1. **Spring Data JPA HandsOn**

**OBJECTIVE QUESTIONS:**

1. **Explain the need and benefit of ORM**

**What is ORM?**ORM (Object-Relational Mapping) is a technique that connects Java objects (classes) with relational database tables. It allows developers to work with databases using Java code instead of SQL.

**Need for ORM:**

* + Writing SQL manually is error-prone and repetitive.
  + Mapping between Java objects and tables manually is complex.
  + Database operations should be abstracted for easier maintenance.

**Benefits of ORM:**

* + Simplifies database access using objects.
  + Reduces boilerplate SQL code.
  + Automatically maps relationships (e.g., one-to-many).
  + Enables database independence.
  + Handles transactions and caching.

1. **Demonstrate the need and benefit of Spring Data JPA**

Spring Data JPA is a Spring-based framework built on top of JPA and Hibernate.

**Need:**

* + JPA requires a lot of boilerplate code (like writing repositories and queries).
  + Developers want a faster way to perform CRUD operations.

**Benefits:**

* + Removes the need to write DAO implementations.
  + Automatically generates repository methods (e.g., findByName).
  + Supports pagination and sorting.
  + Integrates easily with Spring Boot and Hibernate.
  + Reduces time and errors in database operations.

1. **Explain about core objects of hibernate framework**

| **Core Object** | **Purpose** |
| --- | --- |
| Session Factory | A factory for creating Session instances. Usually one per application. |
| Session | Main interface for interacting with the database. Manages CRUD operations. |
| Transaction | Represents a unit of work. Allows commit/rollback. |
| Connection Provider | Provides database connections used internally by Hibernate. |

1. **Explain ORM implementation with Hibernate XML Configuration and Annotation Configuration**

**a) XML Configuration:**

* + Mapping is defined in external XML files.
  + Used in older Hibernate applications.
  + Example files: hibernate.cfg.xml, \*.hbm.xml

**b) Annotation Configuration:**

* + Uses Java annotations (@Entity, @Table, @Column, etc.)
  + Cleaner, easier to manage.
  + Popular in Spring applications.

**Steps in both cases:**

* + Define persistence class (e.g., Employee.java).
  + Map fields to DB columns.
  + Configure DB connection.
  + Load session factory and perform DB operations.

1. **Explain the difference between Java Persistence API, Hibernate and Spring Data JPA**

| **Feature** | **JPA** | **Hibernate** | **Spring Data JPA** |
| --- | --- | --- | --- |
| **Type** | Specification | Implementation of JPA | Abstraction over JPA |
| **Developed By** | Java EE | Red Hat | Spring Framework |
| **Purpose** | Defines API for ORM | Actual ORM engine | Reduces boilerplate for JPA |
| **Features** | @Entity, @Id, etc. | Session, advanced caching, dialects | Repository interfaces like findById, save, etc. |
| **Code Style** | Standard | Hibernate-specific APIs | Interface-driven, annotation-based |

1. **Demonstrate implementation of DML using Spring Data JPA on a single database table**

**DML Operations (Data Manipulation Language):**

* + **Insert:** save(entity)
  + **Update:** save(modifiedEntity)
  + **Delete:** deleteById(id)
  + **Read:** findAll(), findById(id)

**SPRING DATA JPA**

// Insert or Update

countryRepository.save(new Country("IN", "India"));

// Delete

countryRepository.deleteById("IN");

// Read

Optional<Country> c = countryRepository.findById("IN");

List<Country> list = countryRepository.findAll();

**Hands on 1**

**Spring Data JPA - Quick Example**

***File name: Country.java***

package com.cognizant.orm\_learn.model;

import jakarta.persistence.\*;

@Entity

@Table(name = "country")

public class Country {

@Id

@Column(name = "co\_code")

private String code;

@Column(name = "co\_name")

private String name;

public String getCode() {

return code;

}

public void setCode(String code) {

this.code = code;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

@Override

public String toString() {

return "Country [code=" + code + ", name=" + name + "]";

}

}

***File name: CountryRepository.java***

package com.cognizant.orm\_learn.repository;

import com.cognizant.orm\_learn.model.Country;

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

import org.springframework.stereotype.Repository;

