

Lucrul cu baze de date

- Generalități despre baze de date
- JDBC
- Conectarea la o bază de date
- Inregistrarea unui driver
- Specificarea unei baze de date
- Tipuri de drivere
- Realizarea unei conexiuni
- Efectuarea de secvențe SQL
- Obținerea și prelucrarea rezultatelor
- Exemplu simplu
- Lucrul cu meta-date

Ce este o bază de date ?

O *bază de date* reprezintă o modalitate de stocare a unor informații (date) pe un suport extern, cu posibilitatea regăsirii acestora.

- Memorată în unul sau mai multe fișiere
- Eficiență (indecși)
- Siguranță (chei primare, secundare, trigger)
- Securitate (parole)
- Modele: **relațional**, orientat-obiect, distribuit, etc.
- Conține: tabele, proceduri și funcții, utilizatori și grupuri de utilizatori, tipuri de date, obiecte, etc.

DBMS (Database Management System)

Sistem pentru gestiunea bazelor de date (SGBD)

Producători SGBD:

- Oracle
- Sybase
- IBM
- Informix
- Microsoft

Crearea unei baze de date

Folosind aplicații specializate / scripturi SQL.

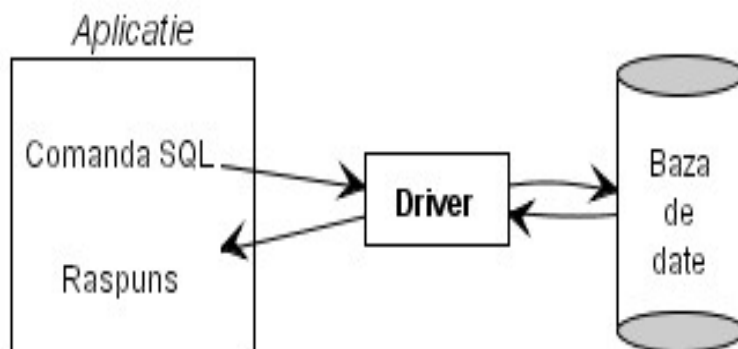
Accesul la baza de date

Se face prin intermediul unui **driver** specific tipului respectiv de SGBD.

Limbajul SQL

(Structured Query Language)

Limbaj **standardizat** pentru interogarea și actualizarea informațiilor din baze de date relaționale.



JDBC

Java Database Connectivity este tehnologia Java de acces la baze de date.

Este **independentă** de tipul bazei de date.

java.sql - nucleul tehnologiei JDBC.

Facilități

1. **Stabilirea unei conexiuni** cu o bază de date.
2. **Efectuarea de secvențe SQL.**
3. **Prelucrarea rezultatelor** obținute.

Conectarea la o bază de date

1. Inregistrarea unui driver
2. Realizarea unei conexiuni

Conexiune (sesiune) = context prin care sunt trimise secvențe SQL și primite rezultate. Într-o aplicație pot exista simultan mai multe conexiuni la baze de date diferite sau la aceeași bază.

Clase și interfețe:

- **DriverManager**
- **Driver**
- **DriverPropertyInfo**
- **Connection**

Inregistrarea unui driver

Incărcarea în memorie a clasei ce implementează driver-ul.

a. **DriverManager:**

```
DriverManager.registerDriver(new TipDriver());
```

b. **Class.forName**

```
Class.forName("TipDriver");
```

```
Class.forName("TipDriver").newInstance();
```

c. **jdbc.drivers**

– De la linia de comandă:

```
java -Djdbc.drivers=TipDriver Aplicatie
```

– Din program:

```
System.setProperty("jdbc.drivers", "TipDriver");
```

Specificarea unei baze de date

JDBC URL - modalitatea de a identifica o BD:

jdbc:sub-protocol:identificator

Sub-protocol: tipul de driver
odbc, oracle, sybase, db2, etc.

