Aplicații Integrate pentru Întreprinderi

Laborator 3

22.10.2012 / 25.10.2012

Gestiunea informațiilor dintr-o bază de date MySQL prin JDBC

Scopul laboratorului îl reprezintă exploatarea unor interogări 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.

Pentru conectarea la baza de date MySQL, se poate folosi **Connector/J**, driver nativ pentru Java dezvoltat de compania Oracle şi distribuit gratuit utilizatorilor.

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

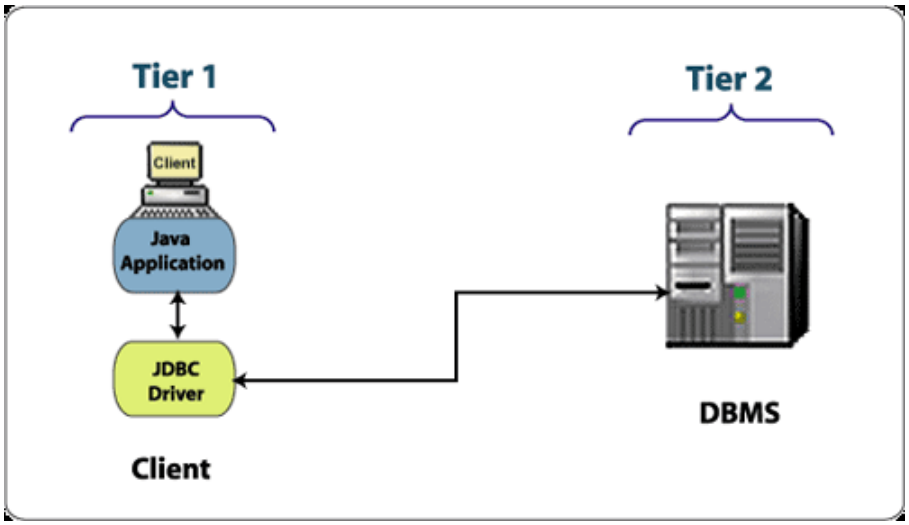

Figura 1 – Conectarea dintr-o aplicație Java la sistemul de gestiune al bazei de date prin intermediul unui driver JDBC

Prin urmare, „driver-ul” pentru sistemul de gestiune al bazei de date realizează legătura între nivelul de logică a aplicației[1] şi nivelul de date (reprezentat prin baza de date propriu-zisă).

[1] În cadrul nivelului de logică a aplicației, poate fi descris un subnivel pentru acces la date care accesează în mod explicit „driver-ul”.

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[2];
- 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[3].

2. Configurare Connector/J

Tot ce trebuie făcut pentru utilizarea driver-ului de conectare din limbajul de programare Java împreună cu sistemul de gestiune pentru bazei de date MySQL[4] este să descărcați arhiva care conține Connector/J[5] 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-5.1.22-bin.jar
<nume_fisier>.java
```

- rulare:

