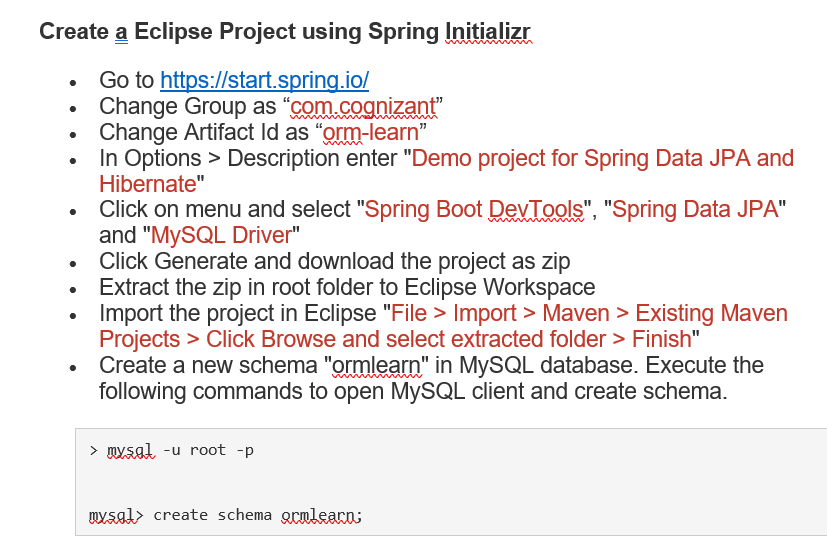
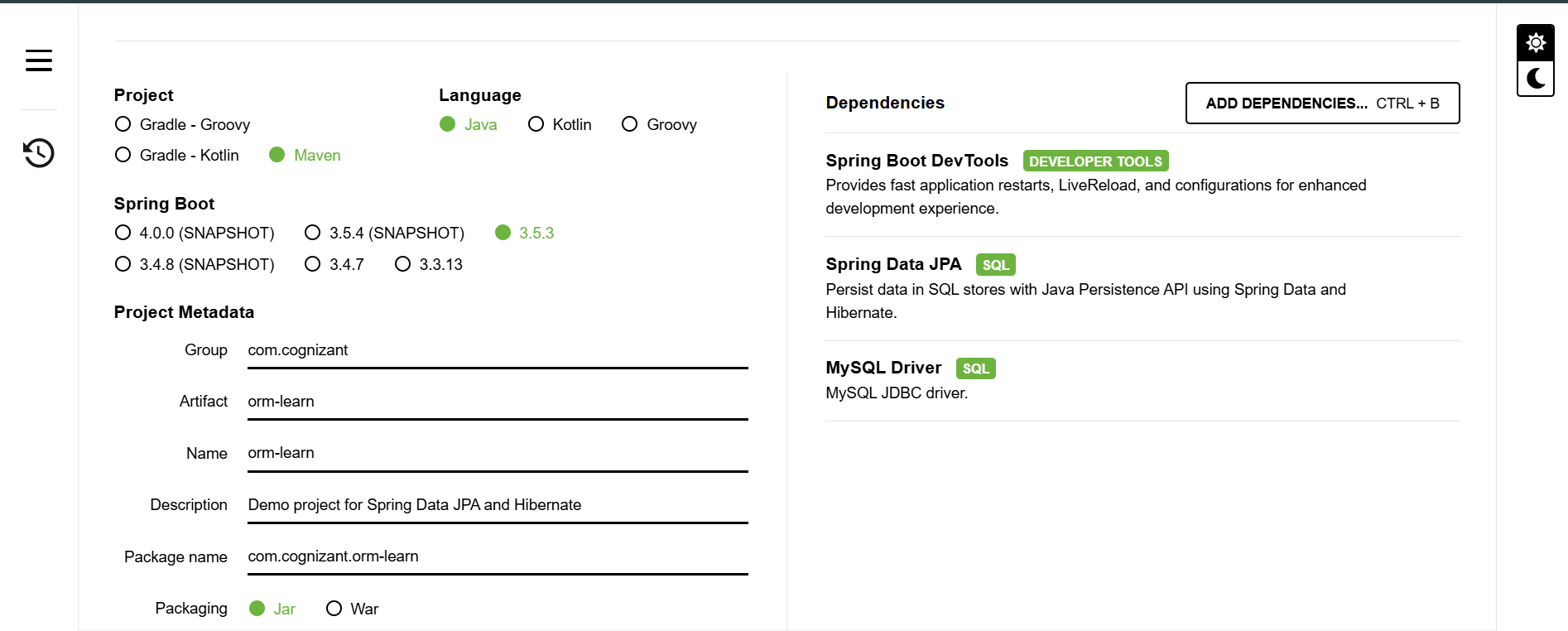
