Spring data JPA and Hibernate

Exercise 1: Employee Management System:

Dependencies:

- Spring Data JPA
- H2 Database
- Spring Web
- Lombok

Using spring initializer:

Open spring initializer / spring.io

Fill in the details as follows:

- **Project**: Maven Project
- **Language**: Java
- **Spring Boot**: 2.7.x or higher (ensure it's compatible with Lombok)
- **Group**: com.example
- **Artifact**: EmployeeManagementSystem
- **Name**: EmployeeManagementSystem
- **Package Name**: com.example.employeemanagementsystem
- **Packaging**: Jar
- **Java Version**: 17 (or your preferred version)

Adding dependencies:

- **Spring Web**: For building web applications.
- **Spring Data JPA**: For interacting with the database using JPA.
- **H2 Database**: An in-memory database for development/testing.
- **Lombok**: To reduce boilerplate code like getters/setters.

Pom.xml

```
<dependencies>
    <!-- Spring Web -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <!-- Spring Data JPA -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>

    <!-- H2 Database -->
    <dependency>
        <groupId>com.h2database</groupId>
        <artifactId>h2</artifactId>
        <scope>runtime</scope>
    </dependency>

    <!-- Lombok -->
    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <optional>true</optional>
    </dependency>

    <!-- Spring Boot Test (optional for testing) -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
</dependencies>
```

Application.properties:

```
# H2 Database configuration
spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=password
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect

# H2 console (optional)
spring.h2.console.enabled=true
spring.h2.console.path=/h2-console
```

Exercise 2: Employee Management System : Define JPA entities for Employee and Department with appropriate relationships.

Employee entity:

```
package com.example.employeemanagementsystem.model;

import jakarta.persistence.*;
import lombok.*;

@Entity
@Table(name = "employees")
@Data
```

```
@NoArgsConstructor
@AllArgsConstructor
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(fetch = FetchType.LAZY)
    @JoinColumn(name = "department_id")
    private Department department;
}
```

Department entity:

```
package com.example.employeemanagementsystem.model;

import jakarta.persistence.*;
import lombok.*;
import java.util.List;

@Entity
@Table(name = "departments")
@Data
@NoArgsConstructor
```

```
@AllArgsConstructor
public class Department {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false, unique = true)
    private String name;

    @OneToMany(mappedBy = "department", cascade = CascadeType.ALL, fetch =
FetchType.LAZY)
    private List<Employee> employees;
}
```

Mapping between the entities and the database tables is handled by JPA annotations

Exercise 3: Employee Management System: Create repositories for Employee and Department entities to perform CRUD operations.

Employee repository:

```
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> {
    // Derived query method to find employees by department name
    List<Employee> findByDepartmentName(String departmentName);

    // Derived query method to find employees by name
    List<Employee> findByNameContaining(String name);

    // Derived query method to find employee by email
    Employee findByEmail(String email);
}
```

Department Repository:

```
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> {

    // Derived query method to find department by name
    Department findByName(String name);
}
```

Exercise 4: Employee Management System:Implement CRUD operations for managing employees and departments.

Employee comptroller

```
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 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 existing employee
    @PutMapping("/{id}")
    public ResponseEntity<Employee> updateEmployee(@PathVariable Long id, @RequestBody
Employee employeeDetails) {
        Optional<Employee> employee = employeeRepository.findById(id);

        if (employee.isPresent()) {
            Employee existingEmployee = employee.get();
            existingEmployee.setName(employeeDetails.getName());
            existingEmployee.setEmail(employeeDetails.getEmail());
            existingEmployee.setDepartment(employeeDetails.getDepartment());
            return ResponseEntity.ok(employeeRepository.save(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();
        }
    }
}
```

Department Controller

```
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 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 an existing department
    @PutMapping("/{id}")
    public ResponseEntity<Department> updateDepartment(@PathVariable Long id,
@RequestBody Department departmentDetails) {
        Optional<Department> department = departmentRepository.findById(id);

        if (department.isPresent()) {
            Department existingDepartment = department.get();
            existingDepartment.setName(departmentDetails.getName());
            return ResponseEntity.ok(departmentRepository.save(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();
        }
    }
}
```

RESTful Endpoints:

- EmployeeController and DepartmentController provide RESTful endpoints for managing employees and departments.
- **POST**: /api/employees and /api/departments to create new employees and departments.
- **GET**: /api/employees and /api/departments to get all employees and departments.
- **GET**: /api/employees/{id} and /api/departments/{id} to get an employee or department by ID.
- **PUT**: /api/employees/{id} and /api/departments/{id} to update an existing employee or department.
- **DELETE**: /api/employees/{id} and /api/departments/{id} to delete an employee or department by ID.

