JAVA – Database Connectivity JDBC

Metodi Avanzati di Programmazione

Laurea Triennale in Informatica

Università degli Studi di Bari Aldo Moro

Docente: Pierpaolo Basile

Introduzione

- Uno dei motivi di successo di Java è dovuto alla possibilità di sviluppare applicazioni client/server indipendenti dalla piattaforma
- L'indipendenza dalla piattaforma deve essere garantita anche per applicazioni che lavorano su basi di dati
  - Per questo è nato lo standard **Java Data Base Connectivity (JDBC)**
- Uno dei problemi principali con le basi di dati è la non compatibilità dei linguaggi, infatti pur basandosi sullo Structured Query Language (SQL-92), ogni database management system aggiunge o modifica qualcosa allo standard

Introduzione

- JDBC è progettato per essere platform-independent. Per permettere ciò JDBC fornisce un driver manager che gestisce dinamicamente tutti gli oggetti driver di cui hanno bisogno le interrogazioni a database
- Pertanto se si hanno tre diversi DBMS allora necessiteranno tre diversi tipi di oggetti driver
- Gli oggetti driver si registrano presso il driver manager al momento del caricamento
- Come tutte le API Java anche JDBC è stato progettato in modo da semplificare tutte le normali operazioni di interfacciamento con un database: connessione, creazione di tabelle, interrogazione e visualizzazione dei risultati

H2

- Per svolgere gli esercizi useremo il Database Engine H2 poiché può essere utilizzato in modo embedded senza necessità di installare un server *(la modalità server è comunque disponibile)*
- Implementa nativamente (100% Java) il protocollo JDBC
  - Gli esempi che forniremo funzioneranno con qualsiasi DBMS che fornisce un driver JDBC: minime differenze possono essere necessarie per la fase di connessione (stringa di connessione, connessione a un server remoto, …)

H2

- Per utilizzare H2 è necessario importare la libreria H2
- Utilizzando Maven è sufficiente inserire la seguente dipendenza nel `pom.xml`

```
<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <version>1.4.200</version>
</dependency>
```

H2 è disponibile all'indirizzo:
https://www.h2database.com/

Connessione ad un database

- Per aprire una connessione è necessario ottenere un oggetto di tipo Connection
  - l'oggetto Connection fornisce tutti i metodi per preparare le query SQL (ed altro)
- Per ottenere una connessione è necessario caricare il **driver** che implementa le API JDBC
  - ciò si ottiene semplicemente chiamando il metodo getConnection della classe **DriverManager** passando la stringa di connessione

Stringa di connessione

- Il metodo DriverManager.getConnection stabilisce una connessione ad un database. Questo metodo richiede una database URL, che dipende dal DBMS, per esempio:
  - MySQL: jdbc:mysql://localhost:3306/, dove localhost è il nome/indirizzo del server e 3306 la porta
  - Java DB: jdbc:derby:testdb;create=true, dove testdb è il nome del database al quale connettersi, e create=true indica al DBMS di creare il DB
  - **H2:** jdbc:h2:/home/user/test/db, dove /home/user/test/db è il file su file system che conterrà il DB
- Altri parametri come ad esempio username e password possono essere specificati attraverso un oggetto **Properties** passato al metodo getConnection insieme alla URL, o utilizzando altre modalità, vedere javadoc di getConnection

Esempi di connessione

```
//connessione senza parametri
Connection conn =
DriverManager.getConnection("jdbc:h2:/home/user/test/db");
//connessione con username e password
Connection conn =
DriverManager.getConnection("jdbc:h2:/home/user/test/db","us
er","1234");
//connessione con oggetto Properties
Properties dbprops = new Properties();
dbprops.setProperty("user", "user");
dbprops.setProperty("password", "1234");
Connection conn =
DriverManager.getConnection("jdbc:h2:/home/user/test/db",
dbprops);
```

SQLException

- Quando JDBC genera un errore durante le interrogazioni su un DB solleva un'eccezione di tipo **SQLException**. L'oggetto di tipo SQLException conterrà una serie di informazioni utili a capire l'errore
  - Una descrizione testuale dell'errore viene data dal metodo **getMessage**
  - **getSQLState** restituisce un codice alfanumerico codificato secondo lo standard ISO/ANSI e Open Group (X/Open)
  - **getErrorCode** restituisce un valore intero che indica un codice di errore specifico del driver che implementa JDBC

Statement

- Le query SQL si eseguono attraverso gli oggetti **Statement**
- Gli oggetti Statement si ottengono tramite l'oggetto Connection
  - È possibile ottenere anche degli statement preimpostati in cui è possibile sostituire a dei segnaposto inseriti nella query SQL dei valori
  - Le query preimpostate sono utili per inserire in maniera corretta all'interno della query dei lettarali applicando le opportune conversioni di tipo

Statement di modifica

- Spesso è necessario eseguire delle query che modificano il DB: creazioni di tabelle, aggiunta di tuple, modifica di tuple
- Queste query si eseguono utilizzando il metodo executeUpdate("SQL query") dell'oggetto Statement
- Gli Statement vanno sempre **chiusi** tramite il metodo close() per liberare risorse

Statement di modifica

```
public static final String CREATE_TABLE = "CREATE TABLE IF NOT
EXISTS store (artId INT PRIMARY KEY, desc VARCHAR(1024), price
DOUBLE, unit INTEGER)";
…
Connection conn =
DriverManager.getConnection("jdbc:h2:/home/user/db/store",
dbprops);
Statement stm = conn.createStatement();
stm.executeUpdate(CREATE_TABLE);
stm.close(); //chiudere lo statement!!!
```

Statement di modifica

```
stm = conn.createStatement();
stm.executeUpdate("INSERT INTO store VALUES(1,'pentola',4.5,20)");
stm.close();
```

Prepared statement

```
PreparedStatement pstm = conn.prepareStatement("INSERT INTO store
VALUES (?, ?, ?, ?)"); // ? è un segnaposto
pstm.setInt(1, 2); //l'indice parte da 1
pstm.setString(2, "piatto"); // i metodi set si occupano di
inserire i letterali nella query SQL
pstm.setDouble(3, 1.5);
pstm.setInt(4, 40);
pstm.executeUpdate();
pstm.close();
```

Statement di interrogazione

- Le query di selezione dei dati SELECT si effettuano sempre attraverso l'oggetto Statement utilizzando il metodo executeQuery("QUERY SQL")
  - È possibile utilizzare anche i PreparedStatement per le SELECT
- executeQuery restituisce un oggetto di tipo ResultSet che permette di navigare nelle tuple restituite (simile ad un iteratore)

Statement di interrogazione

```
Statement stm = conn.createStatement();
ResultSet rs = stm.executeQuery("SELECT artId, desc FROM
store WHERE unit>5");
while (rs.next()) {
    System.out.println(rs.getInt(1) + ": " +
     rs.getString(2));
}
rs.close();
stm.close();
```

Statement di interrogazione

```
PreparedStatement pstm = conn.prepareStatement("SELECT
artId, desc FROM store WHERE unit > ?");
pstm.setInt(1, 20);
rs = pstm.executeQuery();
while (rs.next()) {
    System.out.println(rs.getInt(1) + ": " +
    rs.getString(2));
}
rs.close();
stm.close();
```

Altro

- Attraverso il driver JDBC si possono effettuare tutte le operazioni disponibili su un DMBS
  - accedere ad informazioni relative allo schema del DB attraverso l'oggetto DatabaseMetaData restituito dal metodo getMetaData di Connection
  - commit() metodo di Connection
  - rollback() metodo di Connection


SAY JDBC
ONE MORE TIME