@Repository

public interface CountryRepository extends JpaRepository<Country, String> {

}

***File name: CountryService.java***

package com.cognizant.orm\_learn.service;

import com.cognizant.orm\_learn.model.Country;

import com.cognizant.orm\_learn.repository.CountryRepository;

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 CountryService {

@Autowired

private CountryRepository countryRepository;

@Transactional

public List<Country> getAllCountries() {

return countryRepository.findAll();

}

}

***File name:* OrmLearnApplication*.java***

package com.cognizant.orm\_learn;

import com.cognizant.orm\_learn.model.Country;

import com.cognizant.orm\_learn.service.CountryService;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

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

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.context.ApplicationContext;

import java.util.List;

@SpringBootApplication

public class OrmLearnApplication {

private static CountryService *countryService*;

private static final Logger *LOGGER* = LoggerFactory.*getLogger*(OrmLearnApplication.class);

public static void main(String[] args) {

ApplicationContext context = SpringApplication.*run*(OrmLearnApplication.class, args);

*countryService* = context.getBean(CountryService.class);

*testGetAllCountries*();

}

private static void testGetAllCountries() {

*LOGGER*.info("Start");

List<Country> countries = *countryService*.getAllCountries();

*LOGGER*.debug("countries={}", countries);

*LOGGER*.info("End");

}

}

***File name:* application.properties**

# Database connection

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

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

spring.datasource.username=root

spring.datasource.password=Candy@07

# Hibernate config (Hibernate 6+)

spring.jpa.hibernate.ddl-auto=validate

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

# Logging

logging.level.org.springframework=info

logging.level.com.cognizant=debug

logging.level.org.hibernate.SQL=trace

logging.level.org.hibernate.type.descriptor.sql=trace

***SQL FILE***

CREATE DATABASE ormlearn;

USE ormlearn;

CREATE TABLE country (

co\_code VARCHAR(2) PRIMARY KEY,

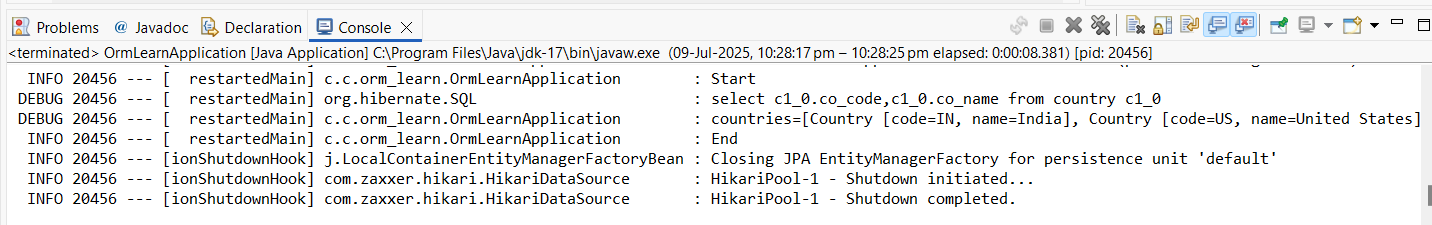
co\_name VARCHAR(50)

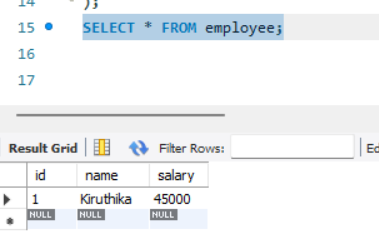
);

INSERT INTO country VALUES ('IN', 'India'), ('US', 'United States');

SELECT \* FROM country;

**OUTPUT**





**Hands on 2**

**Hibernate XML Config implementation walk through**

***File name:* Employee.java**

package com.example.hibernatexml;

public class Employee {

private int id;

private String name;

private double salary;

// Getters & setters

public int getId() { return id; }

public void setId(int 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; }

}

***File name:* Main.java**

package com.example.hibernatexml;

import org.hibernate.Session;

import org.hibernate.SessionFactory;

import org.hibernate.Transaction;

import org.hibernate.cfg.Configuration;

import java.util.List;

public class Main {

public static void main(String[] args) {

Configuration cfg = new Configuration().configure(); // loads hibernate.cfg.xml

SessionFactory factory = cfg.buildSessionFactory();

Session session = factory.openSession();

// 1. INSERT with commit()

try {

Transaction tx = session.beginTransaction();

