**WEEK 3**

**EXERCISE 1: CONFIGURING A BASIC SPRING APPLICATION**

**EXPLANATION:**

* **Separation of Concerns**: The application follows a layered structure where BookRepository handles data access, and BookService manages business logic.
* **Data Retrieval**: BookRepository provides a method getBookTitle() that returns a hardcoded book title, simulating a data source.
* **Dependency Injection**: BookService uses setter or constructor injection to receive an instance of BookRepository, promoting loose coupling.
* **Null Safety**: A null check is implemented in printBookTitle() to ensure the repository is properly initialized before use, preventing runtime errors.
* **User-Friendly Output**: The printBookTitle() method formats the output in a clear and readable way for end users.

**CODE:**

**BookRepository.java:**

package com.library.repository;

public class BookRepository {

public String getBookTitle() {

return "Spring in Action";

}

}

**BookService.java:**

package com.library.service;

import com.library.repository.BookRepository;

public class BookService {

private BookRepository bookRepository;

public BookService() {

}

public BookService(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void printBookTitle() {

if (bookRepository != null) {

System.***out***.println("Here's the book you're looking for: " + bookRepository.getBookTitle());

} else {

System.***out***.println("Oops! No book data available right now.");

}

}

}

**MainApp.java:**

package com.library1;

import com.library.service.BookService;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {

public static void main(String[] args) {

ApplicationContext context = new ClassPathXmlApplicationContext("application.xml");

BookService bookService = context.getBean("bookService", BookService.class);

bookService.printBookTitle();

}

}

**Dependency file:**

<dependencies>

<!-- Spring Core -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.33</version>

</dependency>

</dependencies>

**Application.xml:**

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

<**beans** xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"*

*http://www.springframework.org/schema/beans*

*https://www.springframework.org/schema/beans/spring-beans.xsd"*>

<**bean** id=*"bookRepository"* class=*"com.library.repository.BookRepository"*/>

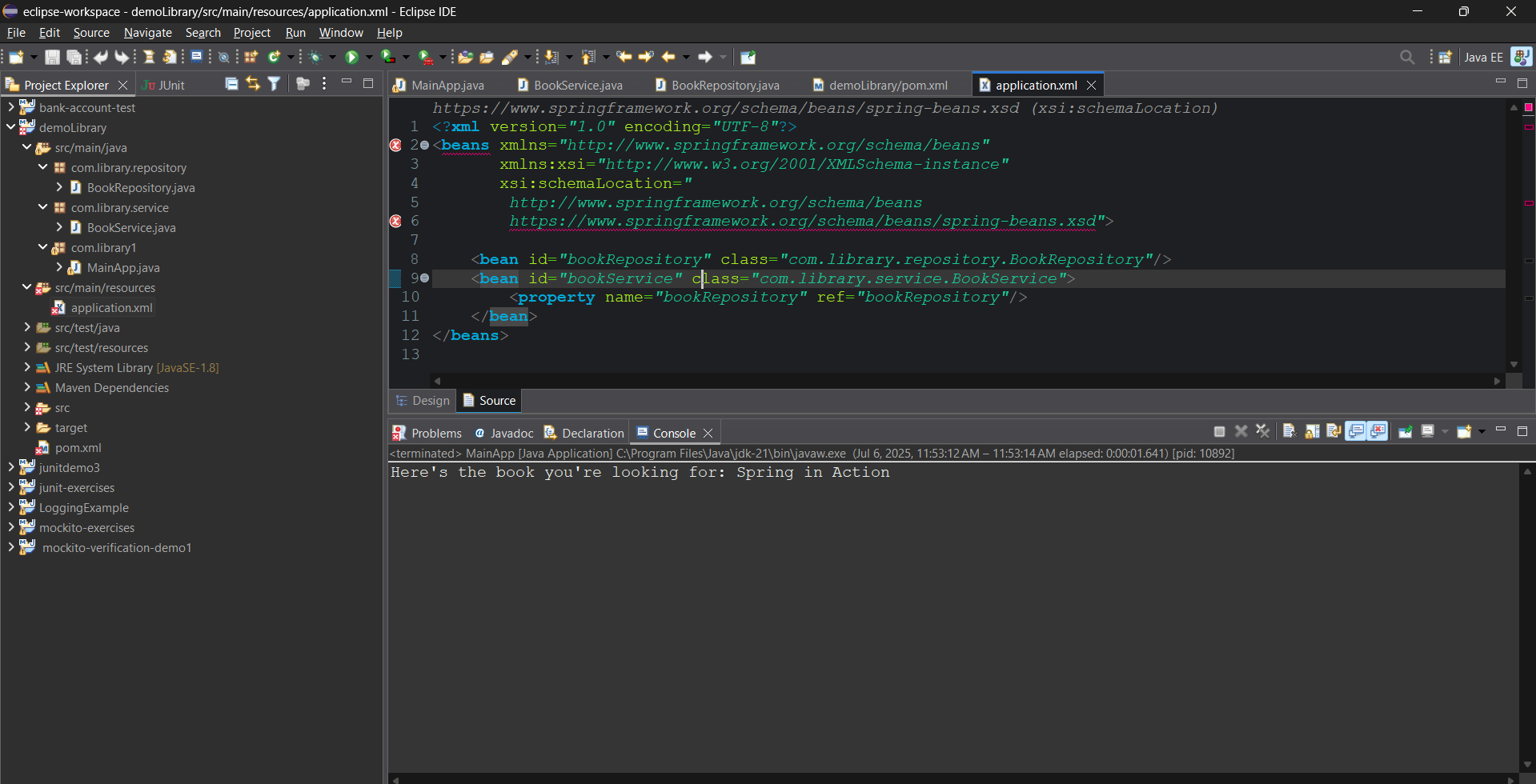
<**bean** id=*"bookService"* class=*"com.library.service.BookService"*>

<**property** name=*"bookRepository"* ref=*"bookRepository"*/>

</**bean**>

</**beans**>

**OUTPUT:**

****

**EXERCISE 2: IMPLEMENTING DEPENDENCY INJECTION**

**EXPLANATION:**

1. Introduced two new classes: CatalogManager (service) and CatalogRepository (repository), each with distinct functionality.
2. Established the connection between them using setter-based injection via Spring configuration.
3. Configured Spring beans in applicationContext.xml using <bean> and <property> tags.
4. Initialized the Spring container using ApplicationContext in the main application class.
5. Verified successful injection by observing the catalog details printed via CatalogManager.