Identificatorul bazei de date

`jdbc:odbc:test`

`jdbc:odbc:test;UID=duke;PWD=java`

`jdbc:oracle:thin@persistentjava.com:1521:test`

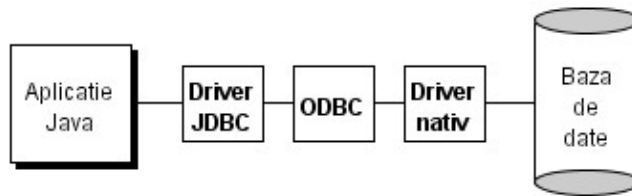
`jdbc:sybase:test`

`jdbc:db2:test`

SQLException - "no suitable driver"

Tipuri de drivere

Tip 1. JDBC-ODBC Bridge

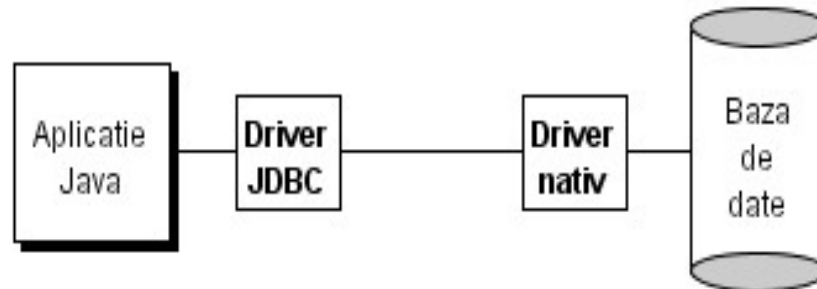


ODBC (Open Database Conectivity)

- identicator DSN (Data Source Name)
- simplu de utilizat
- nu este portabilă
- viteză de execuție slabă

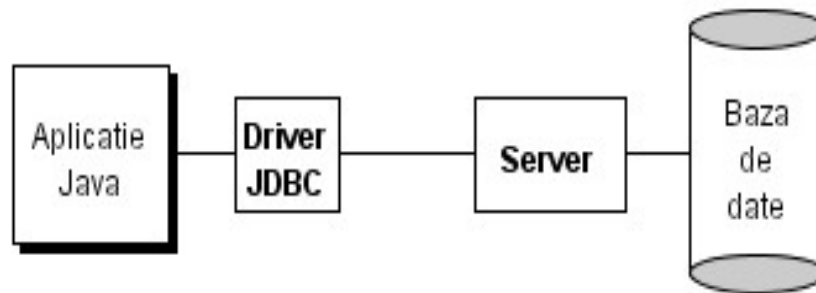
```
sun.jdbc.odbc.JdbcOdbcDriver  
jdbc:odbc:identicator
```

Tip 2. Driver JDBC - Driver nativ



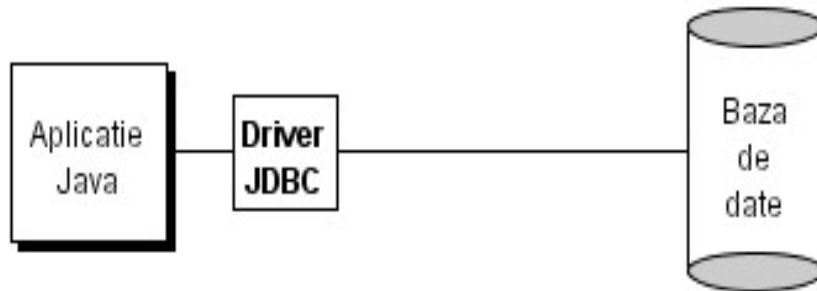
Acest tip de driver transformă cererile JDBC direct în apeluri către driverul nativ al bazei de date, care trebuie instalat în prealabil. Clase Java care implementează astfel de drivere pot fi procurate de la producătorii de SGBD-uri, distribuția standard J2SDK neincluzând nici unul.

Tip 3. Driver JDBC - Server



Acest tip de driver transformă cererile JDBC folosind un protocol de rețea independent, acestea fiind apoi transformate folosind o aplicație server într-un protocol specific bazei de date. Introducerea serverului ca nivel intermediar aduce flexibilitate maximă în sensul că vor putea fi realizate conexiuni cu diferite tipuri de baze, fără nici o modificare la nivelul clientului.

