

## Consideratii privind utilizarea bazelor de date în JBuilder

Prof.dr. Mihaela MUNTEAN, lect.dr. Mirela-Catrinel VOICU  
Catedra de Informatica Economica, Facultatea de Stiinte Economice,  
Universitatea de Vest Timisoara

*This paper contains the most general aspects concerning various databases administration with the help of Jbuilder facilities. This includes non -visual components, used to make a data-base connection, and visual components, used to display the dates*

**Key words:** JBuilder, database, design, programming.

### Introducere

În Java, tehnologia de conectare la bazele de date este *JDBC (Java DataBase Connectivity)*, o interfața standard *SQL* care ne permite lucru cu diferite baze de date relationale. *JDBC* reprezintă *API-ul (Application Programming Interface)* ce oferă aplicațiilor *Java* acces spre diferite baze de date (figura 1). Înainte de a se utiliza biblioteca de clase *JDBC*, trebuie configurat driverul corespunzător bazelor de date relationale utilizate, acesta constituind legătura dintre aplicație și baza de date. Clasele bibliotecii *JDBC* sunt dependente de managerul de drivere, care ne permite să cunoaștem care drivere sunt necesare pentru a avea acces la înregistrările unei baze de date (figura 2). Fiecare format de bază de date utilizat în program necesită un driver diferit. Driverele de baze de date *JDBC* sunt scrise complet în *Java*. Un driver corespunde unei biblioteci (\*.jar), iar clasa de conexiune la baza de date poartă, în general, numele de *Driver*. Biblioteca de clase *JDBC* permite: stabilirea unei conexiuni la o bază de date, în raport cu driverul ales; executarea unei comenzi *SQL* asupra bazei de date; vizualizarea înregistrărilor rezultate în urma execuției comenzii *SQL*.

### Configurarea unui driver în JBuilder

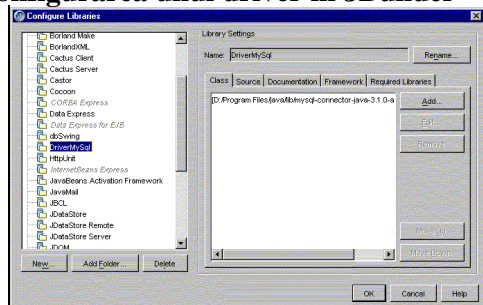


Fig. 1. Conectarea la baze de date

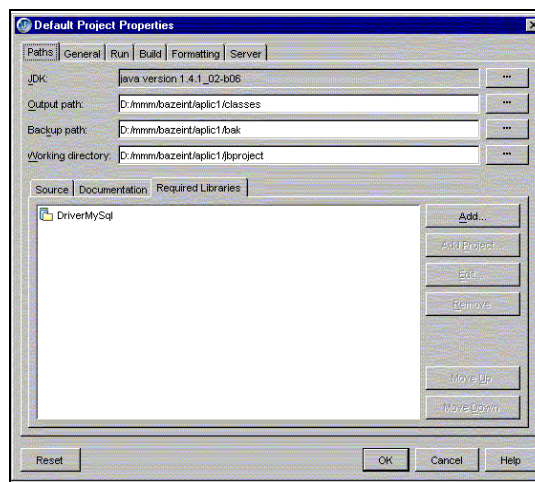


Fig. 2. Biblioteci implicite într-un proiect

În meniul *Tools* se alege opțiunea *Configures Libraries* și în fereastra *Configures Libraries* se va adăuga o nouă bibliotecă. Pentru a fi integrată implicit o bibliotecă într-un proiect, în meniul *Project* se va selecta comanda *Default Project Properties*, iar în foaia cu eticheta *Required Libraries* se va adăuga bibliotecă.

### Componente pentru lucru cu baze de date

Pe lângă tehnologia *JDBC* inerentă în *Java*, pentru accesul la informațiile dintr-o bază de date și manipularea acestora, în *JBuilder* sunt disponibile bibliotecile de componente *DataExpress* (componente nevizuale – figura 3) și *dbSwing More* (componente vizuale – figura 4).