```
>java -classpath .;mysql-connector-java-5.1.22-bin.jar <nume_fisier>
```

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

- *Eclipse* (a fost testată versiunea Juno – 4.2.1)
  - clasele conținute în arhiva .jar trebuie adăugate la calea proiectului prin *„Add external libraries”* (click dreapta pe numele proiectului → Build Path) – Figura 2
  - dacă biblioteca externă a fost adăugată în mod corect, numele ei trebuie să apară în meniul din stânga corespunzător proiectului, secțiunea *„Referenced libraries”* – Figura 3
- *NetBeans* (a fost testată versiunea 7.2)
  - 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 relativă (Refernce As: → Relative Path), arhiva găsindu-se în sistemul de fişiere al proiectului – Figura 5

[2] Un exemplu de driver JDBC de tip 2 de la Oracle este OCI (Oracle Call Interface)
[3] MySQL Connector/J este un exemplu de driver JDBC de tip 4.
[4] Pentru alte sisteme de gestiune ale bazelor de date (Oracle, IBM DB2), „driver-ele” de conectare sunt de obicei disponibile la adresele de unde pot fi descărcate şi produsele propriu-zise. Java DB este distribuită împreună cu un „driver” de conectare la sistemul de gestiune al bazei de date.
[5] Ultima versiune disponibilă pe pagina oficială este 5.1.22.

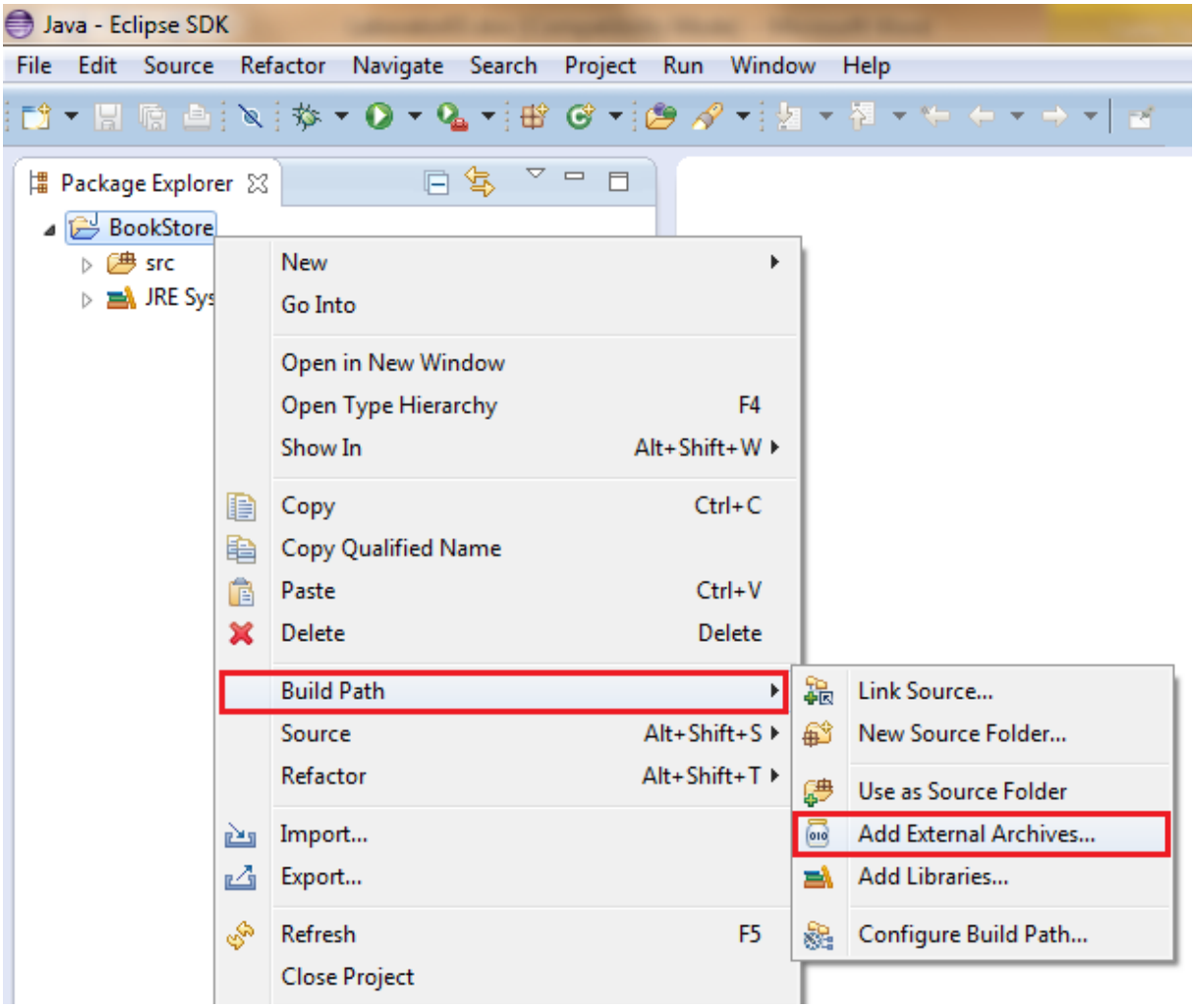

Figura 2 – Adăugarea unei biblioteci externe pentru un proiect în *Eclipse Juno*

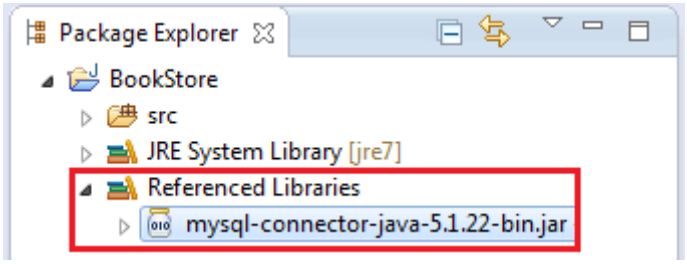

