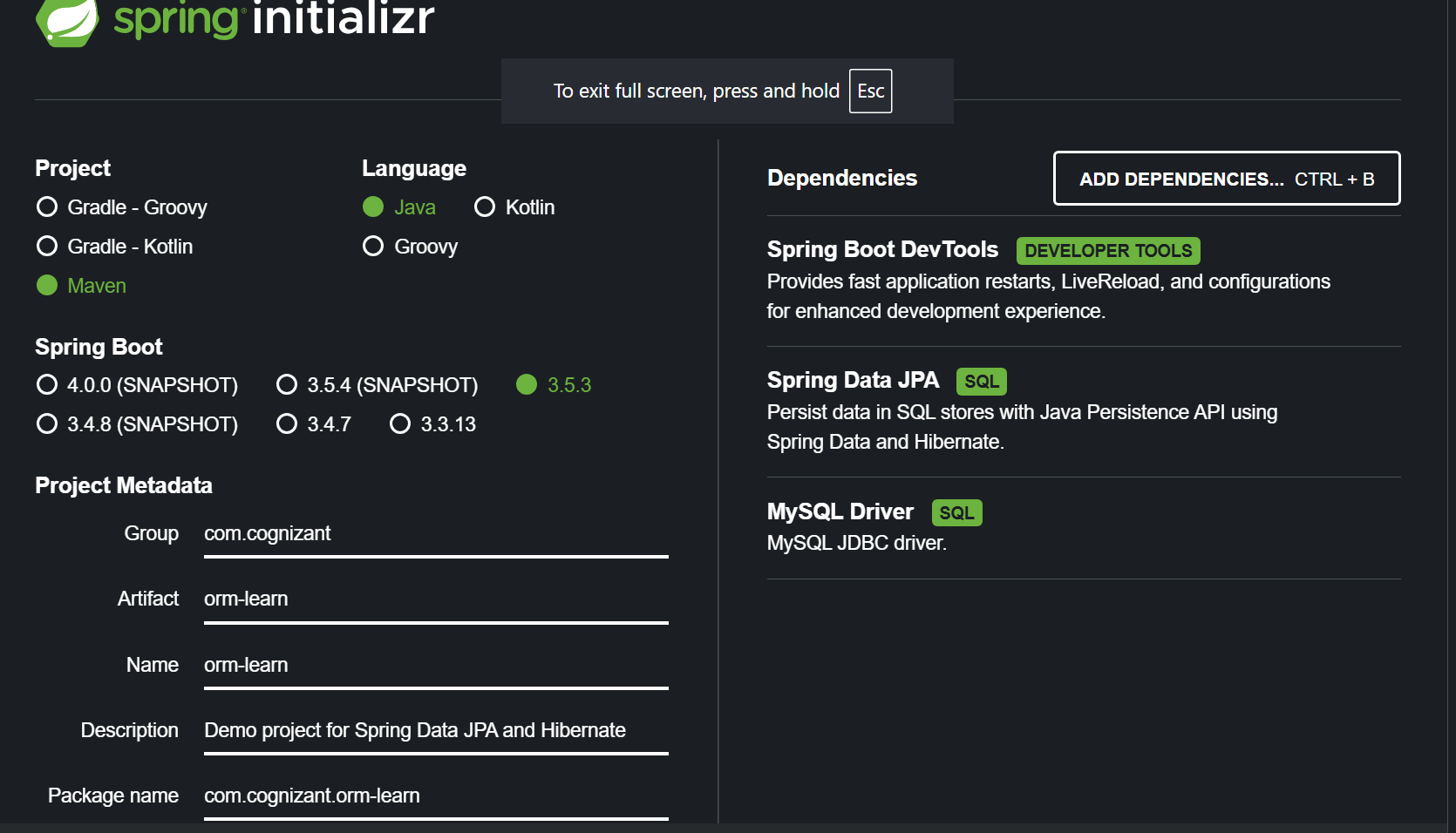
**Spring Data JPA with Spring Boot, Hibernate**

