**WEEK-3**

**Exercise 1: Spring Data JPA - Quick Example**

**Code:**

**OrmLearnApplication.java:**

package com.cognizant.ormlearn;

import java.util.List;

import com.cognizant.ormlearn.model.Country;

import com.cognizant.ormlearn.service.CountryService;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.context.ApplicationContext;

@SpringBootApplication

public class OrmLearnApplication {

private static CountryService countryService;

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

private static void testGetAllCountries() {

LOGGER.info("Start");

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

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

LOGGER.info("End");

}

public static void main(String[] args) {

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

LOGGER.info("Inside main");

countryService = context.getBean(CountryService.class);

testGetAllCountries();

}

}

**Country.java:**

package com.cognizant.ormlearn.model;

import jakarta.persistence.Column;

import jakarta.persistence.Entity;

import jakarta.persistence.Id;

import jakarta.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 + "]";

}

}

**CountryRepository.java:**

package com.cognizant.ormlearn.repository;

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> {

}

**CountryService.java:**

package com.cognizant.ormlearn.service;

import java.util.List;

import jakarta.transaction.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;

@Service

public class CountryService {

@Autowired

private CountryRepository countryRepository;

@Transactional

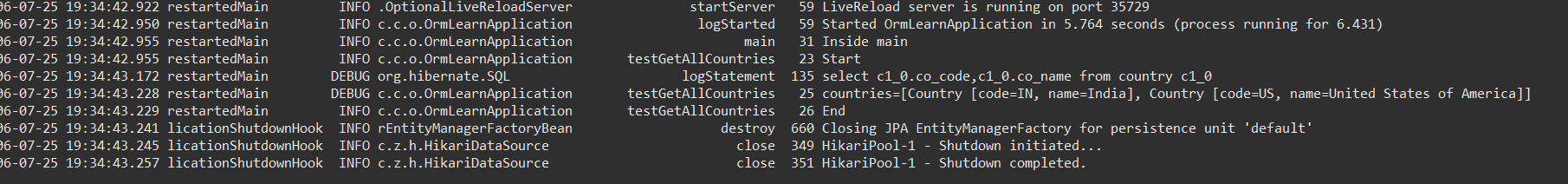
public List<Country> getAllCountries() {

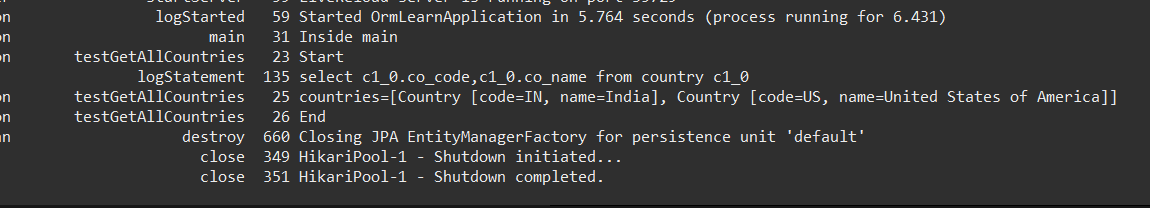
return countryRepository.findAll();

}

}

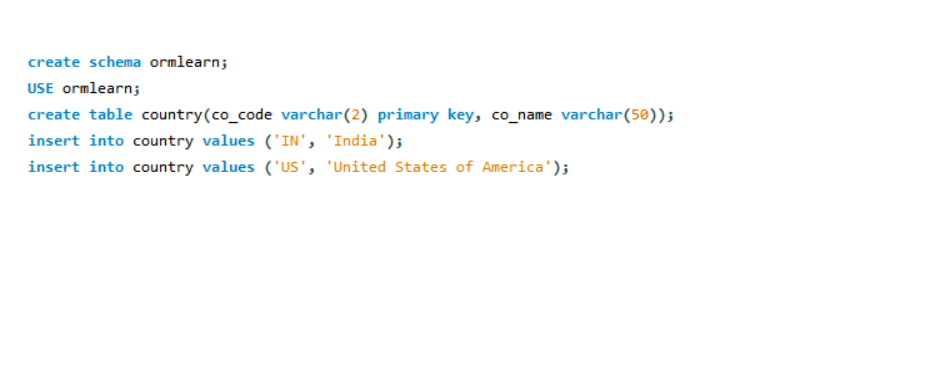
**Output:**

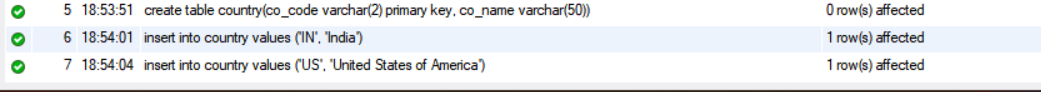




"Inside main” and the list of countries retrieved from database can be seen in the logs above

**MySQL queries**





**Exercise 4: Difference between JPA, Hibernate and Spring Data JPA**

#### 

#### **Java Persistence API (JPA)**

* A specification (JSR 338) for persisting, reading, and managing data from Java objects to a relational database.
* Does not contain any implementation.
* Only defines interfaces like EntityManager,Query, etc.
* Hibernate is a popular implementation of JPA.

#### **Hibernate**

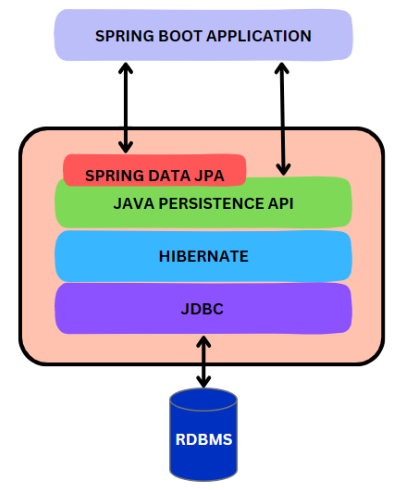
* A concrete implementation of JPA.
* It is an ORM (Object-Relational Mapping) tool that maps Java classes to database tables.
* Provides:  
  + Transaction management
  + Lazy/eager loading
  + Caching
* You write more boilerplate code with Hibernate (session handling, transaction, etc.).

#### 

#### 

#### **Spring Data JPA**

* Built on top of JPA (and often Hibernate).
* Provides abstraction and automation by generating repository code at runtime.
* Reduces boilerplate by allowing you to use interfaces for CRUD operations.
* Spring manages bean injection,transactions,exception handling



#### 

#### **Code Comparison**

#### **Hibernate**

/\* Method to CREATE an employee in the database \*/

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**

**EmployeeRespository.java**

public interface EmployeeRepository extends JpaRepository<Employee, Integer> {

}

**EmployeeService.java**

@Autowire

private EmployeeRepository employeeRepository;

@Transactional

public void addEmployee(Employee employee) {

employeeRepository.save(employee);

}

* From the above codes we can observe that Spring Data JPA significantly reduces boilerplate code by abstracting away low-level database operations like opening sessions, managing transactions, and handling exceptions manually.
* Hibernate gives full control but requires more longer and complex code (managing sessions, transactions, error handling).
* Spring Data JPA allows you to focus on business logic, with automatic transaction handling and CRUD operations through simple repository interfaces.
* In conclusion, Spring Data JPA simplifies persistence logic, making your application cleaner, faster to develop, and easier to maintain.