C13

SQL si JDBC

- Baze de date, concepte, modele de date
- Structured Query Language:
  - DDL, DML
  - exemple
- Java DataBase Connectivity:
  - motivare
  - integrare
  - arhitectura
  - utilizare

Obiective

După parcurgerea acestui curs ar trebuie sa puteţi:

- înţelege conceptele vehiculate in baze de date si cele legate de modelul de date relational;
- intelege bazele si limitele limbajului SQL si cum se utilizeaza in manipularea datelor;
- intelege rolul si necesitatea utilizarii limbajelor de nivel inalt in scrierea aplicatiilor ce opereaza cu baze de date;
- intelege si utiliza clasele cu interfetele JDBC pentru accesul la bazele de date din Java.

Concepte legate de bazele de date relational si modele de date

- *Relaţie (relation)* - în **matematică**, relaţia `R,` este o submulţime a produsului cartezian a două mulţimi `(A, B), R : A x B`. Dacă `(a, b)` este un element din `R` notaţia `a R b` are semnificaţia de "`a` este în relaţia `R` cu `b`" (o notaţie echivalentă folosită, mai des, în teoria bazelor de date relaţionale este `R(a,b)`). O relaţie poate fi *reflexivă* `(a R a)`, *simetrică* `( a R b => b R a)`, *tranzitivă* `(a R b & b R c => a R c)`, *antisimetrică* `(a R b & b R a => a = b)` sau *totală* `(a R b sau b R a)`. În **bazele de date**, relaţia, este *un tabel* dintr-o bază de date relaţională.
- *Bază de date (database)* - una sau mai multe mulţimi de date persistente asociate cu o aplicaţie pentru actualizarea şi interogarea datelor;
- *Persistenţă* - proprietate a unei aplicaţii de a crea obiecte şi date care continuă să existe şi să-şi păstreze valorile între două execuţii ale aplicaţiei.
- *Model de date (data model)* - formalism matematic cu notaţii pentru descrierea structurilor de date şi a mulţimii operatorilor folosiţi pentru manipularea şi validarea datelor;
- *Model de date relaţional (relational model)* - un model de date introdus de E. F. Codd în 1970. În acest model datele sunt organizate în tabele. Mulţimea numelor coloanelor se numeşte schema tabelului. Datele pot fi manipulate folosind algebra relaţională. Limbajul SQL este implementarea unei astfel de algebre.
- *SQL (Structured Query Language)* - un limbaj neprocedural, de interfaţă, între utilizator şi bazele de date relaţionale dezvoltat de IBM în anii '70. Diferenţa esenţială între SQL şi alte limbaje este aceea că în SQL instrucţiunile specifică ce operaţii se vor face cu datele şi nu modul în care se realizează acestea.
- *Bază de date relaţională (relational database, relational database management system)* - o bază de date ce utilizează modelul de date dezvoltat de E. F. Codd. O bază de date relaţională permite definirea structurilor de date, operaţiile de stocare, de extragere a datelor şi a constrângerilor de integritate. Într-o bază de date relaţională (BDR), datele şi toate relaţiile între date sunt organizate în tabele. Un tabel este o colecţie de înregistrări (rânduri) unde fiecare înregistrare a tabelului conţine aceleaşi câmpuri. Anumite câmpuri pot fi desemnate chei. În acest caz viteza de căutare a unor valori specifice acelui câmp va creşte datorită folosirii de indexuri. Între înregistările unor tabele diferite se pot realiza asocieri pe baza valorilor comune ale unui anumit câmp particular din fiecare tabel.
- *Cheie primară* - numărul minim de câmpuri care au fost alese pentru a identifica unic fiecare rând dintr-un tabel.
- *Constrângere de integritate* - o regulă care trebuie să rămână adevărată pentru ca baza de date să-şi păstreze integritatea. De exemplu, într-o bază de date genealogică fiecare individ

trebuie să fie copilul unor părinţi. De asemenea, individul nu poate avea decât doi părinţi naturali.