Figura 3 – Adăugarea corectă a bibliotecii externe este marcată prin adăugarea ei în meniul *Package Explorer* aferent proiectului, secțiunea *Referenced Libraries*

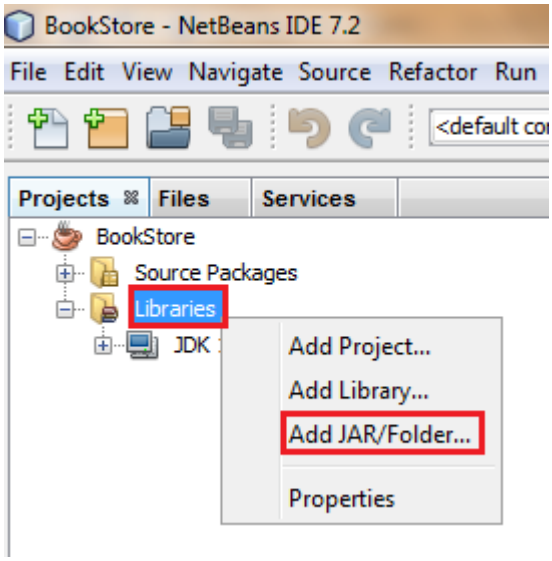

Figura 4 – Adăugarea unei biblioteci externe pentru un proiect în *NetBeans 7.2*

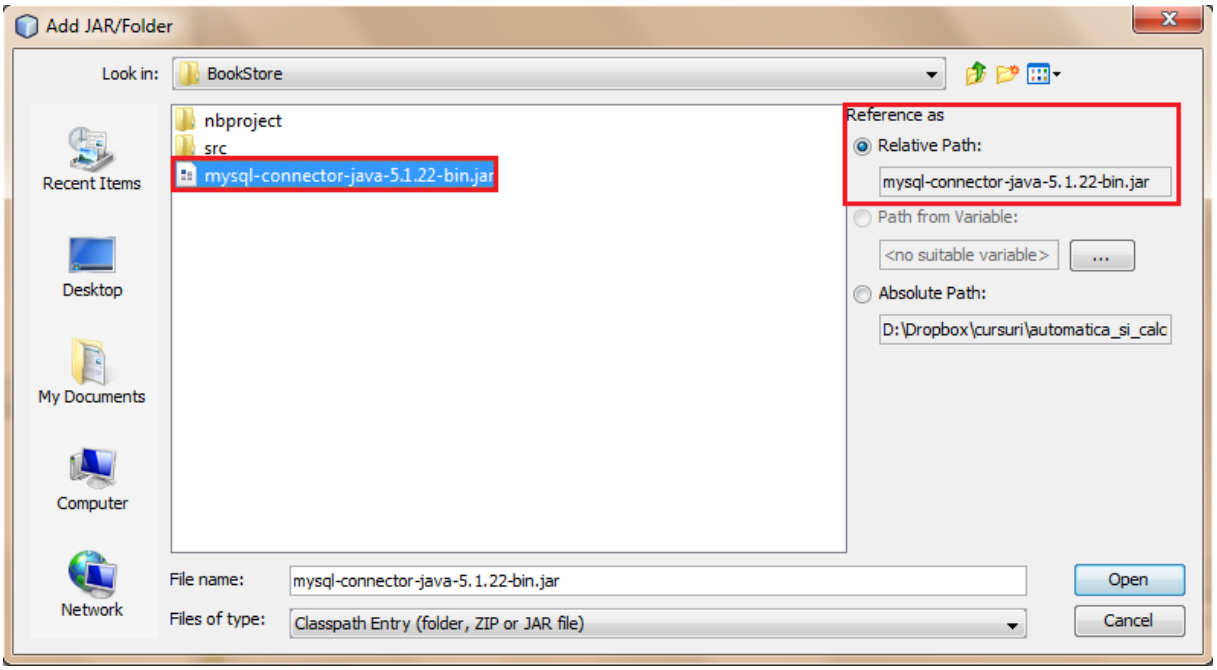

Figura 5 – Referirea căii relative către biblioteca externă în condițiile în care este inclusă în sistemul de fişiere al proiectului

3. Arhitectura JDBC

Arhitectura protocolului JDBC, reprezentată 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;

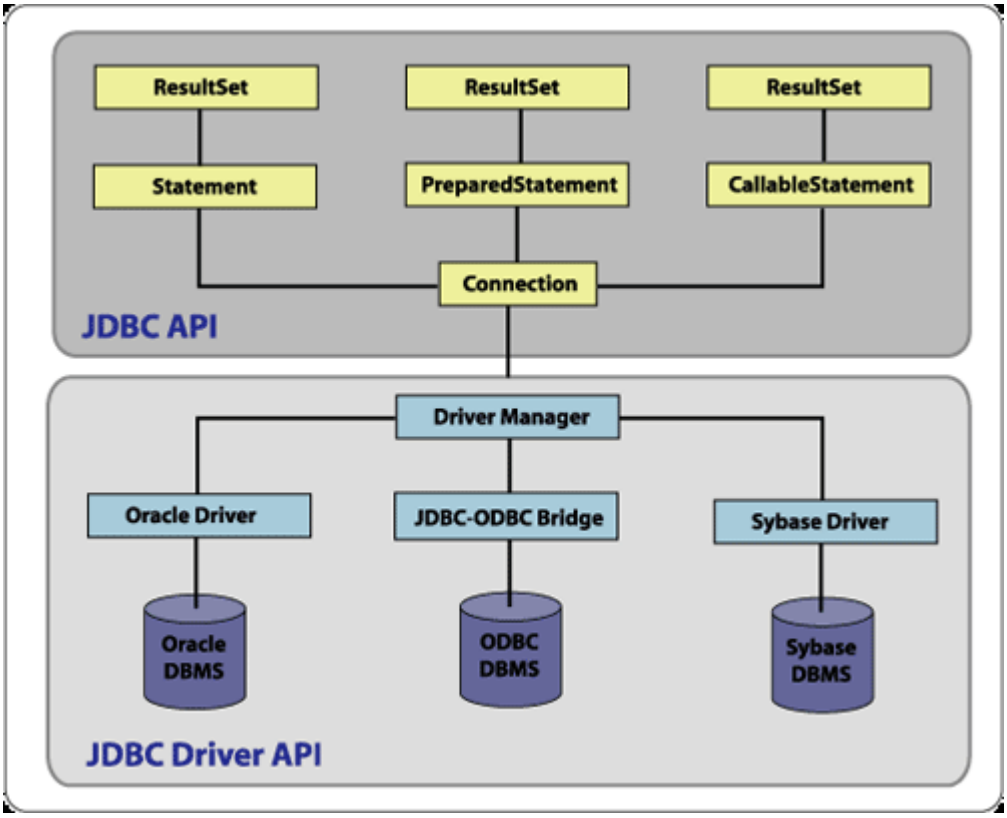

Figura 6 – Arhitectura protocolului JDBC

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[6]:
  - `DriverManager.registerDriver (new com.mysql.jdbc.Driver());`
  - `Class.forName ("com.mysql.jdbc.Driver").newInstance();`
- deschiderea conexiunii la baza de date
- realizarea de interogări[7] către baza de date
- procesarea rezultatelor obținute
- închiderea conexiunii la baza de date

[6] 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`. Începând cu JDBC 4.0, se încarcă în mod automat „driver-ul" identificat în classpath, astfel încât metoda `Class.forName` nu mai trebuie apelată explicit.

[7] Interogarea trebuie să fie „construită" anterior execuției sale.

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_baza_de_date][lista_de_proprietati]
```

