**Introduzione: MyBatis e Hibernate**

* **MyBatis** e **Hibernate** sono strumenti per lavorare con i database in applicazioni Java. Entrambi consentono di eseguire query sul database, ma differiscono nel modo in cui operano:
  + **MyBatis**: fornisce un controllo dettagliato delle query attraverso file XML o annotazioni, dove ogni query deve essere definita esplicitamente.
  + **Hibernate**: è un framework ORM (*Object-Relational Mapping*) che si basa principalmente sulle **annotazioni**. Questo approccio è utile soprattutto in progetti dove le annotazioni possono essere facilmente interpretate e utilizzate dal programma.

**Cos'è Hibernate?**

* Hibernate è un **framework ORM** che facilita la mappatura delle tabelle di un database relazionale in oggetti Java (e viceversa).
* Una volta create le tabelle nel database, possiamo:
  + **Creare classi Java** che rappresentano quelle tabelle (note come **entità** in Hibernate).
  + Utilizzare annotazioni per mappare queste classi alle tabelle, definendo relazioni e proprietà.
  + Eseguire le operazioni standard di **CRUD** (*Create, Read, Update, Delete*) senza scrivere query SQL complesse.

**Configurazione di Hibernate in un progetto Java**

* Per utilizzare Hibernate in un progetto Java, dobbiamo configurare correttamente il progetto, includendo tutte le dipendenze necessarie:
  + Ad esempio, in un progetto **Maven**, possiamo aggiungere le dipendenze di Hibernate nel file pom.xml.
* Elementi non Java (come librerie esterne) devono essere dichiarati e resi disponibili al programma per funzionare.

**Mapping in Hibernate**

* Il **mapping** è il processo di collegamento tra una classe Java e una tabella del database.
* Ogni classe in Hibernate rappresenta una tabella del database e prende il nome di **entità**.
* Il mapping è definito tramite annotazioni, alcune delle quali più comuni sono:
  1. **@Entity**: definisce una classe come entità.
  2. **@Id**: specifica che un attributo è una **chiave primaria**.
  3. **@GeneratedValue**: indica la strategia per generare valori nella colonna della chiave primaria (es. incremento automatico).
     + Una delle strategie comuni è **GenerationType.IDENTITY**, che utilizza il comportamento autoincrementale del database.
  4. **@Column**: permette di definire i dettagli di una colonna (es. nome, lunghezza, unicità, ecc.).
  5. **@Transient**: esclude un attributo dal mapping, indicando che non rappresenta una colonna del database.

**Relazioni tra Entità**

Hibernate consente di rappresentare le relazioni tra tabelle utilizzando annotazioni specifiche. Questi sono i tipi di relazioni supportate:

1. **One-to-Many** (Uno a Molti):
   * Un'entità ha una relazione con molte altre entità.
   * Es.: Un autore può avere più libri.
2. **Many-to-One** (Molti a Uno):
   * Molte entità sono associate a una sola entità.
   * Es.: Più libri appartengono a un solo autore.
3. **One-to-One** (Uno a Uno):
   * Un'entità è associata a una sola altra entità.
   * Es.: Ogni utente ha un profilo.
4. **Many-to-Many** (Molti a Molti):
   * In realtà, si tratta di due relazioni uno-a-molti che si connettono attraverso una **tabella bridge**.
   * Es.: Studenti e corsi (uno studente può iscriversi a più corsi, e ogni corso può avere più studenti).

**Annotazioni Utilizzate:**

* **@OneToMany** e **@ManyToOne**: utilizzate per rappresentare relazioni uno-a-molti e molti-a-uno.
* **@OneToOne**: utilizzata per relazioni uno-a-uno.
* **@ManyToMany**: utilizzata per relazioni molti-a-molti.
* **@JoinColumn**: specifica il nome della colonna utilizzata come **chiave esterna**.
* **mappedBy**: indica la proprietà nella classe correlata che gestisce la relazione.

**Esempio di relazione Uno-a-Molti:**

Se abbiamo una classe Autore e una classe Libro, la relazione potrebbe essere definita così:

|  |
| --- |
| @Entity  **public** **class** Autore {     @Id     @GeneratedValue(strategy = GenerationType.IDENTITY)  **private** Long id;  ​     @OneToMany(mappedBy = "autore")  **private** List<Libro> libri;  }  ​  @Entity  **public** **class** Libro {     @Id     @GeneratedValue(strategy = GenerationType.IDENTITY)  **private** Long id;  ​     @ManyToOne     @JoinColumn(name = "autore\_id")  **private** Autore autore;  }  ​ |

**Particolarità di Hibernate**

* **Gestione delle operazioni di aggiornamento e cancellazione**:
  + Per aggiornare o cancellare un record, Hibernate richiede di caricare prima l'entità dal database. Non possiamo passare direttamente il valore della chiave primaria.
  + Questo meccanismo permette a Hibernate di tenere traccia delle relazioni e di sincronizzare i cambiamenti.

**Metodi Utili**

* **Session.get()**: Recupera un'entità dal database in base alla sua chiave primaria.
* **Session.find()**: Simile a get, ma può includere funzionalità aggiuntive.
* **Classe Criteria**: Utilizzata per costruire query dinamiche basate su criteri, senza scrivere direttamente SQL.

**Conclusione**

Hibernate semplifica notevolmente la gestione del database in applicazioni Java, eliminando gran parte della necessità di scrivere query SQL manuali. Utilizzando le annotazioni e il mapping delle entità, è possibile creare relazioni complesse tra tabelle in modo più naturale e orientato agli oggetti.