SAPIENZA
UNIVERSITÀ DI ROMA

Basi di Dati, Modulo 2

Sapienza Università di Roma
Facoltà di Ing. dell'Informazione, Informatica e Statistica
Laurea in Informatica
Prof. Toni Mancini, Prof. Federico Mari
http://tmancini.di.uniroma1.it
http://mari.di.uniroma1.it

Documento B.4 (D.B.4)

Basi di Dati Relazionali

Applicazioni 3-Tier in Java e JDBC

Versione 2016-02-03

1 Introduzione

In questa esercitazione studieremo brevemente come progettare applicazioni indipendenti che abbiano una connettività al database in Java. A tale scopo introdurremo il concetto generale, tipico dell'ingegneria del software, di *architettura multilivello* (in particolare a 3 livelli) [1] e la libreria *Java DataBase Connectivity* (JDBC) [2, 3] per realizzare tale architettura in Java.

2 Architettura a 3 livelli

In Ingegneria del Software, l'*architettura a 3 livelli* (*three-tier architecture*) è un *pattern*, specializzazione dell'architettura client-server, in cui l'interfaccia utente, le funzioni dell'applicazione e l'immagazzinamento dei dati sono sviluppati e mantenuti separati tra di loro.

I livelli in una applicazione 3-tier sono i seguenti (cfr. Figura 2.1):

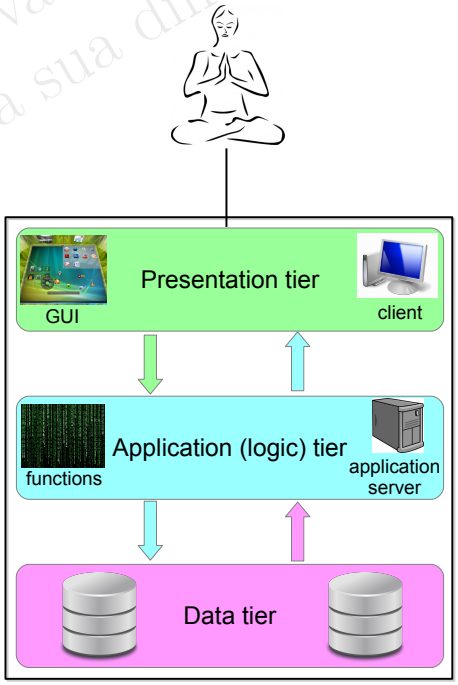

Figura 2.1: Schema di architettura a 3 livelli.

1. *Livello della presentazione* (presentation tier). Questo è il livello più in alto nell'architettura. Il suo scopo è servire da interfaccia tra l'utente e l'applicazione. Esempi per tale livello sono: graphical user interfaces (di programmi indipendenti, di applicazioni web, ...), clients in una architettura client server, ...

2. *Livello dell'applicazione* (application tier). Questo livello contiene le funzionalità dell'applicazione (dalle piccole funzioni centralizzate nei casi semplici ai veri e propri server di applicazioni distribuiti nei casi più complessi, ...). Esso può essere a sua volta suddiviso in più livelli, nel qual caso si parla di *architettura multilivello* o di *architettura a n livelli*. Il livello dell'applicazione raccoglie le richieste dal livello della presentazione, le elabora accedendo (se richiesto) al sottostante livello dei dati e restituisce i risultati della computazione al livello superiore.

3. *Livello dei dati* (data tier). Questo livello consiste nei DBMS. Caratteristica di questo livello è l'*indipendenza* dei dati rispetto alle applicazioni che vi si interfacciano.

Un ulteriore esempio di architettura a 3 livelli è rappresentato in Figura 2.2.