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

```
jdbc:mysql://[host][,failoverhost[8]...][:port]/[database]
[?propertyName1][=propertyValue1][&propertyName2][=propertyValue2]...
jdbc:mysql://localhost:3306/librarie?user=root&password=*****

jdbc:derby:[subsubprotocol[9]:][databaseName][;attribute=value[10]]*
jdbc:derby:librarie;create=true

jdbc:oracle:[protocol]:@[database_host]:[port]:[instance]
jdbc:oracle:thin:@localhost:1521:orcl

jdbc:odbc:Driver={Microsoft Access Driver (*.mdb)};DBQ=path
jdbc:sqlserver://address\\server:port;database=databaseName;user=usr;
password=password;
```

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:3306/librarie";
Connection conexiune = DriverManager.getConnection(URL);
```

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

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

Atunci când driverele gestionate de interfața `Driver` recunosc URL-ul indicat drept parametru metodei `getConnection`, se stabileşte o legătură cu sistemul de gestiune pentru baza de date, întorcându-se o conexiune deschisă care poate fi utilizată pentru formularea de instrucțiuni JDBC translatate ulterior în interogări către baza de date.

---

[8] Connector/J permite specificarea unei baze de date opționale la care să se încerce conectarea, dacă operația a eşuat în cazul bazei de date primare.

[9] Deşi în general este omis, parametrul `subprotocol` indică locația bazei de date (director din sistemul de fişiere, memorie, classpath, fişier .jar).

[10] În cadrul listei de atribute se poate specifica crearea bazei de date, criptarea acesteia, locația fişierelor în care să se păstreze diferite jurnale, numele de utilizator şi parola pentru conectare.

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. Caracteristicile 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 în timp ce este utilizat; conține înregistrările care satisfac condițiile interogării atunci când este executată sau pe măsură ce sunt obținute entitățile;
- `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 de 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;

Deținerea cursorului la realizarea tranzacțiilor[11] se face prin constantele:

- `HOLD_CURSORS_OVER_COMMIT`
- `CLOSE_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,
                                    ResultSet.HOLD_CURSORS_OVER_COMMIT);
```

