**Superset ID: 6394725**

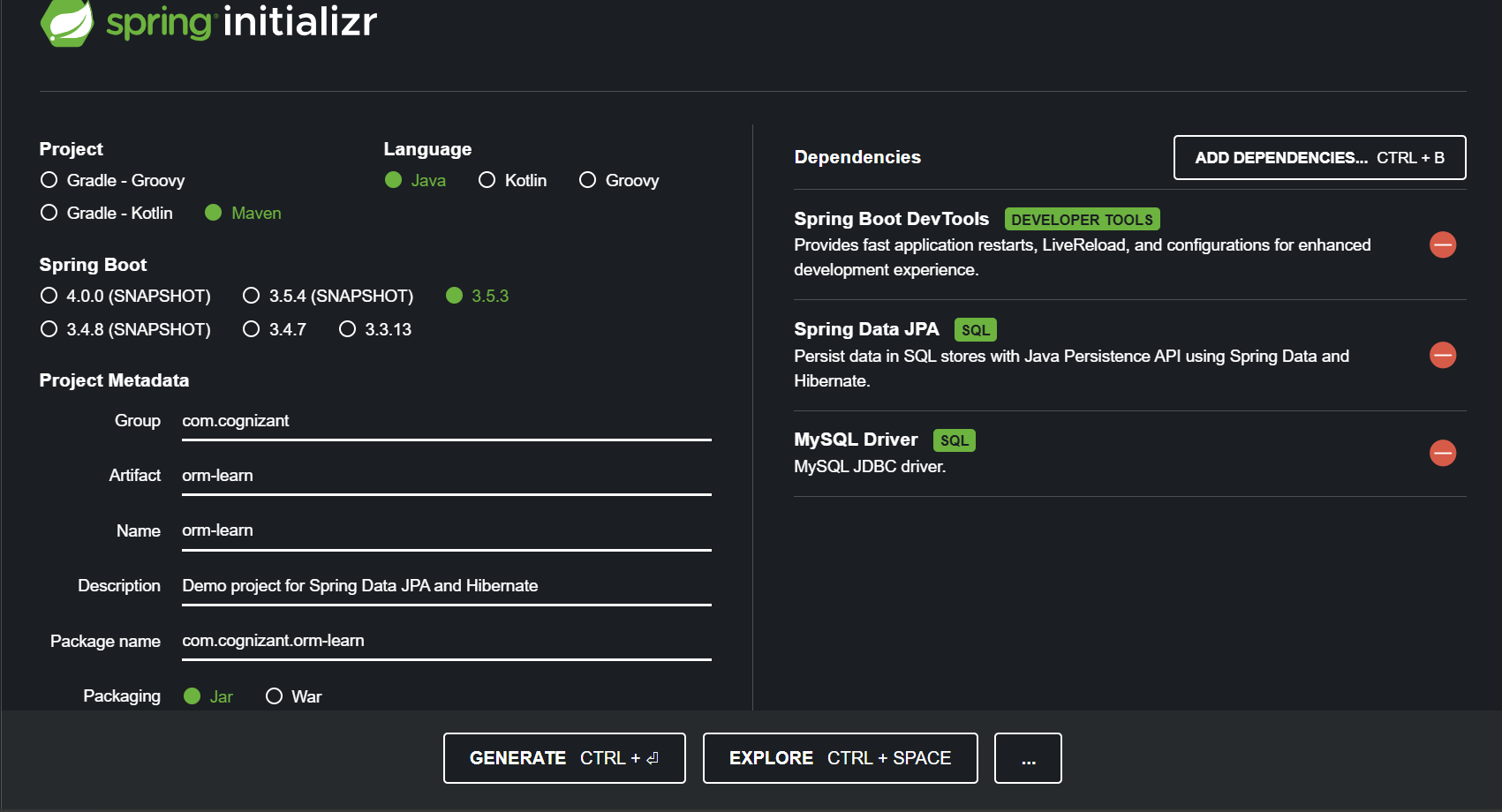
**Spring Data JPA with Hibernate**

**Hands on 1**

**Spring Data JPA - Quick Example**   
**Software Pre-requisites**

* MySQL Server 8.0
* MySQL Workbench 8
* Eclipse IDE for Enterprise Java Developers 2019-03 R
* Maven 3.6.2

**Create a Eclipse Project using Spring Initializer:**



**Create Database Schema:**

CREATE DATABASE ormlearn;

USE ormlearn;

