**GIORNO 03**

**JPA Query PT. 1**

Repo del giorno: https://github.com/nclBaz/epicode-it-u4-d13-2

Recap:

Hibernate è una delle dipendenze installate per i progetti JPA ed è un ORM che fornisce un servizio di mapping tra classi Java e tabelle di database relazionali. In sintesi, permette la traduzione tra il linguaggio Java e il linguaggio SQL. L'alternativa è il JDBC, che è la connessione di base in Java in cui passiamo direttamente le query in formato stringa, con lo svantaggio di rendere il codice più complicato e macchinoso. Hibernate crea automaticamente le tabelle nel database relazionale a partire dalle classi che definiamo nel nostro progetto, grazie all'annotazione @Entity, riducendo così i tempi di sviluppo. L'ORM funge da intermediario tra l'applicazione Java e il database che stiamo utilizzando.

Le annotazioni utilizzate con Hibernate includono:

@Entity, fondamentale per definire la tabella corrispondente alla classe;

@Id, che rappresenta la chiave primaria della tabella;

@GeneratedValue, per creare un valore automatico;

@Enumerated, per gli Enum;

...e altre annotazioni opzionali per definire etichette per la tabella e le colonne (@Table, @Column).

Oggi parleremo di come stabilire relazioni tra le tabelle. La logica di lavoro di Hibernate prevede l'utilizzo dell'EntityManager, che è una classe (o interfaccia) utilizzata per connettersi al database. Ogni entità, cioè ogni classe che viene mappata in una tabella del database, viene gestita dall'EntityManager. Questo permette le operazioni CRUD, ovvero la lettura, modifica, eliminazione, ecc., dei record nelle tabelle. In questo modo possiamo interagire con le tabelle del database. L'EntityManager sa che una particolare classe, e quindi un oggetto istanziato da essa, è collegata a una tabella.

Gli oggetti gestiti dall'EntityManager sono il Persistence Context, ovvero una zona di memoria degli oggetti istanziati dalle classi @Entity, o i record di una tabella che rappresentano oggetti specifici. Il Persistence Context serve a evitare di "sovraccaricare" il database con richieste, fornendo un'area di memoria pre-database in cui possiamo aggiungere, modificare e manipolare gli oggetti, per poi salvarli nel database tramite il metodo .commit(), che contiene il metodo .flush().

Un oggetto istanziato da una classe Entity diventa parte del database solo quando interagiamo con l'EntityManager e diventa persistente (vedi .persistent()). Ogni EntityManager ha il proprio Persistence Context e viene generato dall'EntityManagerFactory (vedi la classe JpaUtil) che si collega al database tramite la persistent-unit nel file persistence.xml. In sintesi, l'EntityManager è una classe che ci permette di interagire con il database attraverso vari metodi (find, refresh, remove, createQuery...).

Libreria: **Project Lombok**. <https://mvnrepository.com/artifact/org.projectlombok/lombok> Tramite particolari annotazioni genera automaticamente: setters, getters, costruttore, toString ecc… dandoci un codice più snello:

Immagine che contiene testo, schermata, software, Software multimediale

Descrizione generata automaticamente

Immagine che contiene testo, schermata, software, Software multimediale

Descrizione generata automaticamente

Differenze tra con e senza lombock che è utile per i casi standard, senza però la possibilità di modificare il costruttore o il toString. In compenso abbiamo molto meno codice. Ovviamente posso incrociare le due tecniche.

Argomento di oggi sono la rappresentazione nel mondo JPA delle RELAZIONI (1 a 1, 1 a molti, molti a molti).

Immagine che contiene testo, schermata, Carattere, documento

Descrizione generata automaticamente

Se non usiamo il @GeneratedValue dobbiamo passare noi i valori delle chiave-primarie (vedi targhe per le auto o ISBN per i libri). Tendenzialmente chiediamo che vengano generate automaticamente.

Immagine che contiene testo, schermata, Carattere, documento

Descrizione generata automaticamente

Esistono 4 strategie per generare le chiavi-primarie:

1. Generazione automatica da parte del DB (se ad esempio abbiamo un long id, allora nel DB sarà Serial; un UUID id sarà una stringa univoca)
2. Forzando un incremento automatico ad ogni inserimento di un record
3. Attraverso la memorizzazione di una sequenze custom in cui setto valori (iniziale, finale, incremento, ecc..)
4. Attraverso tabelle dedicate per tenere in memoria i valori

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

Immagine che contiene testo, schermata, Carattere, algebra

Descrizione generata automaticamente

Immagine che contiene testo, schermata, algebra

Descrizione generata automaticamente

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

In alcuni casi la chiave-primaria può essere una combinazione di due colonne (o più). Un esempio è la colonna Nome-Cognome per Student.

