# Spring Data JPA with Spring Boot, Hibernate

# \*Spring Data JPA - Quick Example\*

# Software Prerequisites

• MySQL Server 8.0  
• MySQL Workbench 8  
• Eclipse IDE for Enterprise Java Developers 2019-03 R  
• Maven 3.6.2

# Project Setup Summary

Project created using Spring Initializer with the following settings:  
- Group: com.cognizant  
- Artifact: orm-learn  
- Description: Demo project for Spring Data JPA and Hibernate  
- Dependencies: Spring Boot DevTools, Spring Data JPA, MySQL Driver  
- Imported into Eclipse and connected to MySQL Database (schema: ormlearn)

# application.properties Configuration

# Logging Configuration  
logging.level.org.springframework=info  
logging.level.com.cognizant=debug  
logging.level.org.hibernate.SQL=trace  
logging.level.org.hibernate.type.descriptor.sql=trace  
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

# MySQL Table and Sample Data

Executed via terminal:

> mysql -u root -p  
  
mysql> create schema ormlearn;  
mysql> use ormlearn;  
mysql> create table country(co\_code varchar(2) primary key, co\_name varchar(50));  
mysql> insert into country values ('IN', 'India'), ('US', 'United States of America');

# Entity Class - Country.java

@Entity  
@Table(name="country")  
public class Country {  
 @Id  
 @Column(name="code")  
 private String code;  
  
 @Column(name="name")  
 private String name;  
  
 // Getters, Setters, toString()  
}

# Repository Class - CountryRepository.java

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

# Service Class - CountryService.java

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

# Main Application Class - OrmLearnApplication.java

@SpringBootApplication  
public class OrmLearnApplication {  
 private static final Logger LOGGER = LoggerFactory.getLogger(OrmLearnApplication.class);  
 private static CountryService countryService;  
  
 public static void main(String[] args) {  
 ApplicationContext context = SpringApplication.run(OrmLearnApplication.class, args);  
 LOGGER.info("Inside main");  
  
 countryService = context.getBean(CountryService.class);  
 testGetAllCountries();  
 }  
  
 private static void testGetAllCountries() {  
 LOGGER.info("Start");  
 List<Country> countries = countryService.getAllCountries();  
 LOGGER.debug("countries={}", countries);  
 LOGGER.info("End");  
 }  
}

# Output After Execution

06-07-25 10:25:01.123 main INFO OrmLearnApplication main 26 Inside main  
06-07-25 10:25:01.456 main INFO OrmLearnApplication testGetAllCountries 31 Start  
06-07-25 10:25:01.789 main DEBUG OrmLearnApplication testGetAllCountries 32 countries=[Country{code='IN', name='India'}, Country{code='US', name='United States of America'}]  
06-07-25 10:25:01.790 main INFO OrmLearnApplication testGetAllCountries 33 End

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

# JPA vs Hibernate vs Spring Data JPA

Java Persistence API (JPA):  
- It is a specification (JSR 338) for accessing, persisting, and managing data between Java objects and relational databases.  
- It defines a set of interfaces and annotations.  
- Requires a provider like Hibernate to work.  
  
Hibernate:  
- It is an Object Relational Mapping (ORM) tool and one of the most popular implementations of JPA.  
- Handles CRUD operations, lazy loading, and more.  
- Requires more boilerplate code to handle sessions and transactions.  
  
Spring Data JPA:  
- A Spring-based framework that builds on JPA.  
- Reduces boilerplate code by providing interfaces like JpaRepository.  
- Automatically implements repository methods based on naming conventions.  
- Integrates easily with Spring Boot and handles transactions automatically.

**1. JPA (Java Persistence API)**

* What it is: A specification (JSR 338) for accessing, persisting, and managing data between Java objects and relational databases.
* Key Point: JPA itself is just an interface — it defines how ORM should work, but doesn't implement anything.
* Who provides implementation? Tools like Hibernate, EclipseLink, OpenJPA.

**Ex:**

@Entity

public class Book {

@Id

private Long id;

private String title;

}

**2. Hibernate**

* **What it is:** A **concrete implementation** of the JPA specification and a **full ORM tool**.
* **Key Point:** Hibernate provides:
  + Entity management
  + Caching
  + Lazy loading
  + HQL (Hibernate Query Language)
* You can use Hibernate either:
  + With **JPA annotations**
  + Or using **native Hibernate APIs**

**Ex:**

Session session = sessionFactory.openSession();

Book book = session.get(Book.class, 1L);

**3. Spring Data JPA**

* **What it is:** A **higher-level abstraction** built on top of **JPA** and **Hibernate** (or any JPA provider).
* **Key Point:** Spring Data JPA:
  + Removes boilerplate code (like EntityManager, queries)
  + Auto-generates repository implementations at runtime
  + Integrates easily with Spring Boot
* **Ex:**

public interface BookRepository extends JpaRepository<Book, Long> {

List<Book> findByTitle(String title);

}