JPA vs Hibernate vs Spring Data JPA

|  |  |  |  |
| --- | --- | --- | --- |
| Feature / Aspect | JPA (Java Persistence API) | Hibernate (ORM Framework) | Spring Data JPA (Spring abstraction) |
| Type | Specification / API (interface-based) | ORM Implementation Framework | Spring-based Data Access Abstraction |
| Defined By | Jakarta EE / Java EE | Red Hat (JBoss) | Spring Team (Pivotal) |
| Purpose | Standard for ORM in Java | Concrete implementation of ORM & JPA | Simplifies JPA usage with powerful repository support |
| Implementation | Requires a provider (e.g., Hibernate, EclipseLink) | Acts as a JPA provider (and adds enhancements) | Works on top of JPA (commonly uses Hibernate underneath) |
| Common Annotations | `@Entity`, `@Id`, `@Table`, `@OneToMany` | Same as JPA, plus specific ones like `@LazyCollection`, `@Fetch` | Uses JPA annotations + `@Repository`, interfaces like `JpaRepository` |
| EntityManager Usage | Must be manually handled | Offers `Session` API (extends EntityManager) | Hidden behind Spring interfaces (`CrudRepository`, `JpaRepository`) |
| Query Language | JPQL (Java Persistence Query Language) | JPQL + HQL (Hibernate Query Language) | JPQL + method names (`findByName`, etc.) + `@Query` annotation |
| Boilerplate Code | More manual coding for queries and entity operations | Less than raw JPA due to helper features | Very minimal — most tasks handled by Spring repositories |
| Ease of Use | Moderate | Easier than plain JPA | Easiest — declarative, convention-over-configuration |
| Advanced Features | Basic ORM support only | Caching, lazy/eager loading, batch fetches | Pagination, Sorting, Derived Queries, Specifications, Projections |
| Best For | When using different ORM providers or custom logic | When using full Hibernate power directly | When rapid development and minimal code is preferred |