**WEEK-3**

**SPRING DATA JPA AND HIBERNATE HANDSON**

**1. Need and Benefits of ORM (Object-Relational Mapping)**

ORM is a method that connects Java (or similar language) objects with database tables. Instead of writing SQL manually, developers work with objects and classes to perform operations. This approach makes the codebase cleaner and easier to maintain.

**Advantages of ORM:**• Cuts down repetitive SQL and JDBC code.  
• Enables automatic mapping between objects and tables for simpler database handling.  
• Allows database flexibility (e.g., switching from MySQL to PostgreSQL with minimal changes).  
• Includes support for features like lazy fetching, caching, and managing transactions.  
• Reduces risk of SQL injection via parameter binding.

**Limitations:**• Can introduce slight performance issues in specific use cases.  
• Might conceal SQL strengths if overused.  
• Complex database operations may still require native SQL or JPQL.

**2. Need and Benefits of Spring Data JPA**

Spring Data JPA is part of the broader Spring framework that enhances JPA and Hibernate capabilities. It streamlines the creation of data access layers in Spring-based applications.

**Evolution Path:**  
• Hibernate originally relied on XML to map objects to tables.  
• Later, annotations like @Entity and @Id simplified this setup.  
• Spring Data JPA further advanced it by removing the need to write boilerplate query code.

**Advantages of Spring Data JPA:**  
• Reduces repetitive code through repositories like JpaRepository.  
• Offers built-in CRUD and derived queries based on method names.  
• Seamlessly integrates with Spring Boot and other Spring components.  
• Supports in-memory databases (e.g., H2) for efficient development and testing.

**3. Core Objects of Hibernate Framework**

Hibernate, a well-known ORM solution in Java, relies on several primary components:

• **SessionFactory** – A thread-safe factory used to create Session objects. Created once per app.  
• **Session** – A lightweight object for single-threaded database operations (CRUD).  
• **Transaction** – Manages commit and rollback processes.  
• **Connection Provider** – Supplies JDBC connections for Hibernate use.  
• **TransactionFactory** – Interface for building transaction objects.

These parts collaborate to manage database access separate from business logic.

**4. ORM Implementation Using Hibernate XML and Annotation Configuration**

Hibernate provides two major ways to configure ORM:

**Using XML:**  
• Design an entity class (e.g., Employee.java).  
• Map fields using Employee.hbm.xml.  
• Place configuration in hibernate.cfg.xml.  
• Load configuration, obtain SessionFactory, manage sessions/transactions.

**Using Annotations:**  
• Use annotations such as @Entity, @Table, @Id, and @Column.  
• No need for .hbm.xml, but still require hibernate.cfg.xml for DB setup.  
• This method is modern and widely preferred.

**Typical Flow (both):** Load configuration → open session → begin transaction → persist data → commit → close session.

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

• **JPA** – Java Persistence API: A specification (only interfaces and rules) for mapping Java objects to tables; no direct implementation.  
• **Hibernate** – A powerful JPA implementation that offers more features and can be used independently.  
• **Spring Data JPA** – A higher-level abstraction built on JPA + Hibernate, reducing code by auto-generating query methods from interface names.

**In summary:**  
• JPA = standard API,  
• Hibernate = implementation of JPA,  
• Spring Data JPA = abstraction to simplify JPA + Hibernate use.

**6. DML Operations using Spring Data JPA (on a single table)**

Spring Data JPA makes CRUD and other DML operations straightforward.

**Steps:**  
• Include dependencies for Spring Boot, JPA, and a database like H2/MySQL.  
• Create entity classes with appropriate annotations (@Entity, @Id, etc.).  
• Define a repository interface extending JpaRepository<Entity, ID>.

**Common DML Methods:**  
• findById(id) – Retrieves a specific record.  
• save(entity) – Inserts or updates data.  
• deleteById(id) – Removes a record.  
• findAll() – Gets all records.  
• Custom query methods like findByNameContaining(String name).

**In application.properties:**

spring.jpa.show-sql=true

spring.jpa.hibernate.ddl-auto=update

These settings assist with query logging and schema auto-generation.