Tip 4. Driver JDBC nativ



Acest tip de driver transformă cererile JDBC direct în cereri către baza de date folosind protocolul de rețea al acesteia. Această soluție este cea mai rapidă, fiind preferată la dezvoltarea aplicațiilor care manevrează volume mari de date și viteza de execuție este critică.

Realizarea unei conexiuni

O conexiune este reprezentată printr-un obiect de tip **Connection**

Crearea unei conexiuni se realizează prin metoda:

DriverManager.getConnection

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

Inchiderea unei conexiuni se realizează prin metoda **close**.

Stabilirea unei conexiuni folosind driverul JDBC-ODBC

```
String url = "jdbc:odbc:test" ;  
// sau url = "jdbc:odbc:test;UID=duke;PWD=java" ;  
try {  
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");  
} catch(ClassNotFoundException e) {  
    System.err.print("ClassNotFoundException: " + e) ;  
    return ;  
}
```

```
Connection con ;  
try {  
    con = DriverManager.getConnection(  
        url, "duke", "java");  
} catch(SQLException e) {  
    System.err.println("SQLException: " + e);  
} finally {  
    try{  
        con.close ;  
    } catch(SQLException e) {  
        System.err.println(SQLException: " + e) ;  
    }  
}
```

Stabilirea unei conexiuni folosind un driver MySQL

```
String url = "jdbc:mysql://localhost/test" ;  
// sau url = "jdbc:mysql://localhost/test?user=duke&pas  
try {  
    Class.forName("com.mysql.jdbc.Driver") ;  
} catch(ClassNotFoundException e) {  
    ...
```

O conexiune va fi folosită pentru:

- Crearea de secvențe SQL utilizate pentru interogarea sau actualizarea bazei.
- Aflarea unor informații legate de baza de date (meta-date).
- Controlul tranzacțiilor din memorie către baza de date prin metodele **commit**, **rollback**, **setAutoCommit**.

Efectuarea de secvențe SQL

- **Statement**
- **PreparedStatement**
- **CallableStatement**

Comenzi SQL uzual folosite:

- **SELECT**
- **INSERT, UPDATE, DELETE**
- **CREATE, ALTER, DROP**
DDL (Data Definition Language)
- **CALL**

ResultSet - obținerea rezultatelor

Interfața Statement

createStatement

```
Connection con = DriverManager.getConnection(url);  
Statement stmt = con.createStatement();
```

1. executeQuery - interogări

```
String sql = "SELECT * FROM persoane";  
ResultSet rs = stmt.executeQuery(sql);
```

2. executeUpdate - actualizări

```
String sql =  
    "DELETE FROM persoane WHERE cod > 100";  
int linii = stmt.executeUpdate(sql);  
// Nr de articole care au fost afectate (sterse)  
  
sql = "DROP TABLE temp";  
stmt.executeUpdate(sql);  
// Returneaza 0
```

3. execute

```
String sql = "comanda SQL necunoscuta";
stmt.execute(sql);
while(true) {
    int rowCount = stmt.getUpdateCount();
    if(rowCount > 0) {
        // Este o actualizare datelor
        System.out.println("Linii afectate:" + rowCount);
        stmt.getMoreResults();
        continue;
    }
    if(rowCount = 0) {
        // Comanda DDL sau nici o linie afectata
        System.out.println("DDL sau 0 actualizari");
        stmt.getMoreResults();
        continue;
    } // rowCount este -1
    // Avem unul sau mai multe ResultSet-uri
    ResultSet rs = stmt.getResultSet();
    if(rs != null) { // Proceseaza rezultatul
        stmt.getMoreResults();
        continue;
    }
    break; // Nu mai avem nici un rezultat
}
```

Integrarea cod sursă - SQL

Clasic

```
int cod = 100;
String nume = "Popescu";

String sql = "SELECT salariu FROM persoane" +
  " WHERE cod=" + cod +
  " OR nume='" + nume + "'";
ResultSet rs = stmt.executeQuery(sql);
...
```

Embedded SQL

```
int cod = 100;
String nume = "Popescu";
double salariu;

SELECT salariu
  INTO :salariu
  WHERE cod=:cod OR nume=:nume;
```

Interfața PreparedStatement

Instanțele de tip `PreparedStatement` conțin secvențe SQL care au fost **deja compilate**, scopul fiind **creșterea vitezei** de execuție.

