Data Base Libri

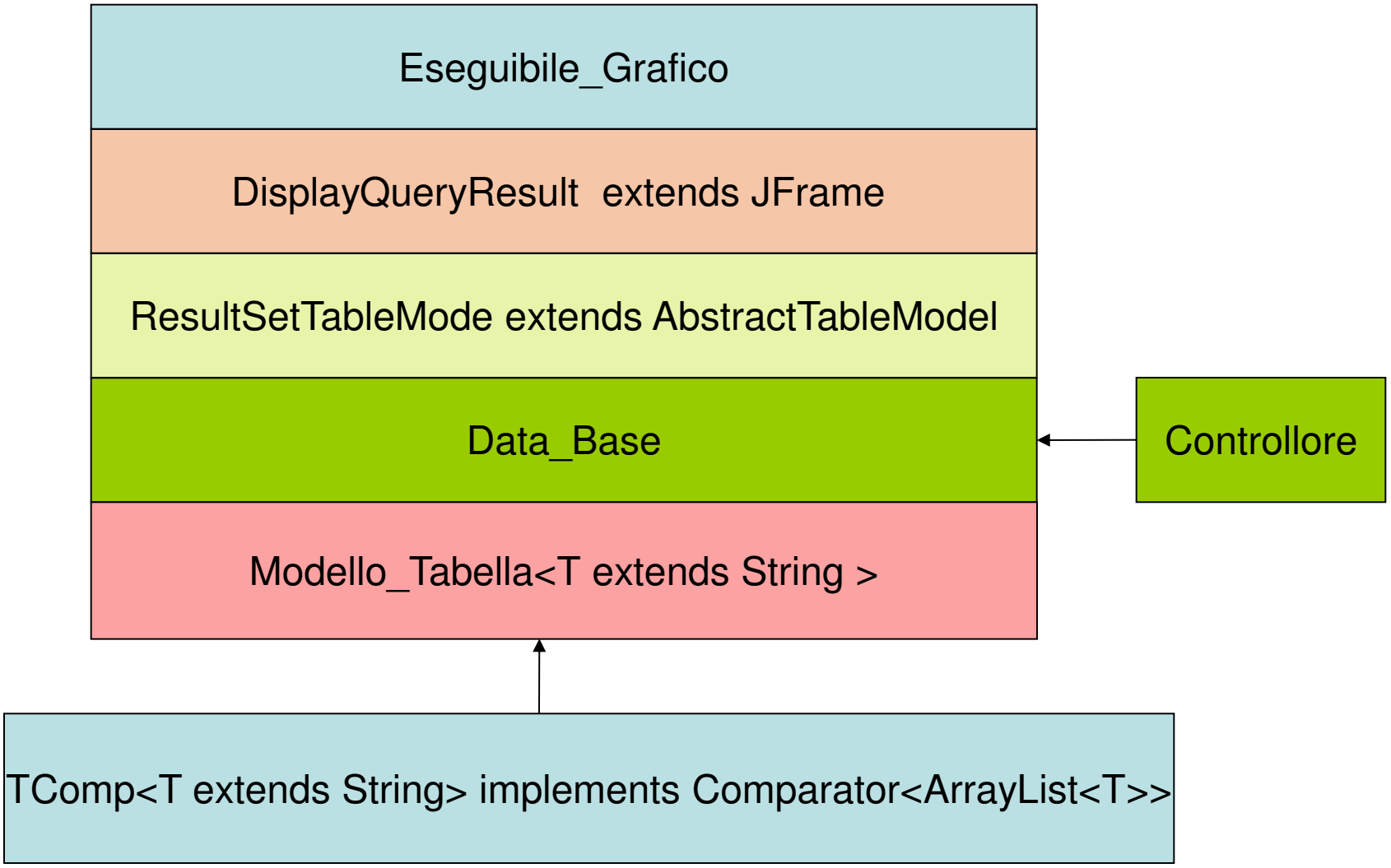
Autore: Campello Davide

Anno: 2006-2007

Linguaggio: Java 6

Presentazione del funzionamento del database e dei parametri del motore

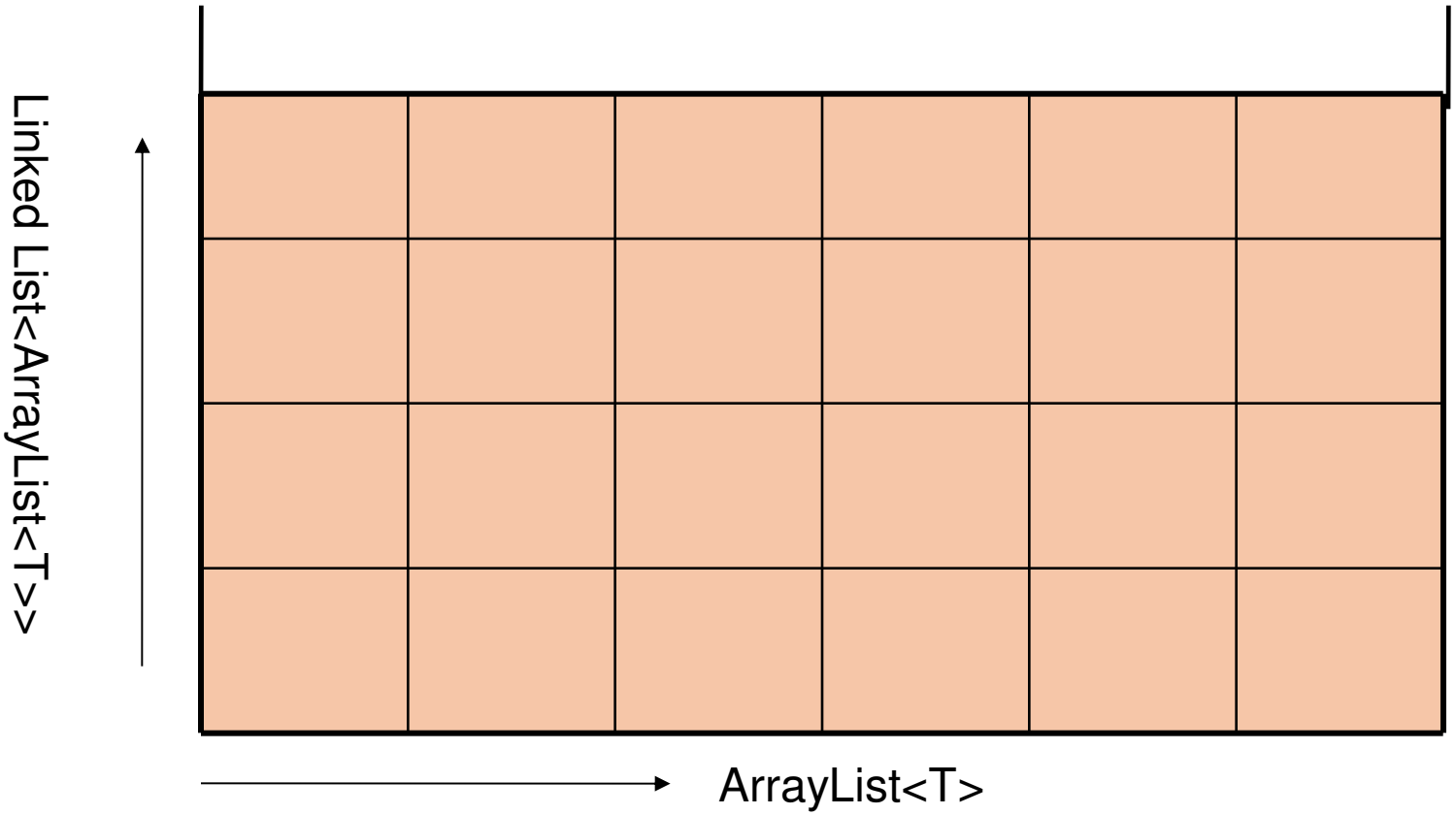
La struttura del programma



Come si vede dal disegno precedente la classe più “profonda” è la classe Modello_Tabella. Questa classe ha,tra l’altro, la responsabilita di ordinare la tabella seguendo l’ordine definito nella classe TComp (che come si vede implementa Comparator). La struttura di Modello_Tabella è quella di una

LinkedList<ArrayList<T>>

Questo significa che ogni riga è formata da un ArrayList<T> , mentre più righe entrano in una LinkedList



La classe Data_base è formata da quattro classi Modello_Tabella.