Employee emp = new Employee();

emp.setName("Kiruthika");

emp.setSalary(55000);

session.save(emp);

tx.commit();

System.*out*.println(" Record inserted and committed.");

} catch (Exception e) {

System.*out*.println(" Insert failed: " + e.getMessage());

}

// 2. SELECT all using createQuery().list()

try {

List<Employee> list = session.createQuery("from Employee", Employee.class).list();

System.*out*.println(" Employee List:");

for (Employee e : list) {

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

}

} catch (Exception e) {

System.*out*.println("Error in list: " + e.getMessage());

}

// 3. FETCH using session.get()

try {

Employee e = session.get(Employee.class, 1); // fetch by ID = 1

if (e != null) {

System.*out*.println(" Fetched: " + e.getName() + " - " + e.getSalary());

} else {

System.*out*.println(" No employee found with ID 1.");

}

} catch (Exception e) {

System.*out*.println("Get failed: " + e.getMessage());

}

// DELETE with rollback demo

try {

Transaction tx = session.beginTransaction();

Employee empToDelete = session.get(Employee.class, 1);

if (empToDelete != null) {

session.delete(empToDelete);

// Uncomment next line to simulate error:

// int error = 10 / 0;

tx.commit();

System.*out*.println("Record deleted and committed.");

} else {

System.*out*.println("No employee found to delete.");

tx.rollback(); // nothing to delete

}

} catch (Exception e) {

System.*out*.println("Error during delete, rolling back: " + e.getMessage());

session.getTransaction().rollback();

}

session.close();

factory.close();

}

}

***File name:* employee.hbm.xml**

<?xml version="1.0"?>

<!DOCTYPE hibernate-mapping PUBLIC

"-//Hibernate/Hibernate Mapping DTD 3.0//EN"

"http://hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping>

<class name="com.example.hibernatexml.Employee" table="employee">

<id name="id" column="id">

<generator class="native"/>

</id>

<property name="name" column="name"/>

<property name="salary" column="salary"/>

</class>

</hibernate-mapping>

***File name:* hibernate.cfg.xml**

<?xml version='1.0' encoding='utf-8'?>

<!DOCTYPE hibernate-configuration PUBLIC

"-//Hibernate/Hibernate Configuration DTD 3.0//EN"

"http://hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

<session-factory>

<!-- DB Config -->

<property name="hibernate.connection.driver\_class">com.mysql.cj.jdbc.Driver</property>

<property name="hibernate.connection.url">jdbc:mysql://localhost:3306/ormlearn</property>

<property name="hibernate.connection.username">root</property>

<property name="hibernate.connection.password">Candy@07</property>

<!-- Hibernate Dialect -->

<property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>

<!-- Show SQL -->

<property name="hibernate.show\_sql">true</property>

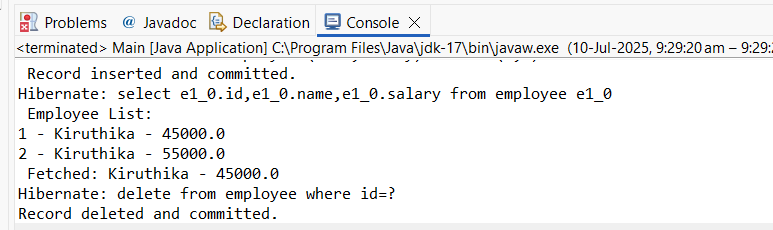
<!-- Mapping File -->

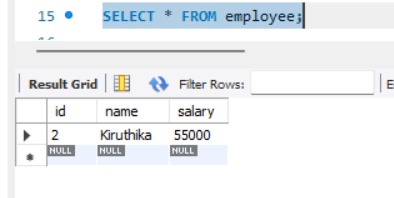
<mapping resource="employee.hbm.xml"/>

</session-factory>

</hibernate-configuration>

**OUTPUT**





**Hands on 3**

**Hibernate Annotation Config implementation walk through**

***File name:* Main.java**

package com.example.hibernatexml;

import org.hibernate.Session;

import org.hibernate.SessionFactory;

import org.hibernate.Transaction;

import org.hibernate.cfg.Configuration;

public class Main {

public static void main(String[] args) {

Configuration cfg = new Configuration().configure().addAnnotatedClass(Employee.class);

