Aplicații Integrate pentru Întreprinderi

Laborator 3

18.10.2011

Gestiunea informațiilor dintr-o bază de date MySQL prin JDBC

Scopul laboratorului îl reprezintă utilizarea unor comenzi MySQL (precum şi rezultatul lor) prin intermediul limbajului de programare Java, folosind un API care respectă protocolul Java Database Connectivity (JDBC).

1. Ce este un *„driver”* pentru un sistem de gestiune al bazei de date ?

Un *„driver”* pentru un sistem de gestiune al bazei de date este o bibliotecă prin care sunt transformate apelurile JDBC (din limbajul de programare Java) într-un format suportat de protocolul de rețea folosit de sistemul de gestiune al bazei de date, permițând programatorilor să acceseze datele din medii eterogene.

În cadrul laboratorului (în vederea rezolvării temei de casă), pentru conectarea la baza de date MySQL, vom folosi un produs denumit **Connector/J**, driver nativ pentru Java dezvoltat de compania Oracle distribuit în mod gratuit utilizatorilor.

Modelul pe care îl vom folosi implică două niveluri şi e descris în Figura 1:

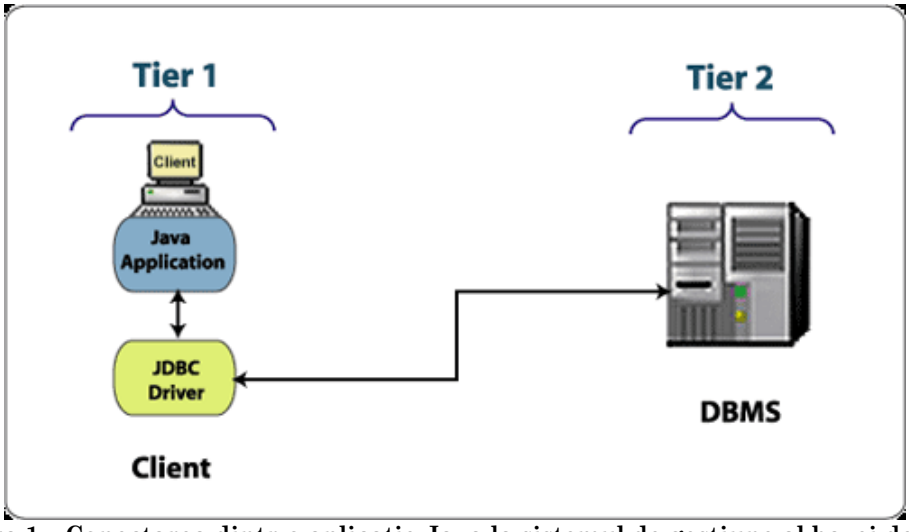

Figura 1 – Conectarea dintr-o aplicație Java la sistemul de gestiune al bazei de date prin intermediul unui driver JDBC

Există patru implementări pentru „drivere” JDBC:

- tipul 1: drivere ce implementează API-ul JDBC ca mapare peste un alt API cum ar fi ODBC (*eng.* Open DataBase Connectivity); portabilitatea lor este relativ redusă, întrucât sunt dependente de o bibliotecă scrisă în cod nativ; această soluție este tranzițională şi ar trebui folosită doar în cazul în care sistemul de gestiune pentru baze de date nu oferă un driver JDBC bazat exclusiv pe Java; Oracle nu implementează acest tip de drivere;
- tipul 2: drivere care sunt scrise parțial în Java şi parțial în cod nativ, folosind o bibliotecă specifică pentru sursele de date la care se conectează, ceea ce le reduce portabilitatea[1];
- tipul 3: drivere dezvoltate exclusiv în Java care comunică cu middleware-ul printr-un protocol independent de baza de date, transmițând către sursa de date interogările;
- tipul 4: drivere scrise în Java care implementează un protocol de rețea specific sistemului de gestiune pentru baze de date, spre a se conecta la sursa de date în mod direct[2].

2. Configurare Connector/J

Tot ce trebuie făcut pentru utilizarea driver-ului de conectare din limbajul de programare Java la sistemul de gestiune al bazei de date (oricare ar fi: MySQL) este să descărcați arhiva care conține Connector/J[3] de pe pagina oficială (http://www.mysql.com/downloads/connector/j/), să o dezarhivați şi să adăugați fişierul .jar din rădăcină la classpath în momentul în care compilați aplicația.

