

TEHNICI AVANSATE DE PROGRAMARE

LUCRARE DE LABORATOR 9

JDBC – Java Database Connectivity

I. SCOPUL LUCRĂRII

Lucrarea de față are rolul de a prezenta și familiariza studentul cu modul de accesare și interogare a bazelor de date, lucrând în cadrul aplicațiilor Java.

La sfârșitul acestei lucrări, studentul va avea posibilitatea să scrie programe Java în care să utilizeze noțiunile învățate.

II. NOȚIUNI TEORETICE

1. Driveri JDBC

JDBC (Java DataBase Connectivity) reprezintă API-ul oferit de Sun Microsystems pentru dezvoltarea de aplicații Java care accesează bazele de date gestionate de diverse DBMS-uri (Sisteme de gestiune a bazelor de date). O funcție foarte importantă a JDBC este faptul că lucrează cu instrucțiuni SQL. Pentru a utiliza API-ul JDBC cu un DBMS particular este nevoie de un driver JDBC care să medieze între tehnologia JDBC și baza de date. Driverul poate fi scris numai în Java sau într-o combinație de Java și metode native JNI (Java Native Interface). Ultima specificație JDBC existentă are vers. 3.0 și conține două pachete:

1. **java.sql**
2. **javax.sql** care oferă capabilități adiționale pe partea de server

Observație:

Aplicațiile care utilizează baze de date trebuie să includă pachetul **java.sql**, ci nu pachetul care conține implementarea driver-ului particular folosit. Totuși calea spre pachetul conținând driverul trebuie să fie prezentă în CLASSPATH.

Există patru tipuri de drivere JDBC:

1. Puntea JDBC-ODBC. Este reprezentat de clasa **sun.jdbc.odbc.JdbcOdbcDriver**. Puntea traduce metodele JDBC în apeluri de funcții ODBC, și necesită ca bibliotecile ODBC native și driverele ODBC să fie instalate și configurate pentru fiecare client ce utilizează un driver de acest tip. Funcționează doar pentru sistemele de operare Microsoft Windows și Sun Solaris.
2. Java-API nativ. Folosesc interfața JNI pentru a face apeluri direct la API-ul unei baze de date locale. Sunt mai rapide decât tipul 1. Dar, de asemenea, necesită instalarea și configurarea bibliotecilor client bază de date native pe mașina client.
3. Java-protocol de rețea. Folosesc un protocol de rețea (de obicei, TCP/IP) pentru a comunica cu aplicația JDBC middleware. Aplicația JDBC middleware traduce cererile JDBC folosind protocolul de rețea în apeluri de funcții specifice bazelor de date. Sunt cele mai flexibile, deoarece nu necesită biblioteci de baze de date native pe client și se pot conecta la mai multe baze de date.
4. Java-protocol bază de date. Implementează un protocol de bază de date pentru a

comunica direct cu baza de date. Sunt cele mai rapide. De asemenea, nu necesită biblioteci de baze de date native pe client Sunt specifice unui singur DBMS.

Pentru gestiunea driverelor folosite într-o aplicație, JDBC API folosește un manager de drivere reprezentat de o instanță a clasei **DriverManager**.

2. Utilizarea punții JDBC -ODBC

În ODBC, bazele de date sunt reprezentate ca surse de date, identificate printr-un nume (DSN – Data Source Name) și accesibile printr-un driver ODBC corespunzător.

Iată un exemplu, care arată cum se procedează pentru o bază de date creată în Access, sub sistemul de operare Windows 98. Presupunem că baza de date a fost deja creată și are numele **mesaje.mdb**. Pentru a putea face interogări asupra ei, trebuie mai întâi să o introducem ca sursă de date în ODBC. Pentru aceasta, se procedează astfel: din Control Panel alegem **ODBC Data Sources**, ceea ce va determina apariția unei ferestre **ODBC Data Sources Administrator**. În rubrica **User DSN**, folosind butonul **Add** și wizardul care se declanșează, selectăm pe rând driverul ODBC (spre Access pentru exemplul nostru), respectiv baza de date **mesaje.mdb**, și punem numele acestei surse de date (Data Source Name) **myMessages.mdb**. Mai departe, în aplicația Java, numele bazei de date utilizat va fi numele sursei de date, adică **myMessages.mdb**.

