TECNICHE DI PROGRAMMAZIONE

30/04/19

Esercizio sui grafi. Esercizio “Arts Mia”.

Come prima cosa andiamo a guardare il diagramma entità-relazioni e notiamo che ci sono 5 tabelle diverse:

* Objects che contiene gli oggetti d’arte.
* Artists che contiene id e nome degli artisti che hanno fatto le opere.
* Authorship che è una tabella temporanea usata per mappare la relazione molti a molti tra Artisti e Oggetti.
* Exhibitions che rappresenta le mostre in cui sono state esposte le opere
* Exhibition\_Objects che mappa la relazione molti a molti tra oggetti e mostre.

Analizziamo poi cosa ci chiede il punto 1. Nel caso in cui non girasse il punto 1, l’esame non è passato quindi è molto importante.

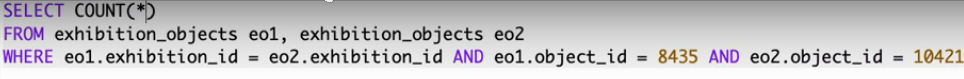
La cosa più importante da capire è la query per aggiungere gli archi, in quanto quella per aggiungere i vertici è molto banale (basta recuperare tutti gli oggetti).

Nella lezione precedente avevamo analizzato 3 modi per trovare questa query e risolvere il problema. La più facile non sempre funziona (in certi esami sì e in altri no). Logicamente funziona mentre praticamente no.

Apriamo quindi Heidi, eseguiamo il file .sql e analizziamo le tabelle per capire cosa rappresentano. Successivamente pensiamo ad una query per aggiungere gli archi. La più semplice diceva che dovevamo fare un doppio ciclo sui vertici e controllare se questi due vertici erano collegati tra di loro.

L’informazione sugli oggetti e sulla loro presenza nelle mostre, ce l’abbiamo nella tabella Exhibition-objects. In essa infatti sono presenti sia gli id degli oggetti che gli id delle mostre. È una tabella quindi che ci permette di capire ad ogni mostra quali oggetti erano presenti. Per avere una buona rappresentazione visiva, andiamo su Heidi, apriamo questa tabella e mettiamo in ordine crescente l’id della mostra (premo su exhibition\_id).

Se vogliamo vedere se due oggetti sono stati alla stessa mostra, dovrò fare il Join di questa tabella (Exhibition-objects) con sé stessa.

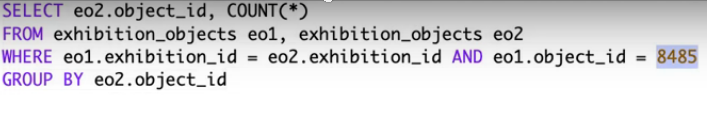


Quindi nel FROM prendiamo le due tabelle rinominate con due nomi differenti, nel WHERE scriviamo la condizione di JOIN e imponiamo i due oggetti diversi. Infine, nel SELECT contiamo quante volte i due oggetti sono presenti nella stessa mostra.

Questa query svolta per tutte le coppie di oggetti mi da tutti gli archi presenti nel grafo. È molto semplice, il problema è che ci sono 85 mila oggetti e quindi per svolgere questa query per tutti gli oggetti, mi servirebbero circa 41 giorni. Nel caso in cui il DB sia molto grande, questo metodo non possiamo usarlo (se all’esame succedesse una cosa del genere, l’esame è considerato con NON passato).

Dobbiamo quindi cercare di far fare più lavoro al DB in una sola volta (cambiare la query).

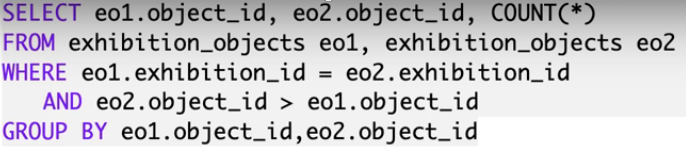
Possiamo provare a tenere fisso un oggetto mentre l’altro variabile. Quindi noi vogliamo che, dato un oggetto, la query ci restituisca tutti gli oggetti ad esso adiacenti con il loro peso. La SELECT cambierà e, dato che abbiamo un parametro insieme ad un COUNT, abbiamo bisogno di una GROUP BY. Il codice è:



Se runniamo questa query otteniamo come risultato tutti gli oggetti che sono stati in almeno una mostra con l’oggetto 8485 e il loro relativo peso. Inoltre, noi non vogliamo che venga preso anche l’oggetto stesso quindi nel WHERE aggiungiamo un AND in cui specifichiamo che l’id dell’oggetto 2 sia maggiore dell’id dell’oggetto 1.

