**Spring Data JPA: Country Management Project Documentation**

### **1. Introduction**

This project demonstrates the implementation of Spring Data JPA to interact with a MySQL database for managing a Country table. It covers the basic configuration, setup, and usage of Spring Boot, JPA, and MySQL with an example of fetching data from a database table.

### **2. Objectives**

* Understand ORM (Object-Relational Mapping) concepts and their benefits.
* Learn how to use Spring Data JPA for interacting with relational databases.
* Set up a Maven-based Spring Boot application.
* Perform read operations from a MySQL database.

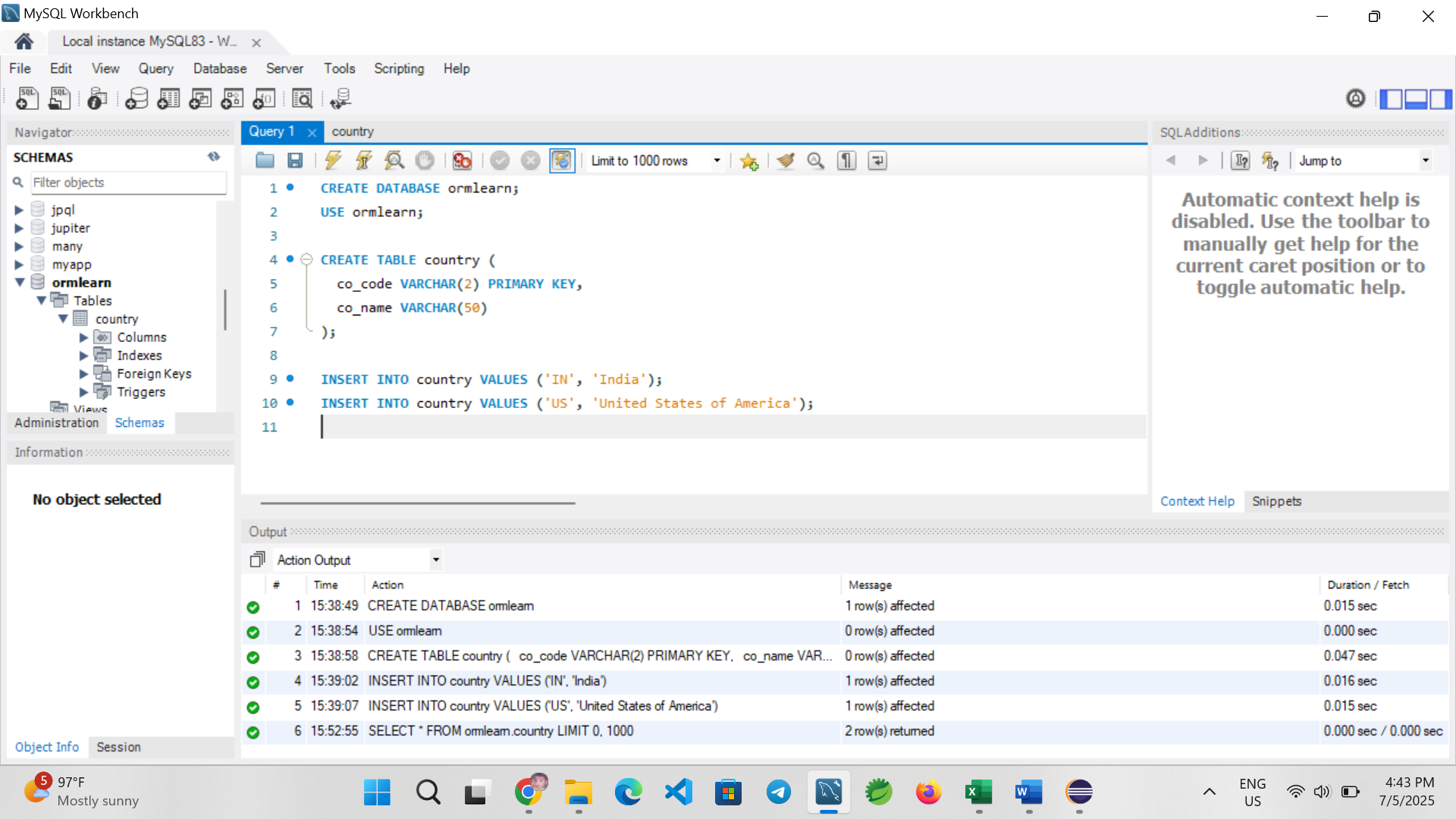
### **3. Tools and Technologies Used**

* Java 17
* Spring Boot 3.5.3
* Spring Data JPA
* MySQL Server 8.0
* Maven 3.6+
* Eclipse IDE
* SLF4J for logging

### **4. Project Setup**

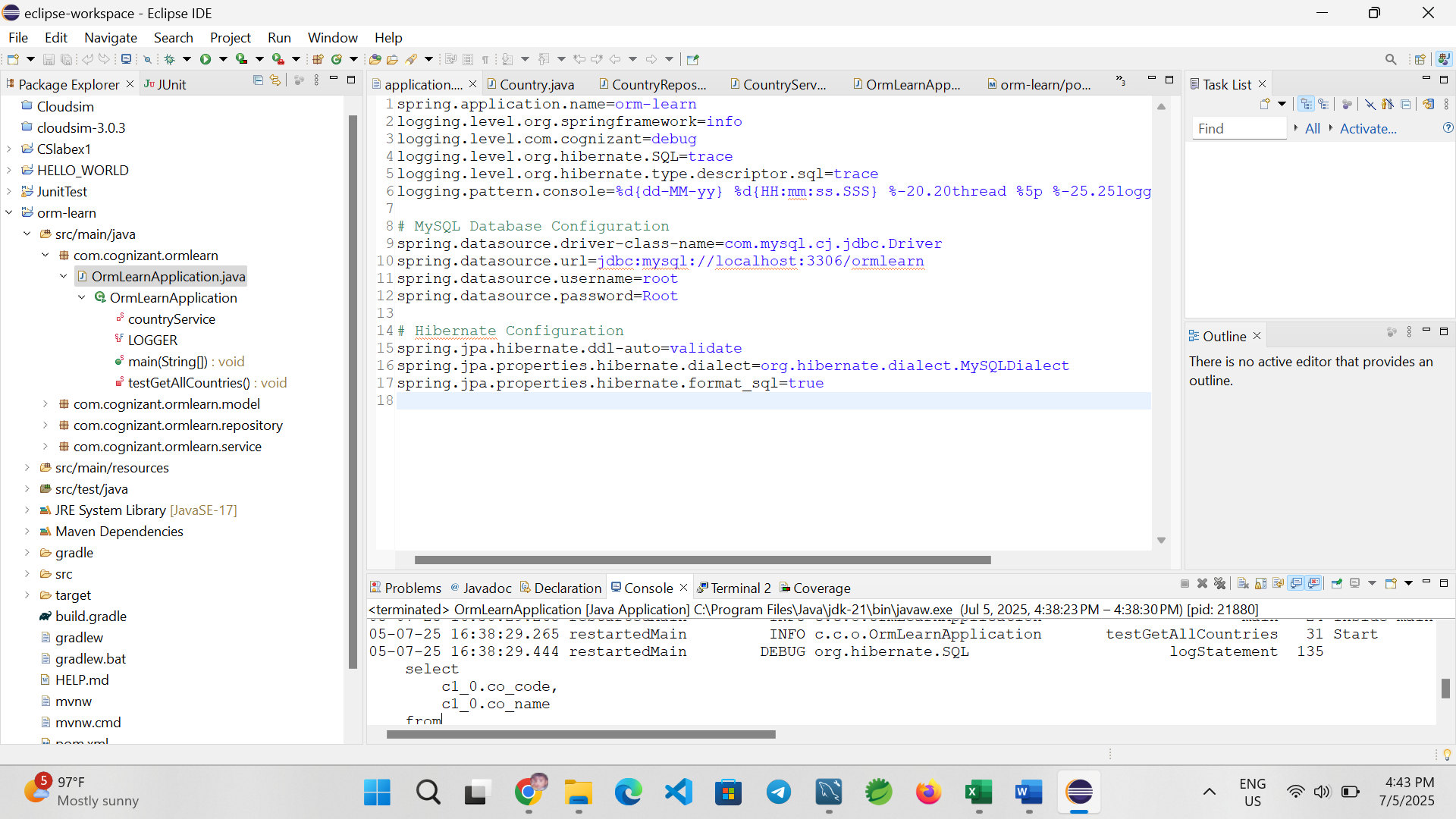
1. Visit <https://start.spring.io/>
2. Choose:
   * Project: Maven
   * Language: Java
   * Group: com.cognizant
   * Artifact: orm-learn
   * Dependencies: Spring Data JPA, MySQL Driver, Spring Boot DevTools
3. Generate and extract the zip file to the Eclipse workspace.
4. Import as Existing Maven Project.

