**DIFFERENCE BETWEEN JPA , HIBERNATE AND SPRING DATA JPA**

**JPA ( JAVA PERSISTENCE API )**

JPA (Java Persistence API) is a standard specification provided by Java to manage relational data in Java applications using ORM (Object-Relational Mapping).

**JPA is just a specification** – it provides only **interfaces and annotations**, **not implementations**.

It tells how persistence (storing Java objects in a database) should be done.

To actually use JPA, you need a provider like:

* **Hibernate** (most common)
* **EclipseLink**
* **OpenJPA**, etc.

**Key Concepts in JPA**

**1. @Entity**

Marks a Java class as a **persistent entity**, i.e., it maps to a table in the database.

java

Copy code

@Entity

@Table(name = "country") // optional

public class Country {

@Id

private String code;

private String name;

}

* @Entity: Tells JPA this class should be stored in a DB table.
* @Table(name = "country"): Optional – gives a specific name to the table.

**2. @Id**

Marks the **primary key** of the entity.

java

Copy code

@Id

private String code;

Every entity **must have a unique ID** field.

**3. @Column**

Optional. Specifies details about the column mapping.

java

Copy code

@Column(name = "country\_name")

private String name;

If omitted, JPA maps the field to a column with the same name by default.

**4. EntityManager**

The **core interface** in JPA that manages the lifecycle of entities:

* Persist (save)
* Find
* Remove (delete)
* Merge (update)

java

Copy code

@EntityManager

private EntityManager entityManager;

entityManager.persist(user); // save

entityManager.find(User.class, id); // retrieve

**5. JPQL (Java Persistence Query Language)**

* A SQL-like language for querying Java entities.
* Works on Java objects, not tables.

Example:

java

Copy code

SELECT c FROM Country c WHERE c.name = 'India'

This is like: SELECT \* FROM country WHERE name = 'India', but for entities.

**🏗️ Example – JPA Basic Setup**

**1. Define the Entity Class:**

java

Copy code

@Entity

public class Country {

@Id

private String code;

private String name;

// getters, setters

}

**2. Create a Repository (manually in pure JPA):**

java

Copy code

public class CountryRepository {

@PersistenceContext

private EntityManager entityManager;

public List<Country> getAllCountries() {

return entityManager.createQuery("from Country", Country.class).getResultList();

}

}

In Spring Data JPA, we use JpaRepository, which automatically gives these methods.

**HIBERNATE**

Hibernate is an ORM (Object-Relational Mapping) framework for Java. It implements the JPA specification, but it also provides many extra powerful features.

**Hibernate** is *how to do it* (it's the implementation).

It **maps Java classes to database tables** and handles all the boilerplate like:

* SQL queries
* ResultSet parsing
* Table joins
* Caching
* Transactions
* Dirty checking, etc.

**SPRING DATA JPA**

Spring Data JPA is a Spring-based abstraction over JPA (and Hibernate) that automatically generates the repository layer (CRUD, queries, paging, sorting, etc.).

You just define an interface, like:

java

Copy code

public interface CountryRepository extends JpaRepository<Country, String> {

}

And it gives you:

* Save
* Update
* Delete
* Find by ID
* Find All
* Paging
* Sorting
* Derived queries like findByName("India")

Behind the scenes, Spring Data JPA uses Hibernate (or any JPA provider) to perform real DB operations.

**Key Concepts in Spring Data JPA**

**1. Repositories**

In Spring Data JPA, you don’t implement DAOs (Data Access Objects). You just **create an interface** that extends JpaRepository, and Spring auto-generates the implementation at runtime.

java

Copy code

@Repository

public interface CountryRepository extends JpaRepository<Country, String> {

}

Now you get all basic methods:

java

Copy code

countryRepository.findAll();

countryRepository.findById("IN");

countryRepository.save(country);

countryRepository.deleteById("US");

**2. Derived Query Methods**

Spring Data JPA can automatically create SQL queries **just from method names**!

java

Copy code

List<Country> findByName(String name);

List<Country> findByNameStartingWith(String prefix);

No need to write SQL or HQL.

