JDBC Access

The Sql package

JDBC

- Un'applicazione per la quale sia fondamentale l'indipendenza dal Database può essere scritta in Java usando le specifiche JDBC. (Package java.sql)
  Non devono essere usate chiamate specifiche del database:
  Si deve usare SOLO la parte di SQL definita da ANSI SQL-2 standard.

  Non si deve fare ALCUN riferimento alla parte specifica di JDBC.

SW Layers to get access to a DB

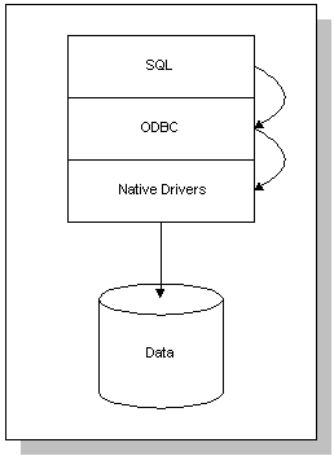
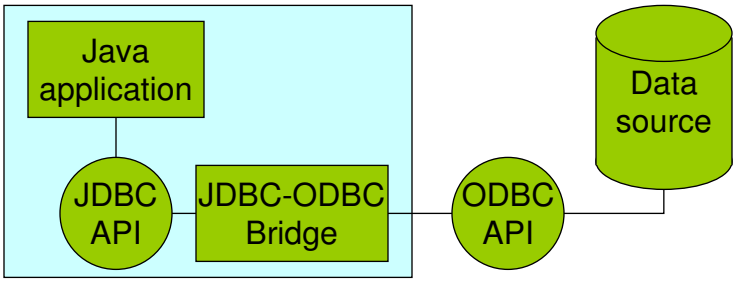


