Diplomamunka

Szathmáry László
2001

Adat- és adatbáziskezelés World Wide Web felületen

Diplomamunka

Készítette:
Szathmáry László
PTM

Témavezető:
Böttkös László

Debreceni Egyetem
2001

Tartalomjegyzék

Bevezetés

A dolgozatom tárgyául választott „Adat- és adatbáziskezelés World Wide Web felületen” című téma feldolgozásánál egyrészt az egyéni érdeklődés, másrészt a téma időszerűsége motivált.

A World Wide Web egyre növekvő népszerűségének köszönhetően egyre jobban emelkednek a tartalmat adó oldalakkal szemben támasztott igények is. Pár évvel ezelőtt még elegendő volt egy egyszerű szerkesztő programmal elkészíteni néhány jól „dizájnolt” oldalt, s az máris rendkívül sok látogatót vonzott. Ezzel szemben a harmadik évezredben már csak az állandóan változó, nagy információtömeget hordozó, és a megfelelő interaktivitást biztosító site-ok számíthatnak igazán nagy látogatottságra, s ebből következően nagy profitra. Ezt a nagy információigényt, s főképp az interaktivitást már nem lehet biztosítani egyszerű HTML oldalakkal, de még önmagukban a dinamikus oldalakat előállító nyelvekkel sem, mint amilyen a PHP. A megnövekedett olvasói igények kielégítéséhez szükségünk van az adatok könnyű tárolását és elérhetőségét biztosító adatbázis-szerverekre.

A dolgozatomban bemutatásra kerülő technológiákat, szoftvereket Linux operációs rendszer alatt próbáltam ki. Miért a Linuxot választottam? A szakirodalom alapos tanulmányozása és személyes tapasztalataim alapján arra a meggyőződésre jutottam, hogy a Linux egy nagyon nagy teljesítményű, testreszabható, nagy megbízhatóságú operációs rendszer. Fejlesztése folyamatos; illetve számos lelkes felhasználója van, akik örömmel megosztják tapasztalataikat a levelezési listákon, melyet én is többször igénybe vettem.

A Linux használata mellett szól például az Idaya cég (www.idaya.co.uk) vezetésével készült felmérés is, melyet 2001 januárjától márciusig a világ ezer legnagyobb internetszolgáltatója között végeztek. A felmérés szerint 2001-ben a Linux-piacon legalább 150 %-os növekedés várható. A webszerverek világában a Linux 2002 közepére jut majd domináns szerephez és 2003 végére a Linux vezető szerepű lesz a web hosting szolgáltatások terén is.

A számos, mindenki által elérhető Linux disztribúciók közül a Mandrake-et választottam, annak is a jelenleg legfrissebb 7.2-es változatát. Azért esett erre a választásom, mert könnyen kezelhető, s itt is biztosított a csomagok folyamatos frissítése. Használata annyira egyszerű, hogy a Windowsról Linuxra áttérőknek is ezt a disztribúciót ajánlják.

A dolgozatban szereplő példák kipróbálását Apache webszerveren végeztem el. A Netcraft cég havonta megjelenő felmérései (www.netcraft.com/survey) alapján a webszerverek piacán az Apache és az IIS áll az első két helyen, de az Apache előnye jelentős.

A következő technológiákat vizsgáltam meg: Java (JDBC adatbázis-kapcsolat, szervletek, JSP), PHP, Perl-CGI.

A Java és a PHP használata egyre népszerűbb. A PHP nyelvről magyar nyelvű szakirodalom (még) alig található, így dolgozatomban igyekeztem ezt a hiányt pótolni.

A Perl-CGI ugyan veszített korábbi népszerűségéből, de még most is rengeteg website-on használják, ezért úgy döntöttem, hogy egy fejezetet ennek ismertetésére szentelek.

Úgy érzem, hogy egy ilyen témával foglalkozó dolgozatban elengedhetetlen egy konkrét webes alkalmazás bemutatása. Választásom egy webes fórumra esett, melyet az Esettanulmány című fejezetben mutatok be. Az itt ismertetett alkalmazás továbbfejleszthető, alapul szolgálva más, hasonló jellegű alkalmazásokhoz.

Két egymáshoz közelálló adatbáziskezelő-rendszert próbáltam ki, a PostgreSQL-t és a MySQL-t. A PostgreSQL pontosabban implementálja az SQL-92 szabványt mint a MySQL, de emiatt egy kicsivel lassabb annál. Mindkettő jól használható kis- és középvállalkozások esetében.

A mellékletek fontos alkotórészét képezik a dolgozatnak; itt helyeztem el az egyes szoftverek installálására, konfigurálására vonatkozó útmutatásokat.

A munkám során felhasznált valamennyi szoftver – beleértve az operációs-rendszert is – ingyenesen használható.

Dolgozatomat elsősorban azoknak ajánlom, akik a web-es technológiák iránt érdeklődnek.

1. Java

1.1. A Java programozási nyelv

Az Internet ill. a World Wide Web a ma legdinamikusabban fejlődő információs és kommunikációs hálózat. A Java nyelv a Sun Microsystems cég nevéhez kötődik, s nyílt, heterogén, elosztott rendszerek nyelveként jelenik meg. A Java különlegessége abban rejlik, hogy a korábbi nyelvekkel ellentétben direkt **hálózati nyelv**ként hozták létre!

A Java tekinthető a C++ továbbfejlesztett változatának; jóval tisztább objektumorientált nyelv mint a C++.

A Java segítségével lehetőségünk nyílik hordozható interaktív alkalmazások készítésére. A Java a fordítóprogramos ill. az interpretált nyelvek között helyezkedik el, ui. a forráskódot egy köztes byte-kód formátumú tárgykódra fordítja le. Ez a tárgykód még nem futtatható önmagában, de szabadon hordozható az egyes platformok között, s a megfelelő interpreterrel már futtathatóvá válik. Az interpreter a JVM (Java Virtual Machine) segítségével hajtja végre a programot, azaz a tárgykódot a JVM interpretálja.

A Java elterjedését az is segítette, hogy a programok írásához szükséges fejlesztői eszközök (fordítóprogram, futtató környezet, dokumentációk) mindenki számára szabadon elérhetőek, ingyenesek (http://java.sun.com).

Sokan úgy vélik, hogy a Java túl lassú. Ezt leginkább applet-ek, ill. Swing-es Java alkalmazások esetén lehet megfigyelni. Azonban maga a Java gyors, csupán az ablakkezelő rendszere nagyon lassú. A továbbiakban szerveroldali Java programozással foglalkozunk (szervletek, JSP-k), ahol nincs szükség a grafikus felhasználói felületre (GUI). Mivel a szervletek/JSP programok képesek két hívás között a memóriában tárolni az adatokat, így valójában lényegesen gyorsabbak, mint a CGI megfelelőik.

1.2. JDBC – adatbázis-programozás Java-ban

Mivel a Java nyelvet direkt hálózati nyelvként hozták létre (a korábbi nyelvekkel ellentétben), ezért ideális kliens-szerver architektúrájú programok létrehozására. Becslések szerint a szoftverfejlesztések fele tartalmaz kliens-szerver műveleteket, így aztán a Java-ban hamar meglátták a platformfüggetlen kliens-szerver alkalmazások elkészítésének a lehetőségét. Ennek megvalósulását a JDBC (Java DataBase Connectivity) biztosítja.

Az adatbázisnyelvek között van egy szabványos nyelv, az SQL (Structured Query Language), de a szabvány ellenére általában tudni kell, hogy most éppen melyik gyártó termékével dolgozunk. A JDBC-t pontosan emiatt hozták létre: használatával programjaink nemcsak platformfüggetlenek, hanem még adatbáziskezelő-függetlenek is lesznek. (Emellett természetesen lehetséges gyártó-specifikus hívások elvégzése JDBC-n keresztül, de tudnunk kell, hogy ezzel sérül a hordozhatóság).

Egy új adatbázis-tábla létrehozásánál (ez az SQL `CREATE TABLE` utasításával történik) meg kell adnia a programozónak minden egyes oszlop típusát. És sajnos ez az a rész, ahol az egyes gyártók termékei lényegesen eltérhetnek egymástól. Különböző adatbázisok ugyanazon szemantikával és struktúrával rendelkező típusoknak is más-más nevet adhatnak. Például a legtöbb adatbázis támogatja a nagyméretű bináris típust: az Oracle ezt a típust `LONG RAW`-nak, a Sybase `IMAGE`-nek, az Informix `BYTE`-nak hívja. Ezért ha a hordozhatóság a célunk, próbáljunk meg **csak általános** SQL **típusazonosítókat** használni!

A JDBC API (Application Programming Interface – programozói interfész) tervezésekor az egyszerűség volt a cél. A metódushívások a következő három csoportba sorolhatók:

- kapcsolódás az adatbázishoz
- SQL utasítás megfogalmazása, végrehajtása
- eredményhalmaz vizsgálata

1.2.1. Két- és háromrétegű adatbázis-elérési modell

A JDBC a következő adatbázis-elérési modelleket támogatja:

- Kétrétegű modell: a program közvetlenül az adatbázis-kezelő rendszerrel kommunikál. Maga az adatbázis akár másik gépen is elhelyezkedhet, mint ahol a program fut, az adatforgalom pedig hálózaton keresztül folyik. Ezt az esetet nevezzük kliens-szerver

konfigurációnak, ahol az adatbázist tároló gép a szerver, a programot futtató gép pedig a kliens.

- Háromrétegű modell: a program adott protokollon keresztül egy „középső” szolgáltató réteggel kommunikál. Ez a réteg a programtól kapott parancsokat értelmezi és átalakítja, majd továbbítja azokat az adatbázis-kezelő rendszerhez. A lekérdezési eredményeket a program szintén a szolgáltató rétegen keresztül kapja meg. Ezen középső réteg bevezetése lehetővé teszi az adatbázis hozzáférések könnyű ellenőrzését és optimalizálását is. A szolgáltató réteg Java implementációja esetén az adatbázissal JDBC-n keresztül történik a kommunikáció.

1.2.2. JDBC meghajtóprogramok

A platformfüggetlenség megvalósításához JDBC meghajtóprogram(ok)ra lesz szükségünk, méghozzá annyira, ahány gyártó termékéhez szeretnénk csatlakozni. A JDBC hívások végrehajtásakor mindig fizikailag is fel kell venni a kapcsolatot a felhasznált adatbázissal. Az egyes adatbázis-kezelők esetén a JDBC hívások megfelelő értelmezését és kiszolgálását a JDBC meghajtóprogramok végzik el. A megfelelő meghajtóprogram osztályát direkt módon a **Class.forName()** statikus metódussal tölthetjük be dinamikusan.

1.2.3. Adatbázis URL-ek

Egy adatbázis URL az elérni kívánt adatbázist jelöli. Általános alakja:
protokoll:alprotokoll:adatforrás , ahol:

- protokoll: **jdbc**
- az alprotokoll nevét a megfelelő meghajtóprogram forgalmazója határozza meg, így az rendszerint egyezik a forgalmazó nevével
- az adatforrás a kért adatbázis eléréséhez szükséges további adatokat tartalmazza, mint pl. adatbázis hálózati címe, adatbázis neve, a felhasználó neve, a felhasználó jelszava

Miután készen állunk az adatbázishoz való kapcsolódásra, ezt a statikus **DriverManager.getConnection()** metódussal tehetjük meg, melynek a következő paramétereket kell átadni: adatbázis URL, felhasználó neve, felhasználó jelszava. A metódus

egy **Connection** objektumot ad vissza, amely segítségével hozzáférhetünk mostmár az adatbázishoz: lekérdezéseket végezhetünk, manipulálhatjuk, stb.

A következő egyszerű példa egy URL cím utolsó 2 (esetleg 3) karakterét kéri be, s megadja annak jelentését (például: fr → france, hu → hungary, stb.).

```
import java.io.*;
import java.sql.*;

public class Postgres
{
   public static void main(String[] args)
   {
      try {
         Postgres test = new Postgres();
      } catch(Exception exc)
      {
         System.err.println("Exception caught!\n" + exc);
         exc.printStackTrace();
      }
   }

   public Postgres() throws  ClassNotFoundException, FileNotFoundException,
                             IOException, SQLException
   {
      Class.forName("org.postgresql.Driver");
      Connection                              conn                              =
      DriverManager.getConnection("jdbc:postgresql://127.0.0.1/"+"areas",
          "jabba","jabba");
      Statement  st   = conn.createStatement();

      System.out.print("Code: ");
      BufferedReader r =
          new BufferedReader(new InputStreamReader(System.in));
      String code = r.readLine();

      ResultSet rs =
          st.executeQuery("select name from codes where code='"+code+"'");
      if (rs!=null)
      {
         while (rs.next())
         {
            System.out.println(rs.getString("name"));
         }
      }

      rs.close();
      st.close();
      conn.close();
   }
}
```

Egy JDBC-t is használó Java programban a `java.sql` JDBC interfészt importálni kell.
Miután létrejött a kapcsolat a **DriverManager.getConnection()** segítségével, a visszakapott **Connection** objektum felhasználható egy **Statement** objektum létrehozására a

createStatement() metódushívással. Az eredményül kapott **Statement** objektum **executeQuery()** metódusának ezután átadható egy SQL-92 szabványnak megfelelő SQL utasítás sztring formájában.

Az **executeQuery()** metódus egy **ResultSet** objektumot eredményez, ami egy iterátor: a **next()** metódus az iterátort a következő rekordra lépteti, vagy **false**-t ad vissza, ha elérte az eredményhalmaz végét. Az **executeQuery()** mindig egy **ResultSet** objektumot ad vissza, még akkor is, ha a lekérdezés eredménye egy üres halmaz. Mielőtt megpróbálnánk bármit is kiolvasni az eredményhalmazból, egyszer meg kell hívnunk a **next()** függvényt. Üres halmaz esetén a **next()** első hívása **false**-t fog visszaadni. Az eredményhalmaz minden egyes rekordja esetén kiválaszthatjuk az egyes mezőket, megadva a megfelelő oszlop nevét (sztringként) vagy sorszámát (hiszen egy relációs táblában az oszlopok sorrendje rögzített). Az oszlop nevének megadásakor mindegy a kis- vagy nagybetű – egy SQL adatbázis esetén ennek nincs jelentősége. A visszakapott érték típusát mi határozhatjuk meg aszerint, hogy melyik függvényt hívjuk meg: **getInt()**, **getFloat()**, **getString()**, stb. Így az adatbázis adatai már a Java saját formátumába kerültek át, s ezután a Java programban már azt kezdünk vele, amit akarunk.

1.2.4. A JDBC használata

A JDBC API típusait a `java.sql` és a `javax.sql` csomagok tartalmazzák, melyek már a Java 2 részei, ezért a JDK 1.2-es változata már tartalmazza őket. A korábbi JDK verziók esetén még külön kellett ezeket a csomagokat installálni.

A JDBC egyik legkézenfekvőbb felhasználási területe a böngészőprogramokkal történő adatlekérdezés és módosítás appletek/szervletek segítségével. Szervletek esetén csak maga a szervlet használja a JDBC-t adatbázis-elérésre a szerveroldalon, a böngészőprogram pedig HTML űrlapok, ill. dinamikusan generált HTML oldalak formájában olvashatja, ill. módosíthatja az adatokat.

1.3. Szervletek

Az Interneten keresztül számos felhasználó (kliens) férhet hozzá könnyedén az adatokhoz, erőforrásokhoz. Ez a fajta hozzáférés a klienstől csupán a következő World Wide Web szabványok használatát követeli meg:

- Hypertext Markup Language (HTML)
- Hypertext Transfer Protocol (HTTP)

Ha egy Internet kliens egy adatbázisba akar például adatokat felvinni, akkor ezt hagyományosan úgy oldják meg, hogy a kliens kap egy HTML lapot néhány szövegmezővel, ill. egy „elküld" (submit) gombbal. A felhasználó beírja a megfelelő információt a szövegmezőkbe, majd megnyomja az „elküld" gombot. Az adatokat megkapja egy CGI (Common Gateway Interface) program a szerveroldalon. A CGI programok leggyakrabban Perl, Python, C, C++ nyelven készülnek, de bármilyen nyelv felhasználható, amely tud olvasni a standard input-ról, ill. tud írni a standard output-ra. A webszerver feladata csupán az, hogy meghívja a CGI programot, átadja neki az elküldött adatokat, ill. a CGI program válaszát visszaküldi a kliensnek. Minden másért a CGI program a felelős: először is leellenőrzi a kapott adatokat, s ha azok formátuma nem megfelelő, akkor előállít egy HTML lapot, melyben leírja, hogy mi a gond. A CGI program a válaszát a standard output-ra írja ki, s ezt a webszerver elküldi a felhasználónak, akinek általában ilyenkor vissza kell lépnie egyet a böngészőjében, s újra kell próbálkoznia. Ha az adatok megfelelőek, akkor a CGI program valamilyen módon feldolgozza az adatokat (pl. hozzáadja egy adatbázishoz). Ezután elő kell állítania még egy megfelelő HTML lapot a felhasználó tájékoztatására.
Ideális lenne ezt a problémát teljesen Java-ra alapozva megoldani – a kliens oldalon lenne egy applet, amely leellenőrizné az adatokat, majd ha azok megfelelőek, akkor elküldené őket; a szerver oldalon pedig egy szervlet fogadná és dolgozná fel az adatokat. Sajnos azonban az appletek alkalmazása a web-en számos problémával jár: nem számíthatunk arra, hogy a kliens azzal a Java verzióval rendelkezik, amit mi szeretnénk; sőt: lehet, hogy a kliens webböngészője nem is támogatja a Java-t egyáltalán (pl. lynx)! A másik gond az, hogy egy applet futtatásához a kliensnek le kell töltenie az applet byte-kódját, ami jelentős hálózati forgalommal járhat. Ezért aztán a legbiztosabb megoldás az, ha az adatok feldolgozását a szerveroldalon végezzük el, s a kliensnek csupán egy HTML lapot küldünk. Ebben az esetben

egyetlen klienst sem kell azzal az indokkal visszautasítanunk, hogy nem rendelkezik a megfelelő szoftverrel.

A szervletek egy nagyszerű megoldást nyújtanak a szerveroldali programozáshoz. Nem csak egy olyan környezetet biztosítanak, amely helyettesíti a CGI programozást (s számos CGI problémától megszabadítanak), hanem a Java nyelv jellemzőiből adódóan a teljes kód platformfüggetlen, ill. az összes Java API (kivéve persze azokat, amelyek a grafikus felületeket produkálják, pl. AWT, Swing) használható.

Egy szervlet tehát valójában egy speciális Java program, amely egy webszerverrel szorosan együttműködve a szerveroldalon lehetővé teszi HTML oldalak dinamikus létrehozását és paraméterezését különböző átviteli (általában HTTP) protokollokon keresztül.

A HTML oldalak dinamikus generálására (korábban) használt CGI megoldásokkal szemben a szervletek a következő előnyökkel rendelkeznek:

- a Java nyelv miatt platformfüggetlen
- a webszerveren belül állandóan futó virtuális gép sokkal gyorsabb kiszolgálást tesz lehetővé, mivel elmarad a minden egyes kérés kiszolgálásához szükséges külső programindítás
- a webszerveren belül állandóan futó virtuális gép miatt leegyszerűsödik a kéréskiszolgálások szinkronizációja
- egy szervlet átirányíthatja a kérést egy másik szervlethez, ezáltal dinamikusan lehet alkalmazkodni a webszerver terheltségéhez
- egy szervlet könnyen megőrizhet információkat az azonos helyről jövő egymás utáni kérések kiszolgálása között

1.3.1. Szervlet fejlesztőkörnyezet

A webszerver és a szervlet közötti kommunikáció a szervlet API-n keresztül történik. Szervleteket ennélfogva csakis olyan webszerverekkel lehet használni, melyek támogatják ezen API-t, és természetesen képesek Java programok futtatására. A szervlet API nem része a standard JDK-nak, de ingyenesen letölthető a szükséges fejlesztőkörnyezet (JSDK – Java Servlet Development Kit). A JSDK tartalmaz még egy egyszerű Java alapú webszervert, melynek segítségével akkor is kipróbálhatjuk szervleteinket, ha nincs a gépünkön más webszerver telepítve. Hátránya viszont az, hogy nem érzékeli a lefordított `.class` file-ok változását, azaz ha módosítottuk a forráskódot, s újrafordítottuk a programot, akkor a JSDK-t

is újra kell indítani a változások érvénybeléptetéséhez. A JSDK telepítése egyszerű. A telepítési könyvtárból (jelöljük ezt **/**-rel) nyíló **/bin** könyvtárban található az indítószkript. Használata:

./servletrunner	ekkor az alapértelmezett portot, a 8080-ast fogja használni
./servletrunner -p <port>	ekkor explicit módon mondjuk meg, hogy melyik portot használja

A **/examples** könyvtárba helyezhetjük el a szervleteket. Egy szervlet meghívása:

`http://127.0.0.1:`*`port`*`/servlet/NameOfTheServlet`

Több dologra is figyelni kell:

1. meg kell adni a portszámot
2. könyvtárnévként servlet-et kell megadni (egyes számban)
3. csak a szervlet nevét kell feltüntetni (.class nélkül)

Az egyik legelterjedtebb szervletfuttató környezet az ApacheJServ. Beintegrálható az Apache szerverbe, így azzal együtt tud dolgozni. Beállítása nem egyszerű, sok fejfájást tud okozni, viszont miután beindult, nagyon megbízhatóan működik (az ApacheJServ felinstallálását az 1. sz. Melléklet tartalmazza). Érzékeli a .class file-ok változását, de csak szervleteket tud futtatni. Nem is tervezik a következő verziók JSP-támogatással való kibővítését, de szerencsére erre ott a Tomcat.

