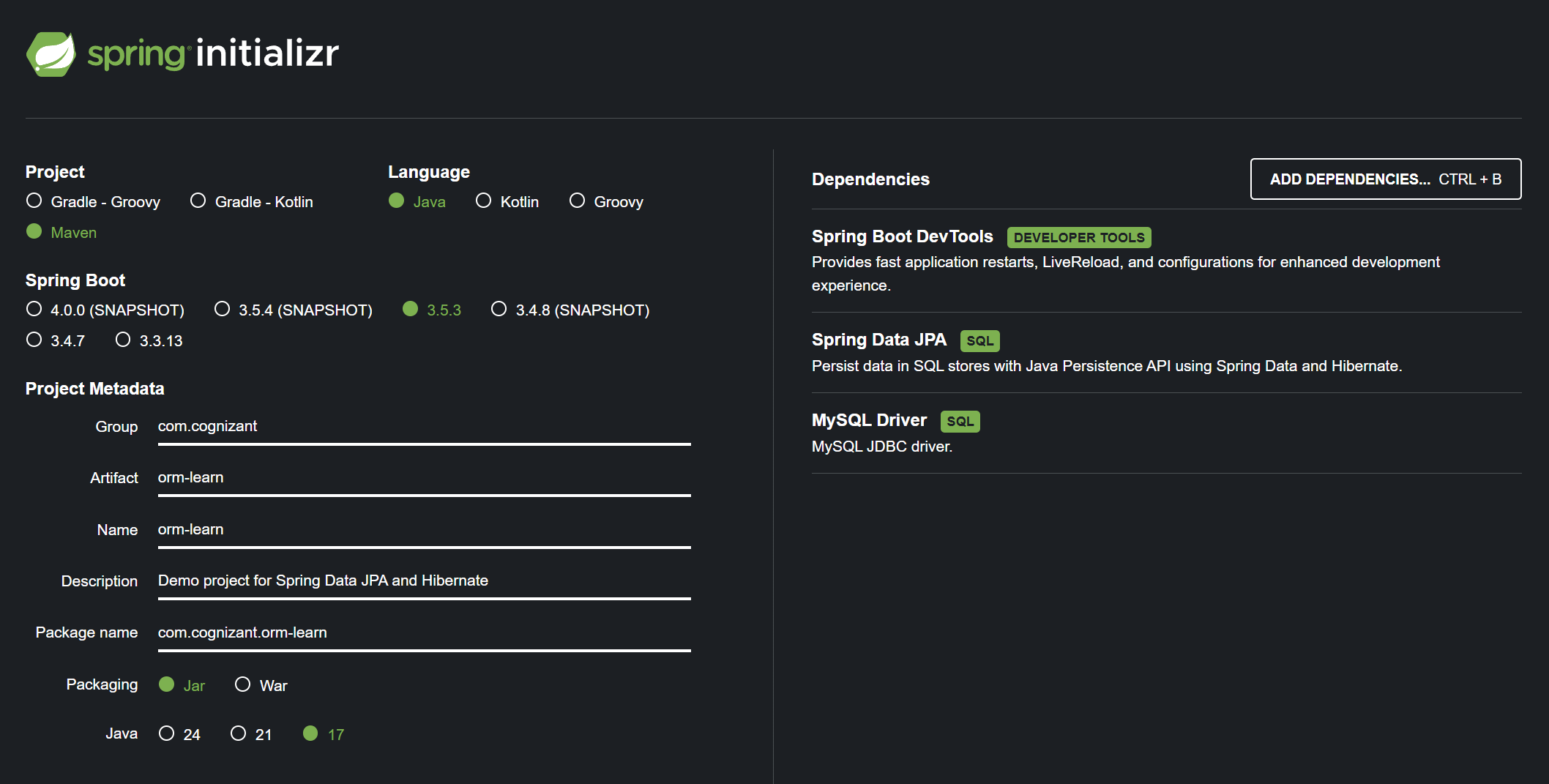
**Cognizant Digital Nurture 4.0**

***WEEK-2 Module 5 – Spring Core\_Maven***

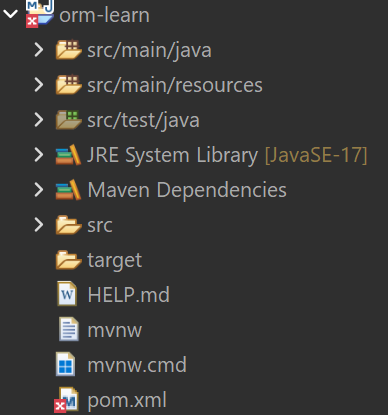
**Hands on 1**

**Spring Data JPA - Quick Example (Mandatory)**

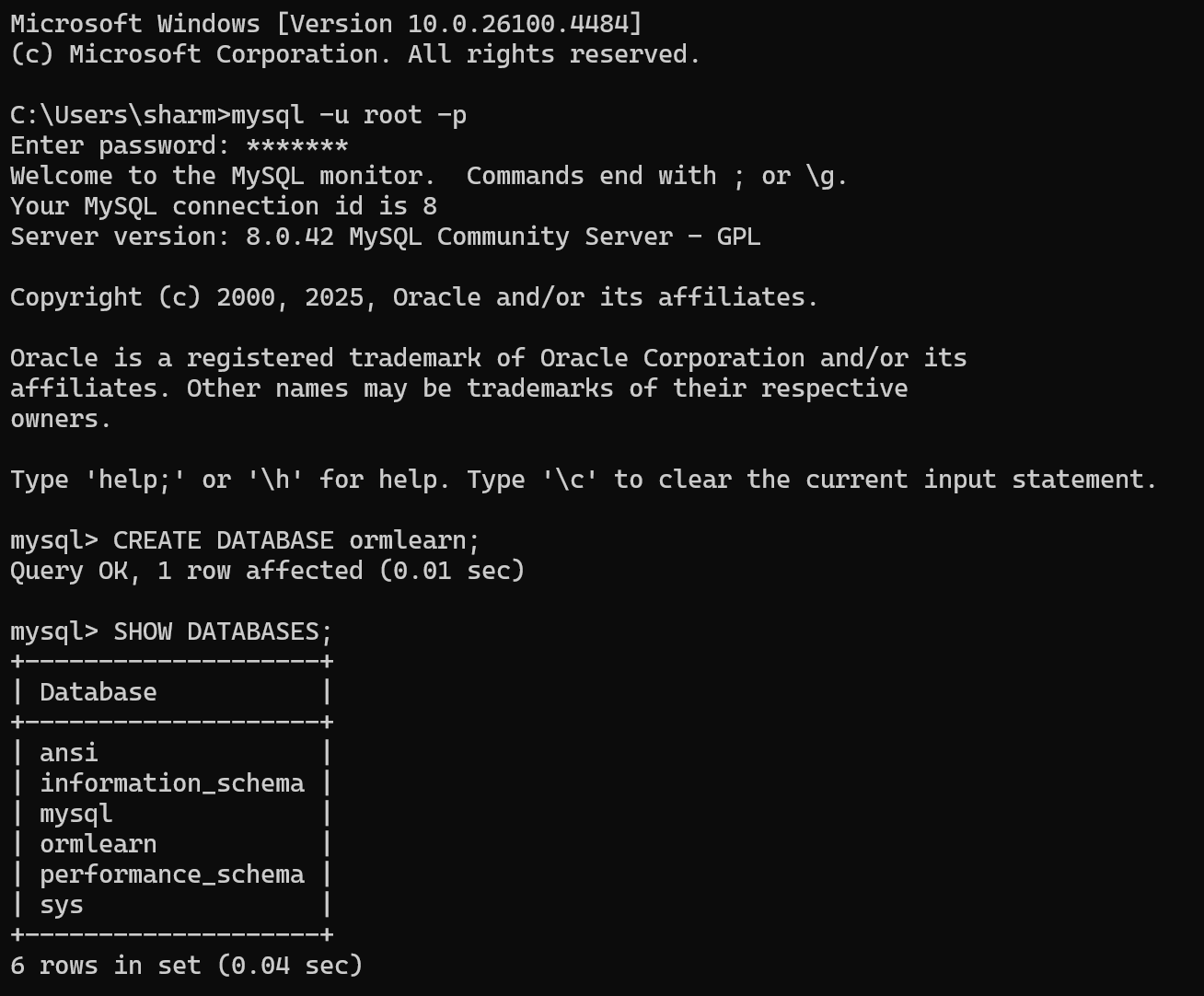
1. Go to https://start.spring.io/
2. Configurations:
   1. Change Group as “com.cognizant”
   2. Change Artifact Id as “orm-learn”