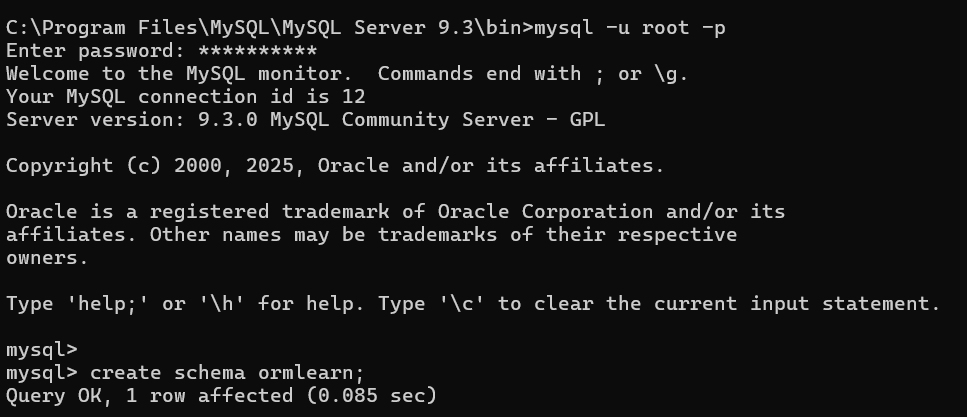
# spring-data-jpa-handson