Egy másik fejlesztőkörnyezet a Jakarta-Tomcat. Ez egy szervlet-konténer ill. JavaServer Pages implementáció. Érdemes inkább ezt használni mint a JSDK-t, hiszen többet tud (JSP-t is), ill. beállítható, hogy figyelje a .class file-ok változását, így nem kell mindig manuálisan újraindítani, hogy életbe lépjenek a módosítások. Önmagában is használható, de számos népszerű webszerverrel is együtt tud működni, pl.:

- Apache (min. 1.3-as verzió) (2. sz. Melléklet)
- Microsoft IIS (min. 4.0-ás verzió)
- Microsoft Personal Web Server (min. 4.0-ás verzió)
- Netscape Enterprise Server (min. 3.0-ás verzió)

A Tomcat további előnye az, hogy ingyenes. Míg az ApacheJServ a Servlet API 2.0-t támogatja, addig a Tomcat 3.2 (mely a JServ teljes újraírása) a Sevlet API 2.2 ill. a JSP 1.1 referenciaimplementációja.

(A Tomcat 4.0 már egészen más fejlesztés: különböző architektúrára, az ún. Catalina architektúrára épül, s a Servlet API 2.3 ill. a JSP 1.2 referenciaimplementációja).

Használata: a telepítési könyvtárból (jelöljük **/**-rel) nyíló **/conf** könyvtárban találhatóak a konfigurációs file-ok. Ezek közül a server.xml ill. a web.xml a legfontosabb. A server.xml file-ban két portszámot is meg kell adni, melyeket a Tomcat használni fog.

Nézzük meg az előző példa szervletes változatát a JSDK-n keresztül! Ehhez szükségünk lesz egy HTML űrlapra, ahol található egy szövegmező ill. egy „submit" gomb, továbbá kell egy szervlet, amely a szerveroldalon fogadja és feldolgozza a számára elküldött adatokat.
A HTML lap:

```
<html>
<body>
<form action="http://127.0.0.1:8080/servlet/Szervlet_pelda" method="POST">
Kód: <input type="text" name="code"><br>
<input type="submit">
</form>
</body>
</html>
```

Egy HTML lapon form segítségével fogjuk össze a logikailag összetartozó elemeket. Minden egyes form-hoz megadható, hogy mi történjen akkor, ha a felhasználó megnyomja az „elküld" gombot. Ezt az `action` paraméter írja le (itt lehet megadni egy CGI, szervlet, JSP, PHP, stb. címét). Továbbá megadható még az elküldés módja: GET, POST. (Erről az 1.3.2. fejezetben lesz szó).
A szervlet kódját a JSDK *examples* könyvtárában helyezzük el. Maga a szervlet:

```
import java.io.*;
import java.sql.*;
import javax.servlet.http.*;

public class Szervlet_pelda extends HttpServlet
{
   public void service(HttpServletRequest keres,
                       HttpServletResponse valasz)
                       throws IOException
   {
      valasz.setContentType("text/html");
      PrintWriter eredmeny = valasz.getWriter();
      eredmeny.println("<html><body>");
      boolean talalt = false;

      try
      {
         Class.forName("org.postgresql.Driver");
         Connection conn =
         DriverManager.getConnection("jdbc:postgresql://127.0.0.1/"+
         "areas","jabba","jabba");
         Statement  st   = conn.createStatement();
```

```
         ResultSet rs = st.executeQuery("select name from codes "+
          "where code='"+keres.getParameterValues("code")[0]+"'");
         if (rs!=null)
         {
            while (rs.next())
            {
               eredmeny.println(rs.getString("name")+"<br>");
               talalt = true;
            }
         }
         if (!talalt)
         eredmeny.println("Sajnos erről nem találtam semmit!<br>");
         rs.close();
         st.close();
         conn.close();
      }
      catch (java.sql.SQLException sqle) {
         eredmeny.println("<b>SQL hiba:</b> "+sqle.toString());
      }
      catch (ClassNotFoundException cnfe) {
         eredmeny.println(cnfe.toString());
      }

      eredmeny.println("</body></html>");
   }
}
```

A továbbiakban csak olyan szervletekkel foglalkozunk, melyek a WWW-n legjobban elterjedt HTTP átviteli protokollt használják a klienssel való kommunikáció megvalósítására. Az ilyen típusú szervletek mindig a `javax.servlet.http.HttpServlet` leszármazottai. A `javax.servlet.http` alcsomag a HTTP átviteli protokollhoz kapcsolódó speciális szolgáltatások elérését biztosító típusokat tartalmazza.

A szervlet inicializációja egyszer történik meg az **init()** metódusa meghívásával, amikor a szervlet-konténer felállása után először betöltődik. Amikor egy kliens hivatkozik a szervletre, a szervlet-konténer „elfogja” a kérést, beállítja a **HttpServletRequest**, **HttpServletResponse** objektumokat, s meghívja a szervlet **service()** metódusát, melynek fő feladata: feldolgozza a kliens által küldött adatokat, s ezek alapján elkészít egy HTTP választ.

Miután a válasz típusát beállítottuk (**setContentType()**, ezt még azelőtt kell megtenni, hogy a **Writer** vagy **OutputStream** segítségével bármit is elküldenénk a kliensnek), a **getWriter()** metódussal előállítunk egy **PrintWriter** objektumot, mellyel karakter-alapú választ tudunk küldeni (ehhez hasonló célt szolgál a **getOutputStream()** is, mely egy **OutputStream** objektumot állít elő, s ezzel bináris adatot lehet elküldeni, pl. egy képet).

1.3.2. Paraméterátadás – GET vagy POST

A kliens oldalon a form-okhoz megadható, hogy az „elküld” gomb lenyomására a form adatai melyik programnak (vagy szkriptnek) milyen módon legyenek elküldve. Az elküldés módjai:

GET: a HTTP GET kérések során az elkódolt paraméterértékek egyszerűen hozzáfűződnek az URL végére a következő formában:

$$URL?param_1=érték_1\&param_2=érték_2\ldots$$

A paramétereket magától az URL-től egy „?” választja el. Ezután következnek a paraméternevek és értékeik felsorolása, s ezen párokat egy „&” jel választja el egymástól.

POST: a HTTP POST kérések egy összetettebb protokoll segítségével kerülnek elküldésre. A 255 karakternél hosszabb URL-ek gondot okozhatnak (régebbi webszerverek nem tudják kezelni az ennél hosszabb URL-eket), így ha nagymennyiségű adatot kívánunk a paraméterek segítségével elküldeni, akkor a POST módszert alkalmazzuk!

További előnye a POST-nak, hogy az átadott paraméternév–érték párok nem fognak megjelenni a szerver naplóállományában (logfile) a kért URL után, hiszen nem képezik az URL részét!

A **HttpServlet** osztályban külön-külön metódusok vannak az eltérő paraméterátadások kezelésére:

- **doGet()** – GET típusú HTTP kérések kezelésekor hívódik meg. A paraméterek az URL végén lévő paramétersztring formájában kerülnek átadásra
- **doPost()** – POST típusú HTTP kérések kezelésekor hívódik meg. A paraméterek ekkor nem látható módon a HTTP fejlécben kerülnek átadásra
- **service()** – mindkét előző típusú kliens kérést kiszolgálja

A **HttpServlet** osztály kibővítésekor tehát csak azon **do***típus* metódusokat kell felüldefiniálni, amilyen HTTP típusú kéréseket szeretnénk kezelni. Ez általában a **doGet** és **doPost** metódusok implementálását jelenti, méghozzá rendszerint úgy, hogy csak az egyiket valósítjuk meg ténylegesen, s a másik metódus erre irányítja át a kérés feldolgozását. Egy másik megoldás pedig az, hogy ezek helyett csupán a **service()** metódust implemetáljuk, hiszen ez mindkét esetet képes lekezelni. A feldolgozás további menete: a kapott paraméterek

kiolvasása, a válasz HTTP fejlécének beállítása, a válasz előállítása, annak kiküldése a kliens adatfolyamra, majd onnan annak elküldése a kliens részére.

Nézzünk egy példát egy olyan szervletre, amellyel a paraméterátadást lehet nagyszerűen demonstrálni:

```
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class Params extends HttpServlet
{
   public void doPost(HttpServletRequest request,
                      HttpServletResponse response)
                      throws ServletException, IOException
   {
      doGet(request,response);
   }

   public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
                      throws ServletException, IOException
   {
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();

            Enumeration e = request.getParameterNames();
            while (e.hasMoreElements())
            {
                  String nev = (String)e.nextElement();
                  String[] ertek = request.getParameterValues(nev);
                  out.print(" " + nev + " = ");
                  for (int i=0; i<ertek.length; ++i)
                        out.print(ertek[i]+" ");
                  out.println("<br>");
            }
   }

}
```

Ebben a szervletben csak a **doGet()** metódust implementáltuk. Ha a paraméterek POST-tal lettek elküldve, akkor a **doPost()** metódus egyszerűen „áthárítja” a feladat megoldását a **doGet()** metódusnak. A **getParameterNames()** egy enumerátort ad vissza, mely felsorolja a klienstől kapott paraméterek nevét. A **getParameterValues()** az adott nevű paraméter értékeit adja vissza.

A szervletek egyik hátránya az, hogy a kliensnek visszaküldendő választ **String** objektumokból kell felépíteni, s ez elég nehézkes (áttekinthetetlen) a sortörések, escape-szekvenciák (″ helyett pl. \″ -t kell írni) és a rengeteg sztringkonkatenáló ″+″ jel miatt. Egy nagyobb HTML oldal elkészítése, megformázása Java sztringekkel nem könnyű feladat, a

formázáson pedig utólag módosítani még nehezebb. Egy HTML lap (vagy kódrészlet) Java sztringgé alakításához készítettem egy Perl szkriptet, mely a 3. sz. Mellékletben található. A probléma kiküszöbölésére egy másik megoldás a JSP használata, mely a szervletek „kiterjesztésének” is tekinthető. Erről az 1.4. fejezetben olvashatunk.

1.3.3. Többszálúság

Elég valószínű, hogy a webszerverhez egyidőben több olyan kérés is befut, amelyek ugyanazt a szervletet akarják elérni. A **service()** metódust emiatt szálbiztos módon kell megírni. A közös erőforrásokhoz (file-ok, adatbázisok) történő bármilyen hozzáférést a **synchronized** kulcsszóval kell védeni.

A következő példa egy **synchronized** blokkba teszi a szál **sleep()** metódusát. Ez minden egyéb szálat blokkol mindaddig, míg le nem telik a megadott idő (5 mp.). Ennek a kipróbálásához indítsunk el több böngészőt, s mindegyiken hozzuk be ezt a szervletet minél gyorsabban. Látni fogjuk, hogy mindegyiknek addig kell várni, amíg rá nem kerül a sor.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class ThreadServlet extends HttpServlet
{
   int i=0;
   public void service(HttpServletRequest request,
                       HttpServletResponse response)
                       throws IOException
   {
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      out.println("<html><body>");
      synchronized(this)
      {
         try {
           ++i;
           Thread.currentThread().sleep(5000);
         } catch (InterruptedException e) {
           System.err.println("Megszakadt.");
         }
      }
      out.println("Vege " + i + "</body></html>");
      out.close();
   }
}
```

Az egész szervletet is szinkronizálhatjuk – ekkor a **synchronized** kulcsszót a **service()** metódus elé kell tenni. Vagy ugyanerre van egy elegánsabb megoldás: implementáljuk a **SingleThreadModel** interfészt. Ezen interfész nem tartalmaz egyetlen metódust sem, de ha

implementáljuk, akkor a szervlet futtatókörnyezet biztosítja, hogy a kiszolgáló **service** metódust egyidőben legfeljebb csak egy programszál fogja meghívni. A bejövő kliens kérések sorbaállításának ez a legegyszerűbb módja.

1.3.4. Klienskapcsolat követése

A HTTP egy „kapcsolat nélküli" protokoll abban az értelemben, hogy a szervert ért két egymás utáni találatból nem lehet megmondani, hogy ugyanaz a személy kérte-e le egymás után ugyanazt az oldalt, vagy két teljesen más személy. Az elektronikus kereskedelem nem működne, ha nem lehetne egy klienst az általa kiválasztott árucikkekkel nyomon követni. Számos módja létezik a klienskapcsolat követésének, de a leggyakoribb módszer a „cookie"-k használata, mely az Internet szabvány része (lásd RFC 2109).
Egy cookie csupán egy kis információtöredék, amit a webszerver küld el a kliens böngészőjének. A böngésző ezt letárolja a merevlemezen, s ha a kliens legközelebb lekéri azt az URL-t, amivel a cookie össze van rendelve, akkor a cookie-t is elküldi észrevétlenül a kéréssel együtt, így adva át a szükséges információt az adott szervernek. A kliensek azonban kikapcsolhatják a böngésző ezen tulajdonságát. Ebben az esetben valamilyen más módon kell megvalósítani a klienskapcsolat követését, pl. rejtett form-mezőkkel.

1.3.4.1. A Cookie osztály

A 2.0-s és afeletti szervlet API-k rendelkeznek a **Cookie** osztállyal, melynek segítségével cookie-kat lehet kezelni. Egy cookie használata csupán annyiból áll, hogy hozzáadjuk a **HttpServletResponse** objektumhoz. A konstruktor egy nevet és egy értéket vár. A cookie-kat még a visszaküldendő tartalom előtt el kell küldeni:

```
Cookie proba = new Cookie(″nev″, ″ertek″);
response.addCookie(proba);
```

A cookie-k visszaolvasása a **HttpServletRequest** objektum **getCookies()** metódusával tehető meg, mely egy cookie objektumokat tartalmazó tömböt ad vissza.

```
Cookie[] cookies = request.getCookies();
```

Egy cookie értékét a **getValue()** metódussal kérhetjük le, mely egy sztringet ad vissza. A fenti példában a **getValue(″nev″)** egy **String**-et eredményezne ″ertek″ tartalommal.

1.3.4.2. A Session osztály

A cookie-k segítségével tetszőleges szöveges információt tárolhatunk a kliensoldalon. Azonban az teljesen a klienstől függ, hogy a hozzá küldött információkat hogyan kezeli. Előfordulhat, hogy a kliens egyáltalán nem fogadja a cookie-kat (a böngészőprogramokban ez kikapcsolható), így ilyenkor valamilyen más módszert kell alkalmazni az információ elmentésére. Egy másik megoldás az, ha az információt nem a kliens, hanem maga a szerver tárolja. Ekkor minden klienskapcsolat, amely ugyanazt a klienskapcsolat környezetet használja, ugyanazon adatokat is fogja látni. Ezen klienskapcsolat környezetet tehát a szerver tartja nyilván. Egy ilyen „session” objektum feladata: információgyűjtés a kliensről (pl. milyen cikkeket tett a bevásárlókosárba), hitelesítés (pl. tegyük fel, hogy egy webhelyre csak úgy lehet belépni, hogy megadjuk a nevünket és kódunkat. A többi weboldalt csak ezután érhetjük el. Ha valaki közvetlenül szeretne egy ilyen „belső” oldalra ugrani, akkor át lesz irányítva a bejelentkezési lapra).

Egy klienskapcsolat környezetet a **HttpSession** osztály reprezentál. Nézzünk egy példát:

```
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class SessionTest extends HttpServlet {
   public void service(HttpServletRequest  req,
                       HttpServletResponse res)
                       throws IOException, ServletException {
      HttpSession session = req.getSession(true);
      res.setContentType("text/html");
      PrintWriter out = res.getWriter();
      out.println("<html><head>");
      out.println("<title>Sessiontest</title>");
      out.println("</head><body><h1>SessionTest</h1>");

      Integer ival = (Integer)session.getValue("session.cnt");
      if (ival==null) ival = new Integer(1);
      else            ival = new Integer(ival.intValue()+1);
      session.putValue("session.cnt", ival);
      out.println("Te <b>"+ival+".</b> alkalommal jarsz itt.");
      out.println("<h2>a session adatai elmentve</h2>");

      out.println("<h3>Session statisztika:</h3>");
      out.println("Session ID: "+session.getId()+"<br>");
      out.println("Uj a session?: "+session.isNew()+"<br>");
      out.println("Letrehozas datuma: "+
         new Date(session.getCreationTime()) + "<br>");
      out.println("Utolso hozzaferes datuma: "+
         new Date(session.getLastAccessedTime()) + "<br>");
      out.println("</body>"); out.close();
   }
}
```

Az aktuális klienskapcsolat környezetet a **HttpServletRequest getSession** metódusával lehet lekérdezni. Ha ez még nem létezik, akkor igaz paraméterérték esetén egy új környezetet hoz létre, és az lesz a visszaadott érték, hamis paraméterérték esetén pedig *null* lesz a visszaadott érték. Miután megszereztük a **HttpSession** referenciát, objektumokat tárolhatunk benne a **setAttribute()** metódussal, elérhetjük az abba már korábban berakott objektumokat a **getAttribute()** metódussal, ill. törölhetjük azokat a **removeAttribute()** meghívásával. A kapcsolat-környezetben tárolt összes objektum nevét a **getAttributeNames()** metódussal lehet lekérdezni, mely egy *String* tömböt ad vissza.

A fenti példában egy „session.cnt" nevű *Integer* objektumot akarunk kivenni. Ha még nincs ilyen (mert még most jött létre a session), akkor létrehozunk egy ilyen objektumot, s betesszük ebbe a klienskapcsolatba. Ha már létezett ilyen, akkor az 1-gyel megnövelt értékét tesszük vissza, s így követjük nyomon, hogy az adott kliens hányszor kérte le az oldalt.

Egy session számos további információt is tárol:

- **getId()** – visszaadja a klienskapcsolat-környezet azonosítóját
- **isNew() –** igazat ad vissza, ha a klienskapcsolati környezet újonnan lett létrehozva
- **getCreationTime()** – visszaadja az 1970.01.01 óta eltelt ezredmásodpercek számát a környezet létrehozásának időpontjáig. (Ha ezt átadjuk a *java.util.Date* osztály konstruktorának, akkor ezt olvasható formába átalakítja).
- **getLastAccessedTime()** – visszaadja az 1970.01.01 óta eltelt ezredmásodpercek számát a környezet legutolsó használatának időpontjáig

Ha már nincs többé szükség egy klienskapcsolat-környezetre, akkor azt mi magunk is megszüntethetjük annak **invalidate()** metódusával. A kapcsolat automatikusan is megszűnik, ha annak érvényességi időtartama alatt nem történik hivatkozás rá.

Lényeges, hogy a klienskapcsolat-környezetet még minden más adat küldése előtt nyissuk meg!

1.4. Java Server Pages

A JSP segítségével szerveroldali szkriptelést tudunk megvalósítani Java módra. Hasonló a Microsoft Active Server Pages (ASP) technológiájához, azonban könnyebben kiterjeszthető, s nincs hozzákötve egyetlen céghez vagy webszerverhez sem. Habár a JSP specifikációt a Sun Microsystems állapítja meg, bárki elkészítheti ennek implementációját a saját rendszerén is.
A JSP alapértelmezésben a Java nyelvet használja, de a specifikáció lehetővé teszi más nyelvek alkalmazását is, mint ahogy az ASP is használhat más nyelveket (JavaScript, VBScript).
A JSP egy egyszerű, könnyen használható nyelv- és eszközrendszert biztosít robusztus web-alkalmazások készítésére. Számos szerveroldali *tag* áll rendelkezésre, melyek lehetővé teszik a dinamikus tartalom kialakítását akár egyetlen sor Java-kód megírása nélkül is. HTML fejlesztők is használhatják a JSP-t anélkül, hogy meg kellene tanulniuk a Java nyelvet. A JSP a szkriptelési lehetőségek mellett arra is lehetőséget ad, hogy JavaBean-eket használjunk, amely nagyban előmozdítja a kód újrafelhasználását.

A JSP egy standard Java kiterjesztés, mely a szervletekre épül. A JSP-t azzal a szándékkal hozták létre, hogy leegyszerűsítsék a dinamikus weboldalak létrehozását, karbantartását.
Mire lesz szükségünk?

- JDK legalább 1.1-es verziója
- Tomcat. Kiválóan implementálja a Java Servlet 2.2-t, ill. a JSP 1.1-et

A JavaServer Pages (JSP) lehetővé teszi, hogy az oldalak dinamikus részét különválasszuk a statikus HTML-től. A szokásos HTML-t a megszokott módon írjuk, bármilyen weboldal-szerkesztőt is használunk. A dinamikus részeket speciális tag-ek közé zárjuk, melyek rendszerint a „`<%`” jelekkel kezdődnek, s „`%>`” jelekkel záródnak. Például nézzük a következő kódrészletet:

```
Thanks for ordering
<I><%= request.getParameter("title") %></I>
```

Ez a `http://host/OrderConfirmation.jsp?title=Core+Web+Programming` URL-en történő meghívásra valami hasonlót eredményez: „Thanks for ordering *Core Web Programming”*.

A file-ok rendszerint .jsp kiterjesztésűek, és általában bárhol elhelyezhetőek, ahol egy normál HTML lap. Annak ellenére, hogy amit írunk inkább hasonlít egy hagyományos HTML-re mint szervletre, a színfalak mögött a JSP oldal egy közönséges szervletté konvertálódik, s a statikus HTML egyszerűen kinyomtatásra kerül az output stream-re, amely a szervlet `service` metódusával van összekapcsolva. Ez rendszerint akkor történik meg, amikor először hivatkozunk az oldalra, s a fejlesztők is lekérhetik egyszerűen az oldalakat installálás után, ha nem akarják, hogy az első igazi felhasználónak sokat kelljen várnia, míg a JSP oldalt szervletté alakítja a rendszer, majd lefordítja és betölti. Számos webszerver lehetővé teszi álnevek (alias-ok) használatát, így egy URL, amely látszólag egy HTML hivatkozás, valójában egy szervletet vagy JSP-t is takarhat.

A szokásos HTML mellett 3 fő JSP konstrukció ágyazható be a lapba: szkriptelemek, direktívák és akciók. A szkriptelemek olyan Java kód megadását teszik lehetővé, amely az előálló szervlet részét fogja képezni. A direktívákkal a szervlet teljes struktúrája irányítható, míg az akciókkal létező használandó komponensek specifikálhatók, ill. a JSP motor viselkedését irányítják. A szkriptelemek egyszerűsítése érdekében számos előredefiniált változó létezik, mint a `request` is a fenti példában.

1.4.1. Szintaxis összefoglaló

JSP elem	Szintaxis	Jelentése	Megjegyzés

JSP kifejezés	`<%= kifejezés %>`	a kifejezés kiértékelődik, s átkerül az outputra	XML megfelelője:
			`<jsp:expression>`
			`kifejezés`
			`</jsp:expression>`
			Előredefiniált változók: `request`, `response`, `out`, `session`, `application`, `config` és `pageContext` (szkriptletekből szintén elérhetők)

JSP szkriptlet	`<% kód %>`	a kód bekerül a `service` metódusba	XML megfelelője:
			`<jsp:scriptlet>`
			`kód`
			`</jsp:scriptlet>`

JSP deklaráció	`<%! kód %>`	a szervletosztály törzsébe kerül a kód, a `service` metóduson kívülre	XML megfelelője:
			`<jsp:declaration>`
			`kód`
			`</jsp:declaration>`

JSP `page` direktíva	`<%@ page att=”érték” %>`	a szervletmotornak szóló direktívák, általános beállítások	XML megfelelője: `<jsp:directive.page att="érték" \>` Szabályos attribútumok (alapértelmezett érték vastagon szedve): • `import="`*`package.class`*`"` • `contentType="`*`MIME-Type`*`"` • `isThreadSafe="`**`true`**`|false"` • `session="`**`true`**`|false"`

			• buffer="*size*kb|none" • autoflush="**true**|false" • extends="*package.class*" • info="*message*" • errorPage="*url*" • isErrorPage="true|**false**" • language="java"
JSP include direktíva	<%@ include file=”url” %>	egy file a lokális rendszeren beillesztésre kerül, mikor a rendszer lefordítja a JSP-t szervletté	XML megfelelője: <jsp:directive.include file="url"\> Az URL-nek relatívnak kell lennie. Használjuk a jsp:include akciót, ha azt akarjuk, hogy a file ne fordítási időben, hanem a kérés pillanatában kerüljön beszúrásra.
JSP megjegyzés	<%-- megjegyzés --%>	Megjegyzés, a JSP lap szervletté fordításakor a rendszer figyelmen kívül hagyja	Ha az eredményképpen előálló HTML-ben is látni szeretnénk a megjegyzést, akkor használjuk a <-- megjegyzés --> hagyományos HTML szintaxist
jsp:include akció	<jsp:include page="relatív URL" flush="true"/>	Beilleszt egy file-t az oldal lekérésének pillanatában	Ha a file-t fordítási időben akarjuk beilleszteni, akkor használjuk ehelyett a page direktívát az include attribútummal. Figyelmeztetés: néhány szerveren a beillesztett file-nak HTML vagy JSP file-nak kell lennie, ezt a szerver dönti el (ez általában a file kiterjesztésén alapszik).
jsp:useBean akció	<jsp:useBean att=érték*/> vagy <jsp:useBean att=érték*> ... </jsp:useBean>	találjon meg vagy hozzon létre egy Java Bean-t	Lehetséges attribútumok: • id="*név*" • scope="**page**|request|session|application" • class="*package.class*" • type="*package.class*" • beanName="package.class"
jsp:setProperty akció	<jsp:setProperty att=érték*/>	bean tulajdonságok beállítása explicit módon vagy egy request paraméteren keresztül	Lehetséges attribútumok: name="beanNeve" property="propertyNeve|*" param="paraméterNeve" value="érték”
jsp:getProperty akció	<jsp:getProperty name="propertyNeve" value="érték"/>	bean tulajdonságainak lekérdezése és output-ra küldése	
jsp:forward akció	<jsp:forward page="relatív URL"/>	a kérést egy másik lapra továbbítja	
jsp:plugin akció	<jsp:plugin attribute="érték"*> ... </jsp:plugin>	a böngésző típusának megfelelően OBJECT vagy EMBED tag-et generál, arra kérve a böngészőt, hogy az applet-et a Java Plugin-on keresztül futtassa	

1.4.2. Sablonszöveg: statikus HTML

A legtöbb esetben egy JSP oldal nagy része statikus HTML-ből, ún. „sablonszöveg"-ből áll. Egyetlen esetet kivéve ez a HTML úgy néz ki mint a normál HTML, ugyanazon szintaktikai szabályok vonatkoznak rá, s az oldal kezelésére létrehozott szervlet egyszerűen csak „átküldi" a kliensnek. A HTML nem csupán *úgy néz ki,* mint a hagyományos HTML, hanem úgy is lehet *létrehozni,* bármilyen eszközt is használjunk weboldalak szerkesztésére.

Az egyetlen apró kivétel – abban a szabályban, hogy a „sablonszöveget egy az egyben átküldi" – az, hogy ha az output-on „`<%`" jeleket szeretnénk, akkor a sablonszövegben „`<\%`" jeleket kell írni.

1.4.3. JSP szkriptelemek

A JSP szkriptelemek lehetővé teszik Java kód beillesztését a JSP lapból generált szervletbe. Ennek három fajtája van:

1. `<%= kifejezés %>` alakú kifejezések, melyek kiértékelődnek, s az eredmény az output-ra kerül
2. `<% kód %>` alakú szkriptletek, melyek bekerülnek a szervlet `service` metódusába, és
3. `<%! kód %>` alakú deklarációk, melyek a szervletosztály törzsébe kerülnek, minden létező metóduson kívülre.

A továbbiakban részletesen is megnézzük mindegyiket. Addig is álljon itt egy példa:

```
<%-- JSP megjegyzes, nem kerul be a generalt HTML-be --%>
<%@ page import="java.util.*" %>
<%!
   long loadTime = System.currentTimeMillis();
   Date loadDate = new Date();
   int hitCount = 0;
%>
<html><body>
<h1>Az oldal mikor lett betoltve: <%= loadDate %></h1>
<h1>Hello Vilag! A mai datum: <%= new Date() %></h1>
<h2>Itt van egy objektum: <%= new Object() %></h2>
<h2>Az oldal betoltese ota
<%= (System.currentTimeMillis()-loadTime)/1000 %>
masodperc telt el.</h2>
<h3>Letoltesek szama: <%= ++hitCount %></h3>
<%
   System.out.println("Viszlat!");
   out.println("szia");
%>
</body></html>
```

Ha többször is behívjuk ezt az oldalt egymás után, azt láthatjuk, hogy a **loadTime**, **loadDate** és **hitCount** változók megőrizték az értéküket, tehát tényleg az osztályhoz tartozó mezők (a *service* metóduson kívül helyezkednek el), s nem pedig lokális változók.

1.4.3.1. JSP kifejezések

A JSP kifejezés Java értékek közvetlen megjelenítésére szolgál az output-on. Alakja:

```
<%= Java kifejezés %>
```

A Java kifejezés kiértékelődik, a rendszer sztringgé alakítja s beilleszti a lapba. Ez a kiértékelés futásidőben történik meg (amikor hivatkoznak az oldalra), és így a kéréssel kapcsolatos információkhoz teljes a hozzáférés. Például a következő JSP kifejezés megmutatja, hogy mikor kérték le az oldalt:

```
Az aktuális idő: <%= new java.util.Date() %>
```

Ezen kifejezések egyszerűsítése érdekében számos előredefiniált változót lehet használni. Ezekről az implicit objektumokról később bővebben is szó lesz, de a kifejezések céljából a legfontosabbak:

- `request` (ez a `HttpServletRequest`)
- `response` (ez a `HttpServletResponse`)
- `session` (ez a kéréshez tartozó (ha van ilyen) `HttpSession`); és
- `out` (ez a `PrintWriter` (output küldése a kliensnek))

A `request` objektum használatára egy példa:

```
Host-név: <%= request.getRemoteHost() %>
```

Végül megjegyezzük, hogy az XML „hívei” a JSP kifejezésekre használhatnak egy alternatív szintaktikát is:

```
<jsp:expression>
Java kifejezés
</jsp:expression>
```

Ne feledjük, hogy az XML elemek a HTML-lel ellentétben érzékenyek a kis- és nagybetűkre, ezért figyeljünk a kisbetűk használatára!

1.4.3.2. JSP szkriptletek

Ha egy egyszerű kifejezés beillesztésénél komplexebb dolgot akarunk megvalósítani, akkor a JSP szkriptletekkel az oldalt generáló szervletmetódusba tetszőleges kódot illeszthetünk be. A szkriptletek a következőképpen néznek ki:

```
<% Java kód %>
```

A szkriptletek ugyanazon automatikusan definiált változókhoz férnek hozzá, mint a kifejezések. Tehát például ha valamit meg akarunk jelentetni az előálló lapon, akkor az `out` változót kell használni:

```
<%
String queryData = request.getQueryString();
out.println("Csatolt GET adat: " + queryData);
%>
```

Fontos, hogy a szkriptletben lévő kód *pontosan* úgy kerül beszúrásra, ahogy azt leírtuk, s minden statikus HTML-t (sablonszöveg) a szkriptlet előtt vagy után a rendszer `print` utasításokká alakít át. Ez azt jelenti, hogy a szkriptleteknek nem kell teljes Java utasításokat tartalmaznia, s a nyitott blokkok hatással lehetnek a szkriptleteken kívül lévő statikus HTML-re. Például a következő JSP részlet, mely sablonszöveget és szkriptleteket vegyesen tartalmaz

```
<% if (Math.random() < 0.5) { %>
Legyen egy <B>jó</B> napod!
<% } else { %>
Legyen egy <B>rossz</B> napod!
<% } %>
```

valahogy így lesz átalakítva:

```
if (Math.random() < 0.5) {
  out.println("Legyen egy <B>jó</B> napod!");
} else {
  out.println("Legyen egy <B>rossz</B> napod!");
}
```

Ha a „`%>`” karaktereket akarjuk használni egy szkritletben, akkor írjunk „`%\>`” jeleket helyette.

Végül a `<% kód %>` XML megfelelője:

```
<jsp:scriptlet>
kód
</jsp:scriptlet>
```

1.4.3.3. JSP deklarációk

Egy JSP deklaráció lehetővé teszi metódusok, mezők definiálását, melyek a szervletosztály törzsébe (a `service` metóduson kívülre) kerülnek be. Alakja a következő:

```
<%! Java kód %>
```

Mivel a deklarációk nem generálnak semmilyen output-ot, így általában valamilyen JSP kifejezéssel vagy szkriptlettel együtt használjuk őket. Például az alábbi JSP részlet azt számlálja, hogy az oldalt hányszor kérték le a szerver indítása (vagy a szervletosztály változtatása és újraindítása) óta:

```
<%! private int accessCount = 0; %>
Az oldalhoz való hozzáférések száma a szerver újraindítása óta:
<%= ++accessCount %>
```

Csakúgy, mint a szkriptleteknél: ha a „`%>`” karaktereket akarjuk használni, akkor „`%\>`” karaktereket írjunk helyettük. Végül a `<%! kód %>` XML megfelelője:

```
<jsp:declaration>
kód
</jsp:declaration>
```

1.4.4. JSP direktívák

A direktívák semmit nem küldenek az *output stream*-re. Egy JSP direktíva a szervletosztály teljes struktúrájára hatással van. Alakja általában a következő:

```
<%@ directive attribútum="érték" %>
```

Több attribútumbeállítás azonban össze is vonható egyetlen direktívába:

```
<%@ directive   attribútum1="érték1"
                attribútum2="érték2"
                ...
                attribútumN="értékN" %>
```

A direktíváknak 2 fő típusa van: `page` (olyan dolgokat tesz lehetővé, mint pl. osztályok importálása, szervlet szuperosztályának megadása, stb.), ill. `include` (a JSP file szervletté fordításakor beszúrható egy file a szervletosztályba).

1.4.4.1. `page` direktívák

A `page`-direktívák típusai:

- `import="package.class"` vagy
  `import="package.class1,…,package.classN"`
  Csomagok importálása. Pl.:
  `<%@ page import="java.util.*" %>`
- `contentType="MIME-Type"` vagy
  `contentType="MIME-Type; charset=Character-Set"`
  Az output MIME típusát írja le. Alapértelmezett: "text/html".
  `<%@ page contentType="text/plain" %>` ekvivalens a következővel:
  `<% response.setContentType("text/plain"); %>`
- `isThreadSafe="`**`true`** `| false"`
  true: normál szervlet feldolgozást jelöl, ahol több egyidejű kérést párhuzamosan lehet feldolgozni egyetlen szervletpéldánnyal, feltételezve, hogy gondoskodtunk a példányváltozók szinkronizált hozzáféréséről
  false: a szervletnek implementálnia kell a *SingleThreadModel* interfészt
- `session="`**`true`** `| false"`
  true: az előredefiniált session változó (*HttpSession* típusú) a létező session-höz kötődik (ha már van ilyen), vagy ha még nem létezik, akkor egy új session jön létre
  false: nem használ semmilyen session-t, így a session változóhoz való bármilyen hozzáférési próbálkozás hibát eredményez
- `extends="package.class"`
  a generált szervlet szuperosztálya adható itt meg
- `info="message"`
  a *getServletInfo* metódussal kérhető le ez az információ

- `errorPage="url"`
  egy olyan JSP, amelynek fel kell dolgoznia az eldobott, de el nem kapott *Throwables* kivételeket
- `isErrorPage="true | false"`
  ez a lap funkcionálhat-e hibalapként más JSP-k számára. Alapértelmezés: false.
- `language="Java"`
  az alkalmazott nyelv

1.4.4.2. `include` direktíva

Fordítási időben teszi lehetővé file-ok beillesztését. Alakja:

```
<%@ include file="relative url" %>
```

A behívandó file címe általában relatív módon van megadva, de jelezhető a rendszer számára, hogy a webszerver HOME könyvtárától adjuk meg a címet relatív módon (ekkor az url-nek ″/″ jellel kell kezdődnie). A beillesztett file-t JSP szövegként kezeli, így statikus HTML-en kívül szkriptelemeket, direktívákat, akciókat is tartalmazhat.
Pl. számos site-on minden oldalon található egy kis navigációs terület. HTML frame-ek helyett ezt gyakran egy kis táblázattal oldják meg úgy, hogy a HTML-t minden oldalra bemásolják. Ha változik, akkor viszont valamennyi helyen módosítani kell. Ilyen esetekben hasznos az *include* direktíva:

```
<html>
<head>
. . .
</head>
<body>
<%@ include file="/navbar.html" %>
<!-- az oldal saját jellemzői -->
</body>
</html>
```

Lényeges, hogy ez fordítási időben kerül be! Ha a navigációs file megváltozik, akkor újra kell fordítani mindazon JSP-ket, amelyek hivatkoznak rá.
Ha egy gyakran változó file-t akarunk beilleszteni, akkor alkalmazzuk inkább a *jsp:include* akciót, mely a JSP-re vonatkozó kéréskor olvassa be a file-t.

1.4.5. Előredefiniált változók

A JSP kifejezések és szkriptletek kódolásának egyszerűsítésére rendelkezésre áll néhány automatikusan definiált változó (implicit objektum). Ezek:

- `request`
  Ez a HttpServletRequest. Lekérdezhetők vele a paraméterek**,** a kérés típusa (GET, POST), ill. a beérkező HTTP fejlécek (cookie-k, stb.).
- `response`
  Ez a HttpServletResponse. Ezen keresztül lehet a kliensnek válaszolni.
- `out`
  Ez a PrintWriter objektum. Szinte kizárólag csak a szkriptletekben fordul elő, hiszen a JSP kifejezések eredménye automatikusan az output-ra kerül.
- `session`
  Ez a HttpSession. A session-ök automatikusan létrejönnek, így ez a változó még akkor is kötődik, ha nem volt beérkező session-referencia. Az egyetlen kivétel, amikor a megfelelő page-direktívával kikapcsoljuk. Ebben az esetben a session változóra való hivatkozás fordítási hibát eredményez.
- `application`
  Ez a ServletContext (**getServletConfig().getContext()**).
- `config`
  Az oldalra vonatkozó ServletConfig objektum.
- `page`
  A *this* szinonimája. Java-ban nem sok értelme van, de azért tették bele, mert számoltak azzal, hogy a szkriptnyelv más is lehet majd, nem csak Java.

Nézzük meg az 1.3.1. fejezetben szereplő példa JSP-s változatát. Ehhez tehát szükség lesz egy HTML file-ra, ill. egy .jsp file-ra is, amit az előző file-ból meghívunk. Ha ezt a Tomcat-ben próbáljuk ki, akkor a jsp file-t a Tomcat-en belül a webapps/proba könyvtárban helyezzük el.

