**Candidate Name**:- SHARANYA K

**Superset ID**:-6419887

**WEEK – 3 HANDS ON EXERCISE (JAVA FSE DEEPSKILLING)**

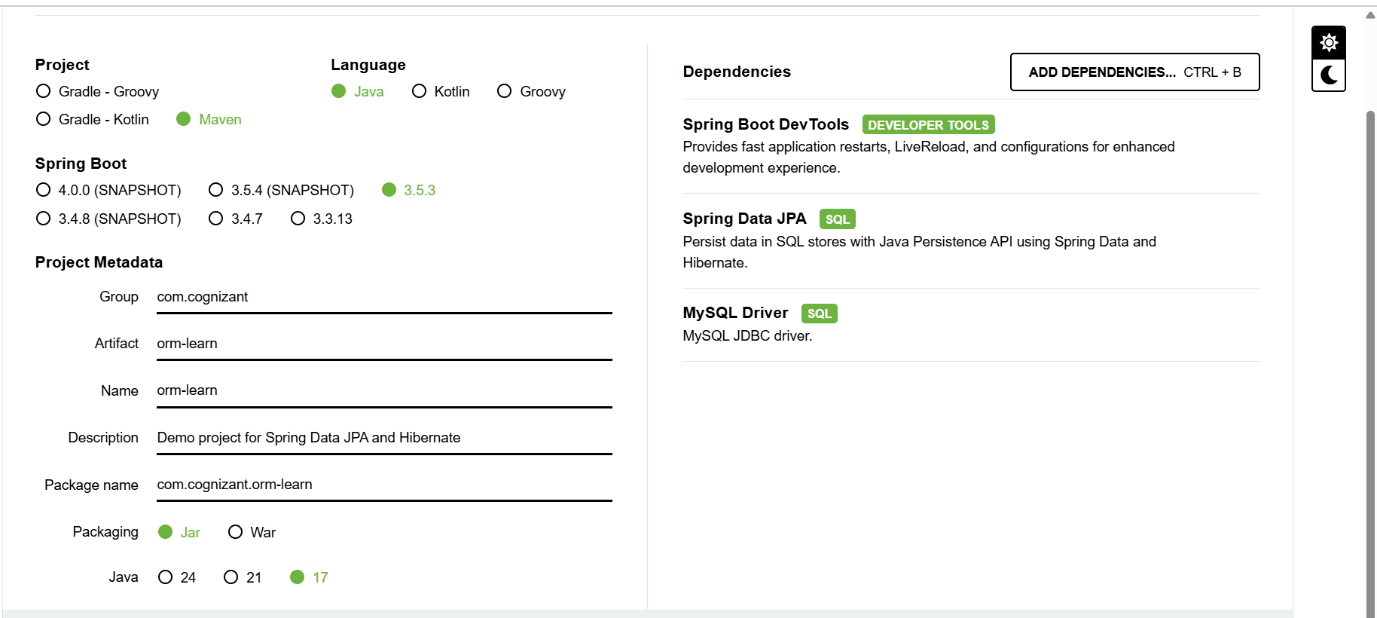
**Module 6 - Spring Data JPA with Spring Boot, Hibernate**