   3. In Options > Description enter "Demo project for Spring Data JPA and Hibernate"
   4. Click on menu and select "Spring Boot DevTools", "Spring Data JPA" and "MySQL Driver"
   5. Click Generate and download the project as zip



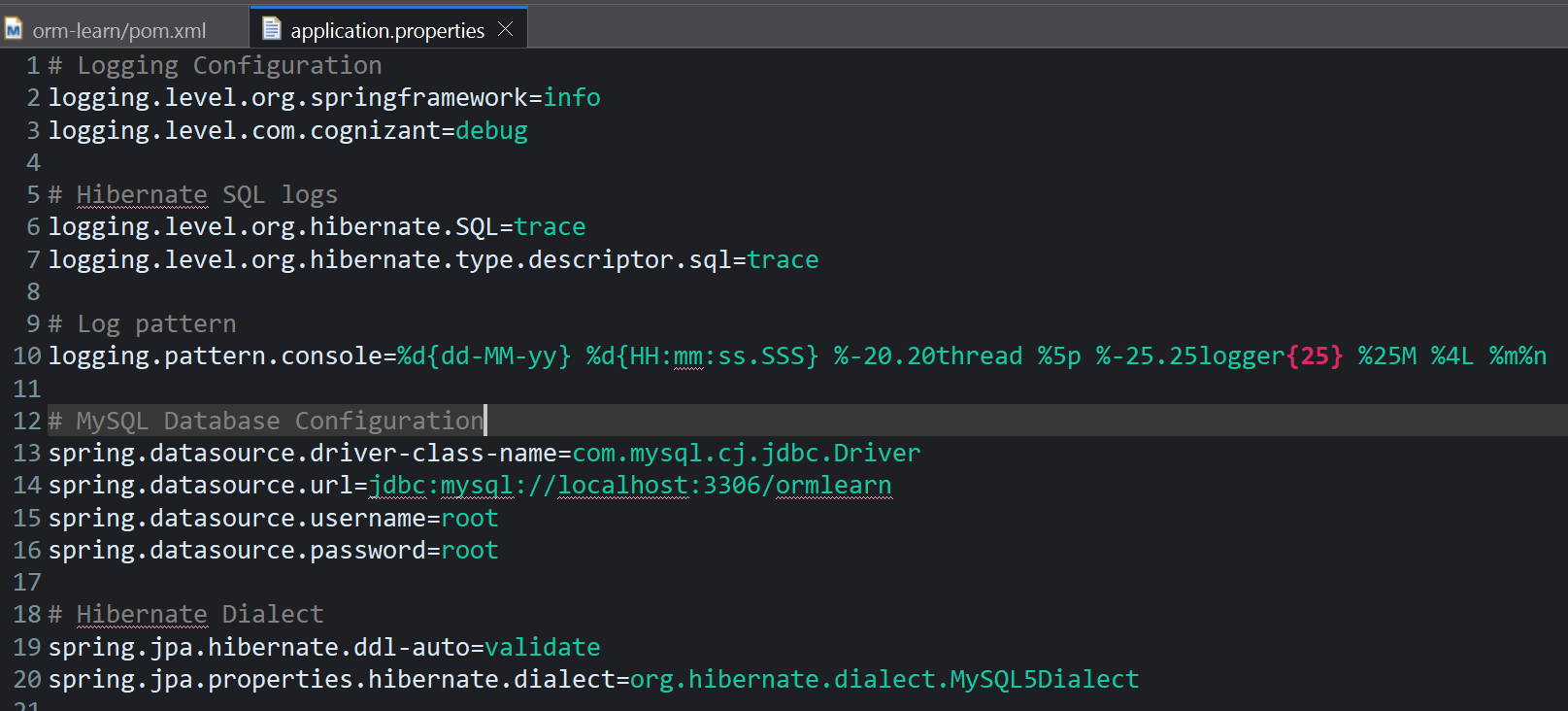
1. Extract the zip in root folder to Eclipse Workspace
2. Import the project in Eclipse "File > Import > Maven > Existing Maven Projects > Click Browse and select extracted folder > Finish"



