Spring Data Core JPA & Hibernate

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

**Resources>Application.properties:**

spring.application.name=EmployeeManagementSystem  
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.h2.console.enabled=true  
spring.h2.console.path=/h2-console

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

**Employee.java:**

package com.example.employeemanagementsystem.entity;

import jakarta.persistence.\*;

import lombok.Data;

@Data

@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;

@ManyToOne

@JoinColumn(name = "department\_id")

private Department department;

}

**Deparment.java:**

package com.example.employeemanagementsystem.entity;

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;

@Column(nullable = false)

private String name;

@OneToMany(mappedBy = "department")

private List<Employee> employees;

}

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

**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 to find employees by department name

List<Employee> findByDepartmentName(String departmentName);

// Derived query method to find an employee by email

Employee findByEmail(String email);

}

**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> {

// Derived query method to find a department by name

Department findByName(String name);

}

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

**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.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 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) {

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

return employee.map(ResponseEntity::ok)

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

}

// Update an Employee

@PutMapping("/{id}")

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

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

if (employeeOptional.isPresent()) {

Employee employee = employeeOptional.get();

employee.setName(employeeDetails.getName());

employee.setEmail(employeeDetails.getEmail());

employee.setDepartment(employeeDetails.getDepartment());

Employee updatedEmployee = employeeRepository.save(employee);

return ResponseEntity.ok(updatedEmployee);

} else {

return ResponseEntity.notFound().build();

}

}

// Delete an Employee

@DeleteMapping("/{id}")

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

if (employeeRepository.existsById(id)) {

employeeRepository.deleteById(id);

return ResponseEntity.noContent().build();

} else {

return ResponseEntity.notFound().build();

}

}

}

**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.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 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) {

Optional<Department> department = departmentRepository.findById(id);

return department.map(ResponseEntity::ok)

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

}

// Update a Department

@PutMapping("/{id}")

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

Optional<Department> departmentOptional = departmentRepository.findById(id);

if (departmentOptional.isPresent()) {

Department department = departmentOptional.get();

department.setName(departmentDetails.getName());

Department updatedDepartment = departmentRepository.save(department);

return ResponseEntity.ok(updatedDepartment);

} else {

return ResponseEntity.notFound().build();

}

}

// Delete a Department

@DeleteMapping("/{id}")

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

if (departmentRepository.existsById(id)) {

departmentRepository.deleteById(id);

return ResponseEntity.noContent().build();

} else {

return ResponseEntity.notFound().build();

}

}

}

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

Employee.java:

package com.example.EmployeeManagementSystem.entity;  
  
import jakarta.persistence.\*;  
import lombok.Data;  
  
@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"  
 )  
})  
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.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;  
import java.util.Optional;  
  
@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) {  
 Optional<Employee> employee = employeeRepository.findById(id);  
 if (employee.isPresent()) {  
 return ResponseEntity.*ok*(employee.get());  
 } else {  
 return ResponseEntity.*notFound*().build();  
 }  
 }  
  
 *// Update an employee by ID* @PutMapping("/{id}")  
 public ResponseEntity<Employee> updateEmployee(@PathVariable Long id, @RequestBody Employee updatedEmployee) {  
 Optional<Employee> employee = employeeRepository.findById(id);  
 if (employee.isPresent()) {  
 Employee existingEmployee = employee.get();  
 existingEmployee.setName(updatedEmployee.getName());  
 existingEmployee.setEmail(updatedEmployee.getEmail());  
 existingEmployee.setDepartment(updatedEmployee.getDepartment());  
 employeeRepository.save(existingEmployee);  
 return ResponseEntity.*ok*(existingEmployee);  
 } else {  
 return ResponseEntity.*notFound*().build();  
 }  
 }  
  
