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

Java Persistence API (JPA)

* JPA is a **standard (JSR 338)** that defines how Java objects are stored in a relational database.
* It’s just a **specification**, meaning it doesn't provide any working code by itself.
* To use JPA, we need an implementation — that’s where **Hibernate** comes in.

Hibernate

* Hibernate is one of the most popular **implementations of JPA**.
* It’s an **Object-Relational Mapping (ORM)** tool that handles the interaction between Java code and the database.
* When we use Hibernate directly, we have to manage things like **sessions**, **transactions**, and exception handling ourselves.

Spring Data JPA

* Spring Data JPA builds on top of JPA and Hibernate, adding another layer of abstraction.
* It doesn’t implement JPA itself but uses an existing implementation like Hibernate underneath.
* The goal is to **reduce boilerplate code** — like opening sessions or writing standard CRUD operations.
* Spring Data JPA also handles **transaction management** automatically with simple annotations.

**Hibernate vs Spring Data JPA**

**Hibernate** (Manual Way)

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** (Simplified Way)

EmployeeRepository.java

public interface EmployeeRepository extends JpaRepository<Employee, Integer> {

}

EmployeeService.java

@Autowired

private EmployeeRepository employeeRepository;

@Transactional

public void addEmployee(Employee employee) {

employeeRepository.save(employee);

}

On comparing, Spring Data JPA helps cut down a lot of boilerplate and makes the code cleaner and easier to maintain.