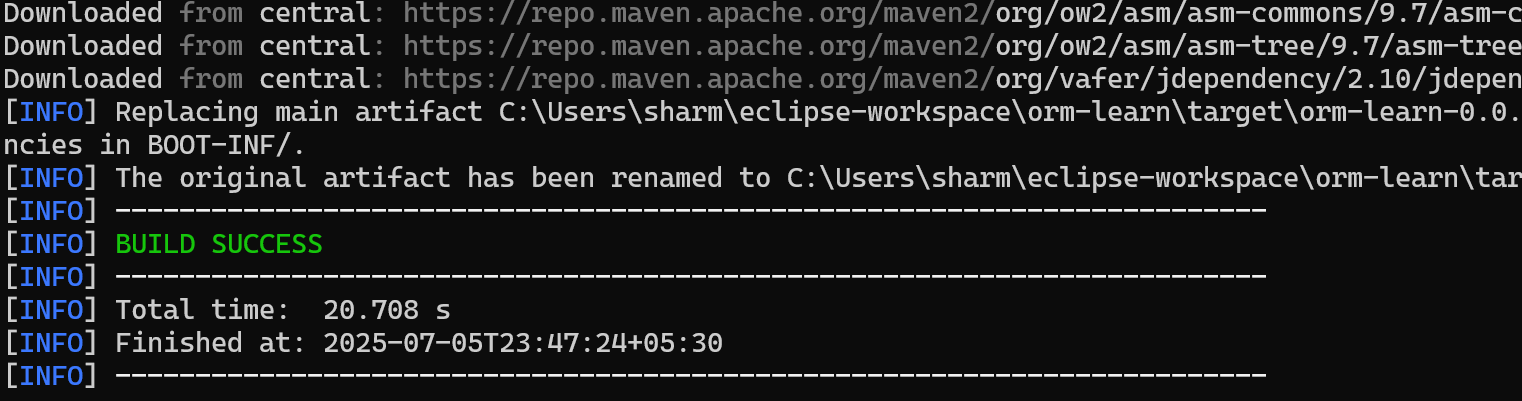
1. Create a new schema "ormlearn" in MySQL database. Execute the following commands to open MySQL client and create schema.



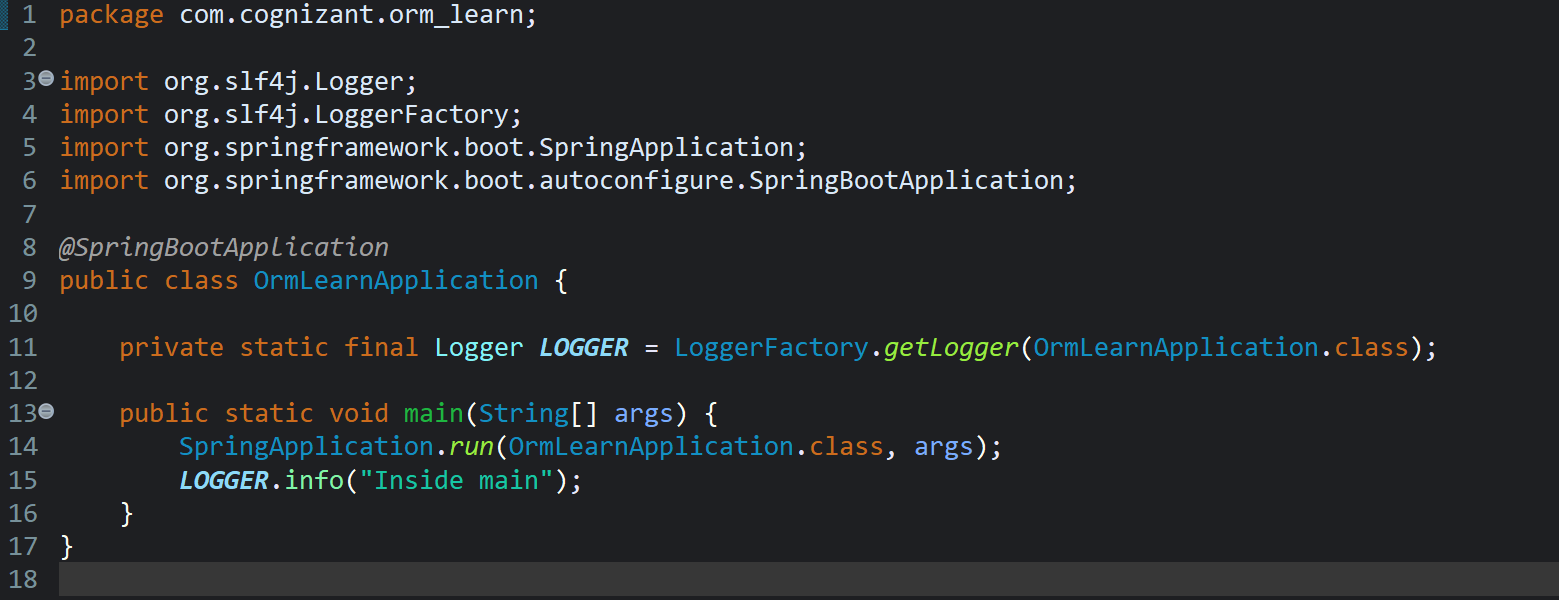
1. In orm-learn Eclipse project, open src/main/resources/application.properties and include the below database and log configuration.