1. **spring-data-jpa-handson**

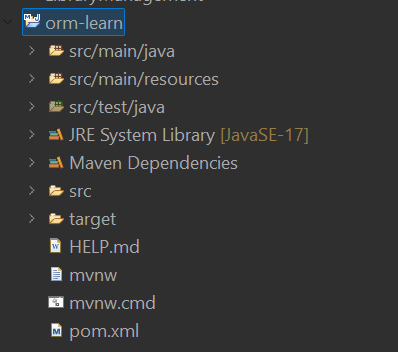
**Hands on 1:**

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

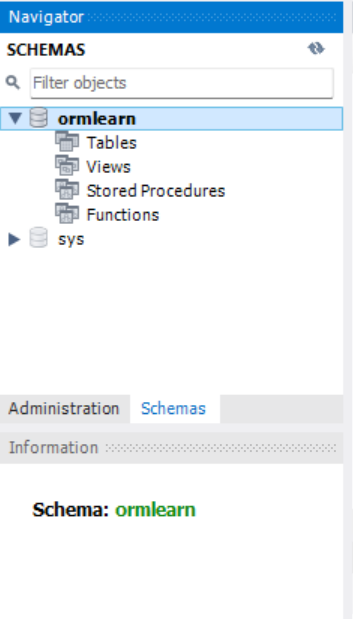
* MySQL Server 8.0
* MySQL Workbench 8
* Eclipse IDE for Enterprise Java Developers 2019-03 R
* Maven 3.6.2
* **Create a Eclipse Project using Spring Initializr**



* **Import the project in Eclipse** "File > Import > Maven > Existing Maven Projects > Click Browse and select extracted folder > Finish"



* **Create a new schema** "ormlearn" in MySQL database. Execute the following commands to open MySQL client and create schema.