Exercise 5: Employee Management System - Defining Query Methods

Business Scenario: Enhance your repository to support custom queries.

Employee Repository:

```
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> {

   // Custom query method to find employees by department name
   List<Employee> findByDepartmentName(String departmentName);

   // Custom query method to find employees by name containing a string
   List<Employee> findByNameContaining(String name);

   // Custom query method to find an employee by email
   Employee findByEmail(String email);

   // Custom query method to find employees by department name and sorted by name
   List<Employee> findByDepartmentNameOrderByNameAsc(String departmentName);
}
```

Employee entity:

```
package com.example.employeemanagementsystem.model;
```

```
import jakarta.persistence.*;
import lombok.*;

import java.util.List;

@Entity
@Table(name = "employees")
@Data
@NoArgsConstructor
@AllArgsConstructor
@NamedQueries({
    @NamedQuery(name = "Employee.findByDepartment",
                query = "SELECT e FROM Employee e WHERE e.department.name =
:departmentName"),
    @NamedQuery(name = "Employee.searchByName",
                query = "SELECT e FROM Employee e WHERE e.name LIKE :name")
})
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(fetch = FetchType.LAZY)
    @JoinColumn(name = "department_id")
    private Department department;
}
```

Employee repository - @Query` annotation

```
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> {

    // Execute a named query defined in the Employee entity
    @Query(name = "Employee.findByDepartment")
    List<Employee>
findEmployeesByDepartmentUsingNamedQuery(@Param("departmentName") String
departmentName);

    // Execute another named query defined in the Employee entity
    @Query(name = "Employee.searchByName")
    List<Employee> searchEmployeesByNameUsingNamedQuery(@Param("name") String
name);
}
```

Exercise 6: Employee Management System - Implementing Pagination and Sorting Business Scenario: Add pagination and sorting capabilities to your employee search functionality.

Update in Employee Controller:

```
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.data.domain.Page;
import org.springframework.data.domain.PageRequest;
import org.springframework.data.domain.Pageable;
import org.springframework.data.domain.Sort;
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;

   // Other CRUD methods...

   // Pagination and Sorting endpoint
   @GetMapping("/paginated")
```

```
    public Page<Employee> getPaginatedEmployees(
            @RequestParam(defaultValue = "0") int page,
            @RequestParam(defaultValue = "10") int size,
            @RequestParam(defaultValue = "id") String sortBy) {

        Pageable pageable = PageRequest.of(page, size, Sort.by(sortBy));
        return employeeRepository.findAll(pageable);
    }

    // Pagination, Sorting, and Searching by Department endpoint
    @GetMapping("/search")
    public Page<Employee> searchEmployeesByDepartment(
            @RequestParam String departmentName,
            @RequestParam(defaultValue = "0") int page,
            @RequestParam(defaultValue = "10") int size,
            @RequestParam(defaultValue = "name") String sortBy) {

        Pageable pageable = PageRequest.of(page, size, Sort.by(sortBy));
        return employeeRepository.findByDepartmentName(departmentName, pageable);
    }
}
```

EmployeeRepository:

```
package com.example.employeemanagementsystem.repository;

import com.example.employeemanagementsystem.model.Employee;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
```

```
@Repository
public interface EmployeeRepository extends JpaRepository<Employee, Long> {

    // Other query methods...

    // Pagination and sorting method for finding employees by department name
    Page<Employee> findByDepartmentName(String departmentName, Pageable pageable);
}
```

For Testing :

GET http://localhost:8080/api/employees/paginated?page=0&size=5&sortBy=name

Department pagination and sorting

GET
http://localhost:8080/api/employees/search?departmentName=HR&page=0&size=5&sortBy=name

Exercise 7: Employee Management System - Enabling Entity Auditing Business Scenario: Implement auditing to track the creation and modification of employees and departments.