Utilizzando questa query, riduciamo il processo di un FOR e il tempo per svolgere la query scende a circa 51 secondi!

Il problema però ci chiede le coppie di oggetti e il numero di volte che sono presenti nella stessa mostra. Rispetto all’immagine precedente dobbiamo ancora aggiungere l’id dell’oggetto 1 nel SELECT e di conseguenza anche nel GROUP BY. Query finale:



Nel WHERE abbiamo tolto l’AND che gestiva l’id del primo oggetto in quanto lo andremo a mettere su Eclipse noi con il punto di domanda (id variabile e non fisso).

Con questa query in 600 ms otteniamo tutte le coppie di oggetti e il numero di volte che sono presenti nella stessa mostra; praticamente abbiamo ottenuto il nostro grafo.

Ora passiamo al programma Java. All’esame noi avremo una serie di cose già pronte:

* L’interfaccia grafica.
* Un DAO con alcuni metodi già fatti (i più semplici). Nel nostro caso abbiamo un metodo che ci consente di recuperare tutti gli oggetti dal DB.
* Il DBConnect con il Connection Pooling.
* Nel Package Model abbiamo già le classi che modellano le entità che ci servono nel DB (oggetto e exhibition nel nostro caso).

Sicuramente ci serviranno delle classi in più. Sconsigliato di modificare le cose già date.

In questo esercizio non è presente il Model nel Package Model e quindi l’andiamo a creare. **È in questa classe che andiamo a creare il grafo.** Il testo molto probabilmente ci dirà il tipo di grafo da creare. Noi dobbiamo creare un grafo semplice, pesato e non ordinato. Abbiamo bisogno di un “SimpleWeightedGraph”.

Definiamo il grafo come un’interfaccia e successivamente lo andiamo ad istanziare con la classe corretta.

Il grafo ha come vertici gli oggetti e come archi “DefaultWeightedEdge” in quanto sono pesati.

Una cosa importante è che noi **abbiamo bisogno dei riferimenti degli oggetti presenti nel grafo**. Per aggiungere un arco tra due vertici, infatti, ho bisogno del riferimento di questi due vertici. Nella lezione precedente abbiamo provato a creare un oggetto nuovo con solo l’id ma abbiamo visto che non andava bene.

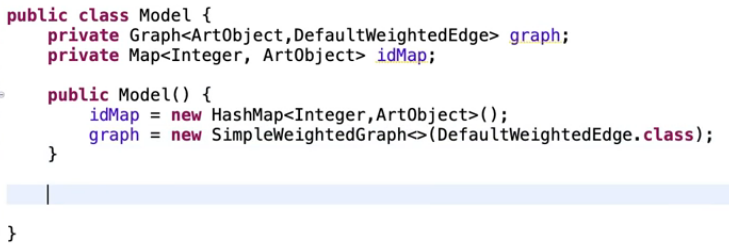
Un’alternativa è quella di assicurarsi che una vota creato un oggetto, io lo abbia creato una sola volta. Così facendo, quando ho bisogno di riusarlo, ho un posto dove andarlo a recuperare.

Potrei usare una Lista, ma come già detto, quando poi mi serve l’oggetto devo ogni volta scorrere la lista.

Usiamo quindi una Mappa chiamata “**Identity Map**”. È una HashMap che uso per salvare gli oggetti che ho recuperato dal DB e creato. Quindi prima di creare un oggetto vado sempre a controllare se c’è già nella mappa.

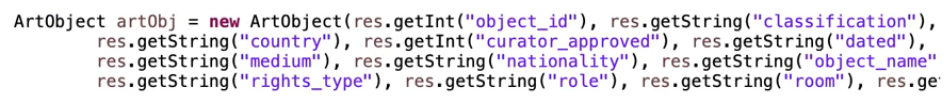
La chiave di questa mappa è di solita la chiave primaria della nostra tabella. Per gli oggetti, la loro chiave è l’id. La mappa avrà un Integer come chiave e un ArtObject come valore.

Nel costruttore del Model andrò ad istanziare questa mappa che ora è vuota. Infine, sempre nel costruttore, istanzio anche il grafo. Il codice è:



Avremo, successivamente, un metodo che ci permette di creare il grafo. Logicamente, questo metodo dovrà essere agganciato al controllore (in particolare al Bottone “Analizza Oggetti”).