SessionFactory factory = cfg.buildSessionFactory();

Session session = factory.openSession();

// Insert a record

Transaction tx = session.beginTransaction();

Employee emp = new Employee();

emp.setName("Annotation User");

emp.setSalary(70000);

session.save(emp);

tx.commit();

System.*out*.println("Record inserted using annotation-based config.");

session.close();

factory.close();

}

}

***File name:* Employee.java**

package com.example.hibernatexml;

import jakarta.persistence.\*;

@Entity

@Table(name = "employee")

public class Employee {

@Id

@GeneratedValue(strategy = GenerationType.*IDENTITY*)

private int id;

@Column(name = "name")

private String name;

@Column(name = "salary")

private double salary;

// Getters and setters

public int getId() {

return id;

}

public void setId(int 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;

}

}

***File name:* hibernate.cfg.xml**

<?xml version='1.0' encoding='utf-8'?>

<!DOCTYPE hibernate-configuration PUBLIC

"-//Hibernate/Hibernate Configuration DTD 3.0//EN"

"http://hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

<session-factory>

<property name="hibernate.connection.driver\_class">com.mysql.cj.jdbc.Driver</property>

<property name="hibernate.connection.url">jdbc:mysql://localhost:3306/ormlearn</property>

<property name="hibernate.connection.username">root</property>

<property name="hibernate.connection.password">Candy@07</property>

<property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>

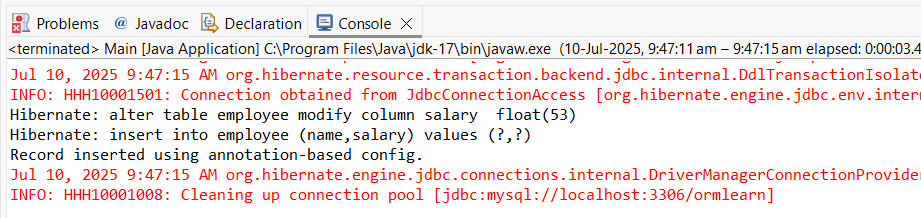
<property name="hibernate.hbm2ddl.auto">update</property>

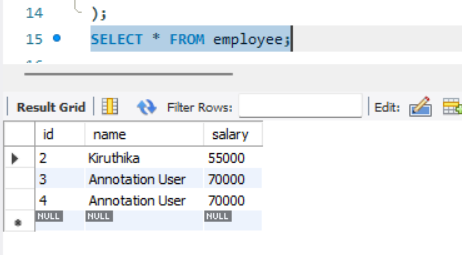
<property name="show\_sql">true</property>

</session-factory>

</hibernate-configuration>

**OUTPUT**





**Hands-on 4**

**Difference Between JPA, Hibernate, and Spring Data JPA**

**1. Java Persistence API (JPA)**

• JPA is a specification (JSR 338) for persisting, reading, and managing data from Java objects.

• It does not contain any implementation — it just defines how ORM should work.

• It provides interfaces like EntityManager, but no concrete implementation.

**2. Hibernate**

• Hibernate is a popular ORM tool that implements the JPA specification.

• It provides a complete ORM framework, handling caching, lazy loading, dirty checking, etc.

• You write more code compared to Spring Data JPA, but you have more control.

**3. Spring Data JPA**

• Spring Data JPA is a higher-level abstraction on top of JPA (and often Hibernate).

• It reduces boilerplate by using interfaces like JpaRepository.

• It handles query generation, transaction management, and simplifies data access.

**Hibernate Approach:**

public Integer addEmployee(Employee employee){

Session session = factory.openSession();

Transaction tx = null;

Integer employeeID = null;

try {

tx = session.beginTransaction();

employeeID = (Integer) session.save(employee);

tx.commit();

} catch (HibernateException e) {

if (tx != null) tx.rollback();

e.printStackTrace();

} finally {

session.close();

}

return employeeID;

}

**Spring Data JPA Approach:**

public interface EmployeeRepository extends JpaRepository<Employee, Integer> { }

@Autowired

private EmployeeRepository employeeRepository;

@Transactional

public void addEmployee(Employee employee) {

employeeRepository.save(employee);

