**Spring Data JPA and Hibernate.docx**

**Exercise-1: Overview and Setup**

**application.properties**

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

spring.datasource.driverClassName=org.h2.Driver

spring.datasource.username=sa

spring.datasource.password=password

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

spring.h2.console.enabled=true

spring.jpa.show-sql=true

spring.jpa.hibernate.ddl-auto=create

**pom.xml**

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

<scope>runtime</scope>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

<optional>true</optional>

</dependency>

</dependencies>

**EmployeeManagementApplication.java**

package com.aru.employeemanagement;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class EmployeeManagementApplication {

public static void main(String[] args) {

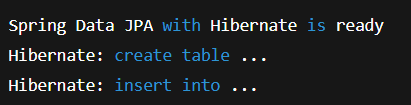
SpringApplication.run(EmployeeManagementApplication.class, args);

System.out.println("Spring Data JPA with Hibernate is ready");

}

}

**Output:**

****

**Exercise:2 Creating Entities**

**Department.java**

package com.aru.employeemanagement.entity;

import jakarta.persistence.\*;

import java.util.List;

@Entity

@Table(name = "department")

public class Department {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

// Optional bi-directional relationship

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

private List<Employee> employees;

public Department() {}

public Department(String name) {

this.name = name;

}

public Long getId() {

return id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public List<Employee> getEmployees() {

return employees;

}

public void setEmployees(List<Employee> employees) {

this.employees = employees;

}

}

**Employee.java**

package com.aru.employeemanagement.entity;

import jakarta.persistence.\*;

@Entity

@Table(name = "employee")

public class Employee {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

private String email;

@ManyToOne

@JoinColumn(name = "department\_id")

private Department department;

public Employee() {}

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

this.name = name;

this.email = email;

this.department = department;

}

public Long getId() {

return id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getEmail() {

return email;

}

public void setEmail(String email) {

this.email = email;

}

public Department getDepartment() {

return department;

}

public void setDepartment(Department department) {

this.department = department;

}

}

**Exercise 3: Creating Repositories for Employee & Department**

**EmployeeRepository.java**

package com.aru.employeemanagement.repository;

import com.aru.employeemanagement.entity.Employee;

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

import org.springframework.stereotype.Repository;

@Repository

public interface EmployeeRepository extends JpaRepository<Employee, Long> {

}

**DepartmentRepository.java**

package com.aru.employeemanagement.repository;

import com.aru.employeemanagement.entity.Department;

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

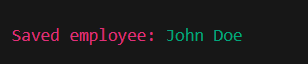
import org.springframework.stereotype.Repository;

@Repository

public interface DepartmentRepository extends JpaRepository<Department, Long> {

}

**Output:**

****

**Exercise 4: Implementing CRUD Operations via REST Controllers**

**EmployeeController.java**

package com.aru.employeemanagement.controller;

import com.aru.employeemanagement.entity.Employee;

import com.aru.employeemanagement.repository.EmployeeRepository;

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

import java.util.List;

@RestController

@RequestMapping("/employees")

public class EmployeeController {

private final EmployeeRepository employeeRepo;

public EmployeeController(EmployeeRepository employeeRepo) {

this.employeeRepo = employeeRepo;

}

@GetMapping

public List<Employee> getAll() {

return employeeRepo.findAll();

}

@GetMapping("/{id}")

public Employee getById(@PathVariable Long id) {

return employeeRepo.findById(id).orElse(null);

}

@PostMapping

public Employee create(@RequestBody Employee employee) {

return employeeRepo.save(employee);

}

@PutMapping("/{id}")

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

return employeeRepo.findById(id).map(emp -> {

emp.setName(updatedEmployee.getName());

emp.setEmail(updatedEmployee.getEmail());

emp.setDepartment(updatedEmployee.getDepartment());

return employeeRepo.save(emp);

}).orElse(null);

}

@DeleteMapping("/{id}")

public void delete(@PathVariable Long id) {

employeeRepo.deleteById(id);

}

}

**DepartmentController.java**

package com.aru.employeemanagement.controller;

import com.aru.employeemanagement.entity.Department;

import com.aru.employeemanagement.repository.DepartmentRepository;

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

import java.util.List;

@RestController

@RequestMapping("/departments")

public class DepartmentController {

private final DepartmentRepository departmentRepo;

public DepartmentController(DepartmentRepository departmentRepo) {

this.departmentRepo = departmentRepo;

}

@GetMapping

public List<Department> getAll() {

return departmentRepo.findAll();

}

@GetMapping("/{id}")

public Department getById(@PathVariable Long id) {

return departmentRepo.findById(id).orElse(null);

}

@PostMapping

public Department create(@RequestBody Department department) {

return departmentRepo.save(department);