La creazione del grafo segue un pattern ben preciso (aggiungo i vertici e aggiungo gli archi). Per fare questo mi serve sicuramente un riferimento al DAO. Noi sappiamo che la classe ArtsmiaDAO ci fornisce già come metodo la lista di tutti gli oggetti. In questo metodo è presente un “new” che mi crea ogni volta un oggetto:



Noi però vogliamo usare l’Identity Map, quindi useremo la new solamente se l’oggetto non esiste ancora. Il modo per poter fare questo è modificare il progetto base (solo in questo caso). **Devo passare la mappa ad ogni metodo del DAO**.



Successivamente, modifico la “new” (immagine precedente) chiedendo se nella mappa è già presente l’oggetto che devo aggiungere (uso l’id come chiave). Se poi mi serve che il metodo restituisca anche una lista, posso anche aggiungere l’oggetto alla lista “result”. Nell’else, non aggiungo un nuovo oggetto ma aggiungo quello che recupero dalla mappa. Quindi il codice sarà il seguente:

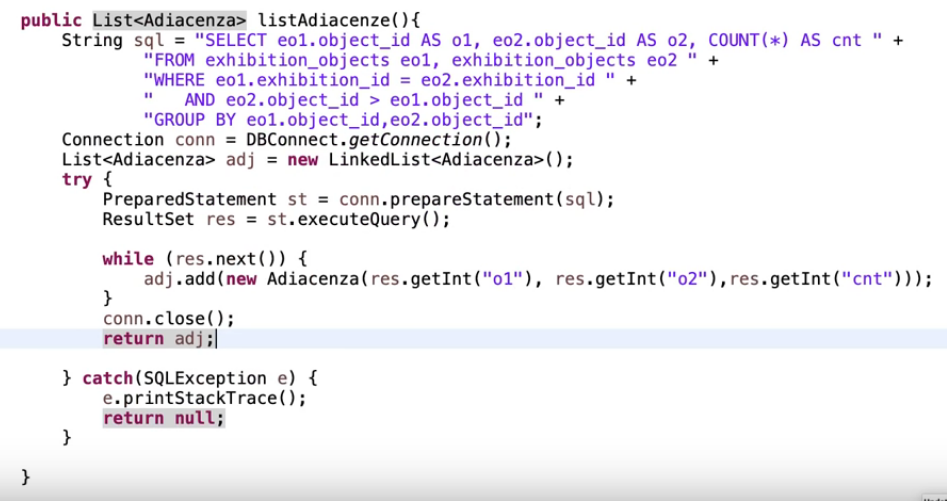


Ora possiamo tornare nel Model e posso richiamare il metodo del DAO passandogli la mappa.   
Aggiungiamo i vertici con il metodo “addAllVertices” in cui specifico il grafo a cui voglio aggiungere il vertice e la collection di vertici (la mappa).

Come ultima cosa, aggiungiamo gli archi (abbiamo deciso di usare il metodo 3 che fa fare tutto al DB). Usando il 3 metodo, abbiamo bisogno di una classe in più per poter salvare la lista di triple ottenute dal Database (oggetto 1, oggetto 2, peso). Nel package Model vado a creare una classe temporanea, chiamata “adiacenza”, che avrà 2 id e un peso. In questa classe, creiamo il costruttore e i vari Getter e Setter. L’unico scopo che ha questa classe è quella di recuperare i dati che la query ci ritorna.

Nel DAO suppongo di avere un metodo “public” che mi ritorna una lista di “Adiacenza”. Qui non mi serve la mappa perché ho supposto che “Adiacenza” sia basata solo sugli id. In questa classe come prima cosa definiamo la stringa sql (copiandola da Heidi e ricordandoci di togliere gli \n e sostituirli con uno spazio).

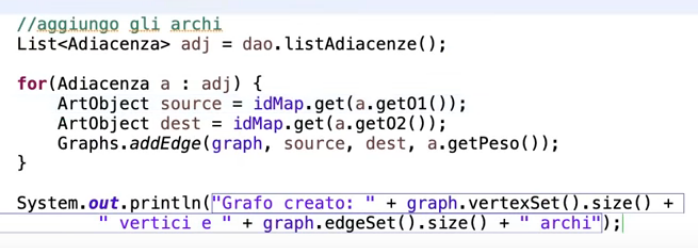
Dopodiché creo la connessione, creo la lista di “Adiacenza” (inizialmente vuota), creo il PreparedStatement, eseguo la query e scorro il ResultSet con il while. In questo while, per ogni riga che conterrà 3 valori, creo una nuova “Adiacenza” e la aggiungo alla mia lista. Come possiamo vedere, non abbiamo creato nessun oggetto Object quindi non mi serve l’Identity Map. Come ultima cosa, chiudo la connessione e ritorno la lista.



