TECNICHE DI PROGRAMMAZIONE

25/03/19

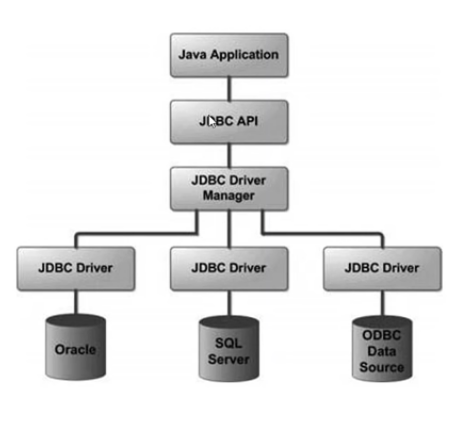
Introduzione al programma HeidiSQL. È un’interfaccia grafica comoda per la gestione del database. Apriamo HeidiSQL e creiamo un nuovo database con login “root” e pass “root”. I database si importano ed esportano in formato sql. Se andiamo nella pagina del corso, andiamo nei datasets e prendiamo “Dizionario” con formato sql. Quando importo un database, mi danno un file sql che contiene l’istruzione sql per ricreare sul mio database esattamente quel database. L’operazione è:

* File… Load SQL file…scelgo il file che ha la stessa immaginetta di HeidiSQlQL ( il file zippato preso dal datasets)

Una volta caricato il database lo runniamo con il tasto del “play” blu. Avremo quindi il nostro database, che potrà contenere delle tabelle, che a loro volta possono contenere dati ecc….

Introduzione a JDBC. Essa è una libreria che permette ad un programma Java di accedere ai dati all’interno di una qualunque basi di dati relazionale. Le classi che ci interessano sono all’interno di due package che sono java.sql e javax.sql . JDBC standardizza una parte dell’interazione tra programma e database.

JDBC è quindi la libreria che mi permette di collegarmi a qualunque database relazionale. La sua architettura è fatta su più livelli.



Affinchè l’applicazione possa comunicare con il database, vengono definite nello standard JDBC delle API che sono delle chiamate, dei metodi, delle classi, delle interfacce. La maggior parte di queste definite nello satndard JDBC sono interfacce. La classe che implementa queste interfacce la fornisce il fornitore del database all’interno di un pacchetto che si chiama JDBC Driver. Quindi la libreria JDBS dell’immagine sopra, la parte superiore è standardizzata (Java Application e JDBC API), mentre la parte sotto è specifica di ogni database.

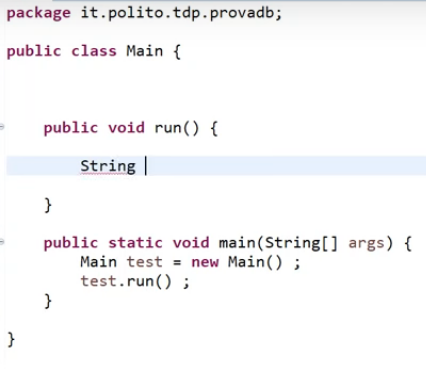
Ogni volta che vogliamo dal nostro programma leggere dei dati (nel nostro esempio potrebbe essere “Dimmi qual è la parola più lunga che hai nel dizionario”) dobbiamo seguire 6 passi che verranno specificati successivamente.

Il JDBC Driver può essere offerto da MySQL (MySQL Connector/J ).

Per aprire una connessione con il database dobbiamo chiedere alla libreria JDBC di analizzare il contenuto della stringa che descrive la connessione, caricare il Driver JDBC opportuno e poi chiedere al Driver di aprire una connessione.

Una connessione è un “Tubo” logico che collega il mio programma con il Database. In questo tubo possono viaggiare avanti e indietro i dati e le query. La connessione rimarrà aperta fino a quando non decidiamo di chiuderla.

Ogni volta che voglio provare una classe, definisco un metodo normale (non statico, fuori dal Main ) e il Main non farà altro che creare un istanza di se stesso e chiamare il metodo su se stesso. Non ci piace lavorare nel Main perché usa solo metodi e variabili statici.





