

SQL

Laboratorio di ***Progettazione di Basi di Dati*** (CdS in Informatica e TPS)

a.a. 2013/2014

<http://www.di.uniba.it/~lisi/courses/basi-dati/bd2013-14.htm>

dott.ssa Francesca A. Lisi
lisi@di.uniba.it

Orario di ricevimento: giovedì ore 10-12

Sommario (VI parte)

- Procedure memorizzate (o *stored procedures*)
- SQL immerso (o *embedded SQL*)
- Call Level Interface

Riferimenti

- cap. 8 di Pratt
- cap. 6, in particolare 6.1, 6.3 e 6.4, di Atzeni et al.
- cap. 9, in particolare 9.4-9.7 di Elmasri & Navathe

Call Level Interface

- Indica genericamente interfacce che permettono di inviare richieste a DBMS per mezzo di parametri trasmessi a funzioni
- standard **SQL/CLI** ('95 e poi parte di SQL-3)
- **ODBC**: implementazione proprietaria di SQL/CLI
- **JDBC**: una CLI per il mondo Java

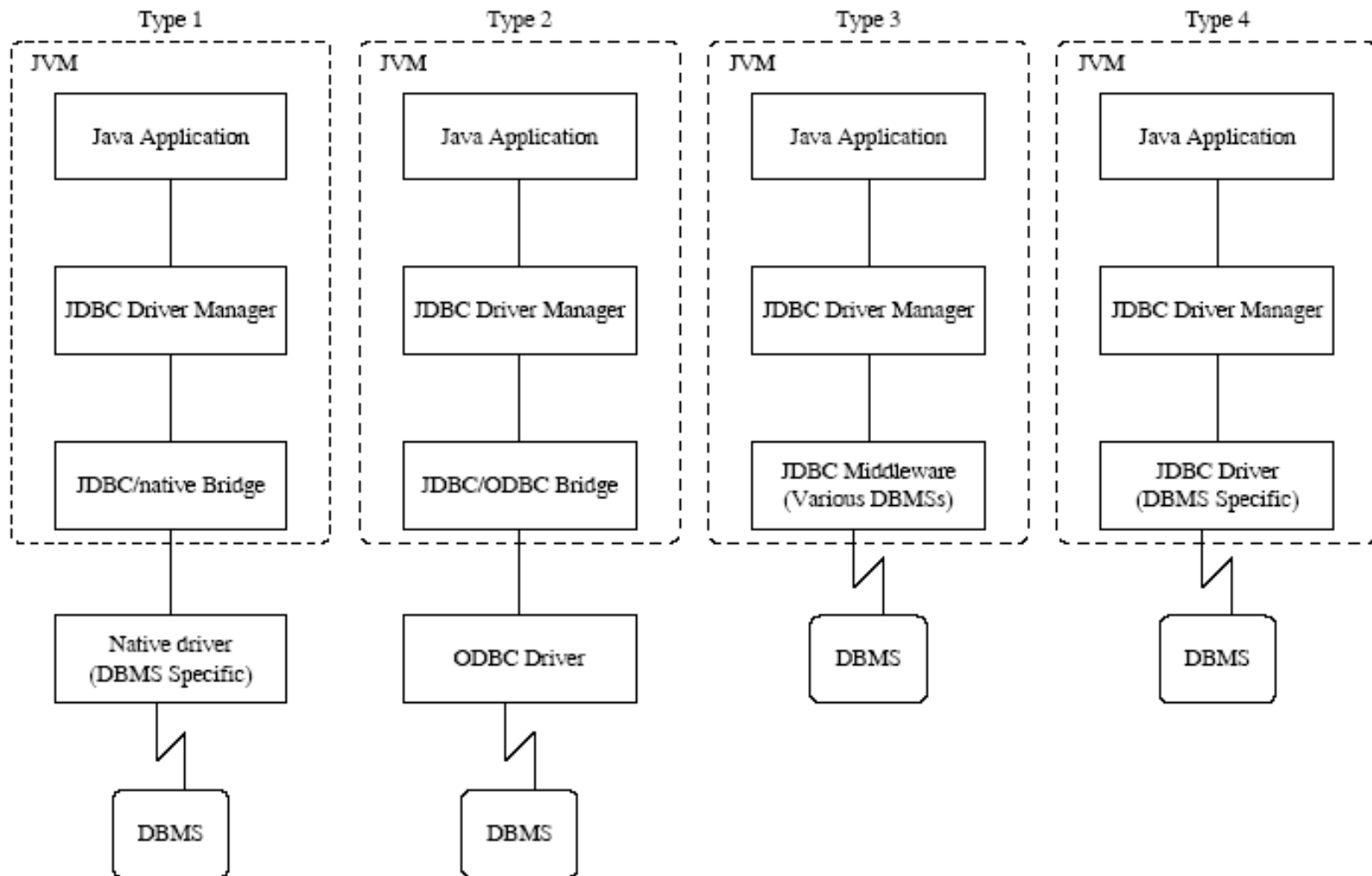
SQL immerso vs CLI

- SQL immerso permette
 - precompilazione (e quindi efficienza)
 - uso di SQL completo
- CLI
 - indipendente dal DBMS
 - permette di accedere a più basi di dati, anche eterogenee