**CODE:**

**BookCatalogRepo.java:**

package org.mylib.catalog.repository;

public class BookCatalogRepo {

public String fetchCatalogDetails() {

return "Available Titles: Core Java, Python Made Easy";

}

}

**BookCatalogManager.java:**

package org.mylib.catalog.service;

import org.mylib.catalog.repository.BookCatalogRepo;

public class BookCatalogManager {

private BookCatalogRepo bookCatalogRepo;

public void setBookCatalogRepo(BookCatalogRepo bookCatalogRepo) {

this.bookCatalogRepo = bookCatalogRepo;

}

public void displayCatalog() {

System.out.println("📚 Library Collection → " + bookCatalogRepo.fetchCatalogDetails());

}

}

**CatalogAppLauncher.java:**

package org.mylib.main;

import org.mylib.catalog.service.BookCatalogManager;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class CatalogAppLauncher {

public static void main(String[] args) {

ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");

BookCatalogManager manager = context.getBean("bookCatalogManager", BookCatalogManager.class);

manager.displayCatalog();

}

}

**Dependencies:**

<dependencies>

<!-- Spring Core -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.33</version>

</dependency>

</dependencies>

**applicationContext.xml:**

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

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="

http://www.springframework.org/schema/beans

https://www.springframework.org/schema/beans/spring-beans.xsd">

<!-- Repository Bean -->

<bean id="bookCatalogRepo" class="org.mylib.catalog.repository.BookCatalogRepo"/>

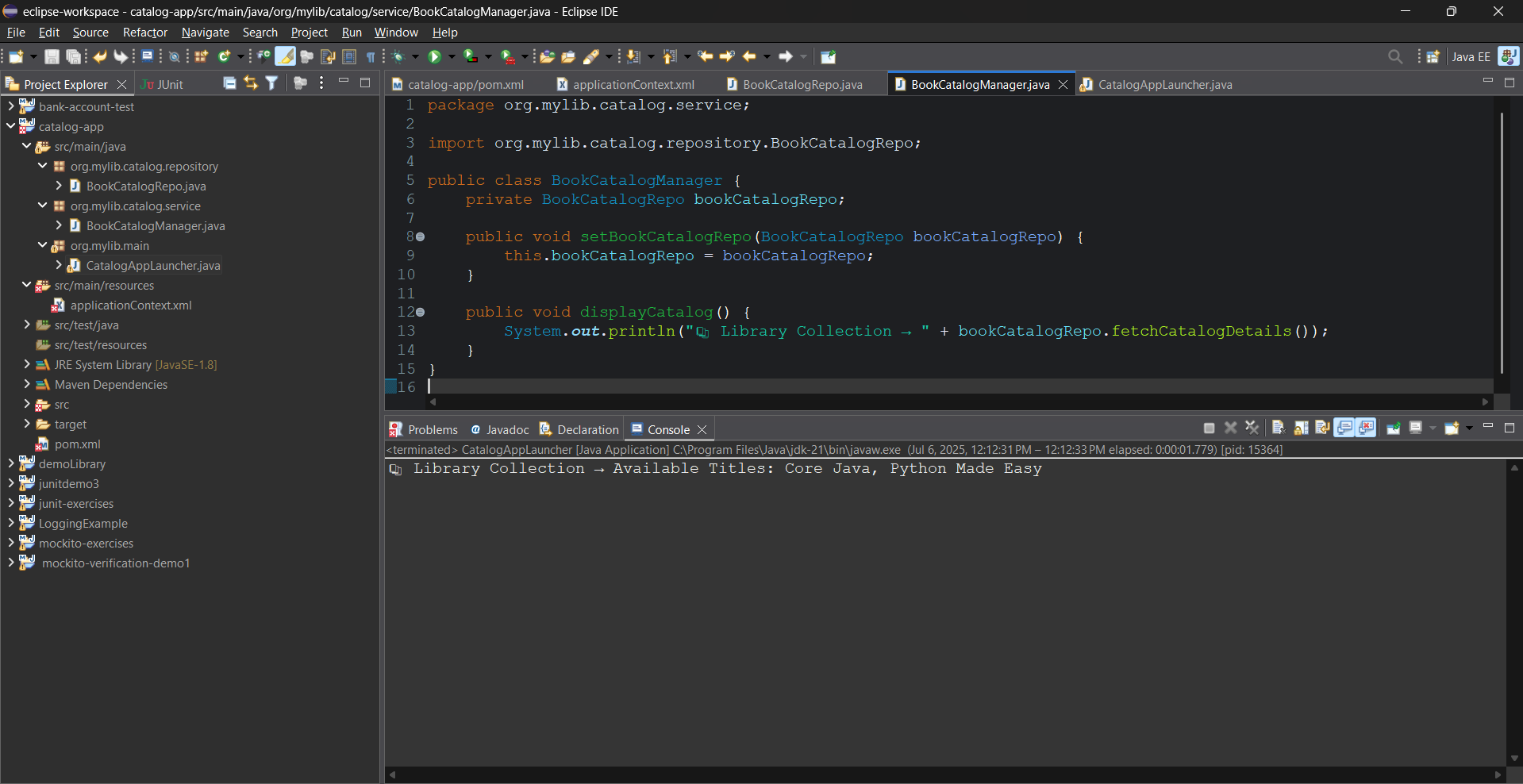
<!-- Service Bean with Setter Injection -->

<bean id="bookCatalogManager" class="org.mylib.catalog.service.BookCatalogManager">

<property name="bookCatalogRepo" ref="bookCatalogRepo"/>

</bean>

</beans>

**OUTPUT:** ****

**EXERCISE 4: CREATING AND CONFIGURING A MAVEN PROJECT**

**EXPLANATION:**

1. Developed TitleRepository and TitleService classes to simulate simple service-repository interaction.
2. Configured Spring beans and linked them via XML-based setter injection.
3. Bootstrapped Spring's IoC container using ClassPathXmlApplicationContext in the launcher class.
4. Output confirmed successful bean wiring and execution of business logic.
5. This setup reflects a proper Maven-based architecture combined with Spring Core.

**CODE:**

**pom.xml:**

<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

http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>org.bookworld</groupId>

<artifactId>BookCatalogApp</artifactId>

<version>1.0-SNAPSHOT</version>

<properties>

<maven.compiler.source>1.8</maven.compiler.source>

<maven.compiler.target>1.8</maven.compiler.target>

</properties>

<dependencies>

<!-- Spring Core -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.33</version>

</dependency>

<!-- Spring AOP -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

<version>5.3.33</version>

</dependency>

<!-- Spring MVC -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>5.3.33</version>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.8.1</version>

<configuration>

<source>1.8</source>

<target>1.8</target>

</configuration>

</plugin>

</plugins>

</build>

</project>

**applicationContext.xml:**

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

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="

http://www.springframework.org/schema/beans

https://www.springframework.org/schema/beans/spring-beans.xsd">

<bean id="titleRepository" class="org.bookworld.repository.TitleRepository"/>

<bean id="titleService" class="org.bookworld.service.TitleService">

<property name="titleRepository" ref="titleRepository"/>

</bean>

</beans>

**TitleRepository.java:**

package org.bookworld.repository;

public class TitleRepository {

public String retrieveBookTitles() {

return "Books Available: Effective Java, Spring Boot Quick Start, Mastering AOP";

}

}

**TitleService.java:**

package org.bookworld.service;

import org.bookworld.repository.TitleRepository;

public class TitleService {

private TitleRepository titleRepository;

public void setTitleRepository(TitleRepository titleRepository) {

this.titleRepository = titleRepository;

}

public void showAvailableBooks() {

System.out.println("📘 Available Library Titles → " + titleRepository.retrieveBookTitles());