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[12].

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 metode de tip *getter* (`getString`[13], `getInt`, `getByte`, `getBoolean`, `getBlob`, `getDate`) care pot primi ca parametru fie numele coloanei[14] fie indexul[15] ei în cadrul tabelei din baza de date.

O rutină de parcurgere a înregistrărilor dintr-o bază de date poate fi:

```
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");
}
```

[11] Această proprietate specifică comportamentul cursorului în momentul în care se apelează metoda `commit`.
[12] Inițial, cursorul se găseşte deasupra primului rând.
[13] Metoda poate fi folosită pentru preluarea oricărui tip de informație din baza de date, mai puțin tipul SQL 3.
[14] Metoda nu ține cont de capitalizarea şirului de caractere care este oferit drept parametru.
[15] Această metodă este mai eficientă. Numerotarea coloanelor începe de la 1.

Mai multe comenzi SQL pot fi executate împreună folosind metodele:

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

După metoda `executeBatch()` aplicată unui obiect interogare, se apelează şi metoda `commit()`[16], astfel încât modificările să fie vizibile în cadrul bazei de date.

În contextul tranzacțiilor, se poate salva starea bazei de date înaintea realizării unor modificări, astfel încât în situația în care acestea produc anumite efecte nedorite la nivelul informațiilor din tabele, să se poată reveni la aceasta:

```
SavePoint stare = conexiune.setSavePoint();
...
conexiune.rollback(stare);
```

Metoda `rollback` încheie tranzacția curentă, astfel încât aceasta va fi apelată întotdeauna la sfârşitul tranzacției.

O stare a bazei de date salvată poate fi eliminată din cadrul tranzacției folosind metoda `releaseSavePoint` obiectului de tip `Statement` corespunzător.

Metoda `executeBatch()` întoarce un vector care conține numărul operațiilor de tip actualizare realizate cu succes, motiv pentru care interogările din cadrul tranzacției NU trebuie să întoarcă un rezultat de tip `ResultSet`. În caz contrar, excepția `BatchUpdateException` va fi generată[17]. Prin urmare, interogările incluse în cadrul unei tranzacții vor fi DDL (`CREATE TABLE`, `ALTER TABLE`, `DROP TABLE`), respectiv DML (`INSERT INTO`, `UPDATE`, `DELETE`).