În linie de comandă, acest lucru poate fi realizat astfel:

- compilare:

```
   >javac -classpath .;mysql-connector-java-5.1.18-bin.jar
<nume_fisier>.java
```

- rulare:

```
   >java -classpath .;mysql-connector-java-5.1.18-bin.jar <nume_fisier>
```

Mai uşor, puteți folosi medii integrate de dezvoltare a aplicațiilor, cum ar fi

- *Eclipse* (a fost testată versiunea Indigo – Service Release 1)
  - bibliotecile conținute în arhiva .jar trebuie adăugate la calea proiectului prin comanda *„Add external libraries”* (click dreapta pe numele proiectului → Build Path) – Figura 2
  - dacă librăria externă a fost adăugată în mod corect, numele librăriei trebuie să apară în meniul din stânga corespunzător proiectului, secțiunea *„Referenced libraries”* – Figura 3
- *NetBeans* (a fost testată versiunea 7.0.1)
  - în meniul din stânga corespunzător proiectului, alegeți *Libraries*, apoi click dreapta şi selectați opțiunea *Add JAR/Folder* – Figura 4
  - va apărea o fereastră de dialog în care aveți grijă să selectați referința drept cale absolută (Refernce As: → Absolute Path), în caz contrar, includeți arhiva în sistemul de fişiere al proiectului – Figura 5

[1] Un exemplu de driver JDBC de tip 2 de la Oracle este OCI (Oracle Call Interface)
[2] MySQL Connector/J este un exemplu de driver JDBC de tip 4.
[3] Ultima versiune disponibilă pe pagina oficială este 5.1.18.

