**Spring Data JPA – Quick Example**

Imagine you want to save and fetch Employee records.

**Entity class**:

@Entity

public class Employee {

@Id

@GeneratedValue

private Long id;

private String name;

private String role;

}

**Repository interface**:

public interface EmployeeRepository extends JpaRepository<Employee, Long> {

List<Employee> findByRole(String role);

}

**Usage in a Service**:

@Autowired

private EmployeeRepository repo;

public void addEmployee() {

Employee emp = new Employee();

emp.setName("John");

emp.setRole("Developer");

repo.save(emp);

}

public List<Employee> getDevelopers() {

return repo.findByRole("Developer");

}

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

| **Aspect** | **JPA** | **Hibernate** | **Spring Data JPA** |
| --- | --- | --- | --- |
| **What it is** | *It is a specification*  (standard API for ORM in Java) | It is a *framework* (implementation of JPA and more) | *It is a library* that simplifies JPA-based data access |
| **Who defines it** | Java EE / Jakarta EE standard | Open-source project | Spring project |
| **Role** | Defines interfaces and annotations (e.g., @Entity, EntityManager) | Provides the actual working code that maps Java objects to DB tables | Adds extra features on top of JPA to reduce boilerplate code |
| **Examples of use** | EntityManager.persist() | Session.save(), JPA EntityManager | CrudRepository.save(), query methods, pagination |
| **Relationship** | Hibernate *implements* JPA | Hibernate = JPA provider + extra features | Uses JPA (often with Hibernate) under the hood |