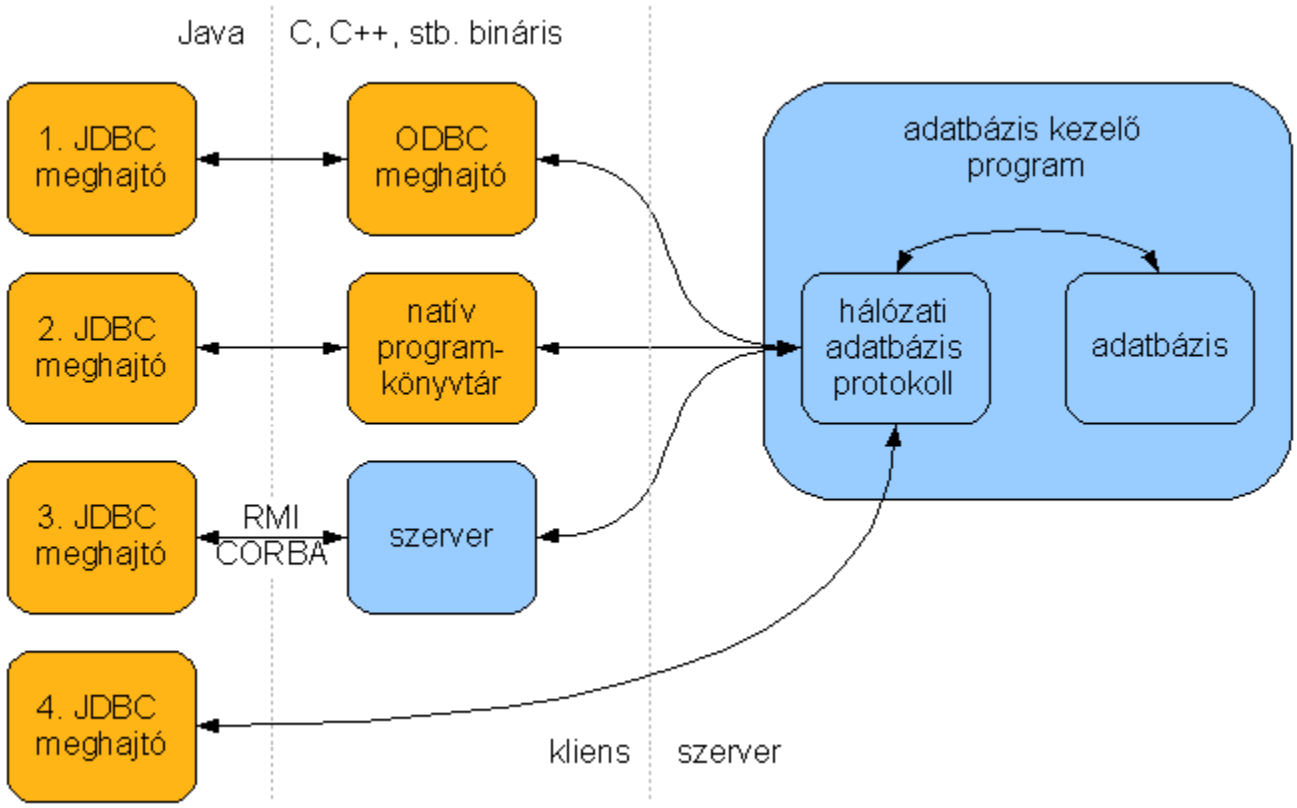
A JDBC használata

Kellene egy rövid leírás a JDBC-rl, valaki bevállalja? 🙂

JDBC meghajtók regisztrációja

A JDBC meghajtók menedzselését DriverManager osztály végzi. Annak biztosítása végett, hogy egyszerre több adatbázishoz is kapcsolódhassunk, létezik egy lista mely a regisztrált meghatókat tartalmazza. Ebbl a listából a DriverManager automatikusan kiválasztja az adott adatbázishoz megfelel meghajtót, levéve ezt a terhet a vállukról. A JDBC meghajtó betöltésekor automatikusan regisztrálja magát ebbe a listába, oly módon, hogy minden meghajtónak van egy statikus inicializátora, melyben a DriverManager osztály registerDriver statikus metódusát meghívja. Meghajtót betölteni kétféleképpen lehet, vagy név szerint hivatkozunk rá és a Class.forName statikus metódussal töltjük be, vagy pedig a jdbc.drivers rendszerparaméterben felsoroljuk – kettsponttal elválasztva – és a DriverManager tölti be ket az inicializáció során. Applet-ek esetén az utóbbi megoldás nem használható mert azok nem állíthatnak be rendszerparamétereket.



Kapcsolat objektum

Egy adatbázishoz kapcsolódni a DriverManager osztály getConnection statikus metódusával lehetséges, mely visszatérési értéke egy Connection típusú objektum. Ezen objektumon keresztül tudunk kommunikálni az adatbázissal. Paraméterként meg kell adni az adatbázis eléréséhez szükséges adatbázis-URL-t melynek a felépítése a következ: *<protokoll>:<alprotokoll>:<adatforrás leírása>*

- **protokoll** – jdbc
- **alprotokoll** – a JDBC meghajtó forgalmazója adja meg, általában az adatbázis kezel neve. MySQL esetén például mysql
- **adatforrás leírása**: az adatbázis eléréséhez szükséges egyéb információk, például adatbázis neve, felhasználó neve, felhasználó jelszava, karakterkódolások, stb. A szintaktikáját és a lehetséges paramétereket szintén a JDBC meghajtó forgalmazója adja meg, MySQL esetén például: http://dev.mysql.com/doc/refman/5.0/en/connector-j-reference-configuration-properties.html

Példaként nézzünk egy konkrét adatbázis-URL-t:

```
DriverManager.getConnection("jdbc:mysql://localhost/web?
user=sirkalmi&password=titkos&useUnicode=true&characterEncoding=utf8&autoReconnect=true");
```

Információ az adatbázisról

A Connection objektum getMetaData metódusával kapott DatabaseMetaData típusú objektumon keresztül lekérdezhetünk mindenféle hasznos információt az adatbázisunkról, a lekérdezett információknak három típusa létezik:

- táblázat szer – ezek egy ResultSet típusú objektumon keresztül járhatók végig, melyrl késbb még lesz szó,
- szöveg vagy szám formájában kapjuk meg, például valamelyik mez nevét vagy egy táblában lév oszlopok számát,
- logikai true/false értéket kapunk, például egy adott funkció támogatott-e az adatbázisban.

Az els két típusba tartozó lekérdez metódusoknak a nevei mindig get-el, az utóbbinál mindig supports-al kezddnek. A lekérdez metódusok egy részében, paraméterként megadhatunk keresési mintákat:

- null – az adott paraméter nem fog szerepelni a keresés kritériumaiban
- üres String – az adott paraméternek nem lehet semmilyen értéke
- % – akármilyen hosszú szöveg állhat a helyén
- _ – egy karakternyi szöveg állhat a helyén
- String – természetesen a keresés kritériumai a String értékének megfelelen alakulnak

Hibakezelés

Az adatbázis kapcsolat használata során különböz SQL hibákkal találkozhatunk, melyek súlyosságuk függvényében vagy SQLException-t vagy SQLWarnig-ot generálnak. Elbbi esetben megszakad programunk futása, ekkor egy SQLException típusú kivétel objektum keletkezik, melyet nekünk kell a megfelel helyen elkapni és lekezelni, utóbbi esetben csak egy hibaüzenetet kapunk, mely nem szakítja meg a programunk futását csak a hibaüzenet szövegét hozzáláncolja az aktuális, végrehajtás alatt lév adatbázis objektumhoz. Az SQLException típusú kivétel objektumok a következ információkat tartalmazzák:

- a hibaüzenet szövegét, melyet a getMessage metódussal tudunk lekérdezni,
- az X/Open SQLstate által megadott SQLstate szöveget, melyet a getSQLState metódussal tudunk lekérdezni,
- az adatbázis-kezel által visszaadott hibakódot, melyet a getErrorCode metódussal tudunk lekérdezni,
- a hivatkozást a következ kivétel objektumra, ezek ugyanis lánc szeren vannak felfzve, melyet a getNextException metódussal tudunk lekérdezni.

Lehetségünk van még továbbá a setNextException metódussal újabb kivételobjektumot fzni a lánc végére. Az SQLWarning típusú kivétel objektumhoz az aktuális, végrahajtás alatt lév adatbázis objektum getWarnings metódusával juthatunk hozzá (például Statement.getWarnings, ResultSet.getWarnings, Connection.getWarnings), magához a hibaüzenethez pedig a getNextWarning metódussal. Itt is lehetségünk van egy setNextWarning matódussal újabb hibaüzenet hozzáfzésére a lánchoz.

Tranzakciókezelés

A tranzakciókezelés akkor jöhet jól, ha több logikailag összetartozó SQL utasítást kell egymás után, egy menetben végrehajtanunk. Abban az esetben például, ha ötezer forintot át akarunk utalni egyik számláról a másikra, akkor ezt csak több SQL utasítással tudjuk megtenni, hiszen elbb az egyik számlát meg kell terhelni, majd a másik számlán jóvá kell írni ezt az összeget. Gond akkor adódhat, ha valamelyik mvelet meghiúsul, mert ebben az esetben vagy a kezdeményez számláját vezet bank vagy pedig a kedvezményezett lesz rövidebb ötezer forinttal. Az ilyen és ehhez hasonló problémákat megúszhatjuk, ha tranzakciókezelést alkalmazunk. Ez úgy mködik, hogy ténylegesen csak akkor hajtódnak végre az SQL utasítások, ha mi erre a programunkból direkt felszólítjuk, egészen addig az adatbázis-kezel elkülönítve kezeli ket. A tranzakció közben bármikor lehetség van az addig kiadott SQL utasítások érvénytelenítésére, azaz bármilyen hiba esetén visszaforgathatjuk az eredeti állapotra az adatbázisunkat.

A Connection objektum alábbi metódusaival kezelhetjük a tranzakciókat:

- setAutoCommit(true) – metódusával kapcsolhatjuk be a tranzakciókezelést mely alapesetben ki van kapcsolva.
- commit() – metódussal véglegesíthetjük a tranzakciót.
- rollback() – metódussal érvényteleníthetjük a tranzakciót.

Tranzakció-izolációs szintek

A tranzakció-izolációs szintek beállításával azok az esetek kezelhetk amikor több tranzakció egyazon idben dolgozik ugyanazokkal az adatokkal. Ilyen eset lehet például, ha a fentebb részletezett banki tranzakció közben egy másik tranzakció lekérdezi az átutalást kezdeményez számláját még mieltt az véglegesítve lenne. Hogy viselkedjen ekkor az adatbázis? Milyen értéket adjon vissza az eredeti egyenleget vagy a már megterhelt egyenleget adja vissza? Ezeket a viselkedéseket meg tudjuk határozni programunkból a Connection objektumon keresztül:

- TRANSACTION_NONE – nincs tranzakciókezelés
- TRANSACTION_READ_UNCOMMITED – olvasáskor mindig az aktuális értéket kapjuk
- TRANSACTION_READ_COMMITED – olvasáskor mindig a véglegesített értéket kapjuk
- TRANSACTION_REPEATABLE_READ – más tranzakció véglegesít hatása ellenére is mindig a mi tranzakciónk kezdetekor érvényben lév értéket kapjuk
- TRANSACTION_SERIALIZABLE – más tranzakció nem írhatja felül a mi tranzakciónk által olvasott értékeket, azaz addig várakoztatja azokat míg be nem fejezdik a tranzakciónk

A szintek föntrl lefele növekednek és minél nagyobb szintet állítunk be az adatbázis-kezelnek annál több adminisztrációs feladata lesz, ezáltal a tranzakciók végrehajtási sebessége is lassul. Az izolációs szinteket a Connection objektum setTransactionIsolation() metódussal állíthatjuk be még a tranzakció megkezdése eltt.