### **5. Database Setup**



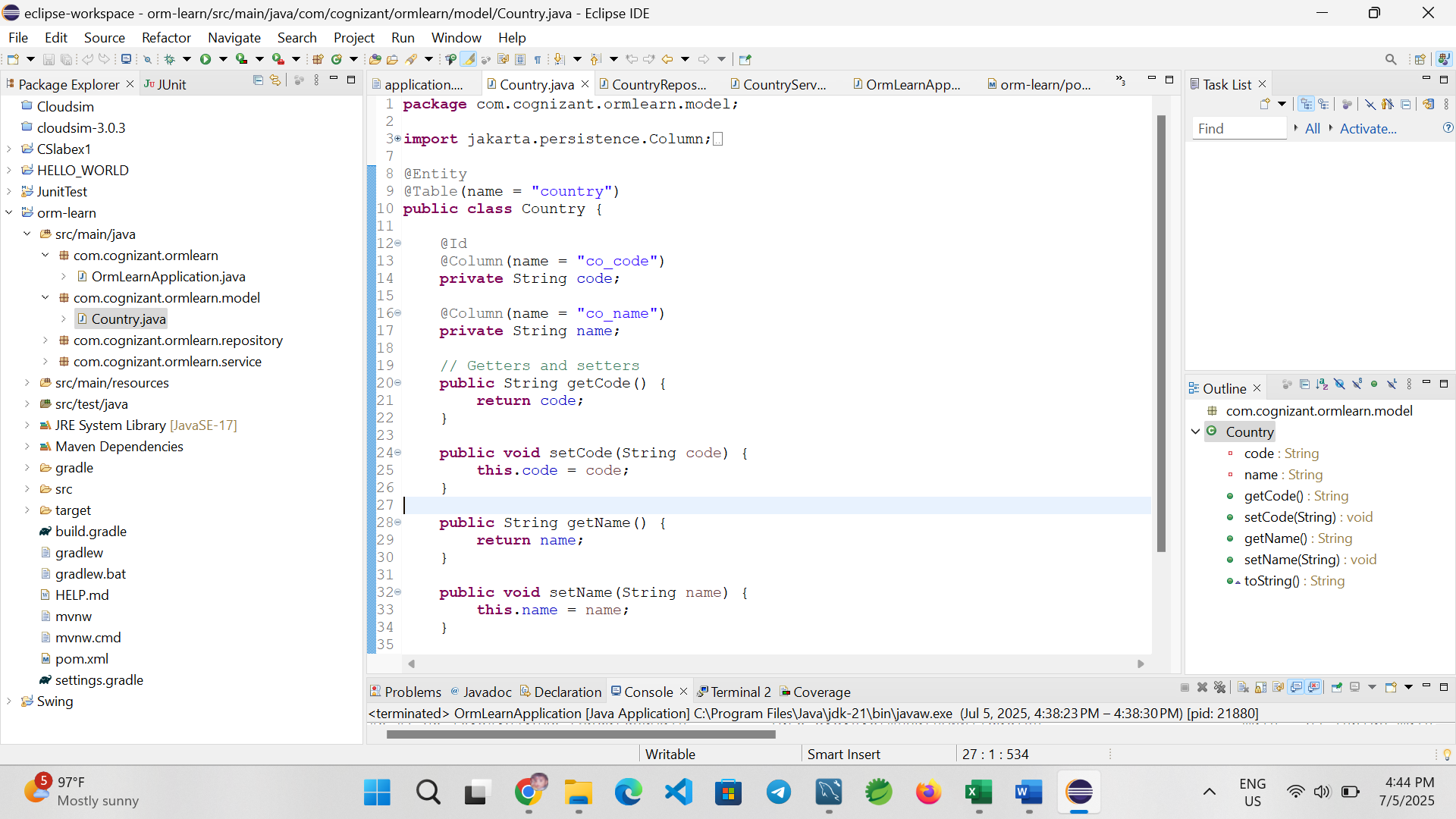
### **6. Configuration**

**application.properties**



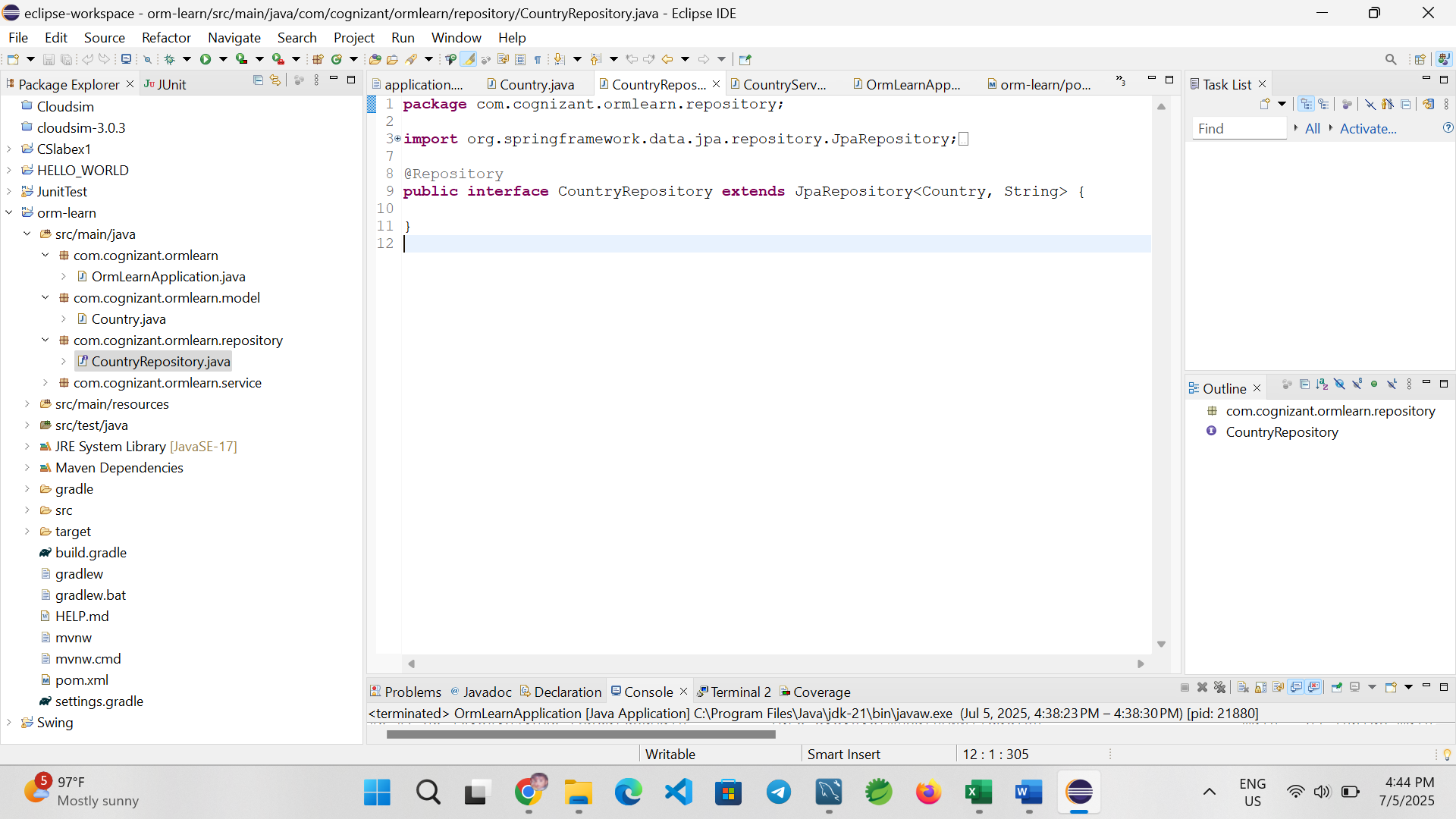
### **7. Entity Class**

**Country.java**



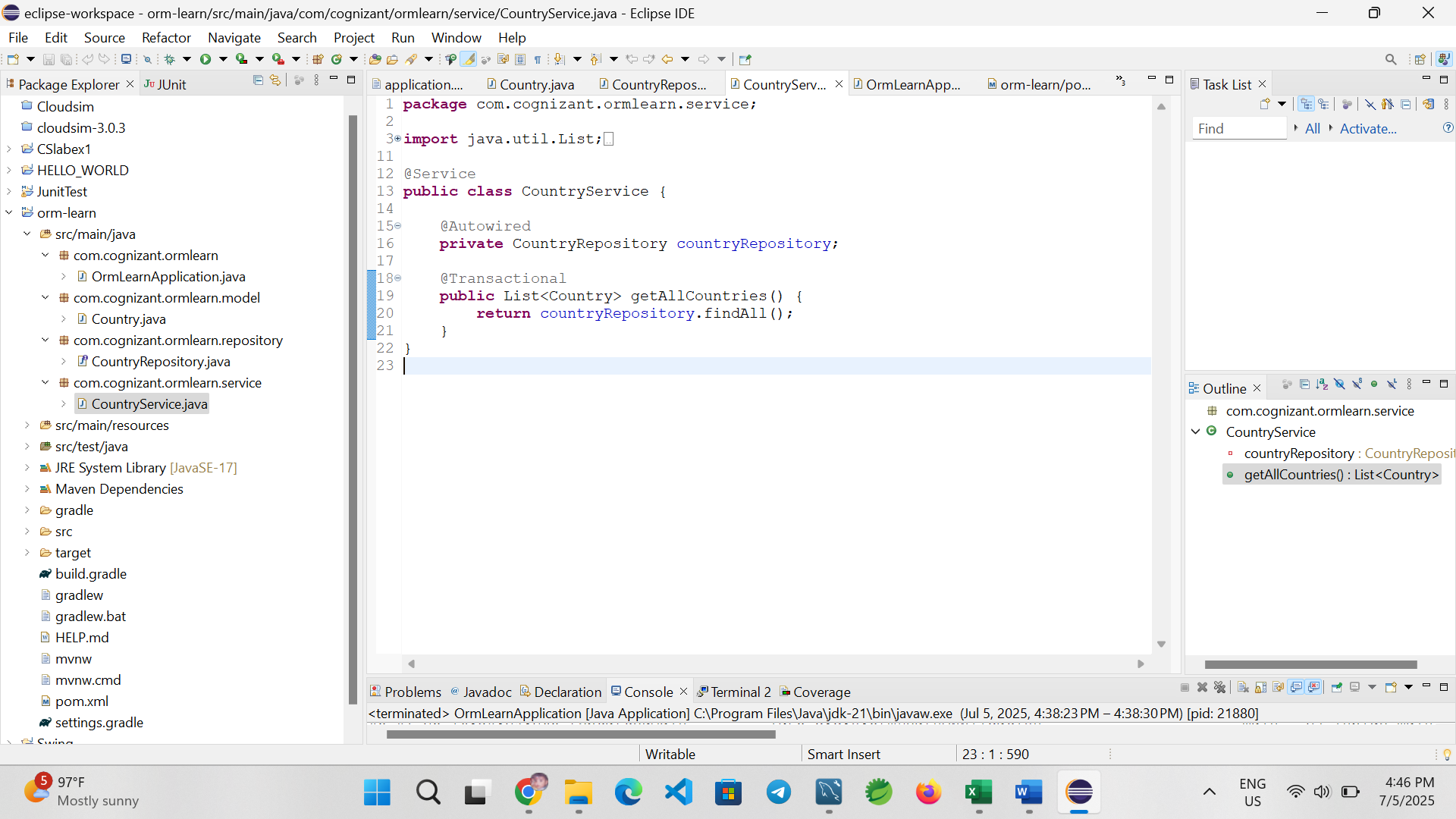
### **8. Repository Layer**

**CountryRepository.java**



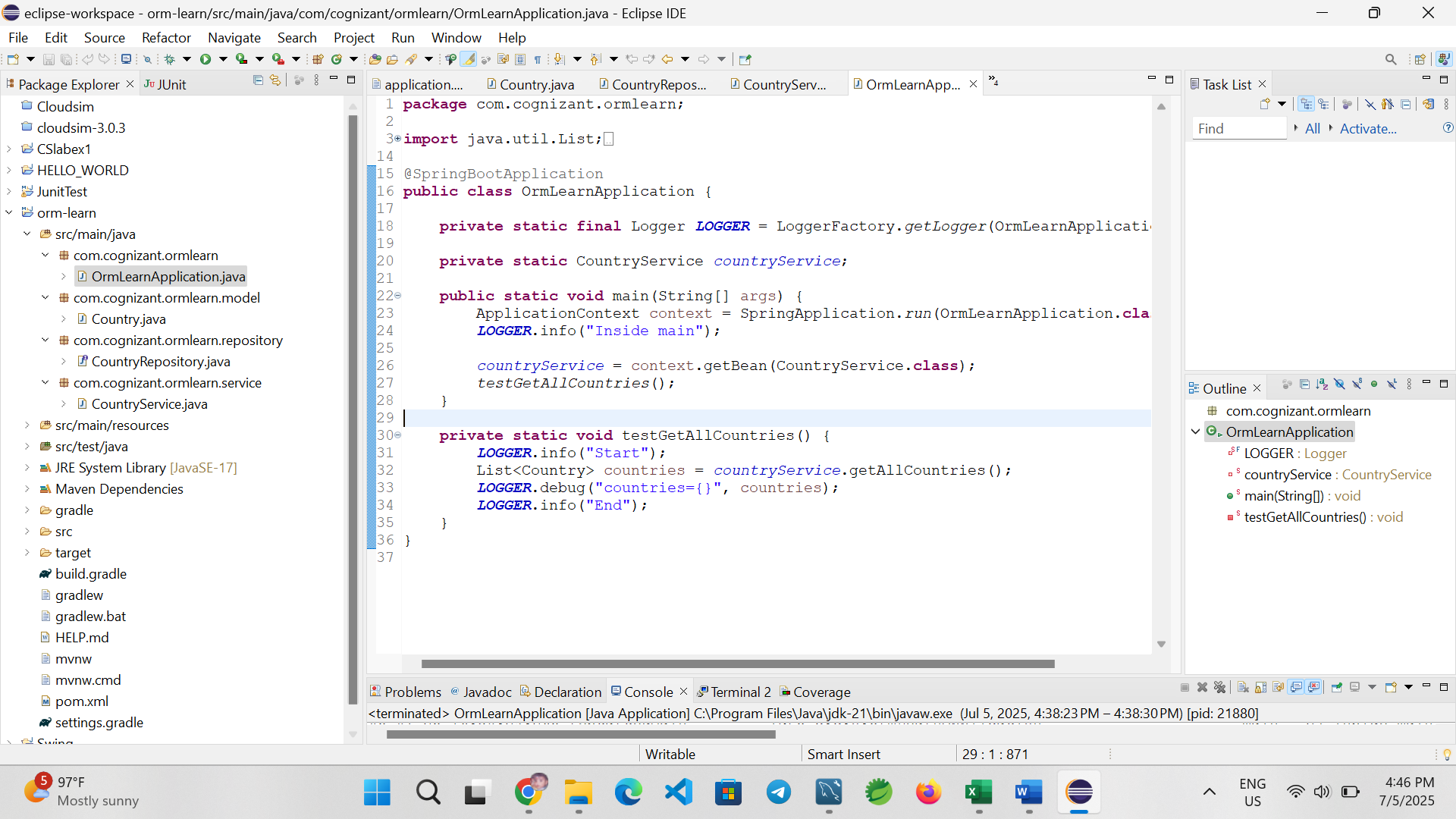
### **9. Service Layer**

**CountryService.java**

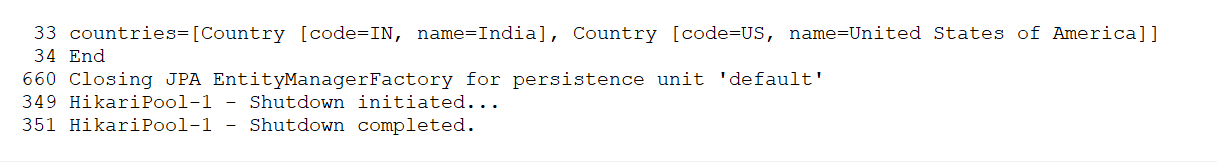


### **10. Main Application Class**

**OrmLearnApplication.java**



### **11. Output**



### **12. Conclusion**

The project successfully demonstrates how to configure and use Spring Data JPA with MySQL. It covers the creation of entity classes, repositories, services, and testing data retrieval using Spring Boot. The implementation confirms a working integration with the database and Spring Boot framework.

**Title: Simplified Employee Management using Spring Data JPA vs Hibernate**

**Introduction**

In enterprise Java applications, managing data access layers efficiently is critical. This document compares Java Persistence API (JPA), Hibernate, and Spring Data JPA and demonstrates their use through a simple Employee Management System.

**Technologies Used**

* Java 17
* Spring Boot 3.5.3
* Spring Data JPA
* Hibernate ORM
* MySQL
* Maven

**1. Understanding the Core Concepts**

**Java Persistence API (JPA)**

* A specification (JSR 338) for object-relational mapping.
* Provides annotations and interfaces to map Java objects to database tables.
* Requires an implementation like Hibernate.