```
<html>
<body>
<form action="http://127.0.0.1:8008/proba/Szervlet_pelda.jsp">
Kód: <input type="text" name="code"><br>
<input type="submit">
</form>
</body>
</html>
```

```
<%@ page import="java.sql.*" %>
<html><body>
<%
   boolean talalt = false;

   try
   {
      Class.forName("org.postgresql.Driver");
      Connection                              conn                                =
      DriverManager.getConnection("jdbc:postgresql://127.0.0.1/"+"areas",
      "jabba","jabba");
      Statement  st   = conn.createStatement();

      ResultSet rs = st.executeQuery("select name from codes "+
      +"where code='"+request.getParameterValues("code")[0]+"'");
      if (rs!=null)
      {
         while (rs.next())
         {
%>
<%= rs.getString("name") %><br>
<%
            talalt = true;
         }
      }
      if (!talalt) %>Sajnos errol nem találtam semmit!<br><%
      rs.close();
      st.close();
      conn.close();
   }
   catch (java.sql.SQLException sqle) {
      out.println("<b>SQL hiba:</b> "+sqle.toString());
   }
   catch (ClassNotFoundException cnfe) {
      out.println(cnfe.toString());
   }
%>
</body></html>
```

1.4.6. JSP akciók

Az akciók XML szintaktikát alkalmaznak, s a szervletmotor viselkedését befolyásolják. Dinamikusan be lehet szúrni egy file-t, újra lehet hasznosítani JavaBean komponenseket, át lehet irányítani a felhasználót egy másik lapra, vagy HTML-t lehet generálni a Java Plugin számára. Az egyes akciók:

- `jsp:include`

  Egy file-t lehet beilleszteni a JSP-be. Szintaxisa:

  `<jsp:include page=”relative url” flush=”true” />`

  A file beszúrása akkor történik meg, amikor hivatkozunk a JSP-re. Ez egy kis hátrányt jelent a teljesítményben, s eleve kizárja, hogy a beillesztett lap általános JSP kódot

tartalmazzon (pl. HTTP header-t nem állíthat be). Akkor hasznos például, amikor híreket akarunk beilleszteni a JSP lapba (ekkor ui. a beillesztendő lap tartalma gyakran változik).

- `jsp:useBean`

  Egy JavaBean tölthető be vele a JSP-be. Segítségével megvalósítható a Java osztályok újrahasznosítása anélkül, hogy fel kellene áldozni azt a kényelmet, amit a JSP nyújt a szervletek felett. A legegyszerűbb szintaxis:

  ```
  <jsp:useBean id=”name” class=”package.class” />
  ```

  Ez általában a következőt jelenti: „példányosítsd a class után megadott osztályt, s rendeld hozzá az id-ben megadott változóhoz”. Megadható még egy scope attribútum is, ami a bean-t akár több laphoz is hozzárendeli, nem csak az aktuálishoz.

  Ez utóbbi esetben jó lenne, ha a már létező bean referenciáját kapnánk meg, s ekkor a jsp:useBean akció azt jelenti, hogy csak akkor példányosít, ha ezen id-vel és scope-pal még nem létezik ilyen bean.

  Miután van egy bean-ünk, beállíthatjuk a tulajdonságait a jsp:setProperty akcióval, vagy egy szkriptlettel, ahol az id-ben megadott változónév segítségével meghívjuk a bean megfelelő metódusait.

Ha azt mondjuk, hogy egy bean-nek van egy X típusú, foo nevű jellemzője, akkor ez 2 dolgot jelent:

1. az osztálynak van egy getFoo metódusa, ami X típusú értéket ad vissza
2. az osztálynak van egy setFoo metódusa. ami X típusú argumentumot vár

Példa:

```
<html>
<body>
<jsp:useBean id="test" class="proba.SimpleBean" />
Uzenet: <jsp:getProperty name="test" property="message" /><br>
<jsp:setProperty  name="test"  property="message"  value="Java  Server
Pages" />
Uzenet: <jsp:getProperty name="test" property="message" /><br>
<% test.setMessage("igy is lehet"); %>
Uzenet: <%= test.getMessage() %>
</body>
</html>
```

```
package proba;

public class SimpleBean
{
   private String message = "Nincs uzenet megadva.";

   public String getMessage() {
      return message;
   }
```

```
    public void setMessage(String message) {
       this.message = message;
    }
}
```

Egy bean használatának a legegyszerűbb módja:

```
<jsp:useBean id="name" class="package.class" />
```

Ez tehát betölti a bean-t. A jellemzők lekérdezése, módosítása:

```
<jsp:getProperty . . .>          <jsp:setProperty . . .>
```

Alkalmazható a konténer-formátum is:

```
<jsp:useBean . . .>
   törzs
</jsp:useBean>
```

Ezzel tudjuk jelölni, hogy a törzsben megadott kód csak akkor fusson le, amikor a bean-t először példányosítjuk, s nem akkor, amikor megtaláltuk s használjuk. A törzs tehát inicializációs szerepet tölt be.

- `jsp:setProperty`

  Miután van egy referenciánk a bean-re, ezzel lehet beállítani a tulajdonságait.

  Kétféleképpen lehet használni:

  1. ```
     <jsp:useBean id="myName" . . .>
     . . .
     <jsp:setProperty name="myName" property="someProperty" . . ./>
     ```

     Ebben az esetben a setProperty mindenképpen lefut, még akkor is, ha most hozta létre a bean-t, vagy egy létező bean-t talált.

  2. ```
     <jsp:useBean id="myName" . . .>
     <jsp:setProperty name="myName" property="someProperty" . . ./>
     </jsp:useBean id="myName">
     ```

     Csak akkor fut le a setProperty, ha egy új objektum lett példányosítva.

  Négy lehetséges attribútuma lehet:

  1. `name` – (kötelező). A bean neve, amelyiknek a tulajdonságát be akarjuk állítani (referencia). Ennek a névnek már korábban szerepelnie kellett egy jsp:useBean -ben.
  2. `property` – (kötelező). Az a tulajdonság, amit be akarunk állítani. Speciális érték: "*". Ez azt jelenti, hogy mindazon tulajdonságok *set* metódusa meghívódik, amelyek szerepeltek a *request* paraméterben (azaz a *request* paraméterekben szereplő összes tulajdonságot beállítja, ha van ilyen tulajdonság).

3. `value` – (opcionális). Beállítja a tulajdonság értékét. Mivel sztringet adunk meg, ezért konvertálni kell az értéket. Erre a megfelelő *valueOf* függvényt használja a rendszer. Nem használható együtt a *param* paraméterrel, de mindkettő el is hagyható.
4. `param` – (opcionális). Itt adható meg, hogy a tulajdonság melyik *request* paraméter értékét vegye fel. Ha nincs ilyen nevű paraméter, akkor nem történik semmi (tehát még *null*-t sem ad át a *setXxx* metódusnak). Azaz: ha van ilyen nevű *request* paraméter, akkor vegye fel ennek az értékét, különben meg ne csináljon semmit.

Mint látható, mind a *value*, mind pedig a *param* elhagyható. Ez azonos azzal, mintha a *param*-nak magának a tulajdonság nevét adnánk meg (azaz azon *request* paraméter értékét vegye fel a tulajdonság, amelyiknek a neve azonos a tulajdonság nevével).

A *request* paraméterértékek automatikus hozzárendelése a megfelelő tulajdonságokhoz: elhagyjuk mind a *value*, mind a *param* paramétereket, s azt írjuk, hogy *property=”*”.

- `jsp:getProperty`

  Egy bean tulajdonságértékét lekérdezi, sztringgé konvertálja, s beilleszti az output-ba. A két kötelező attribútum:

  1. `name` – a bean-referencia (ezt korábban már megadtuk a jsp:useBean -nel)
  2. `property` – mely tulajdonságát akarjuk lekérdezni

  ```
  <jsp:getProperty name=”beanName” property=”someProperty” />
  ```

- `jsp:forward`

  A kérést átirányítja egy másik lapra. A címet relatív módon kell megadni. A cím lehet statikus, vagy futási időben kiszámított, pl.:

  ```
  <jsp:forward page=”/utils/errorReporter.jsp” />
  <jsp:forward page=”<%= Java kifejezés %>” />
  ```

- `jsp:plugin`

  A böngésző típusának megfelelő OBJECT vagy EMBED tag beszúrása, mellyel arra lehet utasítani a bögészőt, hogy egy applet-et a Java plugin-nel jelenítsen meg.

1.4.7. Klienskapcsolati környezet („session”) kezelése JSP-ben

A session-ökről már a szervleteknél is volt szó, s a JSP-ben is ugyanúgy elérhetőek. A következő példa ennek kezelését mutatja be:

```
//SessionObject.jsp
<html><body>
<h1>Session ID: <%= session.getId() %></h1>
<h3><li>Session letrehozasi idopontja:
<%= new java.util.Date(session.getCreationTime()) %>
```

```
<li>Regi MaxInactiveInterval:
<%= session.getMaxInactiveInterval()  %>
<%  session.setMaxInactiveInterval(5); %>
<li>Uj MaxInactiveInterval:
<%= session.getMaxInactiveInterval()  %></li>
<h2>Ha a "kutyus" session objektum meg el,
akkor ez az ertek nem-null lesz:<h2>
<h3><li>A "kutyus" session obj. erteke:
<%= session.getValue("kutyus") %></h3>
<% session.putValue("kutyus","UBUL"); %>
<h1>A kutyus neve:
<%= session.getValue("kutyus") %></h1>

<form type=post action=SessionObject2.jsp>
<input type=submit name=submit
value="Klienskapcsolat megszuntetese"></form>

<form type=post action=SessionObject3.jsp>
<input type=submit name=submit
value="Session obj. tovabbadasa"></form>

</body></html>
```

A **session** nevű objektum implicit objektum, így rögtön elérhető, nem kell külön létrehozni. A **getId()**, **getCreationTime()** és **getMaxInactiveInterval()** függvények segítségével információt jelenítünk meg erről az objektumról.

Ha először hozzuk be ezt az oldalt, akkor a **MaxInactiveInterval** kezdeti értéke pl. 1800 másodperc (30 perc). Ez az érték attól függ, hogy a JSP/szervlet konténer hogyan van bekonfigurálva. A **MaxInactiveInterval** értékét az érdekesség kedvéért 5 másodpercre állítjuk. Ha az oldalt 5 másodpercen belül frissítjük, akkor a következőt kapjuk:

```
A "kutyus" session obj. erteke: UBUL
```

Ha azonban hosszabb idő múlva frissítünk, akkor az objektum értéke **null** lesz.

Annak megvizsgálására, hogy a session információk hogyan adódnak át az egyes lapok között, ill. hogyan lehet közvetlenül megszüntetni egy kapcsolatot még mielőtt lejárna az ideje, két másik JSP lapot is készítünk. Az első (amit az előző lapról a "Klienskapcsolat megszuntetese" gombbal érhetünk el) beolvassa a session információt, majd explicit módon megszünteti azt:

```
//SessionObject2.jsp
<html><body>
<h1>Session ID: <%= session.getId() %></h1>
<h3><li>A "kutyus" session obj. erteke:
<%= session.getValue("kutyus") %></h3>
<% session.invalidate(); %>
</body></html>
```

Ennek kipróbálásához frissítsük a **SessionObject.jsp**-t, majd váltsunk át azonnal a **SessionObject2.jsp**-re. Ekkor még láthatjuk az objektum értékét („UBUL”), s rögtön (mielőtt még letelne az 5 másodperc) frissítsük az oldalt. Látható, hogy a session meg lett szüntetve, s az objektum értéke is **null** lett.

Ha visszalépünk a **SessionObject.jsp**-re és frissítjük, akkor egy új 5 másodperc élettartamú session-t kapunk. Kattintsunk a "Session obj. tovabbadasa" gombra, mely a következő **SessionObject3.jsp** lapra visz el minket:

```
//SessionObject3.jsp
<html><body>
<h1>Session ID: <%= session.getId() %></h1>
<h3><li>A "kutyus" session obj. erteke:
<%= session.getValue("kutyus") %></h3>

<form type=post action=SessionObject.jsp>
<input type=submit name=submit value="Vissza">
</form>

</body></html>
```

Mivel ez a lap nem szünteti meg a session-t, így az objektum addig életben marad, amíg az 5 másodperc letelte előtt frissítjük az oldalt.

2. PHP

A PHP egy HTML-be ágyazott szkriptnyelv. Szintaktikailag leginkább a Perl-hez, C-hez, Java-hoz hasonlít. A nyelv célja: dinamikus oldalak gyors elkészítése.

2.1. A nyelv rövid története

A PHP születése 1994-re tehető, melyet kitalálója, Rasmus Lerdorf 1995-ben tett közzé az Interneten Personal Home Page Tools néven. Még ebben az évben kijött a 2-es verzió PHP/FI néven, melyben már egy form-feldolgozó rész is volt. A nyelv már támogatta az mSQL adatbáziskezelőt. 1997-ben új fejlesztők is bekapcsolódtak, újraírták az interpretert, s így jött létre a PHP3. A nyelv népszerűségét mutatja, hogy rohamosan terjed, s manapság már legalább 1.000.000 site-on használják.

2.2. Mihez hasonlítható a PHP?

Hasonló nyelvek: Perl, ASP, JSP, Cold Fusion Markup Language (CFML). Ezek közül szintaktikailag a Perl a legbonyolultabb. Perl programot nagyon nehéz olvasni, s sokan ezt hátrányként hozzák fel, de az igazi Perl „guruk" erre azt mondják, hogy „Perl programot írni kell, nem olvasni". A PHP sok mindent átvett a Perl-ből, de fejlesztése során végig szem előtt tartották, hogy a forrás könnyen olvasható legyen. Az ASP a Microsoft saját találmánya, így csak IIS szerverekkel együtt használható (bár már létezik Unix alá is, viszont ezt nem a Microsoft szállítja). Az ASP könnyen átlátható; hasonlít a Visual Basic-re.
A Cold Fusion az Allaire cég nevéhez fűződik (ők készítették a HomeSite nevű weboldal-szerkesztőt is). Szintaktikája a HTML-hez hasonló.
A PHP több nyelv formalitását is ötvözi; elsajátítása nagyon egyszerű.
Egy programozási nyelv hatékony használatához azonban nem elegendő az áttekinthető szintaktika. Lényeges a nyelv által biztosított függvénykészlet is. Szerencsére a PHP esetében igazán nem panaszkodhatunk, hiszen a nyelv a következő műveleteket támogatja: adatbáziskezelők elérése, tömbkezelés, matematikai függvények, file-kezelés, programfuttatás, képlétrehozás -manipulálás, kódolás, hálózat, PDF, sztringkezelés, tömörítés, XML, stb. Az Interneten is számos hasznos kiegészítőt találhatunk; ezek közül legismertebb a PHPLib.

2.3. Nyelvi struktúrák

A PHP-ban megtalálhatók a C-ben megszokott vezérlési struktúrák (if-else-if, while, do-while, for, break, continue, switch), ill. használhatók a jól ismert operátorok is (sztringkezelő, matematikai, logikai, összehasonlító, értékadó).

A PHP-ban a Perl-hez hasonlóan nem kötöttek a változótípusok, s a változók egyeztetése automatikus konverzióval történik (pl. értékadásnál).

A PHP programok szintén az űrlapokon megadott adatokkal dolgoznak, így lényeges a webszerver által átadott változók elérése. A PHP háromféle átadási módot különböztet meg:

1. a GET metódus által átadott adatok
2. a POST metódus által átadott adatok
3. HTTP Cookie változók

A kívülről kapott paraméterek kezelése nagyon egyszerűvé válik azáltal, hogy a PHP minden egyes paramétert automatikusan átalakít a programban használható változókká. Pl. ha egy form-ban a kitöltendő mező neve *cim*, akkor a szkript meghívásakor ebből egy *$cim* nevű globális változó lesz. Bonyodalmat az okozhat, ha több helyről is (GET, POST, Cookie) kapunk változókat ugyanazzal a névvel. Ekkor felállíthatunk egy tetszőleges prioritást, vagy akár le is tilthatjuk az egyik forrás automatikus átalakítását.

A PHP-ban a következő változótípusokat használhatjuk: egész szám, valós szám, sztring, tömb, ill. objektum. A nyelv tehát támogatja objektumok/osztályok létrehozását is, melyet elsősorban könyvtárak írására használnak, ui. így az objektumokba zárt változók nem érintik globális társaikat. Az osztályokhoz konstruktorok is megadhatók, továbbá létezik az öröklődés fogalma is.

A tömbkezelés is a PHP egy nagyon fontos része, mivel az adatbázisból nyert adatainkat általában egy vektorban kapjuk vissza (ebből már látható, hogy csakúgy mint Perl-ben, itt is tárolhatunk különböző típusú elemeket egy vektorban). Lehetőségünk van HTML űrlapokban elhelyezett adatokat is vektorként megkapni (gondoljunk csak a checkbox-ok adataira – vektorok használatával egyszerűbben végig tudunk menni a kiválasztott elemeken).

Az asszociatív tömbök szintén nagyon hasznosak, hiszen sokkal egyszerűbb az adatbázis egy rekordját tároló tömbből úgy kiválasztani egy elemet, hogy az oszlop nevével hivatkozunk rá (pl. `szemely['nev']`). Számos tömbkezelő függvény is rendelkezésünkre áll, pl. rendezés (véletlenszerűvé is), elem beszúrása, veremfüggvények, elemcsere, bejárás, stb.

2.4. Fejlesztői környezet kialakítása

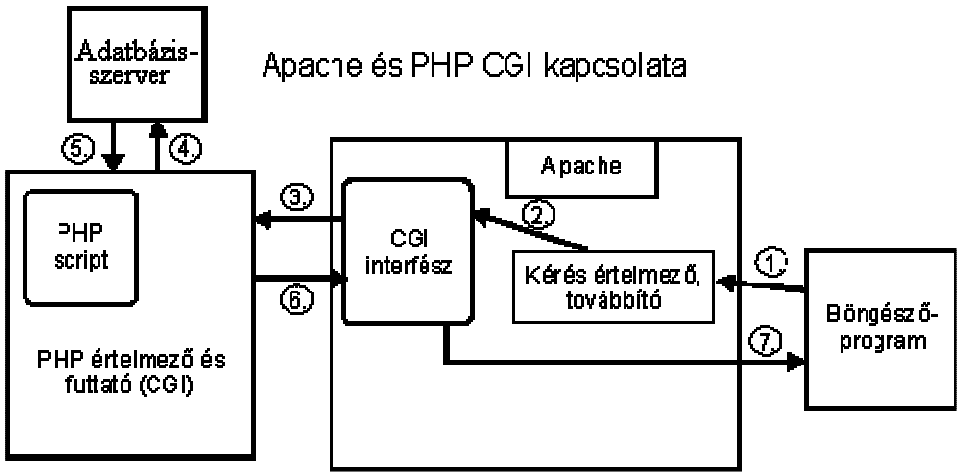
A következőkben azt nézzük meg, hogy hogyan tudjuk a web-re készülő alkalmazásainkat a saját otthoni gépünkön elkészíteni. Ehhez telepíteni kell a PHP-t (4. sz. Melléklet), továbbá szükségünk lesz egy futó webszerre (pl. Apache, lásd 2. sz. Melléklet), ill. egy adatbázisszerverre (pl. MySQL (5. sz. Melléklet), PostgreSQL (6. sz. Melléklet)). A loopback interfész segítségével (127.0.0.1-es IP cím) hálózati környezetet tudunk szimulálni a saját gépünkön, így szerencsére egy tényleges hálózat nélkül is mindent ki tudunk próbálni.

Először is nézzük meg, hogy mi történik a színfalak mögött, vagyis vizsgáljuk meg az Apache ill. a PHP kapcsolatát. A PHP-t kétféleképpen használhatjuk:

- használhatjuk normál CGI-ként, mint pl. a Perl-t, vagy
- futtathatjuk Apache modulként

Mindkét módszernek vannak előnyei és hátrányai is. Ha CGI-ként használjuk, akkor a szkriptek értelmezéséhez mindig egy új PHP intrepreter indul el, mely dolga végeztével megszűnik létezni. Apache modul esetében viszont az interpreter addig a memóriában marad, amíg az őt futtató Apache kiszolgáló is ott van. Ez már egy kisebb forgalmú site-on is elegendően nagy előnnyel jár ahhoz, hogy inkább modulként futtassák a PHP-t. Van viszont egy nagy hátránya, ami miatt sok helyen mégis a CGI-s változatot használják: modulként nincs rá lehetőség, hogy más felhasználóként fusson, mint az *apache*. Így ha több fejlesztő használja ugyanazt a szervert (például egy szolgáltatónál), akkor modul esetében nem oldható meg, hogy ne tudják elérni egymás állományait.

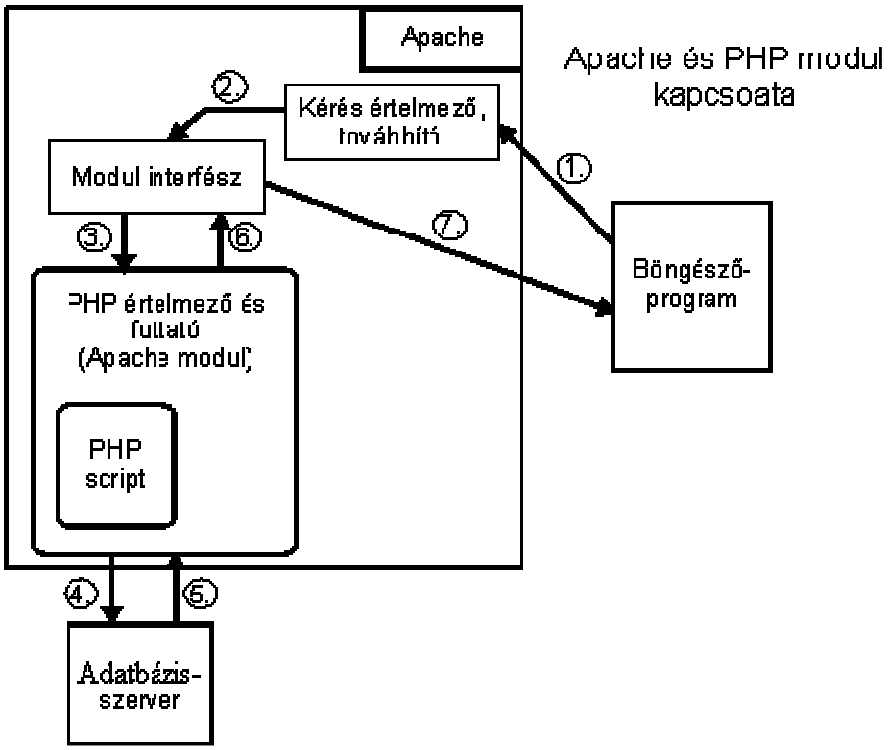
A következő ábra a böngészőtől induló és annál végződő folyamatot mutatja be CGI esetében:



Az egyes lépések:

1. A böngésző elküld a szervernek egy HTTP kérést (legyen pl. a kért oldal neve: index.php).
2. A webszerver ezt fogadja, értelmezi és eldönti, hogy mit kell vele kezdenie. Ebben az esetben a php kiterjesztésről tudja, hogy tovább kell adnia egy CGI programnak.
3. A CGI interfész a szabványnak megfelelő beállításokkal (környezeti változók, adatcsere) elindítja a PHP CGI-s változatát.
4. – 5. Feldolgozza a szkriptet, miközben külső adatforrásokat is igénybe vesz (egy adatbázisszervert pl.).
6. Az előállított adatokat (pl.: HTML oldal, kép, hang) a standard ki/bemeneten keresztül a PHP átadja a CGI interfésznek.
7. A CGI interfész az Apache-on keresztül továbbítja ezt a böngészőnek.

Ugyanez a műveletsor modul esetén:



A két mód közötti alapvető különbség jól látható az ábrán: a kérésértelmező ez esetben nem a CGI interfésznek adja tovább a hívást, hanem a memóriában várakozó PHP modulnak a modul interfészen keresztül. Míg a CGI-s változatnak minden híváshoz újra el kell indulnia,

addig a modul az Apache része, így egy gyors függvényhívással sokkal gyorsabban elintézhető a kérés teljesítése.

2.5. Telepítés Linux alá

A PHP-t szintén Linux (Mandrake 7.2) alatt próbáltam ki, melyet előre lefordított csomagokból telepítettem. Szerencsére az összes előbb említett szoftver (Apache, PHP, MySQL, PostgreSQL) a disztibúció részét képezi, így már a Linux installálásakor fel lehet tenni mindezen programokat. A kérdés inkább az, hogy hogyan tudjuk ezen szoftvereket finomhangolni, hozzáigazítani a saját igényeinkhez.

Tegyük fel, hogy egy *jabba* nevű felhasználóként dolgozunk (`HOME` könyvtára: `/home/jabba`). Itt hozzunk létre egy `public_html/php` könyvtárat, s a PHP programjainkat majd itt helyezzük el. A PHP működőképes voltát a következő példaprogrammal tesztelhetjük:

```
<html>
<body>
<?php echo phpinfo(); ?>
</body>
</html>
```

Mentsük el `test.php` néven, majd a böngészőben hívjuk meg:

`http://127.0.0.1/~jabba/php/test.php`

Egy táblázatot kapunk eredményül, mely a PHP-ról ill. az Apache-ról szolgáltat sok hasznos információt. Ha hibaüzenetet kapunk, akkor nézzük át még egyszer a konfigurációs beállításokat (lásd 4. sz. Melléklet).

2.6. Ismerkedés a PHP nyelvvel

Nézzük meg a „Hello World!” program PHP-s változatát:

```
<html>
<body>
<?php echo ″<h1><b>Hello World!</b></h1>″ ?>
</body>
</html>
```

Látható, hogy a kód a JSP-hez teljesen hasonló módon ágyazható be egy HTML oldalba, csak itt a PHP kódot `<?php` és `?>` jelek közé kell tenni. Ha behozzuk ezt az oldalt egy böngészőbe, s megnézzük az oldal forrását, akkor csak egyszerű HTML-t fogunk látni, hiszen a szerver oldalon megtörtént a PHP kód értelmezése, s a kliens már csak a kód által előállított HTML szöveget kapja meg.

2.7. Űrlap adatainak a feldolgozása

Az űrlapok mezőit egyszerűen a mezők nevével érhetjük el PHP-ból. Itt is beszélhetünk GET ill. POST típusú paraméterátadásról (az ezek közti különbségről már volt szó a szervleteknél). Dinamikusan generált oldalak esetében gyakori, hogy az űrlap adatait nem egy különálló programmal dolgoztatják fel, hanem már magát az űrlapot is egy program állítja elő, s ugyanez a program kapja meg a kitöltött adatokat is feldolgozásra. Ilyenkor az űrlap *action* paraméterénél a php file saját nevét kell megadni. Ezt is lehet automatizálni, ui. a $PHP_SELF értéke mindig az aktuális file neve:

```
<html>
<head>
 <title>Űrlap</title>
</head>
<body>
<form action="<?php echo $PHP_SELF?>" method="post">
Vezetéknév: <input type="text" name="vezeteknev">
Keresztnév: <input type="text" name="keresztnev"><p>
<input type="submit" value="Elküld">
</form>
<?php
 echo "A megadott név: $vezeteknev $keresztnev<br>";
?>
</body>
</html>
```

2.8. Adatbázis alapú weboldalak készítése

A következőkben nézzük meg, hogy a PHP-ból hogyan tudunk adatbázishoz kapcsolódni, abba adatokat beszúrni, az adatbázisból adatokat lekérdezni, stb.
Linuxos körökben két igazán népszerű (és ingyenes) adatbáziskezelő-rendszer van: a MySQL és a PostgreSQL. Ez utóbbi használatára már láttunk példákat korábban, így most nézzük meg a MySQL-t. A MySQL telepítésével ill. kezdeti beállításával az 5. sz. Melléklet foglalkozik.

2.8.1. WebDB példa

Először is készítünk egy adatbázist, s feltöltjük néhány adattal. Ezután elkészítünk néhány PHP programot, melyek ezzel az adatbázissal fognak kommunikálni.
Első lépésben lépjünk be root-ként a MySQL-be (a `$`, ill. a `mysql>` a prompt-ot jelöli):

```
$mysql –u root –p
```

Majd hozzunk létre egy adatbázist, ill. azon belül egy táblát:

```
mysql>create database example;
      use example;
      CREATE TABLE mytable
      ( name char(30),
        phone char(15)
      );
```

Töltsük fel néhány adattal a táblát:

```
mysql>insert into mytable values(”Homer Simpson”, ”555-1234”);
      insert into mytable values(”Bart  Simpson”, ”555-4321”);
      insert into mytable values(”Lisa Simpson”, ”555-3214”);
      insert into mytable values(”Marge Simpson”, ”555-2314”);
      insert into mytable values(”Maggie Simpson”, ”555-3142”);
```

Most létrehozunk egy webuser nevű felhasználót, aki használhatja majd az adatbázist. Csak a localhost-ról kapcsolódhat, s hozzáférhet az example adatbázishoz:

```
mysql>GRANT usage
      ON example.*
      TO webuser@localhost;
```

A webuser felhasználó az example adatbázis összes tábláján kiadhatja az insert, delete, select parancsokat. Jelszónak – az egyszerűség kedvéért – szintén ’webuser’-t adunk meg.

```
mysql>GRANT select, insert, delete
      ON example.*
      TO webuser@localhost
      IDENTIFIED BY ’webuser’;
```

Ezzel elkészítettük az adatbázist; most nézzük meg a PHP programokat: 2 szkriptet készítünk: ez egyikkel lekérdezünk (index.php), míg a másikkal új elemeket töltünk fel (add.php). Az első szkript (index.php):

```
<html>
   <head><title>Web-es adatbázis példa</title></head>
<body bgcolor=#ffffff>
<h1>Adatok a 'mytable' táblából</h1>
<?php
mysql_connect("localhost", "webuser", "webuser");
$query = "SELECT name, phone FROM mytable";
$result = mysql_db_query("example", $query);
if ($result)
{
   echo "Ezeket az elemeket találtam az adatbázisban:<ul>";
   while ($r = mysql_fetch_array($result))
   {
      $name = $r["name"];
      $phone = $r["phone"];
      echo "<li>$name, $phone";
   }
   echo "</ul>";
}
else
```

```
{
   echo "Nincs adat.";
}
mysql_free_result($result);
?>

<p><a href="add.php">Új elem felvitele</a>
</body>
</html>
```

A következő szkript (add.php) bekéri a felhasználótól az adatokat, majd hozzáadja ezen adatokat az adatbázishoz:

```
<html>
   <head>
      <title>Web-es adatbázis példa</title>
   </head>
<body bgcolor=#ffffff>

<?php
mysql_connect("localhost", "webuser", "webuser");
if (isset($name) && isset($phone))
{
   $query = "INSERT INTO mytable VALUES ('$name', '$phone')";
   $result = mysql_db_query("example", $query);
   if ($result)
   {
      echo "<p>$name bekerült az adatbázisba</p>";
   }
}
?>

<h1>Új elem felvitele</h1>
<form>
Név: <input type=text name='name'><br>
Telefon: <input type=text name='phone'><br>
<input type=submit value="Elküld">
</form> <p>
<a href="index.php">Vissza</a>
</body>
</html>
```

3. Perl – CGI

3.1. A Perl nyelv választásának indokai

A Perlt programozóknak tervezték általános feladatokhoz, amelyek túl nehezek vagy túl hordozhatóság-érzékenyek lennének, ill. amelyeket túl fárasztó vagy bonyolult C-ben vagy valamely más nyelven kódolni. Egy Perl nyelven megírt átlagos CGI szkript mérete, ill. a megírására fordított idő csupán töredéke egy hasonló C/C++ nyelvű programénak.
A Perl hatékony szerkezetei lehetővé teszik, hogy minimális erőbefektetéssel azonnali megoldást hozzunk létre. A Perl hordozható, ingyenes, s bármikor rendelkezésre áll, hiszen a mai Linux disztribúciók már tartalmazzák.
A Perl igazi ereje a hatékony szövegfeldolgozó képességében rejlik, ui. a reguláris kifejezéseket beépítve, nyelvi szinten tartalmazza; ezáltal bátran mondhatjuk, hogy a Perl a jelenleg létező legnagyobb tudású szövegfeldolgozó nyelvek egyike. A Perl általános célú nyelvként jött létre, de webprogramozáshoz is tökéletes, hiszen a web óriási mennyiségű szöveges információt tartalmaz. A CGI programoknak általában nagyon nagy mennyiségű szövegfeldolgozó műveletet kell elvégezniük. Ilyen művelet a beérkező adatok elemzése, adatbázisok elérése, HTML oldalak készítése, stb.

3.2. A CGI áttekintése

A CGI (Common Gateway Interface) egy módszert definiál, amellyel adatokat cserélhetünk a webkliensek és a webszerveren futó programok, az ún. CGI-programok között. A CGI-programok szerepe a statikus HTML oldalakban rejlő lehetőségek kiterjesztése. Mivel a CGI-programok a szerveren futnak, így nagyon hatékony eszközök olyan feladatok esetén, amelyek megkövetelik a kölcsönhatást a szerverrel.

3.3. CGI-szkript futtatása

UNIX alapú rendszerekben egy Perl nyelvű szkript végrehajtható file-ként futtatható. A CGI futtatását engedélyezni kell az Apache-ban (lásd 2. sz. Melléklet). Nézzük meg a „Hello World!” program CGI-s változatát:

```
#!/usr/bin/perl -w

print <<END;
Content-type: text/html

<html>
<body>
<h1>Hello World!</h1>
</body>
</html>
END
```

A program első sora a file tartalmát Perl programként azonosítja. A #! karakterek után meg kell adni a Perl interpreter pontos elérési útját (a `which perl` paranccsal könnyen lekérdezhetjük a pontos helyét – feltéve hogy szerepel az elérési útban). A **print** utasítással egy HTML szöveget iratunk ki az END jelig. Ebben az első sor a legfontosabb: a Content-type azonosítja a létrehozandó kimenet típusát. Ezt közvetlenül egy üres sor követi. A legtöbb kezdő CGI programja hibás, mert elfeledkeznek erről az üres sorról, amely elválasztja a fejlécet az azt követő opcionális törzstől. Az üres sor után jöhet a HTML kód.
Az előző programot a 2. sz. Mellékletben leírtaknak megfelelően a következő helyen kell elhelyezni:

`http://localhost/~jabba/cgi-bin/hello.cgi`

3.4. A CGI.pm modul

Az 5.004-es változattal kezdve a szokásos Perl forgalmazás magában foglalja a Lincoln Stein által írt CGI.pm modult, mely nagyban leegyszerűsíti a CGI-programok írását. Mint a Perl maga, a CGI is rendszerfüggetlen, így mindenféle platformon használható. Használatához csupán a

```
use CGI;
```

sort kell beszúrni a program elejére. A *use* utasítás abban hasonlít a C programozásban lévő *#include* utasításhoz, hogy a fordítás ideje alatt átvesz egy programot egy másik file-ból, de választható argumentumokat is megenged, megadva, hogy mely függvényeket és változókat lehet elérni abból a modulból. Ekkor az elnevezett függvényeket és változókat úgy lehet elérni, mintha azok az adott program sajátjai lennének. A CGI.pm egy egész sor kényelmi funkciót tartalmaz az egyszerűsítésre, így az előző példát a következőképpen is írhattuk volna:

```
#!/usr/bin/perl -w

use CGI qw(:standard);

print header(), start_html();
print h1("Hello World!");
print end_html();
```

A print után álló rutinok mindegyike egy karakterláncot (sztringet) ad vissza eredményül.
A CGI.pm ún. *import címkéket* biztosít, amelyek az importálandó függvényeket helyettesítik (ezek mindegyike kettősponttal kezdődik). A `:standard` a szokásos szolgáltatásokat importálja, úgy mint HTML tag-eket, form-okat előállító függvényeket, ill. az összes argumentum-kezelő módszert.

3.5. Adatbáziskezelés CGI-ből

Nézzük meg az 1.3.1. fejezetben bemutatott példa Perl-CGI-s változatát. A HTML file, amely majd meghívja a CGI-programot:

```
<html>
<body>
<form action="http://localhost/~jabba/cgi-bin/postgres.cgi">
Code: <input type="text" name="code"><br>
<input type="submit">
</form>
</body>
</html>
```

A megfelelő CGI szkript:

```
#!/usr/bin/perl -w

use Pg;
use CGI qw(:standard);

print header(), start_html();

      $conn = Pg::connectdb("dbname=areas");
      die $conn->errorMessage unless PGRES_CONNECTION_OK eq $conn->status;

      $code = param("code");

      $result = $conn->exec("SELECT name FROM codes WHERE code='$code'");
      die $conn->errorMessage unless PGRES_TUPLES_OK eq $result->resultStatus;

      while (@row = $result->fetchrow) {
            print @row, "\n";
      }

print end_html();
```

Mint látható, az elv itt is azonos: az átadott paraméterekből a megfelelő SQL lekérdezés előállítása, majd a lekérdezés eredményének visszajuttatása a kliens számára. Az adatbáziskezelő függvények nevei nagyon hasonlóak a PHP-ban látottakhoz. Ezek használatához még importálni kellett a Pg modult (ezt a *postgresql-perl* csomag tartalmazza), melyben a PostgreSQL-t kezelő függvények találhatóak.

3.6. CGI programok hibakeresése

A Perl-ben írt CGI programok parancssorból is futtathatóak. Ekkor interaktív módon meg lehet adni az egyes paramétereket (kulcs=érték formában). Ha valahol hiba van, akkor egyszerűbb itt elolvasni a hibaüzenetet, mintsem különböző log file-okból kideríteni, hogy mi is volt a gond. Azonban ha egy program megfelelően fut parancssorból, akkor az még nem jelenti azt, hogy akkor is futni fog, ha a webkiszolgálóról hívják meg.
Ha webkiszolgálóról indítjuk a CGI programot, akkor hiba esetén mindig a semmitmondó „Internal Server Error” hibaüzenetet kapjuk. Ekkor a következőket nézzük át még egyszer:

- a Content-type és a törzs között ki kell hagyni egy üres sort
- a kiszolgálónak olvasási és végrehajtási jogra van szüksége a szkripthez
- a kiszolgálónak a szkriptet tartalmazó könyvtárba be kell tudnia lépni (végrehajtási jog)
- a szkriptet a megfelelő könyvtárba kell telepíteni, s erre a könyvtárra engedélyezni kell a CGI programok futtatását (lásd 2. sz. Melléklet)
- lehet, hogy a szkript file-nevének egy konkrét utótagra, például *.cgi*-re vagy *.pl*-re kell végződnie
- a szkriptet mindig a `perl -w` kapcsolóval futtassuk, ez ui. pontosabb, bővebb üzeneteket fog eredményezni hiba esetén

3.7. A CGI biztonsága

A CGI-programozás esetén nagyon lényegesek még a biztonsági kérdések. A legfontosabb szabály: kerüljük az olyan CGI kódot, amely a felhasználótól kapott paraméterértékeket közvetlenül átadná külsőleg hívott rendszerparancsoknak. Ha nem tudunk más megoldást alkalmazni, akkor végezzünk a sztringen egy előzetes vizsgálatot, hogy kiszűrjük azon karaktereket, amelyeket a shell speciális módon értelmezne.

Lehetőség van arra is, hogy a CGI-programokba magunk is beépítsünk bizonyos fokú hozzáférési kontrollt. Ez különböző információkra támaszkodhat, mint pl. a kliens gép IP címe (a *remote_addr* környezeti változó tartalmazza ennek az értékét), vagy a távoli felhasználó azonosítója (a *remote_user* környezeti változóban, ha elérhető). A CGI-program ezek alapján eldöntheti, hogy bizonyos adatokhoz, funkciókhoz engedélyezzen-e hozzáférést. Azonban nem lehet 100%-osan megbízni a klienstől érkezett információkban, mivel ezeket viszonylag könnyű hamisítani.

4. Esettanulmány

4.1. A feladat rövid ismertetése

A következőkben egy webes fórumot fogunk elkészíteni. A fórum neve „Party Service” lesz, mivel különböző rendezvényekről („bulikról”) fog információval szolgálni. Egy-egy beküldött hír élettartama beállítható, így az elavult, többé már nem aktuális üzenetek nem fognak megjelenni.

4.2. Előkészületek

4.2.1. Futtató környezet

A feladatot JSP (Java Server Pages, lásd 1.4. fejezet) segítségével fogjuk megoldani. JSP-t legegyszerűbben a Tomcat nevű szoftverrel tudunk futtatni, ennek telepítési helyét jelöljük TOMCAT-tel. Tegyük fel, hogy van egy *jabba* nevű felhasználónk, HOME könyvtára: */home/jabba*. A Tomcat két konfigurációs file-lal is rendelkezik (server.xml, web.xml), ezek a *$TOMCAT/conf* könyvtárban vannak. A server.xml file-ban két portszámot is meg kell adni (pl. 8008, 8009), s ezek közül a böngészőbe beírandó hivatkozásoknál majd az elsőt kell feltüntetni. A „Party Service” projekthez tartozó **összes** file-t egy helyen, a *$TOMCAT/webapps/party* könyvtáron belül helyezzük el.

4.2.2. Adatbázis létrehozása

A feladathoz szükséges adatbázis kezelését PostgreSQL-lel fogjuk megoldani. Ahhoz, hogy ezt Java-ból is el tudjuk érni, szükség lesz a postgresql-jdbc csomagra is. Gondoskodjunk róla, hogy az ebben található `.jar` file bekerüljön a CLASSPATH-ba. Ha a Java-t pl. az */usr/java/jdk1.2.2* könyvtárba telepítettük, akkor egyszerűen csak másoljuk be az előbbi `.jar` file-t az innen nyíló *jre/lib/ext* könyvtárba, ui. az itt található `.jar` file-ok a javac meghívásakor automatikusan látszódni fognak.
Az alkalmazásunkhoz két user-re lesz szükség: egy „mezei” felhasználóra, ill. egy adminisztrátorra. A felhasználó adatokat kérhet le, új üzeneteket vihet fel; míg az

adminisztrátor feladata a fórum karbantartása: egy webes felületen ténylegesen törölheti az adatbázisból az elavult, vagy az oda nem illő híreket.

Első lépésben tehát hozzuk létre ezt a két user-t `party_user` ill. `party_dba` néven, majd állítsuk be a party_user jelszavát party_user –re, a party_dba –nak pedig válasszunk valamilyen komolyabb jelszót (lásd 6. sz. Melléklet). Ezután hozzuk létre az adatbázist:

```
\connect - party_dba
drop table users;
drop table message;
drop sequence message_id_seq;
drop database party;
create database party;
\connect party

create table users(name varchar(10) primary key,
                  password varchar(20) not null,
                  e_mail varchar(20) not null
                 );

create table message(id serial primary key,
                     name varchar(10) references users,
                     sent timestamp,
                     exp  date,
                     body varchar(4096)
                    );