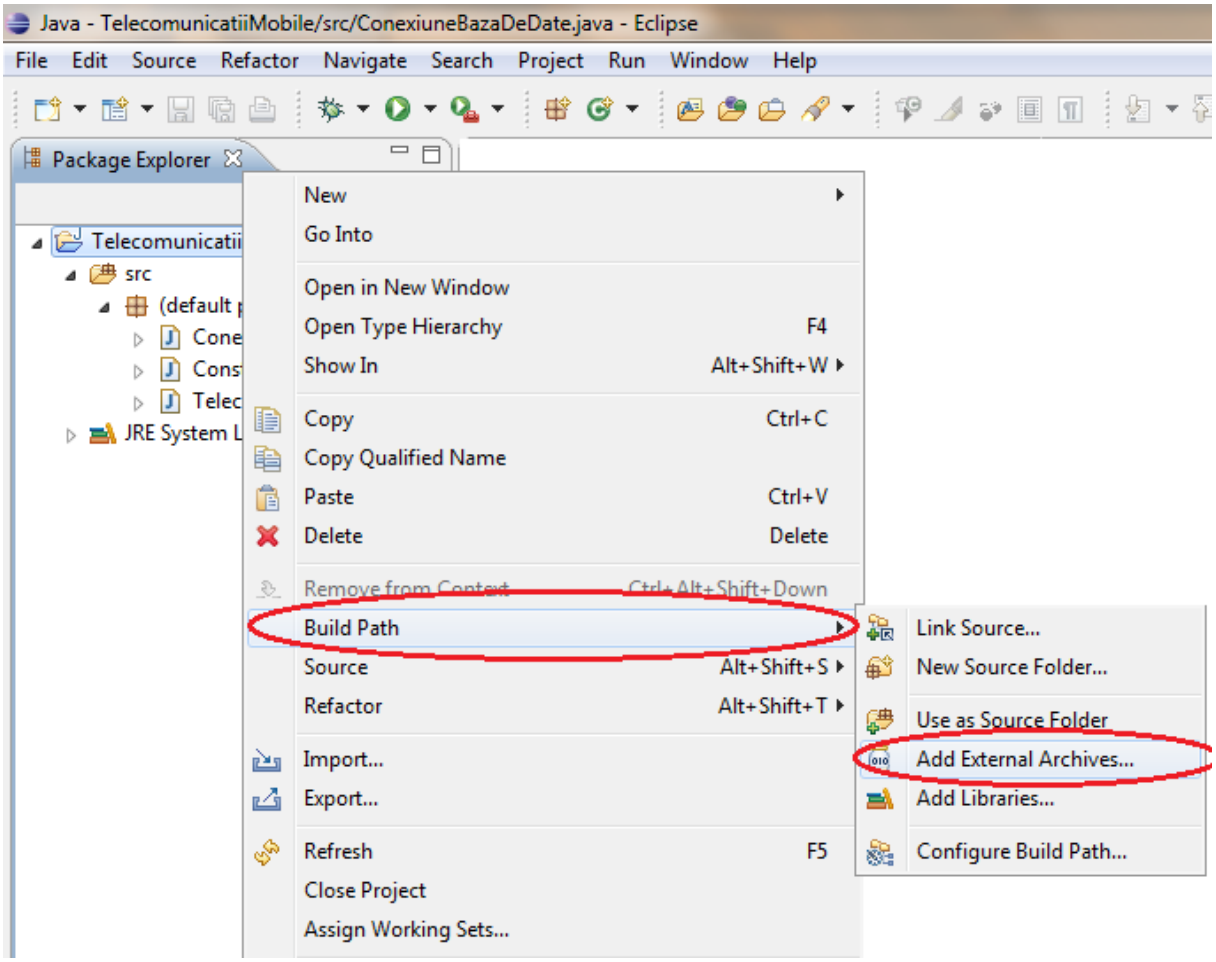

Figura 2 – Adăugarea unei biblioteci externe pentru un proiect în *Eclipse Indigo*

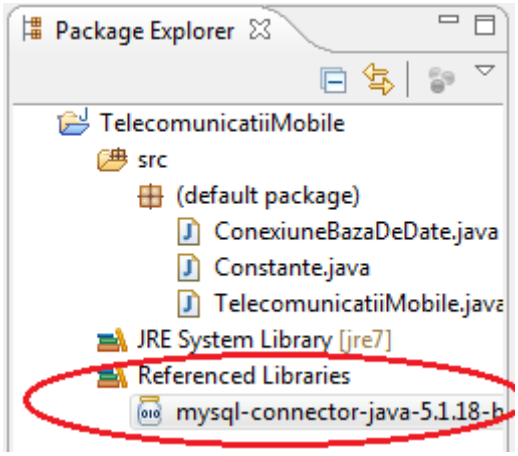

Figura 3 – Adăugarea corectă a bibliotecii externe este marcată prin adăugarea ei în meniul din stânga aferent proiectului, secțiunea *Referenced Libraries*

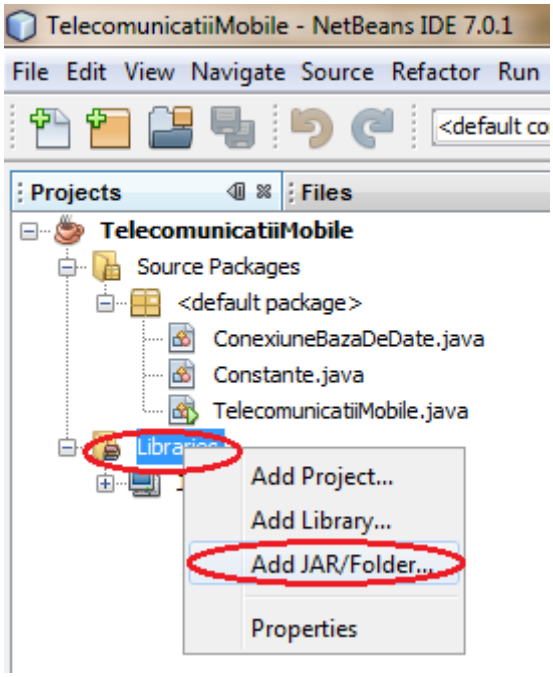

Figura 4 – Adăugarea unei biblioteci externe pentru un proiect în *NetBeans 7.0.1*