****

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

**Application.properties**

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

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

# DB 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=Anees

# Hibernate

spring.jpa.hibernate.ddl-auto=validate

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

* **MySQL Database Setup:**

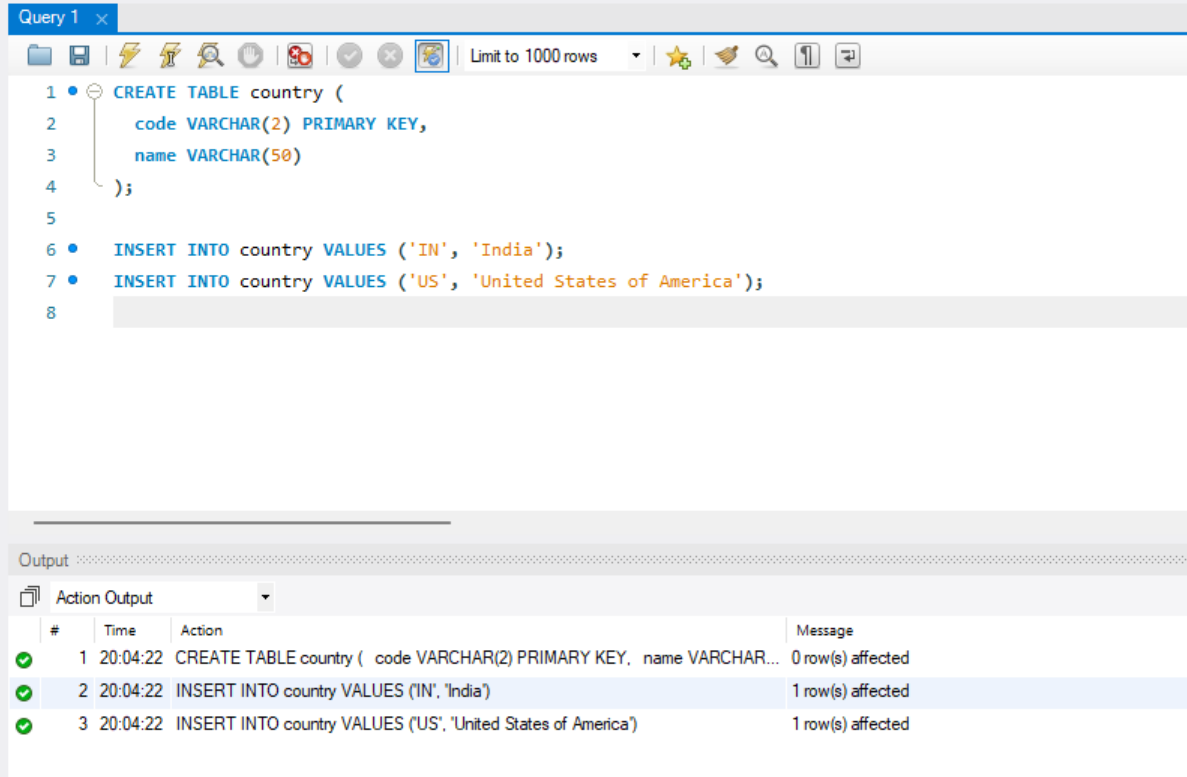
CREATE TABLE country (

code VARCHAR(2) PRIMARY KEY,

name VARCHAR(50)

);

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



* **Create Model,Repository,Service Class**

***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 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();

}

}

* **Create Main class and run the application**

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.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.context.ApplicationContext;

import java.util.List;

*@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);

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

*testGetAllCountries*();

}

private static void testGetAllCountries() {

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

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

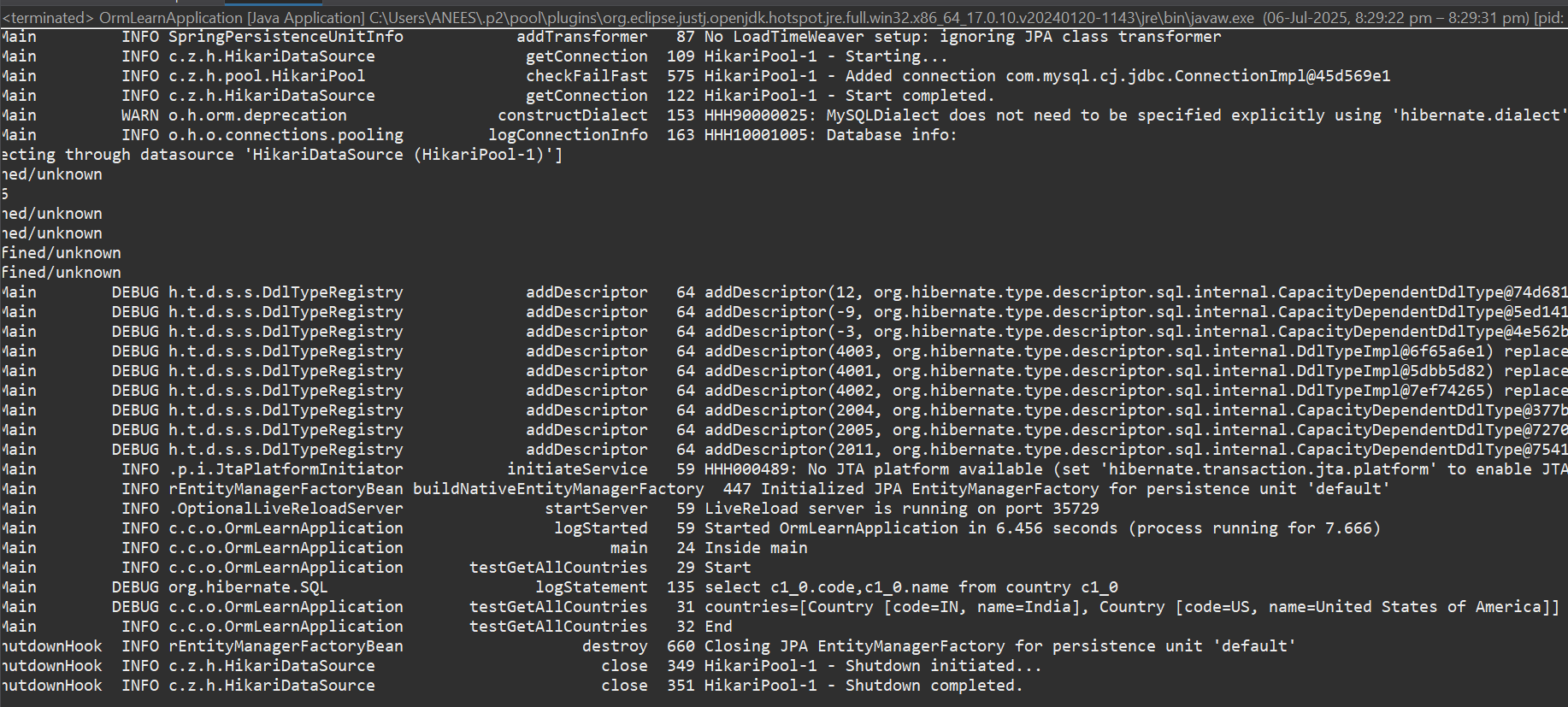
***LOGGER***.debug("countries={}", countries);