1) Tabella Autori

2) Tabella ISBN

3) Tabella Titoli

4) Tabella Editori

E da una LinkedList<ArrayList<T>> result (con codice -1). In quest'ultima lista vengono inseriti le tabelle risolte. Da notare che questa tabella non è gestita dal programmatore SQL ma dal software del programma, e più precisamente dalla classe Controllore.

Come funziona

Il programma implementa (una parte del) il linguaggio SQL. L'utente digita un comando che viene poi interpretato su più livelli. Più esattamente le classi che analizzano il comando digitato dall'utente ed agiscono di conseguenza sono:

DisplayQueryResult (Al livello più alto): oltre a gestire la parte Grafica (estende infatti JFrame) questa tabella passa le parti del comando che servono alla classe sottostante, e identifica quale tabella sarà utilizzata (la Tabella uno indica gli autori, la tabella due gli autori_ISBN, ecc.).

ResultSetTableMode: questa classe è solamante una classe di passaggio che estende la classe astratta AbstractTableModel. Mi costringe quindi a implementare i metodi `getValueAt(int row, int colonna); getColumnCount(); getRowCount(); getColumnCount(); getColumnName(int colonna); getColumnClass()`. Questi metodi richiamano semplicemente i metodi omonomi della classe sottostante (Data_Base).

Data_Base: in questa classe e in quella sottostante viene svolto il grosso del lavoro. In questa classe vengono implementati i metodi sovrascritti della classe AbstractTableModel e parecchi metodi di utilità (e quindi private che servono alla gestione della classe stessa). Da notare l'uso di una finestra_vista che è un clone profondo delle varie classi Modello_Tabella. Questo scelta è legata alla sicurezza. Ciò che vedo non modifica il codice delle classi visualizzate. A questo livello il comando digitalizzato dall'utente viene analizzato per portare alla vista ciò che è richiesto

Modello_Tabella Questa classe gestisce ogni singola tabella (compresa la Lista result). Da notare l'altissimo uso di metodi d'utilità (metodi private) che gestiscono le azioni che vengono compiute da questa classe. L'interfaccia pubblica (a parte i primi cinque metodi che sono usati solo nelle versioni di prova della classe, e che potrebbero essere quindi eliminati nella versione grafica definitiva) si limita ai metodi che vengono chiamati dalla classe Data_Base e che, tra l'altro, si occupano dell'ordinamento ascendente e discendente delle righe. In particolare il metodo pubblico `select_tabella_criterio_ASC(LinkedList<ArrayList<T>> l,int... criterio)` chiama il metodo di utilità `select_tabella_ASC(LinkedList<ArrayList<T>> da_ordinare)`. Tale metodo tramite un albero (classe TreeSet) che accetta un parametro della classe TComp<T> che implementa l'interfaccia Comparator fissa il tipo di ordine da utilizzare nell'ordinamento. L'ordine usato è quello lessigrafico rispetto all'array di interi "criterio". (Se l'elemento della riga corrispondente all'elemento 0 dell'array è uguale l'ordine è deciso dal secondo elemento e così via). A questo livello del codice il comando è gia stato risolto e ci si occupa solo di eseguire ciò che il comando richiede e sulla tabella richiesta. A questo livello vengono gestiti tutti i cambiamenti della tabella tramite metodi privati (in realtà la parte difficile del codice è contenuta qui e nella classe Contollore).

La classe DisplayQueryResult

Questa è la classe che gestisce la parte grafica, estende infatti JFrame. Tra i suoi campi privati c'è un riferimento ad una classe ResultSetTableModel che viene inizializzata dal costruttore costruendo l'oggetto tableModel.