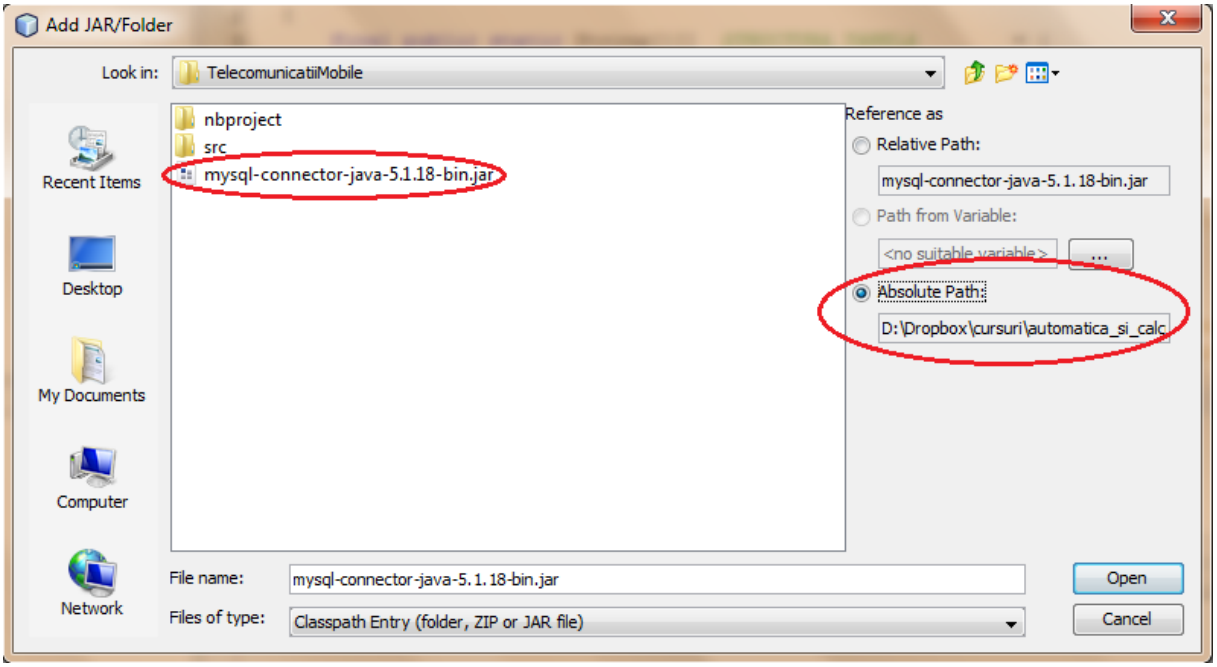

Figura 5 – Referirea căii absolute către librăria externă în condițiile în care este inclusă în sistemul de fişiere al proiectului[4]

[4] Se poate folosi, în acest caz, şi referirea prin cale relativă (*Reference as* → *Relative Path*).

3. Arhitectura JDBC

Arhitectura protocolului JDBC, descrisă în Figura 6, este structurată pe două niveluri:

- un API JDBC responsabil de comunicația dintre aplicația Java şi modulul de gestiune al driver-ului;
- un API JDBC Driver care este responsabil de comunicația dintre modulul de gestiune al driver-ului şi baza de date; un astfel de nivel este independent atât în raport cu baza de date la care se conectează precum şi în raport cu limbajul de programare din care este accesat;

