**Comparison: JPA vs Hibernate vs Spring Data JPA**

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

Think of JPA as a **set of rules or a contract** that Java developers follow to work with databases. It doesn’t actually perform any operations itself—rather, it defines *how* things should be done. It’s similar to an interface in Java: it sets the structure, but the real functionality needs to be implemented by something else—like Hibernate.

**2. Hibernate**

Hibernate is the **actual tool** that brings JPA’s rules to life. It’s a powerful **ORM (Object-Relational Mapping) framework** that handles everything from connecting to the database to saving and retrieving data using Java objects. Not only does it follow the JPA guidelines, but it also includes several advanced features that make database handling easier and more efficient.

**3. Spring Data JPA**

Spring Data JPA builds on top of both JPA and Hibernate, offering an even more convenient way to interact with databases. It helps you avoid repetitive boilerplate code by providing **ready-to-use methods** like save, findById, delete, and more. All you need is a simple interface, and Spring takes care of the rest. It’s tightly integrated with Spring Boot, which makes setup and configuration almost effortless.

**Code Comparison**

**Hibernate Example:**

public Integer addEmployee(Employee employee) {

Session session = factory.openSession();

Transaction tx = null;

Integer employeeID = null;

try {

tx = session.beginTransaction();

employeeID = (Integer) session.save(employee);

tx.commit();

} catch (HibernateException e) {

if (tx != null) tx.rollback();

e.printStackTrace();

} finally {

session.close();

}

return employeeID;

}

**Spring Data JPA Example:**

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

@Autowired

private EmployeeRepository employeeRepository;

@Transactional

public void addEmployee(Employee employee) {

employeeRepository.save(employee);

}