L'unico costruttore, senza parametri, gestisce la parte grafica e tramite una classe anonima, raccogli i comandi inseriti dall'utente ad ogni click del bottone. In questa classe anonima il comando viene diviso in token e raccolto nel membro privato query (che è un oggetto di tipo LinkedList<String>). Tale membro è poi passato come primo parametro al metodo `setQuery(LinkedList<String> c,int n)` chiamato con l'oggetto tableModel. Il secondo parametro di tale metodo fissa invece la tabella da visualizzare. In questo modo i token del comando e la tabella da visualizzare sono passati alla classe sottostante cioè all'oggetto tableModel (di tipo ResultSetTableModel). Si noti che la tabella da usare è contenuta nel campo private di tipo int Tabella, ed è determinata dal metodo private `fissaTabella(String s)`, dove la stringa s è la parte del comando SQL che fissa la tabella da usare e che viene quindi risolta a questo livello, non è passata alle classi sottostanti, che dovranno perciò gestire solamente i comandi veri e propri.

La tabella da usare e il primo comando sono fissati all'apertura del programma dal costruttore della tableModel, poi tali valori sono determinati ad ogni click del bottone dal metodo `setQuery` della tableModel. All'avvio del programma la tabella passata per default è quella autori (cioè la 1) e il comando è "Select * Authors".

La classe ResultSetTableMode

Questa classe estende AbstractTableModel. Deve quindi implementare i metodi astratti di questa classe. L'implementazione di tali metodi è molto semplice. Tali metodi richiamano i metodi corrispondenti della classe sottostante Data_Base a cui è delegata la responsabilità di compiere il vero lavoro. Oltre a questi metodi la classe ResultSetTAbleMode definisce anche il metodo pubblico `setQuery(LinkedList<String>s,int n)` che fa da ponte tra la classe DisplayQueryResult e la classe Data_base. La classe aggrega infatti un oggetto result_set di tipo Data_Base.Il metodo setQuery si limita a chiamare il metodo `esegui_query(LinkedList<String> q, int n)`

Il costruttore della classe ha due parametri: l'intero m che gli viene passato dalla classe soprastante e che viene poi passato al (primo parametro del) costruttore del membro privato result_set , che fissa la tabella da visualizzare; e la LinkedList s che viene sempre passata al (secondo parametro del) costruttore del membro result_set e che rappresenta i token del comando che verranno gestiti dall'oggetto result_set. La tabella da usare e il primo comando sono fissati dal costruttore della classe solo all'apertura del programma, poi tali valori sono determinati ad ogni click del bottone dal metodo setQuery.

L’unico motivo per cui si introduce tale classe e non si usa direttamente la classe Data_Base è la presenza del metodo fireTableStructureChanged() definito nella classe AbstractTableModel. Tale metodo notifica a qualsiasi JTable che usa Oggetti di tipo ResultSetTableMode come modello, che la struttura del modello è cambiata. Il motivo per cui tale metodo non è usato direttamente in Data_Base è legato al fatto che seguendo le direttive della programmazione orientata agli oggetti, ogni classe deve avere poche e ben specifiche responsabilità. In questo senso la responsabilità della classe ResultSetTableMode è quella di notificare alla classe superiore ogni cambiamento avvenuto ai vari Modello_Tabella.

La classe Contollore

Questa è la classe più difficile, è il controllore software di tutto il database.

Questa classe ha un unico costruttore che accetta 5 parametri di tipo LinkedList<ArrayLis<T>> i primi quattro inizializzano i quattro membri private della classe che rappresentano gli autori, gli autori_ISBN, i titoli e gli editori, l' ultimo inizializza la Lista che rappresenta result, cioè la lista dove metto i valori risolti. Quest'ultima lista è comune a tutte le altre quattro.