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

***NOTE : - (I AM USING IntelliJ IDEA INSTEAD OF ECLIPSE)***  




OUTPUT -



* In orm-learn Eclipse project, open src/main/resources/application.properties and include the below database and log configuration.

# Spring Framework and application log

logging.level.org.springframework=info

logging.level.com.cognizant=debug

# Hibernate logs for displaying executed SQL, input and output

logging.level.org.hibernate.SQL=trace

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

# Log pattern

logging.pattern.console=%d{dd-MM-yy} %d{HH:mm:ss.SSS} %-20.20thread %5p %-25.25logger{25} %25M %4L %m%n

# Database configuration

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

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

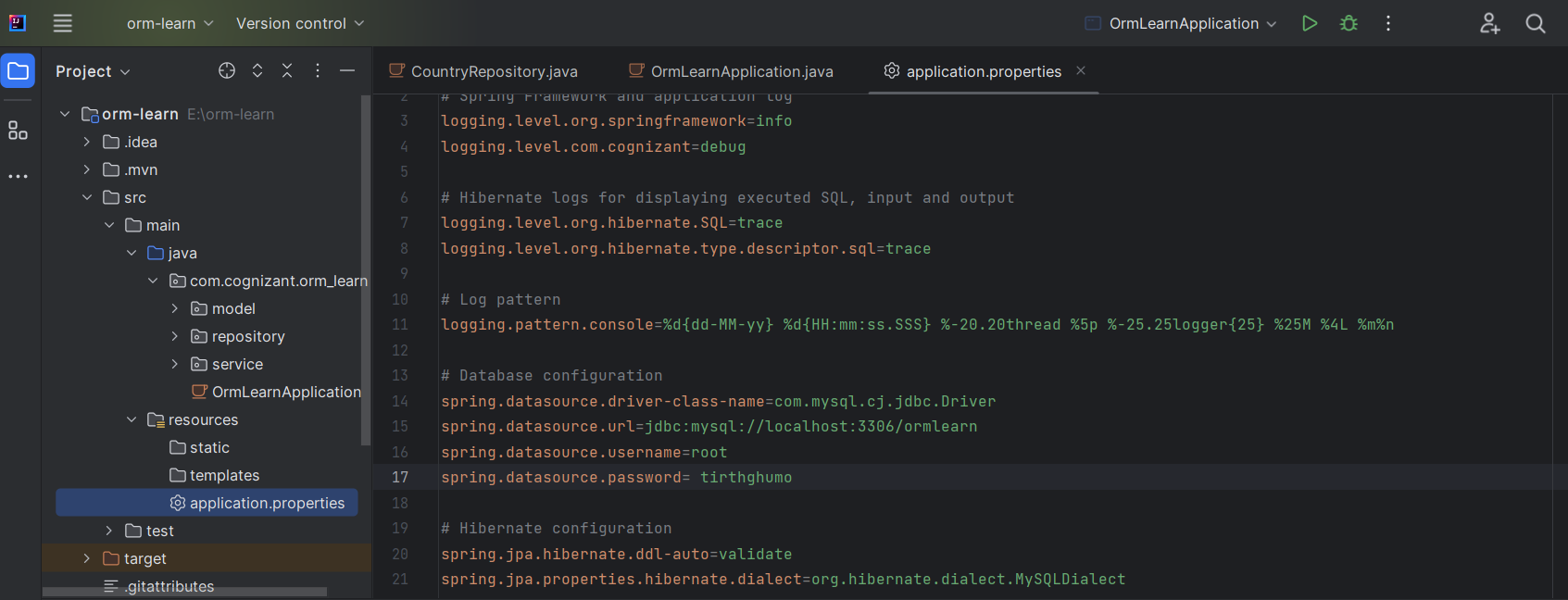