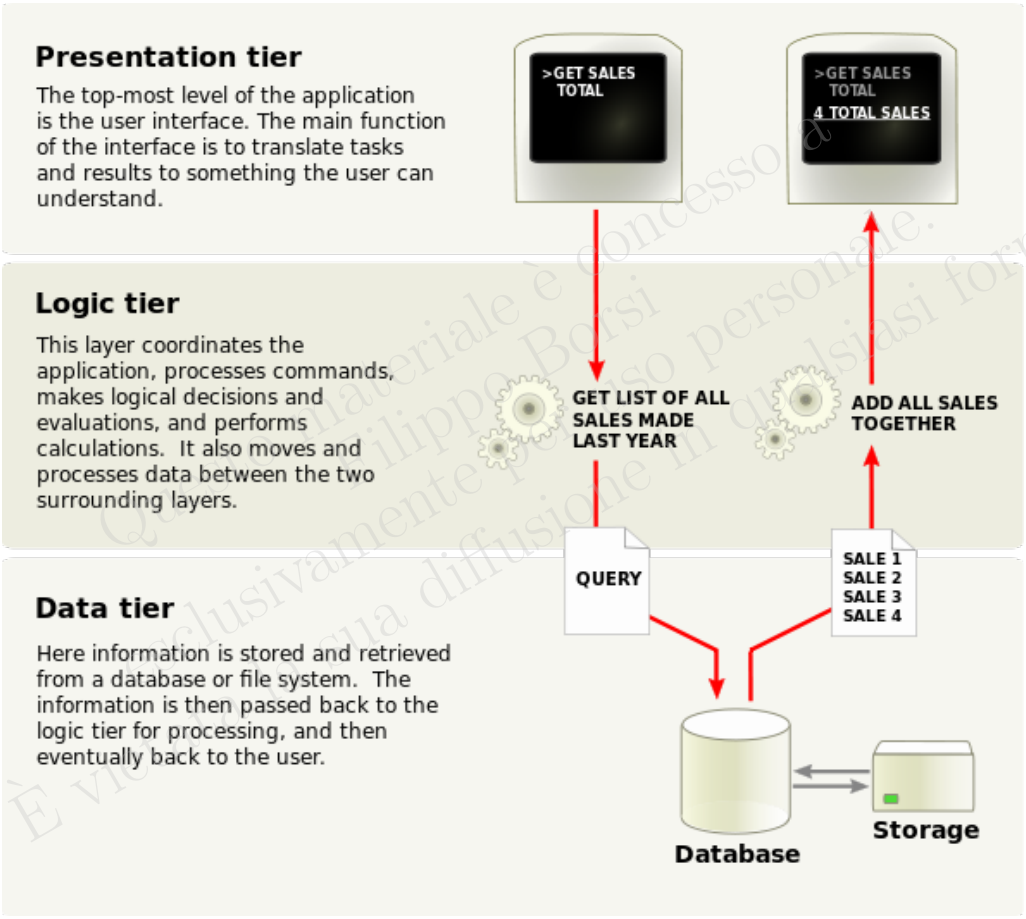

Figura 2.2: Esempio di architettura a 3 livelli.

3 JDBC: Java DataBase Connectivity

3.1 Cosa è JDBC

JDBC sta per Java Database Connectivity, che è uno standard Java *indipendente dal database* per la connessione tra Java e un'ampia gamma di database.

La libreria JDBC include funzionalità per ogni operazione comunemente associata all'utilizzo di database:

- Creare una connessione al database
- Creare comandi SQL
- Eseguire comandi SQL
- Vedere e modificare i risultati di tali interrogazioni

3.2 Creare un'applicazione JDBC

3.2.1 Importare i packages

```
// Step 1: Import required packages
import java.sql.*;
```

3.2.2 Registrare il driver JDBC

Al fine di creare una connessione con un DBMS, è necessario inizializzare il driver relativo a tale DBMS. Ogni DBMS ha un driver e una stringa di connessione dedicati. In questa esercitazione useremo PostgreSQL come DBMS. Il driver JDBC per PostgreSQL può essere scaricato seguendo questo indirizzo: http://jdbc.postgresql.org.

```
//STEP 2: Register JDBC driver
Class.forName("org.postgresql.Driver");
```

3.2.3 Aprire una connessione

Si deve usare il metodo `DriverManager.getConnection()` per creare un oggetto di tipo `Connection` rappresentante la connessione fisica con il DBMS, come segue:

```
//STEP 3: Open a connection
//  Database credentials
static final String DB_URL = "jdbc:postgresql://localhost:5432/
   accademia";
static final String USER   = "username";
static final String PASS   = "password";
System.out.println("Connecting to database...");
conn = DriverManager.getConnection(DB_URL, USER, PASS);
```

Si noti la forma della stringa `DB_URL`, diversa a seconda del DBMS utilizzato.

3.2.4 Eseguire un'interrogazione

Per costruire ed eseguire un'interrogazione si richiede l'utilizzo di un oggetto di tipo `Statement` o `PreparedStatement` come segue:

```
//STEP 4: Execute a query
System.out.println("Creating statement...");
stmt = conn.createStatement();
String sql;
sql = "SELECT id, nome, cognome, posizione FROM Persona";
ResultSet rs = stmt.executeQuery(sql);
```

Nel caso in cui l'interrogazione sia di tipo UPDATE, INSERT o DELETE il codice da utilizzare sarebbe il seguente:

```
//STEP 4: Execute a query
System.out.println("Creating statement...");
stmt = conn.createStatement();
String sql;
sql = "DELETE FROM Persona";
ResultSet rs = stmt.executeUpdate(sql);
```

L'utilizzo di un oggetto di tipo `PreparedStatement` richiede l'uso del carattere speciale ? (punto di domanda) nella stringa della query, come nel seguente esempio:

```
//STEP 4: Execute a query
// initialized incr' and cofName' somewhere in the code
System.out.println("Creating statement...");
pstmt = conn.prepareStatement("UPDATE COFFEES " +
                              "SET TOTAL = TOTAL + ? " +
```

```
                                    "WHERE COF_NAME = ?");
pstmt.setInt(1, incr);
pstmt.setString(2, cofName);
ResultSet rs = pstmt.executeUpdate();
```

Una lista completa dei metodi `setXXX()` del tipo `PreparedStatement` può essere trovata in [3, 2].

