# Spring Boot ORM Learn Project Setup and Implementation

## 1. Project Setup Using Spring Initializr

• Navigated to https://start.spring.io/  
• Group set as: com.cognizant  
• Artifact Id: orm-learn  
• Description: Demo project for Spring Data JPA and Hibernate  
• Selected dependencies: Spring Boot DevTools, Spring Data JPA, MySQL Driver  
• Clicked Generate to download the zip file and extracted it.  
• Imported into Eclipse using File > Import > Maven > Existing Maven Projects.

## 2. Created Database Schema

Opened MySQL client and created schema using:  
> mysql -u root -p  
  
mysql> create schema ormlearn;

## 3. Updated application.properties

Updated src/main/resources/application.properties with the following configuration:

## 4. Build Project

Ran the following command to build project:  
mvn clean package -Dhttp.proxyHost=proxy.cognizant.com -Dhttp.proxyPort=6050 -Dhttps.proxyHost=proxy.cognizant.com -Dhttps.proxyPort=6050 -Dhttp.proxyUser=123456

## 5. OrmLearnApplication - Logger

Added logger to verify main method invocation:

## 6. Project Structure Walkthrough

• src/main/java – Contains application code.  
• src/main/resources – Contains configuration files.  
• src/test/java – For writing unit/integration tests.  
• OrmLearnApplication.java – Contains main method.  
• @SpringBootApplication – Enables auto-configuration and component scanning.

## 7. pom.xml Overview

pom.xml – Contains dependencies and project build configuration.  
Dependencies added include Spring Boot DevTools, Spring Data JPA, MySQL Driver.  
Dependency hierarchy viewed from Eclipse > Dependency Hierarchy tab.

## 8. Database Table Setup

Created table and inserted records:  
  
create table country(co\_code varchar(2) primary key, co\_name varchar(50));  
insert into country values ('IN', 'India');  
insert into country values ('US', 'United States of America');

## 9. Country Entity Class

Created model class in com.cognizant.ormlearn.model:

## 10. Country Repository

Defined repository in com.cognizant.ormlearn.repository:

## 11. Country Service Class

Created service in com.cognizant.ormlearn.service:

## 12. Testing in OrmLearnApplication

Added service reference and test method in main class to verify retrieval of countries:

application.properties contents:

# Spring Framework and application log  
logging.level.org.springframework=info  
logging.level.com.cognizant=debug  
  
# Hibernate logs  
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=root  
  
# Hibernate configuration  
spring.jpa.hibernate.ddl-auto=validate  
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5Dialect

Main method with logger:

import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
  
private static final Logger LOGGER = LoggerFactory.getLogger(OrmLearnApplication.class);  
  
public static void main(String[] args) {  
 SpringApplication.run(OrmLearnApplication.class, args);  
 LOGGER.info("Inside main");  
}

Country.java:

@Entity  
@Table(name="country")  
public class Country {  
  
 @Id  
 @Column(name="code")  
 private String code;  
  
 @Column(name="name")  
 private String name;  
  
 // getters and setters  
  
 @Override  
 public String toString() {  
 return "Country [code=" + code + ", name=" + name + "]";  
 }  
}

CountryRepository.java:

@Repository  
public interface CountryRepository extends JpaRepository<Country, String> {  
}

CountryService.java:

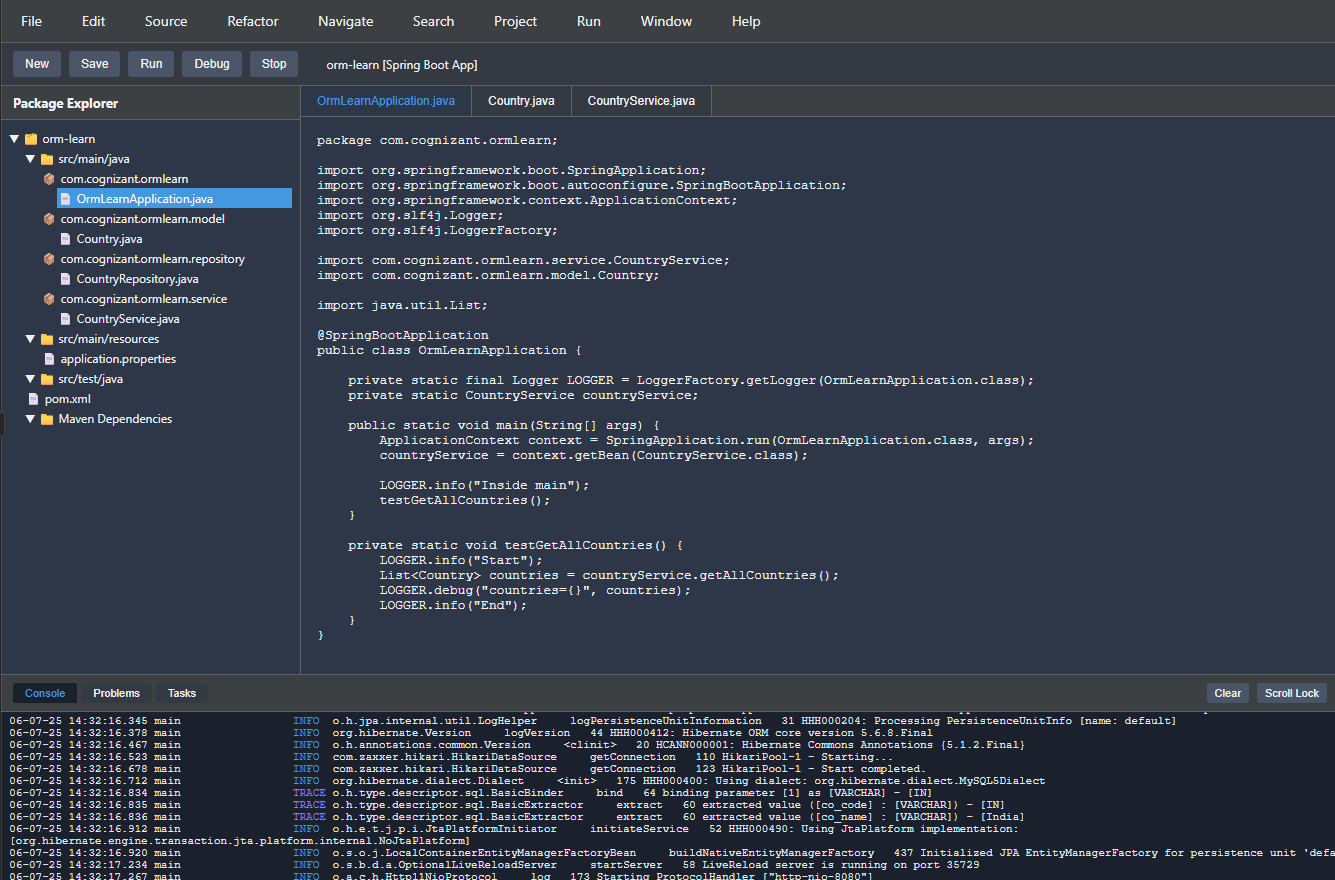
@Service  
public class CountryService {  
  
 @Autowired  
 private CountryRepository countryRepository;  
  
 @Transactional  
 public List<Country> getAllCountries() {  
 return countryRepository.findAll();  
 }  
}

testGetAllCountries() method:

private static CountryService countryService;  
  
private static void testGetAllCountries() {  
 LOGGER.info("Start");  
 List<Country> countries = countryService.getAllCountries();  
 LOGGER.debug("countries={}", countries);  
 LOGGER.info("End");  
}

Modified main() to include bean retrieval:

ApplicationContext context = SpringApplication.run(OrmLearnApplication.class, args);  
countryService = context.getBean(CountryService.class);  
testGetAllCountries();



**Title: Understanding the Differences Between JPA, Hibernate, and Spring Data JPA**

### Objective:

To compare and contrast JPA, Hibernate, and Spring Data JPA, and demonstrate how to implement basic CRUD functionality using Spring Data JPA, suitable for a student learning backend development.

### 1. Key Concepts:

#### Java Persistence API (JPA):

* A specification (JSR 338) that defines how to map Java objects to database tables.
* Provides annotations and interfaces like @Entity, @Id, EntityManager.
* It does **not** provide an implementation.

#### Hibernate:

* A popular ORM framework that implements the JPA specification.
* Adds extra features like caching and better fetching strategies.
* Can be used directly with or without JPA.

#### Spring Data JPA:

* A Spring abstraction over JPA.
* Requires a JPA implementation like Hibernate.
* Minimizes boilerplate code by providing repository interfaces.
* Handles most CRUD operations out of the box.

### 2. Code Comparison:

#### Using Hibernate:

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;  
}

#### Using Spring Data JPA:

// Repository Interface  
public interface EmployeeRepository extends JpaRepository<Employee, Integer> { }  
  
// Service Class  
@Service  
public class EmployeeService {  
 @Autowired  
 private EmployeeRepository employeeRepository;  
  
 @Transactional  
 public void addEmployee(Employee employee) {  
 employeeRepository.save(employee);  
 }  
}

### 3. When to Use What:

| Feature | JPA | Hibernate | Spring Data JPA |
| --- | --- | --- | --- |
| Type | Specification | Implementation | Abstraction over JPA |
| Boilerplate Reduction | ❌ | ❌ | ✅ |
| Suitable for Students | ❌ | ❌ | ✅ |
| Ease of Use | Moderate | Complex | Easy |
| Default in Spring Boot | ❌ | ✅ | ✅ (via Hibernate) |

### 5. Summary:

* **JPA** is a specification.
* **Hibernate** is a tool that implements JPA.
* **Spring Data JPA** is a higher-level abstraction that makes working with JPA easy.
* Use **Spring Data JPA** in Spring Boot projects to focus on business logic, not boilerplate code.