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

1. Creating a Spring Boot Project:
   * Step 1: Open Eclipse IDE.
   * Step 2: Go to File -> New -> Maven Project.
   * Step 3: Select Create a simple project (skip archetype selection) and click Next.
   * Step 4: Enter Group Id (e.g., com.staff) and Artifact Id (e.g., StaffAdministrationSystem).
   * Step 5: Click Finish.
   * Step 6: Open the pom.xml file and add the following dependencies inside the <dependencies> tag:
   * Step 7: Save the pom.xml file. Eclipse will automatically download and include the required dependencies.

**Xml code:**

<dependency>

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

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

</dependency>

<dependency>

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

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

</dependency>

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

<optional>true</optional>

</dependency>

1. Configuring Application Properties:

Step 1: In the src/main/resources directory, open application.properties.

Step 2: Add the following configuration for the H2 database connection:

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

spring.datasource.driverClassName=org.h2.Driver

spring.datasource.username=admin

spring.datasource.password=admin123

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

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

1. Creating JPA Entities:
   * Step 1: In src/main/java/com/staff/StaffAdministrationSystem, create a new package named entity.
   * Step 2: Inside the entity package, create a new class StaffMember.java:

package com.staff.StaffAdministrationSystem.entity;

import javax.persistence.\*;

import lombok.Data;

@Entity

@Table(name = "staff\_members")

@Data

public class StaffMember {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

private String email;

@ManyToOne

@JoinColumn(name = "department\_id")

private Department department;

}

* + Step 3: Create another class Department.java:

package com.staff.StaffAdministrationSystem.entity;

import javax.persistence.\*;

import lombok.Data;

import java.util.List;

@Entity

@Table(name = "departments")

@Data

public class Department {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

@OneToMany(mappedBy = "department")

private List<StaffMember> staffMembers;

}

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

1. Creating Repositories:
   * Step 1: In src/main/java/com/staff/StaffAdministrationSystem, create a new package named repository.
   * Step 2: Inside the repository package, create an interface StaffMemberRepository.java:

package com.staff.StaffAdministrationSystem.repository;

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

import com.staff.StaffAdministrationSystem.entity.StaffMember;

public interface StaffMemberRepository extends JpaRepository<StaffMember, Long> {

}

* + Step 3: Create another interface DepartmentRepository.java:

package com.staff.StaffAdministrationSystem.repository;

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

import com.staff.StaffAdministrationSystem.entity.Department;

public interface DepartmentRepository extends JpaRepository<Department, Long> {

}

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

1. Basic CRUD Operations:
   * Step 1: In src/main/java/com/staff/StaffAdministrationSystem, create a package named controller.
   * Step 2: Inside the controller package, create a class StaffMemberController.java:

package com.staff.StaffAdministrationSystem.controller;

import com.staff.StaffAdministrationSystem.entity.StaffMember;

import com.staff.StaffAdministrationSystem.repository.StaffMemberRepository;

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

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

import java.util.List;

@RestController

@RequestMapping("/staff-members")

public class StaffMemberController {

@Autowired

private StaffMemberRepository staffMemberRepository;

@GetMapping

public List<StaffMember> getAllStaffMembers() {

return staffMemberRepository.findAll(); }

@PostMapping

public StaffMember createStaffMember(@RequestBody StaffMember staffMember) {

return staffMemberRepository.save(staffMember);

}

@PutMapping("/{id}")

public StaffMember updateStaffMember(@PathVariable Long id, @RequestBody StaffMember

* **Step 3:** Similarly, create a DepartmentController.java:

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.web.bind.annotation.\*;

import java.util.List;

@RestController

@RequestMapping("/departments")

public class DepartmentController {

@Autowired

private DepartmentRepository departmentRepository;

@GetMapping

public List<Department> getAllDepartments() {

return departmentRepository.findAll(); }

@PostMapping

public Department createDepartment(@RequestBody Department department) {

return departmentRepository.save(department);

}

@PutMapping("/{id}")

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

Department department = departmentRepository.findById(id).orElseThrow();

department.setName(departmentDetails.getName());

return departmentRepository.save(department); }

@DeleteMapping("/{id})

public void deleteDepartment(@PathVariable Long id) {

Department department = departmentRepository.findById(id).orElseThrow();

departmentRepository.delete(department);