 *// Delete an employee by ID* @DeleteMapping("/{id}")  
 public ResponseEntity<Void> deleteEmployee(@PathVariable Long id) {  
 Optional<Employee> employee = employeeRepository.findById(id);  
 if (employee.isPresent()) {  
 employeeRepository.delete(employee.get());  
 return ResponseEntity.*noContent*().build();  
 } else {  
 return ResponseEntity.*notFound*().build();  
 }  
 }  
  
 *// Additional endpoints to use custom queries  
  
 // Find employees by name containing a specific substring* @GetMapping("/search/by-name")  
 public List<Employee> findByName(@RequestParam String name) {  
 return employeeRepository.findByNameContainingIgnoreCase(name);  
 }  
  
 *// Find employees by department name using method naming convention* @GetMapping("/search/by-department")  
 public List<Employee> findByDepartmentName(@RequestParam String departmentName) {  
 return employeeRepository.findByDepartmentName(departmentName);  
 }  
  
 *// Find employees by department name using custom JPQL query* @GetMapping("/search/by-department-jpql")  
 public List<Employee> findByDepartmentNameJPQL(@RequestParam String departmentName) {  
 return employeeRepository.findByDepartmentNameJPQL(departmentName);  
 }  
  
 *// Find employees by name using native SQL query* @GetMapping("/search/by-name-native")  
 public List<Employee> findByNameNativeSQL(@RequestParam String name) {  
 return employeeRepository.findByNameNativeSQL(name);  
 }  
}

**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.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 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) {  
 Optional<Department> department = departmentRepository.findById(id);  
 if (department.isPresent()) {  
 return ResponseEntity.*ok*(department.get());  
 } else {  
 return ResponseEntity.*notFound*().build();  
 }  
 }  
  
 *// Update a department by ID* @PutMapping("/{id}")  
 public ResponseEntity<Department> updateDepartment(@PathVariable Long id, @RequestBody Department updatedDepartment) {  
 Optional<Department> department = departmentRepository.findById(id);  
 if (department.isPresent()) {  
 Department existingDepartment = department.get();  
 existingDepartment.setName(updatedDepartment.getName());  
 departmentRepository.save(existingDepartment);  
 return ResponseEntity.*ok*(existingDepartment);  
 } else {  
 return ResponseEntity.*notFound*().build();  
 }  
 }  
  
 *// Delete a department by ID* @DeleteMapping("/{id}")  
 public ResponseEntity<Void> deleteDepartment(@PathVariable Long id) {  
 Optional<Department> department = departmentRepository.findById(id);  
 if (department.isPresent()) {  
 departmentRepository.delete(department.get());  
 return ResponseEntity.*noContent*().build();  
 } else {  
 return ResponseEntity.*notFound*().build();  
 }  
 }  
}

EmployeeRepository: 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> {  
  
 *// Find employees by name containing a specific substring (case-insensitive)* List<Employee> findByNameContainingIgnoreCase(String name);  
  
 *// Find employees by department name using method naming convention* List<Employee> findByDepartmentName(String departmentName);  
  
 *// Find employees whose email starts with a specific prefix* List<Employee> findByEmailStartingWith(String prefix);  
  
 *// Custom JPQL query to find employees by department name* @Query("SELECT e FROM Employee e WHERE e.department.name = :departmentName")  
 List<Employee> findByDepartmentNameJPQL(@Param("departmentName") String departmentName);  
  
 *// Custom native SQL query to find employees by name* @Query(value = "SELECT \* FROM employees WHERE name LIKE %:name%", nativeQuery = true)  
 List<Employee> findByNameNativeSQL(@Param("name") String name);  
  
 *// Named Queries* List<Employee> findByDepartmentNameNamed(@Param("departmentName") String departmentName);  
  
 List<Employee> findByEmailNamed(@Param("email") String email);  
}

**Department.java:**

package com.example.EmployeeManagementSystem.entity;  
  
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> {  
  
 *// Add custom queries if needed*}

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

**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;

public interface EmployeeRepository extends JpaRepository<Employee, Long> {

// Method to find all employees with pagination

Page<Employee> findAll(Pageable pageable);