JDBC

- Una API (Application Programming Interface) di Java (intuitivamente: una libreria) per l'accesso a basi di dati, in modo indipendente dalla specifica tecnologia
- JDBC è una **interfaccia**, realizzata da classi chiamate **driver**:
 - l'interfaccia è standard, mentre i driver contengono le specificità dei singoli DBMS (o di altre fonti informative)

I driver JDBC



Il funzionamento di JDBC, in breve

- Caricamento del driver
- Apertura della connessione alla base di dati
- Richiesta di esecuzione di istruzioni SQL
- Elaborazione dei risultati delle istruzioni SQL

Un programma con JDBC

```
import java.sql.*;
public class PrimoJDBC {
    public static void main(String[] arg){
        Connection con = null ;
        try {
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            String url = "jdbc:odbc:Corsi";
            con = DriverManager.getConnection(url);
        }
        catch(Exception e){
            System.out.println("Connessione fallita");
        }
        try {
            Statement query = con.createStatement();
            ResultSet result =
                query.executeQuery("select * from Corsi");
            while (result.next()){
                String nomeCorso = result.getString("NomeCorso");
                System.out.println(nomeCorso);
            }
        }
        catch (Exception e){
            System.out.println("Errore nell'interrogazione");
        }
    }
}
```


Un altro programma con JDBC (1)

```
import java.lang.*;
import java.sql.*;
class ProvaSelectJDBC
{
    public static void main(String argv[])
    {
        Connection con = null;
        try { Class.forName("com.ibm.db2.jcc.DB2Driver");
        }
        catch (ClassNotFoundException exClass) {
            System.err.println("Fallita connessione al database. Errore 1");
        }
        try {
            String url = "jdbc:db2:db04";
            con = DriverManager.getConnection(url);
        }
        catch (SQLException exSQL) {
            System.err.println("Fallita connessione al database. "+
                exSQL.getErrorCode() + " " +
                exSQL.getSQLState() + exSQL.getMessage() );
        }
    }
}
```

Un altro programma con JDBC (2)

```
try{ String padre = ""; String figlio = "" ; String padrePrec = "";
Statement query = con.createStatement();
String queryString =
    "SELECT Padre, Figlio FROM Paternita ORDER BY Padre";
ResultSet result = query.executeQuery(queryString);
while (result.next()){
    padre = result.getString("Padre");
    figlio = result.getString("Figlio");
    if (!(padre.equals(padrePrec))){
        System.out.println("Padre: " + padre +
            "\n Figli: " + figlio);}
    else System.out.println( "          " + figlio ) ;
    padrePrec = padre ;
}
}
catch (SQLException exSQL) {
System.err.println("Errore nell'interrogazione. "+
    exSQL.getErrorCode() + " " + exSQL.getMessage() );
}
}
```

Preliminari

- L'interfaccia JDBC è contenuta nel package `java.sql`

```
import java.sql.*;
```

- Il driver deve essere caricato (trascuriamo i dettagli)

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

- Connessione: oggetto di tipo **Connection** che costituisce un collegamento attivo fra programma Java e base di dati; viene creato da

```
String url = "jdbc:odbc:Corsi";  
con = DriverManager.getConnection(url);
```

Preliminari dei preliminari: origine dati ODBC

- Per utilizzare un driver JDBC-ODBC, la base di dati (o altro) deve essere definita come "origine dati ODBC"
- In Windows (con **YYY**, avendo già definito la base di dati **xxx.yyy** da collegare):
 - Pannello di controllo
 - Strumenti di amministrazione
 - Opzione "Origini dati ODBC"
 - Bottone "Aggiungi" ("Add")
 - Nella finestra di dialogo "Crea Nuova origine dati" selezionare "**YYY Driver**" e nella successiva
 - selezionare il file **xxx.yyy**
 - attribuirgli un nome (che sarà usato da ODBC e quindi da JDBC)

Esecuzione dell'interrogazione ed elaborazione del risultato

Esecuzione dell'interrogazione

```
Statement query = con.createStatement();  
ResultSet result =  
    query.executeQuery("select * from Corsi");
```