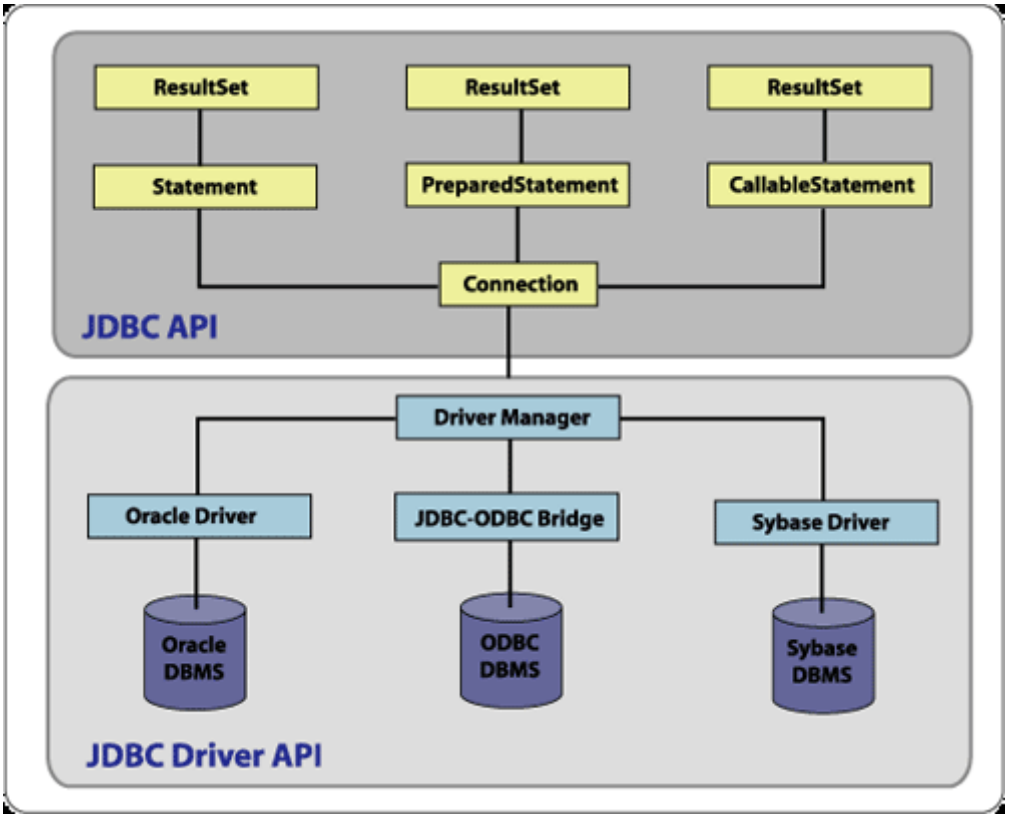

Figura 6 – Arhitectura protocolului JDBC

O aplicație care se conectează la o bază de date folosind protocolul JDBC trebuie să urmeze următorii paşi:

- [înregistrarea „driver"-ului] – opțional, se poate face în două moduri[5]:
  - `DriverManager.registerDriver (new com.mysql.jdbc.Driver());`
  - `Class.forName ("com.mysql.jdbc.Driver").newInstance();`
- deschiderea conexiunii la baza de date
- realizarea de interogări către baza de date
- procesarea rezultatelor obținute
- închiderea conexiunii la baza de date

[5] Metoda `DriverManager.registerDriver` implică existența driver-ului la momentul în care se realizează compilarea, în timp ce metoda `Class.forName` verifică acest lucru la execuție, lipsa claselor respective fiind semnalate printr-o excepție `NoClassDefFoundError`, ceea ce reprezintă un comportament care trebuie evitat, de regulă.

4. Conectarea la sistemul de gestiune al bazei de date

Conectarea unei aplicații Java prin intermediul protocolului JDBC la sistemul de gestiune al bazei de date se poate realiza prin două clase:

- `DriverManager` – presupune încărcarea unui anumit driver prin indicarea unei resurse de localizare de tip URL;
- `DataSource` – metodă mai transparentă de acces la informații, având proprietăți specificate încât să corespundă unor surse de date particulare;

În cadrul laboratorului vom folosi metoda de conectare la baza de date folosind clasa `DriverManager` întrucât este mai intuitivă. Atunci când un client indică un URL pentru a se conecta la o bază de date, clasa `DriverManager` apelează la interfața `Driver` pentru a identifica driver-ul necesar pentru interacțiunea cu sistemul de gestiune al bazei de date.

De obicei, URL-ul respectă următoarea structură:

```
protocol:subprotocol:<nume_bazadedate>[lista_de_proprietati]
```

Câteva exemple de URL-uri corecte (ce corespund formatului de mai sus) sunt

```
jdbc:mysql://<adresa_bazadedate>[:port]/<nume_bazadedate>?user=<utilizator>
&password=<parola>
jdbc:mysql://localhost:3306/telecomunicatii_mobile?user=root&password=*****

jdbc:oracle:<protocol>:@<adresa_bazadedate>:<port>:<instanta>
jdbc:oracle:thin:@localhost:1521:orcl

jdbc:odbc:Driver={Microsoft Access Driver (*.mdb)};DBQ=cale
jdbc:sqlserver://adresa\\server:port;database=baza;user=utilizator;password
=parola;"
```