è formata da due metodi che gestiscono ripettivamente

Il Modello_Tabella: autori,autori_isbn,titoli e la Lista result

Il Modello_Tabella: titoli,editori e la Lista result

Più precisamente il primo metodo cerca nei Modelli_tabella autori e titoli confrontando l'ID autore e il codice ISBN, con ciò che è contenuto in autori_isbn e se c'è corrispondenza questo metodo aggiorna la lista result aggiungendo la riga autori e titoli (controllando anche che tale riga non sia già inserita). Inoltre toglie anche gli elementi matched dai rispettivi Modelli_Tabella. Anche se nella vista questi autori e titoli si vedono nelle rispettive tabelle, in realtà sono stati eliminati (la vista aggiunge infatti anche gli autori e i titoli in result).

Il secondo metodo lavora come il primo solo che controlla i Modelli_Tabella titoli ed editori, e controlla la lista result con il Modello_Tabella editori per trovare corrispondenze tra l'ID editori in titoli o in result e l'ID editori in editori. Se trova corrispondenza sostituisce (in titoli o in result) all'ID editore il suo nome completo. Questo metodo NON toglie l'editore matched dal Modello_Tabella editori.

La classe Data_Base

Questa è la classe che gestisce tutte le classi Modello_Tabella, la Lista result e la classe Contollore.

Ha un unico costruttore che accetta un intero, che determina la tabella a cui passare il comando i cui token sono inseriti in una LinkedList<String>. Il costruttore inizializza i vari Modello_tabella che sono membri privati della classe. Si ricordi che la LinkedList di ArrayList<T> result è comune a tutti i Modelli_Tabella. Il costruttore inizializza inoltre il Contollore che gestisce tutti i modelli tabella e la LinkedList result. Il membro privato int quale_tabella mi evidenzia volta per volta qual'è la tabella attiva che deve eseguire il comando. Per default la finestra_vista viene indirizzata sul Modello_Tabella autori.

È questa la classe che gestisce i comandi SQL e che controlla la classe Contollore.

I comandi SQL sono gestiti in questa classe che li passa alla classe sottostante Modello_Tabella che li implementa a più basso livello.

Questa classe usa una finestra_vista (private) che è un clone profondo della tabella che deve essere visualizzata, le modifiche effettive sulla tabella sono lasciate alla classe Modello_Tabella. In questo senso la classe Data_Base è solo una classe che gestisce le varie tabelle, la Lista result e la classe Contollore, per le modifiche effettive da fare sulle singole tabelle la responsabilità è delegata alla classe Modello_Tabella. Quindi la classe Data_Base interpreta il comando SQL chiamando poi il corrispettivo metodo nella classe Modello_Tabella.

É interessante notare che gli unici metodi pubblici di questa classe (le API) sono quelli che gli vengono passati dalla classe soprastante (cioè la classe ResultSetTableMode extends AbstractTableModel). Cinque di questi sei metodi sono quelli che implementano la classe astratta AbstractTableModel, l'altro è il metodo `esegui_query(LinkedList<String> arg, int n)` che serve a passare il comando digitato dall'utente, la tabella da usare, gestisce la classe Contollore e inizializza la finestra_vista. I token del comando sono contenuti in una LinkedList di Stringhe privata Comando. I parametri passati a tale metodo evidenziano che esso inizializza il riferimento Comando al riferimento arg e fissa la tabella da usare, inizializzando il membro private quale_tabella. Vengono poi chiamati i vari metodi private che eseguono effettivamente il comando chiamando i metodi sottostanti della classe Modello_Tabella. Poichè le variabili quale_tabella e Comando sono già inizializzate i metodi "sanno" quale comando eseguire (perchè Comando è inizializzato) e su quale tabella (perchè quale_tabella è inizializzata).

Tutti gli altri metodi sono metodi di utilità (e quindi private).

La classe Modello_Tabella

Questa è la classe che gestisce ogni singola tabella e la lista result. Mentre la classe Data_Base conosce con che tipo di tabella abbiamo a che fare (autori, editori, autori_isbn o editori) questa classe mette a disposizione metodi (pubblici) diversi a seconda della tabella considerata. Per esempio per inserire dati nella tabella autori la classe Data_Base chiamerà il metodo `inserisciAutori()` del Modello_Tabella autori. Questi metodi di inserimento gestiscono anche l'assegnamento degli ID sia per la tabella autori che quella editori.