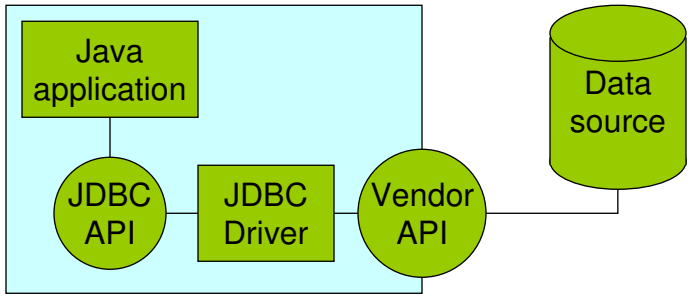
Figure 7.3 Database Access Layers

Type 1 – JDBC-ODBC Bridge

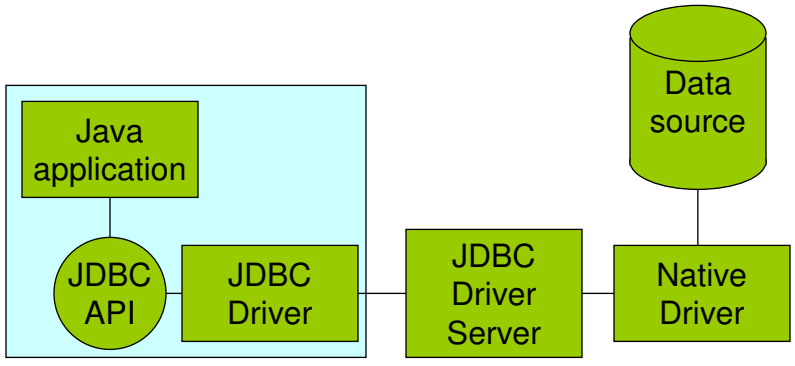


The standard JDK includes
sun.jdbc.odbc.JdbcOdbcDriver

Type 2 – Part Java, Part Native

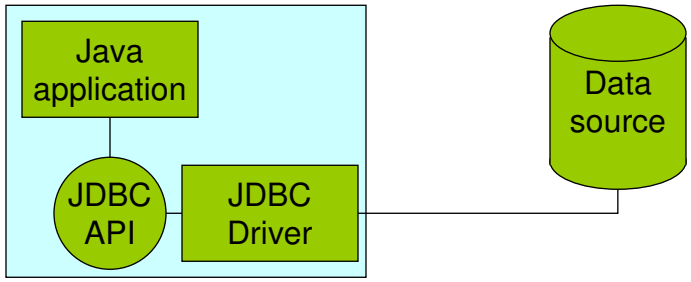


Type 3 – Intermediate DB Access Server

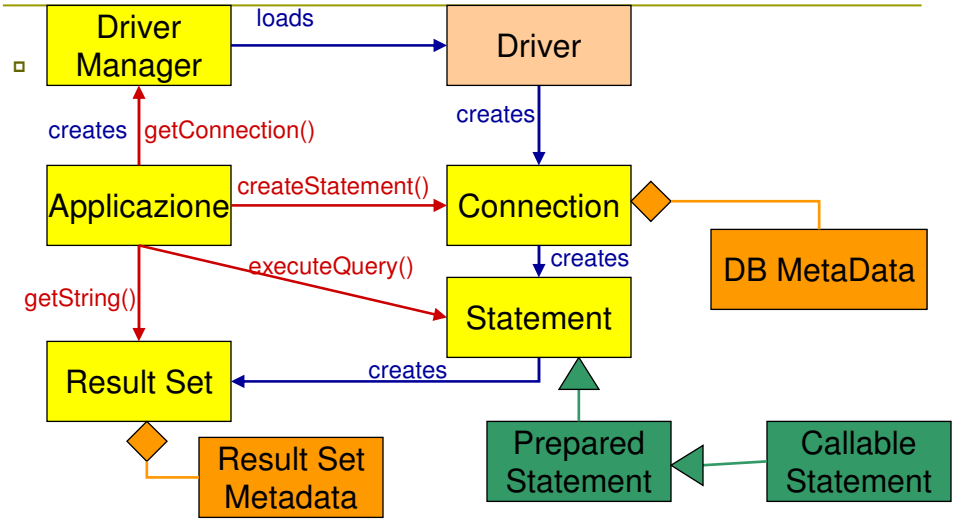


See http://industry.java.com/products/jdbc/drivers

Type 4 – Pure Java



Relations among the java.sql classes



The JDBC object model: the Manager.

- **L'oggetto centrale: java.sql.DriverManager.**
  Ha la responsabilita' di tenere traccia delle varie implementazioni JDBC che possono esistere per una applicazione.

Se, per esempio, un sistema possedesse una implementazione JDBC Sybase e una Oracle, il DriverManager sarebbe responsabile di tenere traccia di queste implementazioni.
Quando una applicazione vuole connettersi ad un database, chiede al DriverManager di darle una database connection, usando una database URL attraverso il metodo DriverManager.getConnection().

Basandosi su questa URL, il DriverManager cerca una implementazione di Driver che accetti la URL. Ottiene quindi una implementazione di Connection da quel Driver e la restituisce all'applicazione.

The JDBC object model: the URL.

- **Per permettere ad un'applicazione di definire con quale DB vuole interagire, JDBC usa il sistema standard Internet "Uniform Resource Locator".**
  Una JDBC URL (Database URL) consiste delle seguenti parti:
  jdbc:<subprotocol>:<subname>

Come per tutte le URL, il primo elemento e' la specifica del protocollo – in questo caso una sorgente di dati JDBC.
Il sottoprotocollo e' specifico della implementazione JDBC. In molti casi, e' il nome e versione DBMS name; per esempio, syb10 indica Sybase System 10.
L'elemento subname e' qualunque informazione specifica del DBMS che specifica dove deve connettersi. Per mSQL, la JDBC URL e' nella forma:
jdbc:msql://hostname:port/database

A JDBC non importa che aspetto ha la URL. La sola cosa importante e' che la implementazione desiderata di JDBC possa riconoscere la URL e avere le informazioni di cui abbisogna per collegarsi al DB tramite la URL.