spring.datasource.username=root

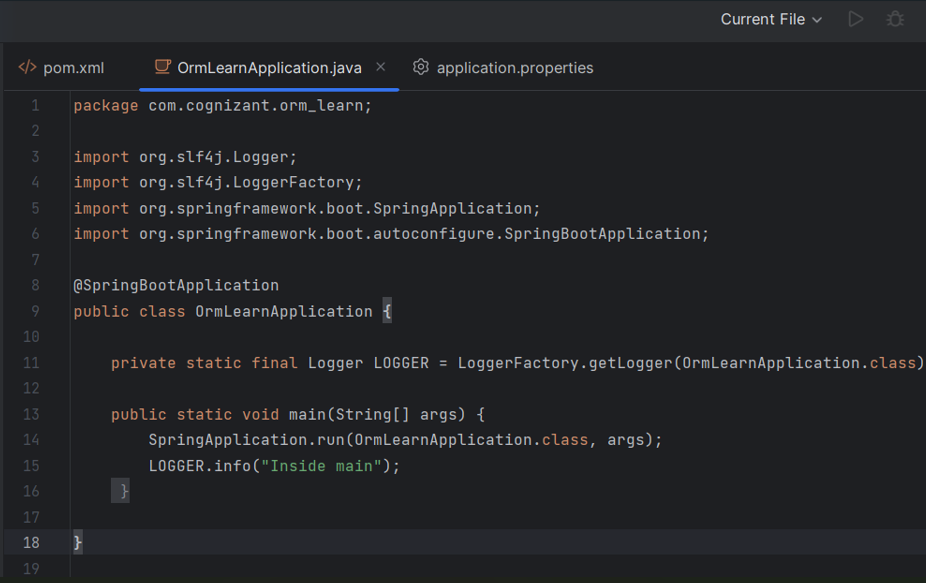
spring.datasource.password=root

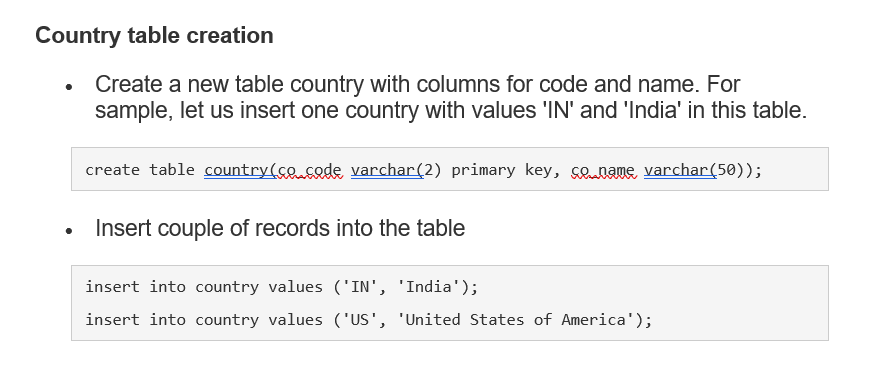
# Hibernate configuration

spring.jpa.hibernate.ddl-auto=validate

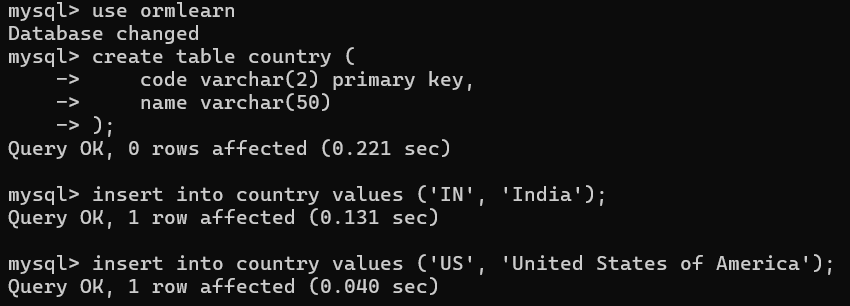
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5Dialect







OUTPUT -



**Persistence Class - com.cognizant.orm-learn.model.Country**

* Open Eclipse with orm-learn project
* Create new package com.cognizant.orm-learn.model
* Create Country.java, then generate getters, setters and toString() methods.
* Include @Entity and @Table at class level
* Include @Column annotations in each getter method specifying the column name.

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="code")

    private String code;

    @Column(name="name")