La classe ha un unico costruttore che inizializza la tabella che stiamo usando tramite il riferimento myList ed inizializza la lista result. Si ricordi che result è comune a tutte le tabelle (come si vede dal costruttore della classe Data_Base). Vengono inoltre inizializzati gli ArrayList<T> che contengono gli ID degli editori e degli autori. Evidentemente all'inizio questi ArrayList<T> sono vuoti. L'unico metodo che riempie questi Array è il metodo `cancella,` che eliminando autori ed editori deve inserire il loro ID che sarà poi a disposizione per futuri inserimenti.

Il metodo `cancella(int caso, LinkedList<T> dato)` gestisce all'interno la scelta della tabella (tramite il parametro intero caso). Il parametro dato è un ulteriore controllo sul comando fatto a livello più basso. Quando un elemento di una tabella viene cancellato il metodo l'ID di quell'elemento (se la tabella è autori o editori) e lo tiene memorizzato in un campo private (un ArrayList<T> IDAutore_da_assegnare nel caso di autori e IDEditore_da_assegnare nel

caso di editori). Per visualizzare le tabelle questa classe mette a disposizione due metodi pubblici select_colonne_tabelle(int caso, int... colonne), select_tabella(int caso). Il primo metodo mi permette di visualizzare solo alcune parti della tabella (la tabella è scelta in base al parametro caso mentre le parti sono scelte mediante l'array colonne). Il secondo metodo semplicemente visualizza tutta la tabella (scelta in base al parametro caso). Questi metodi sono chiamati dal metodo esegui_query della classe soprastante Data_Base. Questo metodo tramite il metodo private inizializza_finestra vista() sceglie la tabella da visualizzare e chiama il metodo select_tabella e gli passa il numero della della tabella da visualizzare. Da notare la doppia verifica. Infatti nel caso della tabella autori il metodo è chiamato dal Modello_Tabella autori e il parametro caso è fissato a 1 (che è il numero degli autori). Questa scelta è obbligatoria. Infatti la classe Modello_Tabella deve conoscere che tabella visualizzare, ogni tabella ha infatti un numero di colonne e rispettive intestazioni diverse.

In questa classe è definito inoltre il metodo select_tabella_ASC(LinkedList<ArrayList<T>> da_ordinare) che accetta un parametro di tipo LinkedList<ArrayList<T>> e ne ordina le riga tramite un albero che accetta come parametro un oggetto di tipo TComp<T> che implementa il tipo di ordine che verrà seguito dall'albero.

Nella classe Modello_Tabella c'è inoltre il metodo cancella(int caso, LinkedList<T> dati) che gestisce a basso livello il comando delete() della classe Data_Base. Tra le altre cose questo metodo gestisce gli ID degli autori e degli editori, inserendo nei rispettivi ArrayList<T> gli ID degli autori o editori eliminati

Il Modello_Tabella Autori

Colonna	Descrizione
`IDAutore`	Il numero che identifica unicamente l'autore nel database. Nel database questo numero è *autoincrementato* automaticamente. Per ogni riga inserita in questa tabella la classe Data_Base automaticamente incrementa l'ID dell'autore per assicurare che ciascun autore abbia il proprio ID. Questa colonna rappresenta la chiave primaria del Modello_Tabella Autori.
`Nome`	Il nome dell'autore(una string).
`Cognome`	Il cognome della'Autore (a string).

IDAUtore	Nome	Cognome
1	Harvey	Deitel
2	Paul	Deitel
3	Tem	Nieto
4	Sean	Santry

Il Modello_Tabella Autori_ISBN

Colonna	Descrizione
IDAutore	L'ID autore, si collega al Modello_Tabella autori
isbn	L'ISBN per un libro, si collega al Modello Tabella titoli.

IDAutore	isbn	IDAutore	isbn
1	0130895725	2	0139163050
2	0130895725	3	0130829293
2	0132261197	3	0130284173
2	0130895717	3	0130284181
2	0135289106	4	0130895601
.			.

