Curs 11
Lucrul cu baze de date în Java

Generalitati despre baze de date

Aplicatiile care folosesc baze de date sunt, în general, aplicatii complexe folosite pentru gestionarea unor informatii de dimensiuni mai mari într-o maniera sigura si eficienta.

Ce este o baza de date ?

O *baza de date* reprezinta o modalitate de stocare a unor informatii (date) pe un suport extern, cu posibilitatea regasirii acestora. Uzual, o baza de date este memorata într-unul sau mai multe fisiere.

Modelul clasic de baza de date este cel *relational*, în care datele sunt memorate în tabele. Pe lânga tabele, o baza de date mai poate contine: proceduri si functii, utilizatori si grupuri de utilizatori, tipuri de date, obiecte, etc.

Dintre producatorii cei mai importanti de baze de date amintim Oracle, Sybase, IBM, Informix, Microsoft, etc.

Crearea unei baze de date

Se face cu aplicatii specializate oferite de producatorul tipului respectiv de baza de date.

Accesul la o baza de date

Se face prin intermediul unui **driver** specific tipului respectiv de baza de date. Acesta este responsabil cu accesul efectiv la datele stocate, fiind legatura între aplicatie si baza de date.

Ce este JDBC ?

Definitie

> *JDBC (Java Database Connectivity)* este o interfata standard SQL de acces la baze de date. JDBC este constituita dintr-un set de clase si interfete scrise în Java, furnizând mecanisme standard pentru proiectantii aplicatiilor de baze de date.

Pachetul care ofera suport pentru lucrul cu baze de date este **java.sql**.

Folosind JDBC este usor sa transmitem secvente SQL catre baze de date relationale. Cu alte cuvinte, nu este necesar sa scriem un program pentru a accesa o baza de date Oracle, alt program pentru a accesa o baza de date Sybase si asa mai departe. Este de ajuns sa scriem un singur program folosind API-ul JDBC si

acesta va fi capabil sa trimita secvente SQL bazei de date dorite. Bineînteles, scriind codul sursa în Java, ne este asigurata portabilitatea programului. Deci, iata doua motive puternice care fac combinatia Java - JDBC demna de luat în seama.

Fiind robust, sigur, usor de folosit, usor de înteles, Java este un excelent limbaj pentru a dezvolta aplicatii de baze de date. Tot ceea ce-i lipseste este modalitatea prin care aplicatiile Java pot comunica cu bazele de date. Aici vine însa JDBC-ul care ofera acest mecanism.

Ce face JDBC-ul?

In linii mari, JDBC face trei lucruri:

- stabileste o conexiune cu o baza de date
- trimite secvente SQL
- prelucreaza rezultatele

Conectarea la o baza de date

Procesul de conectare la o baza de date implica doua operatii:

1. încarcarea în memorie a unui driver corespunzator
2. realizarea unei conexiuni propriu-zise

Definitie

O *conexiune (sesiune)* la o baza de date reprezinta un context prin care sunt trimise secvente SQL si primite rezultate. Intr-o aplicatie pot exista mai multe conexiuni simultan la baze de date diferite sau la aceeasi baza.

Clasele si interfetele responsabile cu realizarea unei conexiuni sunt:

- clasa **`DriverManager`**, ce se ocupa cu înregistrarea driverelor ce vor fi folosite în aplicatie
- interfata **`Driver`**, pe care trebuie sa o implementeze orice clasa ce descrie un driver
- clasa **`DriverPropertyInfo`**
- interfata **`Connection`**, descrie obiectele ce modeleaza o conexiune propriu-zisa cu baza de date

Incarcarea în memorie a unui driver

Primul lucru pe care trebuie sa-l faca o aplicatie în procesul de conectare la o baza de date este sa încarce în memorie clasa ce implementeaza driver-ul necesar comunicarii cu respectiva baza de date. Acest lucru poate fi realizat prin mai multe modalitati:

```
1. DriverManager.registerDriver(new sun.jdbc.odbc.JdbcOdbcDriver());
2. Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
3. System.setProperty("jdbc.drivers", "sun.jdbc.odbc.JdbcOdbcDriver");
4. java -Djdbc.drivers=sun.jdbc.odbc.JdbcOdbcDriver
```