Fig. 3. DataExpress – componente pentru conexiunea la o baza de date

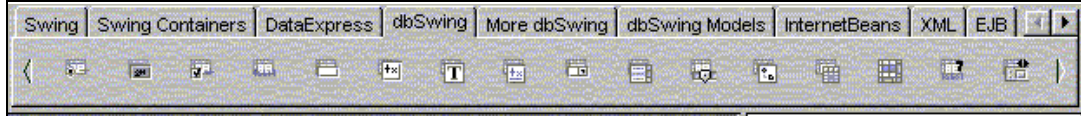


Fig. 4. dbSwing – componente pentru vizualizarea unor informatii

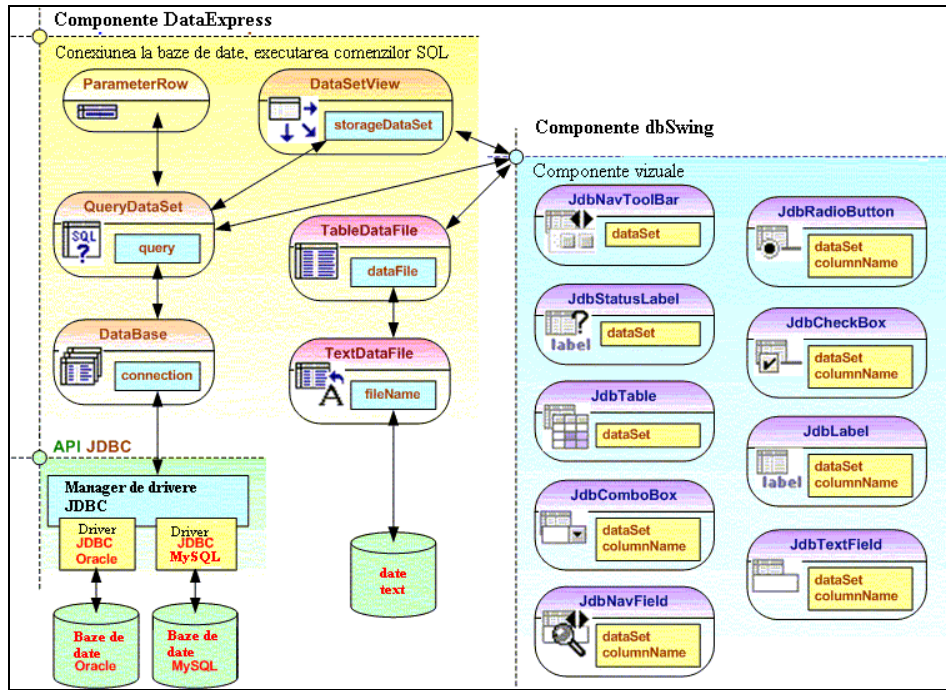


Fig. 5. Aspecte privind lucrul cu bazele de date în JBuilder

Sursa: <http://emmanuel-emy.developpez.com/Java/Tutoriels/BaseDonnees/PrincipeBase/PrincipeBase.htm>

Acestea sunt componentele utilizate cel mai des în lucrul cu bazele de date. Alte componente se găsesc pe paletele *dbSwing Models* (componente ne vizuale) și *More dbSwing* (componente vizuale). Arhitectura *DataExpress* din *JBuilder* este centrata pe dezvoltarea aplicațiilor, appleturilor, servleturilor și *JSP* (figura 5).

Componenta *DataBase* este necesară pentru accesul la datele stocate într-un server la distanță și administrarea conexiunii *JDBC* cu baza de date. Proprietatea cea mai importantă a acestei componente este proprietatea *connection*. La această proprietate se precizează baza de date cu care se face conexiunea și driverul selectat pentru managerul de drivere *JDBC* (figura 6).