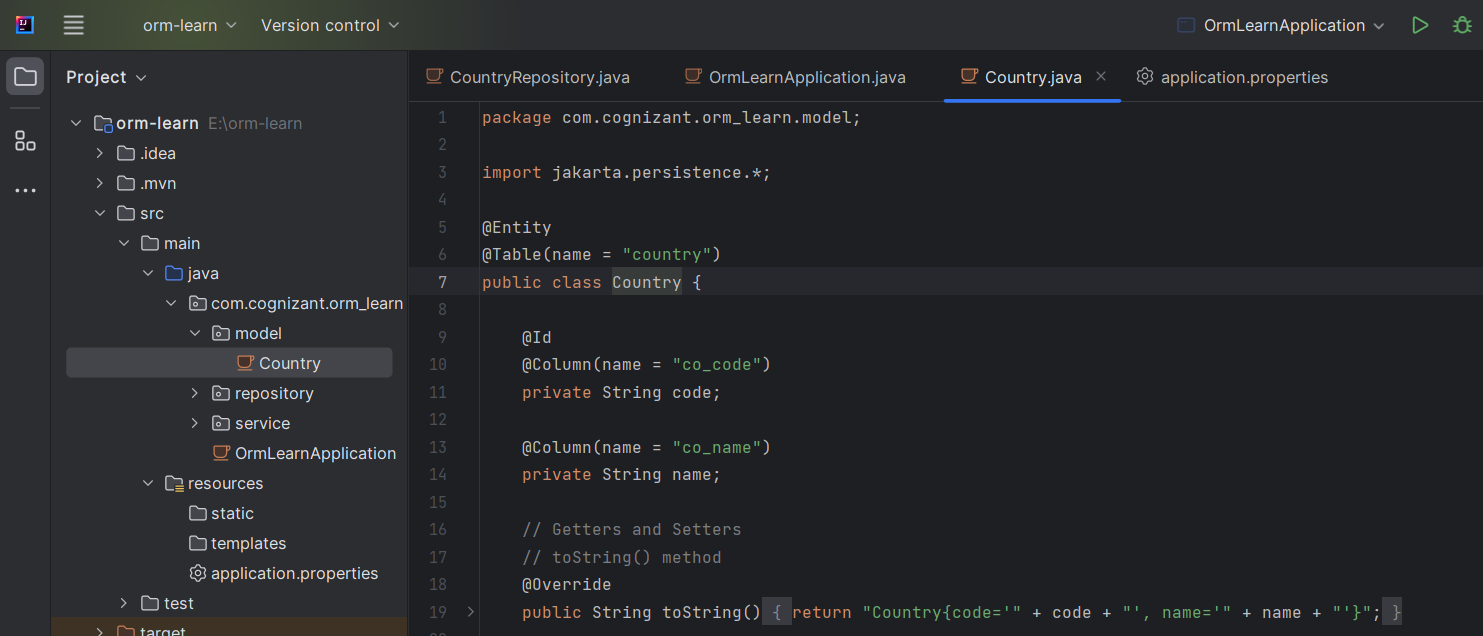
    private String name;

// getters and setters

  // toString()

}

OUTPUT –



**Repository Class - com.cognizant.orm-learn.CountryRepository**

* Create new package com.cognizant.orm-learn.repository
* Create new interface named CountryRepository that extends JpaRepository<Country, String>
* Define @Repository annotation at class level

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

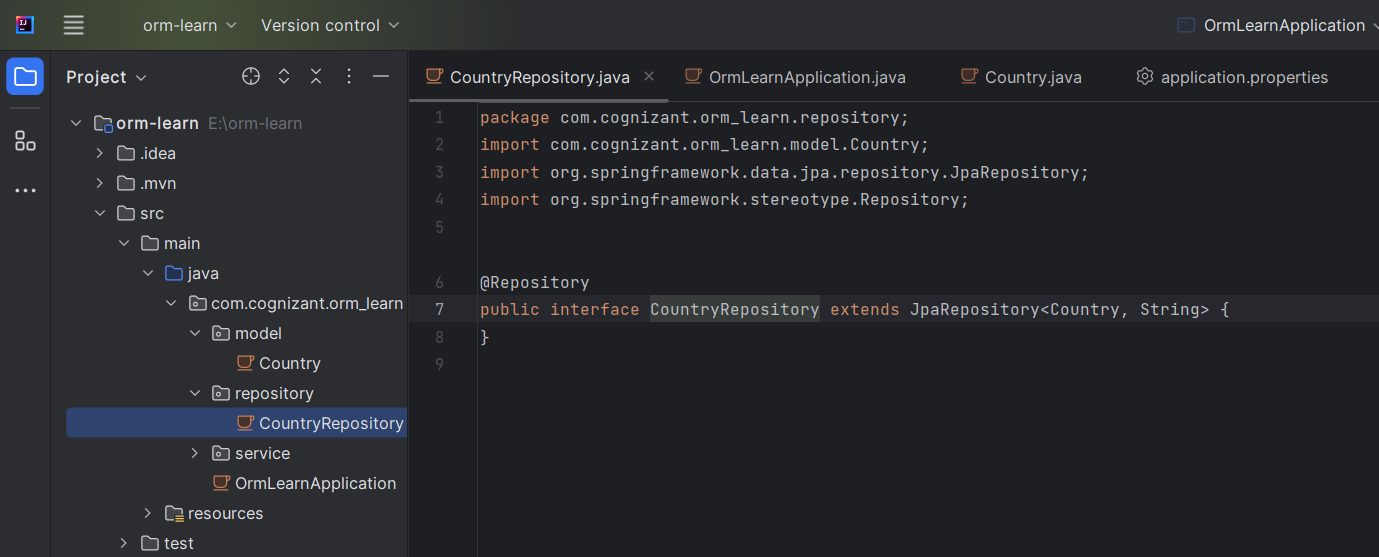
import org.springframework.stereotype.Repository;

