# **JPA vs Hibernate vs Spring Data JPA**

## What is JPA (Java Persistence API)?

* JPA is a Java Specification (JSR 338).
* It defines standard annotations (@Entity, @Id, @Table) and APIs (EntityManager) for Object Relational Mapping (ORM).
* JPA itself has no implementation — it is just an interface and rules.
* You need a provider (like Hibernate) to make JPA work at runtime.

Example:

@Entity  
@Table(name = "STUDENT")  
public class Student {  
 @Id  
 private Long id;  
 private String name;  
}

This uses JPA annotations — but alone does not run.

## What is Hibernate?

* Hibernate is a popular ORM framework for Java.
* It existed before JPA — it inspired the creation of JPA.
* Hibernate is a JPA implementation — it provides the actual code that maps Java classes to database tables.
* Hibernate also adds extra features not covered by JPA: caching, custom queries (HQL), batch processing, etc.

Example:

Configuration cfg = new Configuration();  
cfg.configure("hibernate.cfg.xml");  
SessionFactory factory = cfg.buildSessionFactory();

Hibernate handles the SessionFactory, Session, and transactions.

## What is Spring Data JPA?

* Spring Data JPA is a Spring framework module.
* It is an abstraction layer over JPA (and Hibernate or any JPA provider).
* It removes boilerplate code for data access.
* You only define repository interfaces, and Spring auto-generates the implementation.

Example:

public interface StudentRepository extends JpaRepository<Student, Long> {}

No need to write EntityManager code manually.

## How they connect

|  | JPA | Hibernate | Spring Data JPA |
| --- | --- | --- | --- |
| Type | Specification only | Implementation of JPA | Abstraction above JPA |
| Maintained by | Oracle / JCP | Hibernate team | Spring team |
| Does real work? | NO | YES | YES(delegates to JPA provider) |
| Boilerplate code? | Needs manual EntityManager | Needs SessionFactory | Auto-generates repositories |
| Use case | Standard for all providers | Rich ORM features | Simpler & faster dev |

## How they work together

* JPA: Rules and contracts (e.g., @Entity).
* Hibernate: Provides the engine that implements JPA.
* Spring Data JPA: Auto-creates repository beans that use Hibernate under the hood.

## Practical Flow

Without Spring Data JPA:

* Use Hibernate directly with SessionFactory, Session, manual transactions.

With Spring Data JPA:

* Use JpaRepository interfaces.
* Spring Boot auto-configures Hibernate as the JPA provider.
* Minimal config needed.

Example Spring Boot repository:

public interface BookRepository extends JpaRepository<Book, Long> {}

No SessionFactory or EntityManager needed by the developer.