3. Accesarea unei baze de date folosind JDBC

Trebuie urmați 5 pași:

1. înregistrarea driver-ului JDBC folosind gestionarul de drivere **DriverManager**
2. stabilirea unei conexiuni către baza de date
3. execuția unei instrucțiuni SQL
4. procesarea rezultatelor
5. închiderea conexiunii către baza de date

3.1 Înregistrarea driver-ului JDBC

Un driver JDBC este înregistrat automat de managerul de drivere atunci când clasa driver este încărcată dinamic prin metoda **Class.forName()**. Metoda este statică și permite mașinii virtuale Java să aloce dinamic, să încarce și să facă o legătură la clasa specificată ca argument al metodei. În cazul în care clasa nu este găsită, se aruncă o excepție **ClassNotFoundException**.

Iată un exemplu pentru înregistrarea driver-ului punte JDBC-ODBC:

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

3.2 Stabilirea conexiunii către baza de date

O conexiune este identificată printr-un URL specific. Sintaxa standard pentru URL-ul unei baze de date este:

```
jdbc<subprotocol>:<nume>
```

Prima parte arată că se folosește JDBC pentru stabilirea conexiunii, **<subprotocol>** este un nume de driver valid, iar **<nume>** este un nume logic sau alias care corespunde bazei de date fizice. Dacă baza de date este accesată prin Internet, atunci secțiunea **<nume>** va fi de forma **//numeHost:port/numeDB**

Pentru stabilirea unei conexiuni la o bază de date, se folosește metoda statică **getConnection()** din clasa **DriverManager**:

```
Connection con=DriverManager.getConnection("jdbc:odbc:myMessages");
```

sau dacă baza de date necesită autentificare:

```
Connection con=DriverManager.getConnection("jdbc:odbc:myMessages",
                                           numeUtilizator,Parola);
```

3.3 Execuția unei instrucțiuni SQL

Pentru execuția unei instrucțiuni SQL neparametrizate, se folosește metoda **createStatement()** aplicată unui obiect **Connection**.

```
Statement instructiune=con.createStatement();
```

Se poate aplica apoi una din următoarele metode:

- **executeQuery()** – pentru interogările care returnează mulțimi rezultat (instanțe ale clasei **ResultSet**). Este cazul instrucțiunilor **SELECT**.
- **executeUpdate()** – pentru operațiile de actualizare **INSERT**, **UPDATE**, **DELETE**, cât și pentru interogările SQL DDL de genul **CREATE TABLE**, **DROP TABLE**, **ALTER TABLE**. Metoda returnează un întreg care reprezintă fie numărul înregistrării afectate, fie este 0.
- **execute()** – se utilizează atunci când se obține mai mult de o mulțime rezultat.

```
ResultSet rs=instructiune.executeQuery("select * from myMessages");
String sql="insert into myMessages values ('Popescu','Ion','Craiova')";
int raspuns=instructiune.executeUpdate(sql);
```

Dacă se dorește realizarea de apeluri SQL având date variabile drept intrare, se va folosi clasa **PreparedStatement** care moștenește clasa **Statement**.

```
PreparedStatement instructiune=con.prepareStatement("update myMessages
                                                    set nume=? Where prenume like ?");
instructiune.setString(1,"Popescu");
instructiune.setString(2,"Ion");
instructiune.executeUpdate();
```

Se poate utiliza **PreparedStatement** și atunci când nu avem parametri.

```
PreparedStatement instructiune=con.prepareStatement("select *
                                                    from myMessages");
ResultSet rs=instructiune.executeQuery();
```

3.4 Procesarea rezultatelor

Pentru parcurgerea simplă a înregistrărilor unui obiect din clasa **ResultSet** putem folosi metoda **next()**.

```
while(rs.next()) // implicit cursorul positionat inainte de prima linie
{
    System.out.println(rs.getString("nume")+"", "+rs.getString("prenume")+
                        "", "+rs.getString("oras"));
}
```

3.5 Închiderea conexiunii la baza de date

Se recomandă ca după procesarea datelor, să se închidă explicit conexiunea către baza de

date. Întâi se vor închide obiectele **Statement** și **ResultSet** folosind metodelor lor **close()**, apoi se va închide obiectul **Connection** prin apelarea metodei **close()** a acestuia:

```
try{
    //...
    rs.close();
    instructiune.close();
}
catch(SQLException e){
    System.out.println("Eroare la inchidere interogare: "+e.toString());
}
finally{
    try{
        if(conexiuneBazadata!=null) {
            conexiuneBazadata.close();
        }
    }
    catch(SQLException e){
        System.out.println("Eroare la inchidere conexiune: "+e.toString());
    }
}
```

4. Exemple

Vom crea o bază de date Access numită **StudentiDB**. Se va utiliza apoi puntea JDBC-ODBC pentru această bază de date, așa cum s-a descris în secțiunea 2 a laboratorului. Sursa de date corespunzătoare acestei baze de date se va numi tot **StudentiDB**.

