**WEEK-3**

**SPRING DATA JPA AND HIBERNATE HANDSON**

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

ORM (Object-Relational Mapping) is a technique that bridges the gap between object-oriented programming and relational databases. It allows developers to use objects and classes in place of direct SQL queries when interacting with a database. This leads to cleaner and more manageable code.

**Advantages of Using ORM:**

* Minimizes repetitive SQL statements and the need for manual JDBC coding.
* Eases database operations by automatically linking class attributes to table columns.
* Enhances portability, allowing smooth transitions between different database systems like MySQL and PostgreSQL.
* Offers advanced features such as lazy fetching, caching, and managing transactions efficiently.
* Enhances security by using parameterized queries, thereby reducing the risk of SQL injection attacks.

**Limitations:**

* May introduce a slight performance hit in certain use cases.
* Could obscure the full capabilities of SQL if used improperly.
* For complex operations, developers might still need to rely on raw SQL or JPQL.

**2.** **Need and Benefits of Spring Data JPA**  
  
Spring Data JPA is a module within the Spring framework that builds upon JPA and Hibernate to streamline the creation of data access layers in Spring-based applications.

**Development Timeline:**

* Initially, Hibernate relied on XML configuration files to map Java classes to relational database tables.
* Later, annotations such as @Entity, @Id, and others simplified the mapping process.
* Spring Data JPA enhances this by eliminating the need to manually write common query implementations.

**Advantages of Spring Data JPA:**

* Minimizes repetitive code through built-in repository interfaces like JpaRepository.
* Simplifies CRUD operations and allows query creation directly via method naming conventions.
* Integrates seamlessly with Spring Boot and other components of the Spring framework.
* Supports in-memory databases like H2, enabling quick development and effective testing.

**3. Core Objects of Hibernate Framework**

Hibernate is the most popular ORM implementation in Java, and it relies on a few key components:

* **SessionFactory**: A thread-safe factory that creates Session objects. It's created once and reused.
* **Session**: A lightweight, single-threaded object used to interact with the database (CRUD operations).
* **Transaction**: Handles commit and rollback operations.
* **Connection Provider**: Manages JDBC connections for Hibernate.
* **TransactionFactory**: Strategy interface for creating transaction instances.

These components work together to abstract the database interaction layer from the business logic.

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

There are two main ways to configure Hibernate:

**XML Configuration:**

* Define your entity class (e.g., Employee.java).
* Create an XML file (Employee.hbm.xml) to map class fields to table columns.
* Configure Hibernate settings in hibernate.cfg.xml.
* Load the configuration, get SessionFactory, and manage sessions and transactions.

**Annotation Configuration:**

* Use annotations like @Entity, @Table, @Id, and @Column in the class directly.
* Skip mapping XML, but still use hibernate.cfg.xml for DB settings.
* This is more modern and widely used.

Both approaches follow a similar flow: load config → open session → begin transaction → persist data → commit → close session.

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

* **JPA**: Java Persistence API — a specification (just interfaces and rules). It defines how Java objects should be mapped to database tables, but it doesn't provide actual implementation.
* **Hibernate**: A popular implementation of JPA. It adds many additional features beyond JPA and can be used on its own.
* **Spring Data JPA**: A higher-level abstraction built on top of JPA and Hibernate. It reduces even more boilerplate by allowing developers to define repository interfaces with method names like findByName() without writing query code.

**In short:**

* JPA is the *standard*,
* Hibernate is an *implementation*,
* Spring Data JPA is an *abstraction layer* to make JPA+Hibernate easier to use.

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

Spring Data JPA simplifies basic CRUD and DML operations on entities.

**Setup:**

* Add Spring Boot, JPA, and H2/MySQL dependencies.
* Create your entity class with annotations (@Entity, @Id, etc.).
* Create a repository interface extending JpaRepository<Entity, ID>.

**DML Examples:**

* findById(id) → fetch a single record.
* save(entity) → insert or update a record.
* deleteById(id) → delete a record.
* findAll() → fetch all records.
* Custom queries like findByNameContaining(String name) → use query methods.

Also, configure application.properties:

properties

CopyEdit

spring.jpa.show-sql=true

spring.jpa.hibernate.ddl-auto=update

This helps with logging and schema generation.