The JDBC object model: the Driver.

- **Il DriverManager e' la sola classe istanziata che JDBC 1.1 procura, a parte alcune sottoclassi specializzate di java.util.Date e alcuni oggetti Exception. Le altre chiamate fatte dall'applicazione sono scritte in ossequio alle JDBC interfaces che sono implementate per gli specifici DBMSs.**

The java.sql.Driver Interface

Un Driver e' essenzialmente una Connection factory. Il DriverManager usa un Driver per determinare se e' in grado di trattare una data URL. Se uno dei Driver nella sua lista puo' gestire la URL, quel Driver deve creare un Connection object e restituirlo al DriverManager. Poiche' un applicazione referenzia solo indirettamente un Driver attraverso il DriverManager, le applicazioni in genere non si preoccupano di questa interfaccia.

The JDBC Object Model: Statement.

The java.sql.Statement Interface

- **Uno Statement e' una chiamata SQL non legata al database. It In generale e' una semplice UPDATE, DELETE, INSERT, or SELECT in cui le colonne non sono legate a dei dati Java. Uno Statement fornisce metodi per fare le chiamate SQL e restituisce all'applicazione il risultato di un SELECT statement il numero di righe toccate da un UPDATE, DELETE, o INSERT statement.**

PreparedStatement e' sottoclasse di Statement. E' una chiamata precompilata al DB che richiede di legare alcuni parametri. Se la query deve essere ripetuta più volte, la precompilazione migliora l'efficienza.

Per chiamare stored procedures, una applicazione deve usare la CallableStatement sottoclasse di PreparedStatement.
ATTENZIONE: le stored procedures spostano la logica di business ENTRO il DB!

The JDBCom: Connection & ResultSet.

The java.sql.Connection Interface

- **Una Connection e' una semplice sessione di database. Come tale immagazzina informazioni di stato concernenti la sessione di DB che gestisce e fornisce alla applicazione oggetti di tipo Statement, PreparedStatement, o CallableStatement per fare chiamate durante la sessione.**

The java.sql.ResultSet Interface

Una applicazione ottiene I dati restituiti da una query SELECT attraverso l'implementazione dell'interfaccia java.sql.ResultSet interface. Specificatamente, l'oggetto ResultSet permette all'applicazione di ottenere sequenze di righe di dati restituite dalla precedente SELECT. Il ResultSet fornisce una moltitudine di metodi che permettono di ottenere una data riga in qualunque tipo di dati java abbia senso. Per esempio, se nel DB c'e' una data memorizzata come datetime, la si puo' ottenere tramite il metodo getString() e usarla come una String.

The JDBC object model: the MetaData.

The Meta-Data Interfaces

- **Meta data sono dati che riguardano i dati. In particolare, sono I dati che danno informazioni concernenti il DB e i dati ottenibili dal DB. Java fornisce due meta-data interfaces: java.sql.ResultSetMetaData e java.sql.DatabaseMetaData.**

La ResultSetMetaData interface da' modo di ottenere informazioni su un particolare ResultSet. Per esempio, tra altre cose, ResultSetMetaData dà informazioni sul numero di colonne nel result set, il nome di una colonna, e il suo tipo.

La DatabaseMetaData interface, d'altra parte, fornisce all'applicazione informazioni sul database in generale, come quale livello di supporto ha, il suo nome, versione ecc.