}

| **Feature** | **JPA** | **Hibernate** | **Spring Data JPA** |
| --- | --- | --- | --- |
| **Type** | Specification (JSR 338) | ORM Framework | Abstraction Layer on top of JPA |
| **Ownership** | Oracle (Java Community Process) | Red Hat | Spring Framework Team |
| **Implementation** | No (Just API) | Yes (Implements JPA) | No (Uses JPA/Hibernate underneath) |
| **Boilerplate Code** | High | Medium | Very Low (auto implementation) |
| **Transactions** | Manual via EntityManager | Manual or Spring-managed | Automatic via @Transactional |
| **Query Writing** | JPQL (manual) | JPQL / HQL (manual) | Auto-generated via method names or custom @Query |
| **CRUD Operations** | Manually implemented | Manually written using Session | Provided via JpaRepository |
| **Ease of Use** | Moderate | Moderate to Advanced | Very Easy |
| **Learning Curve** | High | Medium | Low |
| **Popular Use Case** | Standard API layer | Direct ORM control and optimization | Rapid application development |

**Hands on 5**

**Implement services for managing Country**

***File name:* OrmLearnApplication.java**package com.cognizant.ormlearn;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.context.ApplicationContext;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import java.util.List;

import com.cognizant.ormlearn.model.Country;

import com.cognizant.ormlearn.service.CountryService;

import com.cognizant.ormlearn.service.exception.CountryNotFoundException;

@SpringBootApplication

public class OrmLearnApplication {

private static final Logger *LOGGER* = LoggerFactory.*getLogger*(OrmLearnApplication.class);

private static CountryService *countryService*;

public static void main(String[] args) {

ApplicationContext context = SpringApplication.*run*(OrmLearnApplication.class, args);

*countryService* = context.getBean(CountryService.class);

*testGetAllCountries*();

*testAddCountry*();

*testFindCountry*();

*testUpdateCountry*();

// testDeleteCountry();

*testSearchCountry*();

}

private static void testGetAllCountries() {

*LOGGER*.info("Start getAllCountries");

List<Country> countries = *countryService*.getAllCountries();

*LOGGER*.debug("Countries: {}", countries);

*LOGGER*.info("End getAllCountries");

}

private static void testAddCountry() {

*LOGGER*.info("Start addCountry");

Country country = new Country();

country.setCode("ZZ");

country.setName("Zootopia");

*countryService*.addCountry(country);

*LOGGER*.info("End addCountry");

}

private static void testFindCountry() {

*LOGGER*.info("Start findCountryByCode");

try {

Country country = *countryService*.findCountryByCode("ZZ");

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

} catch (CountryNotFoundException e) {

*LOGGER*.error("Error: {}", e.getMessage());

}

*LOGGER*.info("End findCountryByCode");

}

private static void testUpdateCountry() {

*LOGGER*.info("Start updateCountry");

try {

*countryService*.updateCountry("ZZ", "Zootopia Updated");

Country country = *countryService*.findCountryByCode("ZZ");

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

} catch (CountryNotFoundException e) {

*LOGGER*.error("Error: {}", e.getMessage());

}

*LOGGER*.info("End updateCountry");

}

// private static void testDeleteCountry() {

// LOGGER.info("Start deleteCountry");

// countryService.deleteCountry("ZZ");

//LOGGER.info("End deleteCountry");

//}

private static void testSearchCountry() {

*LOGGER*.info("Start searchCountryByName");

List<Country> countries = *countryService*.searchCountryByName("ind");

*LOGGER*.debug("Search Result: {}", countries);

*LOGGER*.info("End searchCountryByName");

}

}

**File name: CountryService.java**

package com.cognizant.ormlearn.service;

import java.util.List;

import java.util.Optional;

import org.springframework.transaction.annotation.Transactional;

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

import org.springframework.stereotype.Service;

import com.cognizant.ormlearn.model.Country;

import com.cognizant.ormlearn.repository.CountryRepository;

import com.cognizant.ormlearn.service.exception.CountryNotFoundException;

@Service

public class CountryService {

@Autowired

private CountryRepository countryRepository;

@Transactional

public List<Country> getAllCountries() {

return countryRepository.findAll();

}

@Transactional

public Country findCountryByCode(String code) throws CountryNotFoundException {

Optional<Country> country = countryRepository.findById(code);

if (!country.isPresent()) {

throw new CountryNotFoundException("Country with code " + code + " not found");

}

return country.get();

}

@Transactional

public void addCountry(Country country) {

countryRepository.save(country);

}

@Transactional

public void updateCountry(String code, String name) throws CountryNotFoundException {

Optional<Country> optional = countryRepository.findById(code);

if (!optional.isPresent()) {

throw new CountryNotFoundException("Country with code " + code + " not found");

}

Country country = optional.get();

country.setName(name);

countryRepository.save(country);

}

@Transactional

public void deleteCountry(String code) {

countryRepository.deleteById(code);

}

@Transactional

public List<Country> searchCountryByName(String namePart) {

return countryRepository.findByNameContainingIgnoreCase(namePart);

}

}

**File name: Country.java**

package com.cognizant.ormlearn.model;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.Id;

import javax.persistence.Table;

@Entity

@Table(name = "country")

public class Country {

@Id

@Column(name = "co\_code")

private String code;

@Column(name = "co\_name")

private String name;

// Getters and Setters

public String getCode() {

return code;

}

public void setCode(String code) {

this.code = code;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

@Override

public String toString() {

return "Country [code=" + code + ", name=" + name + "]";

}

}

**File name: CountryNotFoundException.java**

package com.cognizant.ormlearn.service.exception;

public class CountryNotFoundException extends Exception {

public CountryNotFoundException(String message) {

super(message);

}

}

**File name: CountryRepository.java**

package com.cognizant.ormlearn.repository;

import java.util.List;

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

import org.springframework.stereotype.Repository;

import com.cognizant.ormlearn.model.Country;

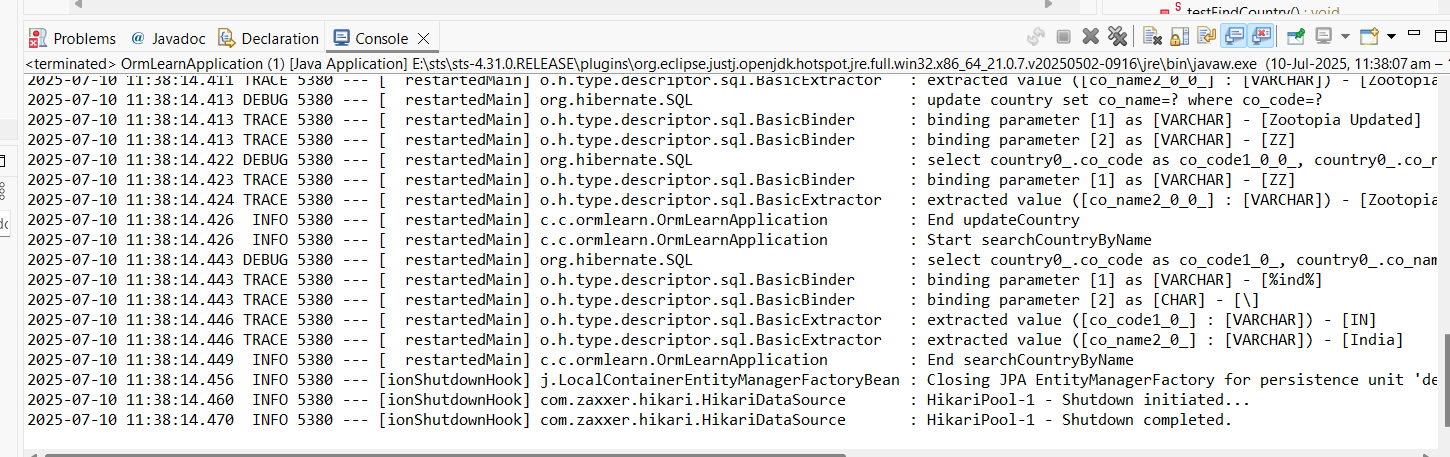
@Repository

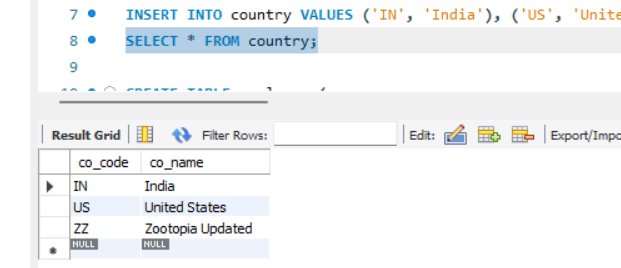
public interface CountryRepository extends JpaRepository<Country, String> {

