Name : Arshiya Tabassum A

Superset ID : 6424209

**WEEK-3**

**SPRING DATA JPA AND HIBERNATE**

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

ORM is needed because applications are built using object-oriented languages like Java, but data is stored in relational databases using tables. This creates a mismatch between how data is represented in code (as objects) and how it's stored in the database (as rows and columns). ORM bridges this gap by automatically mapping Java objects to database tables, making it easier to manage data without writing manual SQL queries.

**Benefits of ORM:**

* Less code: Reduces boilerplate SQL code using simple method calls.
* Faster development: Developers work with objects instead of writing SQL.
* Automatic mapping: Maps class fields to table columns automatically.
* Database portability: Easily switch between databases (e.g., MySQL to PostgreSQL).
* Easy relationship handling: Supports one-to-one, one-to-many, and many-to-many mappings.
* Built-in transaction support: Simplifies transaction management.
* Improved security: Helps prevent SQL injection attacks.
* Lazy loading and caching: Loads only required data, improving performance.

**Drawbacks of ORM:**

* **Learning curve:** Complex to learn for beginners (especially mappings and configurations).
* **Performance issues:** Generated SQL may be less optimized than handwritten SQL.
* **Complex queries are difficult:** Writing joins and subqueries can be harder with ORM.
* **Debugging can be hard:** Tracing errors in mappings or queries is not always easy.
* **Less SQL control:** Developers may lose visibility into what SQL is being executed.
* **Not ideal for reporting:** For analytical tasks, raw SQL is usually better.

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

Spring Data JPA is a part of the Spring ecosystem that builds on top of JPA and Hibernate. It simplifies the development of data access layers in Spring applications.

**Evolution:**

* Initially, Hibernate used XML-based configuration to map classes to database tables.
* Then annotations (like @Entity, @Id, etc.) made configuration easier.
* Spring Data JPA takes it further by eliminating the need to write implementation code for common queries.

**Benefits of Spring Data JPA:**

* Reduces boilerplate code with built-in repository interfaces like JpaRepository.
* Easily supports CRUD operations and custom queries using method names.
* Integrates smoothly with Spring Boot and other Spring modules.
* Supports in-memory databases like H2 for fast development and 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.