Spring Data JPA provides a convenient way to automatically populate auditing fields like created by, created date, last modified by, and last modified date. To implement this:

- **@CreatedBy**: Populates the field with the user who created the entity.
- **@LastModifiedBy**: Populates the field with the user who last modified the entity.
- **@CreatedDate**: Populates the field with the date and time the entity was created.
- **@LastModifiedDate**: Populates the field with the date and time the entity was last modified.

Enabling JPA auditing file:

```
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);

    }

}
```

Base entity that will be extended by other entities like Enployee and Department:

```
package com.example.employeemanagementsystem.model;



import jakarta.persistence.EntityListeners;

import jakarta.persistence.MappedSuperclass;

import lombok.Getter;

import lombok.Setter;

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;

@MappedSuperclass

@EntityListeners(AuditingEntityListener.class)

@Getter

@Setter

public abstract class Auditable {

  @CreatedBy

   private String createdBy;

  @CreatedDate

   private LocalDateTime createdDate;

   @LastModifiedBy

   private String lastModifiedBy;

   @LastModifiedDate

   private LocalDateTime lastModifiedDate;

}
```

Extend the Auditable base entity in employee and department entities:

```
package com.example.employeemanagementsystem.model;

import jakarta.persistence.*;

import lombok.AllArgsConstructor;

import lombok.Data;

import lombok.NoArgsConstructor;

@Entity

@Table(name = "employees")

@Data

@NoArgsConstructor

@AllArgsConstructor

public class Employee extends Auditable {


    @Id

    @GeneratedValue(strategy = GenerationType.IDENTITY)

    private Long id;


    @Column(nullable = false)

    private String name;
    @Column(nullable = false, unique = true)
    private String email;
   @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "department_id")
    private Department department;
}
```

Department Entity:

```
package com.example.employeemanagementsystem.model;
import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import java.util.List;
@Entity
@Table(name = "departments")
@Data
@NoArgsConstructor
@AllArgsConstructor
public class Department extends Auditable {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @Column(nullable = false, unique = true)
    private String name;
    @OneToMany(mappedBy = "department", cascade = CascadeType.ALL, orphanRemoval =
true)
    private List<Employee> employees;
}
```

Implement AuditorAware:

```
package com.example.employeemanagementsystem.model;
import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import java.util.List;

@Entity
@Table(name = "departments")
```

```
@Data

@NoArgsConstructor

@AllArgsConstructor

public class Department extends Auditable {

    @Id

    @GeneratedValue(strategy = GenerationType.IDENTITY)

    private Long id;

   @Column(nullable = false, unique = true)

    private String name;

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

    private List<Employee> employees;

}
```

AuditorAware:

```
package com.example.employeemanagementsystem.config;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.data.domain.AuditorAware;

import org.springframework.data.jpa.repository.config.EnableJpaAuditing;
```

```
@Configuration

@EnableJpaAuditing(auditorAwareRef = "auditorAware")

public class AuditConfig {

   @Bean

   public AuditorAware<String> auditorAware() {

      return new AuditorAwareImpl();

   }

}
```

Exercise 8: Employee Management System - Creating Projections Business Scenario: Create projections to fetch specific data subsets from the employee and department entities.

Interface-Based Projections:

Employee Entity:

```
package com.example.employeemanagementsystem.projection;


public interface EmployeeProjection {

    Long getId();

    String getName();

    String getEmail();

    String getDepartmentName();

}
```

Updating EmployeeRepository:

EmployeeRepository:

```
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> {


    // Fetching only the specific fields using projection

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

           "FROM Employee e JOIN e.department d")

    List<EmployeeProjection> findAllProjectedBy();

}
```

class-Based projections:

```
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;

  }

  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 String getDepartmentName() {

        return departmentName;

    }

  public void setDepartmentName(String departmentName) {

        this.departmentName = departmentName;

    }

}
```

