**Difference between JPA, Hibernate and Spring Data JPA**

1. **JPA (Java Persistence API) — Specification**

| **Aspect** | **Description** |
| --- | --- |
| What | JPA is just an interface/specification. It defines a standard for ORM (Object-Relational Mapping) in Java. |
| Who provides it | Oracle (as part of Java EE / Jakarta EE specification). |
| What it defines | Annotations and interfaces to map Java objects to relational tables (e.g., @Entity, @Id, @OneToMany, EntityManager, etc.). |
| Implements nothing | It does not provide any working implementation; it only defines what needs to be done. |
| Main Goal | Decouple the persistence layer from implementation, allowing switching between providers like Hibernate, EclipseLink. |
| Use case | Use JPA annotations in entity classes. |
| **Example:** |  |

@Entity

public class Employee {

@Id

@GeneratedValue

private Long id;

private String name;

}

**2. Hibernate — JPA Implementation**

| **Aspect** | **Description** |
| --- | --- |
| What | Hibernate is a concrete implementation of JPA. It existed before JPA and became its most common provider. |
| Implements JPA | Hibernate implements all interfaces from JPA and adds additional features beyond JPA. |
| Extra Features | Caching (first- and second-level), Hibernate Query Language (HQL), performance optimizations, and batch processing. |
| How used | It is the engine that executes the persistence logic defined by JPA annotations. |
| Use case | Commonly used in Spring Boot applications as the default JPA provider. |

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect

**3. Spring Data JPA — Abstraction Layer**

| **Aspect** | **Description** |
| --- | --- |
| What | Spring Data JPA is a Spring project that simplifies JPA by reducing boilerplate code. |
| Main Goal | Automatically generates the data access layer (repository layer) and handles common CRUD operations. |
| Features | Query method derivation, @Query annotation support, pagination, sorting, projections, and specifications. |
| Repositories | You define interfaces extending JpaRepository or CrudRepository, and Spring provides the implementation at runtime. |
| Use case |  |

public interface EmployeeRepository extends JpaRepository<Employee, Long> {

List<Employee> findByName(String name);

@Query("SELECT e FROM Employee e WHERE e.salary > :salary")

List<Employee> findHighEarners(@Param("salary") double salary);

}

**Relationship Summary**

| **Layer** | **Depends On** | **Purpose** |
| --- | --- | --- |
| Spring Data JPA | Uses JPA and Hibernate | Simplifies data access, removes DAO boilerplate |
| JPA | Interface only | Specifies ORM mapping and API contract |
| Hibernate | Implements JPA | Performs actual ORM operations: SQL generation, caching, dirty checking, etc. |

**Typical Stack in a Spring Boot Application**

Spring Boot

↓

Spring Data JPA (simplified repository layer)

↓

JPA (annotations and API contract)

↓

Hibernate (actual implementation)

↓

MySQL, PostgreSQL, or other RDBMS

**Use Case Summary**

| **Requirement** | **Best Fit** |
| --- | --- |
| Define entity mappings | JPA |
| Handle persistence, caching, and SQL | Hibernate |
| Automatically generate CRUD repositories | Spring Data JPA |