Crearea

```
Connection con = DriverManager.getConnection(url);  
String sql = "UPDATE persoane SET nume=? WHERE cod=?";  
Statement pstmt = con.prepareStatement(sql);
```

Trimiterea parametrilor

```
pstmt.setString(1, "Ionescu");  
pstmt.setInt(2, 100);
```

Execuția

`executeQuery`, `executeUpdate` sau
`execute`

```
String sql = "UPDATE persoane SET nume=? WHERE cod=?";  
Statement pstmt = con.prepareStatement(sql);  
pstmt.setString(1, "Ionescu");  
pstmt.setInt(2, 100);  
pstmt.executeUpdate();
```

```
pstmt.setString(1, "Popescu");  
pstmt.setInt(2, 200);  
pstmt.executeUpdate();
```

```
sql = "SELECT * from persoane WHERE cod >= ?";  
pstmt = con.prepareStatement(sql);  
pstmt.setInt(1, 100);  
ResultSet rs = pstmt.executeQuery();
```

Tipuri JDBC

Sunt definite în clasa **Types**

Metoda **setObject** permite specificarea explicită a tipului

```
pstmt.setObject(1, "Ionescu", Types.CHAR);  
pstmt.setObject(2, 100, Types.INTEGER); // sau doar  
pstmt.setObject(2, 100);
```

Metoda **setNull**

```
pstmt.setNull(1, Types.CHAR);  
pstmt.setInt(2, null);
```

Date de dimensiuni mari

setBytes sau setString

setBinaryStream, setAsciiStream
și setUnicodeStream

```
File file = new File("date.txt");
int fileLength = file.length();
InputStream fin = new FileInputStream(file);
java.sql.PreparedStatement pstmt =
    con.prepareStatement(
        "UPDATE fisiere SET continut = ? " +
        "WHERE nume = 'date.txt'");
pstmt.setUnicodeStream (1, fin, fileLength);
pstmt.executeUpdate();
```

Interfața CallableStatement

Apelarea procedurilor stocate

Crearea

```
Connection con = DriverManager.getConnection(url);
CallableStatement cstmt = con.prepareCall(
    "{call proceduraStocata(?, ?)}");
```

Trimiterea parametrilor și execuția

```
CallableStatement cstmt = con.prepareCall(
    "{call procedura(?, ?)}");
pstmt.setString(1, "Ionescu");
pstmt.setInt(2, 100);
```

```
CallableStatement cstmt = con.prepareCall(
    "{call calculMedie()}");
cstmt.registerOutParameter(1, java.sql.Types.FLOAT);
cstmt.executeQuery();
float medie = cstmt.getDouble(1);
```

Obținerea și prelucrarea rezultatelor

Interfața ResultSet

```
Statement stmt = con.createStatement();  
String sql = "SELECT cod, nume FROM persoane";  
ResultSet rs = stmt.executeQuery(sql);
```

Structura datelor obiectului *rs*:

cod	nume
100	Ionescu
200	Popescu

Un **ResultSet** înapsează și **meta-informații** legate de interogare.

Extragerea datelor

Parcurgem tabelul linie cu linie și din fiecare valorile de pe coloane. Coloanele sunt numerotate de la stânga la dreapta, începând cu 1.

```
String sql = "SELECT cod, nume FROM persoane";
ResultSet rs = stmt.executeQuery(sql);
while (rs.next()) {
    int cod = r.getInt("cod");
    String nume = r.getString("nume");
    /* echivalent:
    int cod = r.getInt(1);
    String nume = r.getString(2);
    */
    System.out.println(cod + ", " + nume);
}
```