List<Country> findByNameContainingIgnoreCase(String namePart);

}

**OUTPUT**





**Hands on 6**

**Find a country based on country code**

**File name: OrmLearnApplication.java**

package com.cognizant.ormlearn;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.context.ApplicationContext;

import com.cognizant.ormlearn.model.Country;

import com.cognizant.ormlearn.service.CountryService;

import com.cognizant.ormlearn.service.exception.CountryNotFoundException;

@SpringBootApplication

public class OrmLearnApplication {

private static final Logger *LOGGER* = LoggerFactory.*getLogger*(OrmLearnApplication.class);

private static CountryService *countryService*;

public static void main(String[] args) {

ApplicationContext context = SpringApplication.*run*(OrmLearnApplication.class, args);

*countryService* = context.getBean(CountryService.class);

*getAllCountriesTest*();

}

private static void getAllCountriesTest() {

*LOGGER*.info("Start");

try {

Country country = *countryService*.findCountryByCode("IN");

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

} catch (CountryNotFoundException e) {

*LOGGER*.error("Error: {}", e.getMessage());

}

*LOGGER*.info("End");

}

}

**File name: CountryService.java**

package com.cognizant.ormlearn.service;

import java.util.Optional;

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

import org.springframework.stereotype.Service;

import org.springframework.transaction.annotation.Transactional;

import com.cognizant.ormlearn.model.Country;

import com.cognizant.ormlearn.repository.CountryRepository;

import com.cognizant.ormlearn.service.exception.CountryNotFoundException;

@Service

public class CountryService {

@Autowired

private CountryRepository countryRepository;

@Transactional

public Country findCountryByCode(String countryCode) throws CountryNotFoundException {

Optional<Country> result = countryRepository.findById(countryCode);

if (!result.isPresent()) {

throw new CountryNotFoundException("Country with code " + countryCode + " not found");

}

return result.get();

}

// (Optional) other methods like add, update, delete...

}

**File name: Country.java**

package com.cognizant.ormlearn.model;

import javax.persistence.\*;

@Entity

@Table(name = "country")

public class Country {

@Id

@Column(name = "co\_code")

private String code;

@Column(name = "co\_name")

private String name;

public String getCode() {

return code;

}

public void setCode(String code) {

this.code = code;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

@Override

public String toString() {

return "Country [code=" + code + ", name=" + name + "]";

}

}

**File name: CountryNotFoundException.java**

package com.cognizant.ormlearn.service.exception;

public class CountryNotFoundException extends Exception {

public CountryNotFoundException(String message) {

super(message);

}

}

**File name: CountryRepository.java**

package com.cognizant.ormlearn.repository;

import java.util.List;

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

import org.springframework.stereotype.Repository;

import com.cognizant.ormlearn.model.Country;