**Hands on 1**

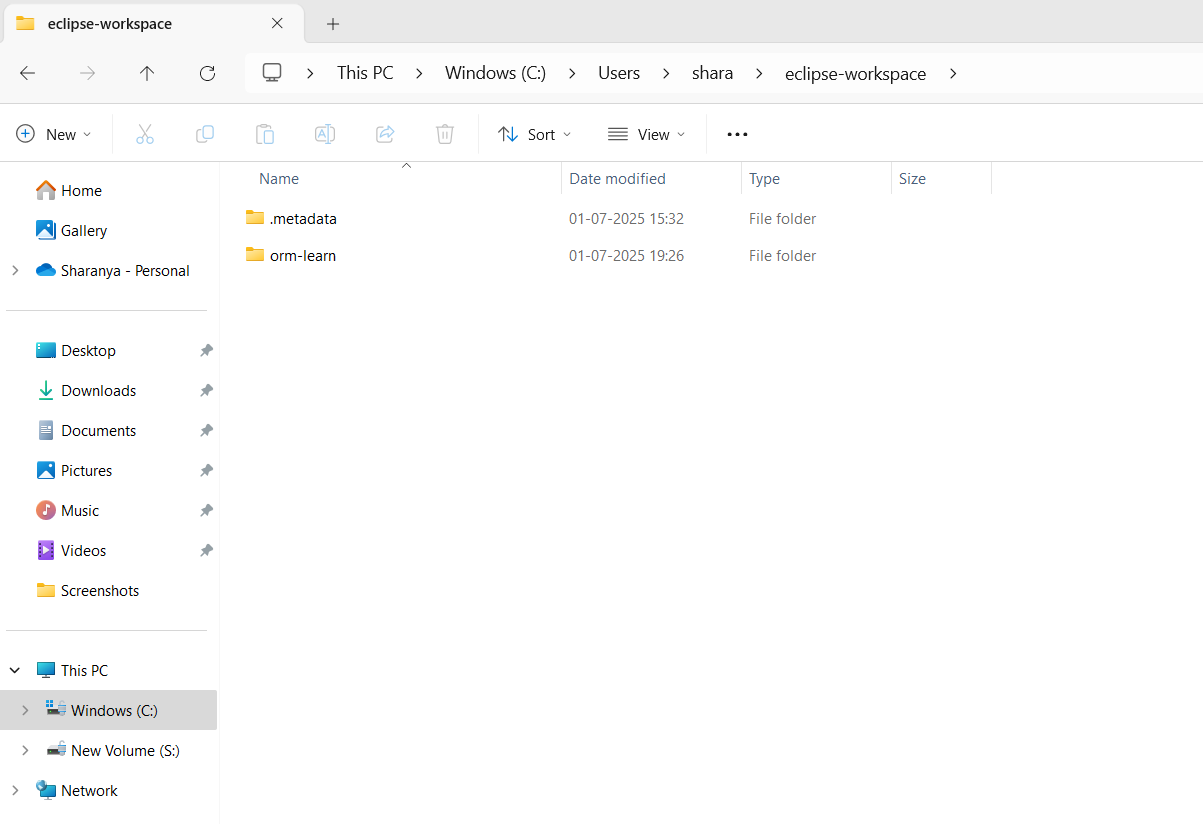
**Spring Data JPA - Quick Example**   
  
**Software Pre-requisites**

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

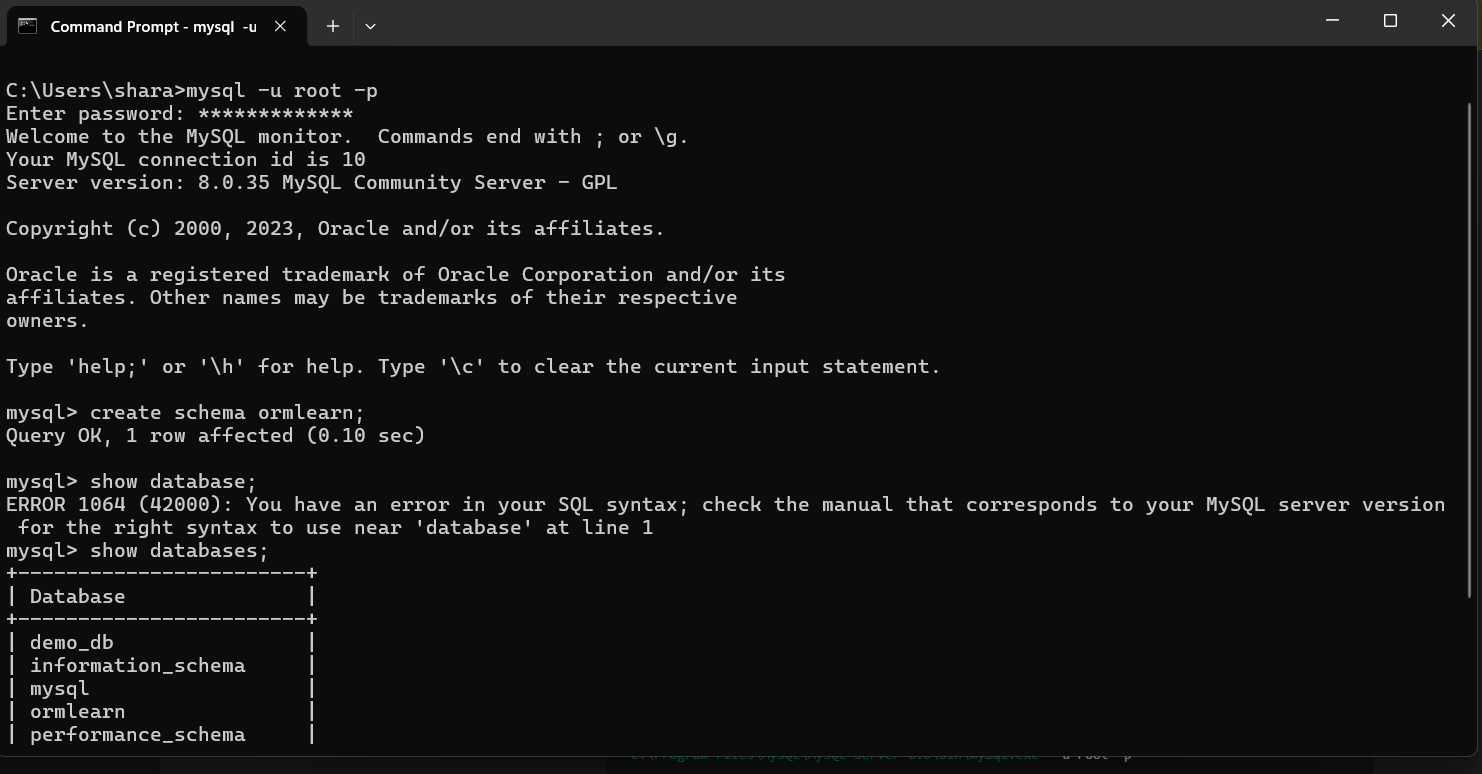
**Create a Eclipse Project using Spring Initializr**