- *Index* - o secvenţă de perechi (cheie, pointer) în care fiecare pointer poantează la o înregistrare din baza de date care conţine valoarea cheii într-un câmp particular. Indexul este sortat pe baza valorilor de chei şi permite căutarea rapidă a unei valori particulare de cheie.
- *Tabel* - un obiect al bazei de date care stochează datele. Este format din mai multe coloane. Fiecare coloană are asociat un tip de dată. Tipul coloanei se foloseşte pentru manipularea corectă a conţinutului.
- *Normalizare* - o relaţie dintr-o bază de date relaţională se zice că este într-o formă normală dacă satisface un anumit grup de constrângeri. Lucrarea iniţială a lui Codd definea trei forme normale.
- *Tuplă* - un obiect de date ce conţine una sau mai multe componente. Componentele unei tuple pot fi de tipuri diferite. Câteva exemple de tuple sunt: (1,2), ("Tuple", Da), (a, (x, y), c).
- *Tranzacţie* - o unitate logică de interacţiune cu o BDR. Aceasta trebuie tratată într-o manieră coerentă şi sigură, independent de alte tranzacţii. Pentru o tranzacţie se garantează, fie realizarea ei completă, fie nerealizarea ei. Dacă o eroare conduce la realizarea parţială a tranzacţiei, tranzacţia poate fi "derulată înapoi" pentru a opri lăsarea bazei de date într-o stare de inconsistenţă.
- *Interogare (query)* - acţiune de extragere a unor informaţii din baza de date.

Conceptul de dată

Pentru cei care au deja experienţă în domeniul calculatoarelor, noţiunea de dată este asociată cu numere, şiruri de caractere, imagini, sunete ce pot fi stocate şi prelucrate pe calculator, eventual transmise mai departe prin reţea. Privind însă conceptul de dată general, data, reprezintă o reflectare, parţială, a realităţii. Deseori, cuvântul de dată apare în contextul conceptului de model. Necesitatea modelului apare datorită complexităţii lumii pe care încercăm să o surprindem. Abundenţa detaliilor ne depăşeşte. Din acest motiv, realitatea va fi simplificată de om, prin crearea unor modele. Modelul se obţine prin eliminarea detaliilor şi păstrarea unor caracterstici minimale din realitate, fără a pierde însă veridicitatea modelului (care dintre carateristici sunt păstrare, deoarece se consideră a fi semnificative, respectiv de ce şi care anume se elimină asta este o altă carte). Datele reprezintă o măsură cantitativă a procesului de evoluţie a caracteristicilor considerate semnificative pentru model (aceste caracteristici descriu modelul). Generalizând, datele reprezintă o colecţie de observaţii pe marginea unor evenimente pe care le considerăm importante.

Datele în sine nu au nici o semnificaţie (ele nu au un înţeles). Ele pot însă deveni informaţie în urma asocierii directe acestora cu anumite caracteristici. Datele vor fi astfel încărcate cu semnificaţie ca urmare a interpretării lor într-un context particular. Această interpretare se poate face de către om sau, automat, de către un sistem de calcul. Daca considerăm numărul întreg “1100”, el este dată dar nu este informaţie. În contextul “Salariul meu este de 1100 RON”, data “1100” are deja un înţeles particular (este valoarea remunerării mele), devenind informaţie. În contextul “Salariul meu de 1100 RON este acceptabil.”, data “1100” permite trecerea la cunoaştere. Deoarece eu cunosc salariile unei mari majorităţi, sunt în stare să calculez un salar mediu şi să mă plasez undeva, în raport cu acesta. Procesul de plasare la nivelul de “acceptabil” are la bază un grup de prelucrări iar rezultatul de “acceptabil” are la bază un proces deductiv. Foarte pe scurt, datele reprezintă materialul primar,

informaţia reprezintă date + înţeles, iar cunoaşterea este informaţie + prelucrare.