Per la chiave composta serve definire una Classe specifica: Nella Classe AutomobileComposite prende due Id per due campi con @Id, Chiave primaria = targa+nazione. Non basta specificare @Id su più campi per generare una chiave composta altrimenti ci ritorna un errore.

Bisogna creare una Classe apposita che spieghi ad @Entity come è fatta la chiave composita.

Tale Classe avrà:

1. come attributi i valori attraverso cui voglio creare la chiave composita;
2. un costruttore vuoto.
3. Un costruttore che accetta i parametri che andranno a creare la chiave composta
4. Deve implementare l’interfaccia Serializable
5. Immagine che contiene testo, schermata, Carattere, software

   Descrizione generata automaticamenteAvere i metodi hashCode ed equals

Immagine che contiene testo, schermata, software, Software multimediale

Descrizione generata automaticamenteNella Classe->Tabella bisogna usare @IdClass che accetta come parametro la classe che abbiamo creato per gestire la chiave composita.

Immagine che contiene testo, Carattere, schermata

Descrizione generata automaticamenteEsempio con lombock -> le annotazioni creano implicitamente ciò di cui abbiamo bisogno

**MAPPING DELLE RELAZIONI**

Immagine che contiene testo, Carattere, algebra, ricevuta

Descrizione generata automaticamente

Relazioni esempi:

1. UNO A UNO:

Immagine che contiene testo, schermata, ricevuta, Carattere

Descrizione generata automaticamenteImmagine che contiene testo, schermata, linea, Carattere

Descrizione generata automaticamenteCODICE FISCALE, IMPRONTE DIGITALI, UN SOLO NUMERO DI TELEFONO O INDIRIZZO EMAIL per una App che stiamo progettando, capitale per una Nazione (e una Nazione per una capitale). Bisogna chiedersi se ribaltando il pov ci sia ancora un legame one-to-one. (Uno studente ha un solo prof, ma il contrario non è vero). L’annotazione One-to-one mette in relazione un campo di una tabella con la chiave-primaria di un’altra tabella.

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

Immagine che contiene testo, Carattere, schermata, algebra

Descrizione generata automaticamente

Cascade: abbiamo due tabelle collegate e se cancello la prima, cancello in cascatai dati della seconda.

Fetch: i join vengono fatti in maniera automatica (eager) o su richiesta (lazy). Eager non viene usato molto ma è preferibile richiamare la tabella collegata su richiesta.

Optional: stabilisce dei vincoli sulla chiave

Immagine che contiene testo, Carattere, schermata, algebra

Descrizione generata automaticamente

Immagine che contiene testo, schermata, Carattere, documento

Descrizione generata automaticamente

Immagine che contiene testo, schermata, software, Software multimediale

Descrizione generata automaticamente Creiamo una Classe Document, che abbia una Entità e un nome per la tabella (@Table). Tale Document avrà anche un id (@Id) con @generatedValue. Ogni documento avrà anche un codice e una data di emissione e Nazione di emissione. Possiamo dare una lunghezza max alle colonne.

Procediamo poi con tutto ciò che è necessario (getter, setter, toString, costruttore pieno e vuoto ecc…)

Immagine che contiene testo, software, Software multimediale, Software per la grafica

Descrizione generata automaticamente

Creiamo anche la Classe User e UserDAO e DocumentDAO(similmente a quanto già fatto)

Immagine che contiene testo, schermata, software, Software multimediale

Descrizione generata automaticamente

Creato tutto quello che serve per collegare Java con DB, stabiliamo una relazione one-to-one tra l’user e il document.

In User:

Immagine che contiene testo, schermata, software, Software multimediale

Descrizione generata automaticamente

con @OnetoOne stabilisco già una relazione univoca tra User e Document.

Immagine che contiene testo, schermata, software, Software multimediale

Descrizione generata automaticamente

Nel main:

Creati due oggetti distinti (un User e un Document che sarà collegato all’user), posso settare il documento ajejeDoc all’user ajeje. Li salvo entrambi con i rispettivi metodi .save() reciprocamente dell’UserDAO e DocumentDAO in modo da creare le tabelle nel DB.

