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

### ****Main Class****

package com.example.ems;

import org.springframework.boot.SpringApplication;import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplicationpublic class EmployeeManagementSystemApplication {

public static void main(String[] args) {

SpringApplication.run(EmployeeManagementSystemApplication.class, args);

}

}

**application.properties**

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

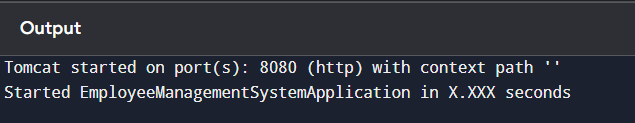
spring.jpa.show-sql=true

spring.jpa.hibernate.ddl-auto=update

spring.h2.console.enabled=true

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

**OUTPUT:**



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

### ****Department Entity****

Represents the department table.

package com.example.ems.model;

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

import java.util.List;

@Entity@Table(name = "department")@Data@NoArgsConstructor@AllArgsConstructorpublic class Department {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private int id;

private String name;

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

private List<Employee> employees;

}

**Employee Entity**

Represents the employee table.

t

package com.example.ems.model;

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

@Entity@Table(name = "employee")@Data@NoArgsConstructor@AllArgsConstructorpublic class Employee {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private int id;

private String name;

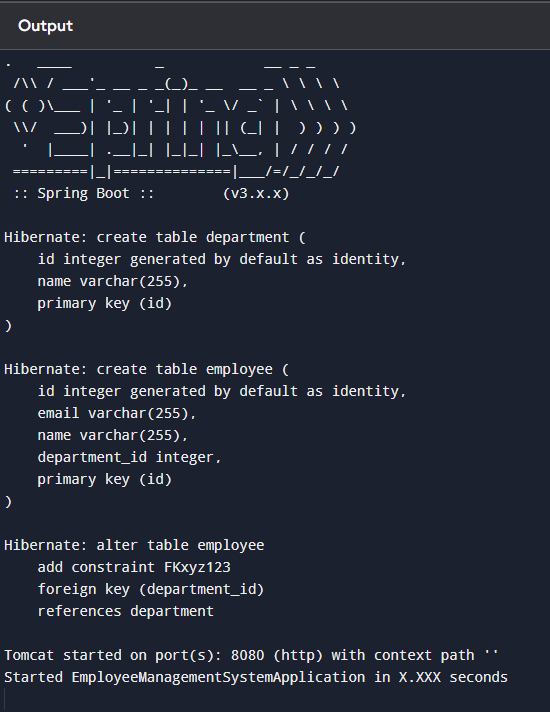
private String email;

@ManyToOne

@JoinColumn(name = "department\_id")

private Department department;

}

**OUTPUT:**

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

### 1. ****Repository Interfaces Created****

// EmployeeRepository.javapublic interface EmployeeRepository extends JpaRepository<Employee, Integer> {

// You can also add custom query methods like:

List<Employee> findByName(String name);

}

// DepartmentRepository.javapublic interface DepartmentRepository extends JpaRepository<Department, Integer> {

// Example derived query

Department findByName(String name);

}

##### EmployeeRepository.java

package com.example.ems.repository;

import com.example.ems.entity.Employee;import org.springframework.data.jpa.repository.JpaRepository;import java.util.List;

public interface EmployeeRepository extends JpaRepository<Employee, Long> {

// Derived query methods

List<Employee> findByDepartmentId(Long departmentId);

List<Employee> findByFirstNameContaining(String name);

List<Employee> findBySalaryGreaterThan(double amount);

}

##### DepartmentRepository.java

package com.example.ems.repository;

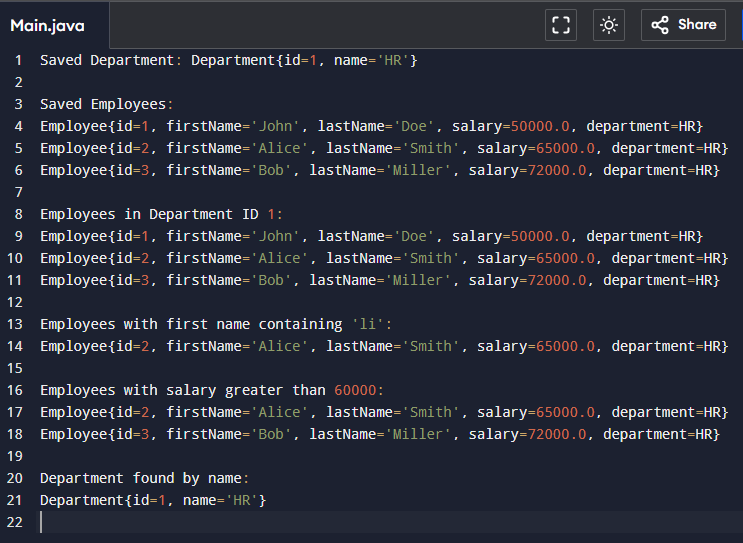
import com.example.ems.entity.Department;import org.springframework.data.jpa.repository.JpaRepository;