Scopul colectării datelor este obținerea de informații sau de cunoaștere ca urmare a unui proces de deducție bazat pe experientă sau pe reguli ce poate fi implementate la nivelul unui sistem de calcul. În bazele de date, data, va fi organizată sub forma unui singur articol de informație (o singură valoare). Deoarece în procesul de colectare a datelor pot să apară erori, termenul de integritate a datelor definește mecanismele de prevenire a erorilor.

Modelul de date relaţional

Se numeşte model de date, un formalism matematic cu notaţii pentru descrierea structurilor de date şi a mulţimii operatorilor folosiţi pentru manipularea şi validarea datelor. Modelul de date relaţional este introdus de E.F. Codd în 1970. În acest model datele sunt structurate (organizate) în relaţii care au nume. Mulţimea numelor coloanelor se numeşte schema relaţiei. Structura de date logică numită relație este bidimensională, fiind cunoscută sub numele de tabel, în baza de date fizică. Cele două dimensiuni ale structurii sunt denumite rânduri şi coloane. Datele modelului se pot modifica folosind operatorii algebrei relaţionale (limbajul SQL este implementarea unei astfel de algebre), ei asigurând selectarea, inserarea, modificarea şi ştergerea datelor din tabele, iar restricţiile de integritate asigură păstrarea consistenţei datelor.

Conceptul de redundanţă

Redundanţa datelor apare atunci când o anumită dată este stocată în mai multe tabele ale bazei de date. Acest surplus de date necesită o atenţie specială atunci când se aduc modificări asupra conţinutului valorilor stocate în tabele. Dacă valorea unei date redundante este modificată, atunci va trebui să ne asigurăm că toate copiile respectivei date au fost aduse la nouă valoare. În această situaţie se zice că valoarea respectivă este consistentă în baza de date. Dacă nu toate copiile au fost actualizate la noua valoare se zice că data este inconsistentă. Se zice că o bază da date este în stare consistentă dacă toate datele acesteia sunt consistente, altfel este în stare inconsistentă.

Terminologii alternative ale modelului relaţional

Termenii folosiţi în modelul de date relaţionali sunt, deseori, producători de confuzie. Deoarece modelul are ca sursă algebra relaţională, matematicienii preferă să folosească anumite denumiri. Proiectanţii bazelor de date au şi ei termeni specifici, dar echivalenţi cu cei din algebra. În final, administratorii de sistem folosesc şi ei propriul lor jargon atunci când realizează operaţii de întreţinere a bazelor de date. Tabelul care urmează îşi propune să prezinte termenii alternativi folosiţi de cele trei categorii de persoane care folosesc acest model.

Algebra relaţională	Proiectant BDR	Administrator de sistem
Relaţie	Tabel	Fişier
Tuplă	Rând	Înregistrare
Atribut	Coloană	Câmp

Atributele reprezintă date elementare care sunt nume ale coloanelor unei relaţii. De exemplu, relaţia `chirie(IDClient, NumeClient)` conţine atributele `IDClient`,`NumeClient`. Valorile actuale pentru

aceste atribute ale relației se stochează în **tuple** sau rânduri ale tabelului. Atributele se grupează pe baza dependenței de valorile din cheia primară. O **cheie primară** (CP) este un atribut, sau un grup de atribute, care indentifică unic un rând din tabel. Un tabel poate avea o singură cheie primară. Deoarece valorile din CP se folosesc pentru identificare, ele nu pot fi vide (nule). Asocierea între două relaţii se face, tipic, printr-un atribut comun celor două relații. Atributul comun este, de obicei, CP într-un tabel şi **cheie străină** (CS) în celălalt. Regulile de **integritate referenţială** dictează valorile CS dintr-o relație în raport cu valorile CP din cealaltă relație.

Proiectarea unei baze de date relaționale se face pe baza **regulilor de normalizare** care dictează apartenența atributelor la relații. Aceste reguli **nu fac obiectul acestui curs însă** folosirea modelului de date relațional normalizat asigură minimizarea redundanței datelor împreună cu protecția contra anomaliilor de inserare, ştergere şi actualizare care apar ca urmare a definirii unor relații incorecte.