Specificarea unei baze de date

O data ce un driver JDBC a fost încarcat în memorie cu `DriverManager`, acesta poate fi folosit la stabilirea unei conexiuni cu o baza de date. Având în vedere faptul ca pot exista mai multe drivere înregistrate în memorie, trebuie sa avem posibilitea de a specifica pe lânga identificatorul bazei de date si driverul ce

trebuie folosit.
Aceasta se realizeaza prin intermediul unei adrese specifice, numita **JDBC URL**, ce are urmatorul format:

`jdbc:sub-protocol:identificator_baza_de_date`

Câmpul *sub-protocol* denumeste tipul de driver ce trebuie folosit pentru realizarea conexiunii si poate fi *odbc, oracle, sybase, db2* si asa mai departe. *Identificatorul bazei de date* este un indicator specific fiecarui driver care specifica baza de date cu care aplicatia doreste sa interactioneze. In functie de tipul driver-ului acest identificator poate include numele unei masini gazda, un numar de port, numele unui fisier sau al unui director, etc.

```
jdbc:odbc:testdb
jdbc:oracle:thin@persistentjava.com:1521:testdb
jdbc:sybase:testdb
jdbc:db2:testdb
```

La primirea unui JDBC URL, `DriverManager`-ul va parcurge lista driver-elor înregistrate în memorie, pâna când unul dintre ele va recunoaste URL-ul respectiv. Daca nu exista nici unul potrivit, atunci va fi lansata o exceptie de tipul `SQLException`, cu mesajul `no suitable driver`.

Realizarea unei conexiuni

Metoda folosita pentru realizarea unei conexiuni este **getConnection** din clasa `DriverManager` si poate avea mai multe forme:

```
Connection c = DriverManager.getConnection(url);
Connection c = DriverManager.getConnection(url, username, password);
Connection c = DriverManager.getConnection(url, dbproperies);
```

O conexiune va fi folosita pentru:

- crearea de secvente SQL ce vor fi folosite pentru interogarea sau actualizarea bazei
- aflarea unor informatii legate de baza de date (meta-date)

Clasa `Connection` asigura suport pentru controlul tranzactiilor din memorie catre baza de date prin metodele `commit, rollback, setAutoCommit .`

Efectuarea de secvente SQL

O data facuta conectarea cu `DriverManager.getConection()`, se poate folosi obiectul `Connection` rezultat pentru a se crea un obiect de tip **Statements**, cu ajutorul caruia putem trimite secvente SQL catre baza de date. Cele mai uzuale comenzi SQL sunt cele folosite pentru:

1. interogarea bazei de date (`SELECT`)
2. actualizarea bazei de date (`INSERT, UPDATE, DELETE`)

```
Connection c = DriverManager.getConnection(url);
Statement s = c.createStatement();
ResultSet r = s.executeQuery("SELECT * FROM un_tabel ORDER BY o_coloana");
s.executeUpdate("DELETE * FROM un_tabel");
```

Metoda `executeQuery` trimite interogari SQL catre baza de date si primeste rasuns într-un obiect de tip `ResultSet`.

Curs 11

Obtinerea si prelucrarea rezultatelor

Interfata ResultSet

```
String query = "SELECT cod, nume FROM localitati ORDER BY nume";
ResultSet r = s.executeQuery( query );
while (r.next()) {
        System.out.println (
                r.getString ("cod") + "," +
                r.getString ("nume") );
```

Interfata ResultSetMetaData

```
ResultSet  r = s.executeQuery(" SELECT * FROM localitati" );
ResultSetMetaData rsmd = r.getMetaData();
System.out.println("Coloane: " + rsmd.getColumnCount());
```

Exemplu

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

public class TestJDBC {
        public static void main (String[] args) {
                String dbUrl = "jdbc:odbc:test";
                String user = "dba";
                String password = "sql";
                try{
                        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
                }
                catch(ClassNotFoundException e) {
                        e.printStackTrace();
                        System.out.println("Eroare incarcare driver!\n" + e);
                }
                try{
                        Connection c=DriverManager.getConnection(dbUrl, user,
password);
                        Statement s= c.createStatement();
                        ResultSet  r =
                                s.executeQuery(
                                        " SELECT cod, nume FROM localitati"+
                                        " ORDER BY nume");

                        while (r.next()) {
                                System.out.println (
                                        r.getString ("cod") + "," +
                                        r.getString ("nume") );
                        }
                        s.close();
                }
                catch(SQLException e) {
                        e.printStackTrace();
                }
        }
}
```