**Spring Data JPA - Quick Example**

A **Spring Boot + Spring Data JPA** application that connects to MySQL, defines a **Country** entity, and retrieves all records using a CountryRepository—all **without implementing the repository** interface manually

### **Core Components**

1. **Entity Definition**  
    A JPA entity class annotated with @Entity and mapped to the country table (columns co\_code, co\_name)
2. **Repository Interface**  
    An interface extending JpaRepository<Country, String> in the com.cognizant.ormlearn.repository package. Spring auto-generates its implementation
3. **Service Layer**  
    A service class annotated with @Service and @Transactional, injecting CountryRepository to call methods like findAll()
4. **Application Configuration**

application.properties configures the MySQL datasource and Hibernate dialect

spring.jpa.hibernate.ddl-auto=validate ensures schema alignment

Logging levels controlled to output SQL via Hibernate:  
 logging.level.org.hibernate.SQL=trace

1. **Main Class**  
    Loads Spring Boot context, retrieves CountryService, and calls a method like testGetAllCountries(), which logs the list

Difference between JPA, Hibernate and Spring Data JPA

|  |  |  |
| --- | --- | --- |
| JPA | Hibernate | Spring Data JPA |
| 1. Specification/API | Implementation of JPA | Framework abstraction over JPA |
| 2. Annotations, EntityManager | ORM engine, HQL, caching | Auto‑generated repositories, query methods |
| 3. Define entities, use EntityManager/JPQL | Implement persistence and advanced ORM features | Extend JpaRepository; call CRUD & custom finder methods without implementation |
| 4. None – needs a provider | Can be used standalone or as JPA provider | Requires a JPA provider (e.g., Hibernate) |
| 5. Requires manual DAO code | Reduces some boilerplate via Session API | Eliminates almost all DAO boilerplate |