Elemente de SQL

SQL este un limbaj neprocedural (este implementarea unei algebre relaţionale) dezvoltat în baza răspândirii modelului relaţional. În ultimii ani a devenit limbajul standard pentru BDR. Obiectivele limbajului sunt:

- crearea bazelor de date şi a relaţiilor;
- realizarea operaţiilor de întreţinere a BDR;
- realizarea interogărilor simple şi complexe.

Ca urmare a standardizării ISO limbajul SQL are două componente majore:

- limbajul pentru definiţia datelor (*Data Definition Language - DDL*): pentru definirea structurii bazei de date şi controlul accesului la date;
- limbajul pentru manipularea datelor (*Data Manipulation Language - DML*): pentru extragerea şi actualizarea datelor.

O *instrucţiune SQL* este formată din *cuvinte rezervate* şi *cuvinte definite de utilizator*. Cuvintele rezervate (numite şi cuvinte cheie) sunt partea fixată a limbajului deoarece au o semnificaţie predefinită. Ele trebuie scrise fără a fi despărţite pe mai multe linii. Cuvintele definite de utilizator sunt "inventate" de el şi reprezintă nume ale obiectelor bazei de date cum sunt tabelele, coloanele etc. Cuvintele unei instrucţiuni SQL se scriu conform mulţimii regulilor de sintaxă specifice limbajului SQL. Multe dintre dialectele SQL folosesc caracterul "punct şi virgulă" ";" pe post de terminator de instrucţiune SQL, însă conform standardului, acestea sunt opţionale.

Pentru exemplele care vor fi dezvoltate în continuare este suficientă prezentarea a numai patru dintre instruncţiunile DML din SQL:

1. `SELECT` - pentru interogarea bazei de date;
2. `INSERT` - pentru inserarea datelor într-un tabel;
3. `UPDATE` - pentru actualizarea datelor unui tabel;
4. `DELETE` - pentru ştergerea datelor unui tabel.

Instrucţiunea SELECT

Permite extragerea şi afişarea datelor din unul sau mai multe tabele.

SELECT cu un singur tabel

Pentru extragerea tuturor numelor de proiecte stocate în câmpul `NumePr` din tabelul **Proiecte** scriem:

```
SELECT NumePr FROM Proiecte;
```

Pentru extragerea numelor de proiecte şi a sumei plătite pe proiect, stocată în câmpul `Suma`, din acelaşi tabel se scrie:

```
SELECT NumePr, Suma FROM Proiecte;
```

Deoarece multe interogări SQL folosesc toate câmpurile unui tabel. Folosirea unui asterisc "*" este o formă prescurată de scriere pentru "toate câmpurile".

```
SELECT * FROM Proiecte;
```

Atunci când dorim să impunem condiţii cu privire la rândurile extrase se va folosi clauza `WHERE`. De exemplu, pentru extragerea proiectelor din tabelul **Proiecte** care au o suma mai mare de 500000 în câmpul `Suma` se va scrie:

```
SELECT * FROM Proiecte WHERE Suma > 500000;
```

Dacă dorim ca rândurile sortate să fie în ordinea crescătoare alfabetică a proiectelor vom folosi clauza `ORDER BY` astfel

```
SELECT * FROM Proiecte ORDER BY NumePr;
```

Pentru sortarea în ordinea descrescătoare a numelor de proiecte scriem:

```
SELECT * FROM Proiecte ORDER BY NumePr DESC;
```