Metoda `getConnection`, disponibilă în clasa `DeviceManager` oferă un obiect conexiune (`Connection`) la baza de date, care poate fi folosit ulterior pentru diferite interogări:

```
String URL = jdbc:mysql://localhost/telecomunicatii_mobile;
Connection conexiune = DriverManager.getConnection(URL);
```

În condițiile în care se citesc dintr-o interfață grafică cu utilizatorul informații de tip utilizator şi parolă, stocate, spre exemplu în variabilele `user` şi `password`, conectarea se poate face şi sub următoarea formă:

```
Connection conexiune = DriverManager.getConnection(URL,user,password);


Connection conexiune =
   DriverManager.getConnection(URL+"?user="+user+"&password="password);
```

Atunci când driverele gestionate de interfața `Driver` recunosc URL-ul dat ca parametru metodei `getConnection`, se stabileşte o legătură cu sistemul de gestiune pentru baza de date, întorcându-se o conexiune deschisă ce poate fi utilizată pentru formularea de instrucțiuni JDBC care să transmită interogări către baza de date.

5. Interogarea bazei de date conform specificației JDBC

Tipurile de interogări pot fi, conform specificației JDBC, sunt:

- `Statement` – folosite pentru interogări SQL fără parametri;
- `PreparedStatement [extends Statement]` – folosite pentru interogări SQL precompilate care pot conține parametrii de intrare;
- `CallableStatement [extends PreparedStatement]`– folosite pentru a executa rutine stocate care pot conține parametrii de intrare şi de ieşire.

Un obiect de tip interogare (`Statement`) se obține prin metoda `createStatement` aplicabilă unui obiect de tip `Connection`:

```
Statement interogare = conexiune.createStatement();
```

În continuare, obiectul de tip interogare poate fi utilizat pentru realizarea unei operații cu baza de date şi obținerea unui set de date rezultat în urma executării instrucțiunii. Există mai multe moduri prin care se poate realiza execuția unei interogări SQL:

- metoda `execute`: întoarce true dacă primul obiect al interogării este de tipul `ResultSet`; prin această metodă pot fi obținute unul sau mai multe (sau nici un) obiect(e) de tipul `ResultSet`; obiectele de tip `ResultSet` pot fi accesate apelând metoda `Statement.getResultSet`;
- metoda `executeQuery`: întoarce **un singur obiect** de tip `ResultSet`;
- metoda `executeUpdate`: întoarce un număr întreg având semnificația înregistrărilor afectate de expresia SQL; este folosită pentru instrucțiuni de tip `INSERT`, `UPDATE`, `DELETE`.

```
String expresie = „SELECT cnp, nume, prenume FROM clienti”;
boolean rezultat = interogare.execute(expresie);
if (rezultat)
    ResultSet inregistrari = interogare.getResultSet();
```

```
String expresie = „SELECT COUNT(*) FROM facturi”;
ResultSet inregistrari = interogare.executeQuery(expresie);
```

```
String expresie = „UPDATE servicii SET pret=pret*1.2 WHERE pret<350”;
int rezultat = interogare.executeUpdate(expresie);
```

Interfața `ResultSet` pune la dispoziția utilizatorului o serie de metode pentru lucrul cu informațiile obținute în urma interogării bazei de date. Obiectele având tipul `ResultSet` au anumite caracteristici care pot fi modificate între care tipul, gestiunea concurenței şi posibilitatea de deținere a cursorului. Astfel de opțiuni pot fi precizate de utilizator în momentul creării unui obiect de tip interogare (`Statement`).