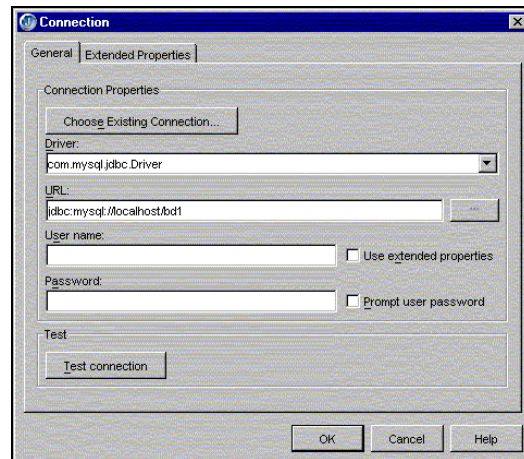


Fig. 6. Proprietatea *Connection*, componenta *DataBase*

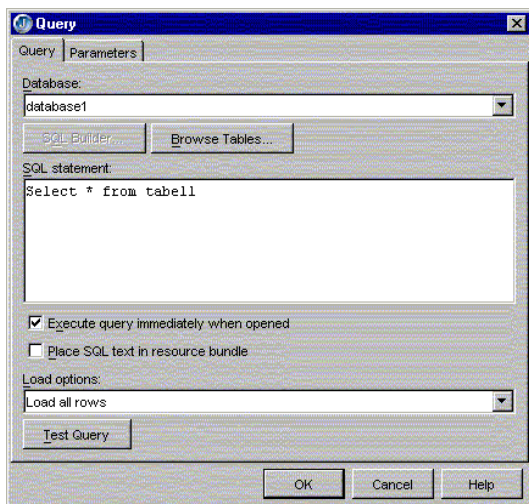







Fig. 7. Proprietatea *Query*, componenta *QueryDataSet*

Componenta *QueryDataSet*  stocheaza rezultatul unei comenzi *SQL* executate pe un server la distanta. Pentru conectarea la baza de date, la proprietatea *Query* se selecteaza o componenta *DataBase* si se introduce comanda *SQL* care se va executa (figura 7).

Componentele *dbSwing* sunt vizuale si proprietatea lor cea mai importanta este *DataSet* (unde se specifica sursa de date, de exemplu o componenta *QueryDataSet*).

Unele dintre ele se refera la câmpurile dintr-o întreaga tabela (cum ar fi *JdbTable*  sau

*JdbNavToolBar* ) iar altele se refera la câte un câmp dintr-o tabela (cum ar fi *JdbTextField* , *JdbComboBox*  etc.); pentru acestea din urma, la proprietatea *columnName* se va specifica si câmpul de legatura.

### Utilizarea comenzilor SQL prin program

Utilizarea comenzilor SQL se poate face atât în *design*, cum s-a prezentat anterior, cât si prin *program*, cum vom prezenta în continuare. Vom considera baza de date *bd1* prezentata în figura 8. Conexiunea la baza de date *bd1* se face utilizând componenta *database1*. Legaturile dintre componente sunt prezentate în figura 9.

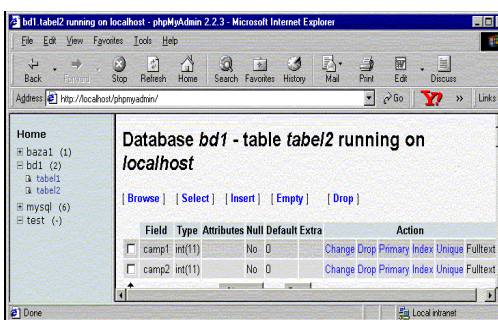


Fig. 8. Baza de date *bd1*

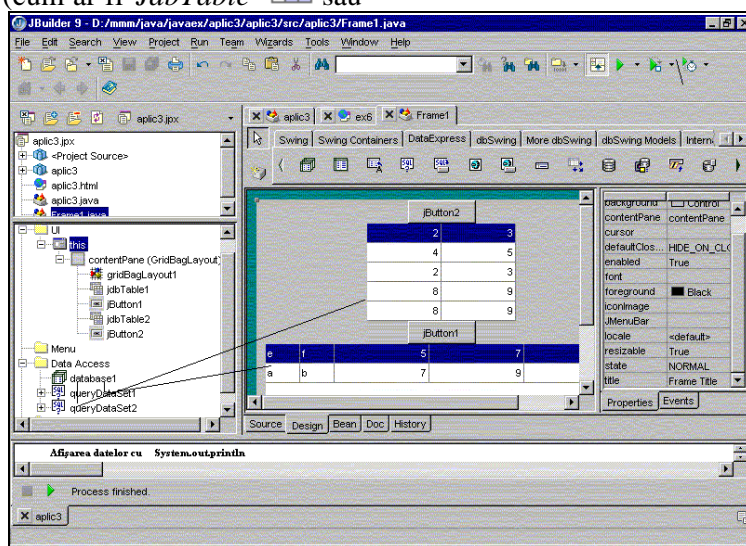


Fig. 9. Vizualizarea si modificarea continutului tabelor în *JBuilder*

Actionând asupra butonului de comanda *JButton1* vor fi vizualizate pe ecran datele

existente în tabela *tabel2* din baza de date *bd1* si actionând asupra butonului de comanda