SELECT cu mai multe tabele

Dacă datele de extras sunt stocate, ca urmare a normalizării, în mai multe tabele, una dintre metodele pentru conectarea tabelelor pe baza unui câmp comun foloseşte instrucţiunea `JOIN`. Să presupunem că dorim să extragem toate proiectele şi tipurile membrilor pe baza câmpurilor `IDPr` din **Proiecte** şi `IDPr` din **Membri**. `IDPr` este CP în **Proiecte** şi CS în **Membri**. Între proiecte şi membri este o *relaţie unu-la-mulţi*, adică unui rând din tabelul **Proiecte** pot să-i corespundă mai multe rânduri ale tabelului **Membri**. Instrucţiunea `INNER JOIN` face ca din cele două tabele să fie extrase doar rândurile care au valori comune în câmpurile de conectare.

```
SELECT NumePr.Proiecte, Membri.Calitate
FROM Proiecte INNER JOIN Membri ON Proiecte.IDPr = Membri.IDPr;
```

Instrucţiunea INSERT

Instrucţiunea `INSERT` permite adăugarea unui rând într-un tabel. Una dintre formele ei este:

```
INSERT INTO NumeTable (lista de câmpuri)
VALUES (lista de valori)
```

Dacă dorim să inserăm un nou proiect în tabelul **Proiecte** scriem:

```
INSERT INTO Proiecte (IDPr, NumeProiect, Suma1)
VALUES ('PR4', Proiect nou', 100000);
```

Instrucţiunea UPDATE

Instrucţiunea `UPDATE` permite modificarea conţinutului unui rând existent dintr-un tabel. Forma ei

este:

```
UPDATE NumeTabel
SET NumeColoana1=Valoare1 [,NumeColoana2=Valoare2 ...]
[WHERE Conditie]
```

Pentru a creşte cu 20% valoarea din câmpul `Suma` ale tuturor proiectelor din tabelul **Proiecte** scriem:

```
UPDATE Proiecte SET Suma=Suma*1.20;
```

Pentru a creşte valoarea din câmpul `Suma` cu 20% a proiectului cu numele "`Proiect1`" din tabelul **Proiecte** scriem:

```
UPDATE Proiecte
SET Suma=Suma*1.20
WHERE NumePr="Proiect1";
```

Instrucţiunea DELETE

Instrucţiunea permite ştergerea unor rânduri ale unui tabel. Forma ei este:

```
DELETE FROM NumeTabel
[WHERE Conditie]
```

Pentru a şterge rândul corespunzător proiectului cu numele "`Proiect1`" din tabelul **Proiecte** scriem:

```
DELETE FROM Proiecte
WHERE NumePr="Proiect1";
```

JDBC

În 1996, Sun, scoate pe piaţă prima variantă de interfaţă de programare (Application Programming Interface - API) a bazelor de date numită Java DataBase Connectivity sau JDBC. Aceasta permitea conectarea, interogarea şi actualizarea datelor pe baza limbajului SQL (Structured Query Language).

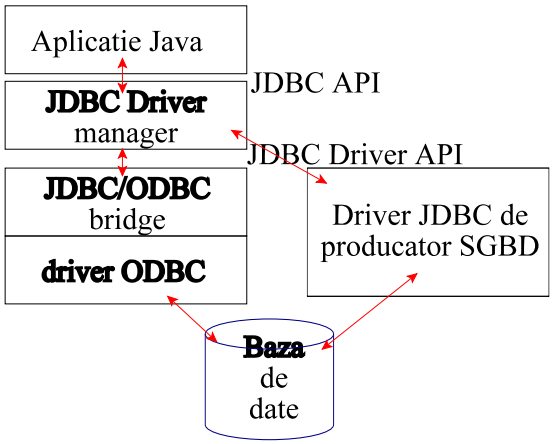
Java şi JDBC aveau, încă de pe atunci, avanatajul, în comparaţie cu alte medii de programare al bazelor de date, independenţa de platformă şi de producătorul SGDB-ului. Aceeaşi bază de date scrisă în Java se putea rula pe servere Windows sau Solaris deoarece codul Java rulează identic pe ambele platforme. Datele puteau fi transferate de exemplu de pe Miscrosoft SQL Server pe Oracle ca urmare a folosirii a unui limbaj comun, SQL, pentru definiţia şi manipularea datelor. Până atunci, aplicaţiile cu baze de date se scriau într-un limbaj proprietar al producătorului de SGBD cumpărătorul fiind limitat la a lucra tot timpul cu acelaşi software.