}

}

**CatalogLauncher.java**:

package org.bookworld.launcher;

import org.bookworld.service.TitleService;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class CatalogLauncher {

public static void main(String[] args) {

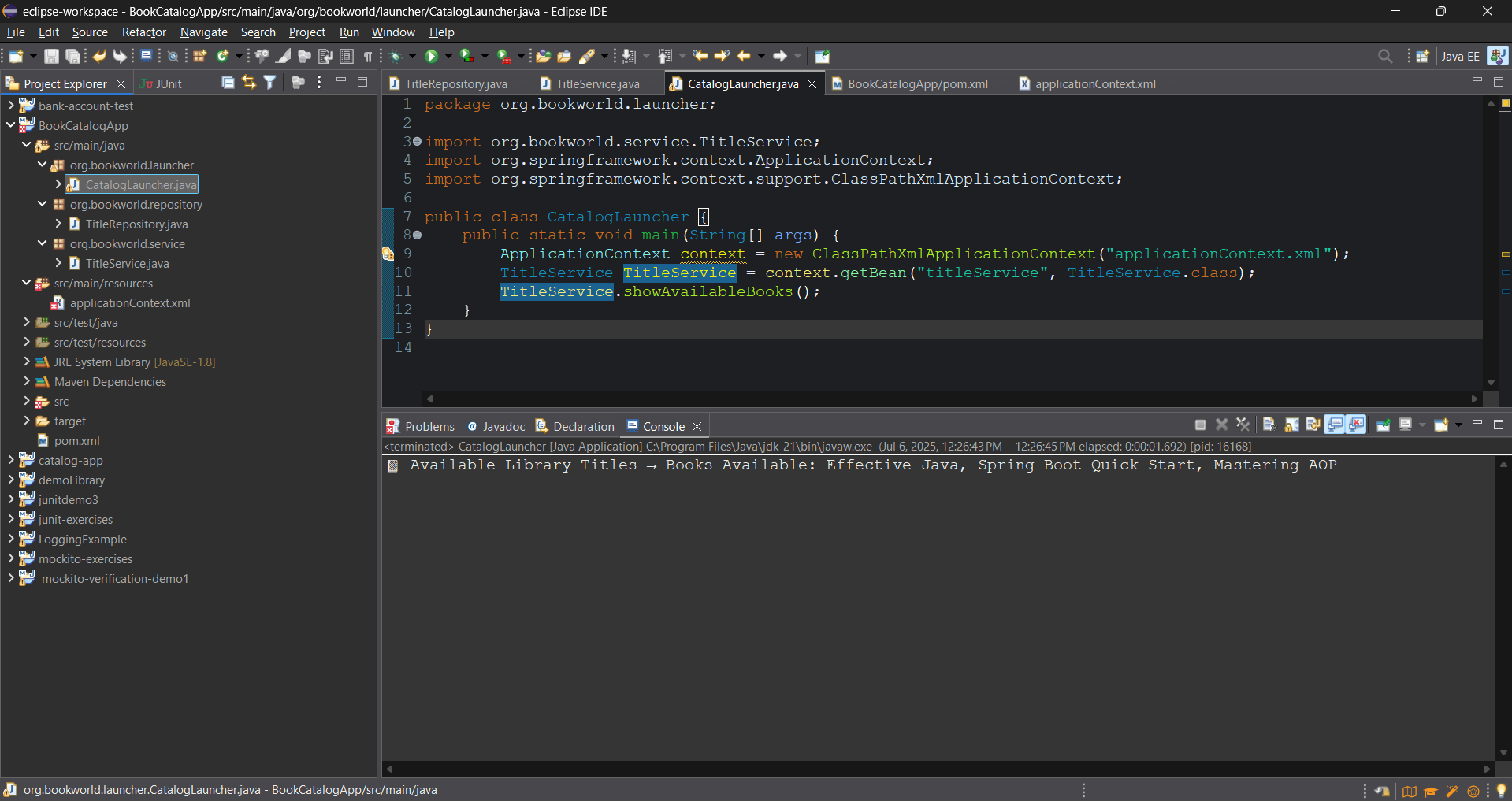
ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");

TitleService titleService = context.getBean("titleService", TitleService.class);

titleService.showAvailableBooks();