da *JButton1*, vor fi adaugate prin program înregistrari în tabela *tabel2*. Programul sursa este urmatorul:

```

package aplic3;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import com.borland.dx.sql.dataset.*;
import com.borland.dbswing.*;
import com.borland.datastore.*;
import com.borland.dx.dataset.*;
import java.sql.*;

/**
 * <p>Title: </p>
 * <p>Description: </p>
 * <p>Copyright: Copyright (c) 2001</p>
 * <p>Company: </p>
 * @author not attributable
 * @version 1.0
 */

public class Frame1 extends JFrame {
    JPanel contentPane;
    Database database1 = new Database();
    QueryDataSet queryDataSet1 = new QueryDataSet();
    GridBagLayout gridBagLayout1 = new GridBagLayout();
    JdbTable jdbTable1 = new JdbTable();
    JButton jButton1 = new JButton();
    JdbTable jdbTable2 = new JdbTable();
    JButton jButton2 = new JButton();
    QueryDataSet queryDataSet2 = new QueryDataSet();

    //Construct the frame
    public Frame1() {
        enableEvents(AWTEvent.WINDOW_EVENT_MASK);
        try {
            jblnit();
        }
        catch (Exception e) {
            e.printStackTrace();
        }
    }

    //Component initialization
    private void jblnit() throws Exception {
        contentPane = (JPanel)this.getContentPane();
        queryDataSet1.setQuery(new
        com.borland.dx.sql.dataset.QueryDescriptor(
            database1, "select * from tabel1", null, true, Load.ALL));
        database1.setConnection(new
        com.borland.dx.sql.dataset.ConnectionDescriptor(
            "jdbc:mysql://localhost/bd1", "", "", false,
            "com.mysql.jdbc.Driver"));
        contentPane.setLayout(gridBagLayout1);
        this.setSize(new Dimension(400, 300));
        this.setTitle("Frame Title");
        jdbTable1.setDataSet(queryDataSet1);
        jButton1.setText("JButton1");
        jButton1.addActionListener(new
        Frame1_jButton1_actionAdapter(this));

        jButton2.setText("JButton2");
        jButton2.addActionListener(new
        Frame1_jButton2_actionAdapter(this));
        queryDataSet2.setQuery(new
        com.borland.dx.sql.dataset.QueryDescriptor(database1, "select *
        from tabel2", null, true, Load.ALL));
        jdbTable2.setDataSet(queryDataSet2);
        contentPane.add(jdbTable1, new GridBagConstraints(0, 3, 1, 1,
        0.0, 0.0
            ,GridBagConstraints.CENTER, GridBagConstraints.NONE,
        new Insets(0, 0, 0, 0), 109, 100));

        contentPane.add(jButton1, new GridBagConstraints(0, 2, 1, 1,
        0.0, 0.0
            ,GridBagConstraints.CENTER, GridBagConstraints.NONE,
        new Insets(0, 0, 0, 0), 0, 0));
        contentPane.add(jdbTable2, new GridBagConstraints(0, 1, 1, 1,
        0.0, 0.0
            ,GridBagConstraints.CENTER, GridBagConstraints.NONE,
        new Insets(0, 0, 0, 0), 0, 0));
        contentPane.add(jButton2, new GridBagConstraints(0, 0, 1, 1,
        0.0, 0.0
            ,GridBagConstraints.CENTER, GridBagConstraints.NONE,
        new Insets(0, 0, 0, 0), 0, 0));
    }

    //Overridden so we can exit when window is closed
    protected void processWindowEvent(WindowEvent e) {
        super.processWindowEvent(e);
        if (e.getID() == WindowEvent.WINDOW_CLOSING) {
            System.exit(0);
        }
    }

    void jButton1_actionPerformed(ActionEvent e) {
        try {
            String dbURL = "jdbc:mysql://localhost/bd1";
            String user = "";
            String password = "";
            // Înregistrarea driverului JDBC folosind clasa DriverManager
            Class.forName("com.mysql.jdbc.Driver");
            // Stabilirea unei conexiuni catre baza de date
            Connection c = DriverManager.getConnection(dbURL, user,
            password);
            // Executia unei comenzi SQL
            Statement s = c.createStatement();
            ResultSet r = s.executeQuery("select * from tabel2");
            // Procesarea rezultatelor (afisarea datelor cu Sys-
            tem.out.println)
            while (r.next()) {
                System.out.println(
                    r.getString("Camp1") + ", " +
                    r.getString("Camp2"));
            }
            // Închiderea unei conexiuni la o baza de date
            r.close();
            s.close();
        }
        catch (ClassNotFoundException a) {
            // Could not find the driver
        }
        catch (SQLException a) {
            // Could not connect to the database
        }

        System.out.println("ok");
    }

    class Frame1_jButton1_actionAdapter
        implements java.awt.event.ActionListener {
        Frame1 adaptee;
        Frame1_jButton1_actionAdapter(Frame1 adaptee) {
            this.adaptee = adaptee;
        }
        public void actionPerformed(ActionEvent e) {
            adaptee(jButton1_actionPerformed(e));
        }
    }

    class Frame1_queryDataSet2_editAdapter
        extends com.borland.dx.dataset.EditAdapter {
        Frame1 adaptee;
        Frame1_queryDataSet2_editAdapter(Frame1 adaptee) {
            this.adaptee = adaptee;
        }
    }

    void jButton2_actionPerformed(ActionEvent e) {
        try {
            String dbURL = "jdbc:mysql://localhost/bd1";
            String user = "";
            String password = "";

