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

**JPA (Java Persistence API)**

**Type**: Specification (interface/standard)

**Provided By**: Java EE (Jakarta EE)

**Purpose**: Defines a standard way to manage relational data in Java applications using ORM (Object Relational Mapping).

**Contains**:

* Annotations like @Entity, @Id, @OneToMany, etc.
* Interfaces like EntityManager, Query

**Does Not Provide**: Any concrete implementation or functionality on its own.

@Entity

public class Student {

@Id

@GeneratedValue

private Long id;

private String name;

}

**Note**: You cannot use JPA directly unless there’s a JPA provider like Hibernate.

**Hibernate**

**Type**: JPA Implementation (JPA Provider)

**Provided By**: Red Hat

**Purpose**: Implements all JPA interfaces and adds extra features beyond JPA specification.

**Role**: Actually handles how Java objects are persisted into a relational database.

**Features**:

* Dirty checking (detecting changes)
* Lazy loading, caching
* Native HQL (Hibernate Query Language)
* hibernate.cfg.xml for configuration

**Can Work With or Without JPA**: Hibernate has its own native API too.

Session session = sessionFactory.openSession();

Transaction tx = session.beginTransaction();

session.save(student);

tx.commit();

**In Spring applications, Hibernate is often configured as the default JPA provider.**

**Spring Data JPA**

**Type**: Framework (Abstraction Layer on Top of JPA)

**Provided By**: Spring Framework (Pivotal)

**Purpose**: Simplifies JPA-based persistence by eliminating boilerplate code.

**Key Features**:

* Auto-implementation of repository interfaces (no need to write SELECT queries for basic CRUD).
* Derived queries (e.g., findByNameContaining).
* Paging and Sorting (Pageable, Sort).
* Custom queries with @Query.

public interface StudentRepository extends JpaRepository<Student, Long> {

List<Student> findByFirstName(String name);

}

**Under the hood, Spring Data JPA uses JPA (standard) and Hibernate (as provider).**