REVOKE ALL on users,message from PUBLIC;
grant select,insert on users,message to party_user;
grant update        on users         to party_user;
```

Ezt érdemes egy file-ba elmenteni, majd miután beléptünk a PostgreSQL-be (psql utasítással), a `\i file_neve` paranccsal tudjuk futtatni. A drop utasítások miatt az első futtatás során még hibát jelez, de ez nem lényeges.

A party nevű adatbázisunk két táblát tartalmaz: felhasználók (nem lehet két azonos nevű felhasználó, így a név lesz az elsődleges kulcs), ill. üzenetek (minden üzenet kap egy egyedi azonosítót, ami mindig 1-gyel nagyobb, mint az utolsó üzenet azonosítója. Ez lesz az elsődleges kulcs. Egy üzenetnek mindig egy felhasználóhoz kell tartoznia, így a felhasználó neve itt külső kulcs lesz). A táblák szerkezete: users – név, jelszó, e-mail; message – azonosító, küldő neve, beküldés ideje, mikor jár le, maga az üzenet szövege.

Az adatbázist a party_dba user hozta létre, így ő minden joggal rendelkezik. Az adatbázishoz rajta kívül senki más nem férhet hozzá, csak a party_user.

4.2.3. Mire lesz még szükség

Üzenetet csak regisztrált felhasználók küldhetnek. A rendszer az új felhasználó által megadott e-mail címre elküld egy köszöntő levelet. Ahhoz, hogy Java-ból levelet tudjunk küldeni, a következő csomagokra lesz még szükségünk:

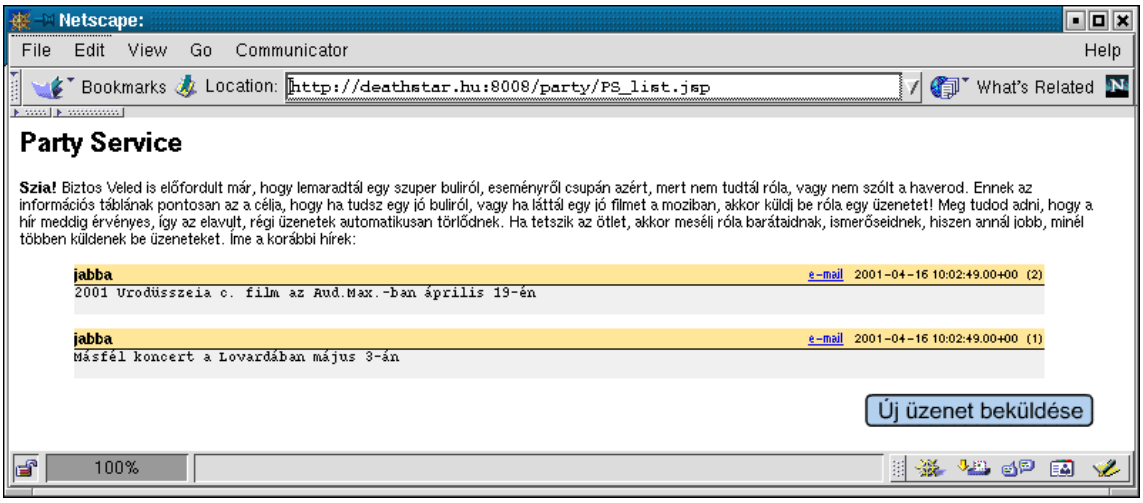
- javamail
- jaf (Java Activation Framework)

Mindkettő letölthető a java.sun.com címről. Az ezekben található `activation.jar`, ill. `mail.jar` file-okat szintén tegyük be a CLASSPATH-ba.

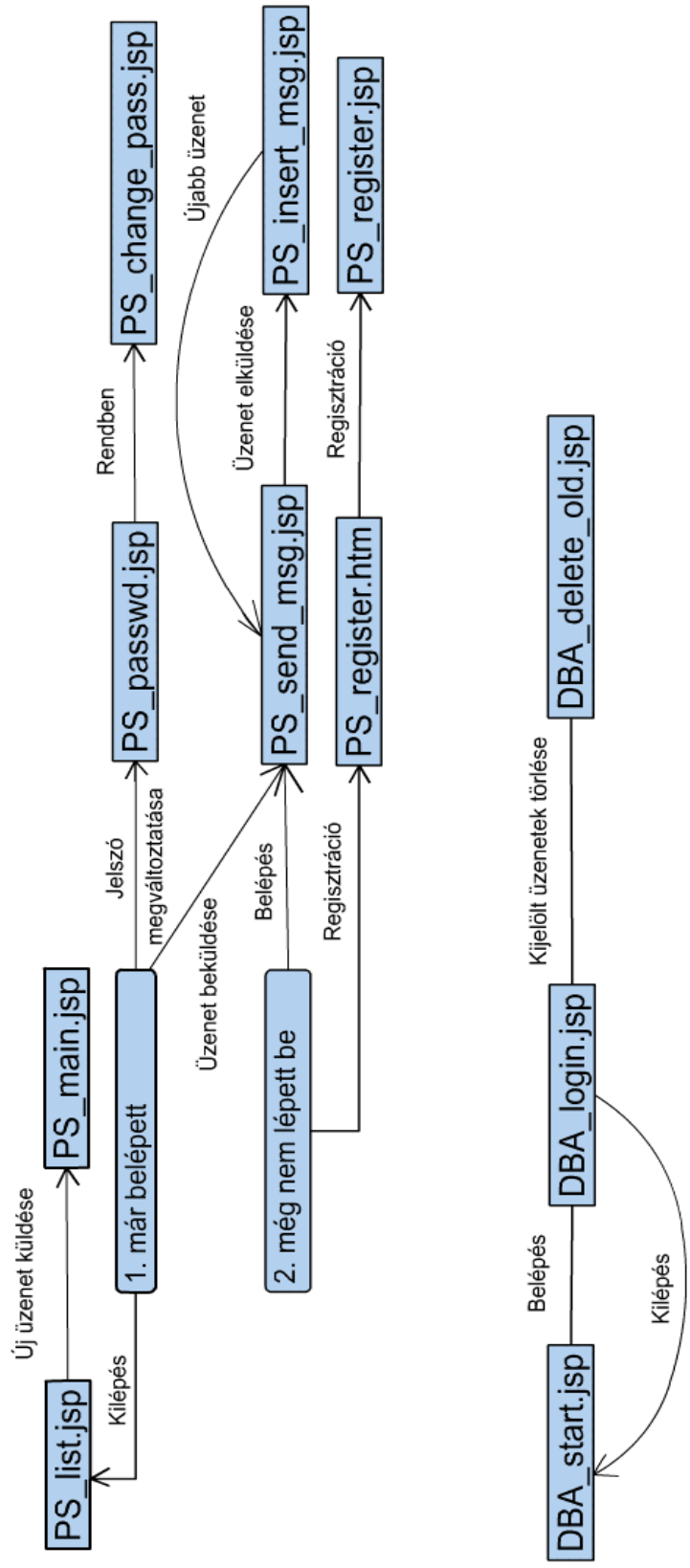
4.3. Megvalósítás

A „Party Service” webes alkalmazáshoz szükséges file-ok közötti kapcsolatokat a következő oldalon lévő ábra szemlélteti. (Az egyes oldalakról a kezdőlapra visszamutató kapcsolatokat nem jelöltem külön). Az ábra két részből tevődik össze: a felső rész az egyes felhasználók lehetőségeit mutatja, míg az alsó rész az adminisztrátor által elérhető oldalakat ábrázolja. Vegyük sorba őket:

PS_list.jsp



Az egyes felhasználók számára ez az oldal jelenti a belépési pontot. Aki csak a híreket akarja elolvasni, az itt meg is állhat. A további lehetőségek eléréséhez (üzenet küldése, új felhasználó regisztrálása, jelszó megváltoztatása) tovább kell lépni.



Az oldalak közötti kapcsolatok

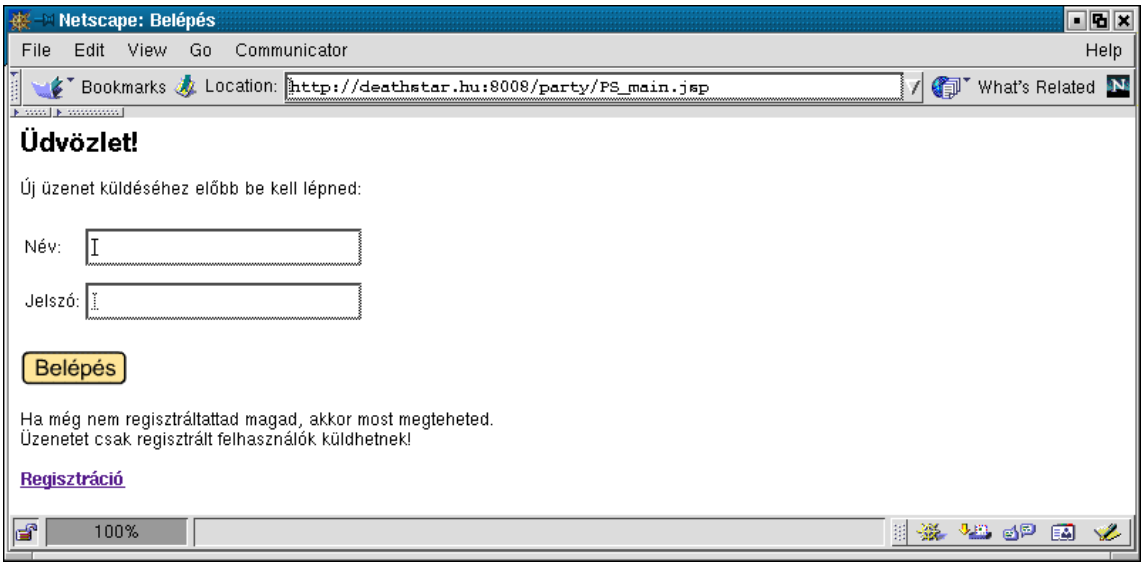
A PS_list.jsp forrása:

```
<%@ page import="java.sql.*" %>
<html><body bgcolor="#ffffff">
<h1>Party Service</h1>
<font size=-1><b>Szia!</b> Biztos Veled is előfordult már, hogy lemaradtál egy szuper buliról,
eseményről csupán azért, mert nem tudtál róla, vagy nem szólt a haverod. Ennek az információs
táblának pontosan az a célja, hogy ha tudsz egy jó buliról, vagy ha láttál egy jó filmet a
moziban, akkor küldj be róla egy üzenetet! Meg tudod adni, hogy a hír meddig érvényes, így az
elavult, régi üzenetek automatikusan törlődnek. Ha tetszik az ötlet, akkor mesélj róla
barátaidnak, ismerőseidnek, hiszen annál jobb, minél többen küldenek be üzeneteket. Íme a
korábbi hírek:</font><p>
<%
   boolean talalt = false;

   try
   {
      Class.forName("org.postgresql.Driver");
      Connection conn = DriverManager.getConnection("jdbc:postgresql://127.0.0.1/"+"party",
                                                    "party_user","party_user");
      Statement  st   = conn.createStatement();
      ResultSet rs = st.executeQuery("select M.id, M.name, M.sent, M.body, U.e_mail "+
                                     "from message M, users U where M.name=U.name and "+
                                     "current_date<M.exp order by M.id desc");
      if (rs!=null)
      {
         while (rs.next())
         {
%>
<TABLE width=90% bgcolor=FFFFFF border=0 cellpadding=0 cellspacing=0 align=center>
<TR>
<TD bgcolor=ffe699 align=left>
 <FONT size=-1 face=Arial, Helvetica><B><%= rs.getString(2) %></B></FONT>
</TD>
<TD bgcolor=ffe699 align=right>
 <FONT SIZE=-2 face=Arial, Helvetica>
 <A HREF="mailto:<%= rs.getString(5) %>">e-mail</A>
    <%= rs.getString(3) %>   (<%= rs.getString(1) %>)
 </FONT>
</TD>
</TR>
<TR><TD colspan=2 height=1 bgcolor=000000 valign=top>
<SPACER type=block width=1 height=1></td></tr>
<TR><TD bgcolor=f0f0f0 colspan=2 valign=top><FONT SIZE=-1 FACE=Arial, Helvetica>
<pre><%= rs.getString(4) %></pre>
</TR>
</TABLE>
<FONT size=0><BR></FONT>
<%
            talalt = true;
         }
      }
      if (!talalt) %>Sajnos még nincs hír. Küldj be valamit!<br><%
      rs.close();
      st.close();
      conn.close();
   }
   catch (java.sql.SQLException sqle) {
      out.println("<b>SQL hiba:</b> "+sqle.toString());
   }
   catch (ClassNotFoundException cnfe) {
      out.println(cnfe.toString());
   }
%>
<a href="PS_main.jsp"><img src="assets/uj_uzenet_bekuldese2.gif" border="0" align="right"></a>
</body></html>
```

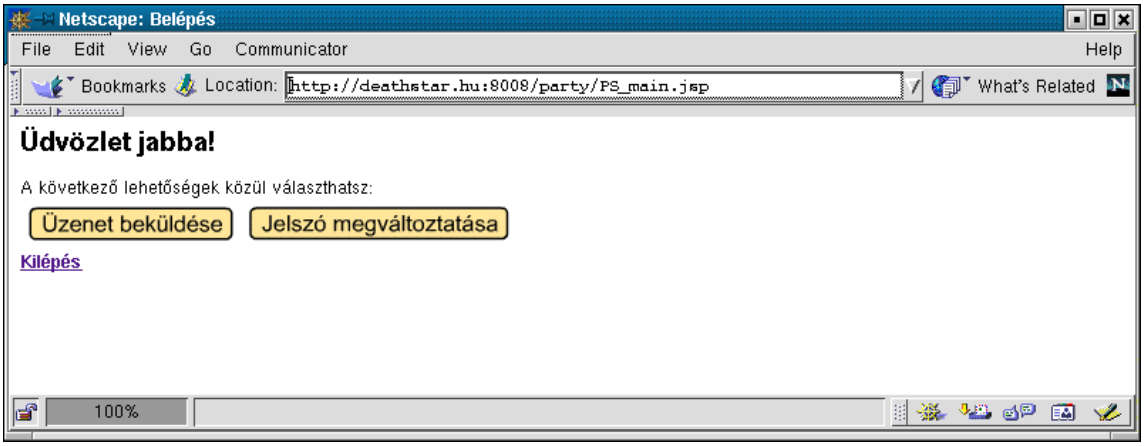
Az oldal elején kiírásra kerül egy tájékoztató jellegű szöveg. Ezután kapcsolódunk az adatbázishoz, s lekérjük a szükséges adatokat. Az üzeneteket úgy jelenítjük meg, hogy a legfrissebb hír kerüljön legelölre. A képeket egységesen egy *assets* nevű könyvtárban helyezzük el.

PS_main.jsp

Ha eddig még nem lépett be a felhasználó:



Ha már belépett korábban:



A PS_main.jsp két különböző output-ot produkál azalapján, hogy a felhasználó belépett-e már, vagy még nem. Ezt a klienskapcsolati környezet (session) alapján tudjuk megállapítani. A session segítségével bizonyos információkat (jelen esetben: felhasználó neve, jelszava) át tudunk adni az oldalak között, így ha valaki belépett, akkor a PS_main.jsp érzékeli, hogy a session-ben már ott vannak a felhasználó adatai, így nem kell még egyszer belépnie. Belépéskor mindkét mezőt ki kell tölteni; ezt érdemes már a kliensoldalon is ellenőrizni egy kis JavaScript rutinnal.

```
<html>
<head>
   <title>Belépés</title>
<SCRIPT LANGUAGE="JavaScript">

function check1(form)
{
   if (form.name.value=="" || form.password.value=="")
      alert("Hiba: mindkét mezőt ki kell tölteni!");
   else form.submit();
}

function check2(form)
{
   if (form.name.value=="" || form.e_mail.value=="")
      alert("Hiba: mindkét mezőt ki kell tölteni!");
   else form.submit();
}

</SCRIPT>
</head>
<body bgcolor="#ffffff">
<%
   String name, password; name=password=null;
   boolean registrated;

   try {
      name     = (String)session.getValue("name_in_session");
      password = (String)session.getValue("password_in_session");
   } catch (NullPointerException npe) {}
   registrated = (name!=null && password!=null);

   if (!registrated)
   {
%>
<h1>Üdvözlet!</h1>
Új üzenet küldéséhez előbb be kell lépned:
<form action="PS_send_msg.jsp" name="send_msg" method="POST">
<table border="0">
<tr><td>Név:<td><input type="text" name="name"><br>
<tr><td>Jelszó:<td><input type="password" name="password">
</table><br>
<a href="javascript:check1(document.send_msg)"><img src="assets/belepes.gif" border="0"></a>
</form>
Ha még nem regisztráltattad magad, akkor most megteheted.<br>
Üzenetet csak regisztrált felhasználók küldhetnek!<br><br>
<a href="PS_register.htm"><b>Regisztráció</b></a>
<%
   }
   else   //if (registrated)
   {
%>
<h1>Üdvözlet <%= name %>!</h1>
A következő lehetőségek közül választhatsz:
<table>
<tr><td><a href="PS_send_msg.jsp"><img src="assets/uzenet_bekuldese.gif" border="0"></a>
    <td><a href="PS_passwd.jsp"><img src="assets/jelszo_megvaltoztatasa.gif" border="0"></a>
</table>
<a href="PS_logout.jsp"><b>Kilépés</b></a>
<%
   }
%>
</body>
</html>
```

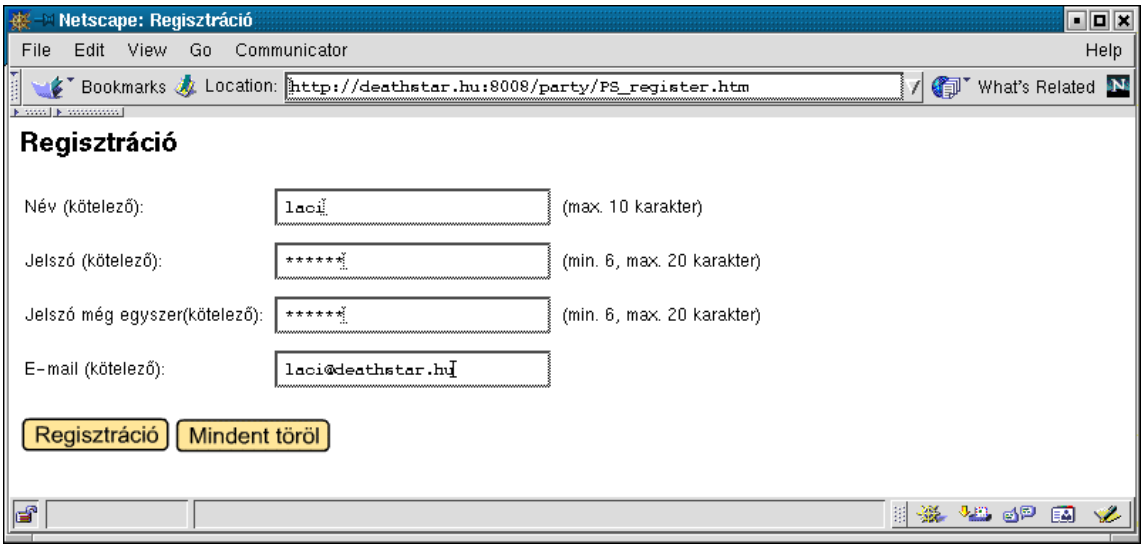
Megpróbáljuk kiolvasni a felhasználó nevét és jelszavát a session-ből. Ha nem sikerült, akkor még nem lépett be, így kitesszük a bejelentkezési képernyőt. Ellenkező esetben a korábban már belépett felhasználó a következők közül választhat: üzenet beküldése, jelszó megváltoztatása vagy kilépés.

PS_logout.jsp

Ha a felhasználó a „Kilépés” gombra kattint, akkor a rendszer megszünteti a hozzárendelt session-t, s átirányítja a kezdőoldalra:

```
<%
   session.invalidate();
   response.sendRedirect("PS_list.jsp");
%>
```

PS_register.htm

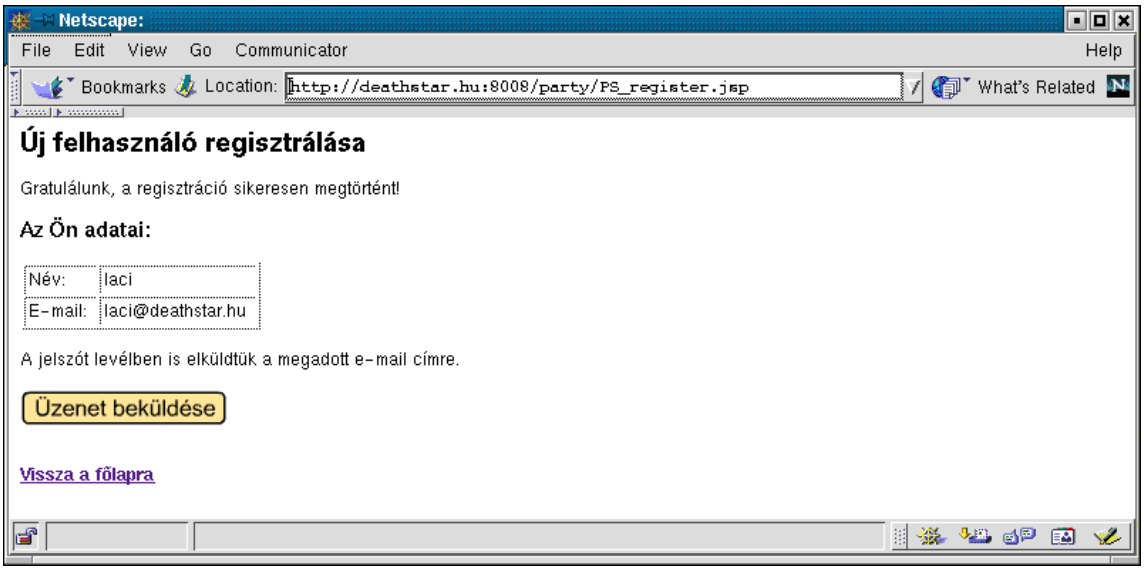


Ha még nem regisztáltuk magunkat, akkor azt itt tehetjük meg. Meg kell adni a felhasználói nevet, a jelszót kétszer, ill. az e-mail címet. Itt is JavaScript-ekkel ellenőrizzük a beírt adatokat.