}

**EmployeeController:**

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.GetMapping;

import org.springframework.web.bind.annotation.RequestParam;

import org.springframework.web.bind.annotation.RestController;

@RestController

@RequestMapping("/api/employees")

public class EmployeeController {

@Autowired

private EmployeeRepository employeeRepository;

// Get all Employees with pagination

@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);

}

}

**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;

public interface EmployeeRepository extends JpaRepository<Employee, Long> {

// Method to find all employees with pagination and sorting

Page<Employee> findAll(Pageable pageable);

}

**EmployeeController:**

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.GetMapping;

import org.springframework.web.bind.annotation.RequestParam;

import org.springframework.web.bind.annotation.RestController;

@RestController

@RequestMapping("/api/employees")

public class EmployeeController {

@Autowired

private EmployeeRepository employeeRepository;

// Get all Employees with pagination and sorting

@GetMapping

public Page<Employee> getAllEmployees(

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

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

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

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

Sort sort = sortDir.equalsIgnoreCase(Sort.Direction.ASC.name()) ? 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**

**Enable Auditing in Configuration**

**.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

public class EmployeeManagementSystemApplication {

public static void main(String[] args) {

SpringApplication.run(EmployeeManagementSystemApplication.class, args);

}

}

```

**Configure AuditorAware:**

**.java**

package com.example.employeemanagementsystem.config;

import org.springframework.context.annotation.Configuration;

import org.springframework.data.domain.AuditorAware;

import java.util.Optional;

@Configuration

public class AuditorAwareImpl implements AuditorAware<String> {

@Override

public Optional<String> getCurrentAuditor() {

// Return a hardcoded username for demonstration purposes

return Optional.of("admin");

}

}

```

**Add Auditing Annotations to Entities**

**.java**

package com.example.employeemanagementsystem.model;

import jakarta.persistence.\*;

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;

@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

private String createdBy;

@CreatedDate

private LocalDateTime createdDate;

@LastModifiedBy

private String lastModifiedBy;

@LastModifiedDate

private LocalDateTime lastModifiedDate;

// Getters and setters...

}

@Entity

@Table(name = "departments")

@EntityListeners(AuditingEntityListener.class)

public class Department {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

@CreatedBy

private String createdBy;

@CreatedDate

private LocalDateTime createdDate;

@LastModifiedBy

private String lastModifiedBy;

@LastModifiedDate

private LocalDateTime lastModifiedDate;

// Getters and setters...

}

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

**Define Projections**

**Interface-Based Projection:**

**.java**

package com.example.employeemanagementsystem.projection;

public interface EmployeeProjection {

Long getId();

String getName();

String getEmail();

String getDepartmentName();

}

public interface DepartmentProjection {

Long getId();

String getName();

}

**Class-Based Projection:**

**.java**

package com.example.employeemanagementsystem.dto;

public class EmployeeDTO {

private Long id;

private String name;

private String email;

private String departmentName;

public EmployeeDTO(Long id, String name, String email, String departmentName) {

this.id = id;

this.name = name;

this.email = email;

this.departmentName = departmentName;

}

// Getters and setters...

}

public class DepartmentDTO {

private Long id;

private String name;

public DepartmentDTO(Long id, String name) {

this.id = id;

this.name = name;

}

// Getters and setters...

}

**Use Projections in Repository Methods**

**Using Interface-Based Projection:**

**.java**

package com.example.employeemanagementsystem.repository;

import com.example.employeemanagementsystem.model.Employee;

import com.example.employeemanagementsystem.projection.EmployeeProjection;

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

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

import org.springframework.stereotype.Repository;

import java.util.List;

@Repository

public interface EmployeeRepository extends JpaRepository<Employee, Long> {

// Other query methods...