import com.cognizant.ormlearn.model.Country;

@Repository

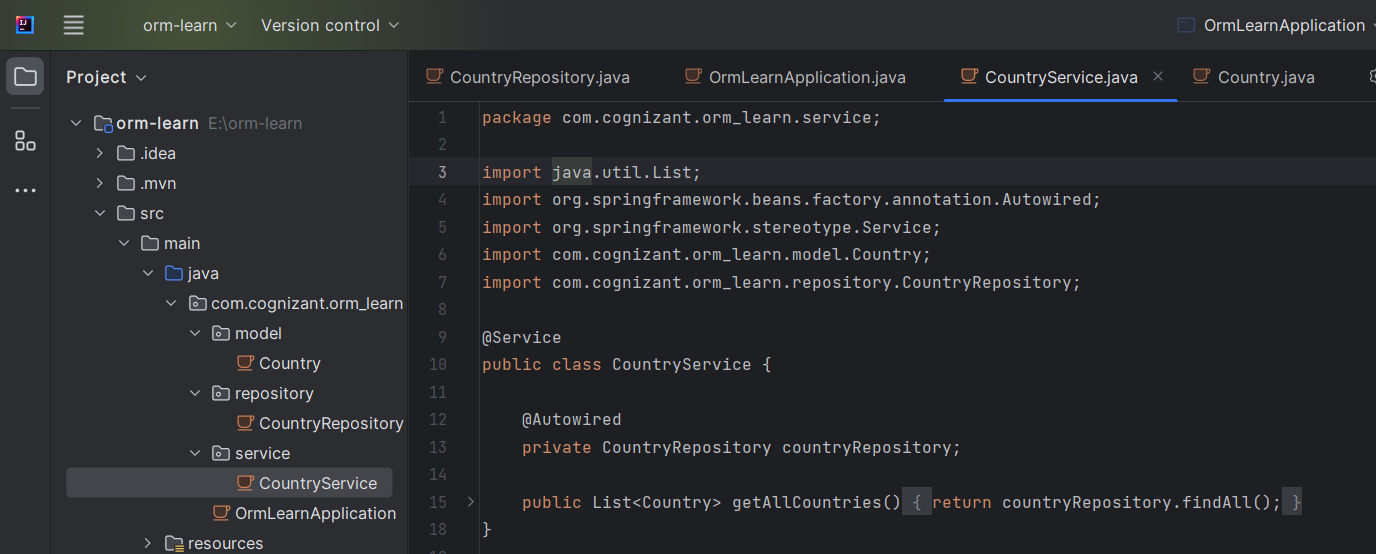
public interface CountryRepository extends JpaRepository<Country, String> {

}



**Service Class - com.cognizant.orm-learn.service.CountryService**

* Create new package com.cognizant.orm-learn.service
* Create new class CountryService
* Include @Service annotation at class level
* Autowire CountryRepository in CountryService
* Include new method getAllCountries() method that returns a list of countries.
* Include @Transactional annotation for this method
* In getAllCountries() method invoke countryRepository.findAll() method and return the result



**Testing in OrmLearnApplication.java**

* Include a static reference to CountryService in OrmLearnApplication class

private static CountryService countryService;

* Define a test method to get all countries from service.