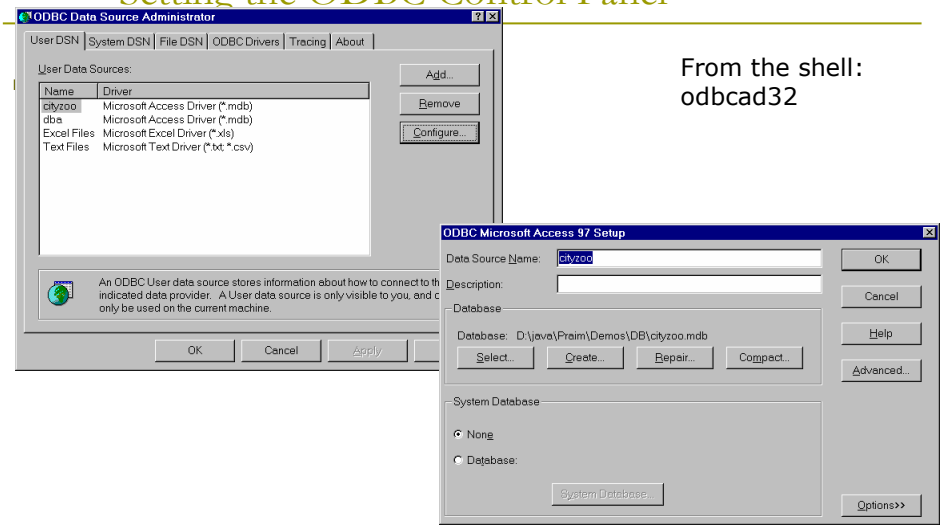
La classe DatabaseMetaData e' usata ad esempio da strumenti di sviluppo come JBuilder.

Simple Database Access Using the JDBC Interfaces

- **Scrivere una applicazione di Database usando solo chiamate JDBC comporta i seguenti passi:**

1.Chiedi al DriverManager una implementazione di Connection.
2.Chiedi alla Connection uno Statement o una sottoclasse Statement per eseguire il tuo SQL.
3.Per le sottoclassi di Statement, lega i parametri da passare alla prepared statement.
4.Esegui lo statement.
5.Per le queries, processa il result set ritornato dalla query. Ripetilo per tutti i result set finche' ce ne sono.
6.Per gli altri other statements, leggi il numero di righe toccate.
7.Chiudi lo statement.
8.Processa cosi' tutti gli statement che servono e poi chiudi la connessione.

Setting the ODBC Control Panel



From the shell:
odbcad32

Esempio

```
package first;
import java.lang.*;
import java.util.*;
import java.sql.*;
import sun.jdbc.odbc.*;
import java.io.*;

public class first
{

   public static void main(String arg[]) {
   int id;
   float amount;
   java.sql.Date dt;
   String companyName;
   String result;
   String item_desc;

     try {
//connect to ODBC database
        Class.forName(
           sun.jdbc.odbc.JdbcOdbcDriver");
        String url = "jdbc:odbc:cityzoo";
        // connect
        Properties p = new Properties();
        p.put("user", "");
        p.put("password","");
        Connection con =
        DriverManager.getConnection(url,p);
```

```
// create Statement object
    Statement stmt = con.createStatement();
    String sqlselect =
       "Select item_nbr, wholesale_cost, "
     + " item_desc, company_name"
     +" from retail_item,company"
     + " where wholesale_cost<9 and"
     + " company.company_id=retail_item.company_id"
     + " order by wholesale_cost";
// run query
    ResultSet rs = stmt.executeQuery(sqlselect);
// process results
    while(rs.next()) {
      result  = "";
      id      = rs.getInt(1);
      amount  = rs.getFloat(2);
      //dt      = rs.getDate(2);
      item_desc   = rs.getString(3);
      companyName   = rs.getString(4);
      result = "#"+result.valueOf(id) + " $";
      result+= result.valueOf(amount) + " <";
      result+= item_desc+"> <"+companyName+">";
      System.out.println("Values are: " + result);
    }
//close connection
    con.close();
  }
  catch(Exception e) {
   System.out.println(e.getMessage());
  }
  try {
   Thread.sleep(20*1000);
  } catch (Exception e) {}
 }
}
```