Cursoare modificabile

```
Statement stmt = con.createStatement(  
    ResultSet.TYPE_SCROLL_INSENSITIVE,  
    ResultSet.CONCUR_UPDATABLE);  
String sql = "SELECT cod, nume FROM persoane";  
ResultSet rs = stmt.executeQuery(sql);
```

Metode suplimentare

- absolute
- updateXXX
- updateRow
- moveToInsertRow
- insertRow
- moveToCurrentRow
- deleteRow

supportsPositionedUpdate/Delete

Exemplu simplu

Listing 1: Exemplu simplu de utilizare JDBC

```
import java.sql.*;

public class TestJdbc {
    public static void main (String[] args) {
        String url = "jdbc:mysql://localhost/test" ;
        try{
            Class.forName("com.mysql.jdbc.Driver");
        } catch(ClassNotFoundException e) {
            System.out.println("Eroare incarcare driver!\n" + e);
            return;
        }
        try{
            Connection con = DriverManager.getConnection(url);

            // Golim tabelul persoane
            String sql = "DELETE FROM persoane";
            Statement stmt = con.createStatement();
            stmt.executeUpdate(sql);

            // Adugam un numar de persoane generate aleator
            // Tabelul persoane are coloanele (cod, nume, salariu)
            int n = 10;
            sql = "INSERT INTO persoane VALUES (?, ?, ?)";
            PreparedStatement pstmt = con.prepareStatement(sql);
            for(int i=0; i<n; i++) {
                int cod = i;
                String nume = "Persoana" + i;
                double salariu = 100 + Math.round(Math.random() *
                    900);
                // salariul va fi intre 100 si 1000
                pstmt.setInt(1, cod);
                pstmt.setString(2, nume);
                pstmt.setDouble(3, salariu);
                pstmt.executeUpdate();
            }

            // Afisam persoanele ordonate dupa salariu
            sql = "SELECT * FROM persoane ORDER BY salariu";
            ResultSet rs = stmt.executeQuery(sql);
            while (rs.next())
```

```
        System.out.println(rs.getInt("cod") + ", " +
                            rs.getString("nume") + ", " +
                            rs.getDouble("salariu"));

// Calculam salariul mediu
sql = "SELECT avg(salariu) FROM persoane";
rs = stmt.executeQuery(sql);
rs.next();
System.out.println("Media: " + rs.getDouble(1));

// Inchidem conexiunea
con.close();

} catch(SQLException e) {
    e.printStackTrace();
}
}
}
```

Lucrul cu meta-date

Interfața `DatabaseMetaData`

Un obiect de tip `DatabaseMetaData` oferă diverse informații (*meta-date* - ”date despre date”) legate de baza respectivă, cum ar fi:

- tabele
- proceduri stocate
- capacitățile conexiunii
- gramatica SQL suportată, etc.

Metoda `getMetaData` din clasa `Connection`.

Listing 2: Folosirea interfeței DatabaseMetaData

```
import java.sql.*;

public class TestMetaData {
    public static void main (String[] args) {
        String url = "jdbc:odbc:test";
        try{
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        } catch(ClassNotFoundException e) {
            System.out.println("Eroare incarcare driver!\n" + e);
            return;
        }
        try{
            Connection con = DriverManager.getConnection(url);
            DatabaseMetaData dbmd = con.getMetaData();
            ResultSet rs = dbmd.getTables(null, null, null, null);
            while (rs.next())
                System.out.println(rs.getString("TABLE_NAME"));
            con.close();
        } catch(SQLException e) {
            e.printStackTrace();
        }
    }
}
```

Interfața ResultSetMetaData

Meta-datele unui **ResultSet** reprezintă informațiile despre rezultatul conținut în acel obiect cum ar fi numărul coloanelor, tipul și denumirile lor, etc.

```
ResultSet rs = stmt.executeQuery("SELECT * FROM tabel")
ResultSetMetaData rsmd = rs.getMetaData();
// Aflam numarul de coloane
int n = rsmd.getColumnCount();

// Aflam numele coloanelor
String nume[] = new String[n];
for(int i=0; i<n; i++)
    nume[i] = rsmd.getColumnName(i);
```