Procesul de actualizare a informațiilor într-o bază de date printr-un obiect de tip `ResultSet` este realizat în 2 etape:

- modificarea valorilor ce se doresc actualizate, la nivel de coloană, pe rândul unde se găseşte cursorul, prin intermediul metodelor de tip `update...()`; acum, 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()`;

Un exemplu de actualizare a informațiilor în baza de date este:

```
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();
}
```

De asemenea, pentru introducerea de informații într-o bază de date folosind un obiect de tip `ResultSet` se pot folosi metodele `moveToInsertRow()` urmată de specificarea atributelor în acelaşi mod ca în exemplul de mai sus (folosind metode de tip `update...()`) pentru ca adăugarea propriu-zisă să fie realizată prin metoda `insertRow()`.

O înregistrare poate fi ştearsă folosind metoda `deleteRow()`.

[16] 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.

[17] Această excepție va fi generată şi în cazul în care una dintre operațiile tranzacției nu a fost executată cu succes.

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 momentul în care acestea sunt create, ele primesc şi o parte din interogarea propriu-zisă, transmisă sistemului de gestiune al bazei de date care îl precompilează, astfel încât execuția sa va fi mai rapidă.

Sunt folosite obiecte de tip `PreparedStatement`, derivate din clasa `Statement`, informațiile necunoscute fiind specificate 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 se face folosind metodele specifice clasei `Statement`.

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[18]);
interogare_rutina.setString(2,jlist.getSelection());
interogare_rutina.execute();
String valoare = interogare_rutina.getString(1);
interogare_rutina.close();
```

JDBC permite utilizarea unor obiecte de tip `RowSet`, derivate din `ResultSet`, care oferă programatorilor posibilitatea de a accesa datele mai uşor, având comportament[19] de componente JavaBeans. Astfel de obiecte sunt considerate conectate sau deconectate de la sursa de date, după cum mențin conexiunea (printr-un „driver") la baza de date pe parcursul ciclului de viață. Un tip de obiect conectat este `JdbcRowSet` (care oferă o funcționalitate asemănătoare cu `ResultSet`) în timp ce tipurile de obiecte deconectate[20] sunt `CachedRowSet`, `WebRowSet`, `JoinRowSet` şi `FilteredRowSet` – acestea se vor conecta la sursa de date doar pentru operații de citire şi de scriere, situație în care vor trebui să verifice şi conflictele care pot apărea în astfel de situații.

Obiectele `JdbcRowSet` pot fi create[21] folosind un obiect `ResultSet`, `Connection`, utilizând un constructor implicit sau dintr-o instanță a clasei `RowSetFactory`. Pentru fiecare obiect `JdbcRowSet` se pot stabili proprietățile (`url`, `username`, `password`, `dataSourceName`). O interogare se poate specifica folosind metoda `setCommand`, iar execuția se face folosind `execute`, ca şi în cazul `ResultSet`.

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

[19] Comportamentul unor componente JavaBeans se referă la accesarea atributelor ca proprietăți precum şi la mecanismul de notificare, de fiecare dată când se modifică poziția cursorului, când sunt executate operații de adăugare, modificare, ştergere la nivelul unui rând dar şi atunci când se modifică conținutul obiectului respectiv.

[20] Obiectele `RowSet` de tip deconectat au şi proprietatea că sunt serializabile ceea ce le face ideale pentru a fi transmise prin intermediul unei rețele.

[21] În toate aceste cazuri se va folosi clasa `JdbcRowSetImpl`.

Pentru tipurile de date deconectate, actualizările implică folosirea metodei `acceptChanges()` pentru a se realiza operația respectivă la nivelul bazei de date. Întrucât pot apărea conflicte[22], se poate folosi mecanismul implicit (`RIOptimisticProvider`) care presupune că astfel de situații sunt foarte rare, iar atunci când sunt întâlnite nu marchează modificările la nivelul bazei de date sau se poate specifica un comportament, prin intermediul clasei `SyncResolver` care pune la dispoziție metoda `setResolvedValue` prin care se stabileşte conținutul bazei de date în funcție de situația specifică.

Activitate de Laborator

Se doreşte proiectarea unei baze de date ce va fi folosită în cadrul unui sistem ERP pentru o librărie care comercializează doar cărți. Vom porni de la schema conceptuală, respectiv de la structura bazei de date construite în cadrul laboratorului anterior.

[0p] 1. În MySQL Workbench, să se execute script-ul `Laborator3.sql` unde se crează structura şi diferite obiecte ale bazei de date `libarie`.

[1p] 2. În clasa `DataBaseConnection` din pachetul `dataaccess`, să se implementeze metoda `getTableNumberOfRows` care determină numărul de înregistrări al unei tabele identificata prin nume (primit ca parametru).

