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

**Java Persistence API (JPA)**

* JPA is **only a specification** (JSR 338) that defines how Java objects are mapped to relational database tables.
* It provides guidelines for **CRUD operations**, relationships, queries, etc.
* JPA **does not provide an implementation** — needs a provider like Hibernate or EclipseLink to actually perform ORM operations.

**Hibernate**

* Hibernate is a **popular ORM tool** that **implements the JPA specification**.
* It provides the actual code and libraries to persist, retrieve, and manage Java objects with a relational database.
* Hibernate can also offer features beyond the JPA spec (e.g., caching, advanced fetching).

**Spring Data JPA**

* Spring Data JPA is **not an implementation of JPA** — it’s an **abstraction layer** on top of a JPA provider like Hibernate.
* It helps **reduce boilerplate code** by generating repository methods automatically.
* It integrates with Spring’s dependency injection and transaction management.
* You mostly work with simple repository interfaces instead of writing boilerplate Session or EntityManager code.

**Code Comparison:**

**Hibernate Example:**

java

CopyEdit

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:**

**EmployeeRepository.java:**

java

CopyEdit

public interface EmployeeRepository extends JpaRepository<Employee, Integer> {

}

**EmployeeService.java:**

java

CopyEdit

@Autowired

private EmployeeRepository employeeRepository;

@Transactional

public void addEmployee(Employee employee) {

employeeRepository.save(employee);

}