**Spring Data JPA and Hibernate**

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

**application.properties**

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

spring.datasource.driverClassName=org.h2.Driver

spring.datasource.username=sa

spring.datasource.password=password

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

spring.jpa.hibernate.ddl-auto=update

spring.jpa.show-sql=true

spring.h2.console.enabled=true

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

**EmployeeManagementSystemApplication.java**

package com.example.EmployeeManagementSystem;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

*@SpringBootApplication*

public class EmployeeManagementSystemApplication {

public static void main(String[] args) {

SpringApplication.*run*(EmployeeManagementSystemApplication.class, args);} }

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

**Employee.java**

package com.example.EmployeeManagementSystem.entity;

import jakarta.persistence.\*;

*@Entity*

*@Table*(name = "employee")

public class Employee {

*@Id*

*@GeneratedValue*(strategy = *GenerationType*.***IDENTITY***)

private Long id;

private String name;

private String email;

// Many employees belong to one department

*@ManyToOne*

*@JoinColumn*(name = "department\_id")

private Department department;

// Constructors

public Employee() {}

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

this.name = name;

this.email = email;

this.department = department;

}

// Getters and setters

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 String getEmail() {

return email;

}

public void setEmail(String email) {

this.email = email;

}

public Department getDepartment() {

return department;

}

public void setDepartment(Department department) {

this.department = department;

}

}

**Department.java**

package com.example.EmployeeManagementSystem.entity;

import jakarta.persistence.\*;

import java.util.List;

*@Entity*

*@Table*(name = "department")

public class Department {

*@Id*

*@GeneratedValue*(strategy = *GenerationType*.***IDENTITY***)

private Long id;

private String name;

// One department has many employees

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

private List<Employee> employees;

// Constructors

public Department() {}

public Department(String name) {

this.name = name;

}

// Getters and setters

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;

}

}

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

**DepartmentRepository.java**

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> {

Department findByName(String name);

}

**EmployeeRepository.java**

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

List<Employee> findByNameContaining(String keyword);

// You can also add:

List<Employee> findByDepartment\_Name(String departmentName);

}

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

**DepartmentController.java**

package com.example.EmployeeManagementSystem.entity;

import jakarta.persistence.\*;

import java.util.List;

*@Entity*

*@Table*(name = "department")

public class Department {

*@Id*

*@GeneratedValue*(strategy = *GenerationType*.***IDENTITY***)

private Long id;

private String name;

// One department has many employees

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

private List<Employee> employees;

// Constructors

public Department() {}

public Department(String name) {

this.name = name;

}

// Getters and setters

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;

}

}

**EmployeeController.java**

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.web.bind.annotation.\*;

import java.util.List;

import java.util.Optional;

*@RestController*

*@RequestMapping*("/employees")

public class EmployeeController {

*@Autowired*

private EmployeeRepository employeeRepository;

*@PostMapping*

public Employee createEmployee(*@RequestBody* Employee employee) {

return employeeRepository.save(employee);

}

*@GetMapping*

public List<Employee> getAllEmployees() {

return employeeRepository.findAll();

}

*@GetMapping*("/{id}")

public Employee getEmployeeById(*@PathVariable* Long id) {

Optional<Employee> employee = employeeRepository.findById(id);

return employee.orElse(null);

}

*@PutMapping*("/{id}")

public Employee updateEmployee(*@PathVariable* Long id, *@RequestBody* Employee updatedEmployee) {

return employeeRepository.findById(id)

.map(emp -> {

emp.setName(updatedEmployee.getName());

emp.setSalary(updatedEmployee.getSalary());

emp.setDepartment(updatedEmployee.getDepartment());

return employeeRepository.save(emp);

}).orElse(null);

}

*@DeleteMapping*("/{id}")

public void deleteEmployee(*@PathVariable* Long id) {

employeeRepository.deleteById(id);

} }

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

1. **Defining Query Methods:**

**EmployeeRepository.java**

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 java.util.List;

public interface EmployeeRepository extends JpaRepository<Employee, Long> {

// Method query: Find by name

List<Employee> findByName(String name);

// Method query: Find by department

List<Employee> findByDepartment(String department);

// Method query: Find employees with salary greater than given amount

List<Employee> findBySalaryGreaterThan(double salary);

// Custom JPQL query: Find by name containing (partial search)

@Query("SELECT e FROM Employee e WHERE e.name LIKE %?1%")