1. Build the project using ‘mvn clean package -Dhttp.proxyHost=proxy.cognizant.com -Dhttp.proxyPort=6050 -Dhttps.proxyHost=proxy.cognizant.com -Dhttps.proxyPort=6050 -Dhttp.proxyUser=123456’ command in command line

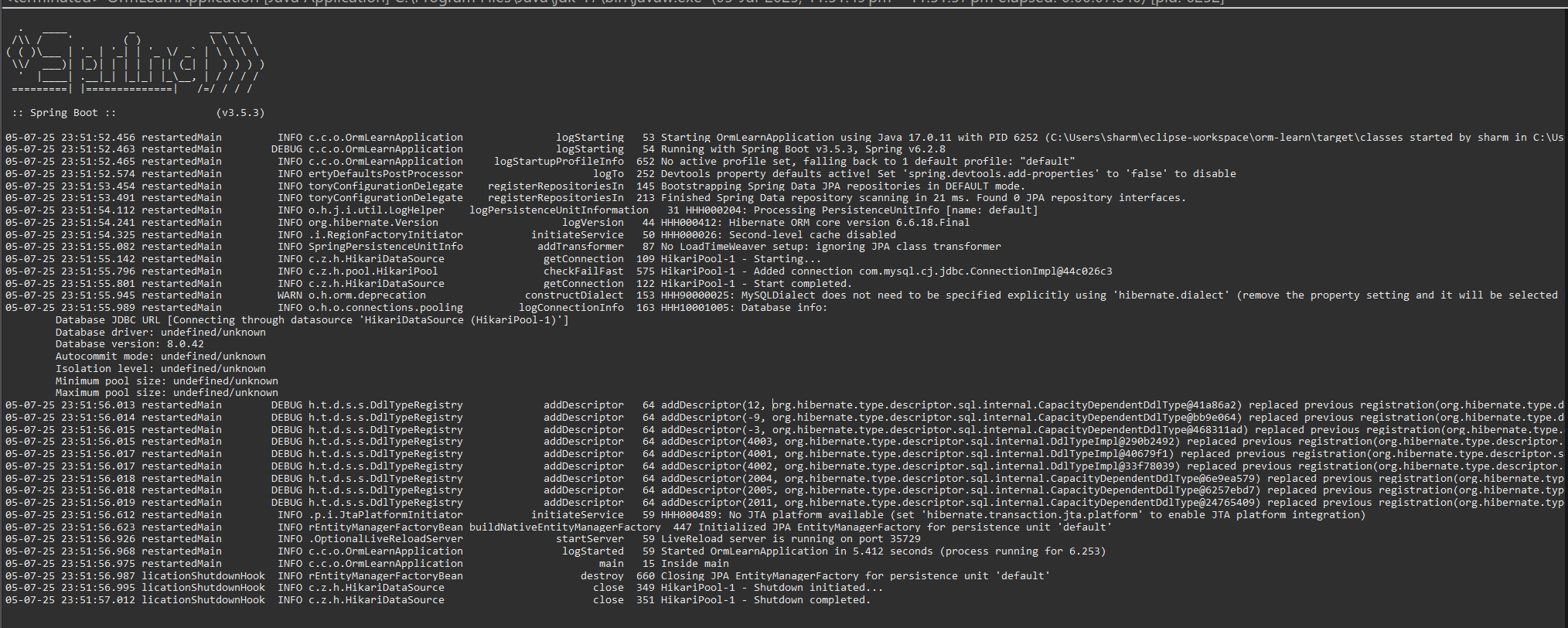


1. Include logs for verifying if main() method is called.

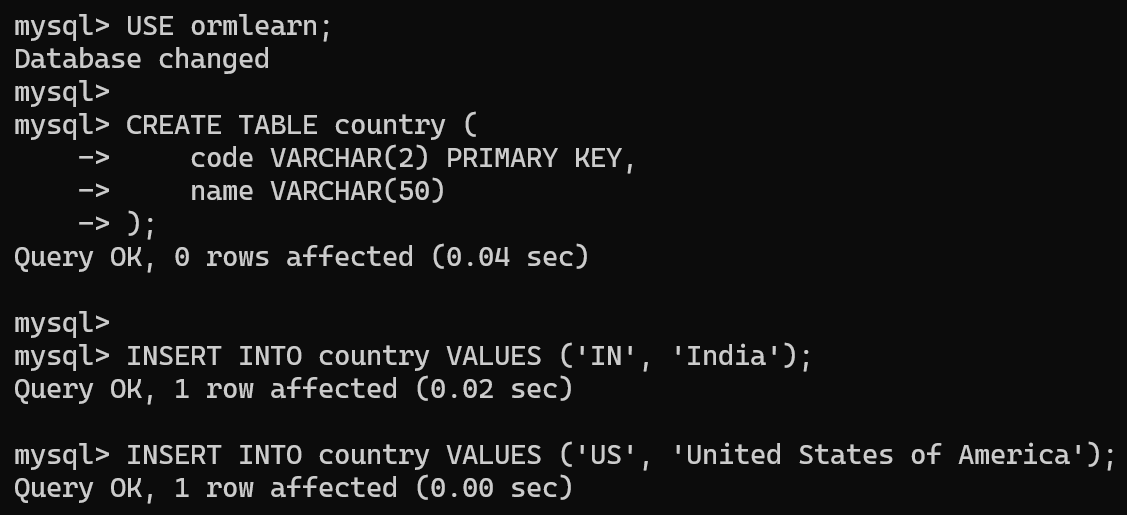


1. Execute the OrmLearnApplication and check in log if main method is called.

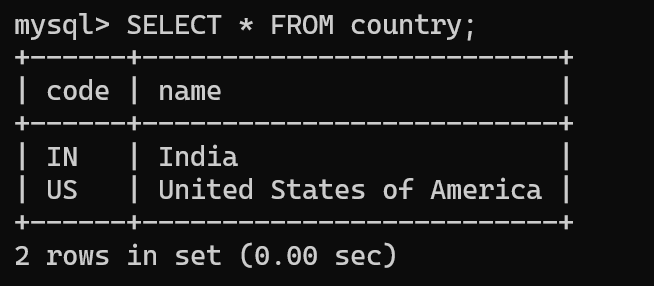
Output:



1. Create a new table country with columns for code and name. For sample, let us insert one country with values 'IN' and 'India' in this table.



