**1.Spring Data JPA – 5 Key Points**

* Spring Data JPA reduces boilerplate code by providing ready-to-use repository interfaces for common database operations.
* It is built on top of the Java Persistence API (JPA) and typically uses Hibernate as the default JPA implementation.
* Developers can create interfaces that extend JpaRepository, and Spring auto-generates CRUD and pagination methods.
* Supports method-name-based query derivation, JPQL (Java Persistence Query Language), and native SQL queries using @Query.
* Seamlessly integrates with Spring Boot, enabling features like auto-configuration, in-memory database support, and transaction management.

**Spring Data JPA – Quick Example**

### Database Configuration

1. **Create a MySQL Schema**:

* CREATE SCHEMA ormlearn;

1. **Configure** ``:

* spring.datasource.url=jdbc:mysql://localhost:3306/ormlearn  
  spring.datasource.username=root  
  spring.datasource.password=root  
  spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver  
    
  spring.jpa.hibernate.ddl-auto=validate  
  spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5Dialect  
    
  logging.level.org.hibernate.SQL=trace  
  logging.level.org.hibernate.type.descriptor.sql=trace

**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()  
}

**CountryRepository.java**

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

**CountryService.java**

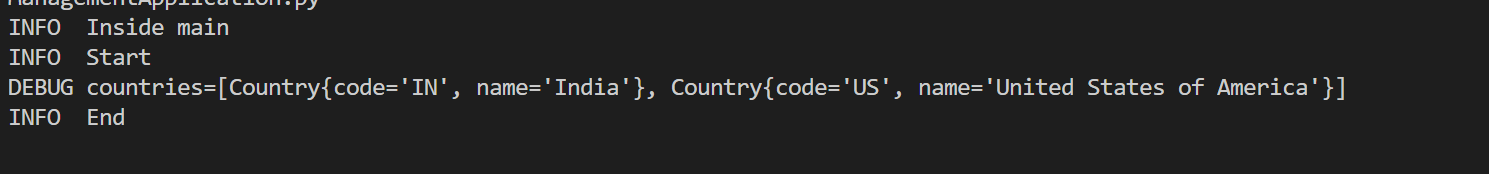
@Service  
public class CountryService {  
 @Autowired  
 private CountryRepository countryRepository;  
  
 @Transactional  
 public List<Country> getAllCountries() {  
 return countryRepository.findAll();  
 }  
}

**OrmLearnApplication.java**

@SpringBootApplication  
public class OrmLearnApplication {  
 private static CountryService countryService;  
 private static final Logger LOGGER = LoggerFactory.getLogger(OrmLearnApplication.class);  
  
 public static void main(String[] args) {  
 ApplicationContext context = SpringApplication.run(OrmLearnApplication.class, args);  
 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**

When you run the application, the following log output is expected:

This output indicates that the CountryService.getAllCountries() method successfully retrieved the list of countries from the MySQL database using Spring Data JPA.

**2.Explain the difference between Java Persistence API, Hibernate and Spring Data JPA**

**Java Persistence API (JPA)**

* **JPA is a specification**, not a framework or tool.
* It defines a set of **standards and guidelines** for object-relational mapping (ORM) in Java.
* It is part of **Jakarta EE (formerly Java EE)** — specified under **JSR 338**.
* **Does not provide an actual implementation**; only defines interfaces and annotations.
* Requires a **provider** (like Hibernate, EclipseLink) to be usable.

**Hibernate**

* **Hibernate is an ORM framework** that provides the actual implementation of the JPA specification.
* It can be used **with or without JPA**, offering advanced features beyond JPA.
* Provides **query language (HQL)**, caching, lazy loading, etc.
* One of the **most popular JPA providers** in the Java ecosystem.

**Spring Data JPA**

* A **part of Spring Data project** — provides an abstraction layer over JPA (typically Hibernate).
* Simplifies data access layers by **eliminating boilerplate code** (e.g., no need to write EntityManager code).
* Uses **Repository interfaces** (CrudRepository, JpaRepository) with auto-implemented methods like:
  + findById(), save(), delete(), findByName(), etc.
* Supports **custom queries** using method names, @Query, or native SQL.