**Hands on 7**

**Add a new country**

* Create new method in CountryService.

@Transactional

public void addCountry(Country country)

* Invoke save() method of repository to get the country added.

countryRepository.save(country)

* Include new testAddCountry() method in OrmLearnApplication. Perform steps below:
  + Create new instance of country with a new code and name
  + Call countryService.addCountry() passing the country created in the previous step.
  + Invoke countryService.findCountryByCode() passing the same code used when adding a new country
  + Check in the database if the country is added

**Objective**: Add a new country record using Spring Data JPA.

**Steps**:

1. **Service Method Created**:

@Transactional

public void addCountry(Country country) {

countryRepository.save(country);

}

1. **Test Method Added in OrmLearnApplication**:

private static void testAddCountry() {

LOGGER.info("Start");

Country newCountry = new Country();

newCountry.setCode("JP");

newCountry.setName("Japan");

countryService.addCountry(newCountry);

Country country = countryService.findCountryByCode("JP");

LOGGER.debug("Added Country: {}", country);

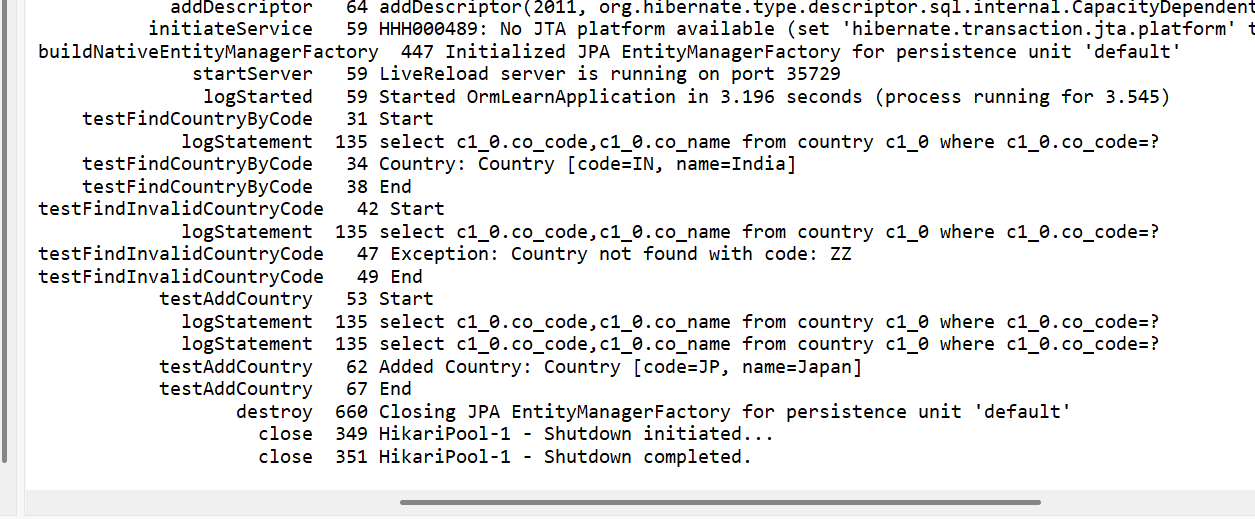
LOGGER.info("End");

}

1. **Result**:

Country "JP - Japan" successfully added and retrieved. Verified in the MySQL database.

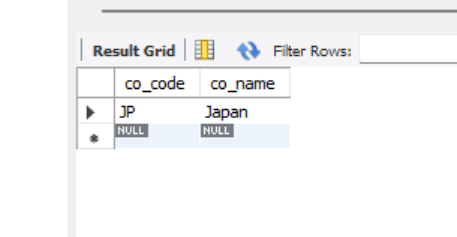
**This is my console output:**



**This is my sql output:**

Verifying in the database.