Updating EmploeeRepository for claa-based Projections:

EmployeeRepository:

```
package com.example.employeemanagementsystem.repository;

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> {

    // Fetching data using class-based projection

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

           "FROM Employee e JOIN e.department d")

    List<EmployeeDTO> findAllEmployeeDTO();

}
```

EmployeeController :

```
package com.example.employeemanagementsystem.controller;

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.web.bind.annotation.GetMapping;

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

import org.springframework.web.bind.annotation.RestController;
import java.util.List;

@RestController

@RequestMapping("/api/employees")

public class EmployeeController {



    @Autowired

    private EmployeeRepository employeeRepository;

   // Endpoint using interface-based projection

    @GetMapping("/projections/interface")

    public List<EmployeeProjection> getEmployeeProjections() {

        return employeeRepository.findAllProjectedBy();

    }

  // Endpoint using class-based projection

    @GetMapping("/projections/class")

    public List<EmployeeDTO> getEmployeeDTOs() {

        return employeeRepository.findAllEmployeeDTO();

    }

}
```

Exercise 9: Employee Management System - Customizing Data Source Configuration Business Scenario: Customize your data source configuration and manage multiple data sources.

Single data source configuration:

application .properties:

```
# application.properties

# Primary Data Source (default)

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

spring.datasource.username=root

spring.datasource.password=yourpassword

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

spring.jpa.hibernate.ddl-auto=update

spring.jpa.show-sql=true
```

Multiple data source configuration :

Application.properties:

```
# application.properties

# Primary Data Source (default)

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

spring.datasource.username=root
spring.datasource.password=yourpassword

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

# Secondary Data Source

secondary.datasource.url=jdbc:postgresql://localhost:5432/departments_db

secondary.datasource.username=postgres

secondary.datasource.password=yourpassword

secondary.datasource.driver-class-name=org.postgresql.Driver
```

Create datsourece configuration classes:

Primary data source configuration:

```
package com.example.employeemanagementsystem.config;

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

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.context.annotation.Primary;

import org.springframework.jdbc.datasource.DriverManagerDataSource;

import javax.sql.DataSource;

@Configuration

public class PrimaryDataSourceConfig {
    @Bean
    @Primary
    @ConfigurationProperties(prefix = "spring.datasource")
    public DataSource primaryDataSource() {
        return new DriverManagerDataSource();
    }
}
```

Secondary datsource configuration:

```
package com.example.employeemanagementsystem.config;
import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.jdbc.datasource.DriverManagerDataSource;
import javax.sql.DataSource;
@Configuration
public class SecondaryDataSourceConfig {
    @Bean
    @ConfigurationProperties(prefix = "secondary.datasource")
    public DataSource secondaryDataSource() {
        return new DriverManagerDataSource();
    }
}
```

Primary data source configuration:

```
package com.example.employeemanagementsystem.config;

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

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

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.context.annotation.Primary;

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

import org.springframework.orm.jpa.JpaTransactionManager;

import org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean;

import org.springframework.transaction.PlatformTransactionManager;

import javax.sql.DataSource;

import java.util.HashMap;

import java.util.Map;

@Configuration

@EnableJpaRepositories(
        basePackages = "com.example.employeemanagementsystem.repository.primary",
        entityManagerFactoryRef = "primaryEntityManagerFactory",
        transactionManagerRef = "primaryTransactionManager"
)
public class PrimaryDataSourceConfig {
    @Autowired
    private DataSource primaryDataSource;
    @Primary
    @Bean(name = "primaryEntityManagerFactory")
    public LocalContainerEntityManagerFactoryBean
primaryEntityManagerFactory(EntityManagerFactoryBuilder builder) {
        return builder
                .dataSource(primaryDataSource)
                .packages("com.example.employeemanagementsystem.model.primary")
                .persistenceUnit("primary")
                .build();
    }

    @Primary
    @Bean(name = "primaryTransactionManager")
    public PlatformTransactionManager primaryTransactionManager(
            LocalContainerEntityManagerFactoryBean primaryEntityManagerFactory) {
        return new JpaTransactionManager(primaryEntityManagerFactory.getObject());
    }
}
```