**Hibernate**

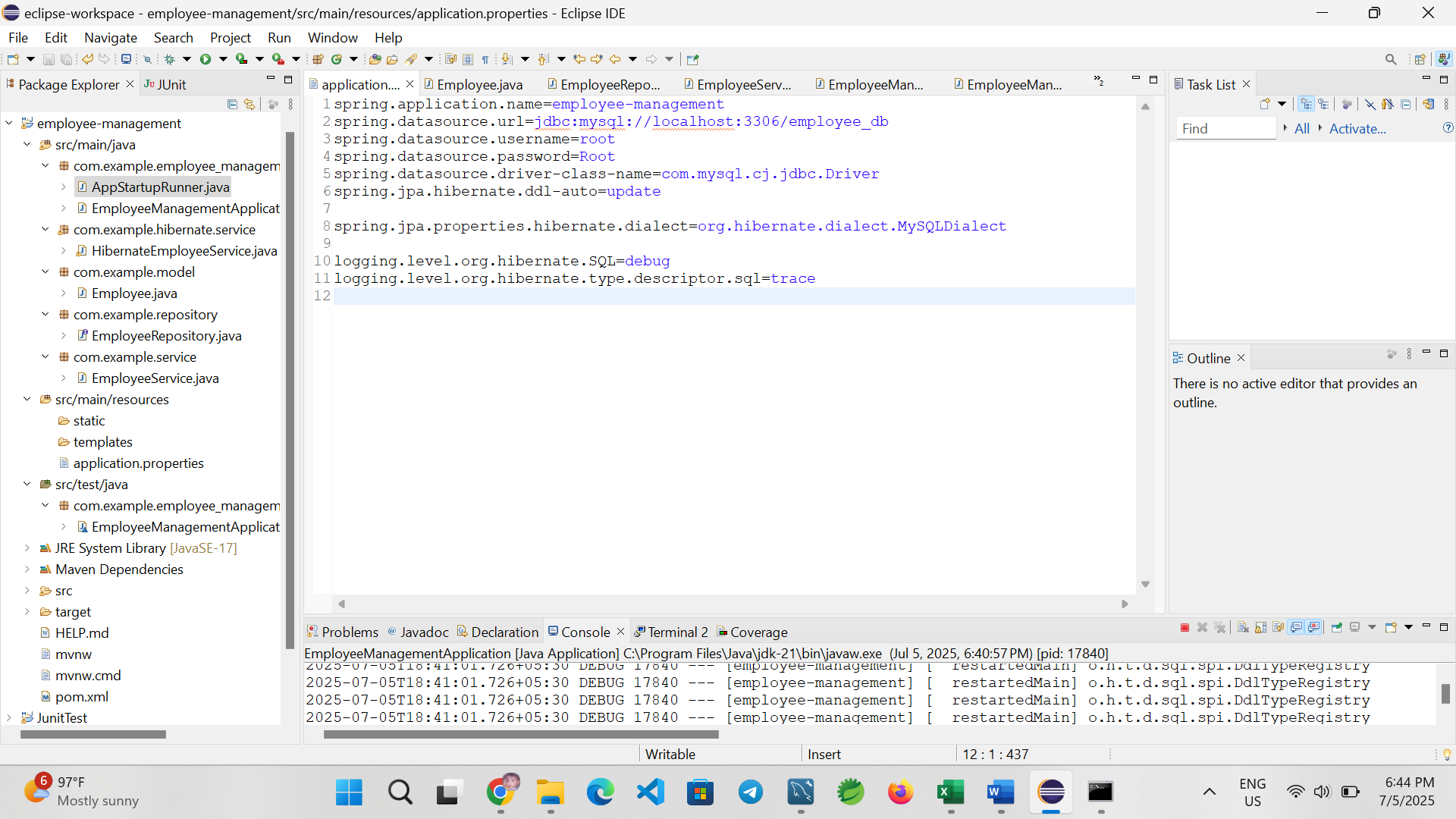
* An ORM tool that implements JPA.
* Offers advanced features like caching and lazy loading.

**Spring Data JPA**

* Builds on top of JPA.
* Reduces boilerplate by generating repository code automatically.
* Integrates seamlessly with Spring Boot.

