## **Difference between JPA, Hibernate and Spring Data Jpa.**

## **What is JPA (Java Persistence API)?**

**JPA** stands for **Java Persistence API**.  
 It’s **not a tool, library, or framework** — it’s a **specification (a set of interfaces and annotations)** provided by Java EE (now Jakarta EE).

JPA defines a **standard way to map Java objects to relational database tables** and how to manage those persistent objects.

But remember: **JPA by itself doesn’t do any actual work**. It just defines *how things should work*, not *how they’re implemented*.

For example, it defines annotations like:

* @Entity — to mark a class as a database entity.
* @Table — to specify the table name.
* @Id — to specify the primary key.
* @Column — to map a field to a table column.

But these annotations are just part of the spec — for them to actually perform actions like saving data to the database, you need an **implementation**.

That’s where Hibernate comes in.

## **What is Hibernate?**

**Hibernate** is a **framework (an actual implementation)** of the **JPA specification**.

Think of it like this:  
 If JPA defines the rules for object-relational mapping, **Hibernate is one of the tools that follows those rules and makes it work**.

When you run a JPA-based application, Hibernate:

* Converts your Java objects to database tables.
* Executes the SQL statements under the hood.
* Manages database connections.
* Handles caching, transactions, lazy-loading, and other advanced ORM features.

Besides implementing the JPA specification, Hibernate also provides **additional powerful features not covered by JPA**, such as:

* Automatic dirty checking (detecting when an entity’s state changes and auto-updating the DB)
* Connection pooling
* 1st and 2nd level caching
* Custom interceptors
* Native SQL support
* Auditing

So, when people say "I’m using JPA," most of the time they mean "I’m using Hibernate as the JPA implementation."

## **What is Spring Data JPA?**

Now, this is where things get even more interesting.  
 **Spring Data JPA** is a part of the **Spring ecosystem** that **builds on top of JPA** and typically uses Hibernate underneath by default.

The idea behind Spring Data JPA is:

* You shouldn’t have to write lots of boilerplate code (like DAOs, CRUD operations, or custom queries).
* Spring will generate the implementation for you, based on the method names in your repository interfaces.

For example:  
 If you want to retrieve a country by its code — in plain JPA with Hibernate, you’d write something like:

Query query = entityManager.createQuery("from Country where code = :code");

query.setParameter("code", "IN");

Country country = (Country) query.getSingleResult();

But in **Spring Data JPA**, you’d simply define an interface:

public interface CountryRepository extends JpaRepository<Country, String> {

Country findByCode(String code);

}

And Spring will automatically create the implementation at runtime.

It does this using **method name conventions** (like findByCode, findByNameContaining, etc.) and intelligent parsing.  
 You don’t have to implement these methods yourself — **Spring Data JPA handles it for you.**

Also — Spring Data JPA provides:

* Pagination and sorting support out of the box.
* Integration with QueryDSL and JPQL.
* The ability to use native SQL queries if needed.
* Transaction management via Spring’s @Transactional annotation.

## **How do they relate to each other?**

In a typical Spring Boot application using **Spring Data JPA**:

* **You use JPA annotations** in your model classes (@Entity, @Id, @Column, etc.).
* **Spring Data JPA generates the repository code** based on your repository interfaces.
* **Under the hood, Spring Data JPA uses a JPA implementation** — by default, **Hibernate** — to perform the actual database operations.

If you don’t specify a JPA implementation, Spring Boot auto-configures Hibernate as the default one.

## **Why do we need all three?**

**JPA alone** is like the rulebook.  
 But you can’t play the game with just a rulebook — you need players and referees.

**Hibernate** is the player who knows how to play the game according to the rulebook.

**Spring Data JPA** is like a game manager that organizes the match for you so you can just watch and enjoy without worrying about setting up players, keeping scores, or calling out fouls — it automates much of the tedious work and lets you focus on writing business logic.

## **Final Summary in Words:**

* **JPA** provides **the specification** for ORM — how Java objects should be mapped to database tables.
* **Hibernate** is **a tool that implements the JPA specification** and adds more features.
* **Spring Data JPA** is a **Spring-based abstraction** that uses JPA and Hibernate internally but eliminates much of the boilerplate code, letting you focus on your application logic by automatically creating data access layers through interfaces and naming conventions.