```

```



Class.forName("com.mysql.jdbc.Driver");
Connection c = DriverManager.getConnection(dbURL, user,
password);
Statement s = c.createStatement();
// Pentru operatii de actualizare sau stergere se foloseste metoda
executeUpdate
s.executeUpdate("Insert into tabel2 values(8,9)");
r.close();
s.close();
// Reactualizarea datelor
queryDataSet2.refresh();
}
catch (ClassNotFoundException a) {
// Could not find the driver
}
catch (SQLException a) {
// Could not connect to the database
}
System.out.println("ok");
}
}

class Frame1_jButton2_actionAdapter implements
java.awt.event.ActionListener {
Frame1 adaptee;

Frame1_jButton2_actionAdapter(Frame1 adaptee) {
this.adaptee = adaptee;
}
public void actionPerformed(ActionEvent e) {
adaptee(jButton2_actionPerformed(e));
}
}
.....

```

### Utilizarea fisierelor \*.txt ca baze de date

Pentru a utiliza fisierele \*.txt ca baze de date, se vor parcurge urmatoarele etape: se va adauga fisierul \*.txt la proiect actionând butonul *Add Files/Packages/Classes*; se adauga o componenta *TextDataFile*  si la proprietatea *fileName* se va selecta fisierul \*.txt (cu toata calea); se va utiliza un control *TableDataSet* , iar la proprietatea *dataFile* se selecteaza componenta *TextDataFile* de legatura (figura 10).

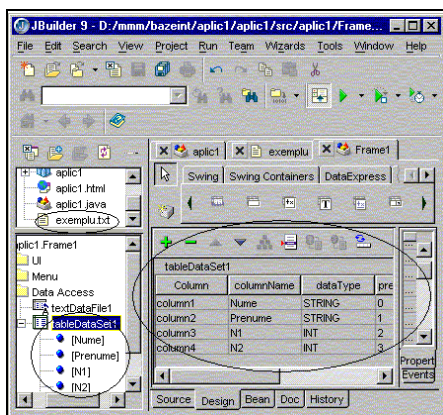


Fig. 10. Componenta *TextDataFile*

Pentru a vizualiza date din fisier sau pentru a introduce noi valori, se vor folosi componentele vizuale *dbSwing*. În cazul în care se vor folosi aceste componente (împreună cu componenta *JdbNavToolBar*), trebuie făcută observația că ele nu sunt suficiente pentru introducerea (salvarea) efectivă a datelor în fișier. Pentru această operație, se va adăuga, în plus, secvența următoare:

```

void jButton1_actionPerformed(ActionEvent e) {
try {
tableDataSet1.getDataFile().save(tableDataSet1);
}
catch(Exception ex) {ex.printStackTrace();}
}

```

asociată, de exemplu, evenimentului *actionPerformed* al unui buton de comandă.

La fiecare execuție a programului, datele existente anterior în fișier vor fi distruse. În folderul în care se află fișierul \*.txt, va fi creat un fișier cu același nume și cu extensia *.schema*. Acest fișier este un fișier text care conține definiția câmpurilor utilizate și proprietățile componente *TextDataFile*.

### Concluzie

*JBuilder* este cel mai bun mediu de implementare a aplicațiilor *Java*. Aplicațiile cu baze de date se implementează foarte ușor în design pentru o sferă foarte largă de situații. Programarea nu este abandonată în *JBuilder*, dar utilizatorul va introduce (datorită performanțelor mediului de dezvoltare) mai puține părți de program, ceea ce va conduce la o implementare mai rapidă și corectă a aplicațiilor, ceea ce demonstrează eficiența utilizării acestui mediu.

### Bibliografie

<http://emmanuel-emy.developpez.com/Java/Tutoriels/BaseDonnees/PrincipeBase/PrincipeBase.htm>

<http://thor.info.uaic.ro/~acf/java/>

<http://javaalmanac.com/egs/java.sql/pkg.html>

<http://java.developpez.com/livres/penserjenjava/>

Stefan Tanasa, Cristian Olaru, Stefan Andrei – *Java de la 0 la expert*, Iasi, Polirom 2003

Borland, *JBuilder9 – Développement d'applications de bases de données*, 2003