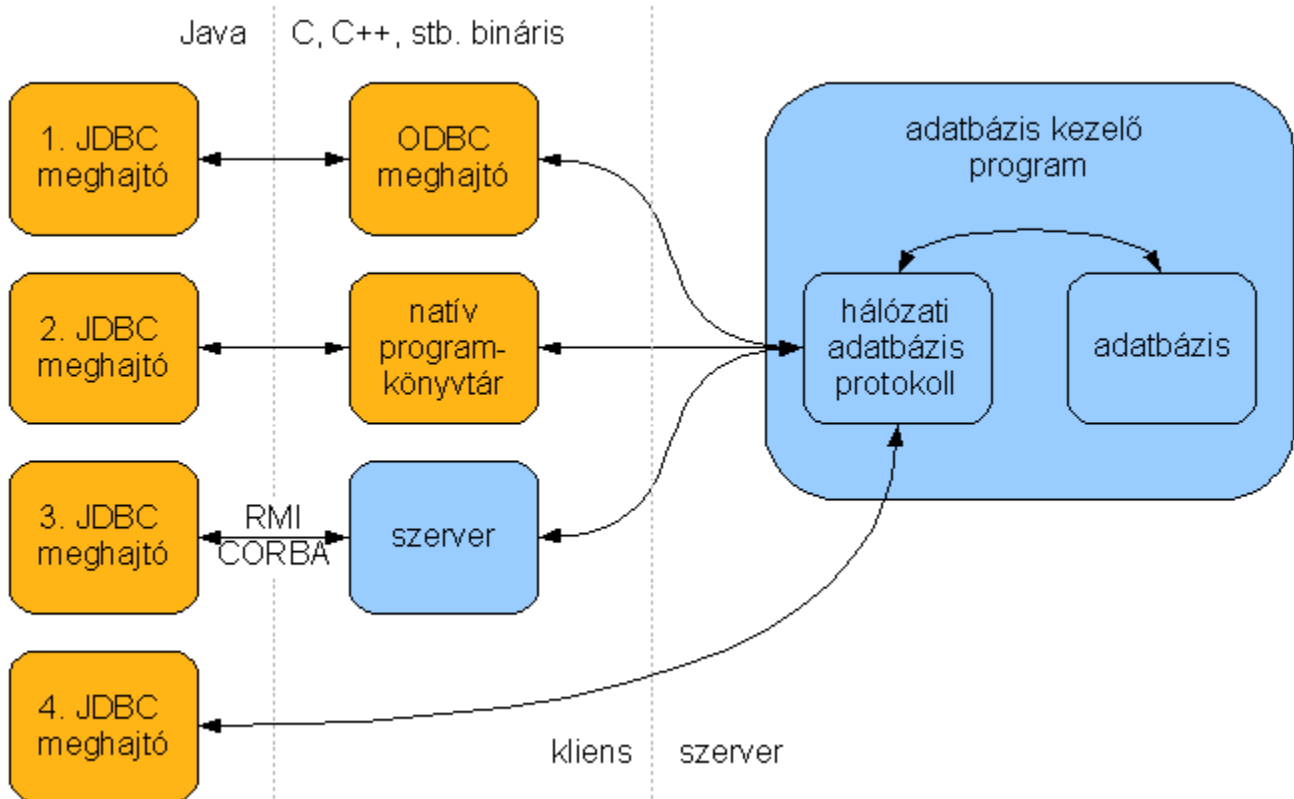


# A JDBC használata

 Kellene egy rövid leírás a JDBC-ről, 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 meghajtókat tartalmazza. Ebből a listából a DriverManager automatikusan kiválasztja az adott adatbázishoz megfelelő 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 következő: `<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 lekérdezhethetünk mindenféle hasznos információt az adatbázisunkról, a lekérdezett információknak három típusa létezik:

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

Az első két típusba tartozó lekérdezési metódusoknak a nevei mindig `get`-el, az utóbbinál mindig `supports`-al kezdődnek. A lekérdezési metódusok egy részében, paraméterként megadhatunk keresési mintákat:

- `null` – az adott paraméter nem fog szerepelni a keresési 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 keresési kritériumai a `String` értékének megfelelően alakulnak

## Hibakezelés

Az adatbázis kapcsolat használata során különböző SQL hibákkal találkozhatunk, melyek súlyosságuk függvényében vagy `SQLException`-t vagy `SQLWarning`-ot generálnak. Előbbi esetben megszakad programunk futása, ekkor egy `SQLException` típusú kivétel objektum keletkezik, melyet nekünk kell a megfelelő 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 lévő adatbázis objektumhoz. Az `SQLException` típusú kivétel objektumok a következő 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-kezelő által visszaadott hibakódot, melyet a `getErrorCode` metódussal tudunk lekérdezni,
- a hivatkozást a következő kivétel objektumra, ezek ugyanis lánc szeren vannak felfűzve, melyet a `getNextException` metódussal tudunk lekérdezni.

Lehetségünk van még továbbá a `setNextException` metódussal újabb kivételobjektumot fűzni a lánc végére. Az `SQLWarning` típusú kivétel objektumhoz az aktuális, végrehajtás alatt lévő adatbázis objektum `getWarnings` metódusával juthatunk hozzá (például `Statement.getWarnings`, `ResultSet.getWarnings`, `Connection.getWarnings`), magához a hibaüzenethez pedig a `getNextWarning` metódussal. Itt is lehetőségünk van egy `setNextWarning` metódussal újabb hibaüzenet hozzáfűzésére a láncához.

## 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 előbb az egyik számlát meg kell terhelnünk, majd a másik számlán jóvá kell írni ezt az összeget. Gond akkor adódhat, ha valamelyik művelet meghiúsul, mert ebben az esetben vagy a kezdeményező számláját vezet bank vagy pedig a kedvezményezett lesz rövidebb ötezer forinttal. Az ilyen és ehhez hasonló problémákat megúszhatjuk, ha tranzakciókezelést alkalmazunk. Ez úgy működik, hogy ténylegesen csak akkor hajtodnak végre az SQL utasítások, ha mi erre a programunkból direkt felszólítjuk, egészen addig az adatbázis-kezelő elkülönítve kezeli ket. A tranzakció közben bármikor lehetőség 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 kezelhetők amikor több tranzakció egyazon időben dolgozik ugyanazokkal az adatokkal. Ilyen eset lehet például, ha a fentebb részletezett banki tranzakció közben egy másik tranzakció lekérdezi az átutalást kezdeményező számláját még mielőtt 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_UNCOMMITTED` – olvasáskor mindig az aktuális értéket kapjuk
- `TRANSACTION_READ_COMMITTED` – olvasáskor mindig a véglegesített értéket kapjuk
- `TRANSACTION_REPEATABLE_READ` – más tranzakció véglegesítési hatása ellenére is mindig a mi tranzakciónk kezdetekor érvényben lévő értéket kapjuk
- `TRANSACTION_SERIALIZABLE` – más tranzakció nem írhatja felül a mi tranzakciónk által olvasott értékeket, azaz addig várakoztatja azokat míg be nem fejeződik a tranzakciónk

A szintek fölött lefele növekednek és minél nagyobb szintet állítunk be az adatbázis-kezelőnek 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 előtt.