}

}

**Exercise 5: Defining Query Methods**

1. Open your Repository Interface:
   * Locate your StaffMemberRepository interface (or similar) that extends JpaRepository or CrudRepository.
2. Add Custom Query Methods:
   * Define methods with names that include keywords like findBy, existsBy, countBy, etc.
   * Example: findByLastName(String lastName)

public interface StaffMemberRepository extends JpaRepository<StaffMember, Long> {

List<StaffMember> findByLastName(String lastName);

List<StaffMember> findByDepartmentId(Long departmentId);

List<StaffMember> findByJobTitle(String jobTitle);

List<StaffMember> findByHireDateAfter(LocalDate hireDate);

}

**b. Using @Query Annotation:**

1. Add @Query Annotation:
   * Use the **@Query** annotation to write JPQL queries.

public interface StaffMemberRepository extends JpaRepository<StaffMember, Long> {

@Query("SELECT s FROM StaffMember s WHERE s.salary > ?1")

List<StaffMember> findStaffMembersWithSalaryGreaterThan(Double salary);

@Query("SELECT s FROM StaffMember s WHERE s.department.id = ?1")

List<StaffMember> findStaffMembersByDepartmentId(Long departmentId);

}

**2. Named Queries**

**a. Define Named Queries:**

1. Add Named Queries to Entity:
   * Use **@NamedQuery** or **@NamedQueries** annotations in your entity class.

@Entity

@NamedQueries({

@NamedQuery(name = "StaffMember.findByLastName", query = "SELECT s FROM StaffMember s WHERE s.lastName = :lastName"),

@NamedQuery(name = "StaffMember.findAll", query = "SELECT s FROM StaffMember s"),

@NamedQuery(name = "StaffMember.findByJobTitle", query = "SELECT s FROM StaffMember s WHERE s.jobTitle = :jobTitle")

})

public class StaffMember {

// fields, getters, setters

}

**b. Execute Named Queries:**

public interface StaffMemberRepository extends JpaRepository<StaffMember, Long> {

@Query(name = "StaffMember.findByLastName")

List<StaffMember> findByLastName(@Param("lastName") String lastName);

@Query(name = "StaffMember.findByJobTitle")

List<StaffMember> findByJobTitle(@Param("jobTitle") String jobTitle);

}

**Exercise 6: Implementing Pagination and Sorting**

**1. Pagination**

**a. Use Page and Pageable:**

1. Modify Repository Method:
   * Change repository methods to return **Page** and use **Pageable** as a parameter.

public interface StaffMemberRepository extends JpaRepository<StaffMember, Long> {

Page<StaffMember> findByDepartmentId(Long departmentId, Pageable pageable);

Page<StaffMember> findByJobTitle(String jobTitle, Pageable pageable);

}

**b. Call the Repository Method:**

public Page<StaffMember> getStaffMembersByDepartment(Long departmentId, int page, int size) {

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

return staffMemberRepository.findByDepartmentId(departmentId, pageable);

}

public Page<StaffMember> getStaffMembersByJobTitle(String jobTitle, int page, int size) {

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

return staffMemberRepository.findByJobTitle(jobTitle, pageable);

}

**2. Sorting**

**a. Combine Sorting and Pagination:**

1. Pass Sorting Information:
   * Use **PageRequest.of(page, size, Sort.by("fieldName"))** to include sorting.

public Page<StaffMember> getStaffMembersByDepartment(Long departmentId, int page, int size) {

Page<StaffMember> staffMembers = staffMemberRepository.findByDepartmentId(departmentId, PageRequest.of(page, size, Sort.by("lastName").ascending()));

return staffMembers;

}

public Page<StaffMember> getStaffMembersByJobTitle(String jobTitle, int page, int size) {

Page<StaffMember> staffMembers = staffMemberRepository.findByJobTitle(jobTitle, PageRequest.of(page, size, Sort.by("jobTitle").descending()));

return staffMembers;

}

**Exercise 7: Enabling Entity Auditing**

**1. Entity Auditing**

**a. Configure Auditing:**

1. Enable Auditing in Application:

@Configuration

@EnableJpaAuditing

public class AuditingConfig {

}

1. Annotate Entity with Auditing Annotations:

@Entity

public class StaffMember {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

// other fields

@CreatedBy

private String createdBy;

@LastModifiedBy

private String lastModifiedBy;

@CreatedDate

private LocalDateTime createdDate;

@LastModifiedDate

private LocalDateTime lastModifiedDate;