Elaborazione del risultato

```
while (result.next()) {  
    String nomeCorso =  
        result.getString("NomeCorso");  
    System.out.println(nomeCorso);  
}
```

La classe Statement

- Un'interfaccia i cui oggetti consentono di inviare, tramite una connessione, istruzioni SQL e di ricevere i risultati forniti
- Un oggetto di tipo **Statement** viene creato con il metodo **createStatement** di **Connection**
- I metodi dell'interfaccia **Statement**:
 - **executeUpdate** per specificare aggiornamenti o istruzioni DDL
 - **executeQuery** per specificare interrogazioni e ottenere un risultato
 - **execute** per specificare istruzioni non note a priori
 - **executeBatch** per specificare sequenze di istruzioni
- Vediamo **executeQuery**

La classe `ResultSet`

- I risultati delle interrogazioni sono forniti in oggetti di tipo `ResultSet` (interfaccia definita in `java.sql`)
- In sostanza, un *result set* è una sequenza di ennuple su cui si può "navigare" (in avanti, indietro e anche con accesso diretto) e dalla cui ennupla "corrente" si possono estrarre i valori degli attributi
- Metodi principali:
 - `next()`
 - `getXXX(posizione)`
 - es: `getString(3) ; getInt(2)`
 - `getXXX(nomeAttributo)`
 - es: `getString("Cognome") ;
getInt("Codice")`

Specializzazioni di Statement

- **PreparedStatement** permette di utilizzare codice SQL già compilato, eventualmente parametrizzato rispetto alle costanti
 - in generale più efficiente di **Statement**
 - permette di distinguere più facilmente istruzioni e costanti (e apici nelle costanti)
- i metodi **setXXX(,)** permettono di definire i parametri
- **CallableStatement** permette di utilizzare "stored procedure", come quelle di Oracle PL/SQL o anche le query memorizzate (e parametriche) di Access


```

import java.sql.*;
import javax.swing.JOptionPane;

public class SecondoJDBCprep {
    public static void main(String[] arg){
        try {
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            String url = "jdbc:odbc:Corsi";
            Connection con = DriverManager.getConnection(url);
            PreparedStatement pquery = con.prepareStatement(
                "select * from Corsi where NomeCorso LIKE ?");
            String param = JOptionPane.showInputDialog(
                "Nome corso (anche parziale)?");
            param = "%" + param + "%";
            pquery.setString(1,param);
            ResultSet result = pquery.executeQuery();
            while (result.next()){
                String nomeCorso = result.getString("NomeCorso");
                System.out.println(nomeCorso);
            }
        } catch (Exception e){System.out.println("Errore");}
    }
}

```

```

import java.sql.*;
import javax.swing.JOptionPane;

public class TerzoJDBCcall {
    public static void main(String[] arg){
        try {
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            String url = "jdbc:odbc:Corsi";
            Connection con = DriverManager.getConnection(url);
            CallableStatement pquery =
                con.prepareCall("{call queryCorso(?)}");
            String param = JOptionPane.showInputDialog(
                "Nome corso (anche parziale)?");
            param = "*" + param + "*";
            pquery.setString(1,param);
            ResultSet result = pquery.executeQuery();
            while (result.next()){
                String nomeCorso =
                    result.getString("NomeCorso");
                System.out.println(nomeCorso);
            }
        } catch (Exception e){System.out.println("Errore");}
    }
}

```

Altre funzionalità

- Molte, fra cui
 - username e password
 - aggiornamento dei ResultSet
 - richiesta di metadati
 - gestione di transazioni

Transazioni in JDBC

- Scelta della modalità delle transazioni: un metodo definito nell'interfaccia `Connection`:

`setAutoCommit(boolean autoCommit)`

- `con.setAutoCommit(true)`
 - (default) "autocommit": ogni operazione è una transazione
- `con.setAutoCommit(false)`
 - gestione delle transazioni da programma
 - `con.commit()`
 - `con.rollback()`
 - non c'è `begin transaction`

Esercitazione con MySQL

- Driver da utilizzare

mysql-connector-java-5.1.7-bin.jar

- Caricamento del driver:

```
new com.mysql.jdbc.Driver()
```

- Creazione di una connessione al database con credenziali di accesso appropriate:

```
connessione =  
    DriverManager.getConnection(  
        "jdbc:mysql://localhost/db_name",  
        username, pwd);
```

Esercitazione con MySQL

- Esercizi 1-3 sul database Prodotti Premiere da cap. 8, pag. 188-189 di Pratt (mediante JDBC);
- Esercizi 6.5-6.13 a pagg. 202-204 di Atzeni et al. (mediante JDBC).