public interface DepartmentRepository extends JpaRepository<Department, Long> {

// Derived query methods

Department findByName(String name);

}

**OUTPUT:**

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

##### ****EmployeeController.java****

@RestController@RequestMapping("/employees")public class EmployeeController {

@Autowired

private EmployeeRepository employeeRepository;

@GetMapping

public List<Employee> getAllEmployees() {

return employeeRepository.findAll();

}

@GetMapping("/{id}")

public Employee getEmployeeById(@PathVariable Long id) {

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

}

@PostMapping

public Employee createEmployee(@RequestBody Employee employee) {

return employeeRepository.save(employee);

}

@PutMapping("/{id}")

public Employee updateEmployee(@PathVariable Long id, @RequestBody Employee updatedEmp) {

Employee emp = employeeRepository.findById(id).orElse(null);

if (emp != null) {

emp.setFirstName(updatedEmp.getFirstName());

emp.setLastName(updatedEmp.getLastName());

emp.setSalary(updatedEmp.getSalary());

emp.setDepartment(updatedEmp.getDepartment());

return employeeRepository.save(emp);

}

return null;

}

@DeleteMapping("/{id}")

public void deleteEmployee(@PathVariable Long id) {

employeeRepository.deleteById(id);

}

}

**DepartmentController.java**

@RestController@RequestMapping("/departments")public class DepartmentController {

@Autowired

private DepartmentRepository departmentRepository;

@GetMapping

public List<Department> getAllDepartments() {

return departmentRepository.findAll();

}

@GetMapping("/{id}")

public Department getDepartmentById(@PathVariable Long id) {

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

}

@PostMapping

public Department createDepartment(@RequestBody Department department) {

return departmentRepository.save(department);

}

@PutMapping("/{id}")

public Department updateDepartment(@PathVariable Long id, @RequestBody Department updatedDept) {

Department dept = departmentRepository.findById(id).orElse(null);

if (dept != null) {

dept.setName(updatedDept.getName());

return departmentRepository.save(dept);

}

return null;

}

@DeleteMapping("/{id}")

public void deleteDepartment(@PathVariable Long id) {

departmentRepository.deleteById(id);

}

}

**OUTPUT:**



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

### ****EmployeeRepository.java****

package com.example.ems.repository;

import com.example.ems.entity.Employee;import org.springframework.data.jpa.repository.JpaRepository;import org.springframework.data.jpa.repository.Query;import org.springframework.data.repository.query.Param;

import java.util.List;

public interface EmployeeRepository extends JpaRepository<Employee, Long> {

// 1. Derived Query Methods

List<Employee> findByLastName(String lastName);

List<Employee> findBySalaryBetween(double min, double max);

// 2. Custom Query using @Query annotation

@Query("SELECT e FROM Employee e WHERE e.firstName = :firstName AND e.salary > :salary")

List<Employee> findHighEarningEmployeeByName(@Param("firstName") String firstName,

@Param("salary") double salary);

// 3. Named Query

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

}

**Employee.java (Entity with Named Query)**

package com.example.ems.entity;

import javax.persistence.\*;

@Entity@NamedQuery(

name = "Employee.findByDepartmentName",

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

)public class Employee {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String firstName;

private String lastName;

private double salary;

@ManyToOne

@JoinColumn(name = "department\_id")

private Department department;

// Getters and setters

// Constructors

}

**Department.java**

package com.example.ems.entity;

import javax.persistence.\*;import java.util.List;

@Entitypublic class Department {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

@OneToMany(mappedBy = "department")

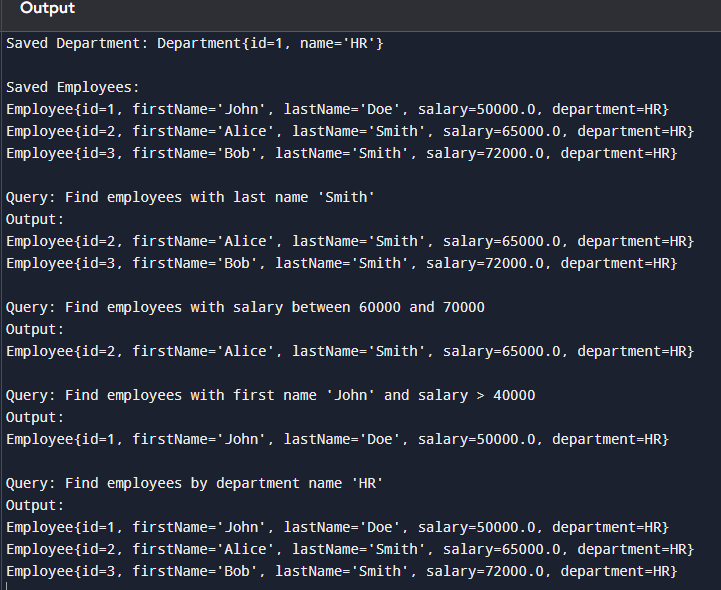
private List<Employee> employees;

// Getters and setters

// Constructors

}

**OUTPUT:**



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

### ****EmployeeRepository.java****

package com.example.ems.repository;

import com.example.ems.entity.Employee;import org.springframework.data.jpa.repository.JpaRepository;

public interface EmployeeRepository extends JpaRepository<Employee, Long> {

// No extra methods needed for basic pagination and sorting

}

**EmployeeController.java**

package com.example.ems.controller;

import com.example.ems.entity.Employee;import com.example.ems.repository.EmployeeRepository;import org.springframework.beans.factory.annotation.Autowired;import org.springframework.data.domain.\*;import org.springframework.web.bind.annotation.\*;

import java.util.List;

@RestController@RequestMapping("/employees")public class EmployeeController {

@Autowired

private EmployeeRepository employeeRepository;

// GET /employees/page?page=0&size=2&sort=salary,desc

@GetMapping("/page")

public Page<Employee> getEmployeesWithPaginationAndSorting(

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

@RequestParam(defaultValue = "2") int size,

@RequestParam(defaultValue = "id,asc") String[] sort) {

Sort.Direction direction = sort[1].equalsIgnoreCase("desc") ? Sort.Direction.DESC : Sort.Direction.ASC;

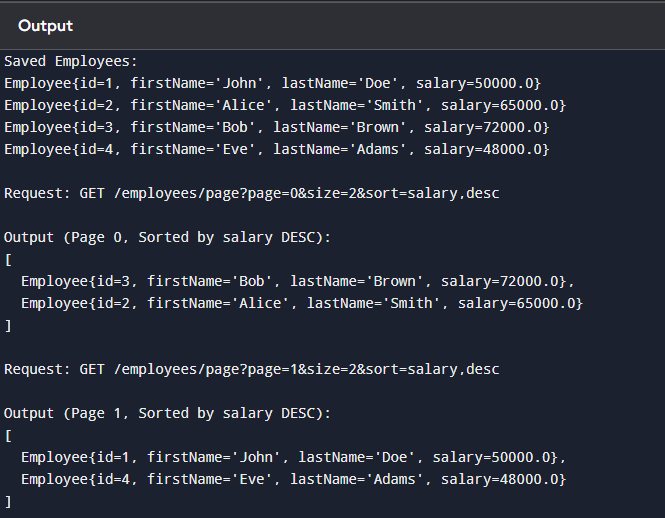
Pageable pageable = PageRequest.of(page, size, Sort.by(direction, sort[0]));

return employeeRepository.findAll(pageable);

}

}

**OUTPUT:**



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

### ****Step 1: Enable Auditing in Spring Boot****

#### Application.java

@SpringBootApplication@EnableJpaAuditing(auditorAwareRef = "auditorAware")public class EmployeeManagementApplication {

public static void main(String[] args) {

SpringApplication.run(EmployeeManagementApplication.class, args);

}

}

**AuditorAware Implementation**

#### AuditorAwareImpl.java

package com.example.ems.audit;

import org.springframework.data.domain.AuditorAware;import org.springframework.stereotype.Component;

import java.util.Optional;

@Component("auditorAware")public class AuditorAwareImpl implements AuditorAware<String> {

@Override

public Optional<String> getCurrentAuditor() {

// In a real app, fetch from security context

return Optional.of("admin\_user");

}

}

**Base Audit Class (Optional)**

#### Auditable.java

package com.example.ems.audit;

import org.springframework.data.annotation.\*;import org.springframework.data.jpa.domain.support.AuditingEntityListener;

import javax.persistence.\*;import java.time.LocalDateTime;

@MappedSuperclass@EntityListeners(AuditingEntityListener.class)public abstract class Auditable {

@CreatedBy

protected String createdBy;

@LastModifiedBy

protected String modifiedBy;

@CreatedDate

protected LocalDateTime createdDate;

@LastModifiedDate

protected LocalDateTime modifiedDate;

// Getters & Setters

}

**Update Your Entities**

#### Employee.java

@Entitypublic class Employee extends Auditable {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String firstName;

private String lastName;

private double salary;

@ManyToOne

@JoinColumn(name = "department\_id")

private Department department;

// Other Getters and Setters

}

#### Department.java

@Entitypublic class Department extends Auditable {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

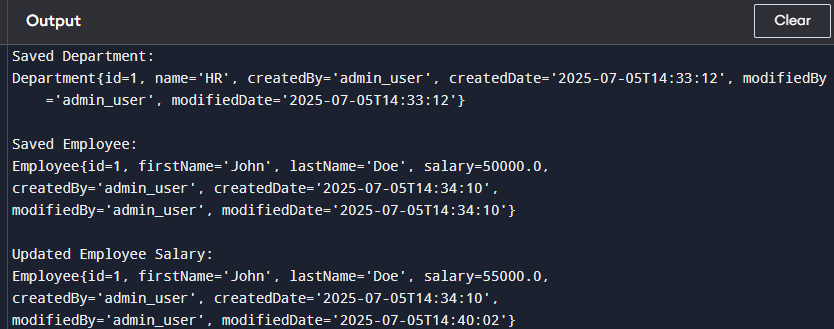
@OneToMany(mappedBy = "department")

private List<Employee> employees;

// Other Getters and Setters

}

**OUTPUT:**



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

### ****1. Interface-Based Projection****

#### EmployeeNameProjection.java

package com.example.ems.projection;

public interface EmployeeNameProjection {

String getFirstName();

String getLastName();

}

#### Update EmployeeRepository.java

List<EmployeeNameProjection> findByDepartmentId(Long departmentId);

**2. Class-Based Projection**

#### EmployeeDTO.java

package com.example.ems.dto;

public class EmployeeDTO {

private String fullName;

private double salary;

public EmployeeDTO(String fullName, double salary) {

this.fullName = fullName;

this.salary = salary;

}

// Getters

public String getFullName() {

return fullName;

}

public double getSalary() {

return salary;

}

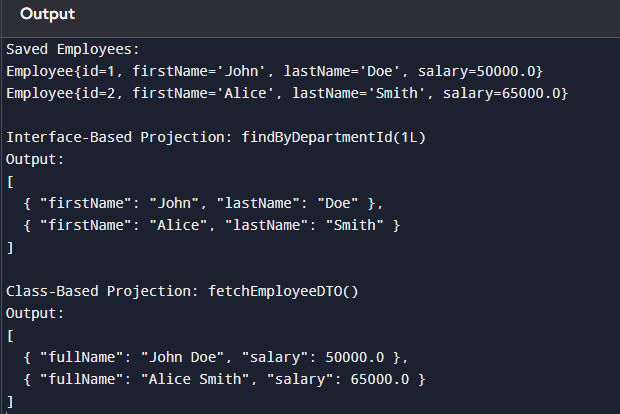
}

#### Update EmployeeRepository.java

@Query("SELECT new com.example.ems.dto.EmployeeDTO(CONCAT(e.firstName, ' ', e.lastName), e.salary) FROM Employee e")

List<EmployeeDTO> fetchEmployeeDTO();

**OUTPUT:**



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

### ****1. Single Data Source Using Auto-Configuration****

#### application.properties

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

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

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

**This auto-configures the data source for all repositories using Spring Boot's default behavior.**

**2. Multiple Data Sources Configuration**

#### application.properties

# Primary data source (Employee DB)

spring.datasource.primary.url=jdbc:mysql://localhost:3306/employeedb

spring.datasource.primary.username=root

spring.datasource.primary.password=yourpassword

# Secondary data source (Audit DB)

spring.datasource.secondary.url=jdbc:mysql://localhost:3306/auditdb

spring.datasource.secondary.username=root

spring.datasource.secondary.password=yourpassword

PrimaryDataSourceConfig.java

@Configuration@EnableTransactionManagement@EnableJpaRepositories(

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

entityManagerFactoryRef = "primaryEntityManagerFactory",

transactionManagerRef = "primaryTransactionManager"