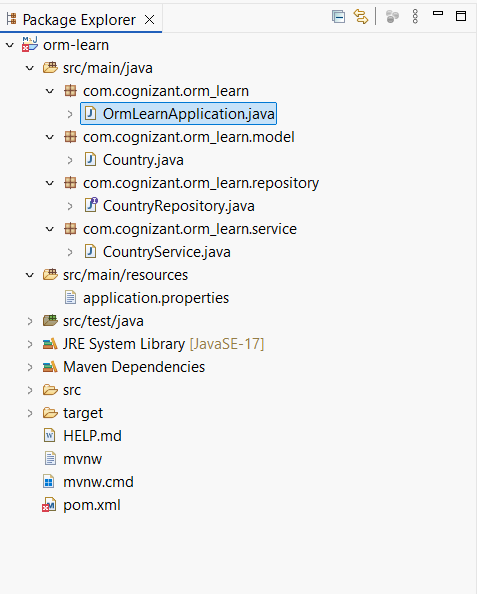
Extract the zip in root folder to Eclipse Workspace



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



**SNAPSHOT OF THE SCHEMA CREATED ACCORDING TO GIVEN INSTRUCTION**

****

**NOW THE CODE OF EACH FILE WILL BE GIVEN BELOW**

**Pom.xml**

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<parent>

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

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

<version>3.5.3</version>

<relativePath/> <!-- lookup parent from repository -->

</parent>

<groupId>com.cognizant</groupId>

<artifactId>orm-learn</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>orm-learn</name>

<description>Demo project for Spring Data JPA and Hibernate</description>

<url/>

<licenses>

<license/>

</licenses>

<developers>

<developer/>

</developers>

<scm>

<connection/>

<developerConnection/>

<tag/>

<url/>

</scm>

<properties>

<java.version>17</java.version>

</properties>

<dependencies>

<dependency>

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

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

</dependency>

<dependency>

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

<artifactId>spring-boot-devtools</artifactId>

<scope>runtime</scope>

<optional>true</optional>

</dependency>

<dependency>

<groupId>com.mysql</groupId>

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

<scope>runtime</scope>

</dependency>

<dependency>

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

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

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

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

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

</project>

**Application.properties**

spring.application.name=orm-learn

# 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=sharanya@1234

# Hibernate configuration

spring.jpa.hibernate.ddl-auto=validate

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

**OrmLearnApplication.java**

package com.cognizant.orm\_learn;

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.orm\_learn.model.Country;

import com.cognizant.orm\_learn.service.CountryService;

@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);

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");

}

}

**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 = "co\_code")

private String code;

@Column(name = "co\_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 org.springframework.transaction.annotation.Transactional;

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

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

@Service

public class CountryService {

@Autowired

private CountryRepository countryRepository;

@Transactional

public List<Country> getAllCountries() {

return countryRepository.findAll();