// getters and setters

}

**b. Ensure Auditors are Set:**

1. Configure AuditorAware Bean:

@Bean

public AuditorAware<String> auditorProvider() {

return new SpringSecurityAuditorAware();

}

**Exercise 8: Creating Projections**

**1. Projections**

**a. Define Interface-based Projections:**

1. Create Projection Interface:

public interface StaffMemberSummary {

String getFirstName();

String getLastName();

String getJobTitle();

}

1. Use Projection in Repository:

public interface StaffMemberRepository extends JpaRepository<StaffMember, Long> {

List<StaffMemberSummary> findByDepartmentId(Long departmentId);

List<StaffMemberSummary> findByJobTitle(String jobTitle);

}

**b. Define Class-based Projections:**

1. Create a Projection Class:

public class StaffMemberSummary {

private String firstName;

private String lastName;

private String jobTitle;

public StaffMemberSummary(String firstName, String lastName, String jobTitle) {

this.firstName = firstName;

this.lastName = lastName;

this.jobTitle = jobTitle;

}

// getters

}

1. Use Constructor Expression:

public interface StaffMemberRepository extends JpaRepository<StaffMember, Long> {

@Query("SELECT new com.example.StaffMemberSummary(e.firstName, e.lastName, e.jobTitle) FROM StaffMember e WHERE e.departmentId = ?1")

List<StaffMemberSummary> findStaffMemberSummariesByDepartmentId(Long departmentId);

@Query("SELECT new com.example.StaffMemberSummary(e.firstName, e.lastName, e.jobTitle) FROM StaffMember e WHERE e.jobTitle = ?1")

List<StaffMemberSummary> findStaffMemberSummariesByJobTitle(String jobTitle);

}

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

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

# Database configuration

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

spring.datasource.driverClassName=org.h2.Driver

spring.datasource.username=sa

spring.datasource.password=password

# JPA configuration

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

spring.jpa.hibernate.ddl-auto=update

spring.jpa.show-sql=true

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

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

application-dev.properties

# Database configuration for development environment

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

spring.datasource.username=devuser

spring.datasource.password=devpassword

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

# JPA configuration for development environment

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

spring.jpa.hibernate.ddl-auto=update

spring.jpa.show-sql=true

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

**application-prod.properties**

# Database configuration for production environment

spring.datasource.url=jdbc:postgresql://localhost:5432/proddb

spring.datasource.username=produser

spring.datasource.password=prodpassword

spring.datasource.driverClassName=org.postgresql.Driver

# JPA configuration for production environment

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

spring.jpa.hibernate.ddl-auto=none

spring.jpa.show-sql=false

* + Manage multiple data sources within your application.

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

import org.springframework.boot.autoconfigure.orm.jpa.JpaProperties;

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

import org.springframework.boot.jdbc.DataSourceBuilder;

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 org.springframework.transaction.PlatformTransactionManager;

import javax.sql.DataSource;

import java.util.HashMap;

@Configuration

public class DataSourceConfig {

@Bean(name = "primaryDataSource")

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

public DataSource primaryDataSource() {

return DataSourceBuilder.create().build();

}

@Bean(name = "primaryEntityManagerFactory")

public LocalContainerEntityManagerFactoryBean primaryEntityManagerFactory(

@Qualifier("primaryDataSource") DataSource dataSource,

JpaProperties jpaProperties) {

LocalContainerEntityManagerFactoryBean em = new LocalContainerEntityManagerFactoryBean();

em.setDataSource(dataSource);

em.setPackagesToScan("com.example.primary");

em.setJpaVendorAdapter(new HibernateJpaVendorAdapter());

HashMap<String, Object> properties = new HashMap<>(jpaProperties.getProperties());

em.setJpaPropertyMap(properties);

return em;

}

@Bean(name = "primaryTransactionManager")

public PlatformTransactionManager primaryTransactionManager(

@Qualifier("primaryEntityManagerFactory") LocalContainerEntityManagerFactoryBean primaryEntityManagerFactory) {

return new JpaTransactionManager(primaryEntityManagerFactory.getObject());

}

@Bean(name = "secondaryDataSource")

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

public DataSource secondaryDataSource() {

return DataSourceBuilder.create().build();