List<Employee> searchByName(String keyword);

// Native query: Get all employees with salary less than a value

@Query(value = "SELECT \* FROM employee WHERE salary < ?1", nativeQuery = true)

List<Employee> findLowSalaryEmployees(double salary);

}

1. **Named Queries**

**Employee.java**

@NamedQueries({

@NamedQuery(name = "Employee.findAllEmployees", query = "SELECT e FROM Employee e"),

@NamedQuery(name = "Employee.findByDept", query = "SELECT e FROM Employee e WHERE e.department = ?1")

})

@Entity

@Table(name = "employee")

public class Employee {

// Your existing fields

}

**Employee.java**

@NamedQueries({

@NamedQuery(name = "Employee.findAllEmployees", query = "SELECT e FROM Employee e"),

@NamedQuery(name = "Employee.findByDept", query = "SELECT e FROM Employee e WHERE e.department = ?1")

})

@Entity

@Table(name = "employee")

public class Employee {

// Your existing fields

}

**EmployeeRepository.java**

**Adding the following Named Queries**

@Query(name = "Employee.findAllEmployees")

List<Employee> getAllEmployeesNamed();

@Query(name = "Employee.findByDept")

List<Employee> getEmployeesByDepartment(String department);

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

**EmployeeRepository.java**

package com.example.EmployeeManagementSystem.repository;

import com.example.EmployeeManagementSystem.entity.Employee;

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

public interface EmployeeRepository extends JpaRepository<Employee, Long> {

}

**EmployeeController.java**

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.\*;

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

import java.util.List;

@RestController

@RequestMapping("/employees")

public class EmployeeController {

@Autowired

private EmployeeRepository employeeRepository;

// Pagination only

@GetMapping("/paginated")

public Page<Employee> getPaginatedEmployees(

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

@RequestParam(defaultValue = "5") int size) {

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

return employeeRepository.findAll(pageable);

}

// Sorting only

@GetMapping("/sorted")

public List<Employee> getSortedEmployees(

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

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

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

return employeeRepository.findAll(sort);

}

// Pagination + Sorting

@GetMapping("/paginated-sorted")

public Page<Employee> getPaginatedAndSortedEmployees(

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

@RequestParam(defaultValue = "5") int size,

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

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

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

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

return employeeRepository.findAll(pageable);

}

}

**Exercise-7**

**EmployeeSystemManagementApplication.java**

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 // Enable JPA Auditing

public class EmployeeManagementSystemApplication {

public static void main(String[] args) {

SpringApplication.run(EmployeeManagementSystemApplication.class, args);

}

}

**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() {

// In real apps, return logged-in username

return Optional.of("SYSTEM");

}

}

**Employee.java**

package com.example.EmployeeManagementSystem.entity;

import jakarta.persistence.\*;

import org.springframework.data.annotation.CreatedBy;

import org.springframework.data.annotation.LastModifiedBy;

import org.springframework.data.annotation.CreatedDate;

import org.springframework.data.annotation.LastModifiedDate;

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

import java.time.LocalDateTime;

@Entity

@EntityListeners(AuditingEntityListener.class)

public class Employee {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

private String email;

private Double salary;

@CreatedBy

private String createdBy;

@LastModifiedBy

private String modifiedBy;

@CreatedDate

private LocalDateTime createdDate;

@LastModifiedDate

private LocalDateTime modifiedDate;

// Getters and setters

}

**Department.java**

package com.example.EmployeeManagementSystem.entity;

import jakarta.persistence.\*;

import org.springframework.data.annotation.CreatedBy;

import org.springframework.data.annotation.LastModifiedBy;

import org.springframework.data.annotation.CreatedDate;

import org.springframework.data.annotation.LastModifiedDate;

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

import java.time.LocalDateTime;

@Entity

@EntityListeners(AuditingEntityListener.class)

public class Department {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

@CreatedBy

private String createdBy;

@LastModifiedBy

private String modifiedBy;

@CreatedDate

private LocalDateTime createdDate;

@LastModifiedDate

private LocalDateTime modifiedDate;