}

@PutMapping("/{id}")

public Department update(@PathVariable Long id, @RequestBody Department updatedDepartment) {

return departmentRepo.findById(id).map(dept -> {

dept.setName(updatedDepartment.getName());

return departmentRepo.save(dept);

}).orElse(null);

}

@DeleteMapping("/{id}")

public void delete(@PathVariable Long id) {

departmentRepo.deleteById(id);

}

}

**Testing with Postman**

**Add Department**

POST /departments

Content-Type: application/json

{

"name": "Engineering"

}

**Add Employee**

POST /employees

Content-Type: application/json

{

"name": "Jane Doe",

"email": "jane@example.com",

"department": {

"id": 1

}

}

**Get All Employees**

GET /employees

**Exercise-5: Defining Query Methods**

**Add Custom Methods in EmployeeRepository.java**

package com.aru.employeemanagement.repository;

import com.aru.employeemanagement.entity.Employee;

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> {

// 1. Derived query

List<Employee> findByName(String name);

// 2. Custom JPQL query

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

List<Employee> findByEmailContaining(String keyword);

// 3. Derived query with Department

List<Employee> findByDepartmentId(Long departmentId);

}

**Named Query (in Employee.java)**

@NamedQuery(

name = "Employee.findAllWithNameStartingWith",

query = "SELECT e FROM Employee e WHERE e.name LIKE :prefix%"

)

@Entity

@Table(name = "employee")

public class Employee {

// existing fields and methods

}

**In repository**

@Query(name = "Employee.findAllWithNameStartingWith")

List<Employee> findAllWithNameStartingWith(@Param("prefix") String prefix);

**Example Usage in Controller or Service**

System.out.println("Employees in Engineering:");

employeeRepo.findByDepartmentId(1L).forEach(emp -> System.out.println(emp.getName()));

System.out.println("Employees with email '@example.com':");

employeeRepo.findByEmailContaining("example.com").forEach(emp -> System.out.println(emp.getEmail()));

**Exercise 6: Implementing Pagination and Sorting**

**Update EmployeeRepository.java**

Page<Employee> findAll(Pageable pageable);

**Update EmployeeController.java to support pagination and sorting**

package com.aru.employeemanagement.controller;

import com.aru.employeemanagement.entity.Employee;

import com.aru.employeemanagement.repository.EmployeeRepository;

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.\*;

@RestController

@RequestMapping("/employees")

public class EmployeeController {

private final EmployeeRepository employeeRepo;

public EmployeeController(EmployeeRepository employeeRepo) {

this.employeeRepo = employeeRepo;

}

// Existing endpoints...

// Paginated and sorted endpoint

@GetMapping("/paginated")

public Page<Employee> getPaginated(

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

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

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

@RequestParam(defaultValue = "asc") String direction

) {

Sort sort = direction.equalsIgnoreCase("desc")

? Sort.by(sortBy).descending()

: Sort.by(sortBy).ascending();

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

return employeeRepo.findAll(pageable);

}

}

**Example Requests:**

GET /employees/paginated?page=0&size=3

GET /employees/paginated?page=1&size=5&sortBy=name&direction=desc

**Output:**

****

**Exercise 7: Enabling Entity Auditing**

**Update EmployeeManagementApplication.java**

package com.aru.employeemanagement;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

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

@SpringBootApplication

@EnableJpaAuditing

public class EmployeeManagementApplication {

public static void main(String[] args) {

SpringApplication.run(EmployeeManagementApplication.class, args);

}

}

**Add Audit Fields to Employee.java**

package com.aru.employeemanagement.entity;

import jakarta.persistence.\*;

import org.springframework.data.annotation.CreatedDate;

import org.springframework.data.annotation.LastModifiedDate;

import java.time.LocalDateTime;

@Entity

@Table(name = "employee")

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;

@CreatedDate

@Column(updatable = false)

private LocalDateTime createdAt;

@LastModifiedDate

private LocalDateTime updatedAt;

public Employee() {}

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

this.name = name;

this.email = email;

this.department = department;

}

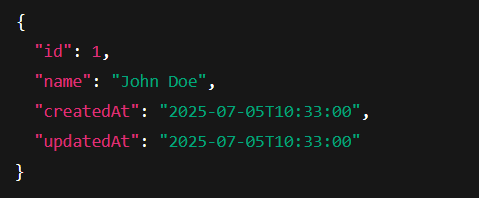
// Getters and setters...

}

**Add AuditingEntityListener to Employee.java**

@EntityListeners(org.springframework.data.jpa.domain.support.AuditingEntityListener.class)

**Output:**

****

**Exercise 8: Projections in Spring Data JPA**

**Projection Interface**

package com.aru.employeemanagement.projection;

public interface EmployeeNameEmailProjection {

String getName();

String getEmail();

}

**Add a query in EmployeeRepository**

package com.aru.employeemanagement.repository;

import com.aru.employeemanagement.entity.Employee;

import com.aru.employeemanagement.projection.EmployeeNameEmailProjection;

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

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

import java.util.List;

public interface EmployeeRepository extends JpaRepository<Employee, Long> {

@Query("SELECT e FROM Employee e")

List<EmployeeNameEmailProjection> findAllNameEmailOnly();

}

**Add REST Endpoint**

@GetMapping("/projection")

public List<EmployeeNameEmailProjection> getNameEmailOnly() {

return employeeRepo.findAllNameEmailOnly();

}

**Class Based Projection**

**Create a DTO class**

package com.aru.employeemanagement.dto;

public class EmployeeDTO {

private String name;

private String departmentName;

public EmployeeDTO(String name, String departmentName) {

this.name = name;

this.departmentName = departmentName;

}

public String getName() {

return name;

}

public String getDepartmentName() {

return departmentName;

}

}

**Add Constructor Query in EmployeeRepository**

@Query("SELECT new com.aru.employeemanagement.dto.EmployeeDTO(e.name, e.department.name) FROM Employee e")

List<EmployeeDTO> fetchNameAndDepartment();

**Add REST Endpoint**

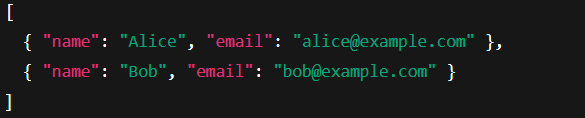
@GetMapping("/dto")

public List<EmployeeDTO> getCustomDTO() {

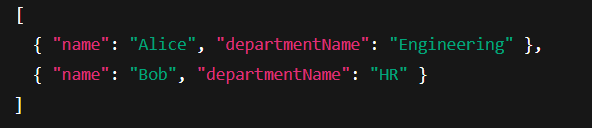
return employeeRepo.fetchNameAndDepartment();

}

**Output interface projection:**

****

**Output DTO projection:**

****

**Exercise-9: Customizing Data Source Configuration**

**application.properties**

# Custom H2 configuration

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

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

spring.datasource.username=sa

spring.datasource.password=

# Hibernate config

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

spring.jpa.hibernate.ddl-auto=update

spring.jpa.show-sql=true

# Enable H2 console

spring.h2.console.enabled=true

**Add second DB properties**

# Primary data source

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

spring.datasource.username=sa

spring.datasource.password=

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

# Secondary data source (custom prefix)

audit.datasource.url=jdbc:h2:mem:auditdb

audit.datasource.username=sa

audit.datasource.password=

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

**Define DataSourceConfig.java**

@Configuration

@EnableTransactionManagement

@EnableJpaRepositories(

basePackages = "com.aru.employeemanagement.repository",

entityManagerFactoryRef = "entityManagerFactory",

transactionManagerRef = "transactionManager"

)

public class PrimaryDataSourceConfig {

@Bean

@Primary

@ConfigurationProperties(prefix = "spring.datasource")

public DataSource primaryDataSource() {

return DataSourceBuilder.create().build();

}

@Bean

@Primary

public LocalContainerEntityManagerFactoryBean entityManagerFactory(

EntityManagerFactoryBuilder builder) {

return builder

.dataSource(primaryDataSource())

.packages("com.aru.employeemanagement.entity")

.build();

}

@Bean

@Primary

public PlatformTransactionManager transactionManager(

EntityManagerFactory entityManagerFactory) {

return new JpaTransactionManager(entityManagerFactory);

}

}

**Exercise 10: Hibernate-Specific Features**

**Custom Column Name and Length**

@Column(name = "emp\_email", length = 150, nullable = false, unique = true)

private String email;

**Immutable Entity**

import org.hibernate.annotations.Immutable;

@Entity

@Immutable

public class AuditLog {

// fields...

}

**Caching (2nd level)**

import org.hibernate.annotations.Cache;

import org.hibernate.annotations.CacheConcurrencyStrategy;

@Cache(usage = CacheConcurrencyStrategy.READ\_WRITE)

@Entity

public class Department {

// fields...

}

**Hibernate Configuration in application.properties**

# Basic settings

spring.jpa.show-sql=true

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

# Recommended dialect (for PostgreSQL, MySQL, etc.)

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

# Batch size tuning

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

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

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

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

**Batch Insert Example**

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

for (int i = 0; i < 1000; i++) {

employees.add(new Employee("User" + i, "user" + i + "@example.com", department));

}

employeeRepo.saveAll(employees);

**Tune Entity Fetching**

@ManyToOne(fetch = FetchType.LAZY)

@JoinColumn(name = "department\_id")

private Department department;