```
<html><head><title>Regisztráció</title>
<SCRIPT LANGUAGE="JavaScript">

function check(form)
{
   var error = false;

   if (form.name.value=="" ||
       form.new_pass.value=="" ||
       form.new_pass.value!=form.new_pass2.value ||
       form.e_mail.value=="") {
  error = true;
  alert("Hiba: hiányos mező, vagy az új jelszó megerősítése nem egyezik az elsőre beírttal!");
   }
   if (error==false) {
      if (form.new_pass.value.length<6 || form.new_pass.value.length>20)
         { error = true; alert("Hiba: a jelszó hossza min. 6, max. 20 karakter lehet!"); }
   }
   if (error==false) {
      if (form.name.value.length>10)
      { error = true; alert("Hiba: a név max. 10 karakter hosszú lehet!"); }
   }
   if (error==false)  form.submit();
}
```

```
function reset(form) { form.reset(); }

</SCRIPT>
</head>
<body bgcolor="#ffffff">
<h1>Regisztráció</h1>
<form action="PS_register.jsp" name="register" method="POST">
<table border="0">
<tr><td>Név (kötelező): <td><input type="text" name="name"><td>(max. 10 karakter)
<tr><td>Jelszó (kötelező): <td><input type="password" name="new_pass">
    <td>(min. 6, max. 20 karakter)
<tr><td>Jelszó még egyszer(kötelező): <td><input type="password" name="new_pass2">
    <td>(min. 6, max. 20 karakter)
<tr><td>E-mail (kötelező): <td><input type="text" name="e_mail">
</table><br>
<a href="javascript:check(document.register)"><img src="assets/regisztracio.gif"
border="0"></a>
<a href="javascript:reset(document.register)"><img src="assets/mindent_torol.gif" border="0">
</a>
</form>
</body>
</html>
```

PS_register.jsp



Ha a felhasználó sikeresen megadta az adatait, s ilyen nevű felhasználó még nem létezett, akkor bekerül az adatbázisba. Az általános tapasztalat azt mutatja, hogy az emberek jó pár site-on regisztráltatják magukat, s nagyon hamar el is felejtik, hogy hova / milyen néven / milyen jelszóval léptek be. Ezért a megadott e-mail címre küldünk egy levelet, amely tartalmazza mindezen információkat.

PS_register.jsp:

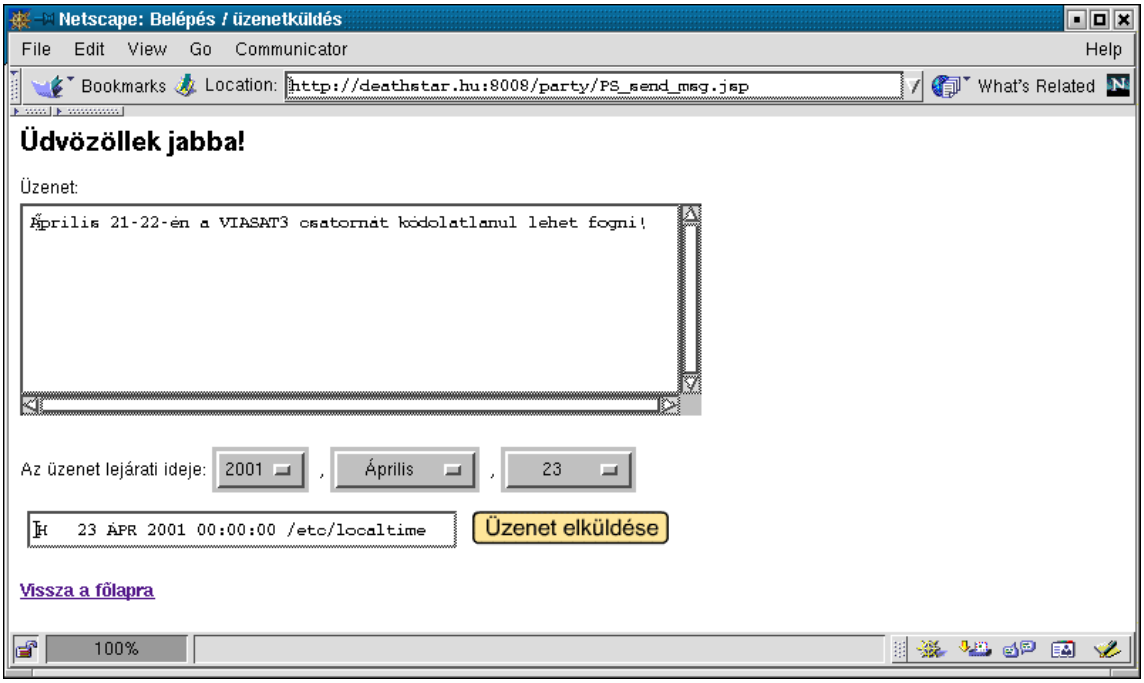
```
<%@ page import="java.sql.*" %>
<html><body bgcolor="#ffffff"><h1>Új felhasználó regisztrálása</h1>
<%
   boolean ok = true;
   boolean talalt = false;
   String name,password,e_mail; name=password=e_mail=null;

   try { name      = request.getParameterValues("name")[0];
      password = request.getParameterValues("new_pass")[0];
      e_mail    = request.getParameterValues("e_mail")[0];
   } catch (NullPointerException npe) {
      ok = false;
      %><b>Hiba: </b>minden mezőt ki kell tölteni!<br></html></body>
      <%
   }

   if (ok)
   {
      try
      {
         Class.forName("org.postgresql.Driver");
         Connection conn = DriverManager.getConnection("jdbc:postgresql://127.0.0.1/"+"party",
                                                       "party_user","party_user");

         Statement  st   = conn.createStatement();
         ResultSet rs = st.executeQuery("select name from users where name='"+name+"'");
         if (rs!=null)
         {
            while (rs.next())
            {
               talalt = true;
            }
         }
         if (talalt)
         {
            ok = false;
            %>Hiba: sajnos már van ilyen nevű felhasználó. Próbálj egy másik nevet!<br><%
         }
         else
         {
            %>
            <%
            String insert ="insert into users values('"+name+"','"+password+"','"+e_mail+"')";
            st.executeUpdate(insert);
         }
      }
      catch (java.sql.SQLException sqle) { %>
         <b>SQL hiba: </b><%= sqle.toString() %>
         <% ok = false;
      }
      catch (ClassNotFoundException cnfe) { %>
         <%= cnfe.toString() %>
         <% ok = false;
      }

      if (ok)
      {
         %>Gratulálunk, a regisztráció sikeresen megtörtént!<p>
         <h3>Az Ön adatai:</h3>
         <table border="1" cellpadding="2">
         <tr><td>Név:    <td><%= name %>
         <tr><td>E-mail: <td><%= e_mail %>
         </table><br>
         <%
            party.SendMail mail = new party.SendMail();
            if (mail.send(name,password,e_mail))
               %>A jelszót levélben is elküldtük a megadott e-mail címre.<p>
            <a href="PS_main.jsp"><img src="assets/uzenet_bekuldese.gif" border="0"></a><p><%
         %>
         <%
      }
   }
%>
<br><a href="PS_list.jsp"><b>Vissza a főlapra</b></a></html></body>
```

SendMail.java

```
package party;

import java.io.*;
import java.net.InetAddress;
import java.util.Properties;
import java.util.Date;

import javax.mail.*;
import javax.mail.internet.*;

public class SendMail
{
   public boolean send(String name, String password, String e_mail)
   {
      String  from, to, subject, mailer, body;

      from = "szathml@delfin.klte.hu";
      to = e_mail;
      subject = "PartyService - regisztracio";
      mailer = "PartyService";
      body = "Kedves "+name+"!\n"+
             "\n"+
             "Koszonjuk, hogy regisztraltad magadat nalunk.\n"+
             "Remeljuk, meg leszel elegedve szolgaltatasainkkal.\n"+
             "\n"+
             "Jelszavad: \""+password+"\" (idezojelek nelkul)\n"+
             "\n"+
             "\tUdvozlettel:\n"+
             "\n"+
             "\t\ta Party Service csapat";

      try
      {
         Properties props = System.getProperties();
         Session session = Session.getDefaultInstance(props, null);

         Message msg = new MimeMessage(session);
         msg.setFrom(new InternetAddress(from));
         msg.setRecipients(Message.RecipientType.TO,
               InternetAddress.parse(to, false));
         msg.setSubject(subject);
         msg.setText(body);
         msg.setHeader("X-Mailer", mailer);
         msg.setSentDate(new Date());

         Transport.send(msg);
      } catch (Exception e)
      {
         //e.printStackTrace();
         return false;
      }
      return true;
   }
}
```

Ezt a forrást a .jsp file-okat tartalmazó könyvtárból nyíló *WEB-INF/classes/party/* könyvtárban helyezzük el. Figyelem! Ezt a file-t manuálisan kell lefordítani (javac)! Míg a Tomcat a `.jsp` file-okat automatikusan lefordítja, addig az ilyen típusú file-ok lefordításáról nekünk kell gondoskodni. A file sikeres fordításához be kell szerezni a JavaMail, ill. a Java Activation Framework csomagot (lásd 4.2.3-as fejezet).

PS_send_msg.jsp



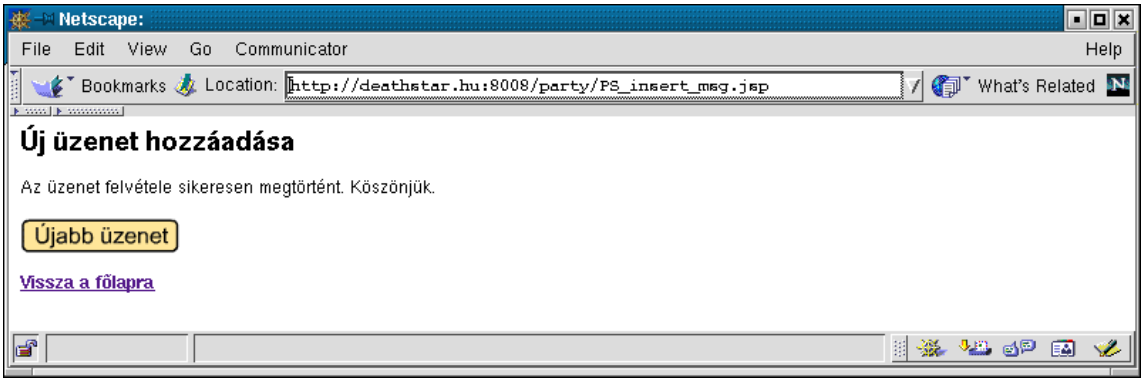
Ha a PS_main.jsp oldalon helyes adatokat adtunk meg, akkor belépünk a rendszerbe. Ez az oldal már be is rakja a klienskapcsolati környezetbe (session) a felhasználó nevét, jelszavát, így ez az információ mindaddig „utazni” fog az oldalak között, míg explicit módon meg nem szüntetjük ezt a kapcsolatot (vagy amíg le nem jár az érvényességi ideje). Itt lehet megadni az üzenet „élettartamát” is, s ha ez lejár, akkor az üzenet már nem fog megjelenni a főoldalon. Itt is szükség lesz néhány JavaScript-re, melyekkel azt ellenőrizzük, hogy lejárati időnek csak az aktuálisnál későbbi dátumot lehessen megadni.

```
<%@ page import="java.sql.*" %>
<html>
<head>
   <title>Belépés / üzenetküldés</title>
<SCRIPT LANGUAGE="JavaScript">

function calcDate(form)
{
   var y = parseInt(form.Ev.options[form.Ev.selectedIndex].value)
   var m = parseInt(form.Ho.options[form.Ho.selectedIndex].value)
   var d = parseInt(form.Nap.options[form.Nap.selectedIndex].value)
   expDate = new Date(y,m,d);
   form.helyi_ido.value = expDate.toLocaleString()
}

function check(form)
{
   var newDate = new Date();

   if (expDate < newDate || form.message.value=="")
      alert("Hiba: a lejárati idő már lejárt, vagy nem írtál be semmit.");
   else form.submit();
}
```

```
</SCRIPT>
</head>
<BODY BGCOLOR="#FFFFFF" onLoad="calcDate(document.form1)">
<%
   boolean talalt = false;

   try
   {
      Class.forName("org.postgresql.Driver");
      Connection conn = DriverManager.getConnection("jdbc:postgresql://127.0.0.1/"+"party",
                                                    "party_user","party_user");
      Statement  st   = conn.createStatement();
      String name = null;
      String password = null;
      try {
         name = request.getParameterValues("name")[0];
         password = request.getParameterValues("password")[0];
      } catch (NullPointerException npe) {}
      if (name==null || password==null)
      try {
         name       = (String)session.getValue("name_in_session");
         password = (String)session.getValue("password_in_session");
      } catch (NullPointerException npe) {}

      ResultSet rs = st.executeQuery("select name from users "+
                                     "where name='"+name+"' and password='"+password+"'");
      if (rs!=null)
      {
         while (rs.next())
         {
            talalt = true;
            if (talalt) break;
         }
      }
      if (!talalt) %><b>Hiba! </b>Ennek két oka lehet: hibás név és/vagy jelszó, vagy
         közvetlen hozzáférés megkísérlése.<br>
         <b>Tipp: </b>Ellenőrizd le még egyszer a beírt adatokat; vagy közvetett módon, a név
         és jelszó megadása után próbálkozz!<br>
         <form action="PS_main.jsp"><input type="submit" value="Vissza"></form>
         <%
      else
      {
          session.putValue("name_in_session",name);
          session.putValue("password_in_session",password);
      %>
         <h1>Üdvözöllek <%= name %>!</h1>
         <form name="form1" action="PS_insert_msg.jsp" method="POST">
         Üzenet:<font size="0"><br></font>
         <textarea name="message" rows="10" cols="50"></textarea><br>
         <br>
         <P>Az üzenet lejárati ideje:
         <SELECT NAME="Ev"  onChange='calcDate(this.form)'>
            <OPTION SELECTED VALUE=2001>2001<OPTION VALUE=2002>2002<OPTION VALUE=2003>2003
            <OPTION VALUE=2004>2004<OPTION VALUE=2005>2005<OPTION VALUE=2006>2006
            <OPTION VALUE=2007>2007<OPTION VALUE=2008>2008<OPTION VALUE=2009>2009
            <OPTION VALUE=2010>2010</SELECT>,
         <SELECT NAME="Ho"  onChange="calcDate(this.form)"><OPTION SELECTED VALUE=0>Január
            <OPTION VALUE=1>Február<OPTION VALUE=2>Március<OPTION VALUE=3>Április
            <OPTION VALUE=4>Május<OPTION VALUE=5>Június<OPTION VALUE=6>Július
            <OPTION VALUE=7>Augusztus<OPTION VALUE=8>Szeptember<OPTION VALUE=9>Október
            <OPTION VALUE=10>November<OPTION VALUE=11>December,</SELECT>,
         <SELECT NAME="Nap" onChange="calcDate(this.form)"><OPTION SELECTED VALUE=1>1
            <OPTION VALUE=2>2<OPTION VALUE=3>3<OPTION VALUE=4>4<OPTION VALUE=5>5
            <OPTION VALUE=6>6<OPTION VALUE=7>7<OPTION VALUE=8>8<OPTION VALUE=9>9
            <OPTION VALUE=10>10<OPTION VALUE=11>11<OPTION VALUE=12>12<OPTION VALUE=13>13
            <OPTION VALUE=14>14<OPTION VALUE=15>15<OPTION VALUE=16>16<OPTION VALUE=17>17
            <OPTION VALUE=18>18<OPTION VALUE=19>19<OPTION VALUE=20>20<OPTION VALUE=21>21
            <OPTION VALUE=22>22<OPTION VALUE=23>23<OPTION VALUE=24>24<OPTION VALUE=25>25
            <OPTION VALUE=26>26<OPTION VALUE=27>27<OPTION VALUE=28>28<OPTION VALUE=29>29
            <OPTION VALUE=30>30<OPTION VALUE=31>31</SELECT><br>
         <table>
         <tr><td><INPUT TYPE=text NAME="helyi_ido" SIZE=32>
             <td><a href="javascript:check(document.form1)">
<img src="assets/uzenet_elkuldese.gif" border="0"></a>
         </table>
         </form>
```

```
         <a href="PS_list.jsp"><b>Vissza a főlapra</b></a>
      <%
      }
      rs.close();
      st.close();
      conn.close();
   }
   catch (java.sql.SQLException sqle) {
      out.println("<b>SQL hiba:</b> "+sqle.toString());
   }
   catch (ClassNotFoundException cnfe) {
      out.println(cnfe.toString());
   }
%>
</body>
</html>
```

PS_insert_msg.jsp



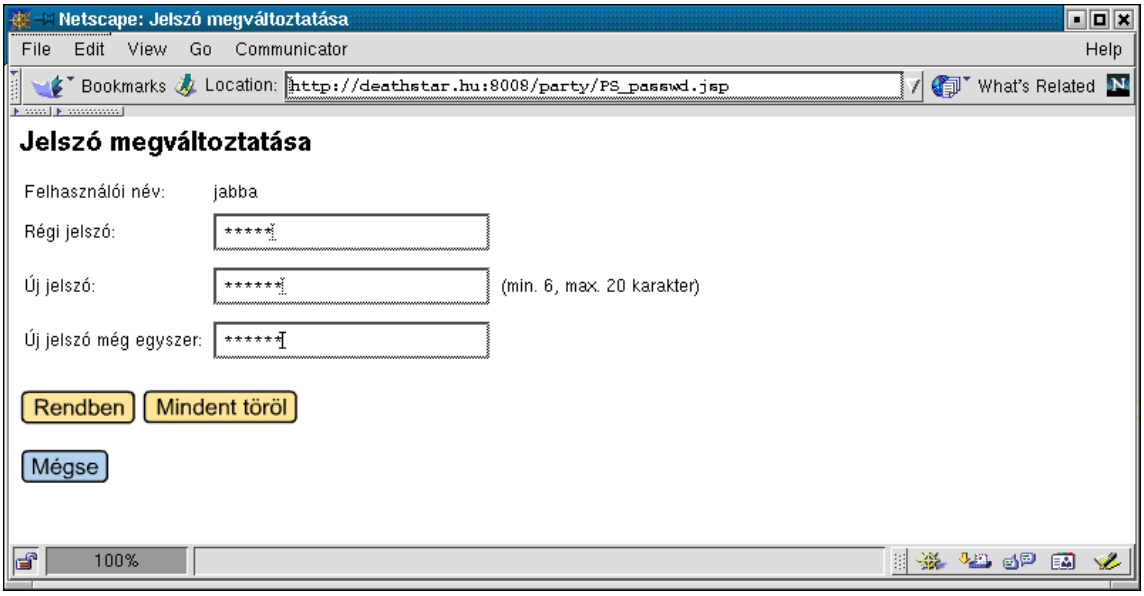
Ez az oldal fogja beszúrni a *message* táblába a megadott üzenetet.