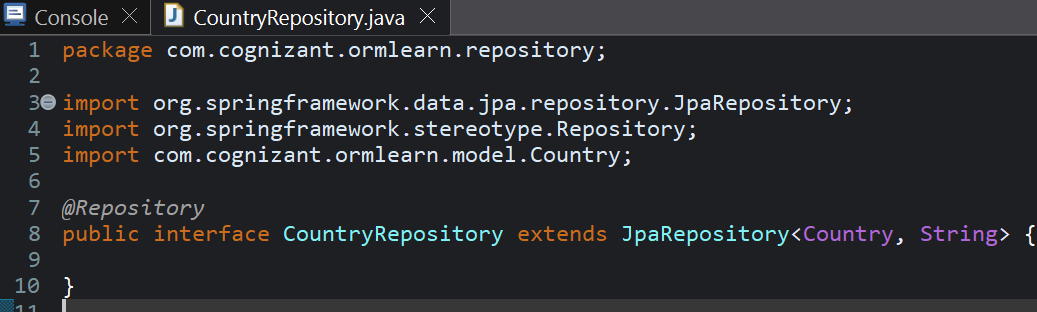
1. Insert couple of records into the table



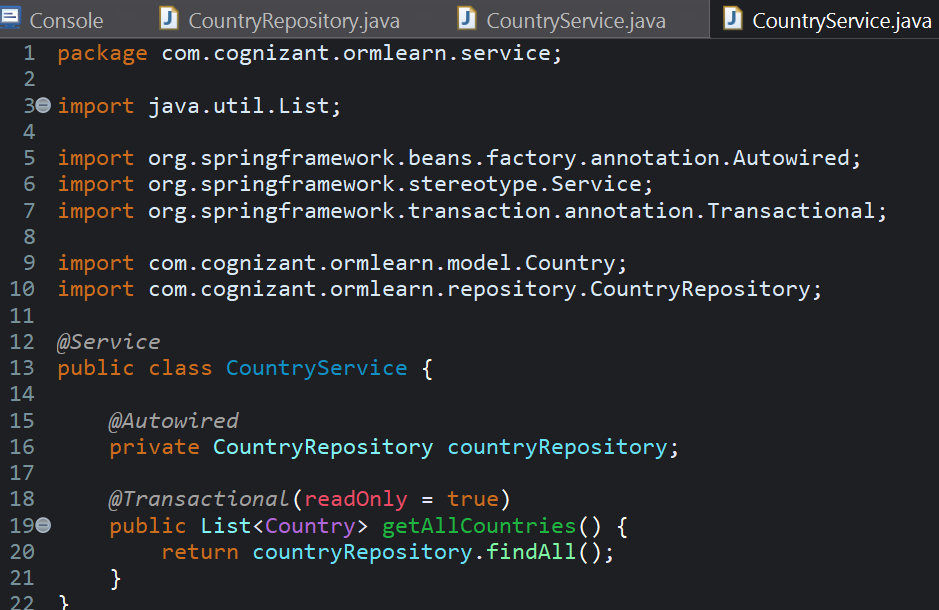
1. In Eclipse, open the orm-learn project, create com.cognizant.orm\_learn.model package, add Country.java with getters, setters, toString(), annotate with @Entity, @Table, and use @Column on getters.



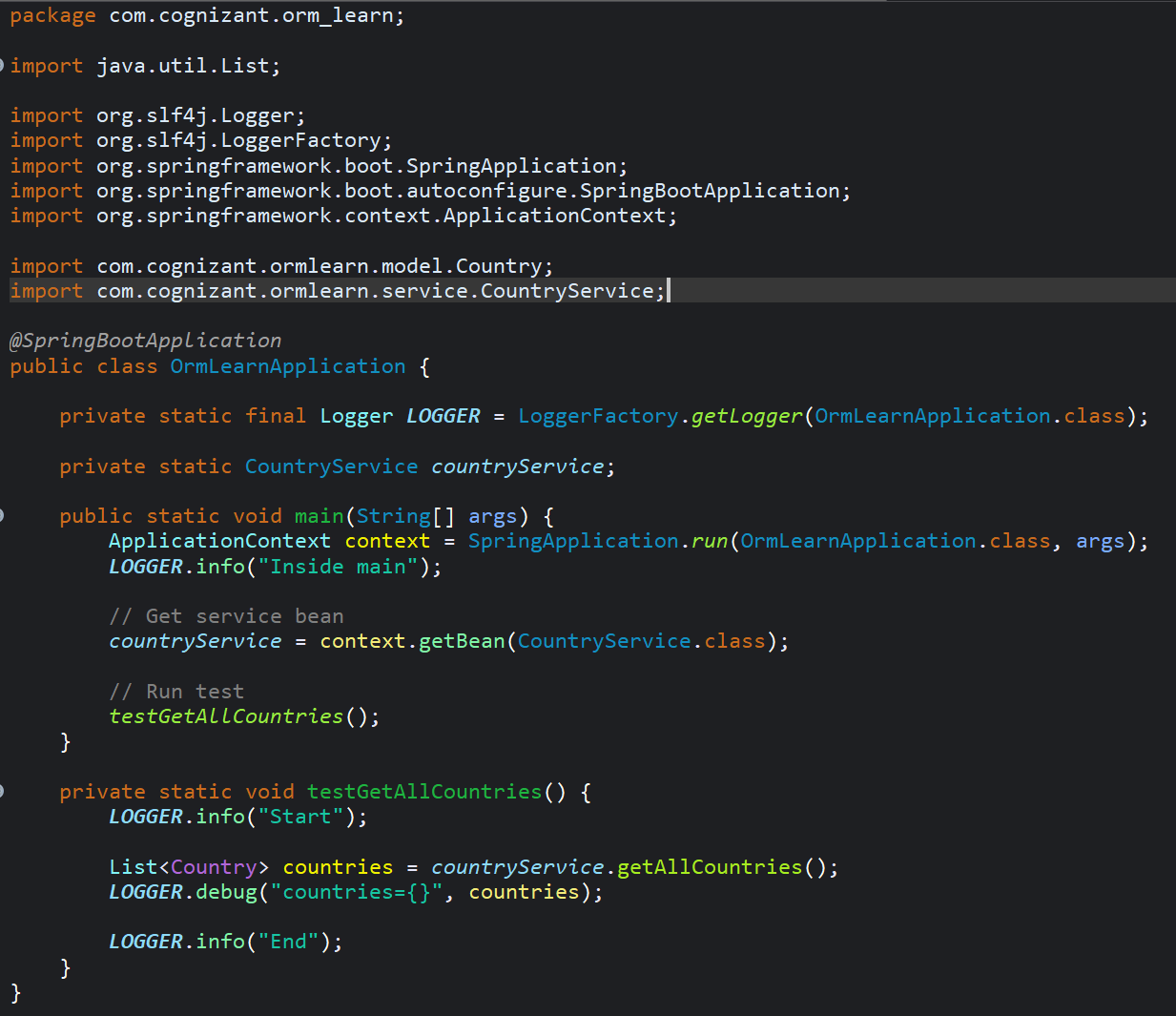
1. Create a CountryRepository interface in the com.cognizant.orm-learn.repository package, extending JpaRepository<Country, String> and annotate it with @Repository.