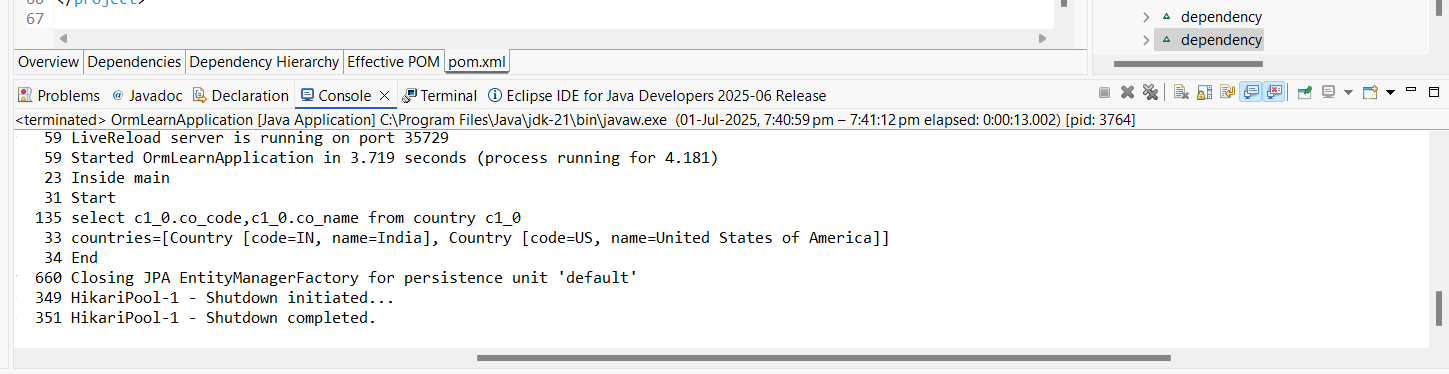
}

}

**Country table creation**



Output after running the main java code file where country being displayed above is the snapshot where the table is created and value is inserted.



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

**1. Introduction**

In modern Java applications, interacting with databases efficiently and cleanly is essential. Java developers use Object Relational Mapping (ORM) techniques to map Java objects to relational database tables. This hands-on activity explores the core concepts and differences between Java Persistence API (JPA), Hibernate, and Spring Data JPA, which are key technologies in this space.

**2. What is JPA?**

Java Persistence API (JPA) is a Java specification (defined by JSR 338) for managing relational data in Java applications. It provides a set of guidelines and annotations to simplify the interaction between Java classes and database tables.

- JPA is not an implementation—it is a standard.

- It allows developers to define how data should be stored, retrieved, and managed using annotations like @Entity, @Table, @Id, and @Column.

- To use JPA, we need an implementation such as Hibernate, EclipseLink, or OpenJPA.

**3. What is Hibernate?**

Hibernate is a powerful ORM framework that implements the JPA specification. It helps in bridging the gap between object-oriented programming and relational databases.

Key features of Hibernate:

- Automatically generates SQL queries based on Java objects.

- Provides a session-based API to interact with the database.

- Supports features like lazy loading, caching, and cascading.

However, using Hibernate directly often requires writing a lot of boilerplate code to manage sessions, transactions, and exceptions.

**4. What is Spring Data JPA?**

Spring Data JPA is a part of the larger Spring ecosystem. It builds on top of JPA and Hibernate to offer a more declarative and simplified approach to database access.

Highlights of Spring Data JPA:

- Reduces boilerplate code by using interfaces like JpaRepository.

- Automatically generates methods for common operations such as save(), findAll(), delete(), etc.

- Integrates easily with Spring Boot and manages transactions and exceptions automatically.

- Makes development faster and more maintainable.

**5. Code Comparison**

Below is a comparison of how to perform the same operation (saving an employee) using Hibernate and Spring Data JPA.

**A. 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;

}

**B. Spring Data JPA (Simplified)**

EmployeeRepository.java

public interface EmployeeRepository extends JpaRepository<Employee, Integer> {

}

EmployeeService.java

@Autowired

private EmployeeRepository employeeRepository;

@Transactional

public void addEmployee(Employee employee) {

employeeRepository.save(employee);

}

**6. Conclusion**

JPA, Hibernate, and Spring Data JPA are all crucial components in Java-based applications that interact with databases. While JPA sets the rules, Hibernate provides the engine, and Spring Data JPA simplifies the entire process. By reducing boilerplate and integrating seamlessly with Spring Boot, Spring Data JPA enables developers to focus more on business logic rather than repetitive data access code.