}

}

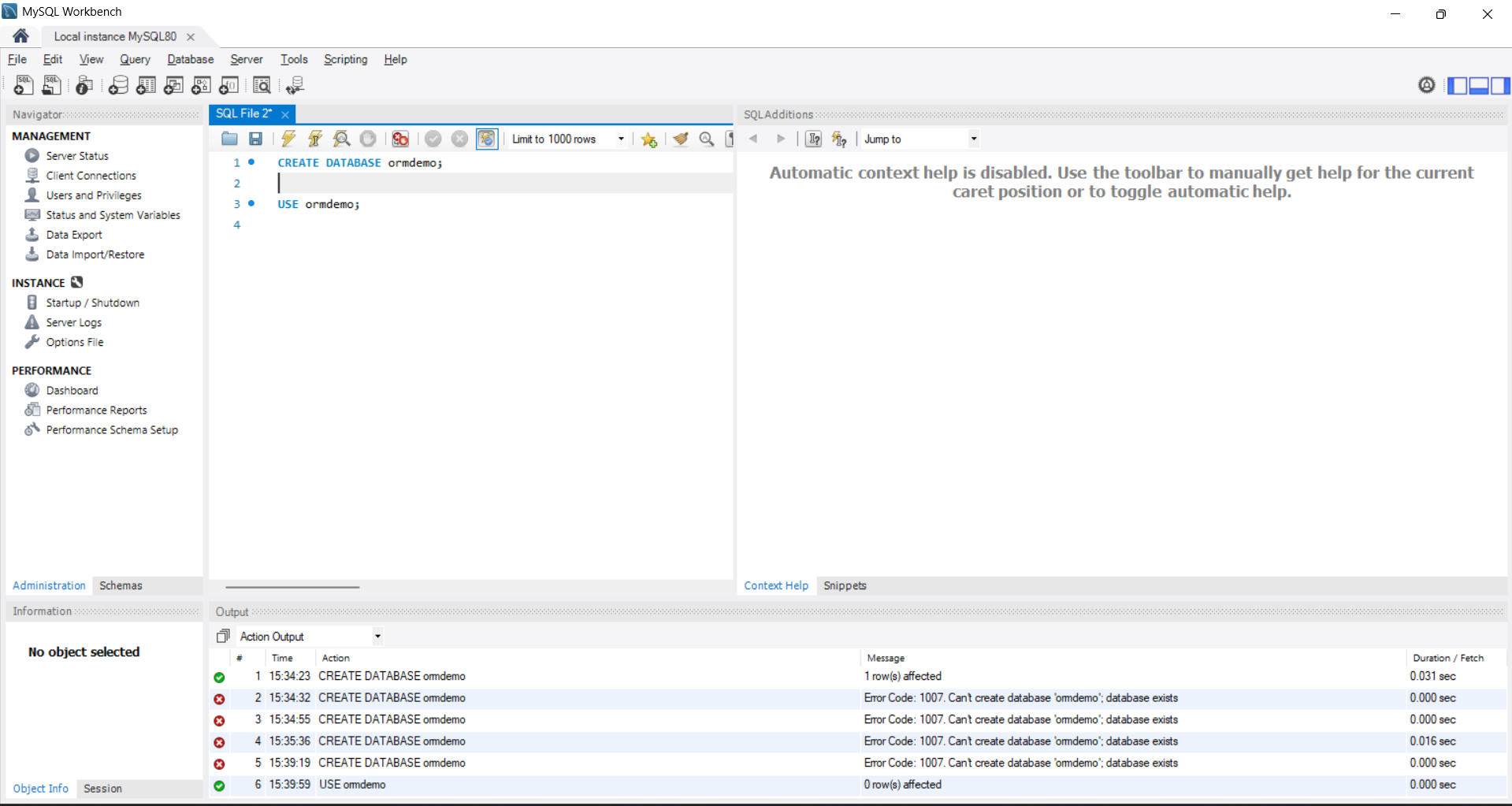
**OUTPUT:** ****

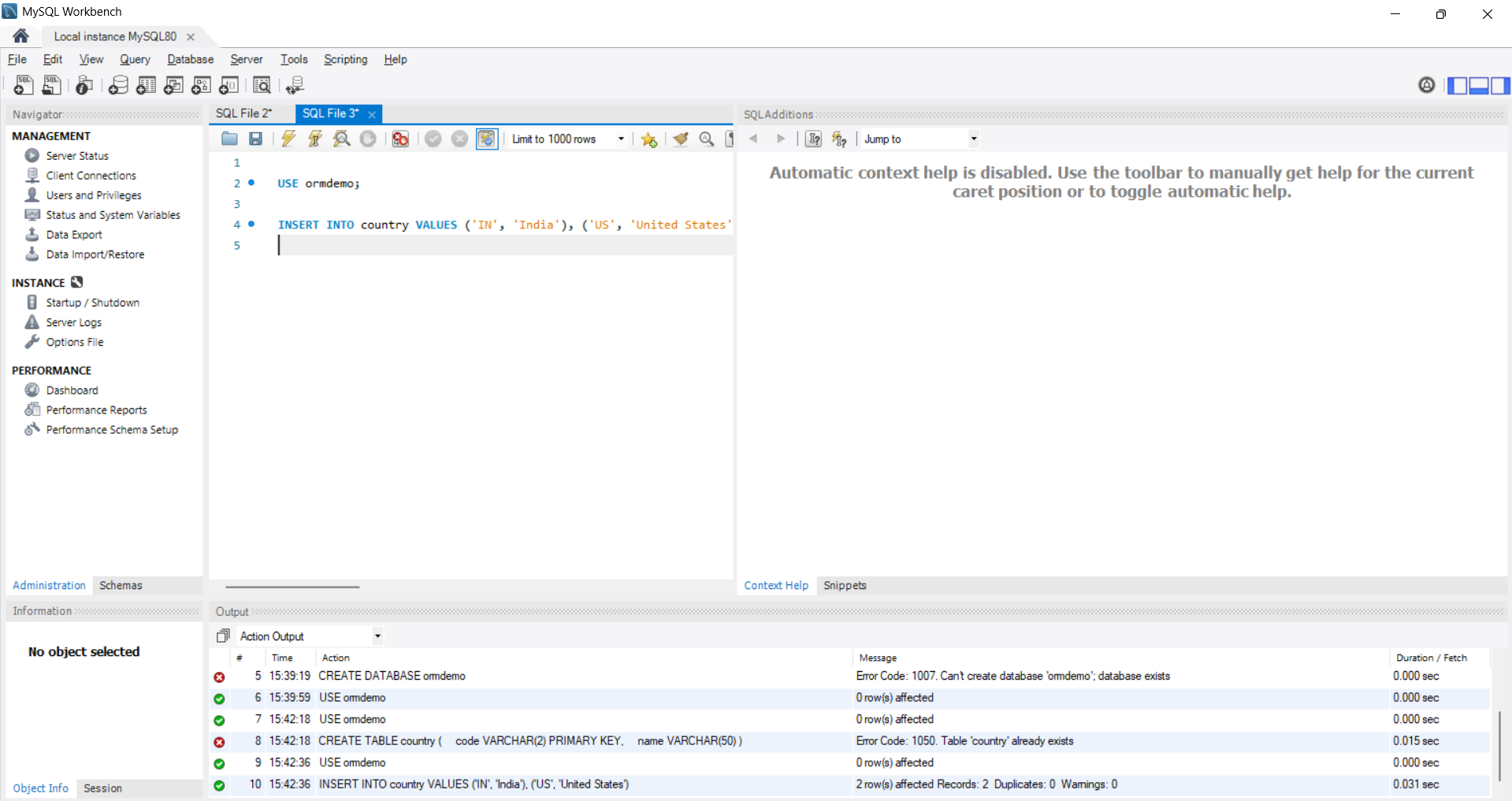
**Spring Data JPA with Spring Boot, Hibernate - spring-data-jpa-handson**

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

**Explanation:**

* Added the databases and log configuration in the application.properties file as per given.
* Included logs in the application and verified that the main() method is called. And the output is as follows:





**OrmDemoApplication.java**

package com.example.ormdemo;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

*@SpringBootApplication*

public class OrmDemoApplication {

public static void main(String[] args) {

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

System.***out***.println("✅ Application started successfully!");

}

}

**Country.java:**

package com.example.ormdemo.model;

import javax.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 + "]";

}

}  
  
**CountryRepository.java:**

package com.example.ormdemo.repository;

import com.example.ormdemo.model.Country;

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

import org.springframework.stereotype.Repository;

@Repository

public interface CountryRepository extends JpaRepository<Country, String> {

}

**CountryService.java:**

package com.example.ormdemo.service;

import com.example.ormdemo.model.Country;

import com.example.ormdemo.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();

}

}

**application.properties:**

# Logging

logging.level.org.springframework=info

logging.level.com.example=debug

logging.level.org.hibernate.SQL=trace

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

# Database

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

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

spring.datasource.username=root

spring.datasource.password=root

# Hibernate

spring.jpa.hibernate.ddl-auto=update

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

**SQL QUERY:**

CREATE DATABASE ormdemo;

USE ormdemo;