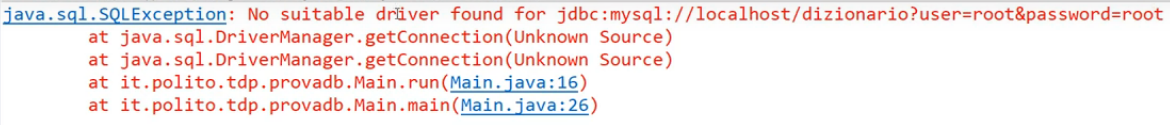
Questa stringa, mi permette di accedere ad database che abbiamo creato in HeidiSQL (dizionario).

Cosa me ne faccio? La uso chiedendo al Driver Manager di darmi una Connection.



Scritto così (nomedellaclasse.metodo) questo è un metodo statico della classe DriverManager. Questo perché non abbiamo creato nessun nuovo oggetto, non abbiamo nessun new. Ci pensa questo metodo statico a fare la connessione (metodo di Factory).

Connection è un interfaccia non è una classe!! (se passo il cursore sopra c’è una I blu).

Il metodo getConnection può creare un’ eccezione di tipo SQLException. Gestiamo questo problema con un try catch. Se runniamo il programma, Java ci dà un errore del tipo:

Questo perché il Driver Manager va a chiedere in tutte le classi che trova, in tutti i package, in tutte le librerie che abbiamo configurato, quali sono le classi che implementano l’interfaccia “Java.SQL.Driver”. A tutte le classi che trova che implementano quell’interfaccia, gli chiede se gestisce il database di tipo MySQL. Il Driver Manager ha cercato nel patch di ricerca di Java e non ha trovato nessun classe che gestiva le connessioni con database di tipo MySQL. Eppure noi l’abbiamo scaricata. Il problema è che non è accessibile a questo progetto Java, è in una cartella mia salvata da qualche parte.

Possiamo modificare questo se andiamo nelle proprietà dell’oggetto(Java Build Patch/Libraries/Add External Jars). Facendo così però non va molto bene. Decidiamo quindi di creare proprio una cartella all’interno del progetto (una copia) e dentro di questa cartella ci copio una copia del Jar di MySQL. Cosi facendo ogni progetto ha una copia di quel file. Ora prendo il JAR scaricato e lo trascino all’interno della cartella appena creata. Copy files. Se faccio ora il Run , non lo trova lo stesso… perché? Perché ho solo copiato un file all’interno del progetto. Devo dire ad Eclipse che quel file è una libreria. Per fare questo premo il tasto destro sulla libreria nella cartella appena creata e vado su Build Path. Add to Build Path. Ora se faccio Run non ho più errori ma il programma non fa niente.

Quindi cosa abbiamo fatto fino ad adesso?

1. Dalla mia Applicazione abbiamo chiamato sulla classe Driver Manager il metodo getConnection.
2. Il Driver Manager ha caricato una classe Driver.
3. La classe Driver crea un oggetto di tipo Connection.
4. L’oggetto Connection viene restituito all’applicazione.

Ora quindi l’applicazione ha finalmente un oggetto di tipo Connection (tubo di cui parlavamo prima).

Ora in questo tubo io posso creare una query e mandarla al database.

**APRIRE LA CONNESSIONE ERA IL PASSO 1 DEI 6 PASSI DI CUI PARLAVAMO PRIMA.**

Il passo 6 è quello di chiudere la connessione con il metodo connection.close();



Su HeidiSQL nel tab Processi possiamo vedere quante sono le connessioni attive. Prima o poi le connessioni scadono, ma scadono quando muore il Soket ( un timeout di 10/15/20 minuti dopo il quale la connessione si chiude).

Quindi tra il codice in cui apro la connessione e quello in cui la chiudo, io posso interagire con il mio database. Come?

Le istruzioni vengono inviate e ritornano con un meccanismo di navette ( STATEMENT ).

Quindi io devo creare una navetta, un contenitore, all’interno di questo contenitore metterci l’istruzione e poi dire al mio Driver JDBC di mandare l’istruzione al database. Successivamente il database esegue l’istruzione e mi manda indietro il risultato. Sulla stessa connessione possono esserci tante navette, anche attivi contemporaneamente. Per creare lo statement faccio fare tutto al Factory mediante il codice:



Nel nostro caso avremo conn.createStatement() ;

Tutti questi codici che scriviamo non sono oggetti di tipo New!!

Dopo aver creato il nostro statement, usiamo il metodo ExecuteQuery che ci permette di mandare la query al database. Ora stiamo usando la classe Statement ma successivamente vedremo che è meglio usare la classe PrepareStatement.

Ora proviamo a scrivere una query su HeidiSQL come ad esempio la query che ci permette di estrarre dal nostro database tutte le parole.



Dove parola è la nostra tabella del database. Se aggiungiamo la condizione:

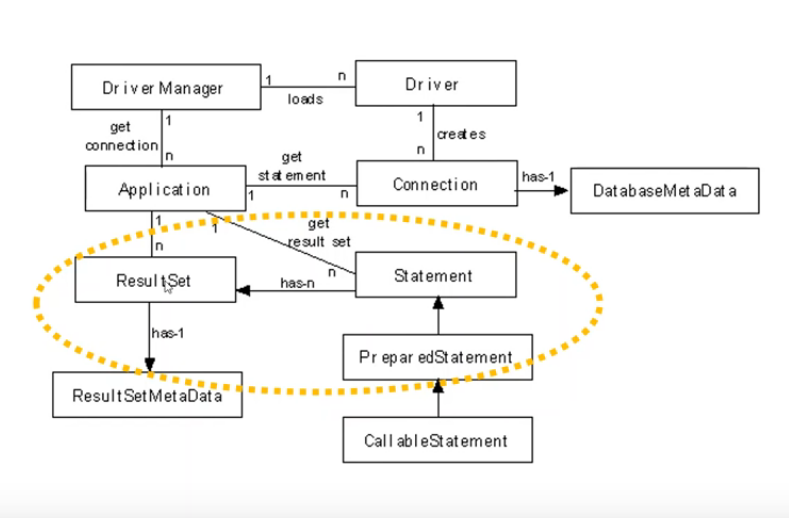


possiamo cercare le parole che iniziano con la lettera Z...

Ora questa query scritta su HeidiSQL la copiamo e incolliamo su Eclipse in una stringa chiamata sql, sotto forma di testo. Togliamo i segni di “a capo” su eclipse e ci ricordiamo di mettere lo spazio. Ora chiediamo allo statement di eseguirla. Questo codice restituirà indietro un risultato in un oggetto che si chiama ResulSet che è un’interfaccia. Quindi il codice finale è:



Quindi riprendendo il diagramma delle classi, avevamo l’applicazione che aveva ricevuto un riferimento all’oggetto Connection, su questo oggetto Connection abbiamo fatto il getStatement e questo oggetto ci ha dato uno Statement. A questo oggetto statement n oi applicazione abbiamo chiesto executeQuery e lo Statement ha creato un ResultSet e ci ha dato in riferimento l’applicazione che “contiene” il risultato della query.

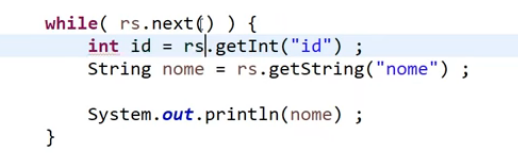


Il ResultSet implementa un cursore sulla query dei risultati ( quest’ultimo potrebbe contenere 4000 righe). Questo cursore si muove sulla tabella virtuale dei risultati. Muovendo questo cursore posso vedere una riga per volta della tabella. Il metodo ResultSet.next() ci permette di muovere il cursore sulla riga dopo. Quindi l’oggetto statement implementa due metodi: quello per muovere il cursore e quello per ottenere i dati delle varie colonne della sola riga su cui è posizionato il cursore in quel momento.

All’inizio il cursore è posizionato prima della prima riga. L’ultima posizione occupata dalla riga è invece quella dopo l’ultima riga. Sono comunque presenti altri metodi che mi permettono di andare avanti di più righe, tornare indietro ecc…

Il metodo next() è un metodo di tipo boolean che ci restituisce valore true se il cursore si è spostato su una riga successiva che esiste, falso se si è spostato dopo l’ultima riga perché non c’è piu nessuna riga.

Quindi l’idioma classico per poter muovere questo cursore è il seguente:



Leggo l’id chiedendo al ResultSet mediante un metodo che si chiama getInt() in quanto l’id è un int. Poi posso avere la parola e quindi chiedo al ResultSet di leggere la parola sotto forma di String.