Secondary data source configuration:

```
package com.example.employeemanagementsystem.config;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.orm.jpa.EntityManagerFactoryBuilder;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.Primary;
import org.springframework.data.jpa.repository.config.EnableJpaRepositories;
import org.springframework.orm.jpa.JpaTransactionManager;
import org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean;
import org.springframework.transaction.PlatformTransactionManager;
import javax.sql.DataSource;
import java.util.HashMap;
import java.util.Map;
```

```
@Configuration

@EnableJpaRepositories(

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

        entityManagerFactoryRef = "primaryEntityManagerFactory",

        transactionManagerRef = "primaryTransactionManager"

)

public class PrimaryDataSourceConfig {

   @Autowired

    private DataSource primaryDataSource;

    @Primary

    @Bean(name = "primaryEntityManagerFactory")

    public LocalContainerEntityManagerFactoryBean
primaryEntityManagerFactory(EntityManagerFactoryBuilder builder) {

        return builder

                .dataSource(primaryDataSource)

                .packages("com.example.employeemanagementsystem.model.primary")

                .persistenceUnit("primary")

                .build();

    }
    @Primary

    @Bean(name = "primaryTransactionManager")

    public PlatformTransactionManager primaryTransactionManager(

            LocalContainerEntityManagerFactoryBean primaryEntityManagerFactory) {

        return new JpaTransactionManager(primaryEntityManagerFactory.getObject());

    }

}
```

Exercise 10: Employee Management System - Hibernate-Specific Features

Business Scenario: Leverage Hibernate-specific features to enhance your application's performance and capabilities.

Using @Type and @Formula

```
package com.example.employeemanagementsystem.model;

import org.hibernate.annotations.Formula;

import org.hibernate.annotations.Type;

import javax.persistence.*;

import java.time.LocalDateTime;

@Entity

@Table(name = "employees")

public class Employee {

    @Id

    @GeneratedValue(strategy = GenerationType.IDENTITY)

    private Long id;

    @Column(name = "name")

    private String name;

    @Column(name = "email")

    private String email;
    @Column(name = "status")
    @Type(type = "org.hibernate.type.StringType")
    private String status;

    @Formula("(SELECT AVG(s.salary) FROM salaries s WHERE s.employee_id = id)")
    private Double averageSalary;

    // Getters and Setters
}
```

Optimizing Fetch Strategies:

```
package com.example.employeemanagementsystem.model;
import org.hibernate.annotations.BatchSize;
import org.hibernate.annotations.Fetch;
import org.hibernate.annotations.FetchMode;
import javax.persistence.*;
import java.util.List;
@Entity
@Table(name = "departments")
public class Department {
    @Id

    @GeneratedValue(strategy = GenerationType.IDENTITY)

    private Long id;

    @Column(name = "name")

    private String name;


    @OneToMany(mappedBy = "department")

    @BatchSize(size = 10)

    @Fetch(FetchMode.SUBSELECT)

    private List<Employee> employees;

}
```

Setting Hibernate Dialect in application.properties:

```
# application.properties

# Hibernate Dialect for MySQL

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

# Additional Hibernate properties

spring.jpa.properties.hibernate.format_sql=true

spring.jpa.properties.hibernate.use_sql_comments=true
spring.jpa.properties.hibernate.jdbc.batch_size=20

spring.jpa.properties.hibernate.order_inserts=true

spring.jpa.properties.hibernate.order_updates=true

spring.jpa.properties.hibernate.cache.use_second_level_cache=true

spring.jpa.properties.hibernate.cache.use_query_cache=true
```

Implementing Batch Processing in service layer:

```
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 org.springframework.transaction.annotation.Transactional;


import java.util.List;

@Service

public class EmployeeService {

  @Autowired

    private EmployeeRepository employeeRepository;
    @Transactional

    public void batchInsertEmployees(List<Employee> employees) {

        int batchSize = 20; // Matches the hibernate.jdbc.batch_size configuration

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

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

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

                // Flush and clear the session to manage memory and avoid OutOfMemoryError

                employeeRepository.flush();

                employeeRepository.clear();

            }

        }

    }


}
```