```
public static int getTableNumberOfRows(String tableName)
```

Folosind metoda `getTableNumberOfRows`, să se determine numărul de înregistrări stocate în baza de date librărie.

[2p] 3. Folosind metoda `getTableContent`, să se creeze un fişier `books.txt` în care să se creeze lista tuturor cărților disponibile în librărie (existente în stoc). Pentru fiecare carte în parte se va preciza numele şi prenumele autorilor, titlul, editura precum şi anul apariției.

[1p] 4. Să se adauge în tabela `utilizatori` o înregistrare ale cărei atribute sunt introduse de la tastatură.

Pentru a citi o valoare de la tastatură în Java, se poate folosi următorul cod sursă:

```
BufferedReader buffer = new BufferedReader(new InputStreamReader(System.in));
value = buffer.readLine();
```

[1p] 5. Pe baza metodei `updateRecordsIntoTable`, să se modifice toate comenzile de aprovizionare către editura Aramis (`id_editura = 73, CIF = '247764320'`) astfel încât cantitățile să fie crescute cu 20%.

Modificările nu vor fi operate decât asupra comenzilor care nu au fost încă onorate (`stare='plasată'`).

[2p] 6. În clasa `DataBaseConnection` din pachetul `dataaccess`, să se implementeze metoda `deleteRecordsFromTable` care elimină dintr-o tabelă identificată prin nume (primit ca parametru) acele înregistrări care au anumite valori corespunzătoare unor atribute sau care respectă o anumită condiție.

```
public static void deleteRecordsFromTable(String tableName,
ArrayList<String> attributes, ArrayList<String> values, String whereClause)
throws Exception
```

Dacă `attributes = (attribute1, ..., attributen)`, respectiv `values = (value1, ..., valuen)`, atunci vor fi şterse înregistrările pentru care `attribute1=value1 AND ... AND attributen=valuen` sau pentru care este îndeplinită condiția `whereClause`. În cazul în care sunt precizate attributes/values, parametrul whereClause va fi ignorat.

[22] Un tip de conflict frecvent întâlnit este atunci când o altă tranzacție a modificat aceleaşi date asupra cărora a operat şi tranzacția curentă.

Folosind metoda `deleteRecordsFromTable` astfel implementată să se şteargă acele edituri care nu au cărți comercializate de librarie.

[1p] 7. Folosind procedura stocată `calculate_user_total_bill_value` să se determine lista utilizatorilor împreună cu vânzările asociate fiecăruia.

[1p] 8. Folosind funcția stocată `calculate_supply_order_value` să se determine lista editurilor către care s-au efectuat cele mai mari plăți.

[1p] 9. În clasa `DataBaseConnection` din pachetul `dataaccess`, să se implementeze metoda `getReferrencedTables` ce determină pentru o tabelă dată prin nume (primit ca parametru) care sunt tabelele pe care le referă, precum şi atributele care fac obiectul constrângerii de tip `FOREIGN KEY`.

Se presupune că respectivul câmp de legătură are aceeaşi denumire în cele două tabele şi că acesta este cheie primară în tabela referită.

```
public ArrayList<ReferrencedTable> getReferrencedTables (String tableName)
class ReferrencedTable {
	String parentTable;
	String attributeName;
}
```

Bibliografie

Java Tutorial – JDBC™ Database Access
http://docs.oracle.com/javase/tutorial/jdbc/index.html