Il Modello_Tabella Titoli

Column	Description
`isbn`	ISBN del libro (una string). La chiave primaria del Modello_Tabella titoli.
`titolo`	Titolo del libro (una string).
`NumeroEdizione`	Numero di edizione del libro (an intero).
`copyright`	Anno di copyright del libro (una string).
`IDEditore`	IDEditore (un intero). Si collega al Modello_Tabella editori
`prezzo`	Prezzo del libro (un numero reale)

Il Modello_Tabella editori

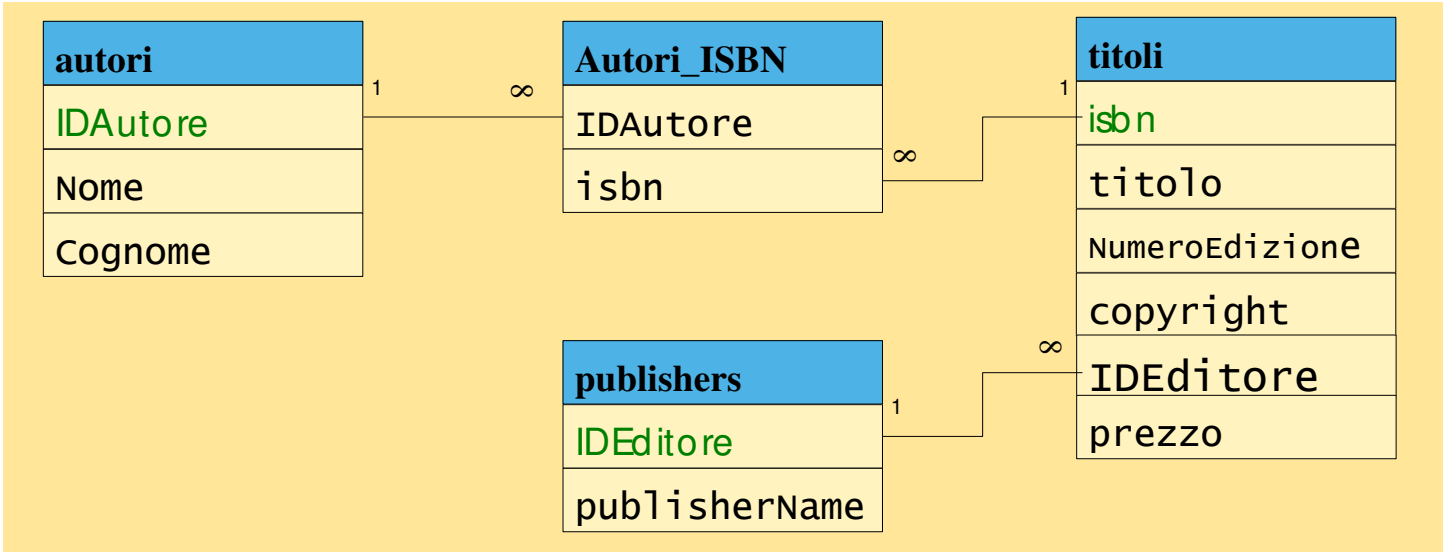
Colonna	Descrizione
`IDEditore`	L'ID dell'editore nel database è gestito dalla classe Data_base che lo incrementa automaticamente. Questo intero incrementato automaticamente è la chiave primaria del Modello_Tabella editori
`NomeEditore`	Il nome dell'editore(una stringa).