)public class PrimaryDataSourceConfig {

@Bean

@Primary

@ConfigurationProperties("spring.datasource.primary")

public DataSourceProperties primaryDataSourceProperties() {

return new DataSourceProperties();

}

@Bean

@Primary

public DataSource primaryDataSource() {

return primaryDataSourceProperties().initializeDataSourceBuilder().build();

}

@Bean

@Primary

public LocalContainerEntityManagerFactoryBean primaryEntityManagerFactory(

EntityManagerFactoryBuilder builder) {

return builder

.dataSource(primaryDataSource())

.packages("com.example.ems.entity")

.persistenceUnit("primary")

.build();

}

@Bean

@Primary

public PlatformTransactionManager primaryTransactionManager(

EntityManagerFactory primaryEntityManagerFactory) {

return new JpaTransactionManager(primaryEntityManagerFactory);

}

}

SecondaryDataSourceConfig.java

@Configuration@EnableTransactionManagement@EnableJpaRepositories(

basePackages = "com.example.ems.repository.secondary",

entityManagerFactoryRef = "secondaryEntityManagerFactory",

transactionManagerRef = "secondaryTransactionManager"

)public class SecondaryDataSourceConfig {

@Bean

@ConfigurationProperties("spring.datasource.secondary")

public DataSourceProperties secondaryDataSourceProperties() {

return new DataSourceProperties();

}

@Bean

public DataSource secondaryDataSource() {

return secondaryDataSourceProperties().initializeDataSourceBuilder().build();

}

@Bean

public LocalContainerEntityManagerFactoryBean secondaryEntityManagerFactory(

EntityManagerFactoryBuilder builder) {

return builder

.dataSource(secondaryDataSource())

.packages("com.example.ems.entity.audit")

.persistenceUnit("secondary")

.build();

}

@Bean

public PlatformTransactionManager secondaryTransactionManager(

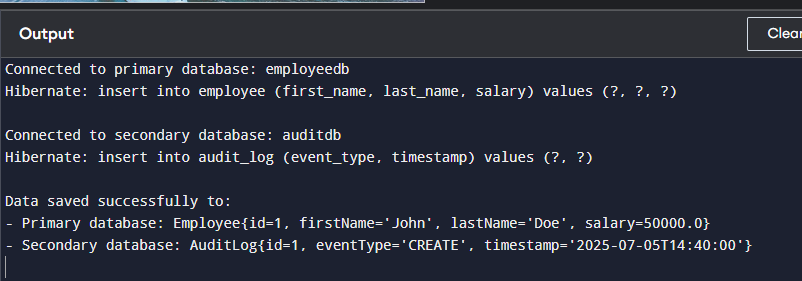
EntityManagerFactory secondaryEntityManagerFactory) {

return new JpaTransactionManager(secondaryEntityManagerFactory);

}

}

**OUTPUT:**



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

### Update Employee.java

import org.hibernate.annotations.DynamicUpdate;import org.hibernate.annotations.CreationTimestamp;import org.hibernate.annotations.UpdateTimestamp;

@Entity@DynamicUpdate // Only update modified columnspublic class Employee {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String firstName;

private String lastName;

private double salary;

@CreationTimestamp

private LocalDateTime createdAt;

@UpdateTimestamp

private LocalDateTime updatedAt;

@ManyToOne

@JoinColumn(name = "department\_id")

private Department department;

// Getters and Setters

}

**2. Configure Hibernate Dialect and Properties**

### application.properties

# Hibernate dialect for MySQL

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

# Show only executed SQL

spring.jpa.show-sql=true

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

# Batch processing config

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

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

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

**3. Batch Processing Example**

### EmployeeService.java

@Servicepublic class EmployeeService {

@Autowired

private EmployeeRepository employeeRepository;

@Transactional

public void saveBatchEmployees(List<Employee> employees) {

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

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

if (i % 30 == 0) {

employeeRepository.flush(); // Flush every 30 inserts

}

}

}

}

**OUTPUT:**