3.2.5 Estrarre dati dal risultato

Quando si vogliono leggere i risultati di un'interrogazione si può usare il metodo `ResultSet.getXXX()` appropriato al caso, come segue:

```
//STEP 5: Extract data from result set
while (rs.next()) {
    // Retrieve by column name
    int id  = rs.getInt("id");
    String nome = rs.getString("nome");
    String cognome = rs.getString("cognome");
    String posizione = rs.getString("posizione");

    // Display values
    System.out.print("ID: " + id);
    System.out.print(", Nome: " + nome);
    System.out.print(", Cognome: " + cognome);
    System.out.println(", Posizione: " + posizione);
}
```


Si noti che è prevista la conoscenza del tipo di dato di ogni colonna. Ad esempio si usa il metodo `rs.getInt()` per la colonna intera `id` mentre si usa il metodo `rs.getString()` per la colonna `nome` (`varchar(100)`, cfr. esercitazioni precedenti). Una lista completa di metodi `ResultSet.getXXX()` associati ai tipi di dato dei DBMS può essere trovata in [3, 2].

3.2.6 Ripulire l'ambiente

Al termine dell'utilizzo si devono rilasciare le risorse con il seguente codice:

```
//STEP 6: Clean-up environment
rs.close();
stmt.close();
conn.close();
```

3.2.7 Catturare le eccezioni

Ogni programma che si rispetti considera le situazioni eccezionali. Il seguente codice è un esempio minimale di programma Java in cui si raccolgono tutti i passi finora descritti e in

cui si prevede la cattura delle eccezioni `SQLException` possibilmente generate dall'apertura della connessione e delle eccezioni di ogni altro tipo ovunque generate (ad esempio l'inizializzazione del driver potrebbe generare una eccezione `ClassNotFoundException` qualora non sia correttamente localizzato il driver per PostgreSQL).

```
//STEP 1. Import required packages
import java.sql.*;

public class FirstExample {
   // JDBC driver name and database URL
   static final String JDBC_DRIVER = "org.postgresql.Driver";
   static final String DB_URL = "jdbc:postgresql://localhost
      :5432/accademia";

   //  Database credentials
   static final String USER = "username";
   static final String PASS = "password";

   public static void main(String[] args) {
   Connection conn = null;
   Statement stmt = null;
   try{
      //STEP 2: Register JDBC driver
      Class.forName(JDBC_DRIVER);

      //STEP 3: Open a connection
      System.out.println("Connecting to database...");
      conn = DriverManager.getConnection(DB_URL, USER, PASS);

      //STEP 4: Execute a query
      System.out.println("Creating statement...");
      stmt = conn.createStatement();
      String sql;
      sql = "SELECT id, nome, cognome, posizione FROM Persona";
      ResultSet rs = stmt.executeQuery(sql);

      //STEP 5: Extract data from result set
      while(rs.next()){
         //Retrieve by column name
         int id  = rs.getInt("id");
         String nome = rs.getString("nome");
         String cognome = rs.getString("cognome");
         String posizione = rs.getString("posizione");

         //Display values
         System.out.print("ID: " + id);
         System.out.print(", Nome: " + nome);
```

```
            System.out.print(", Cognome: " + cognome);
            System.out.println(", Posizione: " + posizione);
        }
        //STEP 6: Clean-up environment
        rs.close();
        stmt.close();
        conn.close();
    } catch(SQLException se) {
        //Handle errors for JDBC
        se.printStackTrace();
    } catch(Exception e) {
        //Handle errors for Class.forName
        e.printStackTrace();
    } finally {
        //finally block used to close resources
        try{
            if(stmt!=null)
                stmt.close();
        }catch(SQLException se2){
        }// nothing we can do
        try{
            if(conn!=null)
                conn.close();
        }catch(SQLException se){
            se.printStackTrace();
        }//end finally try
    }//end try
    System.out.println("Goodbye!");
}//end main
}//end FirstExample
```

Una lista completa dei metodi a disposizione di un oggetto di tipo `SQLException` può essere trovata in [3, 2].

3.2.8 Compilazione del programma

Assumiamo di aver scaricato il driver JDBC4 per PostgreSQL e per JVM 1.6 (o superiore) nella cartella corrente e di aver salvato il codice sopra descritto in `FirstExample.java`. Allora la compilazione ed esecuzione del programma è la seguente:

```
$ javac FirstExample.java
$ java -classpath postgresql-9.2-1002.jdbc4.jar:. FirstExample
```

Bibliografia

[1] Multitier architecture, http://en.wikipedia.org/wiki/Multitier_architecture.

[2] Oracle Tutorial JDBC, http://docs.oracle.com/javase/tutorial/jdbc/.

[3] Tutorialspoint: JDBC Tutorial, http://www.tutorialspoint.com/jdbc/jdbc-quick-guide.htm.