JDBC-ul suferă pe parcursul anilor mai multe extinderi însă nici după ultimul release nu are instrumente pentru dezvoltarea de interfeţe grafice cu bazele de date. Aceste generatoare de formulare, interogări sau rapoarte sunt dezvoltate de diferite firme specializate numai pe acest domeniu.

Arhitectura JDBC

Producătorii de SGBD-uri au fost de acord foarte repede ca Sun să folosească un protocol de reţea unic pentru accesul la baze de date, cu condiţia ca acel protocol să fi a lor. Cum piaţa avea deja producători de SGBD-uri mulţi şi serioşi Sun era în impas. Noroscul vine din partea producătorilor de SGBD-uri care se pun de acor cu privire asupra faptului că Sun să creeze API-uri pure de Java pentru acces prin SQL la bazele de date împreună cu un manager de drivere care să încarce drivere externe, fabricate de firme terţe sau de către producătorii SGBD-urilor, prin care să se realizeze conexiunea la baza de date. Apoi, printr-un mecanism simplu de înregistrare al driver-elor în manager-ul de driver, toate driver-ele să respecte aceleaşi cerinţe şi opereze la fel. Ca urmare a acestor cerinţe două intefeţe au fost create, JDBC API pentru programatori şi JDBC Driver API pentru producătorii de driver-e. Aplicaţiile comunică prin API cu manager-ul de driver-e care foloseşte, mai departe, driver-ul dat de producător pentru a comunica cu baza de date.

Drivere-ele JDBC sunt organizate pe următoarele categorii:

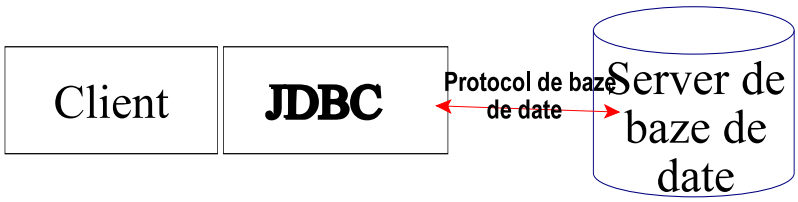


- driver tip 1: traduce JDBC în ODBC şi se bazează pe driver-ul de ODBC pentru a comunica cu baza de date. Sun dă un singur driver de acest fel JDBC/ODBC bridge în JDK. Necesită configurarea corectă a driver-ului de ODBC.
- driver de tip2: este scris paţial în Java şi paţial în cod nativ, comunică API-urile client ale SGBD-ului. Utilizarea unui astfel de driver obligă la instalarea unei biblioteci Java dependente de platofrmă.
- driver de tip 3: este o bibliotecă Java pură de tip client ce foloseşte un protocol independent de baza de date pentru comunicare cu baza de date utilizând o componentă serve ce traduce toate cererile în cerei dependente de baza de date. Codul dependent de baza de date este stocat pe server.
- driver tip 4: o biblioteca Java pură ce traduce cererile JDBC direct în cereri specifice protocolului bazei de date.

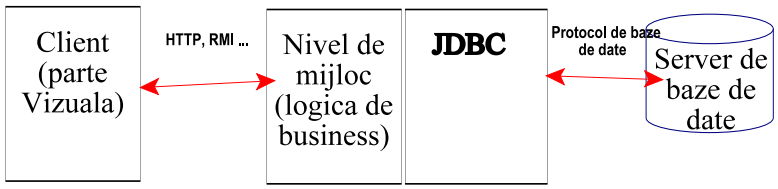
Majoritatea producătorilor de baze de date furnizează driver-ele de tip 3 sau 4.

Utilizari tipice ale JDBC

Modelul tradiţional este cel **client/server** cu interfaţă grafică complexă la client şi o bază de date stocată pe un server.