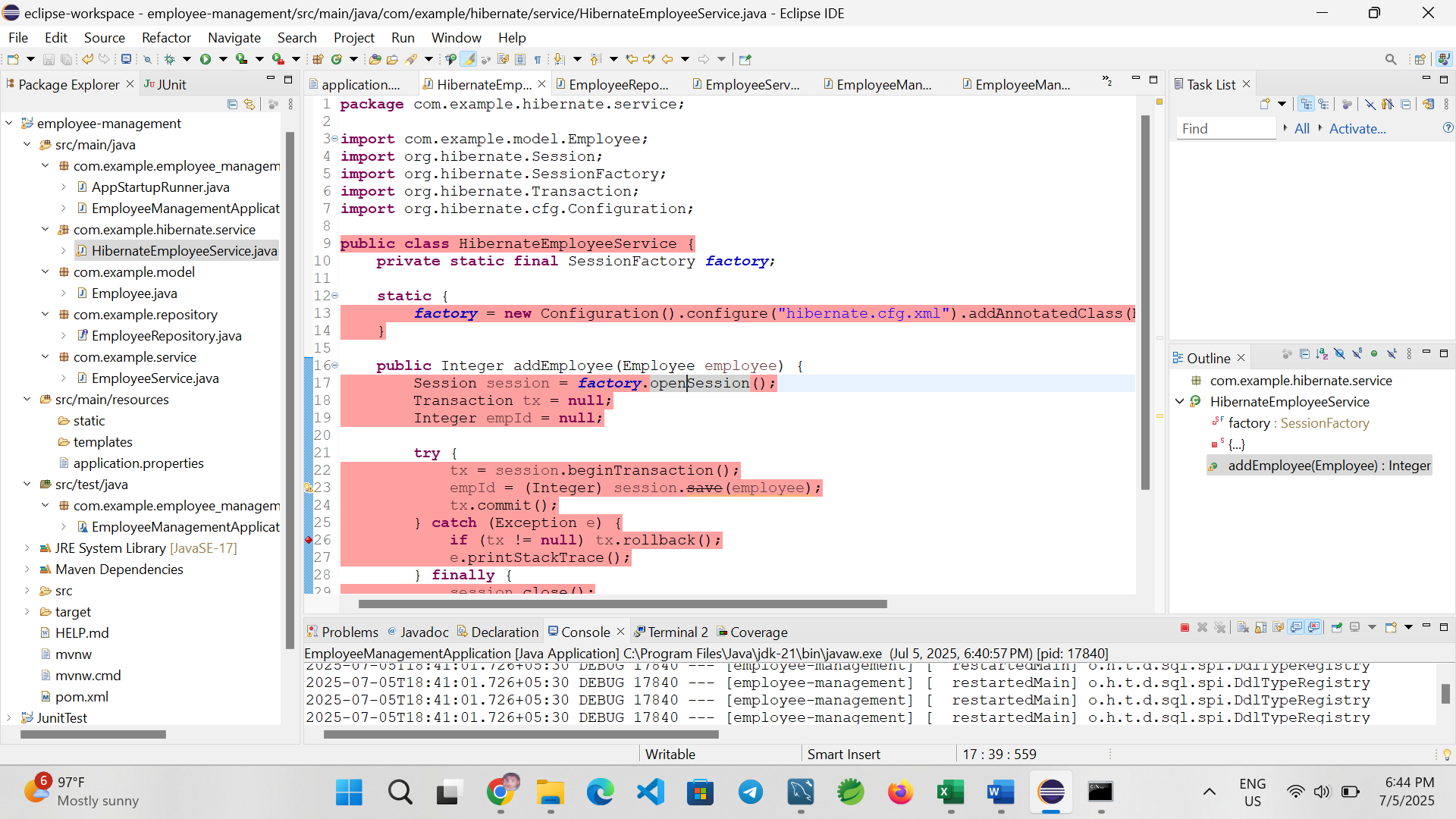
**2. Project Setup**

**application.properties**

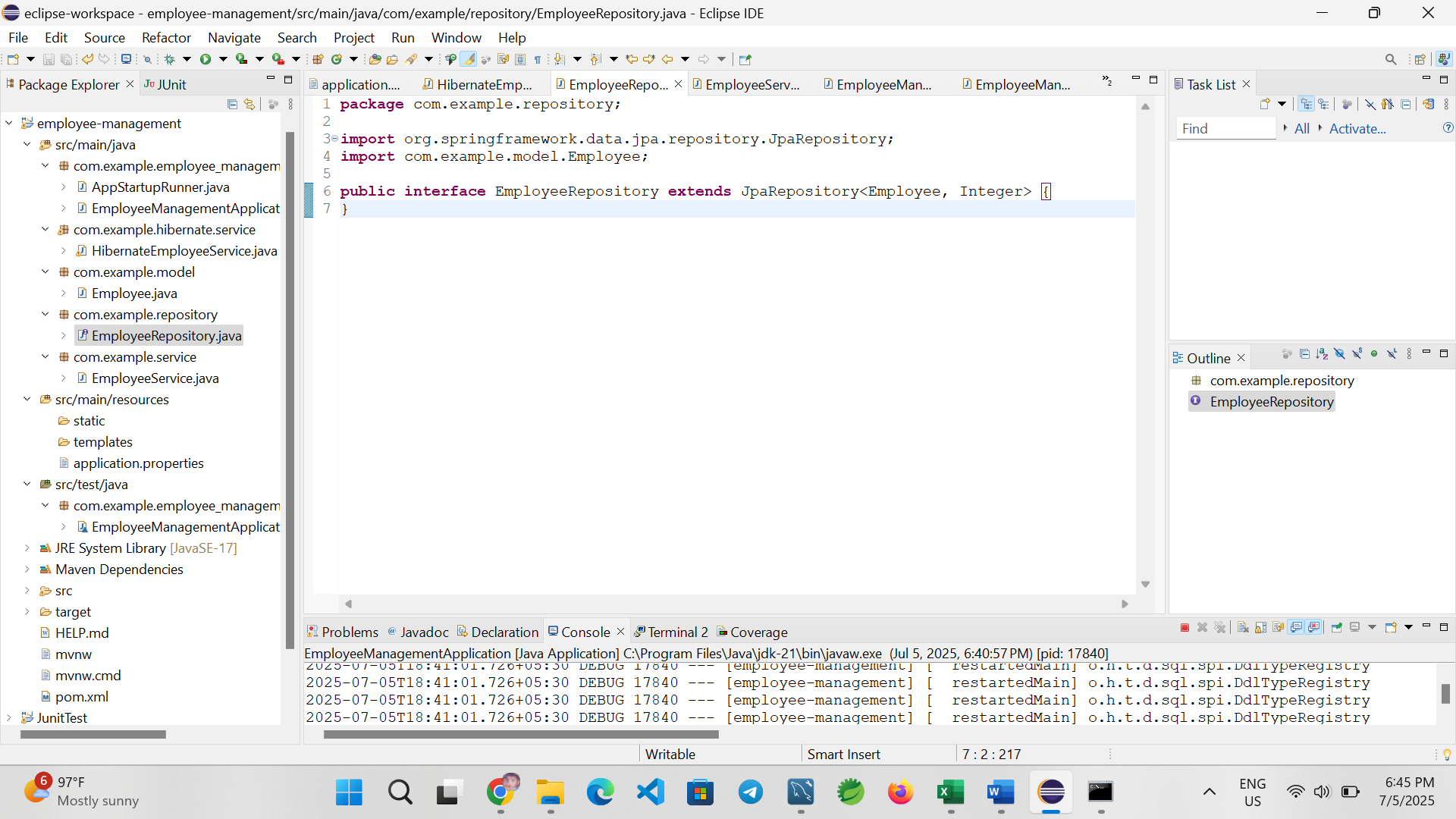


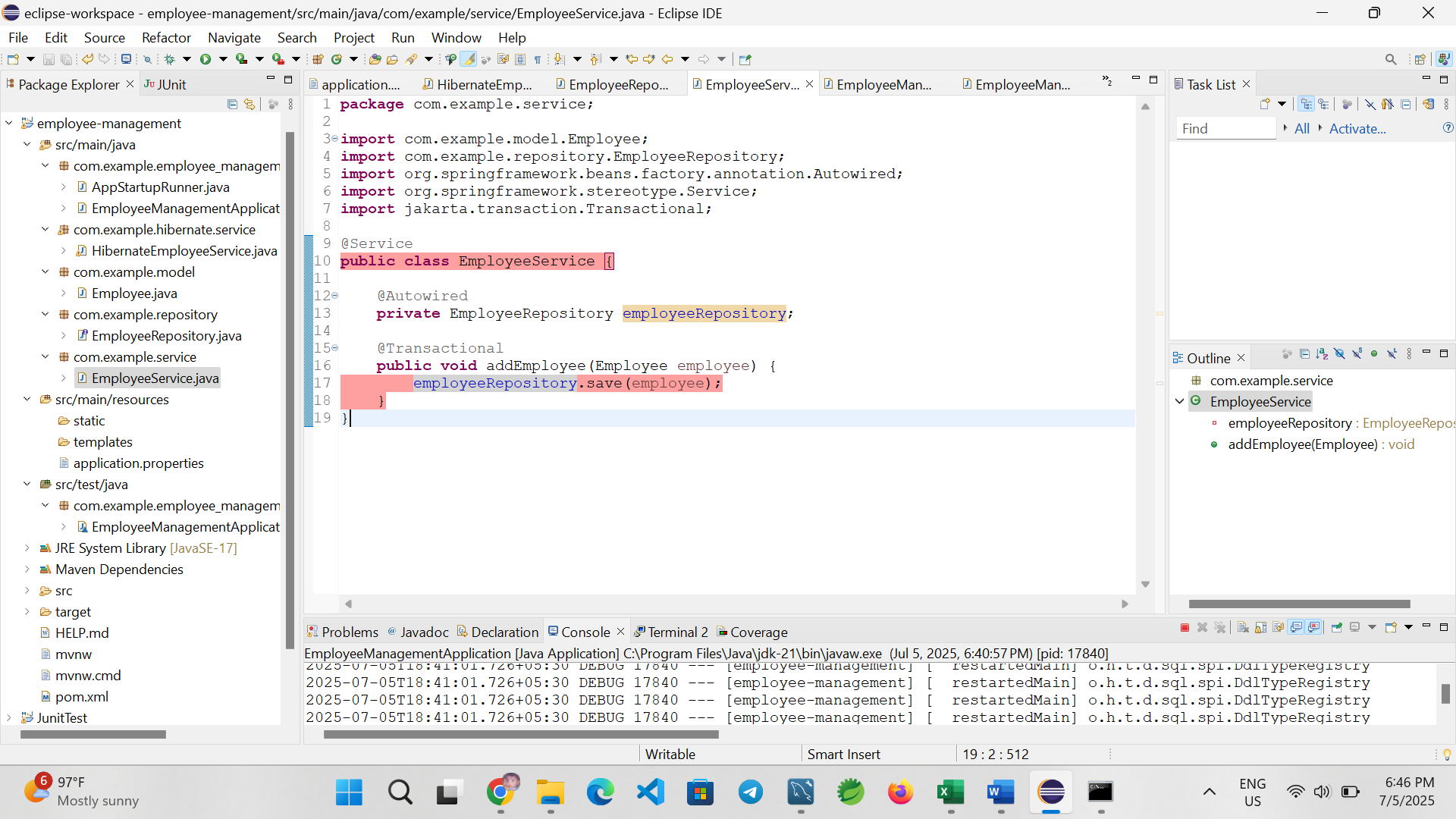
**3. Code Comparison**

**Hibernate Style (Manual)**



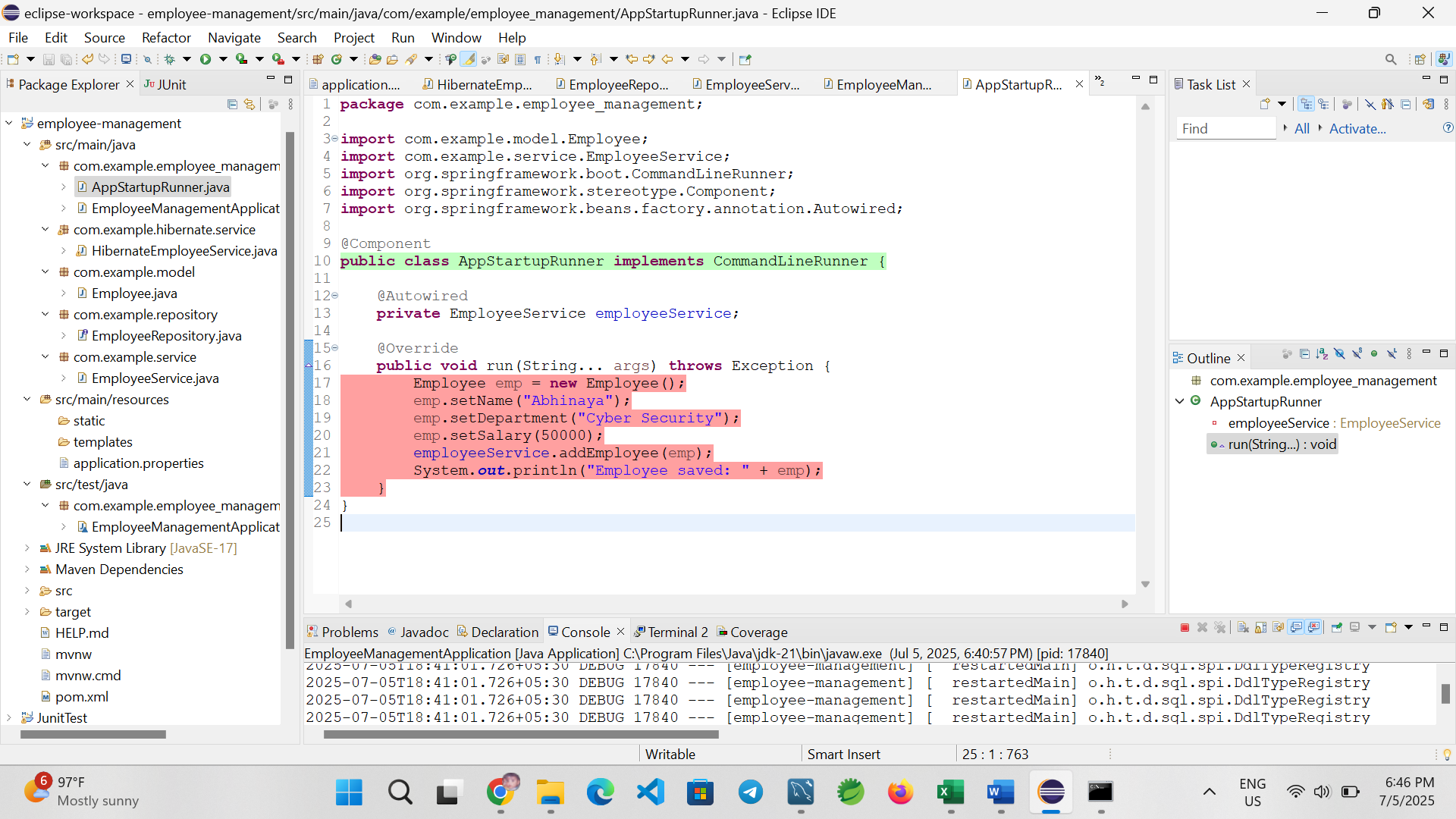
*EmployeeRepository.java*

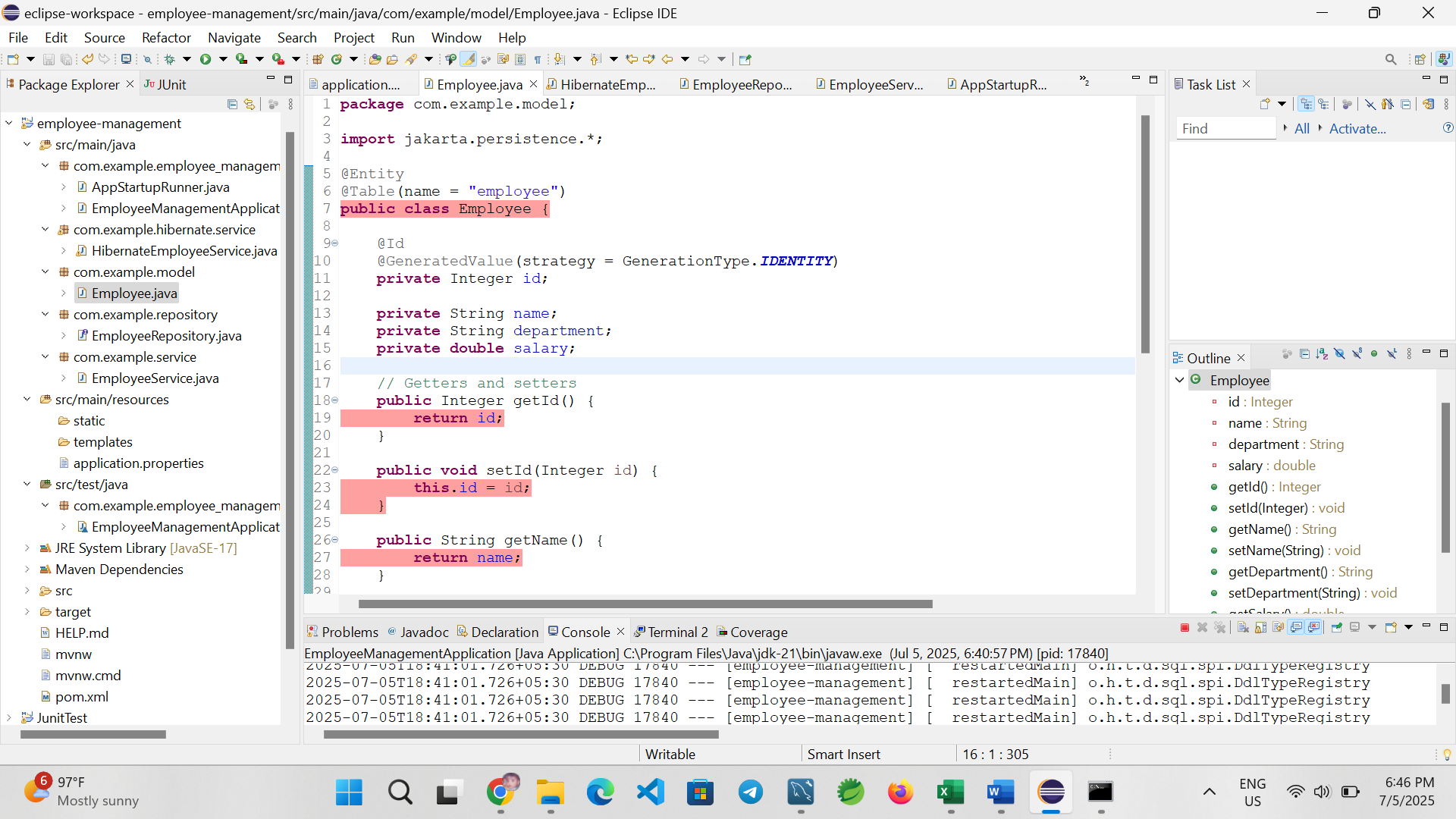