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

}

}

**Output:**



**Hands-on 4: Difference between JPA, Hibernate, and Spring Data JPA**

**Introduction :**

In modern Java enterprise applications, working with relational databases is common. Instead of writing plain SQL by hand, developers usually use Object-Relational Mapping (ORM) tools to map Java objects to database tables. In this context, JPA, Hibernate, and Spring Data JPA are essential for simplifying database access and operations.

**Java Persistence API (JPA) :**

JPA is a Java specification for ORM in Java applications. It defines standard annotations and interfaces like @Entity, @Id, @Table, and @Column for mapping Java classes to database tables. JPA itself does not perform any persistence; it requires an implementation like Hibernate or EclipseLink. It is vendor-agnostic, so developers can switch providers with minimal code changes.

**Hibernate :**

Hibernate is a popular ORM framework in the Java ecosystem. It is a specific implementation of JPA but also offers many extra features not included in JPA, such as:

* Lazy loading
* Caching
* HQL (Hibernate Query Language)
* Dirty checking (automatic detection of changes to objects)

With Hibernate, developers gain more control, but they also need to manage sessions, transactions, and boilerplate code manually, unless they use it with Spring.

**Spring Data JPA**

Spring Data JPA is a Spring module built on top of JPA. It provides a higher level of abstraction, allowing developers to carry out database operations with little or no code.

**Features:**

* Auto-generates DAO implementations using interfaces.
* Deep integration with Spring Boot.
* Built-in transaction management using @Transactional.
* Removes the need for boilerplate CRUD code.

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

|  |  |  |  |
| --- | --- | --- | --- |
| **Feature** | **JPA** | **Hibernate** | **Spring Data JPA** |
| **Type** | Specification (API) | Framework (Implementation) | Abstraction (Spring Module) |
| **Implementation** | No | Yes | No (depends on Hibernate/JPA) |
| **Boilerplate Code** | High | Moderate | Very Low |
| **Transaction Management** | No | Manual | Automatic (via Spring) |
| **Default Query Language** | JPQL | HQL | JPQL (auto-generated queries) |
| **CRUD Implementation** | Manual | Manual | Auto-generated |
| **Spring Integration** | Partial | Partial | Full |

**Code Comparison**

Hibernate Example (Manual ORM using Session API)

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;

}

Explanation:

You must manage sessions, transactions, and exception handling on your own. This can make the code lengthy and harder to maintain, especially in large applications.

Spring Data JPA Example

*EmployeeRepository.java*

public interface EmployeeRepository extends JpaRepository<Employee, Integer> {

}

*EmployeeService.java*

public class EmployeeService {

private EmployeeRepository employeeRepository;

public void addEmployee(Employee employee) {

employeeRepository.save(employee);

}

}

Explanation:

Spring automatically generates the implementation for EmployeeRepository. There is no need to write any SQL or transaction logic. The code is clean, maintainable, and adheres to “convention over configuration.”