**Spring Data Core JPA & Hibernate**

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

Resources>Application.properties:

# Application Name

spring.application.name=EmployeeManagementSystem

# H2 Database Configuration

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

# H2 Console Configuration

spring.h2.console.enabled=true

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

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

Deparment.java:

package com.example.EmployeeManagementSystem.model;  
  
import jakarta.persistence.\*;  
import java.util.HashSet;  
import java.util.Set;  
  
@Entity  
@Table(name = "departments")  
public class Department {  
  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Long id;  
  
 @Column(name = "name", nullable = false, unique = true)  
 private String name;  
  
 @OneToMany(mappedBy = "department", cascade = CascadeType.*ALL*, fetch = FetchType.*LAZY*)  
 private Set<Employee> employees = new HashSet<>();  
  
 // No-argument constructor  
 public Department() {}  
  
 // Constructor with department name  
 public Department(String name) {  
 this.name = name;  
 }  
  
 // Getter for id  
 public Long getId() {  
 return id;  
 }  
  
 // Setter for id  
 public void setId(Long id) {  
 this.id = id;  
 }  
  
 // Getter for name  
 public String getName() {  
 return name;  
 }  
  
 // Setter for name  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 // Getter for employees set  
 public Set<Employee> getEmployees() {  
 return employees;  
 }  
  
 // Setter for employees set  
 public void setEmployees(Set<Employee> employees) {  
 this.employees = employees;  
 }  
  
 // Method to add an employee  
 public void addEmployee(Employee employee) {  
 employees.add(employee);  
 employee.setDepartment(this);  
 }  
  
 // Method to remove an employee  
 public void removeEmployee(Employee employee) {  
 employees.remove(employee);  
 employee.setDepartment(null);  
 }  
}

Employee.java:

package com.example.EmployeeManagementSystem.model;  
  
import jakarta.persistence.\*;  
  
@Entity  
@Table(name = "employees")  
public class Employee {  
  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Long id;  
  
 @Column(name = "name", nullable = false)  
 private String name;  
  
 @Column(name = "email", nullable = false, unique = true)  
 private String email;  
  
 @ManyToOne(fetch = FetchType.*LAZY*)  
 @JoinColumn(name = "department\_id", nullable = false)  
 private Department department;  
  
 // Default constructor  
 public Employee() {}  
  
 // Parameterized constructor  
 public Employee(String name, String email, Department department) {  
 this.name = name;  
 this.email = email;  
 this.department = department;  
 }  
  
 // Getter for id  
 public Long getId() {  
 return id;  
 }  
  
 // Setter for id  
 public void setId(Long id) {  
 this.id = id;  
 }  
  
 // Getter for name  
 public String getName() {  
 return name;  
 }  
  
 // Setter for name  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 // Getter for email  
 public String getEmail() {  
 return email;  
 }  
  
 // Setter for email  
 public void setEmail(String email) {  
 this.email = email;  
 }  
  
 // Getter for department  
 public Department getDepartment() {  
 return department;  
 }  
 public void setDepartment(Department department) {  
 this.department = department;  
 }  
}

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

EmployeeRepository.java:

package com.example.EmployeeManagementSystem.repository;

import com.example.EmployeeManagementSystem.model.Employee;

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

import org.springframework.stereotype.Repository;

import java.util.List;

@Repository

public interface EmployeeRepository extends JpaRepository<Employee, Long> {

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

List<Employee> findAllByName(String name);

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

Employee findOneByEmail(String email);

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

List<Employee> findAllByDepartmentId(Long departmentId);

}

DepartmentRepository.java:

package com.example.EmployeeManagementSystem.repository;

import com.example.EmployeeManagementSystem.model.Department;

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

import org.springframework.stereotype.Repository;

@Repository

public interface DepartmentRepository extends JpaRepository<Department, Long> {

// Query to locate a department using its name

Department findByDepartmentName(String name);

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

EmployeeController.java:

package com.example.EmployeeManagementSystem.controller;

import com.example.EmployeeManagementSystem.model.Employee;

import com.example.EmployeeManagementSystem.repository.EmployeeRepository;

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

import org.springframework.http.ResponseEntity;

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

import java.util.List;

import java.util.Optional;

@RestController

@RequestMapping("/api/employees")

public class EmployeeController {

@Autowired

private EmployeeRepository employeeRepo;

// Endpoint to create a new employee

@PostMapping

public Employee addEmployee(@RequestBody Employee employee) {

return employeeRepo.save(employee);