Cu privire la modalitatea în care poate fi manipulat cursorul (aspecte ce țin şi de senzitivitatea cursorului), există următoarele constante:

- `TYPE_FORWARD_ONLY` – cursorul se poate muta doar înainte, ne-existând posibilitatea parcurgerii în ambele sensuri a setului de date obținut ca rezultat al interogării;
- `TYPE_SCROLL_INSENSITIVE` – cursorul se poate muta înainte şi înapoi, poziționându-se în diferite locații relative față de poziția curentă sau absolute, dar nu este afectat de modificările realizate de alții;
- `TYPE_SCROLL_SENSITIVE` – cursorul se poate muta înainte şi înapoi, poziționându-se în diferite locații relative față de poziția curentă sau absolute, şi este afectat de modificările realizate de alții;

Tipul de concurență indică operațiile pe care utilizatorul are permisiunea spre a le realiza:

- `CONCUR_READ_ONLY` – utilizatorul are doar dreptul de a consulta informațiile, fără a le modifica;
- `CONCUR_UPDATABLE` – utilizatorul poate citi şi poate scrie informațiile reținute în setul rezultat;

Menținerea cursorului după realizarea tranzacțiilor se face prin constantele:

- `HOLD_CURSORS_OVER_COMMIT`
- `HOLD_CURSORS_AT_COMMIT`

Un exemplu de creare a unui obiect de tip interogare realizat pentru obținerea unui set de date în care cursorul poate fi mutat în ambele direcții, dar nu poate fi modificat este redat mai jos:

```
Statement interogare =
          conexiune.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
                                    ResultSet.CONCUR_READ_ONLY);
```

Un obiect de tip `ResultSet` conține mai multe (sau nici un) tuplu(ri), în funcție de specificitatea interogării, având asociat un cursor care indică la orice moment rândul curent[6].

Unele dintre metodele care pot fi utilizate pentru a realiza poziționări ale cursorului în cadrul setului de date sunt:

metoda	descriere
`next()`	mută cursorul pe înregistrarea următoare
`previous()`	mută cursorul pe înregistrarea precedentă
`first()`	mută cursorul pe prima înregistrare
`last()`	mută cursorul pe ultima înregistrare
`beforeFirst()`	mută cursorul înainte de prima înregistrare
`afterLast()`	mută cursorul după prima înregistrare
`relative (int n)`	mută cursorul la n poziții distanță față de poziția curentă
`absolute (int n)`	mută cursorul la poziția n (absolută) din set

Obținerea informațiilor (valorilor asociate atributelor) se realizează prin așa-numitele metode *getter* (`getString`[7], `getInt`, `getByte`, `getBoolean`, `getBlob`, `getDate`) care pot primi ca parametru fie numele coloanei[8] fie indexul[9] ei în cadrul tabelei din baza de date.

O rutină de parcurgere a tuturor înregistrărilor dintr-o bază de date poate fi cea descrisă mai jos:

```
ResultSet rezultat = interogare.executeQuery
                    ("SELECT denumire, pret FROM produse");
while (rezultat.next())
// se muta cursorul pe inregistrarea urmatoare daca aceasta exista
{
    String denumire = rezultat.getString(1);
    float pret = rezultat.getFloat("pret");
}
```

Mai multe comenzi SQL pot fi executate împreună folosind metodele:

- `void addBatch (String sql) throws SQLException`
- `void clearBatch () throws SQLException`
- `int[] executeBatch () throws SQLException`

Procesul de actualizare a informațiilor dintr-o bază de date e realizat în 2 faze

- modificarea valorilor ce se doresc actualizate, la nivel de coloană, pe rândul unde se găseşte cursorul, prin intermediul metodelor `update...()`; în acest moment, nici o modificare nu este marcată la nivelul tabelei;
- actualizarea rândului curent în care au fost marcate spre modificare valorile coloanelor prin intermediul metodei `updateRow()`;

[6] Inițial, cursorul se găseşte deasupra primului rând.
[7] Metoda poate fi folosită pentru preluarea oricărui tip de informație din baza de date, mai puțin tipul SQL 3.
[8] Metoda nu ține cont de capitalizarea şirului de caractere care este oferit drept parametru.
[9] Această metodă este mai eficientă. Numerotarea coloanelor începe de la 1.