// Getters and setters

}

**application.properties**

# Enable SQL logging (optional)

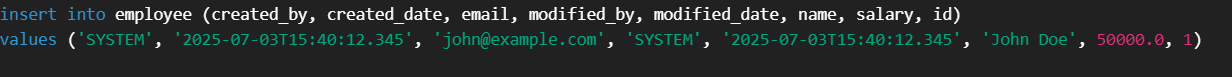
spring.jpa.show-sql=true

# Set Hibernate to update schema

spring.jpa.hibernate.ddl-auto=update

# Format SQL (optional)

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

**Output:**

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

**EmployeeNameSalary.java**

package com.example.demo.projection;

public interface EmployeeNameSalary {

String getName();

Double getSalary();

}

**EmployeeDTO.java**

package com.example.demo.projection;

public class EmployeeDTO {

private String name;

private String email;

public EmployeeDTO(String name, String email) {

this.name = name;

this.email = email;

}

// Getters

public String getName() {

return name;

}

public String getEmail() {

return email;

}

}

**EmployeeRepository.java**

package com.example.demo.repository;

import com.example.demo.entity.Employee;

import com.example.demo.projection.EmployeeNameSalary;

import com.example.demo.projection.EmployeeDTO;

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> {

// Interface-based projection

List<EmployeeNameSalary> findBySalaryGreaterThan(Double salary);

// Class-based projection using constructor expression

@Query("SELECT new com.example.demo.projection.EmployeeDTO(e.name, e.email) FROM Employee e WHERE e.salary < ?1")

List<EmployeeDTO> fetchNameEmailBySalaryLessThan(Double salary);

}

**EmployeeController.java**

package com.example.demo.controller;

import com.example.demo.projection.EmployeeNameSalary;

import com.example.demo.projection.EmployeeDTO;

import com.example.demo.repository.EmployeeRepository;

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

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

import java.util.List;

@RestController

@RequestMapping("/employees")

public class EmployeeController {

@Autowired

private EmployeeRepository employeeRepository;

// Interface-based projection endpoint

@GetMapping("/high-salary")

public List<EmployeeNameSalary> getHighSalaryEmployees(@RequestParam double salary) {

return employeeRepository.findBySalaryGreaterThan(salary);

}

// Class-based projection endpoint

@GetMapping("/low-salary")

public List<EmployeeDTO> getLowSalaryEmployeeDTOs(@RequestParam double salary) {

return employeeRepository.fetchNameEmailBySalaryLessThan(salary);

}

}

**Output:**

****

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

**application.properties**

# ====================

# Primary (Employee)

# ====================

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

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

spring.datasource.primary.username=sa

spring.datasource.primary.password=

# ====================

# Secondary (Department)

# ====================

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

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

spring.datasource.secondary.username=sa

spring.datasource.secondary.password=

# JPA Properties

spring.jpa.show-sql=true

spring.jpa.hibernate.ddl-auto=update

**PrimaryDataSOurceConfig.java**

package com.example.demo.config;

import javax.sql.DataSource;

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

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

import org.springframework.boot.orm.jpa.EntityManagerFactoryBuilder;

import org.springframework.context.annotation.\*;

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

import org.springframework.orm.jpa.\*;

import org.springframework.transaction.PlatformTransactionManager;

@Configuration

@EnableJpaRepositories(

basePackages = "com.example.demo.employee.repository",

entityManagerFactoryRef = "primaryEntityManagerFactory",

transactionManagerRef = "primaryTransactionManager"

)

public class PrimaryDataSourceConfig {

@Bean

@ConfigurationProperties("spring.datasource.primary")

public DataSourceProperties primaryDataSourceProperties() {

return new DataSourceProperties();

}

@Bean

public DataSource primaryDataSource() {

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

}

@Bean

public LocalContainerEntityManagerFactoryBean primaryEntityManagerFactory(

EntityManagerFactoryBuilder builder) {

return builder

.dataSource(primaryDataSource())

.packages("com.example.demo.employee.entity")

.persistenceUnit("primary")

.build();

}

@Bean

public PlatformTransactionManager primaryTransactionManager(

EntityManagerFactoryBuilder builder) {

return new JpaTransactionManager(primaryEntityManagerFactory(builder).getObject());

}

}

**SecondaryDataSourceConfig.java**

package com.example.demo.config;

import javax.sql.DataSource;

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

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

import org.springframework.boot.orm.jpa.EntityManagerFactoryBuilder;

import org.springframework.context.annotation.\*;

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

import org.springframework.orm.jpa.\*;

import org.springframework.transaction.PlatformTransactionManager;

