# Spring Data JPA with Spring Boot, Hibernate

**Spring Data JPA - Quick Example**

**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  
 http://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <modelVersion>4.0.0</modelVersion>  
  
 <groupId>com.cognizant</groupId>  
 <artifactId>orm-learn</artifactId>  
 <version>0.0.1-SNAPSHOT</version>  
 <packaging>jar</packaging>  
  
 <name>orm-learn</name>  
  
 <parent>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-parent</artifactId>  
 <version>3.2.5</version>  
 <relativePath/> <!-- Lookup from Maven Central -->  
 </parent>  
  
 <properties>  
 <java.version>17</java.version>  
 </properties>  
  
 <dependencies>  
 <!-- Spring Web -->  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-web</artifactId>  
 </dependency>  
  
 <!-- Spring Data JPA -->  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-data-jpa</artifactId>  
 </dependency>  
  
 <!-- MySQL Driver -->  
 <dependency>  
 <groupId>com.mysql</groupId>  
 <artifactId>mysql-connector-j</artifactId>  
 <scope>runtime</scope>  
 </dependency>  
  
 <!-- Optional: Lombok (for annotations like @Getter) -->  
 <dependency>  
 <groupId>org.projectlombok</groupId>  
 <artifactId>lombok</artifactId>  
 <optional>true</optional>  
 </dependency>  
  
 <!-- Spring Boot Test -->  
 <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:**

logging.level.org.springframework=info  
logging.level.com.cognizant=debug  
logging.level.org.hibernate.SQL=debug  
logging.level.org.hibernate.type.descriptor.sql=trace  
  
spring.datasource.url=jdbc:mysql://localhost:3306/ormlearn  
spring.datasource.username=root  
spring.datasource.password=Shunmadhuri@2005  
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver  
  
spring.jpa.hibernate.ddl-auto=validate  
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect

**Country.java:**

package com.cognizant.ormlearn.model;  
  
import jakarta.persistence.\*;  
  
@Entity  
@Table(name = "country")  
public class Country {  
  
 @Id  
 @Column(name = "co\_code")  
 private String code;  
  
 @Column(name = "co\_name")  
 private String name;  
  
 // Getters and setters  
 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.ormlearn.repository;  
  
import com.cognizant.ormlearn.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.cognizant.ormlearn.service;  
  
import com.cognizant.ormlearn.model.Country;  
import com.cognizant.ormlearn.repository.CountryRepository;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
  
import java.util.List;  
  
@Service  
public class CountryService {  
  
 @Autowired  
 private CountryRepository countryRepository;  
  
 public List<Country> getAllCountries() {  
 return countryRepository.findAll();  
 }  
}

**OrmLearnApplication.java:**

package com.cognizant.ormlearn;  
  
import com.cognizant.ormlearn.model.Country;  
import com.cognizant.ormlearn.service.CountryService;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
import org.springframework.context.ApplicationContext;  
  
import java.util.List;  
  
@SpringBootApplication  
public class OrmLearnApplication {  
  
 private static CountryService *countryService*;  
  
 public static void main(String[] args) {  
 ApplicationContext context = SpringApplication.*run*(OrmLearnApplication.class, args);  
 *countryService* = context.getBean(CountryService.class);  
  
 *testGetAllCountries*();  
 }  
  
 private static void testGetAllCountries() {  
 List<Country> countries = *countryService*.getAllCountries();  
 for (Country c : countries) {  
 System.*out*.println(c);  
 }  
 }  
}

**MySQL:**

CREATE DATABASE ormlearn;

USE ormlearn;

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');

**OUTPUT:**

A screenshot of a computer program

AI-generated content may be incorrect.

**Difference between JPA, Hibernate and Spring Data JPA**

**JPA:**

Java Persistence API is referred to as JPA.

Consider JPA as a specification that outlines the proper way to store Java objects in a database. The problem is that JPA by itself isn't sufficient. It merely outlines what must be done, not how.

For instance, it offers annotations such as @Entity, @Id, @OneToMany, and so on, but it lacks any logic for data retrieval or persistence. Tools like Hibernate are useful in this situation.

**Hibernate:**

The JPA specification is implemented by the well-known ORM (Object Relational Mapping) tool Hibernate.

Therefore, Hibernate does the actual work of saving your objects into the database and retrieving them, while JPA establishes the rules.

In addition, Hibernate enhances JPA with additional features like automatic dirty checking, lazy loading, and caching. However, using Hibernate directly frequently requires manual session and transaction management, which can result in a large amount of boilerplate code.

**Spring Data JPA:**

Data from Spring JPA is a component of the broader Spring Data initiative. It abstracts much of the complexity of writing database access code and builds upon JPA.

Writing intricate DAO (Data Access Object) classes is not necessary when using Spring Data JPA. All you have to do is design a basic interface that extends JpaRepository, and Spring will take care of creating the implementation.

Moreover, it manages transactions in the background, simplifying and improving the readability of your code.

**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**

**EmployeeRepository.java:**

public interface EmployeeRepository extends JpaRepository<Employee, Integer> {

}

**EmployeeService.java:**

@Autowired

private EmployeeRepository employeeRepository;

@Transactional

public void addEmployee(Employee employee) {

employeeRepository.save(employee);

}

The Spring Data JPA version is cleaner and more maintainable.