Un exemplu de actualizare a informațiilor în baza de date este redat mai jos:

```
Statement interogare = conexiune.createStatement
                                  (ResultSet.TYPE_SCROLL_SENSITIVE,
                                   ResultSet.CONCUR_UPDATABLE);
ResultSet rezultat = interogare.executeQuery
                    ("SELECT emitere, scadenta FROM facturi");
while (rezultat.next())
{
     rezultat.updateDate(scadenta,'CURDATE()');
     updateRow();
}
```

6. Utilizarea interogărilor parametrizabile

Atunci când nu toate datele interogării sunt cunoscute la momentul în care progamul este compilat, există posibilitatea ca interogarea să fie generică, urmând a fi completată cu informații (provenite dintr-un fişier sau introduse chiar de către utilizator) atunci când ele sunt disponibile, şi anume la rulare, înainte de execuția interogării asupra bazei de date.

În acest sens, sunt folosite obiecte de tip `PreparedStatement`, derivate din clasa `Statement`, care primesc la momentul construcției şi interogarea corespunzătoare, în care informațiile necunoscute sunt înlocuite prin caracterul `?`:

```
String expresie = "UPDATE clienti SET tip = ? WHERE data_contract = ?";
PreparedStatement interogare = conexiune.prepareStatement (expresie);
```

Înainte de a executa o astfel de interogare, trebuie specificate valorile care corespund atributelor lipsă, lucru care se face prin metode de tip *setter*:

```
interogare.setString (1, Integer.parseInt(jscrollbar.getValue()));
interogare.setDate (2, Date.valueOf(jtextarea.getTextArea().getText()));
```

Execuția interogării odată ce toate datele sunt cunoscute se face prin comanda `executeUpdate`:

```
interogare.executeUpdate();
conexiune.commit()[10];
```

În mod similar se procedează pentru apelarea unei rutine stocate, obiectul fiind de tip `CallableStatement`:

```
String expresie = "{? = CALL valoare_factura (?)}";
CallableStatement interogare_rutina = conexiune.prepareCall(expresie);
interogare_rutina.registerOutParameter(1, java.sql.Types.VARCHAR[11]);
interogare_rutina.setString(2,jlist.getSelection());
interogare_rutina.execute();
String valoare = interogare_rutina.getString(1);
interogare_rutina.close();
```

Aplicație de Laborator

Se doreşte dezvoltarea unui modul pentru un sistem ERP al unei companii care are drept domeniu de afaceri telecomunicațiile mobile. Se va implementa serviciul financiar-contabil care se ocupă de procesul de facturare.

Tabelele din baza de date respectă schema conceptuală descrisă în cadrul laboratorului 2.

[10] Este important ca la începutul tranzacțiilor să se apeleze `connection.setAutoComit(false)` pentru a nu se produce modificări în baza de date până când acest lucru nu este specificat explicit.

[11] Tipurile de date din interfața `java.sql.Types` au acelaşi nume ca cele din MySQL.

(1p) Să se execute script-ul `laborator3.sql` unde se crează structura bazei de date sunt populate tabelele şi se crează o funcție stocată.
(2p) Să se apeleze metoda `adauga_inregistrari_tabela` din clasa `ConexiuneBazaDeDate` pentru a introduce noi înregistrări în tabela `clienti`;
(2p) Să se apeleze metoda modifica_inregistrari_tabela din clasa `ConexiuneBazaDeDate` pentru a edita unele înregistrări în tabela `facturi`;
(2p) Să se apeleze metoda `sterge_inregistrari_tabela` din clasa `ConexiuneBazaDeDate` pentru a elimina nişte înregistrări din tabela `servicii`;
(2p) Să se implementeze metoda `apeleaza_functie` astfel incat sa poata fi folosita pentru o interogare ce cuprinde si apelul unei functii stocate, dupa modelul campurilor din atributul `functie` al interfetei `Constante`.
(1p) Să se verifice corectitudinea implementarii metodei `apeleaza_functie`.
BONUS
(1p) Să se implementeze metoda `calculeaza_statistici` astfel incat sa poata fi folosita pentru interogari ce cuprind si apelul unor functii de grup, dupa modelul campurilor din atributul `statistici` al interfetei `Constante`. Trebuie suportate clauze precum `GROUP BY` şi `ORDER BY`. Se poate presupune că gruparea se face după toate atributele din interogare care nu sunt funcții de grup, iar ordonare se face după atributele din interogare care sunt funcții de grup, în ordine descrescătoare.
(1p) Să se verifice corectitudinea implementării metodei `calculeaza_statistici`.