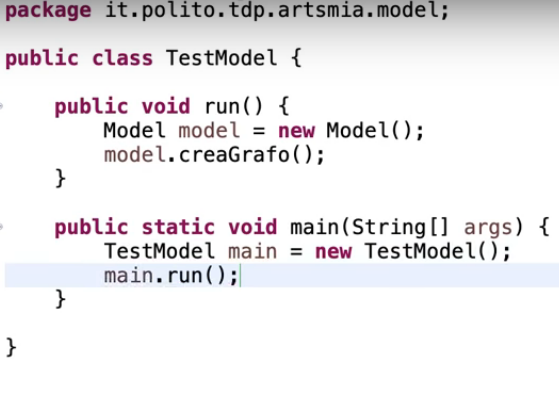
Terminato questo metodo nel DAO, ritorno al Model e recupero questa lista di Adiancenze. Fatto ciò, scorro queste adiacenze con un for e, per ogni coppia, creo l’arco. Notiamo che qui ci serve l’Identity Map perché io nell’Adiacenza ho salvato l’id non l’oggetto! Devo avere quindi il modo di recuperare il vertice sorgente e il vertice destinazione di questa Adiacenza.

Nell’altra lezione, in questo caso, avremmo creato un oggetto “ArtObject” solo con l’id e gli altri campi nulli. Ora, invece, abbiamo la mappa e possiamo recuperare l’unico oggetto che abbiamo creato che ha quell’id (un po' difficile da capire). Dopo aver recuperato sorgente e destinazione, posso andare ad aggiunger l’arco.

Come ultima cosa stampiamo il grafo (visualizziamo solo il numero di vertici e archi aggiunti).

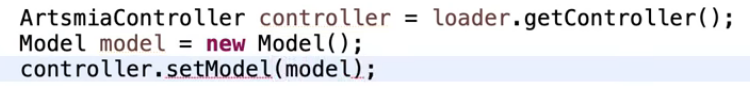


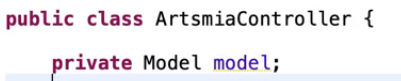
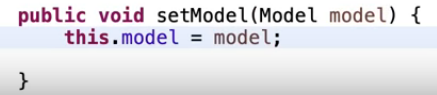
Proviamo il programma con una classe TestModel con il Main. In questa classe definisco il metodo run() che ha come obiettivo quello di creare un’istanza del Modello e di creare il grafo.



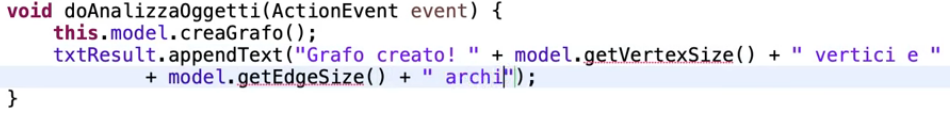
Avviando il programma notiamo che c’è qualche problema con il DB in quanto “hanno cancellato qualche riga dalla tabella ed è rimasto il riferimento nell’altra tabella”. Ci sono degli id nella tabella ExhibitionObject che però non corrispondono più a nessun oggetto (è stato eliminato). Bisogna catturare questa eccezione (try-catch) nel Model nella parte di codice in cui aggiungiamo un arco. In qualunque caso, è un problema del Database e non del nostro programma quindi tutto va bene.

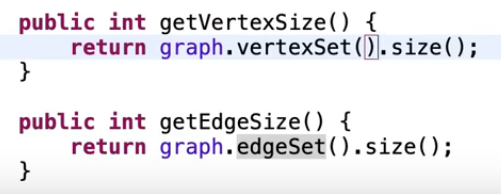
Come ultima cosa proviamo a collegare il programma al Controller. Andiamo quindi nel Main, ci creiamo il Modello e aggiungiamo il metodo setModel (ancora da creare nel controller). Andiamo quindi nel controller, creiamo il riferimento al nostro Model e creiamo il metodo setModel.



Infine, gestiamo i bottoni.Nel metodo “doAnalizzaOggetti” aggiungiamo il metodo creaGrafo e inseriamo un messaggio che ci conferma che il grafo è stato creato e che ci indica il numero di vertici e di archi (ce lo facciamo dire dal Model). Usiamo per questo due metodi getVertexSize e getEdgeSize (ancora da creare nel model). Essi hanno come scopo solo la restituzione delle delle grandezze che ci servono. Facciamo questo perché il nostro grafo è privato nella classe Model.



 Questi due metodi sono aggiunti nel Model!

Opzione numero 3 funziona sempre! FINE.