Exercise 1: Employee Management System - Overview and Setup

Creating a Spring Boot Project:

- **Step 1:** Use Spring Initializr (https://start.spring.io/) to create a new Spring Boot project named EmployeeManagementSystem.
- **Step 2:** Add the following dependencies:
    - Spring Data JPA: For data persistence.
    - H2 Database: An in-memory database for quick setup.
    - Spring Web: For building RESTful APIs.
    - Lombok: To reduce boilerplate code (e.g., getters, setters, constructors).

Configuring Application Properties:

- **Step 1:** Open the src/main/resources/application.properties file.
- **Step 2:** Configure the H2 database connection as follows:

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

Exercise 2: Employee Management System - Creating Entities

Creating JPA Entities:

- **Step 1:** Create the Employee entity class.
    - Fields: id, name, email, department.
- **Step 2:** Create the Department entity class.
    - Fields: id, name.

Mapping Entities to Database Tables:

- Use annotations such as:
    - @Entity: Marks the class as a JPA entity.
    - @Table(name = "table_name"): Optional, specifies the table name.
    - @Id and @GeneratedValue: To indicate the primary key.
- Define a @OneToMany relationship between Department and Employee.

@Entity

```
public class Department {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
    @OneToMany(mappedBy = "department")
    private List<Employee> employees = new ArrayList<>();
}
@Entity
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;
}
```

Exercise 3: Employee Management System - Creating Repositories

Overview of Spring Data Repositories:

- Spring Data repositories simplify data access by providing CRUD methods out of the box.

Creating Repositories:

- **Step 1:** Create EmployeeRepository and DepartmentRepository interfaces.
- **Step 2:** Extend JpaRepository for both entities.

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

Exercise 4: Employee Management System - Implementing CRUD Operations

Basic CRUD Operations:

- **Step 1:** Implement CRUD operations using JpaRepository methods.

- **Step 2:** Create EmployeeController and DepartmentController to expose RESTful endpoints.

```
@RestController
@RequestMapping("/employees")
public class EmployeeController {
    @Autowired
    private EmployeeRepository employeeRepository;
    @GetMapping
    public List<Employee> getAllEmployees() {
        return employeeRepository.findAll();
    }
    @PostMapping
    public Employee createEmployee(@RequestBody Employee employee) {
        return employeeRepository.save(employee);
    }
    @PutMapping("/{id}")
    public Employee updateEmployee(@PathVariable Long id, @RequestBody Employee
employeeDetails) {
        Employee employee = employeeRepository.findById(id)
            .orElseThrow(() -> new ResourceNotFoundException("Employee not found"));
        employee.setName(employeeDetails.getName());
        employee.setEmail(employeeDetails.getEmail());
        return employeeRepository.save(employee);
    }
    @DeleteMapping("/{id}")
    public ResponseEntity<?> deleteEmployee(@PathVariable Long id) {
        Employee employee = employeeRepository.findById(id)
            .orElseThrow(() -> new ResourceNotFoundException("Employee not found"));
        employeeRepository.delete(employee);
        return ResponseEntity.ok().build();
    }
}
```

Exercise 5: Employee Management System - Defining Query Methods

Defining Query Methods:

- Use method names to create custom queries, e.g., findByDepartmentName(String departmentName).

Named Queries:

- Use @Query to define complex queries or @NamedQuery for reusable queries.

```
@Query("SELECT e FROM Employee e WHERE e.department.name = :departmentName")
List<Employee> findByDepartmentName(@Param("departmentName") String departmentName);
```

Exercise 6: Employee Management System - Implementing Pagination and Sorting

Pagination:

- Implement pagination using Page and Pageable in the repository and controller.

```
@GetMapping("/employees")
public Page<Employee> getAllEmployees(Pageable pageable) {
    return employeeRepository.findAll(pageable);
}
```

Sorting:

- Add sorting functionality to your queries.

```
@GetMapping("/employees")
public List<Employee> getAllEmployees(Sort sort) {
    return employeeRepository.findAll(sort);
}
```

Exercise 7: Employee Management System - Enabling Entity Auditing

Entity Auditing:

- Enable auditing by configuring the application and annotating entities with @CreatedDate, @LastModifiedDate, etc.

```
@Entity
@EntityListeners(AuditingEntityListener.class)
public class Employee {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
```

```
    private Long id;
    private String name;
    private String email;
    @CreatedDate
    private LocalDateTime createdDate;
    @LastModifiedDate
    private LocalDateTime lastModifiedDate;
    // Other fields and methods...
}
```

Exercise 8: Employee Management System - Creating Projections

Projections:

- Define interface-based and class-based projections to control the data returned from the repository.

```
public interface EmployeeNameProjection {
    String getName();
}
@Query("SELECT e.name as name FROM Employee e")
List<EmployeeNameProjection> findEmployeeNames();
```

Exercise 9: Employee Management System - Customizing Data Source Configuration

Spring Boot Auto-Configuration:

- Use Spring Boot's auto-configuration for setting up data sources.

Externalizing Configuration:

- Use application.properties to manage configurations and switch between multiple data sources.

Exercise 10: Employee Management System - Hibernate-Specific Features

1. **Hibernate-Specific Annotations:**
    - Use annotations like @BatchSize, @Cache, @LazyCollection, etc., to optimize performance.
2. **Configuring Hibernate Dialect and Properties:**
    - Fine-tune Hibernate settings in application.properties for better performance.

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

spring.jpa.properties.hibernate.format_sql=true

3. **Batch Processing:**
    - Implement batch processing for bulk operations to improve performance.

@Modifying

@Query("UPDATE Employee e SET e.salary = :salary WHERE e.department.id = :departmentId")

int bulkUpdateSalary(@Param("salary") double salary, @Param("departmentId") Long departmentId);