**⚙️ Quando servire una Entity dedicata**

🔍 **Contesto**:  
Hai una tabella ponte **con campi aggiuntivi** (es. stato\_libro, data\_prestito, ecc.).   
In questo caso non puoi più affidarti a un solo @ManyToMany; ti serve:

* Una **Entity** Java corrispondente alla tabella ponte.
* Una **chiave primaria composta** basata sulle due foreign key.

**Possiamo usare due approcci diversi: @IdClass e @EmbeddedId**

**🔍 Spiegazione Semplificata**

1. **@IdClass**
   * **Pensa a due moduli distinti:**
     + **Modulo Chiave (LibreriaId): contiene solo i campi chiave.**
     + **Modulo Entity (Libreria): ripete gli stessi campi chiave e gli altri attributi.**
   * **Vantaggio: rispecchia esattamente lo schema DB.**
   * **Svantaggio: più codice duplicato.**
2. **@EmbeddedId**
   * **Pensa a un contenitore unico (LibreriaKey) che “incapsula” i due campi chiave.**
   * **Entity (Libreria) ha un solo riferimento al contenitore.**
   * **Vantaggio: codice più compatto, modifiche più semplici.**
   * **Svantaggio: leggermente meno “esplicito” rispetto allo schema fisico.**

**Scegli @IdClass se devi rispettare tabelle legacy o vuoi rispecchiare esattamente le colonne.**

**Scegli @EmbeddedId se preferisci un design più pulito, con tutte le chiavi incapsulate in un unico oggetto.**

**In entrambi i casi implementa sempre equals() e hashCode() nella key class per garantire il corretto funzionamento di JPA.**

**🗝️ EmbeddedId: implementazione passo-passo**

// Creazione di una nuova relazione “utente possiede libro”

1. Crea un oggetto LibreriaKey con (idLibro, idUtente)

2. Crea un oggetto Libreria passando la key e lo stato

3. Imposta le reference a Libro e Utente (facoltativo, ma consigliato)

4. entityManager.persist(libreria)

5. JPA inserirà nella tabella libreria (id\_libro, id\_utente, stato\_libro)

**1. 📋 Struttura generale**

* **Classe embeddable** (LibreriaKey): incapsula le due FK
* **Entity dedicata** (Libreria): usa @EmbeddedId e aggiunge i campi extra

**2. 🔨 Definizione della key class**

**// LibreriaKey.java**

package com.mast.readup.entities;

import jakarta.persistence.Embeddable;

import java.io.Serializable;

import java.util.Objects;

**@Embeddable** // annotazione che indica che questa classe può essere usata come PK incorporata

**public class LibreriaKey implements Serializable** {

private Long idLibro; // FK verso Libro

private Long idUtente; // FK verso Utente

**//** ✅ **Costruttore vuoto richiesto da JPA**

public LibreriaKey() {}

**//** ✅ **Costruttore di comodo**

public LibreriaKey(Long idLibro, Long idUtente) {

this.idLibro = idLibro;

this.idUtente = idUtente;

}

**// ✅ equals():** due key sono uguali se idLibro e idUtente coincidono

@Override

public boolean equals(Object o) {

if (this == o) return true;

if (o == null || getClass() != o.getClass()) return false;

LibreriaKey other = (LibreriaKey) o;

return Objects.equals(idLibro, other.idLibro) &&

Objects.equals(idUtente, other.idUtente);

}

**// ✅ hashCode():** consistente con equals

@Override

public int hashCode() {

return Objects.hash(idLibro, idUtente);

}

**// ✅ Getter e Setter**

public Long getIdLibro() { return idLibro; }

public void setIdLibro(Long idLibro) { this.idLibro = idLibro; }

public Long getIdUtente() { return idUtente; }

public void setIdUtente(Long idUtente) { this.idUtente = idUtente; }

}

**Nota sul tipo di dato:**In Java, per mappare gli attributi delle entità JPA sulle colonne del database, scegliamo tipi wrapper (es. Long, Integer) invece dei primitivi (long, int).