// Interface-based projection

@Query("SELECT e.id as id, e.name as name, e.email as email, e.department.name as departmentName FROM Employee e")

List<EmployeeProjection> findAllEmployeeProjections();

}

**Using Class-Based Projection:**

**.java**

package com.example.employeemanagementsystem.repository;

import com.example.employeemanagementsystem.model.Employee;

import com.example.employeemanagementsystem.dto.EmployeeDTO;

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

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

import org.springframework.stereotype.Repository;

import java.util.List;

@Repository

public interface EmployeeRepository extends JpaRepository<Employee, Long> {

// Other query methods...

// Class-based projection using constructor expression

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

List<EmployeeDTO> findAllEmployeeDTOs();

}

```

**Fetching Projections in the Service Layer**

**.java**

package com.example.employeemanagementsystem.service;

import com.example.employeemanagementsystem.dto.EmployeeDTO;

import com.example.employeemanagementsystem.projection.EmployeeProjection;

import com.example.employeemanagementsystem.repository.EmployeeRepository;

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

import org.springframework.stereotype.Service;

import java.util.List;

@Service

public class EmployeeService {

@Autowired

private EmployeeRepository employeeRepository;

public List<EmployeeProjection> getAllEmployeeProjections() {

return employeeRepository.findAllEmployeeProjections();

}

public List<EmployeeDTO> getAllEmployeeDTOs() {

return employeeRepository.findAllEmployeeDTOs();

}

// Other service methods...

}

```

**Fetching Projections in the Controller Layer**

**.java**

package com.example.employeemanagementsystem.controller;

import com.example.employeemanagementsystem.dto.EmployeeDTO;

import com.example.employeemanagementsystem.projection.EmployeeProjection;

import com.example.employeemanagementsystem.service.EmployeeService;

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 EmployeeService employeeService;

@GetMapping("/projections")

public List<EmployeeProjection> getEmployeeProjections() {

return employeeService.getAllEmployeeProjections();

}

@GetMapping("/dto")

public List<EmployeeDTO> getEmployeeDTOs() {

return employeeService.getAllEmployeeDTOs();

}

// Other endpoints...

}

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

**Default Data Source Configuration:**

```properties

# Default Data Source Configuration

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

spring.datasource.username=sa

spring.datasource.password=password

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

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

spring.jpa.hibernate.ddl-auto=update

```

**Externalizing Configuration**

**Externalize Configuration in `application.properties`:**

You can externalize your data source configurations in the `application.properties` file. Here's an example for an H2 and a MySQL data source:

```properties

# Default H2 Data Source Configuration

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

spring.datasource.username=sa

spring.datasource.password=password

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

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

spring.jpa.hibernate.ddl-auto=update

# MySQL Data Source Configuration

app.datasource.mysql.url=jdbc:mysql://localhost:3306/employee\_db

app.datasource.mysql.username=root

app.datasource.mysql.password=yourpassword

app.datasource.mysql.driver-class-name=com.mysql.cj.jdbc.Driver

```

**Manage Multiple Data Sources:**

**.java**

package com.example.employeemanagementsystem.config;

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

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.context.annotation.Primary;

import org.springframework.core.env.Environment;

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

import org.springframework.jdbc.datasource.DataSourceTransactionManager;

import org.springframework.orm.jpa.JpaTransactionManager;

import org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean;

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

import javax.sql.DataSource;

import java.util.HashMap;

@Configuration

@EnableJpaRepositories(

basePackages = "com.example.employeemanagementsystem.repository",

entityManagerFactoryRef = "entityManagerFactory",

transactionManagerRef = "transactionManager"

)

public class DataSourceConfig {

@Autowired

private Environment env;

@Primary

@Bean(name = "dataSource")

@ConfigurationProperties(prefix = "spring.datasource")

public DataSource dataSource() {

return DataSourceBuilder.create().build();

}

@Bean(name = "mysqlDataSource")