SELECT \* FROM country WHERE co\_code = 'JP';



**Demonstrate implementation of Query Methods feature of Spring Data JPA**

**For setting up the project:**

Technologies used:

* Spring Boot
* Spring Data JPA
* H2 (in-memory DB)
* Lombok

1. **Create Entity - Employee.java**

package com.example.demo.entity;

import jakarta.persistence.\*;

import lombok.\*;

import java.time.LocalDate;

@Entity

@Data

@NoArgsConstructor

@AllArgsConstructor

public class Employee {

@Id

@GeneratedValue(strategy = GenerationType.*IDENTITY*)

private Long id;

private String name;

private Double salary;

private LocalDate joinDate;

}

1. **Create Repository - EmployeeRepository.java**

package com.example.demo.repository;

import com.example.demo.entity.Employee;

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

import java.time.LocalDate;

import java.util.List;

public interface EmployeeRepository extends JpaRepository<Employee, Long> {

// 1. Search by containing text

List<Employee> findByNameContainingIgnoreCase(String namePart);

// 2. Sorting by name ascending

List<Employee> findAllByOrderByNameAsc();

// 3. Filter with starting text

List<Employee> findByNameStartingWithIgnoreCase(String prefix);

// 4. Fetch between two dates

List<Employee> findByJoinDateBetween(LocalDate start, LocalDate end);

// 5. Greater than or less than salary

List<Employee> findBySalaryGreaterThan(Double salary);

List<Employee> findBySalaryLessThan(Double salary);

// 6. Top-N highest salaries

List<Employee> findTop3ByOrderBySalaryDesc();

}

1. **Insert Sample Data and Test Queries - DemoApplication.java**

package com.example.demo;

import com.example.demo.entity.Employee;

import com.example.demo.repository.EmployeeRepository;

import org.springframework.boot.CommandLineRunner;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.context.annotation.Bean;

import java.time.LocalDate;

import java.util.List;

@SpringBootApplication

public class DemoApplication {

public static void main(String[] args) {

SpringApplication.*run*(DemoApplication.class, args);

}

@Bean

CommandLineRunner run(EmployeeRepository repo) {

return args -> {

// Adding sample data

repo.saveAll(List.*of*(

new Employee(null, "Alice", 50000.0, LocalDate.*of*(2020, 1, 10)),

new Employee(null, "Bob", 60000.0, LocalDate.*of*(2021, 3, 15)),

new Employee(null, "Charlie", 70000.0, LocalDate.*of*(2022, 7, 5)),

new Employee(null, "David", 45000.0, LocalDate.*of*(2023, 5, 20)),

new Employee(null, "Diana", 80000.0, LocalDate.*of*(2021, 11, 30)),

new Employee(null, "Daniel", 55000.0, LocalDate.*of*(2022, 9, 1))

));

// 1. Search by text

System.*out*.println("Employees containing 'a':");

repo.findByNameContainingIgnoreCase("a")

.forEach(e -> System.*out*.println(e.getName()));

// 2. Sorting by name ascending

System.*out*.println("\n Employees sorted by name:");

repo.findAllByOrderByNameAsc()

.forEach(e -> System.*out*.println(e.getName()));

// 3. Starting with 'D'

System.*out*.println("\n Employees starting with 'D':");

repo.findByNameStartingWithIgnoreCase("D")

.forEach(e -> System.*out*.println(e.getName()));

// 4. Between two dates

System.*out*.println("\n Employees joined between 2021-01-01 and 2022-12-31:");

repo.findByJoinDateBetween(LocalDate.*of*(2021, 1, 1), LocalDate.*of*(2022, 12, 31))

.forEach(e -> System.*out*.println(e.getName() + " - " + e.getJoinDate()));

// 5. Salary greater than 55000

System.*out*.println("\n Employees with salary > 55000:");

repo.findBySalaryGreaterThan(55000.0)

.forEach(e -> System.*out*.println(e.getName() + " - ₹" + e.getSalary()));

// 6. Salary less than 55000

System.*out*.println("\n Employees with salary < 55000:");

repo.findBySalaryLessThan(55000.0)