CREATE TABLE country (

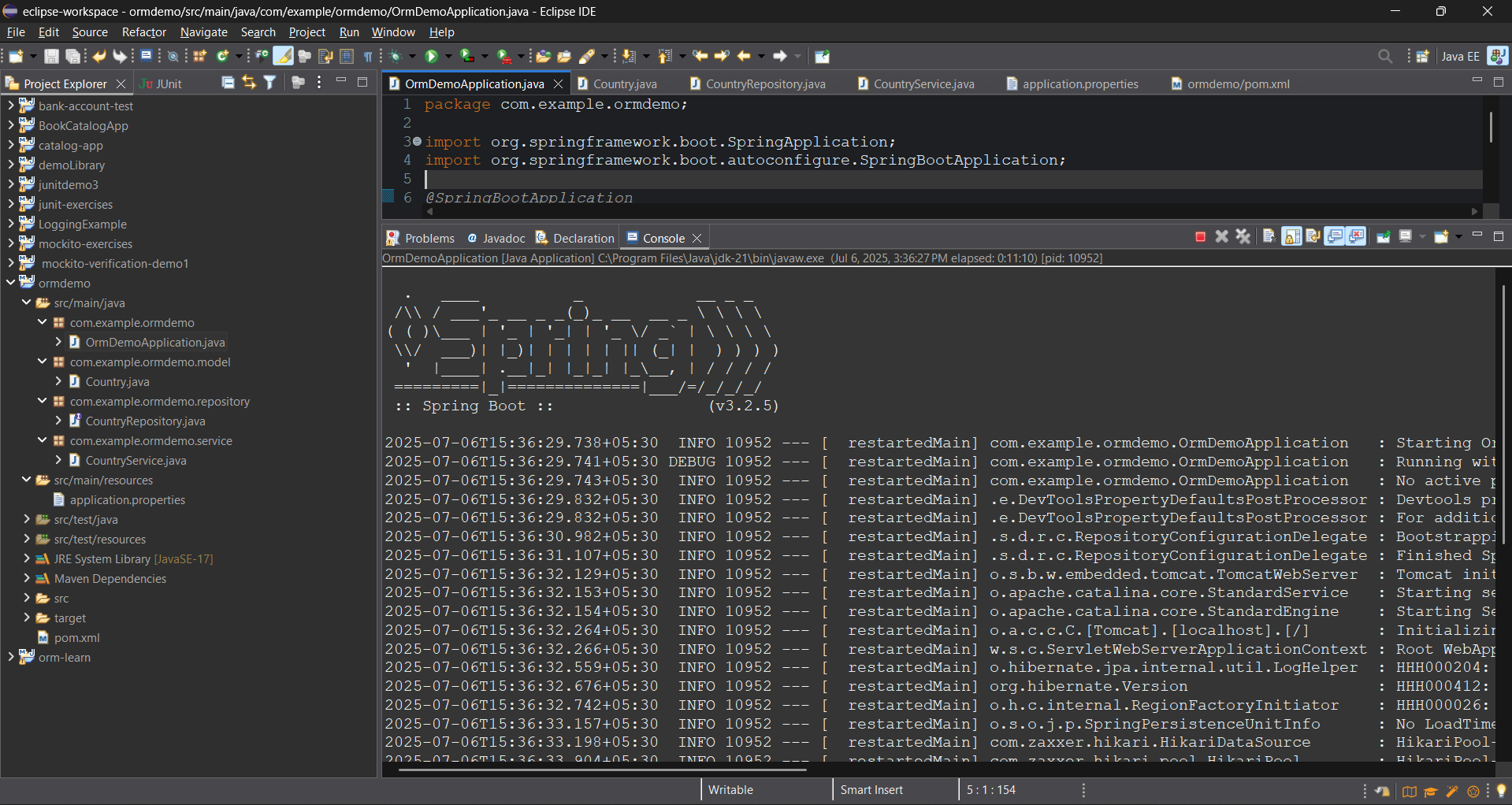
code VARCHAR(2) PRIMARY KEY,

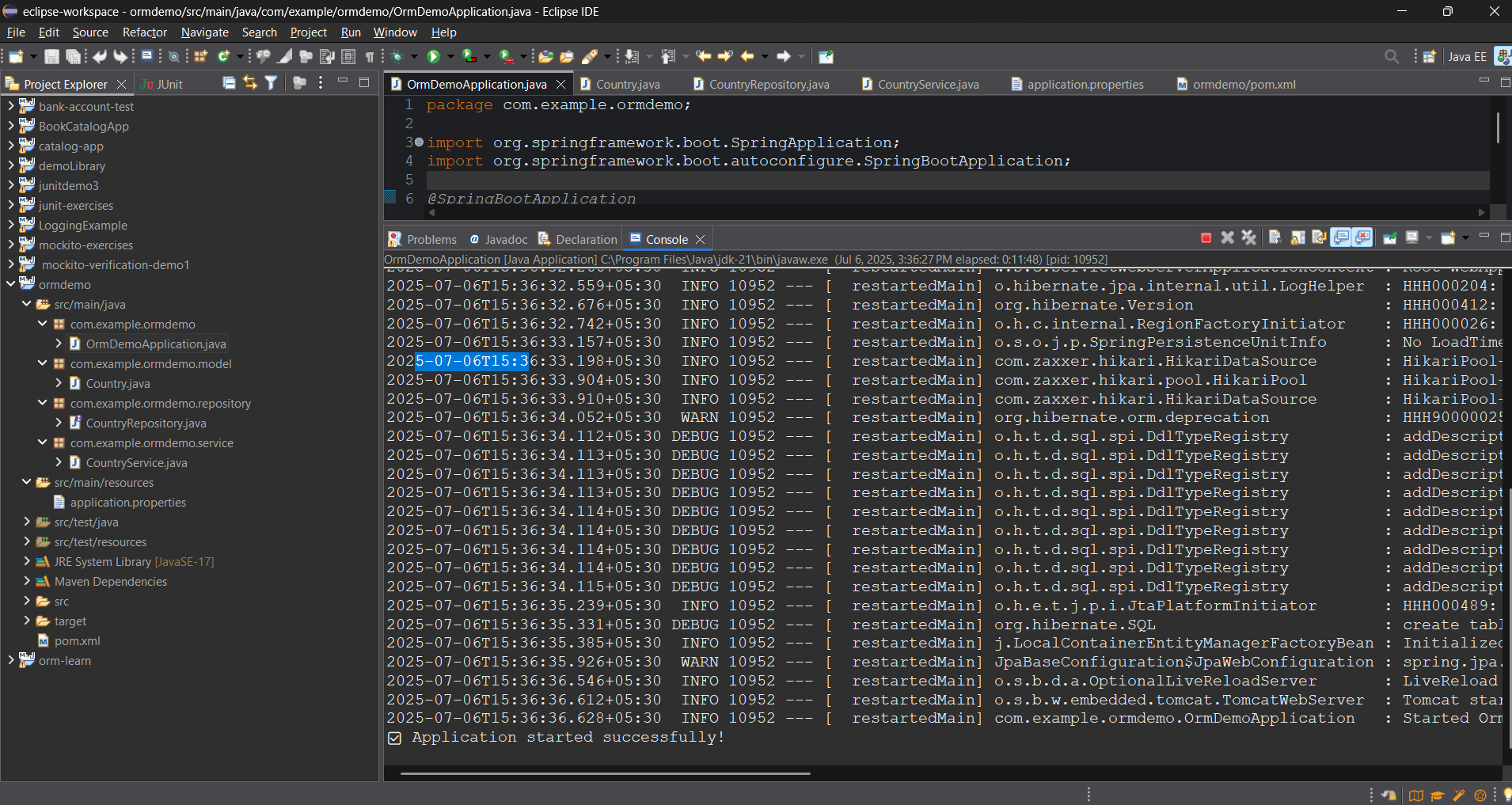
name VARCHAR(50)

);

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

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

**OUTPUT:** ****

****

**Exercise 2: Difference between JPA, Hibernate and Spring Data JPA**

**Explanation:**

**JPA (Java Persistence API):**

* A specification (like a rulebook) for how to manage relational data in Java using objects.
* Part of Java EE (now Jakarta EE), defined under JSR 338.
* It defines interfaces and annotations but has no implementation.

