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

**Java Persistence API (JPA)**

The Java Persistence API (JPA) is a formal specification (JSR 338) that defines a standardized way to map Java objects to relational database tables, manage entity lifecycles, and perform queries using JPQL or the Criteria API. As a pure interface and annotation model, JPA itself provides no runtime behavior; it simply prescribes how persistence should work in a provider‑independent fashion, ensuring that code written against JPA interfaces can switch implementations without significant rewrites.

// EntityManager usage under pure JPA

EntityManagerFactory emf = Persistence.createEntityManagerFactory("my-pu");

EntityManager em = emf.createEntityManager();

em.getTransaction().begin();

em.persist(new Employee("Alice", "Engineering"));

em.getTransaction().commit();

em.close();

emf.close();

**Hibernate**

Hibernate is a full‑featured object‑relational mapping (ORM) framework that serves as one of the most popular JPA implementations. It not only fully honors the JPA contracts (via its EntityManager façade) but also offers its own native API (Session), multi‑level caching, performance‑tuning tools, and proprietary extensions. Those who require advanced caching strategies, custom database dialects, or fine‑grained control often turn to Hibernate’s additional capabilities beyond the baseline JPA specification.

// Native Hibernate Session usage

SessionFactory factory = new Configuration().configure().buildSessionFactory();

Session session = factory.openSession();

session.beginTransaction();

Integer id = (Integer) session.save(new Employee("Bob", "Marketing"));

session.getTransaction().commit();

session.close();

factory.close();

**Spring Data JPA**

Spring Data JPA is a Spring‑managed abstraction layer that sits atop any JPA provider (commonly Hibernate) to drastically reduce boilerplate code. By defining simple repository interfaces—extending JpaRepository—developers gain auto‑generated implementations for CRUD operations, pagination, sorting, and query derivation based on method names. Spring Data JPA also leverages Spring’s declarative transaction management, allowing persistence operations to execute within well‑defined transactional boundaries without explicit EntityManager or Session handling.

// 1. Repository interface

public interface EmployeeRepository extends JpaRepository<Employee, Integer> { }

// 2. Service using the repository

@Service

public class EmployeeService {

@Autowired

private EmployeeRepository repo;

@Transactional

public Employee addEmployee(Employee employee) {

return repo.save(employee);

}

}