În continuare vom prezenta câteva programe Java foarte simple care: creează un tabel **studenti** pentru baza noastră de date, listează informațiile conținute în acest tabel, inserează o înregistrare în tabel. De asemenea, vom prezenta cum se pot obține informații despre MetaDate (denumirile coloanelor unui tabel, tipurile coloanelor).

```
//CreateStud.java
import java.sql.*;

public class CreateStud {

    public static void main(String args[]) {

        String url = "jdbc:odbc:StudentiDB";
        Connection con;
        String sir;
        sir = "create table studenti " +
            "(Nume_student varchar(32), " +
            "Nr_matricol int, " +
            "An_stud int, " +
            "Grupa char(10), " +
            "Nota1 int, " +
            "Nota2 int, " +
            "Nota3 int, " +
            "Media double)";
        Statement stmt;

        try {
```

```

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

    } catch(java.lang.ClassNotFoundException e) {
        System.err.print("ClassNotFoundException: ");
        System.err.println(e.getMessage());
    }

    try {
        con = DriverManager.getConnection(url,"", "");
        stmt = con.createStatement();
        stmt.executeUpdate(sir);
        stmt.close();
        con.close();

    } catch(SQLException ex) {
        System.err.println("SQLException:" + ex.getMessage());
    }
}

//Interogare.java
import java.sql.*;
public class Interogare {
    public static void main(String args[]) {
        String url = "jdbc:odbc:StudentiDB";
        Connection con;
        Statement stmt;
        String interogare="select Nume_student,Nr_matricol
                           from Studenti";

        try {
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        }
        catch(java.lang.ClassNotFoundException e) {
            System.err.print("ClassNotFoundException: ");
            System.err.println(e.getMessage());
        }
        try {
            con = DriverManager.getConnection(url,"", "");
            stmt = con.createStatement();
            ResultSet rs = stmt.executeQuery(interogare);
            System.out.println("ListA Studentilor:");
            while (rs.next()) {
                String s = rs.getString("Nume_Student");
                int f = rs.getInt("Nr_matricol");
                System.out.println(s + "    " + f);
            }

            stmt.close();
            con.close();

        } catch(SQLException ex) {
            System.err.println("SQLException:" + ex.getMessage());
        }
    }
}

//InsertStud.java
import java.sql.*;
public class InsertStud {

```

```

public static void main(String args[]) {

    String url = "jdbc:odbc:StudentiDB";
    Connection con;
    String sir;
    sir = "insert into studenti values
            ('Gancea Alin',1456,1,123,5,7,6,6)" ;

    Statement stmt;

    try {
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

    } catch (java.lang.ClassNotFoundException e) {
        System.err.print("ClassNotFoundException: ");
        System.err.println(e.getMessage());
    }

    try {
        con = DriverManager.getConnection(url,"", "");
        stmt = con.createStatement();
        stmt.executeUpdate(sir);
        stmt.close();
        con.close();

    } catch (SQLException ex) {
        System.err.println("SQLException:" + ex.getMessage());
    }

}

//ListCol.java
import java.sql.*;

class ListCol {

public static void main(String args[]) {
    String url = "jdbc:odbc:StudentiDB";
    Connection con;
    Statement stmt;
    String query = "select * from Studenti";
    try {
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    }
    catch (java.lang.ClassNotFoundException e) {
        System.err.print("ClassNotFoundException: ");
        System.err.println(e.getMessage());
    }

    try {
        con = DriverManager.getConnection(url);
        stmt = con.createStatement();
        ResultSet rs = stmt.executeQuery(query);
        ResultSetMetaData rsmd = rs.getMetaData();

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

```

```

int numberOfColumns = rsmd.getColumnCount();

for (int i = 1; i <= numberOfColumns; i++)
{
    if (i > 1) System.out.print(", ");
    String columnName = rsmd.getColumnName(i);
    System.out.print(columnName);
}

        for (int i = 1; i <= numberOfColumns; i++)
        {
            if (i > 1) System.out.print(", ");
            String columnType = rsmd.getColumnTypeName(i);
            System.out.print(columnType);
        }

System.out.println("");

while (rs.next()) {
    for (int i = 1; i <= numberOfColumns; i++)
    {
        if (i > 1) System.out.print(", ");
        String columnValue = rs.getString(i);
        System.out.print(columnValue);
    }
    System.out.println("");
}

stmt.close();
con.close();
} catch (SQLException ex) {
    System.err.print("SQLException: ");
    System.err.println(ex.getMessage());
}
}
}