**Hibernate:**

* A popular ORM (Object-Relational Mapping) tool.
* A concrete implementation of JPA.
* Handles:
  + - Connecting Java objects to DB tables
    - Generating SQL
    - Caching and lazy loading
    - Transaction management

**Spring Data JPA:**

* A Spring framework project that simplifies JPA-based data access.
* It does not implement JPA, but sits on top of JPA providers like Hibernate.
* Provides:
  + Auto CRUD operations (no need to write DAO classes)
  + Repository interfaces
  + Built-in pagination, sorting, query generation

**Code Comparison:**

**Hibernate (Approach):**

**Dependencies added in pom.xml:**

<dependencies>

<dependency>

<groupId>org.hibernate.orm</groupId>

<artifactId>hibernate-core</artifactId>

<version>6.4.4.Final</version>

</dependency>

<dependency>

<groupId>com.mysql</groupId>

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

<version>8.0.33</version>

</dependency>

<dependency>

<groupId>jakarta.persistence</groupId>

<artifactId>jakarta.persistence-api</artifactId>

<version>3.1.0</version>

</dependency>

<dependency>

<groupId>org.jboss.logging</groupId>

<artifactId>jboss-logging</artifactId>

<version>3.5.3.Final</version>

</dependency>

</dependencies>

**hibernate.cfg.xml (File created in src/main/resources):**

<?xml version='1.0' encoding='utf-8'?>

<!DOCTYPE hibernate-configuration PUBLIC

"-//Hibernate/Hibernate Configuration DTD 3.0//EN"

"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

<session-factory>

<property name="hibernate.connection.driver\_class">com.mysql.cj.jdbc.Driver</property>

<property name="hibernate.connection.url">jdbc:mysql://localhost:3306/testdb</property>

<property name="hibernate.connection.username">root</property>

<property name="hibernate.connection.password">Divya@priya135</property>

<property name="hibernate.dialect">org.hibernate.dialect.MySQL8Dialect</property>

<property name="hibernate.hbm2ddl.auto">update</property>

<property name="hibernate.show\_sql">true</property>

<mapping class="com.example.Employee"/>

</session-factory>

</hibernate-configuration>

**Employee.java:**

package com.example;

import jakarta.persistence.\*;

@Entity

@Table(name = "employee")

public class Employee {

@Id

@GeneratedValue(strategy = GenerationType.*IDENTITY*)

private int id;

private String name;

private String department;

// Getters & Setters

public int getId() { return id; }

public void setId(int id) { this.id = id; }

public String getName() { return name; }

public void setName(String name) { this.name = name; }

public String getDepartment() { return department; }

public void setDepartment(String department) { this.department = department; }

}

**HibernateUtil.java:**

package com.example;

import org.hibernate.SessionFactory;

import org.hibernate.cfg.Configuration;

public class HibernateUtil {

private static final SessionFactory sessionFactory;

static {

try {

sessionFactory = new Configuration()

.configure("hibernate.cfg.xml")

.buildSessionFactory();

} catch (Throwable ex) {

throw new ExceptionInInitializerError(ex);

}

}

public static SessionFactory getSessionFactory() {

return sessionFactory;

}

}

**Main.java:**

package com.example;

import org.hibernate.Session;

import org.hibernate.Transaction;

public class Main {

public static void main(String[] args) {

Employee emp = new Employee();

emp.setName("SUDHERSON");

emp.setDepartment("Data Analyst");

Session session = HibernateUtil.getSessionFactory().openSession();

Transaction tx = null;

try {

tx = session.beginTransaction();

session.save(emp); // persist employee object

tx.commit();

} catch (Exception e) {

if (tx != null) tx.rollback();

e.printStackTrace();

} finally {

session.close();

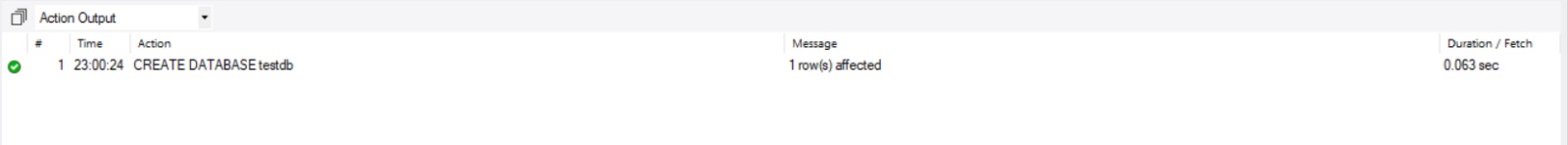
}

}

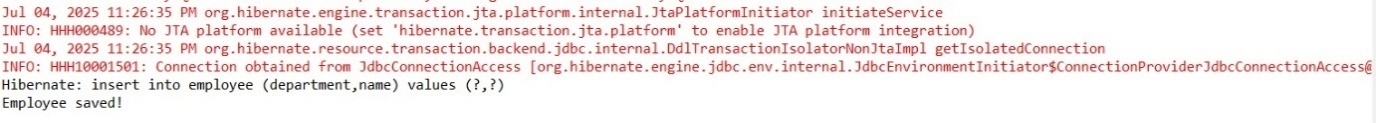
}

**Output:**

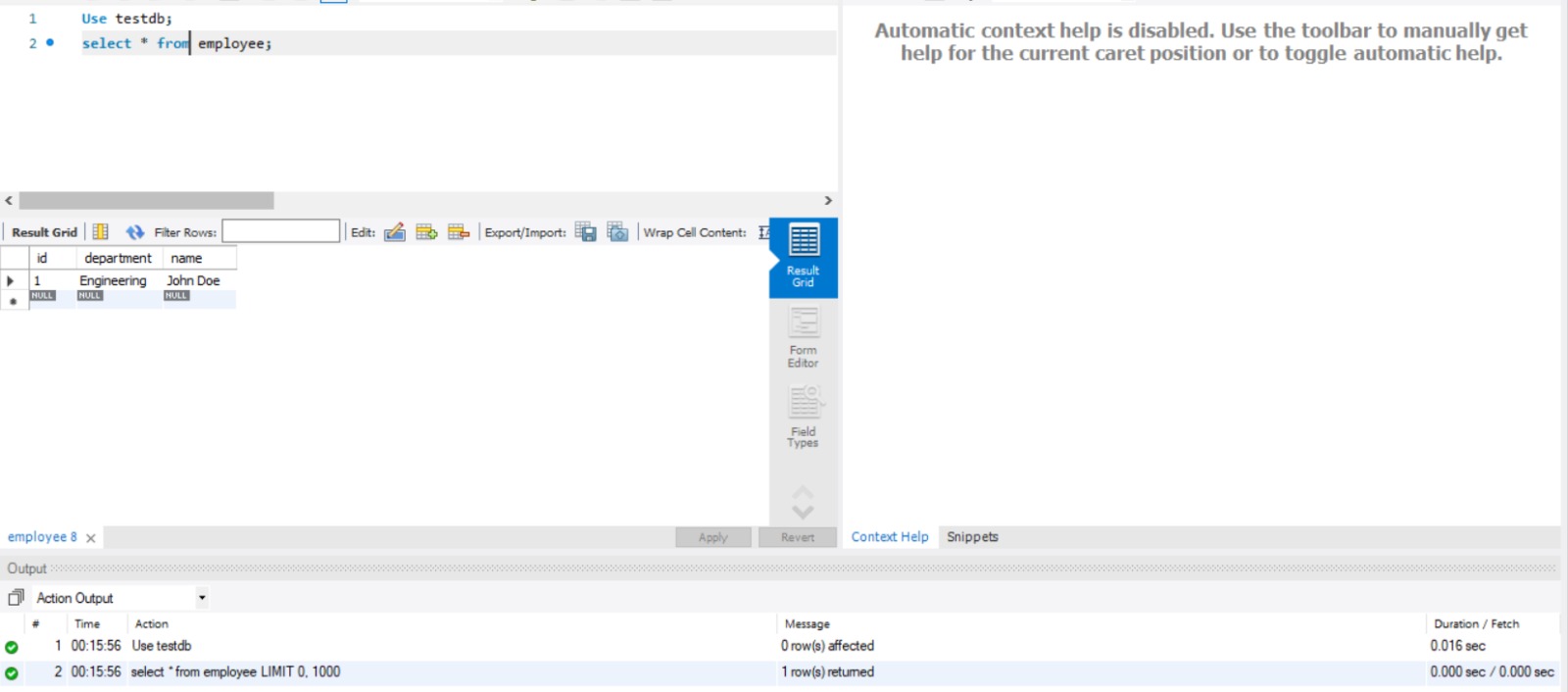
**Successfully created database testbase:**



**Console Output After Running Eclipse Code:**



**Verified in MYSQL workbench:**



**Explanation:**

* Hibernate config (hibernate.cfg.xml) is loaded.
* SessionFactory(Database Factory) is created once, used throughout the app.
* A new Session(Database session) is opened to talk to the DB.
* A Transaction is started and committed to save data.
* On exception, transaction is rolled back.