.forEach(e -> System.*out*.println(e.getName() + " - ₹" + e.getSalary()));

// 7. Top 3 salaries

System.*out*.println("\n Top 3 highest paid employees:");

repo.findTop3ByOrderBySalaryDesc()

.forEach(e -> System.*out*.println(e.getName() + " - ₹" + e.getSalary()));

};

}

}

**Output:**

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

**Demonstrate implementation of O/R Mapping**

We’ll create 3 entities:

1. Department
2. Employee
3. Project

With these relationships:

| **Relationship** | **Example** |
| --- | --- |
| @ManyToOne | Many Employees → One Department |
| @OneToMany | One Department → Many Employees |
| @ManyToMany | Many Employees ↔ Many Projects |
|  |  |

1. **Department.java(entity class)**

package com.example.demo.entity;

import jakarta.persistence.\*;

import lombok.\*;

import java.util.List;

@Entity

@Data

@NoArgsConstructor

@AllArgsConstructor

public class Department {

@Id

@GeneratedValue(strategy = GenerationType.*IDENTITY*)

private Long id;

private String name;

// One department has many employees

@OneToMany(mappedBy = "department", fetch = FetchType.*LAZY*)

private List<Employee> employees;

}

1. **Employee.java(entity class)**

package com.example.demo.entity;

import jakarta.persistence.\*;

import lombok.\*;

import java.time.LocalDate;

import java.util.List;

@Entity

@Data

@NoArgsConstructor

@AllArgsConstructor

public class Employee {

@Id

@GeneratedValue(strategy = GenerationType.*IDENTITY*)

private Long id;

private String name;

private Double salary;

private LocalDate joinDate;

// Many employees belong to one department

@ManyToOne(fetch = FetchType.*EAGER*)

@JoinColumn(name = "department\_id")

private Department department;

// Many employees can work on many projects

@ManyToMany(fetch = FetchType.*EAGER*) // Optional for testing

@JoinTable(

name = "employee\_project",

joinColumns = @JoinColumn(name = "employee\_id"),

inverseJoinColumns = @JoinColumn(name = "project\_id")

)

private List<Project> projects;

}

1. **Project.java(entity class)**

package com.example.demo.entity;

import jakarta.persistence.\*;

import lombok.\*;

import java.util.List;

@Entity

@Data

@NoArgsConstructor

@AllArgsConstructor

public class Project {

@Id

@GeneratedValue(strategy = GenerationType.*IDENTITY*)

private Long id;

private String title;

// Many employees can work on this project

@ManyToMany(mappedBy = "projects")

private List<Employee> employees;

}

1. **EmployeeRepository.java (repository class)**

package com.example.demo.repository;

import com.example.demo.entity.Employee;

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

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

import java.time.LocalDate;

import java.util.List;

public interface EmployeeRepository extends JpaRepository<Employee, Long> {

// 1. Search by containing text

List<Employee> findByNameContainingIgnoreCase(String namePart);

// 2. Sorting by name ascending

List<Employee> findAllByOrderByNameAsc();

// 3. Filter with starting text

List<Employee> findByNameStartingWithIgnoreCase(String prefix);

// 4. Fetch between two dates

List<Employee> findByJoinDateBetween(LocalDate start, LocalDate end);

// 5. Greater than or less than salary

List<Employee> findBySalaryGreaterThan(Double salary);

List<Employee> findBySalaryLessThan(Double salary);

// 6. Top-N highest salaries

List<Employee> findTop3ByOrderBySalaryDesc();

// Eagerly fetch projects and department with employees

@EntityGraph(attributePaths = {"projects", "department"})

List<Employee> findAll();

}

1. **DepartmentRepository.java (repository class)**

package com.example.demo.repository;

import com.example.demo.entity.Department;

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

public interface DepartmentRepository extends JpaRepository<Department, Long> {

}

1. **ProjectRepository.java (repository class)**

package com.example.demo.repository;

import com.example.demo.entity.Project;