*EmployeeService.java*



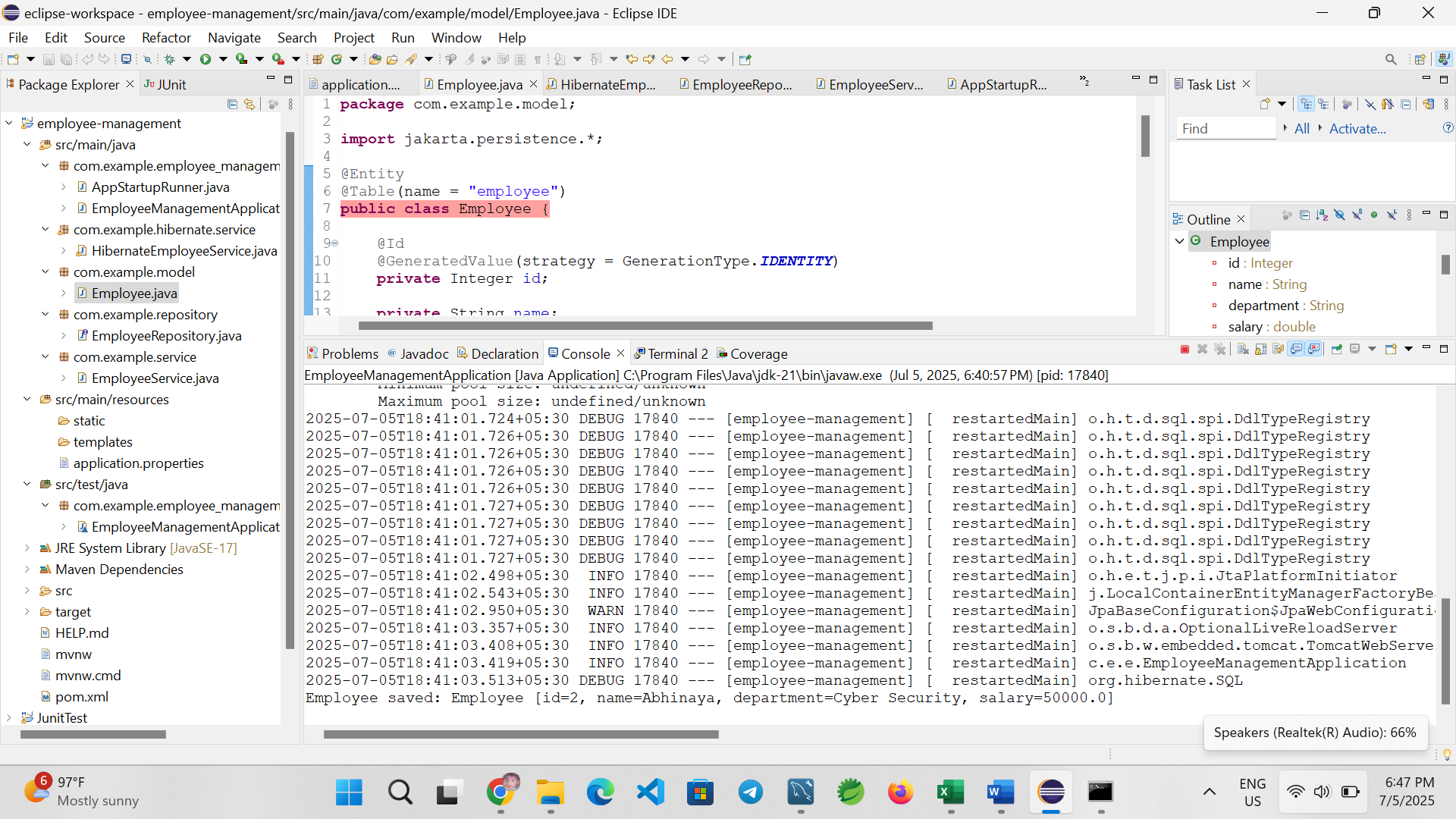
**4. Running the Application**

**AppStartupRunner.java** (Bootstrap data on app startup)

**Employee Entity**



**Output**

This validates successful data insertion and shows how Spring Data JPA abstracts complexity while maintaining power and flexibility.

**5. Conclusion**

Spring Data JPA significantly simplifies persistence logic by reducing boilerplate code. With minimal setup, it provides robust transaction management, repository interfaces, and integration with relational databases. This example demonstrates how to implement a real-world use case with ease compared to traditional Hibernate.