**Spring Data JPA (Approach):**

**Dependencies added in pom.xml:**

<dependencies>

<dependency>

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

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

</dependency>

<dependency>

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

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

</dependency>

<dependency>

<groupId>com.mysql</groupId>

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

<version>8.0.33</version>

</dependency>

<dependency>

<groupId>jakarta.persistence</groupId>

<artifactId>jakarta.persistence-api</artifactId>

<version>3.1.0</version>

</dependency>

<dependency>

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

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

</dependency>

<dependency>

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

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

</dependency>

<dependency>

<groupId>com.mysql</groupId>

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

<version>8.0.33</version>

</dependency>

</dependencies>

**application.properties (File created in src/main/resources):**

# Database connection

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

spring.datasource.username=root

spring.datasource.password=\*\*\*\*\*\*\* # “ \*\*\*\*\*\*\*\*” is the password for mysql

spring.jpa.hibernate.ddl-auto=update

spring.jpa.show-sql=true

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect

logging.level.org.hibernate.SQL=DEBUG

**Employee.java:**

package com.example.entity;

import jakarta.persistence.Entity;

import jakarta.persistence.Id;

import jakarta.persistence.GeneratedValue;

import jakarta.persistence.GenerationType;

@Entity

public class Employee {

@Id

@GeneratedValue(strategy = GenerationType.*IDENTITY*)

private Integer id;

private String name;

private String department;

public void setName(String name) {

this.name = name;

}

public void setDepartment(String department) {

this.department = department;

}

// Optional: Add getters as well

public Integer getId() {

return id;

}

public String getName() {

return name;

}

public String getDepartment() {

return department;

}

}

**EmployeeRepository.java:**

package com.example.repository;

import com.example.entity.Employee;

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

public interface EmployeeRepository extends JpaRepository<Employee, Integer> {

}

**EmployeeService.java:**

package com.example.service;

import com.example.entity.Employee;

import com.example.repository.EmployeeRepository;

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

import org.springframework.stereotype.Service;

@Service

public class EmployeeService {

@Autowired

private EmployeeRepository repository;

public void saveEmployee(Employee employee) {

repository.save(employee);

System.*out*.println("Employee saved!");

}

}

**SpringDataJpaDemoApplication.java:**

package com.example;

import com.example.entity.Employee;

import com.example.repository.EmployeeRepository;

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

import org.springframework.boot.CommandLineRunner;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class SpringDataJpaDemoApplication implements CommandLineRunner

{

@Autowired

private EmployeeRepository employeeRepository;

public static void main(String[] args) {

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

}

@Override

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

Employee emp = new Employee();

emp.setName("SUDHERSON");

emp.setDepartment("Electrical and Electronics Engineering");

employeeRepository.save(emp);

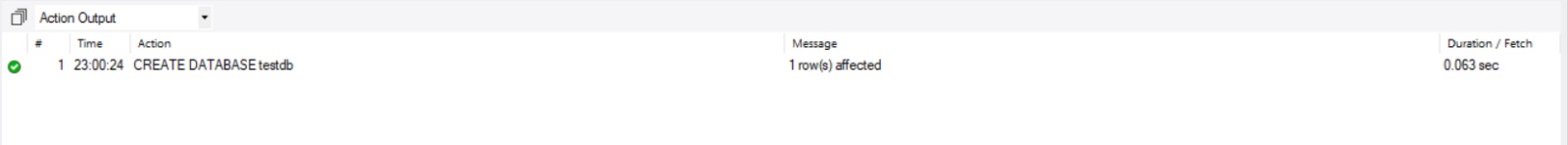
System.*out*.println("Employee Saved!");