    private static void testGetAllCountries() {

        LOGGER.info("Start");

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

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

        LOGGER.info("End");

    }

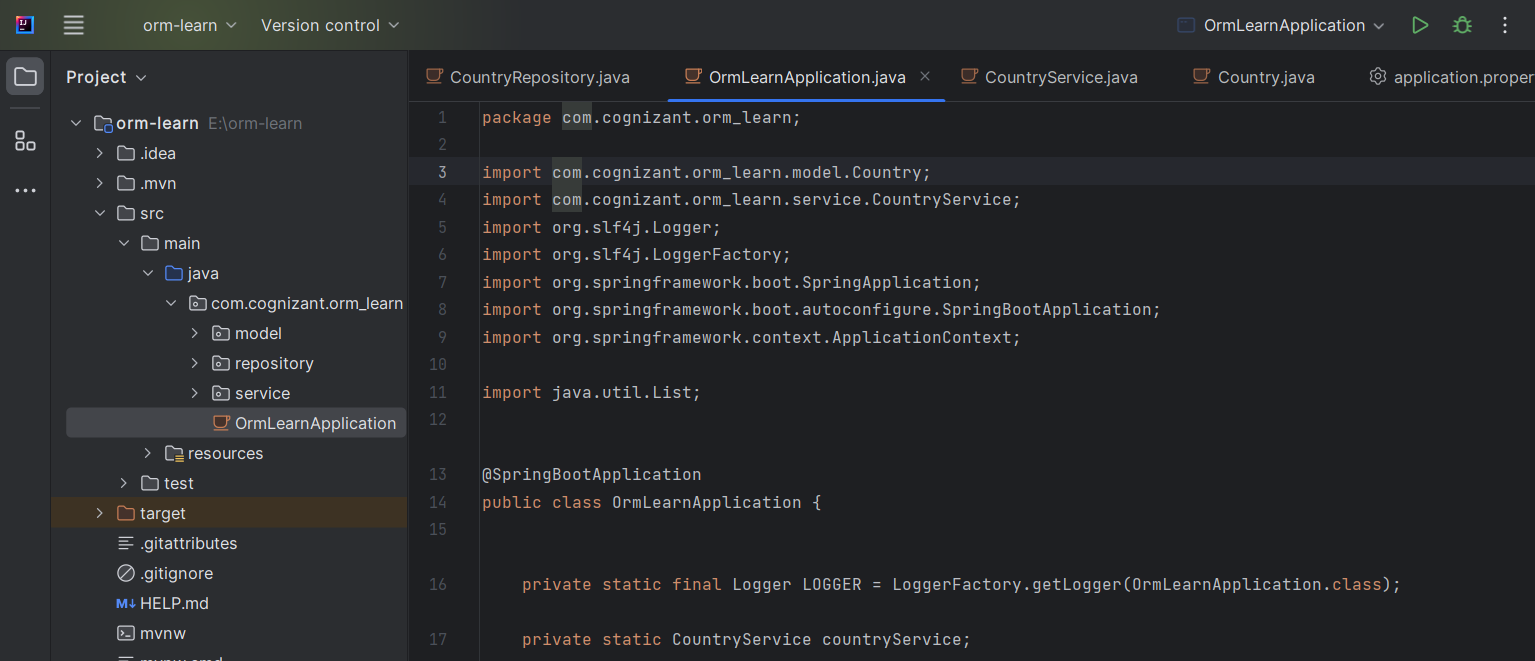
* Modify SpringApplication.run() invocation to set the application context and the CountryService reference from the application context.

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

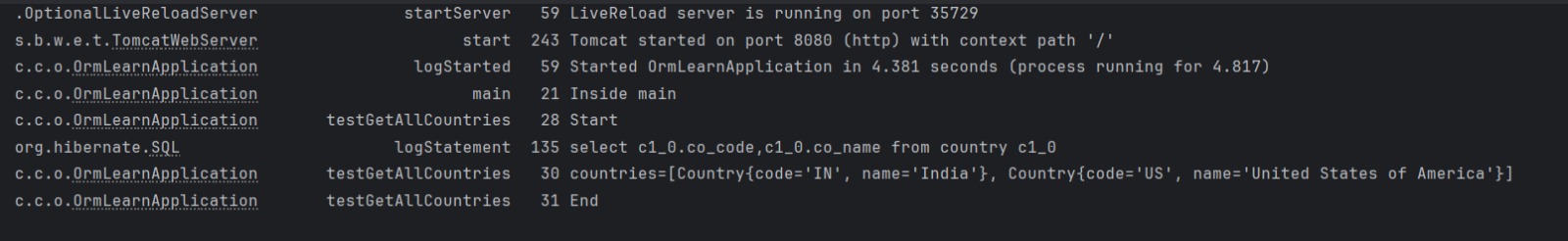
        countryService = context.getBean(CountryService.class);

        testGetAllCountries();

* Execute main method to check if data from ormlearn database is retrieved.





OUTPUT -

**Hands on :**

**Difference between JPA, Hibernate and Spring Data JPA**   
  
Java Persistence API (JPA)

* JSR 338 Specification for persisting, reading and managing data from Java objects
* Does not contain concrete implementation of the specification
* Hibernate is one of the implementation of JPA

Hibernate

* ORM Tool that implements JPA

Spring Data JPA

* Does not have JPA implementation, but reduces boiler plate code
* This is another level of abstraction over JPA implementation provider like Hibernate
* Manages transactions

**SOLUTION**

**🔹 Java Persistence API (JPA)**

* **Definition**: JPA is a **Java specification (JSR 338)** for object-relational mapping (ORM) that defines how Java objects are mapped to database tables.
* **Key Points**:
  + It is **only a specification**, meaning it does **not provide any implementation**.
  + Developers write code using JPA interfaces and annotations.
  + Requires a **JPA provider** for actual implementation (like Hibernate, EclipseLink, etc.).

**🔹 Hibernate**

* **Definition**: Hibernate is a **popular ORM framework** that provides the **implementation of the JPA specification**.
* **Key Points**:
  + Implements JPA and also provides **additional features beyond JPA**.
  + Handles database interactions, transactions, and entity lifecycle management.
  + Offers **SessionFactory**, **Session**, **Transaction**, etc., for manual data operations.

**🔹 Spring Data JPA**

* **Definition**: Spring Data JPA is a **Spring module** that adds an abstraction layer **on top of JPA**.
* **Key Points**:
  + It **does not implement JPA**, but **uses an existing implementation** (like Hibernate) underneath.
  + Eliminates boilerplate code using **Repository interfaces**, such as CrudRepository, JpaRepository, etc.
  + Supports **method name-based queries**, pagination, and transaction management with minimal code.

**🔁 Code Comparison**

**🔸 Hibernate Example (Manual ORM Handling)**

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;

}

* Manual handling of **session**, **transactions**, and **error handling**.
* More code and complexity.

**🔸 Spring Data JPA Example (Abstraction)**

**Repository:**

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

**Service Layer:**

@Autowired

private EmployeeRepository employeeRepository;

@Transactional

public void addEmployee(Employee employee) {

employeeRepository.save(employee);

}

* Simplified approach with **no need to manage transactions or sessions manually**.
* Just call save() method on repository interface.

**Summary Table**

| **Feature** |  | **JPA** | **Hibernate** | **Spring Data JPA** |
| --- | --- | --- | --- | --- |
| Type |  | Specification | Implementation of JPA | Abstraction over JPA & implementation |
| Boilerplate Code |  | Moderate | High | Very Low |
| Implementation |  | None | Yes | No (relies on Hibernate, EclipseLink) |
| Provided by |  | Java EE | org.hibernate | Spring Framework |
| Developer Effort |  | Medium | High | Low |