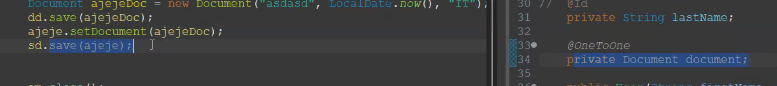


Immagine che contiene testo, schermata, software, Software multimediale

Descrizione generata automaticamente

In pgAdmin troveremo le tabelle User e Document con la creazione di una relazione tra document\_id e l’id dell’user.

Tale relazione è stata instaurata da @OnetoOne in User e dal setDocument nel main:

Immagine che contiene testo, schermata, Software multimediale, software

Descrizione generata automaticamente

In questo momento la relazione non è ancora propriamente completata: posso passare dall’User al Document ma il viceversa non è ancora valido. Non è ancora bidirezionale.

Per renderla tale bisogna creare in Document un private User user che ha annotazione mappedBy:

Immagine che contiene testo, Carattere, schermata

Descrizione generata automaticamente

ATTENZIONE!

Se nel main ora cerco l’user tramite id e ne stampo il documento:

Rischiamo un errore di StackOverflow causato dal toString creato da lombock che crea un loop. La soluzione è fare il toString manualmente senza farci stampare l’user, o in generale tutto ciò che crea un collegamento circolare ed evitare una stampa ricorsiva che ci porta in stackoverflow.

Risolto il problema del toString, nel main posso farmi stampare il documento partendo dall’id dell’user e viceversa:

Immagine che contiene testo, schermata, Software multimediale, software

Descrizione generata automaticamente

Abbiamo raggiunto una circolarità che ci permette di passare dall’user al suo documento e viceversa.

CASCADING

Immagine che contiene testo, schermata, documento, Carattere

Descrizione generata automaticamente

Si tratta di un’opzione per gestire cosa fare in caso di cancellazione di un user: devo cancellare anche il documento?

Di default non c’è nessun cascading e va abilitato. Può essere abilitato anche per altre relazioni, non solo one-to-one.

1. UNO A MOLTI:

Ogni user e document (legato all’user) ha un id come chiave-primaria. Posso accedere alle caratteristiche del documento tramite il document\_id dell’user (identificato dal suo id che è primaria dell’user). Document\_id è la secondaria di users.

Immagine che contiene testo, schermata, Rettangolo, diagramma

Descrizione generata automaticamenteOgni user scrive N post (relazione one-to-many). Ogni post (con proprio id) avrà anche un id identificativo dell’user (user\_id che sarà quella secondaria). In questo modo ogni post ha un riferimento all’user (e quindi ai suoi dati quali id, name,surname e document\_id) che l’ha scritto. La tabella user non ha nessun “riferimeto” al post. Se così fosse stabiliremmo una relazione one-to-one univoca e sarebbe come dire che quell’user (esempio un giornalista) abbia scritta un unico articolo nella sua vita (articolo identificato da una propria chiave-primaria, id).

Immagine che contiene testo, schermata, algebra, Carattere

Descrizione generata automaticamente

Creata la Classe dei Blog, e blogDAO (come sempre), nella Classe User:

Immagine che contiene testo, schermata, Software multimediale, software

Descrizione generata automaticamente

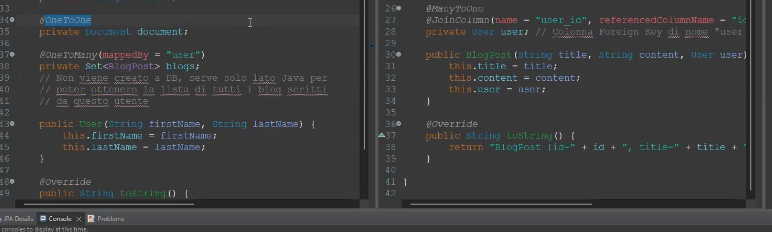
Ad UNO User avremo MOLTI blogs, ma per MOLTI blog avremo UN user.

Ora abbiamo una relazione circolare tra un User e i suoi Blogs.

Ora possiamo cercare un user dal suo id (tramite i metodi del UserDAO) e farmi un get dei suoi blog e stamparli. In più da ogni blog, che avrà un unico user, avrò uno User che posso stamparmi tramite l’id del post e i metodi del BlogDAO.

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

RIGUARDARE ULTIMI 15/20 MINUTI DELLA LEZIONE!

MOLTI A MOLTI: