Java tutorial

Adatbáziskezelés

- Relációs adatbáziskezelők
- Noha a Java objektum-elvű, egyelőre nem az objektum-elvű adatbáziskezelőket támogatja
  - Vannak egyáltalán igazán működő ilyenek?
- Alapelv: platform-függetlenség, ebbe beletartozik az adatbáziskezelőktől való függetlenség is
- Megoldás: JDBC

JDBC

- Valami interfész, amit a Java programok és az adatbáziskezelők közé lehet rakni
- Ne kelljen olyan kódot írni, amely egy picit is egy konkrét ABkezelő rendszertől függ
- Egy művelethalmaz, amit a Java programozók használhatnak
- Funkcionalitás, amit az adatbáziskezelő készítőjének meg kell valósítania
  - JDBC-compliant
- Szerződés-modell

JDBC: Java Data Base Connectivity

- Adatbáziskezelő készítője: csinálnia kell egy meghajtó-programot (driver), amellyel az adatbáziskezelő kielégíti a JDBC specifikációt
- Java programozó: a java.sql és a javax.sql csomagokban található osztályok, interfészek segítségével dolgozik, és a programhoz csatolja a használt adatbáziskezelő JDBC-meghajtóprogramját

Relációs adatbáziskezelés - áttekintés

- Az adatok fizikai ábrázolása mellékes (remélhetőleg hatékony)
- Logikailag az adatok relációkként jelennek meg
- A relációk tábláknak felelnek meg
- A reláció elemei a tábla sorai
- A reláció típusát az oszlopok határozzák meg
- A relációkon műveletek végezhetők (pl. join)
- Kitüntetett projekciók: kulcsok

Példa: bróker cég

- Ügyfelek tábla

Azon.	Név	Cím
1	John Lennon	London, Abbey Road 1
2	Jim Morrison	LA, Love Street 7
3	Presszer Gábor	Bp, Miénkitta tér 3

- Részvények tábla

Név	Ár
FORTE	190
TVK	120

Adatmanipuláció: SQL

- Structured Query Language, IBM
- Elterjedt, burjánzó, de szabványosították (ISO-ANSI, 1988)
- A JDBC az ANSI SQL-2 szabványt követi
- Lekérdezések, adatfelvitel, törlés, módosítás adatbázis-adminisztráció

Példák

- `select * from Ügyfelek`
- `select Név from Részvények where Ár < 150`
- `insert into Részvények (Név, Ár) values ('MOL',111)`
- `update Részvények set Ár=121 where Név='MOL'`
- `delete from Részvények where Név='MOL'`
- `create table Tulajdonok (Azonosító int, Mennyiség int)`
- `drop table Tulajdonok`

Java tutorial

Mini SQL: mSQL

- Nagyon egyszerű adatbáziskezelő (Hughes)
- Nem mindent tud, amit az SQL szerint kéne
- Kis adatbázisokhoz hatékony, kis erőforrás-igényű
- Van hozzá: terminál program, relációséma nézegető, C-nyelvi API, JDBC meghajtó
- Kliens/szerver támogatása TCP/IP-n keresztül

Használat

- Indítsuk el az adatbázis motort. Ehhez az **`msql2d`** programot kell futtatni.
- **`relshow`**
  A séma nézegető: milyen adatbázisok vannak
- **`relshow StockMarket`**
  Milyen táblák vannak benne
- **`relshow StockMarket Customer`**
  egy tábla definíciójának megtekintése
- **`msql StockMarket`**
  belépés a terminál programba, AB megnyitása

Az msql program

- Írjunk be SQL parancsokat
- A végére **`\g`** kell, ez zárja le a parancsot (a parancs lehet több soros is)
- Segítség: **`\h`**
- Kilépés: **`\q`**
- Fájlból parancsok:
  `msql StockMarket <input.txt`

Elérés TCP/IP-n keresztül

msql -hostname *gépnév adatbázis*
msql -h *gépnév adatbázis*

```
msql -h localhost StockMarket
relshow -h localhost StockMarket Stock
```

Az adatbázis elérhető a 1112 TCP porton keresztül (alapértelmezés szerint)

Adatbáziskezelés Java programból

- JDBC nélkül: AB-kezelőtől függő kód, API-hívások JNI-n keresztül (Java Native Interface)
- JDBC-vel: kód, mely független az AB-kezelőtől. Bármikor le lehet cserélni...
- Az adatmanipulációs nyelv továbbra is az SQL
- Nem beágyazott, az SQL parancsok sztringekben vannak
  - A fordító nem ellenőrzi, futás közben derül ki minden hiba... :-(

mSQL és Java

- Az **msql-jdbc-1-0.jar** fájlban van a meghajtóprogram
- Ezt a jar fájlt is használjuk a fordításkor és a futtatáskor
  -classpath msql-jdbc-1-0.jar:.
- Mi van benne? Egy csomó osztály, ami megvalósít fontos interfészeket
  - Driver, Connection, Statement, ResultSet, stb.
  - Pl. a com.imaginary.sql.msql.MsqlDriver osztályt valahogy a JVM tudomására kell hozni, mondjuk példányosítani

```
import java.sql.*;
class DBTeszt {
  public static void main(String args[])
  throws SQLException {
     new com.imaginary.sql.msql.MsqlDriver();
     Connection c = DriverManager.getConnection
        ("jdbc:msql://localhost:1112/StockMarket");
     Statement s = c.createStatement();
     ResultSet rs = s.executeQuery
                        ("select * from Stock");
     while( rs.next() ){
        System.out.println(rs.getString(1));
     }
     rs.close(); s.close(); c.close();
  }
}
```

Az adatbáziskezelés menete

- Amikor a JVM betölti a Driver interfészt megvalósító osztályt, akkor az beregisztrálja magát a DriverManager-nél
- A DriverManager-től kérhetünk egy Connection-t
- A Connection-től egy Statement-et
- A Statement-tel végrehajthatunk egy SQL utasítást. Lekérdezés eredménye egy ResultSet
- A ResultSet-et bejárhatjuk soronként

Ezeket a kékeket mind megvalósítja a meghajtóprg.

Java tutorial

A JDBC meghajtók

- Egy Java program több adatbáziskezelőt is használhat
- Mindegyikhez kell a megfelelő meghajtóprg.
- A DriverManager tartja nyilván a meghajtókat
- Kiválasztja a megfelelőt:

```
Connection c = DriverManager.getConnection
   ("jdbc:msql://localhost:1112/StockMarket");
```

- A programozó dolga csak az, hogy a JVM-mel betöltesse a Driver-t implementáló osztályt, mert az automatikusan beregisztrálja magát a DriverManager-nél (egy statikus inicializátorral)

A meghajtó betöltése

- Példány létrehozása

```
new com.imaginary.sql.msql.MsqlDriver();
```

- forName metódussal

```
try{ Class.forName
        ("com.imaginary.sql.msql.MsqlDriver");
} catch (ClassNotFoundException exc){}
```

- .class attribútummal

```
Class cl =
    com.imaginary.sql.msql.MsqlDriver.class;
```

- jdbc.drivers jellemző beállítása
  Hogy a kód semmi adatbáziskezelő-specifikus részt ne tartalmazzon...

jdbc.drivers

```
import java.sql.*;
class DBTeszt {
  public static void main(String args[])
  throws SQLException {
     Connection c =
          DriverManager.getConnection(args[0]);
     ...
  }
}
```

```
javac DBTeszt.java
java -classpath msql-jdbc-1-0.jar:.
  -Djdbc.drivers=com.imaginary.sql.msql.MsqlDriver
  DBTeszt jdbc:msql://localhost:1112/StockMarket
```

- Több meghajtó is betölthető (Windows ; és UNIX : elv.)

Java tutorial

Különböző adatbáziskezelők

- Egy programban több adatbázis is használható, akár több adatbáziskezelőn keresztül is
- Az áttérés egyik adatbáziskezelőről egy másikra viszonylag egyszerű
- Főleg, ha az SQL utasításokban megmaradunk a szabványos lehetőségeknél
- Nagyon sokféle adatbáziskezelőhöz készítettek már JDBC meghajtó programot (lásd Java honlap)
- Van egy JDBC-ODBC kapcsoló meghajtó, amivel minden olyan AB-kezelő használható, amihez van ODBC (Open Data Base Connectivity) interfész

Connection létrehozása

```
str = "jdbc:msql://localhost:1112/StockMarket";
Connection c = DriverManager.getConnection(str);
```

`jdbc:`*alprotokoll*`:`*alnév*

Például. `jdbc.odbc.Object.StockMarket`

- A paraméterként átadott sztringet végigkérdezi a regisztrált meghajtóktól. Az elsőt, amelyik tud vele mit kezdeni (alprotokoll), megbízza a munkával
- Kiválasztási sorrend: először a jdbc.drivers, utána az explicit betöltött osztályok

Statement

- SQL utasítások végrehajtására

```
Statement s = c.createStatement();
```

- Vannak lekérdezések és adatmanipulációk
- Lekérdezés

```
ResultSet rs = s.executeQuery
               ("select * from Stock");
```

- Adatmanipuláció

```
int sorok = s.executeUpdate
 ("insert into Stock values ('ABC',90)");
```

- Több eredményt adó SQL utasítás végrehajtása:

```
boolean vanEredmény = s.execute("...");
```

Java tutorial

ResultSet

- Eredménytábla, lekérdezés eredménye

```
ResultSet rs = s.executeQuery
           ("select * from Stock");
```

- A sorait egymás után feldolgozhatjuk

```
while( rs.next() ){ ... }
```

- Kezdetben a kurzor az első sor elé mutat
- Az aktuális sor komponenseit kiszedhetjük

```
String s = rs.getString(1));
```

- Egy Statement-hez egyszerre csak egy ResultSet lehet megnyitva

Eredménysorok feldolgozása

- Az aktuális sorból az oszlopok (komponensek) kiszedhetők a **`getXXX`** metódusokkal

```
String getString(int oszlop)
int getInt(int oszlop)
```

- Az oszlopok 1-től számozódnak
- Lehet az oszlop száma helyett a nevét megadni
  - kevésbbé hatékony
- Az oszlopokat balról jobbra kell végigolvasni, és csak egyszer

ResultSet a JDBC 2.0 szerint

- Az eredménytábla kurzorát előre is lehet mozgatni, lehet benne pozícionálni
- Lehet módosítani az eredménytáblát, és rajta keresztül az adatbázist
- Be lehet pl. állítani, hogy más SQL utasítások hatása ne jelenjen meg a feldolgozás alatt álló eredménytáblában

```
Statement stmt = con.createStatement(
   ResultSet.TYPE_SCROLL_INSENSITIVE,
   ResultSet.CONCUR_UPDATABLE );
```

Pozícionálás és sor módosítása

```
rs.absolute(5);
rs.updateString("Név",
                "Janis Joplin");
rs.updateRow();
```

- Szkrollozható és módosítható ResultSet
- JDBC 2.0 kell hozzá
- Az **`updateXXX`** metódusok szabályai ugyanazok, mint a **`getXXX`** metódusokéi

Sor beszúrása és törlése

```
rs.moveToInsertRow();
rs.updateString(1,"Sting");
rs.updateInt(2,35);
rs.updateBoolean(3,true);
rs.insertRow();
rs.moveToCurrentRow();
rs.deleteRow();
```

Java típusok és JDBC típusok

- getString, getInt, ...
  BIT, TINYINT, SHORTINT, INT, LONGINT
- java.math.BigDecimal - NUMERIC
- getASCIIStream - LONGVARCHAR
- getBinaryStream - LONGVARBINARY, getCharacterStream
- java.sql.Blob, java.sql.Clob
- java.sql.Date, Time, Timestamp

PreparedStatement

- Ha egy SQL utasítást többször is végre akarunk hajtani
- Előfordított SQL utasítás
  - Ha a meghajtó támogatja az előfordítást
- Hatékonyabb, mint többször egy Statement-et
- Paraméterezhető
  - setXXX metódusok
  - A paraméter típusának megfelelő setXXX kell
- A Statement leszármazottja

Példa

```
Alkalmazott[] beosztottak = ...

PreparedStatement s = c.prepareStatement(
   "insert into Alkalmazott" +
   "(Név, Fizetés, Id) values (?,?,?)"  );

for(int i=0; i<beosztottak.length; i++){
   s.setString(1, beosztottak[i].getNév());
   s.setInt(2, beosztottak[i].getFizetés());
   s.setInt(3, 100*i)
   s.executeUpdate();
}
```

Java tutorial

Callable Statement

- Nem-SQL utasítások, pl. tárolt eljárások végrehajtására
- JDBC eljáráshívási escape-szekvencia
  {call <*procedure-name*>[<*arg1*>,<*arg2*>, ...]}
  {?= call <*procedure-name*>[<*arg1*>, ...]}
- Lehetnek bemeneti és kimeneti paraméterei
  - És vegyes...
  - A kimenetiek típusát regisztráni kell végrehajtás előtt
- A visszaadott eredményeket (pl. ResultSet) előbb kell feldolgozni, mint a kimeneti paramétereket
- PreparedStatement leszármazottja

```
CallableStatement s = c.prepareCall
  ( "{call return_seats[?, ?, ?]}" );

s.setString(1, "MA-723");
s.registerOutParameter
           (2, java.sql.Types.BOOLEAN);
s.registerOutParameter
           (3, java.sql.Types.INTEGER);
s.execute();
// eredmények feldolgozása, ha vannak
boolean dohányzó = s.getBoolean(2);
int szabadHelyek = s.getInt(3);
```

Statement.execute()

```
boolean tábla = s.execute("...");
int sorok = s.getUpdateCount();
while ( tábla || (sorok != -1) ){
   if(tábla){
      ResultSet rs = s.getResultSet();
      // csinálunk rs-sel valamit
   } else {
      // csinálunk sorok-kal valamit
   }
   tábla = s.getMoreResults();
   sorok = s.getUpdateCount();
}
```

Java tutorial

javax.sql - szabványos kiterjesztés

- JNDI - logikai név az adatbázisok eléréséhez
  - függetlenség az adatbázis nevétől és pontos elérési útvonalától
- Adatbázis-kapcsolatok cache-elése
  - a kapcsolat felépítése nagyon időigényes
- JTA - Java Transaction API használata
  - kétfázisú protokoll tranzakciókezeléshez
- Adattáblák kezelése off-line

Többrétegű (multi-tier) alkalmazások

- Két rétegű modell: a felhasználói interfész az adatbázishoz kapcsolódik
- Három rétegű modell: közbeiktatunk egy szerveroldali programot
  - a felhasználói felület a hálózaton kapcsolódik a szerverhez
    - HTML form (+szervlet), applet, program
    - HTTP, TCP - UDP, RMI, CORBA
  - a szerver az adatbázis(ok)kal tartja a kapcsolatot
  - biztonság, megbízhatóság, rugalmasság, könnyebb használat, konfigurálhatóság

Tranzakciók

- Logikailag összetartozó adatbáziskezelő utasítások együtese
- Hajtódjon végre az egész (vagy semmi)
- Véglegesítés: commit, visszavonás: rollback
- Konkurrens adatbáziskezelés esetén problémák: sorbarendezhetőség (serializability)
  - Hatásban legyen olyan, mintha valamilyen sorrendbben egymás után hajtódtak volna végre
  - Inkonzisztens adatok elkerülése
  - Holtpont-veszély!

Tranzakciók készítése

- Alapértelmezett: automatikus nyugtázás
  - Minden SQL utasítás befejeződése után nyugtáz
    - Végrehajtódott, és nem ad vissza eredményt
    - Az utasítást tartalmazó SQL objektmot újra végrehajtjuk
    - Eredménytála utolsó sorát is feldolgoztuk
    - Az eredménytáblát lezárjuk
- Manuális nyugtázási mód
  - setAutoCommit
  - A programozó hívja meg a commit és rollback metódusokat

Tranzakció izolációs szintek

- A kapcsolathoz rendelhetjük (Connection)
- setTransactionIsolation
  - TRANSSACTION_NONE: nincs tranz. kez.
  - _READ_UNCOMMITED: olvasáskor mindig az aktuális értéket látjuk
  - _READ_COMMITED: olvasáskor mindig a legutóbbi véglegesített eredményt látjuk
  - _REPEATABLE_READ: a tranzakció által olvasott értékek megegyeznek a kezdeti értékkel
  - _SERIALIZABLE: a tranzakció ideje alatt az olvasott értékeket más tranzakciók nem írhatják felül

Hibakezelés

- SQLException
  - getMessage()
  - SQLstate (X/OPEN SQLstate szabvány szerint)
  - Hibakód (adatbáziskezelő-specifikus kód)
  - Hivatkozás a következő hibára (kiegészítések)
- SQLWarning
  - Automatikusn lekezelik a JDBC metódusok
  - Az SQL-objektumokhoz láncolva, listában
  - getWarnings(), clearWarnings()

DatabaseMetaData

- Az adatbázis jellemzői
- getMetaData()
- SQL fogalmak adatbázisspecikifus megvalósításai, annak korlátai
  - getCatalogSeparator(), getIdentifierQuoteString(), getMaxConnections(), getMaxColumnsInTable()
- Az adatbáziskezelő tudása
  - supportsFullJoins(), supportsResultSetConcurrency()
  - JDBC COMPLIANT
    - supportsColumnAliasing(), supportsSubqueriesInExists()...

Kötegelt (batch) végrehajtás

- Gyorsabb adatmanipulációs/definíciós utasítások

```
int[] sorok;
Statement s = c.createStatement();
s.addBatch("insert into Nevek" +
  "values ('Lennon', 'McCartney')");
s.addBatch("insert into Nevek" +
  "values ('Szőrényi', 'Bródy')");
sorok = s.executeBatch();
```

Java tutorial