**🔢 Range:**

* **int copre valori da –2.147.483.648 a 2.147.483.647.**
* **long (e Long) copre da –9.223.372.036.854.775.808 a 9.223.372.036.854.775.807.**

**🗳️ Nullabilità:**

* **I tipi primitivi non possono essere null: un int vale sempre almeno 0.**
* **I wrapper (Integer, Long) possono essere null, rappresentando l’assenza di valore.**

Immagine che contiene testo, schermata, Carattere, numero

Il contenuto generato dall'IA potrebbe non essere corretto.

**🏷️ Definizione dell’Entity che utilizza @EmbeddedId (Libreria)**

**// Libreria.java**

package com.mast.readup.entities;

import jakarta.persistence.\*;

**@Entity**

**@Table(name = "libreria")** // nome esatto della tabella ponte

public class Libreria {

**@EmbeddedId** // chiave primaria composta

private LibreriaKey id;

**@Column(name = "stato\_libro")**

private String statoLibro; // esempio di campo extra

**// ✅ Relazioni facoltative per navigazione bidirezionale**

@MapsId("idLibro") // lega id.idLibro al campo idLibro della key

@ManyToOne

@JoinColumn(name = "id\_libro")

private Libro libro;

@MapsId("idUtente") // lega id.idUtente al campo idUtente della key

@ManyToOne

@JoinColumn(name = "id\_utente")

private Utente utente;

**// ✅ Costruttore vuoto**

public Libreria() {}

**// ✅ Costruttore di comodo**

public Libreria(LibreriaKey id, String statoLibro) {

this.id = id;

this.statoLibro = statoLibro;

}

**// ✅ Getter e Setter**

public LibreriaKey getId() { return id; }

public void setId(LibreriaKey id) { this.id = id; }

public String getStatoLibro() { return statoLibro; }

public void setStatoLibro(String statoLibro) { this.statoLibro = statoLibro; }

}

**🔗 Relazioni per Navigazione in JPA**

Quando usiamo una **entity dedicata** con chiave composta, spesso vogliamo anche poter **navigare** da quella entity verso le entità correlate (Libro, Utente).   
Le righe in questione servono esattamente a questo.

**1. @ManyToOne**

* **Cos’è**  
  Un’annotazione che indica che **molte** istanze di questa entity (Libreria) possono riferirsi a **una sola** istanza di un’altra entity (Libro o Utente).
* **A cosa serve**  
  Permette di caricare l’oggetto correlato direttamente in Java, invece di gestire manualmente gli ID.
* **Parametro fetch = FetchType.LAZY**
  + ✅ **Lazy loading**: l’entità correlata viene caricata solo quando serve (chiamando getLibro() o getUtente()), migliorando le performance.
  + ⚠ Se occorre accedere ai dati correlati, ricordarsi di farlo dentro una sessione aperta.

**2. @JoinColumn**

* **Cos’è**  
  Mappa il campo Java su una colonna del database, specificando il **nome della colonna FK**.
* **A cosa serve**  
  Collega la proprietà Java (libro, utente) alla colonna fisica (id\_libro, id\_utente) nella tabella libreria.
* **insertable = false, updatable = false**
  + 🚫 Impedisce a JPA di includere queste colonne nelle istruzioni INSERT o UPDATE proprio perché le gestiamo via @EmbeddedId.
  + ✅ Evita conflitti e duplicazioni: i valori delle FK *arrivano* esclusivamente dalla chiave composta.

**3. @MapsId**

* **Cos’è**  
  Associa un campo della chiave composta (id.idLibro o id.idUtente) a una relazione @ManyToOne.
* **A cosa serve**
  1. **Collega** in modo dichiarativo il valore della FK nella chiave composta con l’oggetto di relazione.
  2. **Sincronizza** automaticamente:
     + Se assegni libreria.setLibro(unLibro), JPA mette id.idLibro = unLibro.getId().
     + Non devi settare manualmente i due valori separati.