}

@Bean(name = "secondaryEntityManagerFactory")

public LocalContainerEntityManagerFactoryBean secondaryEntityManagerFactory(

@Qualifier("secondaryDataSource") DataSource dataSource,

JpaProperties jpaProperties) {

LocalContainerEntityManagerFactoryBean em = new LocalContainerEntityManagerFactoryBean();

em.setDataSource(dataSource);

em.setPackagesToScan("com.example.secondary");

em.setJpaVendorAdapter(new HibernateJpaVendorAdapter());

HashMap<String, Object> properties = new HashMap<>(jpaProperties.getProperties());

em.setJpaPropertyMap(properties);

return em;

}

@Bean(name = "secondaryTransactionManager")

public PlatformTransactionManager secondaryTransactionManager(

@Qualifier("secondaryEntityManagerFactory") LocalContainerEntityManagerFactoryBean secondaryEntityManagerFactory) {

return new JpaTransactionManager(secondaryEntityManagerFactory.getObject());

}

@EnableJpaRepositories(

basePackages = "com.example.primary",

entityManagerFactoryRef = "primaryEntityManagerFactory",

transactionManagerRef = "primaryTransactionManager"

)

public interface PrimaryEmployeeRepository extends JpaRepository<PrimaryEmployee, Long> {}

@EnableJpaRepositories(

basePackages = "com.example.secondary",

entityManagerFactoryRef = "secondaryEntityManagerFactory",

transactionManagerRef = "secondaryTransactionManager"

)

public interface SecondaryEmployeeRepository extends JpaRepository<SecondaryEmployee, Long> {}

}

**application.properties**

# Primary Data Source (H2)

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

spring.datasource.primary.username=sa

spring.datasource.primary.password=password

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

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

# Secondary Data Source (MySQL)

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

spring.datasource.secondary.username=secondaryuser

spring.datasource.secondary.password=secondarypassword

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

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

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

**Instructions:**

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

import jakarta.persistence.Entity;

import jakarta.persistence.GeneratedValue;

import jakarta.persistence.GenerationType;

import jakarta.persistence.Id;

import org.hibernate.annotations.LazyCollection;

import org.hibernate.annotations.LazyCollectionOption;

import org.hibernate.annotations.NaturalId;

import java.util.List;

@Entity

class Employee {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@NaturalId

private String email;

private double salary;

private double tax;

public Long getId() {

return id;

}

public void setId(Long id) {

this.id = id;

}

public String getEmail() {

return email;

}

public void setEmail(String email) {

this.email = email;

}

public double getSalary() {

return salary;

}

public void setSalary(double salary) {

this.salary = salary;

}

public double getTax() {

return tax;

}

public void setTax(double tax) {

this.tax = tax;

}

}

@Entity

class Department {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

@OneToMany(mappedBy = "department")

@LazyCollection(LazyCollectionOption.EXTRA)

private List<Employee> employees;

public Long getId() {

return id;

}

public void setId(Long id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public List<Employee> getEmployees() {

return employees;

}

public void setEmployees(List<Employee> employees) {

this.employees = employees;

}

}

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

* Configure Hibernate dialect and properties for optimal performance

spring.jpa.database-platform=org.hibernate.dialect.MySQL8Dialect

spring.jpa.show-sql=true

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

# Second-level cache

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

spring.jpa.properties.hibernate.cache.region.factory\_class=org.hibernate.cache.jcache.JCacheRegionFactory

# Batch processing

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

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

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

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

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

import org.springframework.stereotype.Service;

import org.springframework.transaction.annotation.Transactional;

import java.util.ArrayList;

import java.util.List;

@Service

public class EmployeeService {

@Autowired

private EmployeeRepository employeeRepository;

@Transactional

public void saveEmployeesInBatch(List<Employee> employees) {

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

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

if (i % 20 == 0) { // Flush and clear session every 20 inserts

employeeRepository.flush();

employeeRepository.clear();

}

}

}

@Transactional

public void updateEmployeeSalaries(List<Employee> employees) {

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

Employee employee = employeeRepository.findById(employees.get(i).getId()).orElseThrow();

employee.setSalary(employee.getSalary() + 1000);

if (i % 20 == 0) { // Flush and clear session every 20 updates

employeeRepository.flush();

employeeRepository.clear();

}

}

}

public void testBatchProcessing() {

List<Employee> employees = new ArrayList<>();

for (int i = 1; i <= 1000; i++) {

Employee employee = new Employee();

employee.setName("Employee " + i);

employee.setEmail("employee" + i + "@example.com");

employees.add(employee);

}

saveEmployeesInBatch(employees);

}

}