A JDBC adatbázis kapcsolat

Az elz cikkben megnéztük, hogyan lehetséges létrehozni egy szervletet és ez a szervlet hogyan küldhet adatokat a böngésznek. Mivel a megjelenítend adatokat általában egy adatbázisból vesszük ezért ebben a cikkben megnézzük, hogyan mködik mindez. Tovább bvítjük elz programunkat amely ezentúl nem a világot fogja köszönteni, hanem a felhasználót. Ha a szervlet nem tudja a felhasználó nevét akkor bekéri és tárolja azt egy adatbázisban, majd legközelebb már innen fogja kiolvasni. Adatbázisként a MySQL-t választottam mert én ezt ismerem, de mivel a Java JDBC bármely adatbázissal képes együttmködni melyhez adnak JDBC meghajtót, ezért nem muszáj ehhez ragaszkodni.

Nézzük lépésrl lépésre, mik a tennivalók:

- Töltsük le a MySQL JDBC meghajtót, ha más adatbázis programot használunk akkor itt érdemes keresni hozzá driver-t.
- Másoljuk be az alkalmazásunk WebContent/WEB-INF/lib könyvtárja alá a mysql-connector-java-3.1.12-bin.jar file-t.

Hozzuk létre az adatbázisban a táblát:

```
CREATE TABLE `Names` (
`ID` char(32) collate utf8_hungarian_ci NOT NULL default '',
`Name` varchar(40) collate utf8_hungarian_ci NOT NULL default ''
) ENGINE=MyISAM DEFAULT CHARSET=utf8 COLLATE=utf8_hungarian_ci
```

A Java szervletek életciklusából fakadó elnyt kihasználva logikus a kapcsolatot többször felhasználhatóvá tenni, azaz mindegyik kéréshez ugyanazt a kapcsolatot használjuk fel, így nem kell állandóan felépíteni majd lebontani azt. Ehhez a Connection con referenciát osztályszinten deklaráljuk és a szervlet init metódusában inicializáljuk ahol is betöltjük a JDBC meghajtót – és ezzel egyben regisztráljuk azt – majd létrehozzuk a kapcsolatot. Az init metódusról elég annyit tudnunk egyelre, hogy garantáltan az els kérés kiszolgálása eltt hajtódik végre egyszer:

```
/**
 * A környezet inicializáló paramétereinek lekérdezése
 */
String driver = getServletContext().getInitParameter("connection.driver");
String url = getServletContext().getInitParameter("connection.url");
String user = getServletContext().getInitParameter("user");
String password = getServletContext().getInitParameter("password");
String useUnicode = getServletContext().getInitParameter("useUnicode");
String characterEncoding = getServletContext().getInitParameter("characterEncoding");
String autoReconnect = getServletContext().getInitParameter("autoReconnect");

try
{
  Class.forName(driver);
  con = DriverManager.getConnection(url + "?user=" + user
      + "&password=" + password
      + "&useUnicode=" + useUnicode
      + "&characterEncoding=" + characterEncoding
      + "&autoReconnect=" + autoReconnect );
} catch (ClassNotFoundException e)
{
  System.out.println("\nNem lehet betölteni az adatbázis meghajtót!\n" + e);
} catch (SQLException e)
{
  System.out.println("\nNem lehet létrehozni az adatbázis kapcsolatot!\n" + e);
}
```

Tehát a Class.forName metódus segítségével betöltjük a JDBC meghajtót a memóriába. Ez mindjárt regisztrálja is magát a java.sql.DriverManager segítségével így elérhetvé válik. Ha kíváncsiak vagyunk akkor bele lehet nézni a com.mysql.jdbc.Driver forráskódjába. Ezek után inicializálhatjuk a Connection con refernciát a DriverManager.getConnection(URL) metódussal. A különböz adatbázisokhoz való kapcsolódás URL-en keresztül lehetséges, mely eltér lehet az egyes adatbázis programoknál. Alább a MySQL-hez való kapcsolódást mutatom be, ha mást használunk akkor nézzünk utána az adatbázis programot szállító szervezet, vállalat honlapján ahol biztosan találunk ezzel kapcsolatos dokumentációt. MySQL esetén az URL -nek az alábbi szerkezetnek kell lennie:

```
jdbc:mysql://[host][,failoverhost...][:port]/[database][?propertyName1][=propertyValue1][&propertyName2]
[=propertyValue2]...
```

Alapértelmezett értékek, ha nem adunk meg semmit:

- host – 127.0.0.1 azaz a localhost
- port – 3306 azaz a MySQL alapértelmezett portja

