration:**

package com.example.employeemanagementsystem.config;

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

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.orm.jpa.JpaTransactionManager;

import org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean;

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

import javax.persistence.EntityManagerFactory;

import javax.sql.DataSource;

@Configuration

public class SecondaryDataSourceConfig {

@Bean(name = "secondaryEntityManagerFactory")

public LocalContainerEntityManagerFactoryBean secondaryEntityManagerFactory(

@Qualifier("secondaryDataSource") DataSource dataSource) {

LocalContainerEntityManagerFactoryBean em = new LocalContainerEntityManagerFactoryBean();

em.setDataSource(dataSource);

em.setPackagesToScan("com.example.employeemanagementsystem.entity");

em.setJpaVendorAdapter(new HibernateJpaVendorAdapter());

return em;

}

@Bean(name = "secondaryTransactionManager")

public JpaTransactionManager secondaryTransactionManager(

@Qualifier("secondaryEntityManagerFactory") EntityManagerFactory entityManagerFactory) {

return new JpaTransactionManager(entityManagerFactory);

}

}

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

**Step 1: Using Hibernate-Specific Annotations**

package com.example.employeemanagementsystem.entity;

import lombok.Data;

import lombok.NoArgsConstructor;

import org.hibernate.annotations.CreationTimestamp;

import org.hibernate.annotations.UpdateTimestamp;

import javax.persistence.\*;

import java.time.LocalDateTime;

@Data

@NoArgsConstructor

@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;

@CreationTimestamp

@Column(updatable = false)

private LocalDateTime createdDate;

@UpdateTimestamp

private LocalDateTime lastModifiedDate;

@Column(length = 2000)

private String description; // Example of customizing column length

}

**Step 2: Customize Entity Mappings**

package com.example.employeemanagementsystem.entity;

import org.hibernate.annotations.Type;

import javax.persistence.\*;

@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;

@Type(type = "text")

@Column(columnDefinition = "TEXT")

private String longTextField; // Example of custom data type mapping

}

**Step 3: Configure Hibernate Dialect and Properties in application.properties**

# Hibernate Dialect and Properties

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

spring.jpa.properties.hibernate.hbm2ddl.auto=update

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

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

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

**Step 4: Enable Batch Processing**

# Batch Processing Configuration

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

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

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

**Step 5: Implement Batch Processing in Code**

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 org.springframework.transaction.annotation.Transactional;

import java.util.List;

@Service

public class EmployeeService {

@Autowired

private EmployeeRepository employeeRepository;

@Transactional

public void saveEmployeesInBatch(List<Employee> employees) {

final int batchSize = 20;

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

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

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

employeeRepository.flush(); // Flush batch to database

}

}

}

}