Azi, arhitectura client/server este depăşită fiind înlocuită cu **arhitectură pe 3 nivele** (3 tier model) sau chiar de arhitecturi mai avansate pe multi-nivele. În acest model clientul nu acceseaza baza de date direct. El accesează direct doar un nivel intermediar, de mijloc, care derulează comunicaţia cu baza de date. Este separată partea de pezentare, vizuală (rulată de client) de partea de logică (middle tire) şi de datele fizice (din baza de date). Ca urmare a acestei separări este posibil accesul la aceleaşi date cu aceleşi regului de business dar de pe clienţi multipli.



Comunicaţia între client şi nivelul de mijloc se face prin HTTP dacă clientul este un navigator de web sau prin RMI când se foloseşte o aplicaţie Java. JDBC gestionează comunicaţia între nivelul de mijloc şi baza de date. Aceste model are numeroase variaţii, în particular Java 2 Enterprise defineşte o structură pentru server-ul de aplicaţii ce gestionează modulele de cod numite Enterprise JavaBeans ce asigură servicii ce accesare rapidă, securizată a bazei de date.

Concepte de baza in programarea JDBC

Programarea cu JDBC nu diferă de programarea clasică cu obiecte din Java. Obiecte se creează pe bază de clase JDBC, eventual dacă este cazul se moştenesc şi se extind.

Identificarea bazei de date
Conectarea la o bază de date presupune specificarea sursei împreună cu parametrii adiţionali cum sunt portul de comunicaţie sau valori de atribute specifice de driver (mai alesc în cazul celor ODBC). URL-ul bazei de date specifică sursa de date. Printre altele aceasta trebuie sa conţina numele bazei de date şi locul în care se află aceasta.

```
private static final String accessDBURLPrefix = "jdbc:odbc:Driver={Microsoft Access Driver (*.mdb)};DBQ=";

private static final String accessDBURLSuffix = ";DriverID=22;READONLY=false}";

pivate static final String filename = "D:\\work\\test.mdb";

filename = filename.trim();

String databaseURL = accessDBURLPrefix + filename + accessDBURLSuffix;
```

Realizarea conexiunii
Realizarea conexiunii se poate realiza doar dacă driver-ul JDBC pentru accesul la baza de date MS Access, care în exemplul prezentat este ODBC, este configurat corect. Apoi acesta trebuie înregistrat cu mâna prin codul:

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

Metoda `getConnection`() întoarce obiectul `Connection`. Aceste este folosit în continuare pentru executarea de instrucţiuni SQL:

```
DriverManager.getConnection(databaseURL, "", "");
```

Executarea unei comenzi SQL
Pentru executarea unei instrucţiuni SQL trebuie creat un obiect `Statement` prin lunia de cod:

```
Statement statement = connection.createStatement();
```

Apoi, instrucţiunea SQL de executate se pune într-un `String`:

```
String query = "SELECT * FROM Table1";
```

Iar apoi se apelează metoda `executeQuery()`

```
ResultSet resultSet = statement.executeQuery(query);
```

Acesta întoarce un obiect din clasa `ResultSet` care este o mulţime de date ce trebuie parcurse secvenţial, rând cu rând. Codul tipic pentru o astfel de ciclare este:

```
while(resultSet.next()) {
 //afisare unui rand din multimea de randuri
}
```

Codul funcţional pentru această afişare este:

```
while(resultSet.next()) {
  out="";
  for(int i=1;i<=resultSet.getMetaData().getColumnCount();++i)
    out+=resultSet.getString(i) + " ";
  System.out.println(out);
}
```

Dacă comanda SQL execută acţiuni cu sunt INSERT, UPDATE sau DELET sau comenzi de definiţie a datelor (CREATE TABLE sau DROP TABLE) atunci în locul lui `executeQuery()` se execută metoda `executeUpdate()`. Ea întoarce numărul de rânduri afectate de comandă.

C13- Întrebari

1. Care este motivul utilizarii JDBC in accesarea bazelor de date?
2. Ce este JDBC?
3. Ce rol are obiectul Connection?
4. Pentru ce fel de interogari se foloseste clasa Statement?