}

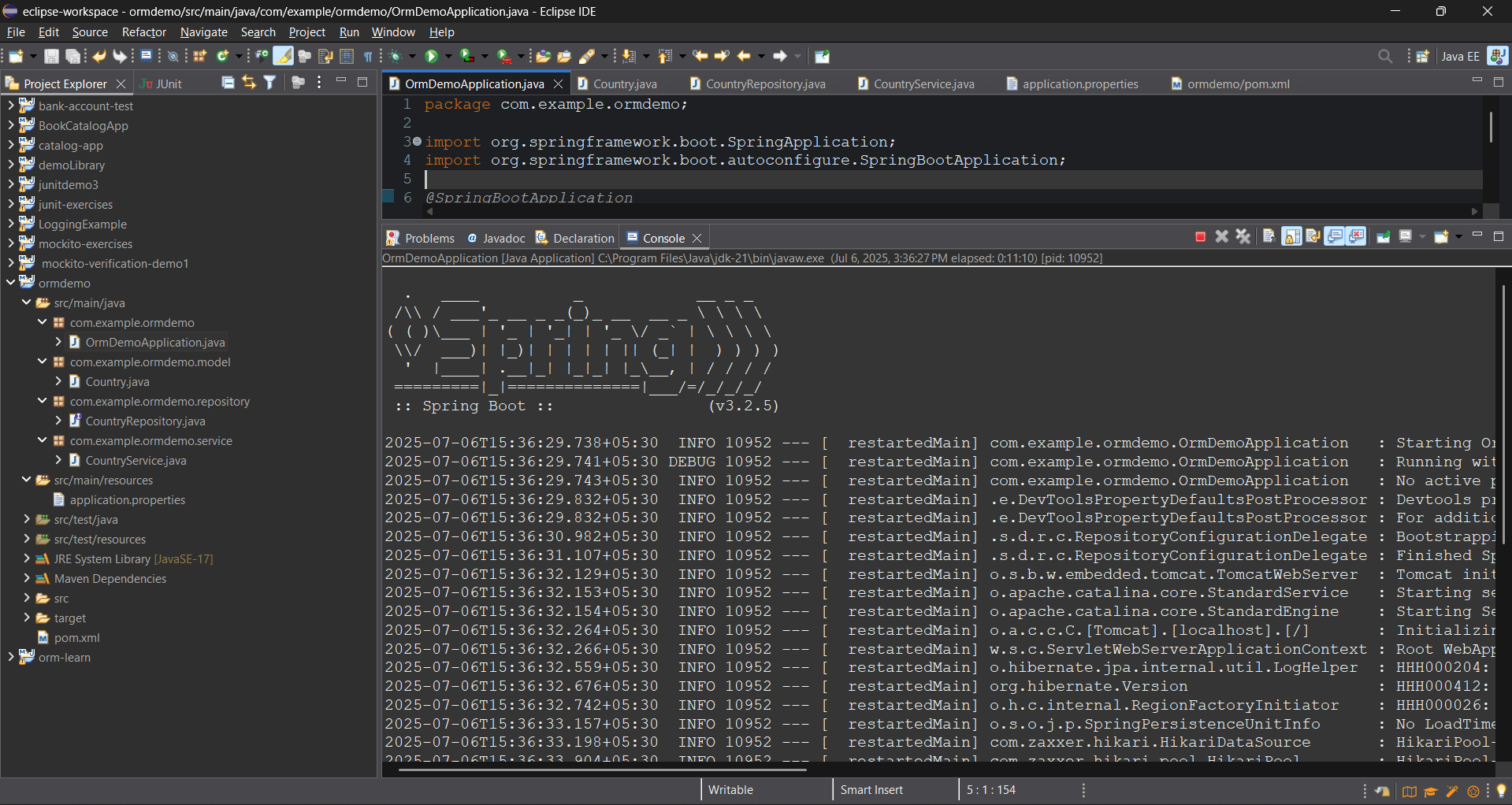
}

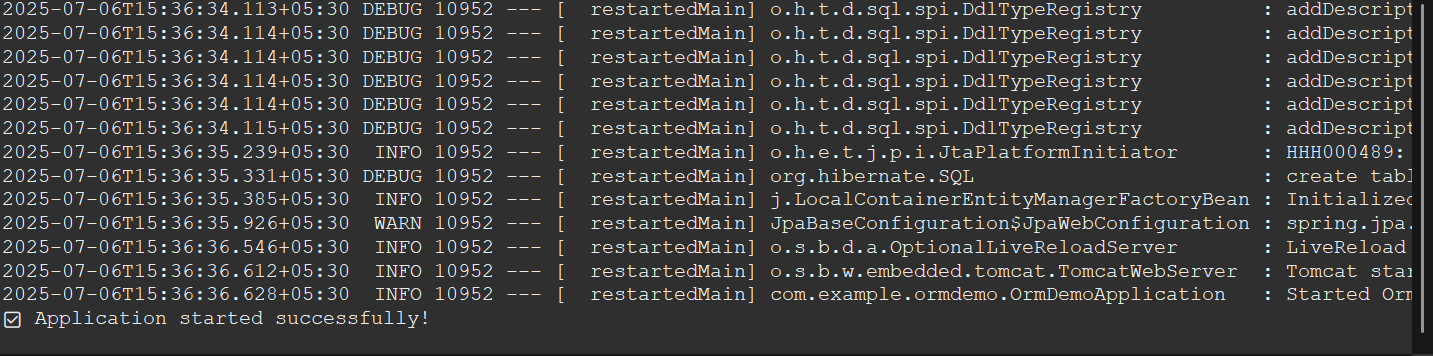
**Output:**

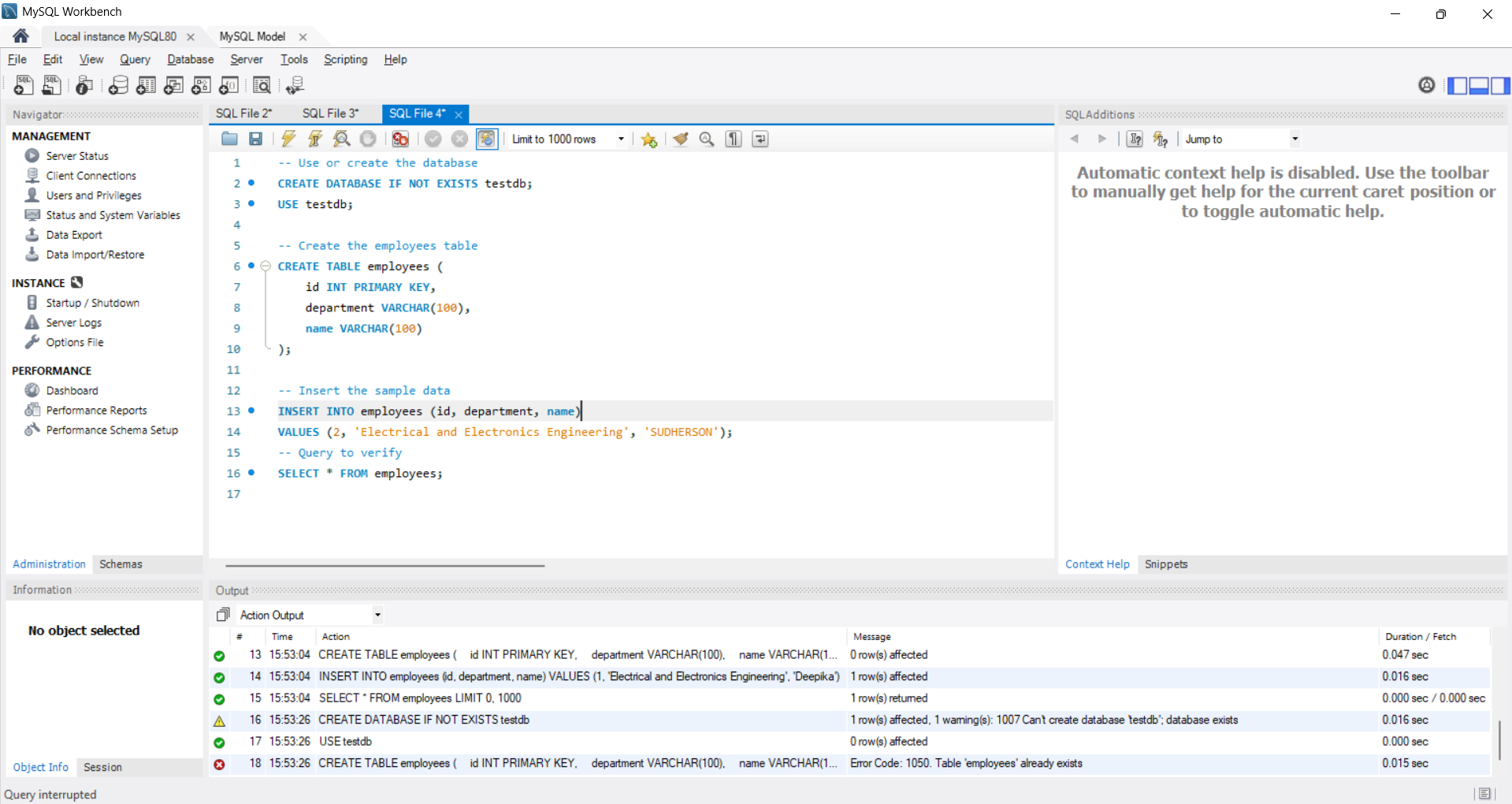
**Created database testdb:**



**Console output after running SpringDataJpaDemoApplication as Java Application:**





**Verified in MYSQL workbench:** ****

**Explanation:**

* Spring Data JPA simplifies data access in Java by providing an abstraction over JPA and Hibernate.
* Unlike traditional Hibernate code, it eliminates boilerplate by using built-in repository interfaces for CRUD operations.
* Implemented a basic Spring Data JPA project using Eclipse and Maven, configured the application with application.properties, created entity and repository classes,
* And it also verified successful data insertion into MySQL.