Az URL szerkezetérl és paramétereinek részleteirl itt találsz további információt. Célszer az URL-ben lév paraméter értékeit a web.xml file-ban tárolni és onnan kiolvasni a ServletContext.getInitParameter() metódus segítségével mert így, ha valamilyen változás van akkor nem kell a programot piszkálni, csak a web.xml file-t szerkeszteni.

```
<!-- Adatbázis kapcsolat felépítéséhez szükséges paraméterek -->
<context-param>
  <param-name>connection.driver</param-name>
  <param-value>com.mysql.jdbc.Driver</param-value>
</context-param>
<context-param>
  <param-name>connection.url</param-name>
  <param-value>jdbc:mysql://localhost/web</param-value>
</context-param>
<context-param>
  <param-name>user</param-name>
  <param-value>servlet</param-value>
</context-param>
<context-param>
  <param-name>password</param-name>
  <param-value>blablabla</param-value>
</context-param>
<context-param>
  <param-name>useUnicode</param-name>
  <param-value>true</param-value>
</context-param>
<context-param>
  <param-name>characterEncoding</param-name>
  <param-value>utf8</param-value>
</context-param>
<context-param>
  <param-name>autoReconnect</param-name>
  <param-value>true</param-value>
</context-param>
```

A forName metódus ClassNotFoundException kivételt a getConnection metódus pedig SQLException kivételt válthat ki. Ezeket el kell kapni különben fordítási hibát kapunk.

Hogy név szerint köszönthessük a felhasználót, valahogyan azonosítani kell azt. Általában ugye be kell jelentkeznünk a felhasználói nevünk és jelszavunk megadásával. Azért, hogy egyszersítsük a dolgot mert a példa szempontjából most jelentéktelen ez, azt találtam ki, hogy nemes egyszerséggel a sessionId fogja azonosítani kuncsaftunkat. Így létrehozunk egy session-t és végtelenre állítjuk az életét neki:

```
HttpSession session = request.getSession();
session.setMaxInactiveInterval(-1);
```

Majd ez után jön a lényeg: elvégzünk egy adatbázis lekérdezést ami azt vizsgálja, hogy az adott sessionId-vel (felhasználóval) találkozott-e már a szervletünk. Ha talál ilyen bejegyzést akkor név szerint tudja köszönteni a felhasználót, ha nem akkor pedig egy egyszer form-on keresztül megkérdezi a nevét.

```
ResultSet rs = null;
try
{
  PreparedStatement pstmt = con.prepareStatement("SELECT Name FROM Names WHERE ID = ?");
  pstmt.clearParameters();
  pstmt.setString(1, session.getId());
  rs = pstmt.executeQuery();
  if(rs.next())
  {
    out.println("<p>Hello " + rs.getString("Name") + "!</p>");
  } else
  {
    out.println("<form action=\"\" method=\"post\">");
    out.println("<p>Mi a neved cimbora?</p>");
    out.println("<input name=\"name\" type=\"text\" />");
    out.println("<input type=\"submit\" value=\"OK\" />");
    out.println("</form>");
  }
} catch (SQLException e)
{
  e.printStackTrace();
}
```

Mint látható létrehoztunk egy PreparedStatement típusú objektumot, mely paramétereinek beállítása után végrehajtjuk az SQL utasítást. A lekérdezés eredménye az rs objektumban jön létre melyet soronként lekérdezhetünk. Mivel most biztosak vagyunk benne, hogy csak maximum egy találatot kaptunk így nem szükséges ciklusban végigjárni. Az rs.next() metódus mindaddig true értékkel tér vissza míg el nem éri az utolsó sor utáni pozíciót. Tehát ha van találat akkor arra pozicionálja a mutatót ha nincs akkor egybl az utolsó sor utánra lép így false értéket ad vissza. Szándékosan nem részletezem jobban a kódot bár lehetne még errl sokat írni, ugyanis készülök egy JDBC cikkel is és majd ott kivesézem a dolgokat.

Az elbbi kódrészletbl láthatjuk, hogy ha van felhasználó akkor köszöntjük, ha nincs akkor az else ágban egy form-ot iratunk ki, mely meghívja a post metódust és paraméterül megadja a felhasználó nevét. Ebben a metódusban semmi mást nem csinálunk mint, hogy a PreparedStatement segítségével felveszünk egy új rekordot az adatbázisunkba a felhasználó nevével és a session id -el, majd a response.sendRedirect-el visszahívjuk a get metódust:

```
try
{
  PreparedStatement pstmt = con.prepareStatement("INSERT INTO Names (ID, Name) VALUES (?, ?)");
  pstmt.clearParameters();
  pstmt.setString(1, session.getId());
  pstmt.setString(2, name);
  pstmt.execute();
  pstmt.close();
} catch (SQLException e)
{
  e.printStackTrace();
}
response.sendRedirect("./" + getServletName());
```

Ez a metódus akkor hívódik meg, ha már jó ideje nem érkezett be kérés és várhatóan nem lesz szükség a szervletünkre. Ebben a metódusban kell felszabadítani azokat az erforrásokat, melyeket a szervletünk lefoglalt mködése során. Esetünkben ez a con objektum által fogvatartott adatbázis kapcsolat melyet felszabadítunk abban az esetben, ha tényleg szükséges:

```
try
{
  if (con != null) con.close();
} catch (SQLException e)
{
  e.printStackTrace();
}
```

Ez a köszöns példa meglehetsen primitív és életszerbb alkalmazások fejlesztése közben szembe fogunk találkozni olyan problémákkal amikrl itt nem ejtettem szót. A cikk tényleg csak az els sikerélmény megszerzésének a segítségére született, innentl kezdve rajtunk a sor! A cikkben szerepl példaprogram egyben leszedhet innen.