```

5. Exemple folosind interfata grafica

Se va utiliza aceeași bază de date **StudentiDB** creată în secțiunea precedentă. În continuare vom prezenta un program Java (incomplet) care implementează o interfață grafică prin care utilizatorul poate să formuleze fraze SQL și să primească răspunsul.

```

import java.awt.*;
import java.awt.event.*;

public class DBQueryGUI extends Frame implements
    WindowListener, ActionListener {

    TextField sqlExec;
    Button executeButton;
    Label label1;
    static TextArea edit1;
    Label label2;
    Button quitButton;
    Button resetButton;
    SimpleSelect ob;

    public void init() {

```

```

this.addWindowListener(this);
LayoutManager layoutman = new BorderLayout();
this.setLayout(layoutman);
Panel p1 = new Panel();
Panel p2 = new Panel();
Panel p3 = new Panel();
sqlExec = new TextField(50);
edit1 = new TextArea(12, 80);
label1 = new Label("Enter an SQL statement:");
label2 = new Label("Output");
executeButton = new Button("Execute");
quitButton = new Button("Quit");
resetButton = new Button("Reset");
p1.add(label1);
p1.add(sqlExec);
p2.add(executeButton);
p2.add(resetButton);
p2.add(quitButton);
p3.add(label2);
p3.add(edit1);
this.add("North", p1);
this.add("Center", p3);
this.add("South", p2);
executeButton.addActionListener(this);
quitButton.addActionListener(this);
resetButton.addActionListener(this);
}

public void actionPerformed(ActionEvent e) {
    String command = e.getActionCommand();
    if (command.equals("Execute"))
        clickedExecuteButton();
    if (command.equals("Quit"))
        clickedQuitButton();
    if (command.equals("Reset"))
        sqlExec.setText("");
}

public void clickedExecuteButton() {
    edit1.setText("");
    edit1.repaint();
    if (ob == null) {
        try {
            ob = new SimpleSelect();
        } catch (ClassNotFoundException cnfe) {
            cnfe.printStackTrace();
            return;
        }
    }
    if (!ob.open()) {
        edit1.setText("Nu se poate stabili o conexiune cu baza de
                        date !");
        return;
    }
    String sqlString, result;

    sqlString = sqlExec.getText(); result = ob.queryDatabase(sqlString);
    if (result != null)
        edit1.setText(result);
    else

```

```

        edit1.setText("A aparut o eroare !");
    }
    public void clickedQuitButton() {
        if (ob != null)
            ob.close();
        System.exit(0);
    }

    public static void main(String argv[]) {
        DBQueryGUI sql = new DBQueryGUI();
        sql.init();
        sql.pack();
        sql.setTitle("JDBC/ODBC GUI");
        sql.show();
    }
    public void windowActivated(WindowEvent e) {}
    public void windowClosed(WindowEvent e) {}
    public void windowClosing(WindowEvent e) { System.exit(0); }
    public void windowDeactivated(WindowEvent e) {}
    public void windowDeiconified(WindowEvent e) {}
    public void windowIconified(WindowEvent e) {}
    public void windowOpened(WindowEvent e) {}
}

```

III. MODUL DE LUCRU

 Clasic:

1. Se editează codul sursă al programului Java folosind un editor de text disponibil (de ex., se poate utiliza Notepad).
2. Se salvează fișierul cu extensia **.java**.
3. Compilarea aplicației Java se va face din linia de comandă:
javac nume_fișier_sursă.java
 În cazul în care programul conține erori acestea vor fi semnalate și afișate.
4. Pentru rularea aplicației Java, se lansează interpretorul Java:
java nume_clasă_care_conține_main

 Se folosește utilitarul disponibil în laborator J2SDK Net Beans IDE.

IV. TEMĂ

1. Se vor parcurge toate exemplele prezentate în platforma de laborator testându-se practic și explicând rezultatele obținute.
2. Scrieți un program Java pentru a șterge informații din tabelul **studenti** și pentru a actualiza anumite înregistrări din acest tabel.
3. Completați programul din laborator, implementând clasa **SimpleSelect**. Metoda principală a clasei este **queryDatabase ()** care primește ca parametru un obiect de tipul String ce reprezintă o frază SQL și returnează un obiect SQL care reprezintă rezultatul sau o excepție.