@Configuration

@EnableJpaRepositories(

basePackages = "com.example.demo.department.repository",

entityManagerFactoryRef = "secondaryEntityManagerFactory",

transactionManagerRef = "secondaryTransactionManager"

)

public class SecondaryDataSourceConfig {

@Bean

@ConfigurationProperties("spring.datasource.secondary")

public DataSourceProperties secondaryDataSourceProperties() {

return new DataSourceProperties();

}

@Bean

public DataSource secondaryDataSource() {

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

}

@Bean

public LocalContainerEntityManagerFactoryBean secondaryEntityManagerFactory(

EntityManagerFactoryBuilder builder) {

return builder

.dataSource(secondaryDataSource())

.packages("com.example.demo.department.entity")

.persistenceUnit("secondary")

.build();

}

@Bean

public PlatformTransactionManager secondaryTransactionManager(

EntityManagerFactoryBuilder builder) {

return new JpaTransactionManager(secondaryEntityManagerFactory(builder).getObject());

}

}

**Employee.java**

package com.example.demo.employee.entity;

import jakarta.persistence.\*;

@Entity

public class Employee {

@Id

@GeneratedValue

private Long id;

private String name;

private Double salary;

// Getters & Setters

}

**EmployeeRepository.java**

package com.example.demo.employee.repository;

import com.example.demo.employee.entity.Employee;

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

public interface EmployeeRepository extends JpaRepository<Employee, Long> {}

**Department.java**

package com.example.demo.department.entity;

import jakarta.persistence.\*;

@Entity

public class Department {

@Id

@GeneratedValue

private Long id;

private String name;

// Getters & Setters

}

**DepartmentRepository.java**

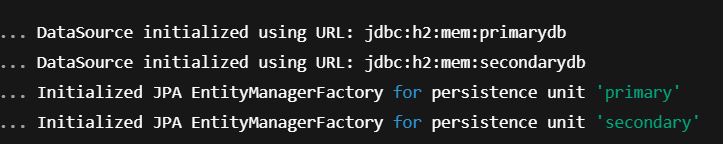
package com.example.demo.department.repository;

import com.example.demo.department.entity.Department;

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

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

**Output:**



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

**Employee.java**

package com.example.demo.entity;

import jakarta.persistence.\*;

import org.hibernate.annotations.DynamicInsert;

import org.hibernate.annotations.DynamicUpdate;

@Entity

@Table(name = "employee")

@DynamicInsert

@DynamicUpdate

public class Employee {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

private double salary;

// Getters and setters

}

**application.properties**

spring.jpa.show-sql=true

spring.jpa.hibernate.ddl-auto=update

# Set Hibernate dialect

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

# Enable Hibernate batch processing

spring.jpa.properties.hibernate.jdbc.batch\_size=30

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

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

spring.jpa.properties.hibernate.generate\_statistics=true

**EmployeeService.java**

package com.example.demo.service;

import com.example.demo.entity.Employee;

import com.example.demo.repository.EmployeeRepository;

import jakarta.persistence.EntityManager;

import jakarta.transaction.Transactional;

import org.springframework.stereotype.Service;

import java.util.List;

@Service

public class EmployeeService {

private final EmployeeRepository employeeRepository;

private final EntityManager entityManager;

public EmployeeService(EmployeeRepository employeeRepository, EntityManager entityManager) {

this.employeeRepository = employeeRepository;

this.entityManager = entityManager;

}

@Transactional

public void batchInsert(List<Employee> employees) {

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

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

if (i % 30 == 0) {

entityManager.flush();

entityManager.clear();

}

}

}

}

**EmployeeController.java**

package com.example.demo.controller;

import com.example.demo.entity.Employee;

import com.example.demo.service.EmployeeService;

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

import java.util.ArrayList;

import java.util.List;

@RestController

@RequestMapping("/employee")

public class EmployeeController {

private final EmployeeService employeeService;

public EmployeeController(EmployeeService employeeService) {

this.employeeService = employeeService;

}

@PostMapping("/batch")

public String batchInsert() {

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

for (int i = 1; i <= 100; i++) {

Employee emp = new Employee();

emp.setName("Emp\_" + i);

emp.setSalary(10000 + i);

employees.add(emp);

}

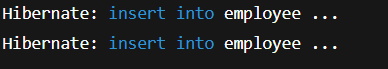
employeeService.batchInsert(employees);

return "Batch insert complete!";

}

}

**Output:**

****