**3. Custom Queries**

If needed, you can still write manual queries using JPQL or native SQL:

java

Copy code

@Query("SELECT c FROM Country c WHERE c.name LIKE %:name%")

List<Country> searchByName(@Param("name") String name);

Or native SQL:

java

Copy code

@Query(value = "SELECT \* FROM country WHERE name = ?", nativeQuery = true)

List<Country> nativeSearch(String name);

**4. Paging and Sorting**

Spring Data JPA supports pagination and sorting out-of-the-box.

java

Copy code

Pageable pageable = PageRequest.of(0, 5, Sort.by("name"));

Page<Country> page = countryRepository.findAll(pageable);

**5. Transactional Service Layer**

Spring recommends creating a @Service layer where you autowire your repository and use it in business logic.

Example in your project:

java

Copy code

@Service

public class CountryService {

@Autowired

private CountryRepository countryRepository;

@Transactional

public List<Country> getAllCountries() {

return countryRepository.findAll();

}

}

**6. Spring Boot Integration**

Spring Boot auto-configures everything:

* It auto-detects JpaRepository interfaces
* It sets up Hibernate under the hood
* It even configures your DataSource, dialect, transaction manager

As long as you include this dependency:

xml

Copy code

<dependency>

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

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

</dependency>

**Detailed Comparison: JPA vs Hibernate vs Spring Data JPA**

| **Feature / Aspect** | **JPA (Java Persistence API)** | **Hibernate** | **Spring Data JPA** |
| --- | --- | --- | --- |
| **Type** | Specification / API (Interface only) | ORM Framework / Implementation of JPA | Abstraction / Framework built on top of JPA and Hibernate |
| **Provided By** | Oracle (Java EE / Jakarta EE standard) | Red Hat | Spring Framework (Pivotal / VMware) |
| **Purpose** | Standardize persistence (ORM) in Java | Provide full implementation of JPA and additional features | Simplify JPA usage by reducing boilerplate (auto-repository generation, abstraction) |
| **Is It an Implementation?** | No – Just defines interfaces and annotations | Yes – Concrete implementation of JPA | No – Uses Hibernate (or another JPA provider) internally |
| **Needs External Provider?** | Yes (like Hibernate, EclipseLink) | No – It is a provider | Yes – Internally depends on Hibernate or any other JPA provider |
| **Commonly Used With** | Java EE, Jakarta EE, or Spring (via Hibernate) | Java SE, Java EE, Spring | Spring Boot / Spring Framework |
| **Key Annotations** | @Entity, @Id, @OneToMany, @ManyToOne, @Column, etc. | Same as JPA (plus extras like @Cascade, @LazyCollection) | Reuses JPA annotations, adds repository annotations like @Repository |
| **Entity Management** | Requires manual use of EntityManager | Uses EntityManager or Hibernate Session | No manual code – uses JpaRepository, CrudRepository |
| **DAO / Repository Layer** | Must write repository manually | Must write repository manually | Auto-generated – just extend JpaRepository<T, ID> |
| **Integration with Spring** | Requires manual setup | Can be used with Spring but with more configuration | Seamless with Spring Boot – auto-configuration, minimal setup |
| **Ease of Use** | Medium – requires boilerplate and config | Harder – more config, more control | Easiest – minimum code, Spring Boot integration |
| **Use Case** | For standard, implementation-agnostic ORM needs | For full control and advanced ORM needs | For rapid development with minimal boilerplate |
| **Learning Curve** | Moderate | High – more complex APIs and options | Low – abstraction, convention over configuration |
| **Default in Spring Boot?** | No | Yes – Hibernate is the default JPA provider | Yes – Spring Data JPA is the default abstraction layer |
| **Extensible?** | Yes | Yes | Yes – Custom Repositories, QueryDSL, Specifications |
| **Open Source?** | Yes (as part of Jakarta EE) | Yes | Yes |