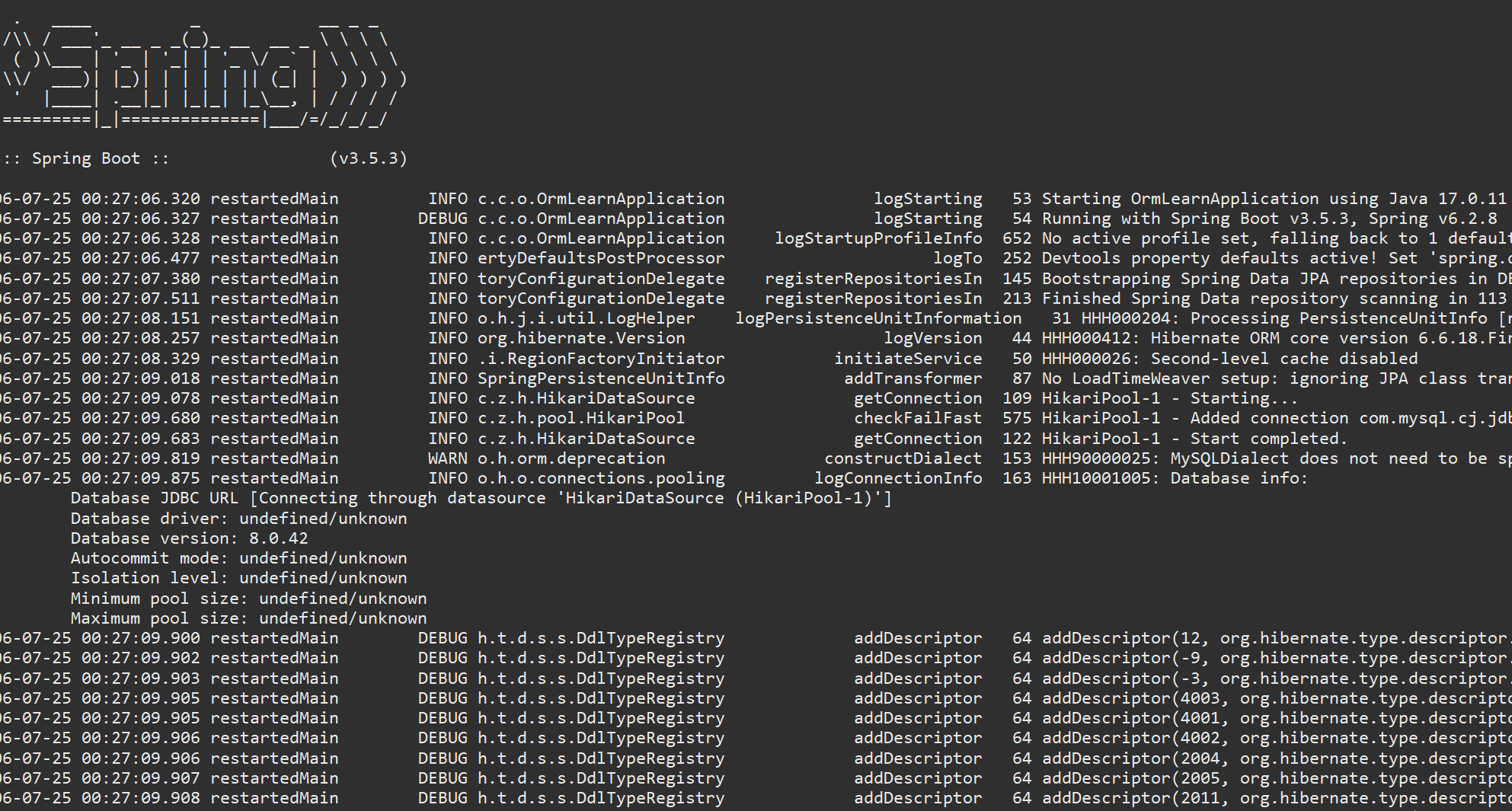
1. Create a CountryService class in the com.cognizant.orm-learn.service package, annotate it with @Service, autowire CountryRepository, and add a @Transactional getAllCountries() method that returns countryRepository.findAll().



1. Changes made in OrmLearnApplication class



Output:

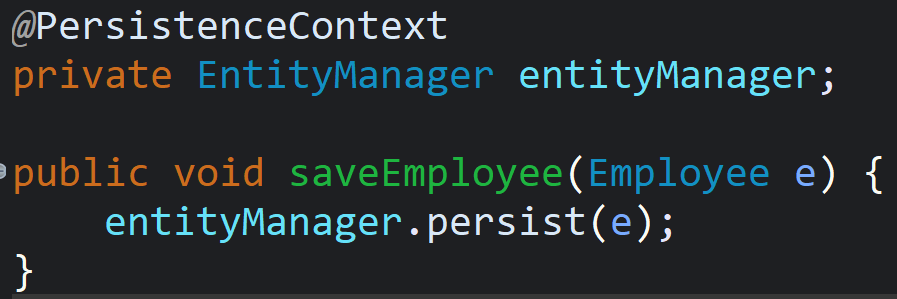


**Hands on 4**

**Difference between JPA, Hibernate and Spring Data JPA (Mandatory)**

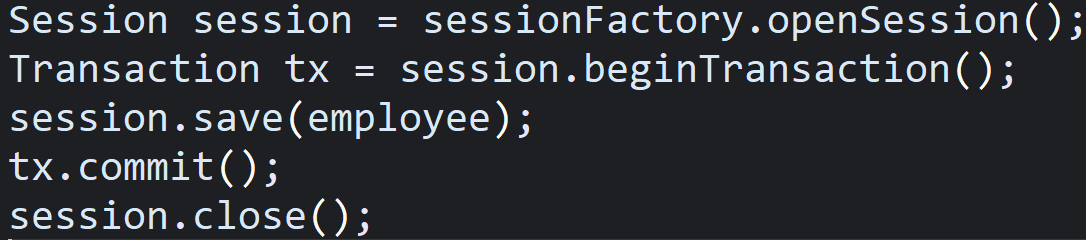
**JPA:**

**JPA** (Java Persistence API) is a specification defined by Java to manage relational data using Java objects in an object-relational mapping (ORM) approach. It outlines a set of interfaces and annotations that developers can use to interact with databases in a standardized way, without tying their application to a specific implementation. JPA does not provide any actual working code or functionality on its own; it relies on implementations like Hibernate or EclipseLink to perform the real database operations. Its purpose is to allow developers to write database-independent persistence code that can work with any JPA-compliant provider. Ex. Hibernate, EclipseLink, OpenJPA.



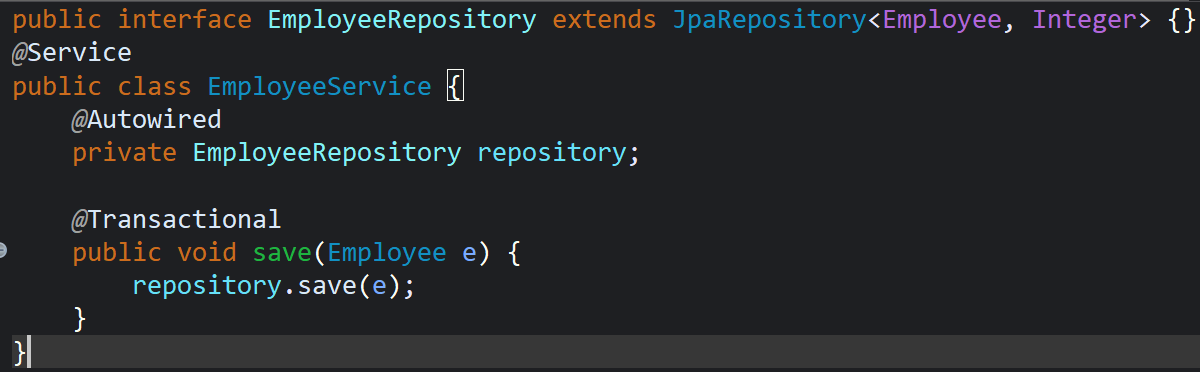
**Hibernate:**

**Hibernate** is a popular and powerful ORM framework that provides a concrete implementation of the JPA specification. In addition to fulfilling all JPA requirements, Hibernate also offers several advanced features such as caching, lazy loading, and custom data types, which are not part of the standard JPA. It allows developers to map Java classes to database tables and perform CRUD operations using HQL (Hibernate Query Language) or native SQL. Unlike Spring Data JPA, Hibernate requires developers to manage sessions and transactions manually (unless integrated with Spring), which can result in more boilerplate code and configuration.



**Spring Data JPA:**

Spring Data JPA is an abstraction layer built on top of JPA that integrates tightly with the Spring Framework. It simplifies the process of interacting with a database by reducing the amount of boilerplate code required for repository and CRUD operations. Developers can define interfaces that extend JpaRepository, and Spring Data JPA automatically provides the implementation at runtime. It handles transactions, query execution, and even custom queries using method names or annotations. Spring Data JPA does not implement JPA itself but relies on a JPA provider like Hibernate underneath to perform the actual data persistence operations.



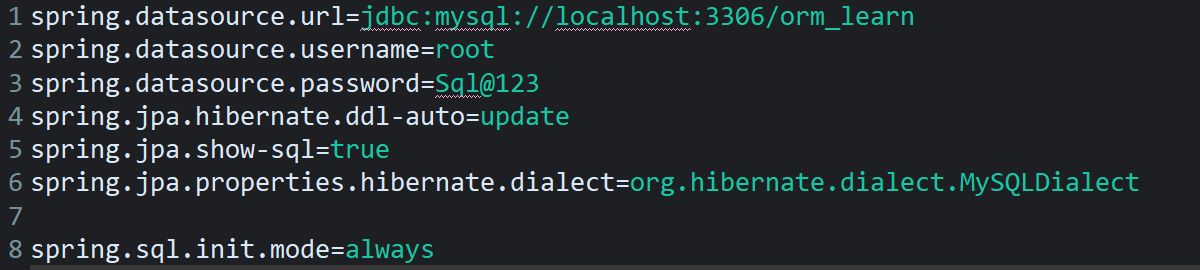
**Additional important hands-on**

**Hands on 5**

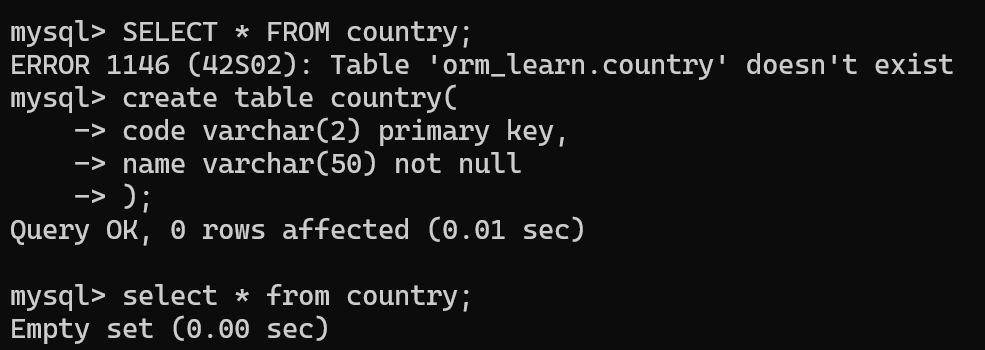
**Implement services for managing Country**   
  