@ConfigurationProperties(prefix = "app.datasource.mysql")

public DataSource mysqlDataSource() {

return DataSourceBuilder.create().build();

}

@Primary

@Bean(name = "entityManagerFactory")

public LocalContainerEntityManagerFactoryBean entityManagerFactory() {

LocalContainerEntityManagerFactoryBean em = new LocalContainerEntityManagerFactoryBean();

em.setDataSource(dataSource());

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

HibernateJpaVendorAdapter vendorAdapter = new HibernateJpaVendorAdapter();

em.setJpaVendorAdapter(vendorAdapter);

em.setJpaPropertyMap(hibernateProperties());

return em;

}

@Bean(name = "mysqlEntityManagerFactory")

public LocalContainerEntityManagerFactoryBean mysqlEntityManagerFactory() {

LocalContainerEntityManagerFactoryBean em = new LocalContainerEntityManagerFactoryBean();

em.setDataSource(mysqlDataSource());

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

HibernateJpaVendorAdapter vendorAdapter = new HibernateJpaVendorAdapter();

em.setJpaVendorAdapter(vendorAdapter);

em.setJpaPropertyMap(hibernateProperties());

return em;

}

@Primary

@Bean(name = "transactionManager")

public JpaTransactionManager transactionManager() {

JpaTransactionManager transactionManager = new JpaTransactionManager();

transactionManager.setEntityManagerFactory(entityManagerFactory().getObject());

return transactionManager;

}

@Bean(name = "mysqlTransactionManager")

public DataSourceTransactionManager mysqlTransactionManager() {

DataSourceTransactionManager transactionManager = new DataSourceTransactionManager();

transactionManager.setDataSource(mysqlDataSource());

return transactionManager;

}

private HashMap<String, Object> hibernateProperties() {

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

properties.put("hibernate.hbm2ddl.auto", env.getProperty("spring.jpa.hibernate.ddl-auto"));

properties.put("hibernate.dialect", env.getProperty("spring.jpa.database-platform"));

return properties;

}

}

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

**Hibernate-Specific Annotations**

**Example of Hibernate-Specific Annotations:**

**.java**

package com.example.employeemanagementsystem.model;

import jakarta.persistence.\*;

import org.hibernate.annotations.Cache;

import org.hibernate.annotations.CacheConcurrencyStrategy;

import org.hibernate.annotations.CreationTimestamp;

import org.hibernate.annotations.UpdateTimestamp;

import java.time.LocalDateTime;

@Entity

@Table(name = "employees")

@Cache(usage = CacheConcurrencyStrategy.READ\_WRITE)

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")

private Department department;

@CreationTimestamp

private LocalDateTime createdDate;

@UpdateTimestamp

private LocalDateTime lastModifiedDate;

// Getters and setters...

}

- `@Cache`: Configures caching for the entity.

- `@CreationTimestamp` and `@UpdateTimestamp`: Automatically manage timestamps for creation and update events.

**Configuring Hibernate Dialect and Properties**

\*\*Configuring Hibernate Properties in `application.properties`:\*\*

```properties

# Hibernate Configuration

spring.jpa.hibernate.ddl-auto=update

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

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

spring.jpa.properties.hibernate.use\_sql\_comments=true

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

```

**Batch Processing:**

Configure batch processing in `application.properties`:

```properties

# Hibernate Batch Processing

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

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

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

```

**Implementing Batch Processing:**

**.java**

package com.example.employeemanagementsystem.service;

import com.example.employeemanagementsystem.model.Employee;

import com.example.employeemanagementsystem.repository.EmployeeRepository;

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

import org.springframework.stereotype.Service;

import jakarta.transaction.Transactional;

import java.util.List;

@Service

public class EmployeeService {

@Autowired

private EmployeeRepository employeeRepository;

@Transactional

public void saveAllEmployees(List<Employee> employees) {

employeeRepository.saveAll(employees);

}

// Other service methods...

}

```