# Difference between JPA, Hibernate, and Spring Data JPA

In Java persistence and ORM (Object Relational Mapping) frameworks, JPA, Hibernate, and Spring Data JPA are closely related but serve different purposes. This document explains the difference between them.

## 1. JPA (Java Persistence API)

• JPA is a specification, not an implementation.  
• It defines a set of interfaces and rules for object-relational mapping and managing relational data in Java applications.  
• Part of Java EE (Jakarta EE) standards.  
• It does not provide any working code by itself — requires an implementation (like Hibernate, EclipseLink, etc.).  
• Key annotations: @Entity, @Table, @Id, @GeneratedValue, @OneToMany, etc.

## 2. Hibernate

• Hibernate is one of the most popular implementations of the JPA specification.  
• It is a full-fledged ORM framework that can also work in non-JPA mode using its proprietary APIs.  
• Provides additional features beyond JPA (caching, better fetching strategies, custom queries with HQL).  
• Acts as a bridge between Java objects and database tables.  
• Supports automatic schema generation, lazy loading, and dirty checking.

## 3. Spring Data JPA

• Spring Data JPA is a part of the Spring Data project.  
• It is not a JPA implementation — it builds on top of an existing JPA provider (like Hibernate).  
• Simplifies the use of JPA by reducing boilerplate code for repositories.  
• Provides the Repository and CrudRepository interfaces for automatic query generation.  
• Allows defining queries using method names or JPQL without manually writing DAO implementations.

## 4. Comparison Table

|  |  |  |  |
| --- | --- | --- | --- |
| Aspect | JPA | Hibernate | Spring Data JPA |
| Type | Specification | Implementation of JPA | Framework built on top of JPA |
| Provides Working Code? | No | Yes | Yes (depends on JPA provider) |
| Purpose | Defines ORM standards | Maps Java objects to DB tables | Simplifies JPA repository usage |
|  |  |  |  |