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

public interface ProjectRepository extends JpaRepository<Project, Long> {

}

1. **DemoApplication.java (main class)**

package com.example.demo;

import com.example.demo.entity.Employee;

import com.example.demo.entity.Department;

import com.example.demo.entity.Project;

import com.example.demo.repository.EmployeeRepository;

import com.example.demo.repository.DepartmentRepository;

import com.example.demo.repository.ProjectRepository;

import org.springframework.boot.CommandLineRunner;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.context.annotation.Bean;

import java.time.LocalDate;

import java.util.List;

@SpringBootApplication

public class DemoApplication {

public static void main(String[] args) {

SpringApplication.*run*(DemoApplication.class, args);

}

@Bean

CommandLineRunner run(EmployeeRepository employeeRepo,

DepartmentRepository departmentRepo,

ProjectRepository projectRepo) {

return args -> {

// Save departments

Department dev = departmentRepo.save(new Department(null, "Development", null));

Department hr = departmentRepo.save(new Department(null, "HR", null));

// Save projects

Project p1 = projectRepo.save(new Project(null, "Spring Boot App", null));

Project p2 = projectRepo.save(new Project(null, "Cloud Migration", null));

// Save employees

employeeRepo.saveAll(List.*of*(

new Employee(null, "Alice", 50000.0, LocalDate.*of*(2020, 1, 10), dev, List.*of*(p1, p2)),

new Employee(null, "Bob", 60000.0, LocalDate.*of*(2021, 3, 15), dev, List.*of*(p2)),

new Employee(null, "Diana", 80000.0, LocalDate.*of*(2021, 11, 30), hr, List.*of*(p1))

));

// Display all employees with department and projects

System.*out*.println("Employees with departments and projects:");

employeeRepo.findAll().forEach(e -> {

System.*out*.println(e.getName() + " - " + e.getDepartment().getName());

e.getProjects().forEach(p -> System.*out*.println(" Project: " + p.getTitle()));

});

};

}

}

**Output:**

A screenshot of a computer

AI-generated content may be incorrect.

**Demonstrate writing Hibernate Query Language and Native Query**

**HQL vs JPQL**

| Feature | HQL (Hibernate Query Language) | JPQL (Java Persistence Query Language) |
| --- | --- | --- |
| Based On | Hibernate ORM model | JPA entity model |
| Syntax | Object-oriented | Object-oriented |
| Vendor | Hibernate-specific | JPA-standard |
| Portability | Not portable across all JPA providers | Portable across all compliant JPA providers |
| Keyword usage | HQL may have extra features like fetch | JPQL is standardized; fetch used with joins |

**Employee.java(entity class)**

package com.example.demo.entity;

import jakarta.persistence.\*;

import java.time.LocalDate;

import java.util.List;

@Entity

public class Employee {

@Id

@GeneratedValue(strategy = GenerationType.*IDENTITY*)

private Long id;

private String name;

private Double salary;

private LocalDate dateOfJoining;

@ManyToOne

private Department department;

@ManyToMany

@JoinTable(

name = "employee\_project",

joinColumns = @JoinColumn(name = "employee\_id"),

inverseJoinColumns = @JoinColumn(name = "project\_id")

)

private List<Project> projects;

public Employee() {}

public Employee(Long id, String name, Double salary, LocalDate dateOfJoining, Department department) {

this.id = id;

this.name = name;

this.salary = salary;

this.dateOfJoining = dateOfJoining;

this.department = department;

}

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 Double getSalary() {

return salary;

}

public void setSalary(Double salary) {

this.salary = salary;

}

public LocalDate getDateOfJoining() {

return dateOfJoining;

}

public void setDateOfJoining(LocalDate dateOfJoining) {

this.dateOfJoining = dateOfJoining;

}

public Department getDepartment() {

return department;

}

public void setDepartment(Department department) {

this.department = department;