`IDEditore`	**`NomeEditore`**
0	Prentice Hall
1	Prentice Hall PTG

La LinkedList<ArrayList<T>> result

ID Autore	Nome	Cognome	isbn	titolo	Numero Edizione	copy-right	IDEditore	prezzo
0	Paul	Dietel	0130895725	C How to Program	3	2001	1	74.95
0	Paul	Dietel	0130384747	C++ How to Program	4	2002	1	74.95
1	Harvey	Dietel	0130461342	Java Web Services for Experienced Programmers	1	2002	1	54.95

La relazione tra i Modelli_Tabella



23.4.1 Il comando base Select

La forma più semplice di una query Select

```
SELECT * FROM NomeTabella
```

```
SELECT * FROM authors
```

Seleziona campi specifici da una tabella

```
SELECT IDautore, cognome FROM authors
```

IDautore	cognome
1	Deitel
2	Deitel
3	Nieto
4	Santry

La clausola Order by

La clausola opzionale **ORDER BY**

SELECT * **FROM** *NomeTabella* **ORDER BY** *column* **ASC**

SELECT * **FROM** *NomeTabella* **ORDER BY** *column* **DESC** **(Non implementato ma facilmente ottenibile dal codice scritto)**

ORDER BY campi multipli

SELECT * **FROM** *NomeTabella* **ORDER BY** *column1 column2* **ASC**

Nel nostro dataBase non implementiamo il comando

SELECT *colonna_1 … colonna_n* **FROM** *NomeTabella* **ORDER BY** *colonna_n1…* colonna_nn **ASC (o DESC)**

Dove n1…nn è un sottoinsieme di 1…n.

Esempio della clausola ORDER BY

```
SELECT * FROM authors
    ORDER BY cognome ASC
```

IDAutore	Nome	Cognome
2	Paul	Deitel
1	Harvey	Deitel
3	Tem	Nieto
4	Sean	Santry

Il comando insert into

Inserisce una riga in una tabella

INSERT INTO *NomeTabella* **VALUES** '*value1*' … '*valueN*'

INSERT INTO authors **VALUES** 'Sue' 'Smith'

IDAutore	nome	cognome
1	Harvey	Deitel
2	Paul	Deitel
3	Tem	Nieto
4	Sean	Santry
5	Sue	Smith

Da notare che l'IDAutore di Sue Smith è stato assegnato automaticamente dalla classe Modello_Tabella

Il comando Update

Modifica i dati in una tabella

UPDATE FROM *NomeTabella*

VALUES *'vecchiovalore1'*
…'vecchiovaloren' 'nuovovalore1'… 'nuovovaloren'

UPDATE FROM authors

VALUES *'Sue' 'Smith' 'Sue' 'Jones'*

authorID	firstName	lastName
1	Harvey	Deitel
2	Paul	Deitel
3	Tem	Nieto
4	Sean	Santry
5	Sue	Jones

Da notare che l'IDAutore di Sue Jones è uguale a quello di Sue Smith e NON è controllabile dal programmatore SQL

Il comando DELETE

Rimuove i dati da una tabella

DELETE FROM *NomeTabella* **VALUES** ‘nomedaeliminare_1’…’nomedaeliminare_n’

DELETE FROM autori **VALUES** ‘Sue’ ‘Jones’

authorID	firstName	lastName
1	Harvey	Deitel
2	Paul	Deitel
3	Tem	Nieto
4	Sean	Santry

Da notare che l’IDAutore di Sue Jones deve essere memorizzato in qualche lista in Modello_Tabella (più precisamente l’ArrayList<T> IDAutore_da_assegnare) per poter essere riutilizzato in futuri inserimenti e, come al solito, NON è a disposizione del programmatore SQL.

Il comando FIND

Cerca dati nella tabella

FIND INTO Nome_Tabella VALUES
"nomedacercare_1'…'nomedacercare_n'

FIND INTO autori **VALUES** 'Sue' 'Jones'
FIND INTO autori **VALUES** 'Sue' 'param::blank'

Per il comando FIND non è ancora stata scritta una tabella dei risultati. Questi appaiono nella riga di comando indicando la posizione del campo trovato
Il comando FIND consente di fare ricerche anche non specificando tutti i campi in modo da ricercare, ad esempio tutti gli autori con un certo nome ecc. Nel caso si voglia omettere un parametro sarà necessario scrivere param::blank

Note varie

L'interfaccia grafica supporta una modalità amministratore che consente di effettuare operazioni come la cancellazione del database.

Per il login da amministratore è necessaria una password che inizialmente è 0000. Una volta loggati è possibile cambiarla.