}

// Endpoint to retrieve a list of all employees

@GetMapping

public List<Employee> listAllEmployees() {

return employeeRepo.findAll();

}

// Endpoint to retrieve an employee by their ID

@GetMapping("/{id}")

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

Optional<Employee> employeeOpt = employeeRepo.findById(id);

return employeeOpt.map(ResponseEntity::ok)

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

}

// Endpoint to update details of an existing employee

@PutMapping("/{id}")

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

return employeeRepo.findById(id)

.map(existingEmployee -> {

existingEmployee.setName(updatedDetails.getName());

existingEmployee.setEmail(updatedDetails.getEmail());

existingEmployee.setDepartment(updatedDetails.getDepartment());

employeeRepo.save(existingEmployee);

return ResponseEntity.ok(existingEmployee);

})

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

}

// Endpoint to remove an employee by their ID

@DeleteMapping("/{id}")

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

return employeeRepo.findById(id)

.map(employee -> {

employeeRepo.delete(employee);

return ResponseEntity.noContent().build();

})

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

}

DepartmentController.java:

package com.example.EmployeeManagementSystem.controller;

import com.example.EmployeeManagementSystem.model.Department;

import com.example.EmployeeManagementSystem.repository.DepartmentRepository;

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

import org.springframework.http.ResponseEntity;

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

import java.util.List;

import java.util.Optional;

@RestController

@RequestMapping("/api/departments")

public class DepartmentController {

@Autowired

private DepartmentRepository departmentRepo;

// Endpoint to add a new department

@PostMapping

public Department addDepartment(@RequestBody Department department) {

return departmentRepo.save(department);

}

// Endpoint to retrieve all departments

@GetMapping

public List<Department> listDepartments() {

return departmentRepo.findAll();

}

// Endpoint to retrieve a department by its ID

@GetMapping("/{id}")

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

Optional<Department> departmentOpt = departmentRepo.findById(id);

return departmentOpt.map(ResponseEntity::ok)

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

}

// Endpoint to update an existing department

@PutMapping("/{id}")

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

return departmentRepo.findById(id)

.map(existingDepartment -> {

existingDepartment.setName(updatedDepartment.getName());

departmentRepo.save(existingDepartment);

return ResponseEntity.ok(existingDepartment);

})

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

}

// Endpoint to remove a department by its ID

@DeleteMapping("/{id}")

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

return departmentRepo.findById(id)

.map(department -> {

departmentRepo.delete(department);

return ResponseEntity.noContent().build();

})

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

}

}

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

Employee.java:

package com.example.EmployeeManagementSystem.model;

import jakarta.persistence.\*;

import lombok.Data;

@Data

@Entity

@Table(name = "employees")

@NamedQueries({

@NamedQuery(

name = "Employee.findByDeptName",

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

),

@NamedQuery(

name = "Employee.findByEmail",

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

)

})

public class Employee {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

private String email;

@ManyToOne(fetch = FetchType.LAZY)

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

private Department department;

}

EmployeeController.java:

package com.example.EmployeeManagementSystem.controller;

import com.example.EmployeeManagementSystem.model.Employee;

import com.example.EmployeeManagementSystem.repository.EmployeeRepository;

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

import org.springframework.http.ResponseEntity;

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

import java.util.List;

import java.util.Optional;

@RestController

@RequestMapping("/api/employees")

public class EmployeeController {

@Autowired

private EmployeeRepository empRepo;

@PostMapping

public Employee addEmployee(@RequestBody Employee employee) {

return empRepo.save(employee);

}

@GetMapping

public List<Employee> listEmployees() {

return empRepo.findAll();

}

@GetMapping("/{id}")

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

return empRepo.findById(id)

.map(ResponseEntity::ok)

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

}

@PutMapping("/{id}")

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

return empRepo.findById(id)

.map(existingEmployee -> {

existingEmployee.setName(updatedEmployee.getName());

existingEmployee.setEmail(updatedEmployee.getEmail());

existingEmployee.setDepartment(updatedEmployee.getDepartment());

empRepo.save(existingEmployee);

return ResponseEntity.ok(existingEmployee);

})

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

}

@DeleteMapping("/{id}")

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

return empRepo.findById(id)

.map(employee -> {

empRepo.delete(employee);

return ResponseEntity.noContent().build();

})

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

}

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

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

return empRepo.findByNameContainingIgnoreCase(name);

}

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

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

return empRepo.findByDeptName(deptName);

}

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

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

return empRepo.findByDeptNameJPQL(deptName);

}

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

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

return empRepo.findByNameNativeSQL(name);

}

}

DepartmentController.java:

package com.example.EmployeeManagementSystem.controller;

import com.example.EmployeeManagementSystem.model.Department;

import com.example.EmployeeManagementSystem.repository.DepartmentRepository;

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

import org.springframework.http.ResponseEntity;

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

import java.util.List;

import java.util.Optional;

@RestController

@RequestMapping("/api/departments")

public class DepartmentController {

@Autowired

private DepartmentRepository deptRepo;

@PostMapping

public Department addDepartment(@RequestBody Department department) {

return deptRepo.save(department);

}

@GetMapping

public List<Department> listDepartments() {

return deptRepo.findAll();

}

@GetMapping("/{id}")

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

return deptRepo.findById(id)

.map(ResponseEntity::ok)

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

}

@PutMapping("/{id}")

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

return deptRepo.findById(id)

.map(existingDepartment -> {

existingDepartment.setName(updatedDepartment.getName());

deptRepo.save(existingDepartment);

return ResponseEntity.ok(existingDepartment);

})

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

}

@DeleteMapping("/{id}")

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

return deptRepo.findById(id)

.map(department -> {

deptRepo.delete(department);

return ResponseEntity.noContent().build();

})

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

}

}

EmployeeRepository:

package com.example.EmployeeManagementSystem.repository;

import com.example.EmployeeManagementSystem.model.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> {

List<Employee> findByNameContainingIgnoreCase(String name);

List<Employee> findByDeptName(String deptName);

List<Employee> findByEmailStartingWith(String prefix);

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

List<Employee> findByDeptNameJPQL(@Param("deptName") String deptName);

@Query(value = "SELECT \* FROM employees WHERE name LIKE %:name%", nativeQuery = true)

List<Employee> findByNameNativeSQL(@Param("name") String name);

List<Employee> findByDeptNameNamed(@Param("deptName") String deptName);

List<Employee> findByEmailNamed(@Param("email") String email);

}

DepartmentRepository.java:

package com.example.EmployeeManagementSystem.repository;

import com.example.EmployeeManagementSystem.model.Department;

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

import org.springframework.stereotype.Repository;

@Repository

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

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

**EmployeeRepository.java**

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.data.jpa.repository.Query;

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

import org.springframework.stereotype.Repository;

@Repository

public interface EmployeeRepository extends JpaRepository<Employee, Long> {

Page<Employee> findByNameContainingIgnoreCase(String name, Pageable pageable);

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

Page<Employee> findByEmailStartingWith(String prefix, Pageable pageable);

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

Page<Employee> findByDepartmentNameJPQL(@Param("departmentName") String departmentName, Pageable pageable);

@Query(value = "SELECT \* FROM employees WHERE name LIKE %:name%", nativeQuery = true)

Page<Employee> findByNameNativeSQL(@Param("name") String name, Pageable pageable);

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

List<Employee> findByEmailNamed(@Param("email") String email);

}

**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.Page;

import org.springframework.data.domain.Pageable;

import org.springframework.data.web.PageableDefault;

import org.springframework.http.ResponseEntity;

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

import java.util.Optional;

@RestController

@RequestMapping("/api/employees")

public class EmployeeController {

@Autowired

private EmployeeRepository employeeRepository;

@PostMapping

public Employee createEmployee(@RequestBody Employee employee) {

return employeeRepository.save(employee);

}

@GetMapping

public Page<Employee> getAllEmployees(@PageableDefault(size = 10, sort = "name") Pageable pageable) {

return employeeRepository.findAll(pageable);

}

@GetMapping("/{id}")

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

return employeeRepository.findById(id)

.map(ResponseEntity::ok)

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

}

@PutMapping("/{id}")

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

return employeeRepository.findById(id)

.map(existingEmployee -> {

existingEmployee.setName(updatedEmployee.getName());

existingEmployee.setEmail(updatedEmployee.getEmail());

existingEmployee.setDepartment(updatedEmployee.getDepartment());

employeeRepository.save(existingEmployee);

return ResponseEntity.ok(existingEmployee);

})

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

}

@DeleteMapping("/{id}")

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

return employeeRepository.findById(id)

.map(employee -> {

employeeRepository.delete(employee);

return ResponseEntity.noContent().build();

})

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

}

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

public Page<Employee> findByName(@RequestParam String name, @PageableDefault(size = 10, sort = "name") Pageable pageable) {

return employeeRepository.findByNameContainingIgnoreCase(name, pageable);

}

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

public Page<Employee> findByDepartmentName(@RequestParam String departmentName, @PageableDefault(size = 10, sort = "name") Pageable pageable) {

return employeeRepository.findByDepartmentName(departmentName, pageable);

}

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

public Page<Employee> findByDepartmentNameJPQL(@RequestParam String departmentName, @PageableDefault(size = 10, sort = "name") Pageable pageable) {

return employeeRepository.findByDepartmentNameJPQL(departmentName, pageable);

}

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

public Page<Employee> findByNameNativeSQL(@RequestParam String name, @PageableDefault(size = 10, sort = "name") Pageable pageable) {

return employeeRepository.findByNameNativeSQL(name, pageable);

}

}package com.example.EmployeeManagementSystem.entity;

**Department.java**

import jakarta.persistence.\*;

import lombok.Data;

import java.util.List;

@Data

@Entity

@Table(name = "departments")

public class Department {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

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

private List<Employee> employees;

}

**DepartmentRepository.java**

package com.example.EmployeeManagementSystem.repository;

import com.example.EmployeeManagementSystem.entity.Department;

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

import org.springframework.stereotype.Repository;

@Repository

public interface DepartmentRepository extends JpaRepository<Department, Long> {

}

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

**Application Configuration:**

package com.example.EmployeeManagementSystem.config;

import org.springframework.context.annotation.Configuration;

import org.springframework.data.jpa.config.JpaAuditingConfigurer;

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

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

@Configuration

@EnableJpaRepositories

@EnableJpaAuditing

public class JpaConfig extends JpaRepositoryConfigExtension implements JpaAuditingConfigurer {

@Override

public void configureJpaAuditing(JpaAuditingConfigurer configurer) {

// Configuration can be done here if needed

}

}

**Employee.java:**

package com.example.EmployeeManagementSystem.entity;

import jakarta.persistence.\*;

import lombok.Data;

import org.springframework.data.annotation.CreatedBy;

import org.springframework.data.annotation.CreatedDate;

import org.springframework.data.annotation.LastModifiedBy;

import org.springframework.data.annotation.LastModifiedDate;

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

import java.time.LocalDateTime;

@Data

@Entity

@Table(name = "employees")

@EntityListeners(AuditingEntityListener.class)

@NamedQueries({

@NamedQuery(

name = "Employee.findByDepartmentNameNamed",

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

),

@NamedQuery(

name = "Employee.findByEmailNamed",

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

)

})

public class Employee {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

private String email;

@ManyToOne(fetch = FetchType.LAZY)

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

private Department department;

@CreatedBy

private String createdBy;

@CreatedDate

private LocalDateTime createdDate;

@LastModifiedBy

private String lastModifiedBy;

@LastModifiedDate

private LocalDateTime lastModifiedDate;

}

**Department.java:**

package com.example.EmployeeManagementSystem.entity;

import jakarta.persistence.\*;

import lombok.Data;

import org.springframework.data.annotation.CreatedBy;

import org.springframework.data.annotation.CreatedDate;

import org.springframework.data.annotation.LastModifiedBy;

import org.springframework.data.annotation.LastModifiedDate;

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

import java.time.LocalDateTime;

import java.util.List;

@Data

@Entity

@Table(name = "departments")

@EntityListeners(AuditingEntityListener.class)

public class Department {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

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

private List<Employee> employees;

@CreatedBy

private String createdBy;

@CreatedDate

private LocalDateTime createdDate;

@LastModifiedBy

private String lastModifiedBy;

@LastModifiedDate

private LocalDateTime lastModifiedDate;

}

**AuditorAware Implementation:**

package com.example.EmployeeManagementSystem.config;

import org.springframework.data.domain.AuditorAware;

import org.springframework.security.core.Authentication;

import org.springframework.security.core.context.SecurityContextHolder;

import org.springframework.stereotype.Component;

import java.util.Optional;

@Component

public class SpringSecurityAuditorAware implements AuditorAware<String> {

@Override

public Optional<String> getCurrentAuditor() {

Authentication authentication = SecurityContextHolder.getContext().getAuthentication();

if (authentication != null && authentication.isAuthenticated()) {

return Optional.of(authentication.getName());

}

return Optional.empty();

}

}

**JpaConfig.java:**

package com.example.EmployeeManagementSystem.config;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.data.jpa.config.JpaAuditingConfigurer;

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

@Configuration

@EnableJpaRepositories

public class JpaConfig implements JpaAuditingConfigurer {

@Bean

public AuditorAware<String> auditorProvider() {

return new SpringSecurityAuditorAware();

}

}

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

**Configuration**

**Employee Projection Interface:**

package com.example.EmployeeManagementSystem.projection;

import com.example.EmployeeManagementSystem.entity.Department;

public interface EmployeeProjection {

Long getId();

String getName();

String getEmail();

Department getDepartment(); // You can include or exclude related entities as needed

}

**Department Projection Interface:**

package com.example.EmployeeManagementSystem.projection;

public interface DepartmentProjection {

Long getId();

String getName();

}

**EmployeeProjection Class:**

package com.example.EmployeeManagementSystem.projection;

import com.example.EmployeeManagementSystem.entity.Department;

public class EmployeeProjection {

private Long id;

private String name;

private String email;

private Department department;

public EmployeeProjection(Long id, String name, String email, Department department) {

this.id = id;

this.name = name;

this.email = email;

this.department = department;

}

public Long getId() {

return id;

}

public String getName() {

return name;

}

public String getEmail() {

return email;

}

public Department getDepartment() {

return department;

}

}

**DepartmentProjection Class:**

package com.example.EmployeeManagementSystem.projection;

public class DepartmentProjection {

private Long id;

private String name;

public DepartmentProjection(Long id, String name) {

this.id = id;

this.name = name;

}

public Long getId() {

return id;

}

public String getName() {

return name;

}

}

**EmployeeRepository.java:**

package com.example.EmployeeManagementSystem.repository;

import com.example.EmployeeManagementSystem.entity.Employee;

import com.example.EmployeeManagementSystem.projection.EmployeeProjection;

import com.example.EmployeeManagementSystem.projection.DepartmentProjection;

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> {

// Using interface-based projection

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

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

// Using class-based projection

@Query("SELECT new com.example.EmployeeManagementSystem.projection.EmployeeProjection(e.id, e.name, e.email, e.department) FROM Employee e WHERE e.department.name = :departmentName")

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

// Example for department projection

@Query("SELECT new com.example.EmployeeManagementSystem.projection.DepartmentProjection(d.id, d.name) FROM Department d")

List<DepartmentProjection> findAllDepartments();

}

**EmployeeProjection Interface:**

package com.example.EmployeeManagementSystem.projection;

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

public interface EmployeeProjection {

@Value("#{target.id}")

Long getId();

@Value("#{target.name}")

String getName();

@Value("#{target.email}")

String getEmail();

@Value("#{target.department.name}")

String getDepartmentName();

}

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

**Employee.java:**

package com.example.EmployeeManagementSystem.entity;

import jakarta.persistence.\*;

import lombok.Data;

import org.hibernate.annotations.BatchSize;

import org.hibernate.annotations.DynamicInsert;

import org.hibernate.annotations.DynamicUpdate;

@Data

@Entity

@Table(name = "employees")

@NamedQueries({

@NamedQuery(

name = "Employee.findByDepartmentNameNamed",

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

),

@NamedQuery(

name = "Employee.findByEmailNamed",

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

)

})

@DynamicInsert

@DynamicUpdate

@BatchSize(size = 20)

public class Employee {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

private String email;

@ManyToOne(fetch = FetchType.LAZY)

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

private Department department;

}

**Department.java:**

package com.example.EmployeeManagementSystem.entity;

import jakarta.persistence.\*;

import lombok.Data;

import org.hibernate.annotations.BatchSize;

import java.util.List;

@Data

@Entity

@Table(name = "departments")

@BatchSize(size = 20)

public class Department {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

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

private List<Employee> employees;

}

**application.properties:**

# Hibernate Dialect

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

# Show SQL Statements

spring.jpa.show-sql=true

# Format SQL Statements

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

# Enable Hibernate SQL Logging

logging.level.org.hibernate.SQL=DEBUG

# Enable Hibernate Parameter Logging

logging.level.org.hibernate.type.descriptor.sql.BasicBinder=TRACE

# Connection Pool Properties

spring.datasource.hikari.maximum-pool-size=10

spring.datasource.hikari.minimum-idle=5

# Hibernate Caching

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

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

spring.jpa.properties.hibernate.cache.provider\_configuration\_file\_resource\_name=ehcache.xml