**1. I created a new Maven project in Eclipse**  
I used the Spring Initializr to generate a Spring Boot project with the necessary dependencies:

* Spring Web
* Spring Data JPA
* MySQL Driver
* Spring Boot DevTools

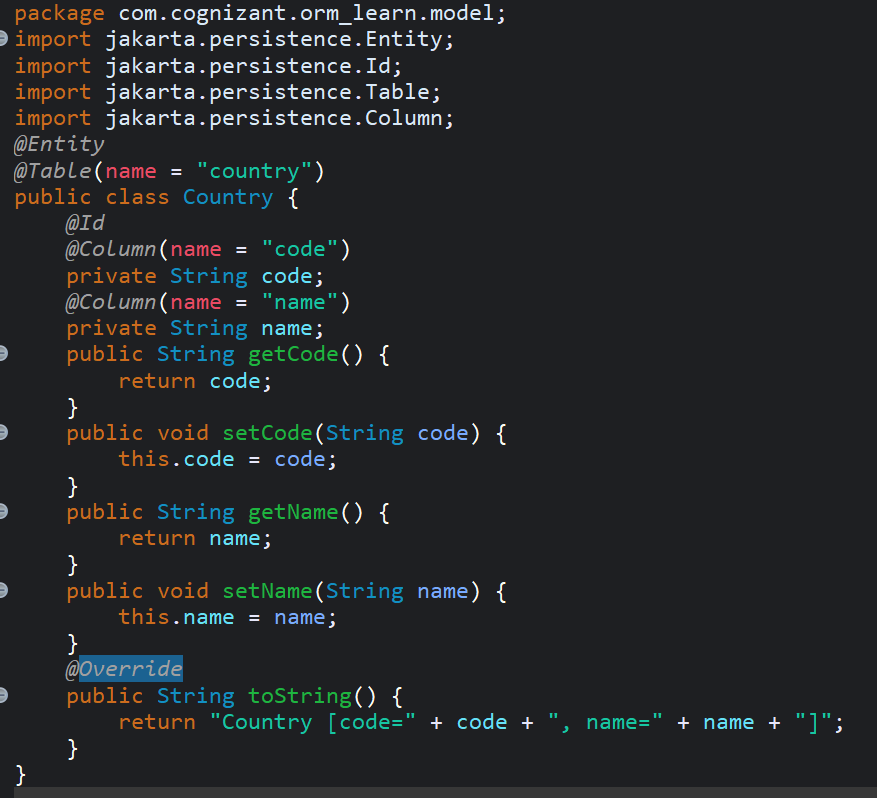
**2. I configured the application.properties file**  
I added my MySQL database details to connect Spring Boot with my local database:



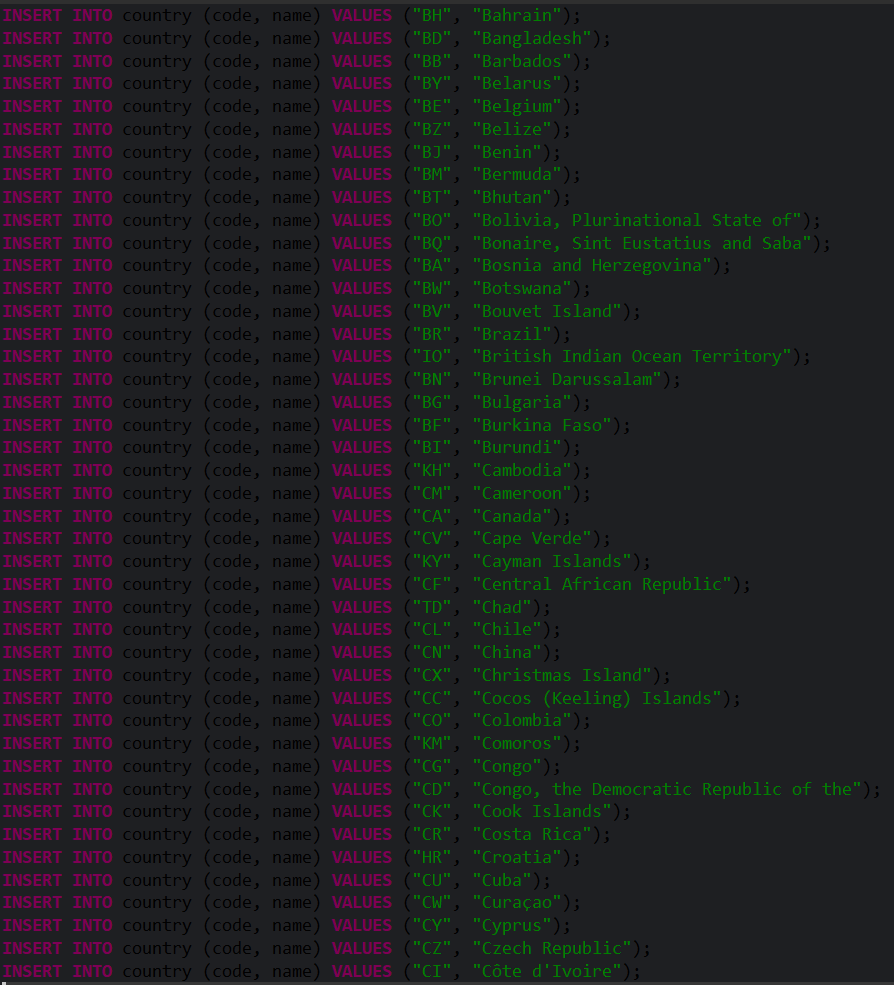
**3. I created the database and the table manually**  
I opened MySQL and created the database orm\_learn, then executed the following SQL:



**4. I created the Country entity**  
Inside the com.cognizant.orm\_learn.model package, I created the Country class and annotated it with @Entity:



**5. I added test data in data.sql**  
To pre-populate the table, I added a file named data.sql in src/main/resources with insert queries like:



Output:

