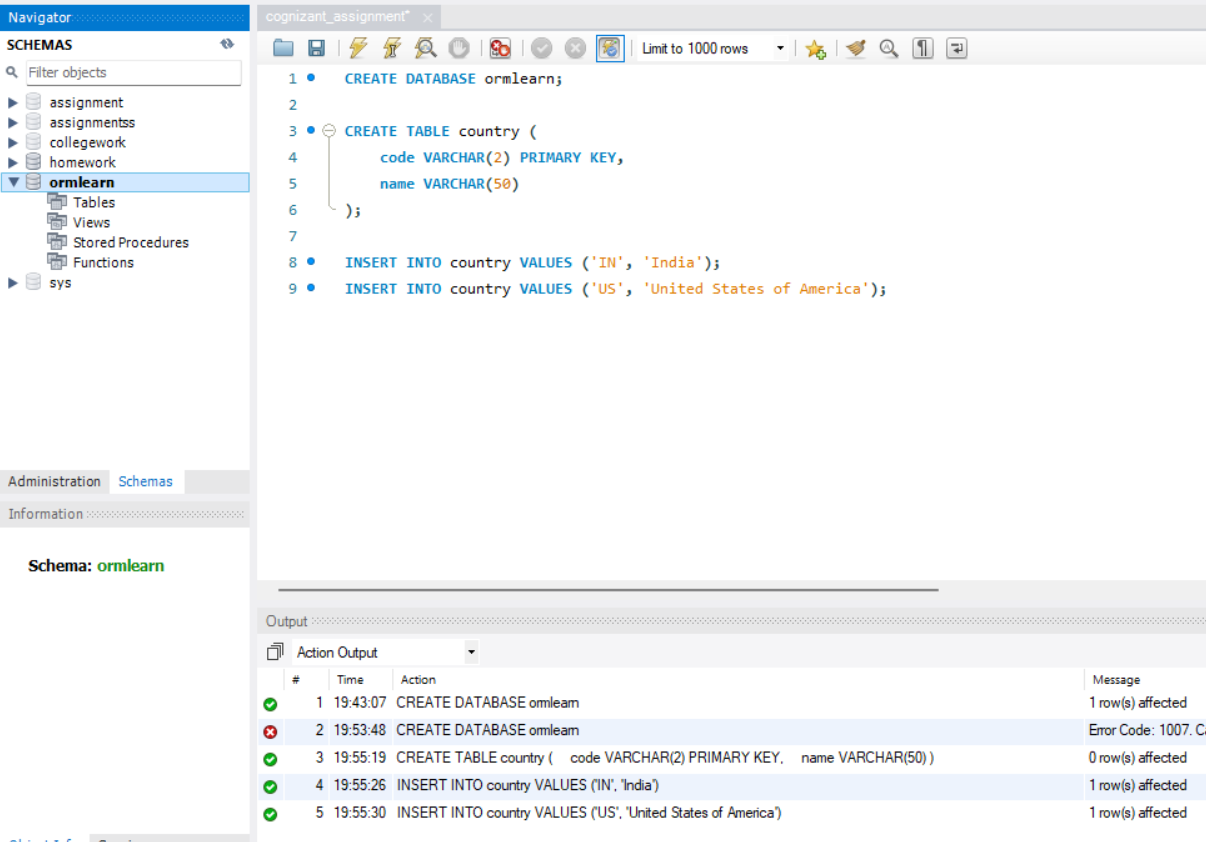
**Chirag H M-6373750(superset id)**

**Hands on 1**

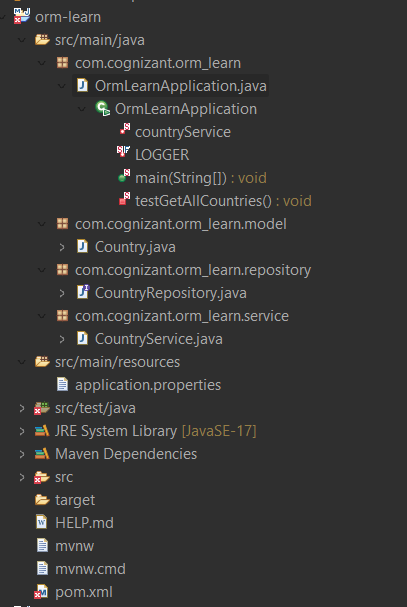
**Spring Data JPA - Quick Example**

**After following all the instructions from the question the schema in sql workbench is as follow:**

****

**Further create the maven project using the spring initializer as per the given details in the question.**

**Then extract the files and load it into eclipse and create the packages and files as follow:**

****

**OrnLearnApplication.java**

package com.cognizant.orm\_learn;

import java.util.List;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

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

import org.springframework.boot.CommandLineRunner;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

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

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

*@SpringBootApplication*

public class OrmLearnApplication implements CommandLineRunner {

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

*@Autowired*

private CountryRepository countryRepository;

public static void main(String[] args) {

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

}

*@Override*

public void run(String... args) throws Exception {

System.***out***.println("Start");

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

countries.forEach(System.***out***::println);

System.***out***.println("End");

}

}

**Country.java**

package com.cognizant.orm\_learn.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 = "code")

private String code;

*@Column*(name = "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 + "]";

}

}

**CountryRepository.java**

package com.cognizant.orm\_learn.repository;

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

import org.springframework.stereotype.Repository;

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

*@Repository*

public interface CountryRepository extends JpaRepository<Country, String> {

}

**CountryService.java**

package com.cognizant.orm\_learn.service;

import java.util.List;

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

import org.springframework.stereotype.Service;

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

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

import jakarta.transaction.Transactional;

*@Service*

public class CountryService {

*@Autowired*

private CountryRepository countryRepository;

*@Transactional*

public List<Country> getAllCountries() {

return countryRepository.findAll();

}

}

**In your orm-learn\src\main\resources\application.properties**

**Add your mysql workbench root username and password**

# Database Configuration

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

spring.datasource.username=root

spring.datasource.password=\*\*\*\*\*\*\*

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

# JPA/Hibernate

spring.jpa.hibernate.ddl-auto=validate

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

**here I have hided my password but you guys can put your password when you execute.**

**Also ensure you have added all the dependencies properly**

<dependencies>

<!-- Spring Boot Starter Data JPA -->

<dependency>

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

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

</dependency>

<!-- MySQL Driver -->

<dependency>

<groupId>com.mysql</groupId>

<artifactId>mysql-connector-j</artifactId>

<scope>runtime</scope>

</dependency>

<!-- Spring Boot Starter -->

<dependency>

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

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

</dependency>

<!-- Logging (optional but helpful) -->

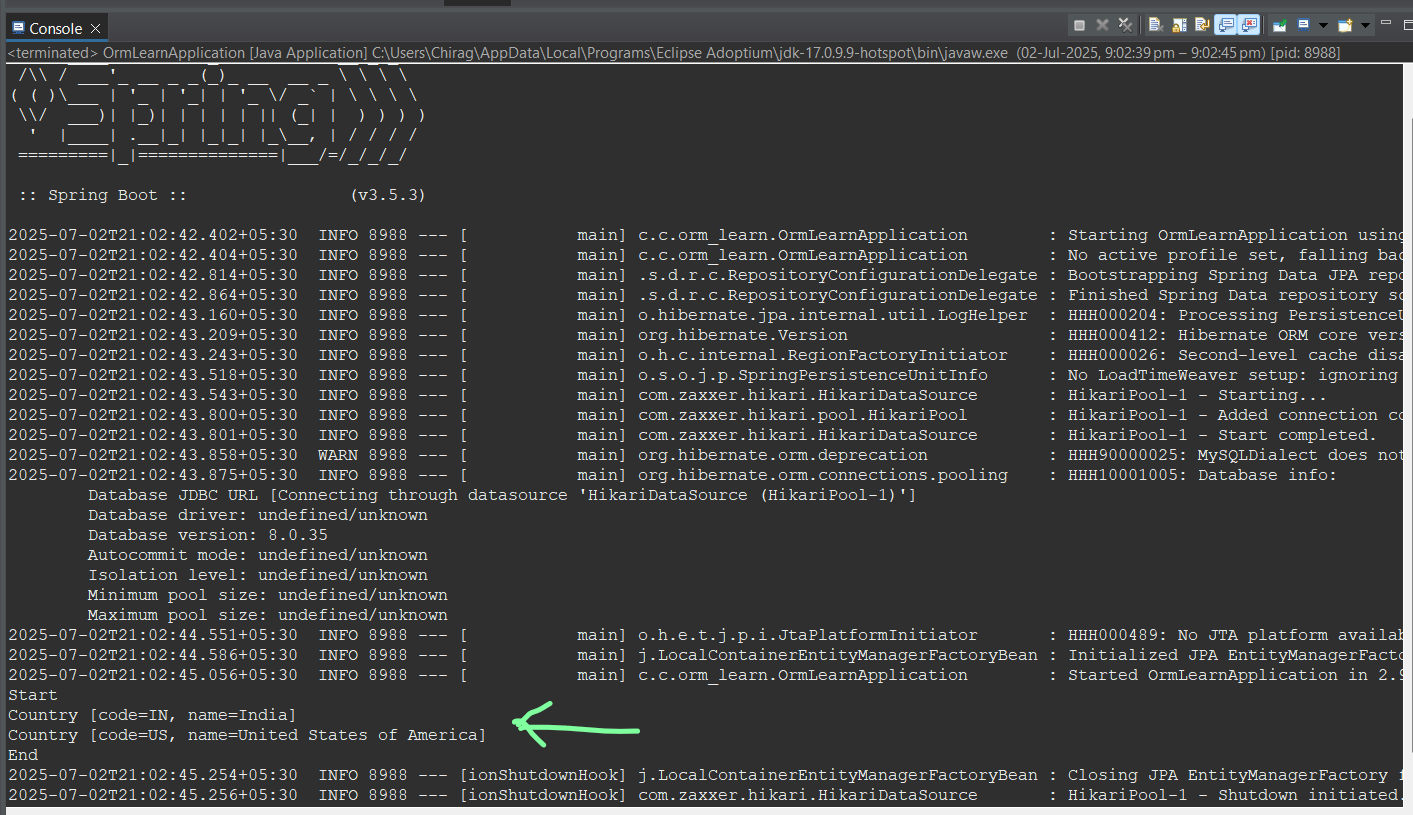
<dependency>

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

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

</dependency>

</dependencies>

**After all this setup is done then run the OrmLearnApplication.java for the output:**

**We can observe that the inserted country data to the database through Mysql workbench is fetched properly through our spring application in the eclipse.**

**Hands on 4**

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

**Java Persistence API (JPA)**

* **JPA** is a **specification** for persisting, reading, and managing data in Java objects.
* It **does not provide any implementation**.
* It defines APIs like EntityManager, Query, and annotations such as @Entity, @Id, @OneToMany, etc.
* **Hibernate** is a **popular implementation** of JPA.

### Hibernate

* Hibernate is an **ORM (Object Relational Mapping) tool**.
* It is one of the **concrete implementations of JPA**.
* It manages database operations using JPA APIs or its own proprietary APIs.
* Requires manual configuration of sessions, transactions, and handling exceptions.

### Spring Data JPA

* Spring Data JPA is a **Spring project** that builds on top of JPA.
* It **does not implement JPA**; instead, it provides **abstraction over JPA implementations** like Hibernate.
* **Removes boilerplate code** such as writing DAO/Repository classes.
* Provides **auto-generated queries**, transaction management, and repository support via interfaces like JpaRepository.

**Code Comparison**

**Using Hibernate (Manual Implementation)**

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;

}

**Using Spring Data JPA (Simplified)**

EmployeeRepository.java

public interface EmployeeRepository extends JpaRepository<Employee, Integer> {

}

EmployeeService.java

@Service

public class EmployeeService {

@Autowired

private EmployeeRepository employeeRepository;

@Transactional

public void addEmployee(Employee employee) {

employeeRepository.save(employee);

}

}

If we compare both the methods then we see that Spring Data JPA **removes a lot of boilerplate code** — no need to manage sessions or transactions manually.