}

public List<Project> getProjects() {

return projects;

}

public void setProjects(List<Project> projects) {

this.projects = projects;

}

}

**Department.java (entity class)**

package com.example.demo.entity;

import jakarta.persistence.\*;

import java.util.List;

@Entity

public class Department {

@Id

@GeneratedValue(strategy = GenerationType.*IDENTITY*)

private Long id;

private String name;

@OneToMany(mappedBy = "department")

private List<Employee> employees;

public Department() {}

public Department(Long id, String name, List<Employee> employees) {

this.id = id;

this.name = name;

this.employees = employees;

}

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 List<Employee> getEmployees() {

return employees;

}

public void setEmployees(List<Employee> employees) {

this.employees = employees;

}

}

**EmployeeRepository,java**

package com.example.demo.repository;

import com.example.demo.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> {

// HQL Query

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

List<Employee> findEmployeesWithSalaryGreaterThan(@Param("salary") Double salary);

// Fetch Join HQL

@Query("SELECT e FROM Employee e JOIN FETCH e.department")

List<Employee> findAllWithDepartmentFetch();

// Aggregate Function

@Query("SELECT AVG(e.salary) FROM Employee e")

Double getAverageSalary();

// Native Query

@Query(value = "SELECT \* FROM employee WHERE name LIKE %:name%", nativeQuery = true)

List<Employee> searchByNameNative(@Param("name") String name);

@Query(value = "SELECT MAX(salary) FROM employee", nativeQuery = true)

Double getMaxSalaryNative();

}

**DemoApplication.java**

package com.example.demo;

import com.example.demo.entity.Employee;

import com.example.demo.entity.Department;

import com.example.demo.entity.Project;

import com.example.demo.repository.EmployeeRepository;

import com.example.demo.repository.DepartmentRepository;

import com.example.demo.repository.ProjectRepository;

import org.springframework.boot.CommandLineRunner;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.context.annotation.Bean;

import java.time.LocalDate;

import java.util.List;

@SpringBootApplication

public class DemoApplication {

public static void main(String[] args) {

SpringApplication.*run*(DemoApplication.class, args);

}

@Bean

CommandLineRunner demo(EmployeeRepository employeeRepo, DepartmentRepository deptRepo, ProjectRepository projectRepo) {

return args -> {

Department it = deptRepo.save(new Department(null, "IT", null));

Department hr = deptRepo.save(new Department(null, "HR", null));

Project p1 = projectRepo.save(new Project(null, "Project A"));

Project p2 = projectRepo.save(new Project(null, "Project B"));

Employee e1 = new Employee(null, "Alice", 50000.0, LocalDate.*of*(2021, 1, 10), it);

Employee e2 = new Employee(null, "Bob", 60000.0, LocalDate.*of*(2022, 5, 12), it);

Employee e3 = new Employee(null, "Charlie", 45000.0, LocalDate.*of*(2023, 3, 1), hr);

e1.setProjects(List.*of*(p1));

e2.setProjects(List.*of*(p1, p2));

e3.setProjects(List.*of*(p2));

employeeRepo.saveAll(List.*of*(e1, e2, e3));

System.*out*.println("Employees with salary > 50000:");

employeeRepo.findEmployeesWithSalaryGreaterThan(50000.0)

.forEach(e -> System.*out*.println(e.getName()));

System.*out*.println("Average Salary: " + employeeRepo.getAverageSalary());

System.*out*.println("Max Salary (Native): " + employeeRepo.getMaxSalaryNative());

System.*out*.println("Employees with Fetch Join (Departments):");

employeeRepo.findAllWithDepartmentFetch()

.forEach(e -> System.*out*.println(e.getName() + " - " + e.getDepartment().getName()));

System.*out*.println("Search by name (Native):");

employeeRepo.searchByNameNative("a")

.forEach(e -> System.*out*.println(e.getName()));

};

}

}

**Output**

**A screenshot of a computer

AI-generated content may be incorrect.**