**Spring Data JPA - Quick Example**

**Create a Eclipse Project using Spring Initializr**

* Go to <https://start.spring.io/>
* Change Group as “com.cognizant”
* Change Artifact Id as “orm-learn”
* In Options > Description enter "Demo project for Spring Data JPA and Hibernate"
* Click on menu and select "Spring Boot DevTools", "Spring Data JPA" and "MySQL Driver"
* Click Generate and download the project as zip
* Extract the zip in root folder to Eclipse Workspace
* Import the project in Eclipse "File > Import > Maven > Existing Maven Projects > Click Browse and select extracted folder > Finish"
* Create a new schema "ormlearn" in MySQL database. Execute the following commands to open MySQL client and create schema.

**Application.properties**

# Log config

logging.level.org.springframework=info

logging.level.com.cognizant=debug

logging.level.org.hibernate.SQL=trace

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

# Console 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

**OrmLearnApplication.java**

**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");

}

**Pom.xml**

<dependencies>

<dependency>

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

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

</dependency>

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<scope>runtime</scope>

</dependency>

<dependency>

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

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

</dependency>

<dependency>

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

<artifactId>spring-boot-devtools</artifactId>

<optional>true</optional>

</dependency>

</dependencies>

**Country table creation**

* 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.

create table country(co\_code varchar(2) primary key, co\_name varchar(50));

* Insert couple of records into the table

insert into country values ('IN', 'India');

insert into country values ('US', 'United States of America');

**Persistence Class - com.cognizant.orm-learn.model.Country**

* Open Eclipse with orm-learn project
* Create new package com.cognizant.orm-learn.model
* Create Country.java, then generate getters, setters and toString() methods.
* Include @Entity and @Table at class level
* Include @Column annotations in each getter method specifying the column name.

**Country.java**

**package** com.cognizant.ormlearn.model;

**import** javax.persistence.\*;

@Entity

@Table(name="country")

**public** **class** Country {

@Id

@Column(name="co\_code")

**private** String code;

@Column(name="co\_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.cognizant.ormlearn.repository;

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

**import** org.springframework.stereotype.Repository;

**import** com.cognizant.ormlearn.model.Country;

@Repository

**public** **interface** CountryRepository **extends** JpaRepository<Country, String> {

}

**CountryService.java**

**package** com.cognizant.ormlearn.service;

**import** java.util.List;

**import** javax.transaction.Transactional;

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

**import** org.springframework.stereotype.Service;

**import** com.cognizant.ormlearn.model.Country;

**import** com.cognizant.ormlearn.repository.CountryRepository;

@Service

**public** **class** CountryService {

@Autowired

**private** CountryRepository countryRepository;

@Transactional

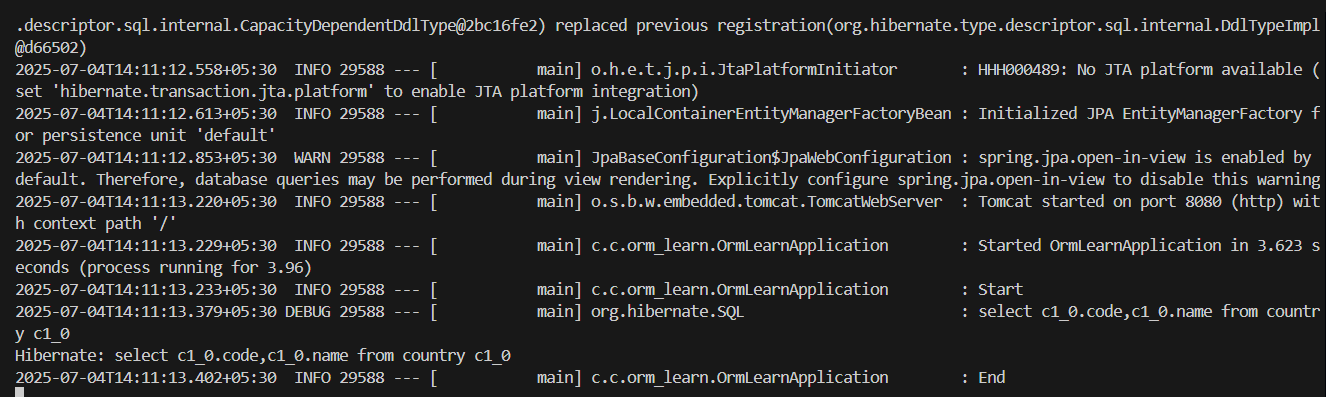
**public** List<Country> getAllCountries() {

**return** countryRepository.findAll();

}

}

**OUTPUT:**



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

| **Aspect** | **JPA (Java Persistence API)** | **Hibernate** | **Spring Data JPA** |
| --- | --- | --- | --- |
| **Type** | Specification (interface) | Implementation of JPA | Abstraction over JPA (helper framework) |
| **Provided By** | Java (by Oracle) | Red Hat | Spring Framework |
| **Purpose** | Defines standard APIs for ORM | Provides actual ORM functionality | Simplifies data access using repository abstraction |
| **What it is?** | A set of interfaces and annotations | A library that implements JPA (and adds extras) | A Spring module built on top of JPA |
| **Contains** | @Entity, @Id, EntityManager, etc. | Session, Criteria API, HQL, caching, etc. | Repositories like JpaRepository, custom queries |
| **Example Use** | EntityManager.persist(obj) | session.save(obj) | userRepo.save(obj) |
| **Configuration Required** | Needs manual setup of provider (like Hibernate) | Needs configuration of sessionFactory, etc. | Minimal setup — Spring Boot handles most automatically |
| **Ease of Use** | Moderate — lower-level APIs | More powerful but more verbose | Very easy — focuses on conventions and productivity |
| **Custom Queries** | Use JPQL or native queries | Supports JPQL, HQL, Criteria API | Supports JPQL, native SQL, derived methods |
| **Learning Curve** | Medium | Slightly higher (because of additional features) | Lowest (especially with Spring Boot) |

**Summary**

- JPA is just a standard — it defines how Java objects relate to database tables.  
 - Hibernate is a JPA provider — it implements JPA and offers additional features.  
 - Spring Data JPA is a Spring abstraction — it makes working with JPA easier by removing boilerplate code.

**Simple Example**

// JPA  
 EntityManager em = emf.createEntityManager();  
 em.persist(new User());  
  
 // Hibernate  
 Session session = sessionFactory.openSession();  
 session.save(new User());  
  
 // Spring Data JPA  
 @Autowired  
 UserRepository repo;  
 repo.save(new User());