CREATE TABLE country (

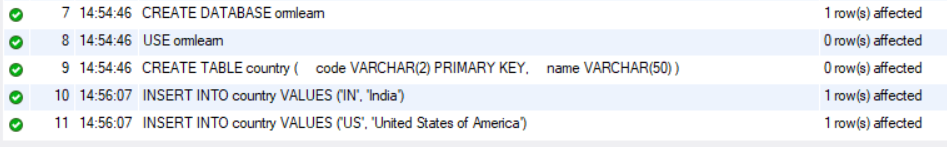
code VARCHAR(2) PRIMARY KEY,

name VARCHAR(50)

);

INSERT INTO country VALUES ('IN', 'India');

INSERT INTO country VALUES ('US', 'United States of America');



**Import Eclipse project into Eclipse:**

* Open Eclipse → File > Import.
* Choose: Maven > Existing Maven Projects.
* Browse to the unzipped folder → Select → Finish.

**Configure application.properties:**

src/main/resources/application.properties in Eclipse

spring.application.name=orm-learn

# 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

# Console log format

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

# 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=root

# Hibernate config

spring.jpa.hibernate.ddl-auto=validate

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

## **Part 7: Create Entity Class - Country.java**

## src/main/java/com/cognizant/ormlearn/model in Eclipse

package com.cognizant.ormlearn.model;

import jakarta.persistence.\*;

*@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 + "]";

}

}

## **Create Repository Interface - CountryRepository.java**

## src/main/java/com/cognizant/ormlearn/repository in Eclipse

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> {

}

## **Create Service Class - CountryService.java**

### src/main/java/com/cognizant/ormlearn/service in Eclipse

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();

}

}

## **Modify OrmLearnApplication.java for Testing**

### src/main/java/com/cognizant/ormlearn/OrmLearnApplication.java in Eclipse

package com.cognizant.ormlearn;

import java.util.List;

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;

*@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);

***LOGGER***.info("Inside main");

*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");

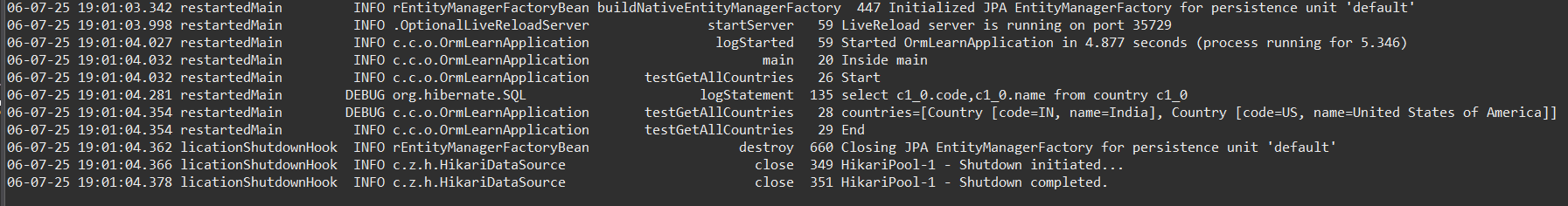
}

}

## **Run the Project**

## Eclipse → OrmLearnApplication.java → Right-click → Run As → Java Application

Output:



**Hands on 4**

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

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

* It is a specification (JSR 338) for ORM (Object Relational Mapping).
* It defines interfaces and annotations but has no implementation itself.
* You need a JPA provider (e.g., Hibernate) to use it.
* Requires EntityManager for operations.
* Example-based and somewhat manual in usage.

**Code Example (Using JPA with EntityManager):**

@PersistenceContext

private EntityManager entityManager;

@Transactional

public void addEmployee(Employee employee) {

entityManager.persist(employee);

}

1. **Hibernate**

* It is a concrete ORM framework that implements JPA specification.
* Provides session management, caching, and lazy loading.
* Can be used both as a JPA provider and as a standalone ORM tool.
* Requires manual session and transaction handling if used standalone.

**Code Example (Using Hibernate Directly):**

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;

}

1. **Spring Data JPA**

* It is a Spring abstraction built on top of JPA.
* Uses Spring Repositories to reduce boilerplate code.
* Automatically manages transactions using @Transactional.
* Provides built-in query methods and supports custom queries easily.
* Works with any JPA provider like Hibernate underneath.

**Code Example (Using Spring Data JPA):**

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

@Service

public class EmployeeService {

@Autowired

private EmployeeRepository employeeRepository;

@Transactional

public void addEmployee(Employee employee) {

employeeRepository.save(employee);

}

}