**WEEK – 3**

**SPRING DATA JPA WITH SPRING BOOT, HIBERNATE**

**DIFFERENCE BETWEEN JPA, HIBERNATE AND SPRING DATA JPA**

The Java Persistence API (JPA) is a standard specification (defined in JSR 338) that provides a way to map Java objects to relational database tables and vice versa. It offers a consistent approach for data persistence without tying the developer to a specific implementation. JPA is only a specification — it does not contain any working code. One of the most popular implementations of JPA is Hibernate.

Hibernate is an Object-Relational Mapping (ORM) framework that implements the JPA specification. It allows developers to map Java classes to database tables and manage relationships between objects. However, when using Hibernate directly, developers need to manage boilerplate code such as session handling, transactions, exception management, etc.

Spring Data JPA is a Spring-based abstraction over JPA. It doesn't implement JPA itself but internally uses an implementation like Hibernate. Spring Data JPA significantly simplifies JPA-based data access layers. It automatically handles common tasks like transaction management, query creation, and CRUD operations by extending repository interfaces. This reduces the amount of code developers need to write.

Below is a comparison of how the same task — adding an employee to the database — is done using Hibernate vs. Spring Data JPA.

**Using Hibernate:**

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;

}

In the above code using Hibernate, the developer needs to explicitly manage the session, transaction, error handling, and cleanup.

**Using Spring Data JPA:**

**EmployeeRepository.java**:

public interface EmployeeRepository extends JpaRepository<Employee, Integer> {

}

**EmployeeService.java:**

@Autowired

private EmployeeRepository employeeRepository;

@Transactional

public void addEmployee(Employee employee) {

employeeRepository.save(employee);

}

In the Spring Data JPA version, boilerplate code is removed. The JpaRepository interface provides all basic operations, and Spring automatically handles the transaction, session management, and exception handling behind the scenes.