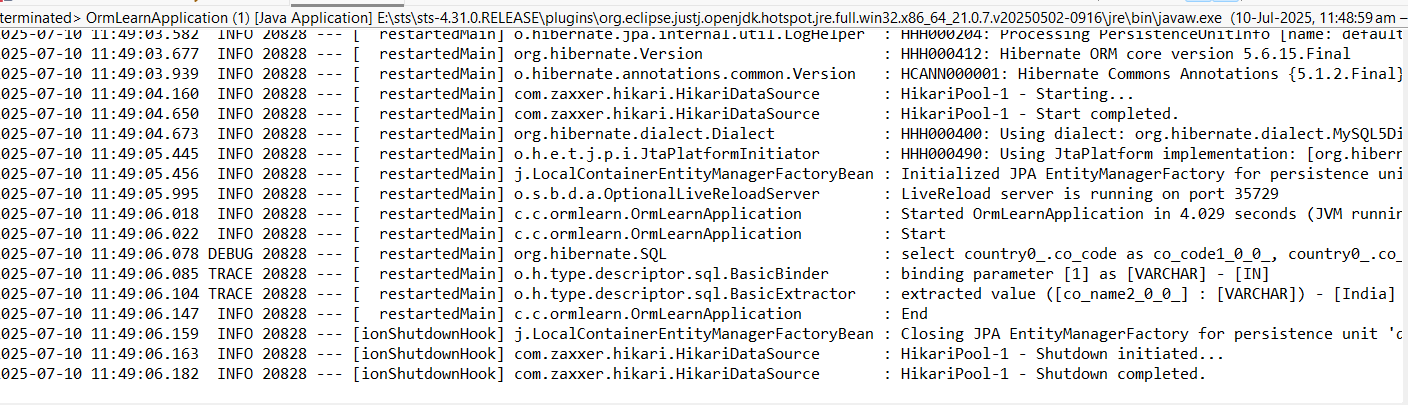
@Repository

public interface CountryRepository extends JpaRepository<Country, String> {

List<Country> findByNameContainingIgnoreCase(String name);

}

**OUTPUT**



**Hands on 7**

**Add a new country   
package com.cognizant.ormlearn;**

**import org.slf4j.Logger;**

**import org.slf4j.LoggerFactory;**

**import org.springframework.boot.SpringApplication;**

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

**import org.springframework.context.ApplicationContext;**

**import com.cognizant.ormlearn.model.Country;**

**import com.cognizant.ormlearn.service.CountryService;**

**import com.cognizant.ormlearn.service.exception.CountryNotFoundException;**

**@SpringBootApplication**

**public class OrmLearnApplication {**

**private static final Logger *LOGGER* = LoggerFactory.*getLogger*(OrmLearnApplication.class);**

**private static CountryService *countryService*;**

**public static void main(String[] args) {**

**ApplicationContext context = SpringApplication.*run*(OrmLearnApplication.class, args);**

***countryService* = context.getBean(CountryService.class);**

***testAddCountry*(); // 👈 This is for Hands-On 7**

**}**

**private static void testAddCountry() {**

***LOGGER*.info("Start testAddCountry");**

**Country country = new Country();**

**country.setCode("JP");**

**country.setName("Japan");**

***countryService*.addCountry(country);**

**try {**

**Country added = *countryService*.findCountryByCode("JP");**

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

**} catch (CountryNotFoundException e) {**

***LOGGER*.error("Error: {}", e.getMessage());**

**}**

***LOGGER*.info("End testAddCountry");**

**}**

**}**

**package com.cognizant.ormlearn.service;**

**import java.util.Optional;**

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

**import org.springframework.stereotype.Service;**

**import org.springframework.transaction.annotation.Transactional;**

**import com.cognizant.ormlearn.model.Country;**

**import com.cognizant.ormlearn.repository.CountryRepository;**

**import com.cognizant.ormlearn.service.exception.CountryNotFoundException;**

**@Service**

**public class CountryService {**

**@Autowired**

**private CountryRepository countryRepository;**

**@Transactional**

**public Country findCountryByCode(String code) throws CountryNotFoundException {**

**Optional<Country> result = countryRepository.findById(code);**

**if (!result.isPresent()) {**

**throw new CountryNotFoundException("Country with code " + code + " not found");**

**}**

**return result.get();**

**}**

**@Transactional**

**public void addCountry(Country country) {**

**countryRepository.save(country);**

**}**

**}**

**package com.cognizant.ormlearn.model;**

**import javax.persistence.\*;**

**@Entity**

**@Table(name = "country")**

**public class Country {**

**@Id**

**@Column(name = "co\_code")**

**private String code;**

**@Column(name = "co\_name")**

**private String name;**

**public String getCode() {**

**return code;**

**}**

**public void setCode(String code) {**

**this.code = code;**

**}**

**public String getName() {**

**return name;**

**}**

**public void setName(String name) {**

**this.name = name;**

**}**

**@Override**

**public String toString() {**

**return "Country [code=" + code + ", name=" + name + "]";**

**}**

**}**

**package com.cognizant.ormlearn.service.exception;**

**public class CountryNotFoundException extends Exception {**

**public CountryNotFoundException(String message) {**

**super(message);**

**}**

**}**

**package com.cognizant.ormlearn.repository;**

**import java.util.List;**

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

**import org.springframework.stereotype.Repository;**

**import com.cognizant.ormlearn.model.Country;**

**@Repository**

**public interface CountryRepository extends JpaRepository<Country, String> {**

**List<Country> findByNameContainingIgnoreCase(String name);**

**}**

**OUTPUT**