```
<%@ page import="java.sql.*" %>
<html><body bgcolor="#ffffff">
<h1>Új üzenet hozzáadása</h1>
<%
   boolean ok = true;
   String name,exp,body; name=exp=body=null;
   String ev,ho,nap; ev=ho=nap=null;

   try {
      name = (String)session.getValue("name");
      ev   = request.getParameterValues("Ev")[0];
      ho   = request.getParameterValues("Ho")[0];
         ho = Integer.toString(Integer.parseInt(ho)+1);
      nap  = request.getParameterValues("Nap")[0];
      exp  = ev+"-"+ho+"-"+nap;
      body = request.getParameterValues("message")[0];
   } catch (NullPointerException npe) {}
     catch (NumberFormatException nfe) {}
   if (name==null)
   try {
      name = (String)session.getValue("name_in_session");
   } catch (NullPointerException npe) {}

   if (name == null)
   {
      %><b>Hiba:</b> erre az oldalra csak közvetett módon lehet eljutni!<br>
      </body></html>
      <%
   }
   else
```

```
   {
      try
      {
         Class.forName("org.postgresql.Driver");
         Connection conn = DriverManager.getConnection("jdbc:postgresql://127.0.0.1/"+"party",
                                                       "party_user","party_user");

         Statement  st   = conn.createStatement();
         String insert ="insert into message(name,sent,exp,body) "+
                        "values('"+name+"',current_timestamp,'"+exp+"','"+body+"')";
         st.executeUpdate(insert);
      }
      catch (java.sql.SQLException sqle) { %>
         <b>SQL hiba: </b><%= sqle.toString() %>
         <% ok = false;
      }
      catch (ClassNotFoundException cnfe) { %>
         <%= cnfe.toString() %>
         <% ok = false;
      }

      if (ok)
      {
         %>Az üzenet felvétele sikeresen megtörtént. Köszönjük.<p>
         <a href="PS_send_msg.jsp"><img src="assets/ujabb_uzenet.gif" border="0"></a><p>
         <a href="PS_list.jsp"><b>Vissza a főlapra</b></a>
         <%
      }
      %></html></body>
<%    }
%>
```

PS_passwd.jsp

Ha már sikeresen bejelentkeztünk, akkor a PS_main.jsp oldalon lehetőségünk van meghívni ezt az oldalt, ahol meg tudjuk változtatni a jelenlegi jelszavunkat:

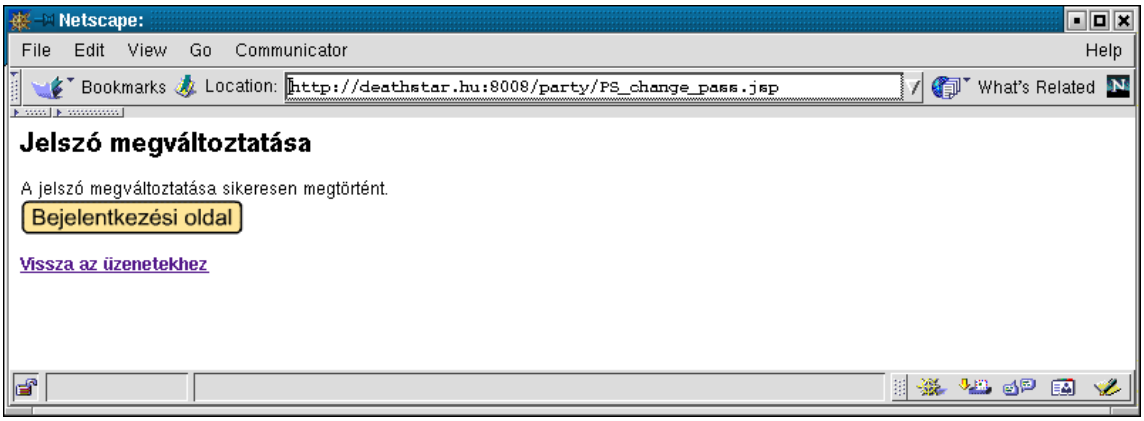


Itt is JavaScript ellenőrzi, hogy az új jelszó hossza megfelel-e a kívánalmaknak, ill. a beírt jelszó egyezik-e a jelszó megerősítésével.

```
<html>
<head>
   <title>Jelszó megváltoztatása</title>
<SCRIPT LANGUAGE="JavaScript">

function check(form)
{
   var error = false;

   if (form.old_pass.value==""    ||
       form.new_pass.value==""   ||
       form.new_pass.value!=form.new_pass2.value) {
   error = true;
   alert("Hiba: hiányos mező, vagy az új jelszó megerősítése nem egyezik az elsőre
beírttal!");
   }
   if (error==false) {
      if (form.new_pass.value.length<6 || form.new_pass.value.length>20)
      { error = true; alert("Hiba: az új jelszó hossza min. 6, max. 20 karakter lehet!"); }
   }
   if (error==false)  form.submit();
}

function reset(form) { form.reset(); }

</SCRIPT>
</head>
<body bgcolor="#ffffff">
<h1>Jelszó megváltoztatása</h1>
<%
   String name=null;
   boolean registrated;

   try {
      name       = (String)session.getValue("name_in_session");
   } catch (NullPointerException npe) {}
   registrated = (name!=null);

   if (!registrated)
   {
      %><b>Hiba: </b>a jelszó megváltoztatásához előbb be kell jelentkezned!<p>
      <a href="PS_main.jsp"><img src="assets/bejelentkezes.gif" border="0"></a>
      <%
   }
   else
   {
%>
<form action="PS_change_pass.jsp" name="change_pass" method="POST">
<table border="0">
<tr><td>Felhasználói név:<td><%= name %>
<tr><td>Régi jelszó:       <td><input type="password" name="old_pass">
<tr><td>Új jelszó:       <td><input type="password" name="new_pass">
    <td>(min. 6, max. 20 karakter)
<tr><td>Új jelszó még egyszer: <td><input type="password" name="new_pass2">
</table><br>
<a href="javascript:check(document.change_pass)"><img src="assets/rendben.gif" border="0"></a>
<a href="javascript:reset(document.change_pass)"><img src="assets/mindent_torol.gif"
border="0">
</a>
</form>
<a href="PS_main.jsp"><img src="assets/megse2.gif" border="0"></a>
<%
   }
%>
</body>
</html>
```

A felhasználónak nem kell megadnia a nevét, ui. ezt a session-ből ki tudjuk venni. Ebből is látszik, hogy ezt az oldalt csak akkor lehet behívni, ha már korábban sikeresen bejelentkeztünk.

PS_change_pass.jsp



Ez az oldal végzi el a *users* tábla frissítését. Mivel megváltozott az adott felhasználó jelszava, s nem szeretnénk, hogy emiatt még egyszer be kelljen lépnie, ezért a session-ben is módosítjuk a jelszavát.

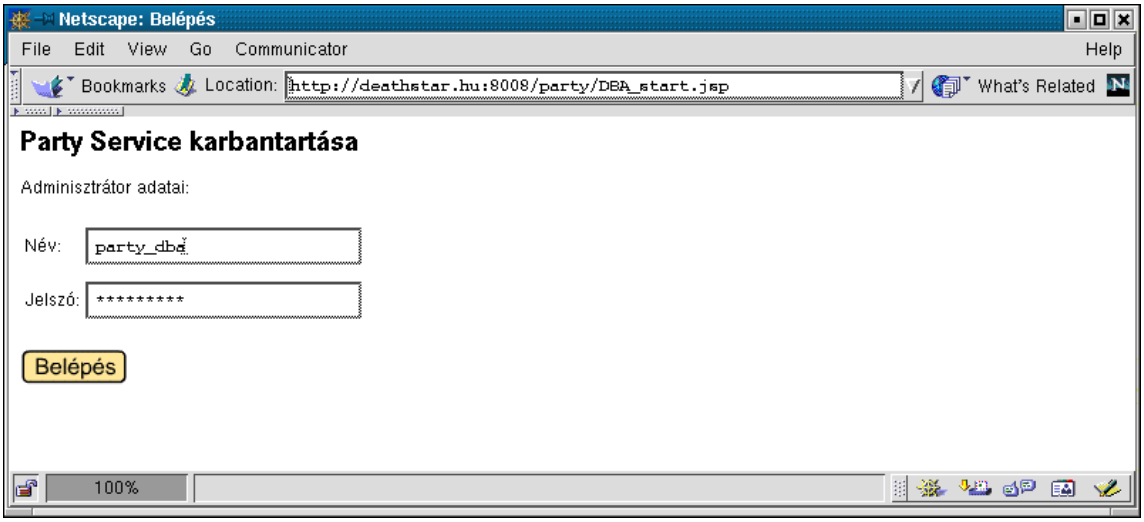
```
<%@ page import="java.sql.*" %>
<html><body bgcolor="#ffffff">
<h1>Jelszó megváltoztatása</h1>
<%
   boolean ok = true;
   String name,old_pass,new_pass,new_pass2; name=old_pass=new_pass=new_pass2=null;

   try {
      name         = (String)session.getValue("name_in_session");
      old_pass      = request.getParameterValues("old_pass")[0];
      new_pass      = request.getParameterValues("new_pass")[0];
      new_pass2    = request.getParameterValues("new_pass2")[0];
   } catch (NullPointerException npe)
   {
      ok = false;
      %><b>Hiba: </b>minden mezőt ki kell tölteni és/vagy előbb be kell jelentkezni!<p>
      <%
   }

   if (ok)
   {
      try
      {
         Class.forName("org.postgresql.Driver");
         Connection conn = DriverManager.getConnection("jdbc:postgresql://127.0.0.1/"+"party",
                                                       "party_user","party_user");

         Statement  st    = conn.createStatement();
         String update = "update users set password='"+new_pass+
                         "' where name='"+name+"' and password='"+old_pass+"'";
         ok = (st.executeUpdate(update) != 0);
         if (ok)
         {
            session.removeValue("password_in_session");
            session.putValue("password_in_session",new_pass);
         }
      }
      catch (java.sql.SQLException sqle) { %>
         <b>SQL hiba: </b><%= sqle.toString() %>
         <% ok = false;
      }
      catch (ClassNotFoundException cnfe) { %>
         <%= cnfe.toString() %>
         <% ok = false;
      }
```

```
      if (ok)
      {
         %>A jelszó megváltoztatása sikeresen megtörtént.<br><%
      }
      else
      {
         %><b>Hiba: </b>nem sikerült megváltoztatni a jelszót.<p>
         <%
      }
%>
<%   }
%>
<a href="PS_main.jsp"><img src="assets/bejelentkezesi_oldal.gif" border="0"></a><p>
<a href="PS_list.jsp"><b>Vissza az üzenetekhez</b></a>
</html></body>
```

Ezzel az oldallal befejeződött az alkalmazás azon része, amelyet egy „átlagos” felhasználó láthat.

DBA_start.jsp

A „Party Service” alkalmazás két részből tevődik össze. Az első részét – melyet a felhasználók látnak, használhatnak – már láttuk.
Most következik a második rész, melyet csak az alkalmazás adminisztrátora használhat – erről a felhasználóknak nem is kell tudniuk.



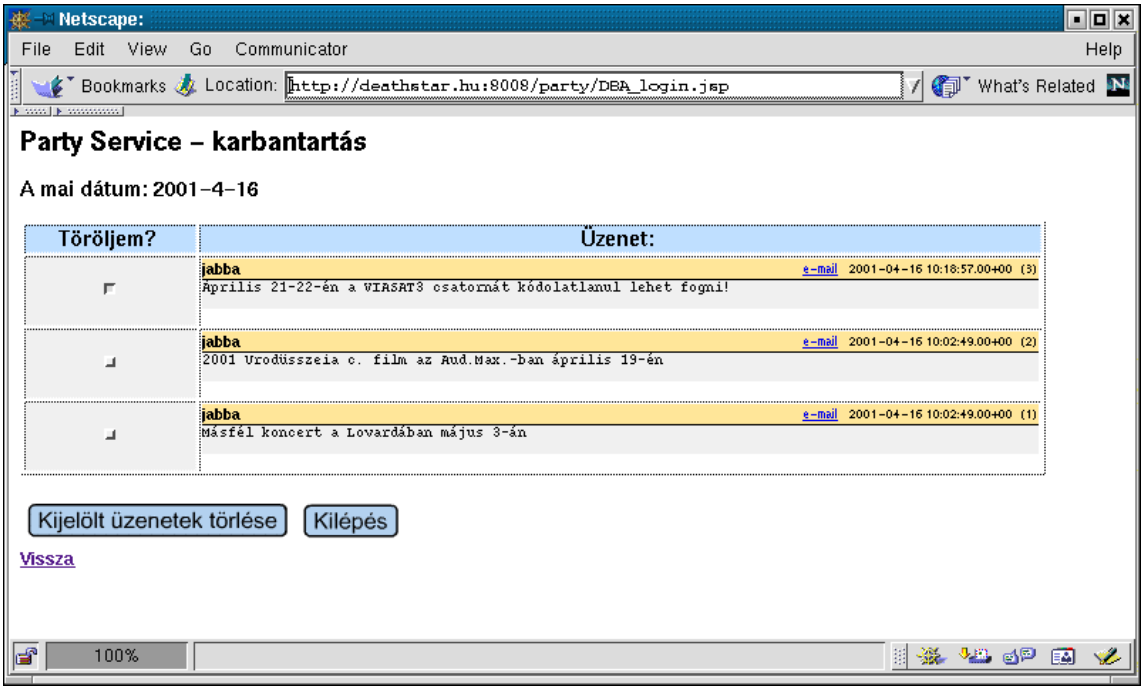
Ez az oldal jelenti az adminisztrátor számára a belépési pontot.

```
<html>
<head>
   <title>Belépés</title>
<SCRIPT LANGUAGE="JavaScript">

function check(form)
{
   if (form.name.value=="" || form.password.value=="")
      alert("Hiba: mindkét mezőt ki kell tölteni!");
   else form.submit();
}
```

```
</SCRIPT>
</head>

<body bgcolor="#ffffff">
<h1>Party Service karbantartása</h1>
Adminisztrátor adatai:
<form action="DBA_login.jsp" name="login" method="POST">
<table border="0">
<tr><td>Név:<td><input type="text" name="name"><br>
<tr><td>Jelszó:<td><input type="password" name="password">
</table><br>
<a href="javascript:check(document.login)"><img src="assets/belepes.gif" border="0"></a>
</form>
</body>
</html>
```

Ez az oldal egyetlen JSP elemet sem tartalmaz, így lehetne a kiterjesztése `.htm` is. Ennek ellenére ez nem egy HTML lap, hanem egy szabályos JSP oldal, melyből a Tomcat a színfalak mögött ugyanúgy szervletet fog generálni (lásd 1.4. fejezet).

DBA_login.jsp



Ez az oldal a PS_list.jsp –hez hasonlóan megjeleníti az üzeneteket. Az elavult üzeneteket – melyeknek az érvényességi ideje már lejárt – automatikusan kijelöli törlésre. Vagyis a lejárt üzenetek benne maradnak az adatbázisban, csupán a felhasználók számára nem fognak már megjelenni, így ezek tényleges törléséről mindig az adatbázis-adminisztrátornak kell gondoskodnia. Az egyes üzeneteket a mellettük lévő nyomógomb segítségével lehet törlésre kijelölni. A „Kilépés” gomb hatására itt is megszűnik a klienskapcsolati környezet.

```
<%@ page import="java.sql.*" %>
<html><body bgcolor="#ffffff">
<h1>Party Service - karbantartás</h1>
<%
   int number_of_rows;
   boolean ok = true;

   try
   {
      String name = null;
      String password = null;
      try {
         name = request.getParameterValues("name")[0];
         password = request.getParameterValues("password")[0];
      } catch (NullPointerException npe) {}
      if (name==null || password==null)
      {
         try {
            name       = (String)session.getValue("name_in_session");
            password = (String)session.getValue("password_in_session");
         } catch (NullPointerException npe) {}
      }
      if (name==null || password==null)
      {
         ok = false;
         %><b>Hiba: </b>hiba a név és/vagy jelszó megadásánál!<p><%
      }

      if (ok)
      {
         Class.forName("org.postgresql.Driver");
         Connection conn = DriverManager.getConnection("jdbc:postgresql://127.0.0.1/"+"party",
                                                       name,password);
             session.putValue("name_in_session",name);
             session.putValue("password_in_session",password);
         %>
<script language="JavaScript"><!--
time();                         //fv. meghivasa
function time()
{
   var now       = new Date()
   var year      = now.getYear()+1900
   var month    = now.getMonth()+1
   var day       = now.getDate()
   if (month.length==1) alert("hopp")//month = "0"+month
   document.write("<h3>A mai dátum: "+year+"-"+month+"-"+day+"</h3>")
}
<!-- vege -->
</script>
         <%
         Statement  st   = conn.createStatement();
         ResultSet rs;

         rs = st.executeQuery("select count(*) from message");
            rs.next();
            number_of_rows = rs.getInt(1);
         if (number_of_rows > 0)
         {
            rs = st.executeQuery("select M.id, M.name, M.sent, M.body, U.e_mail, M.exp, "+
                                 "current_date>=M.exp from message M, users U "+
                                 "where M.name=U.name order by M.id desc");
            if (rs!=null)
            {
   %><form action="DBA_delete_old.jsp" name="delete_old" method="POST">
   <table width="95%" border="1">
   <tr><td bgcolor="#BFDFFF" align="center"><b><font size="4">Töröljem?</font></b>
       <td bgcolor="#BFDFFF" align="center"><font size="4"><b>Üzenet:</b></font>
   <%
               while (rs.next())
               {
                  boolean old_msg = rs.getBoolean(7);
   %>
      <tr><td bgcolor=f0f0f0 align="center">
         <input type="checkbox" <% if (old_msg) out.print("checked "); %>
                name="what_to_delete" value="<%= rs.getString(1) %>">
                <% if (old_msg) out.print("<br>lejárt<br>("+rs.getString(6)+")"); %>
          <td>
```

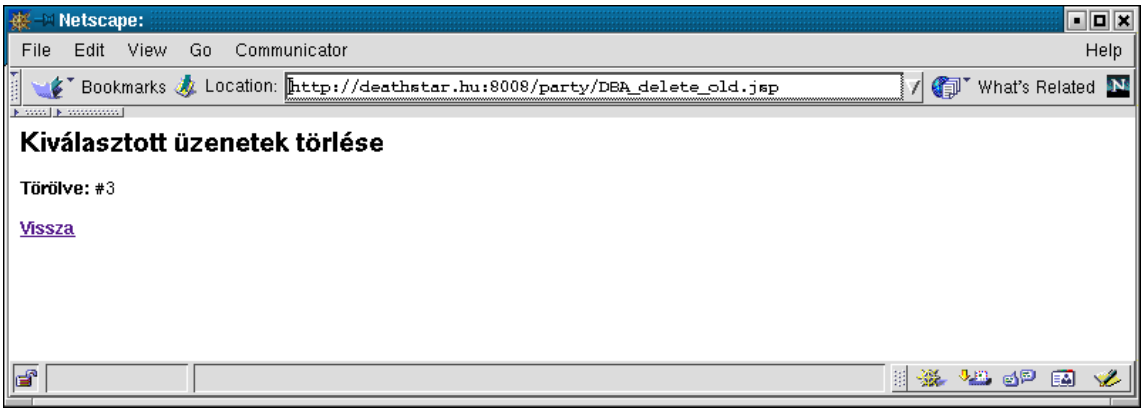
```
         <TABLE width=100% bgcolor=FFFFFF border=0 cellpadding=0 cellspacing=0 align=center>
         <TR>
         <TD bgcolor=ffe699 align=left>
          <FONT size=-1 face=Arial, Helvetica><B><%= rs.getString(2) %></B></FONT>
         </TD>
         <TD bgcolor=ffe699 align=right>
          <FONT SIZE=-2 face=Arial, Helvetica>
          <A HREF="mailto:<%= rs.getString(5) %>">e-mail</A>
             <%= rs.getString(3) %>   (<%= rs.getString(1) %>)
          </FONT>
         </TD>
         </TR>
         <TR><TD colspan=2 height=1 bgcolor=000000 valign=top>
             <SPACER type=block width=1 height=1></TD></TR>
         <TR><TD bgcolor=f0f0f0 colspan=2 valign=top><FONT SIZE=-1 FACE=Arial, Helvetica>
         <pre><%= rs.getString(4) %></pre>
         </TR>
         </TABLE>
   <FONT size=0><BR></FONT>
   <%
               }
   %></table><%
             }
         }
         else %>Üres az adatbázis, nincs semmi üzenet.<br><%
         rs.close();
         st.close();
         conn.close();
%>
<p>
<table>
<tr><td><a href="javascript:document.delete_old.submit()">
<img src="assets/kijelolt_uzenetek_torlese2.gif" border="0"></a>
    <td><a href="DBA_logout.jsp"><img src="assets/kilepes2.gif" border="0"></a>
</tr>
</table>
<%
      }
   } catch (java.sql.SQLException sqle)
     { out.println("<b>SQL hiba:</b>"+sqle.toString()+"<p>"); }
     catch (ClassNotFoundException cnfe) { out.println(cnfe.toString()); }
%>
<a href="DBA_start.jsp"><b>Vissza</b></a>
</body></html>
```

DBA_logout.jsp

```
<% session.invalidate(); response.sendRedirect("DBA_start.jsp"); %>
```

Megszünteti a session-t, s behívja a bejelentkezési képernyőt.

DBA_delete_old.jsp



Ez az oldal megkapja a törlendő üzenetek azonosítóját (id), s törli őket az adatbázisból.

```
<%@ page import="java.sql.*" %>
<html><body bgcolor="#ffffff">
<h1>Kiválasztott üzenetek törlése</h1>
<%
   boolean ok = true;
   int row;
   String name=null, password=null;

   try {
      name     = (String)session.getValue("name_in_session");
      password = (String)session.getValue("password_in_session");
   } catch (NullPointerException npe)
   {
      ok = false;
      %><b>Hiba: </b>erre az oldalra csak közvetett úton lehet eljutni!<br>
      <%
   }
   ok = (name!=null && password!=null);

   if (!ok)
   {
      %><b>Hiba: </b>hiba a név és/vagy jelszó megadásánál!<br>
      <%
   }
   else
   {
      try
      {
         Class.forName("org.postgresql.Driver");
         Connection conn = DriverManager.getConnection("jdbc:postgresql://127.0.0.1/"+"party",
                                                       name,password);

         Statement  st   = conn.createStatement();
         String del_query;
         String[] msgs = request.getParameterValues("what_to_delete");
         for (int i=0; i<msgs.length; ++i)
         {
            del_query = "delete from message where id="+msgs[i];
            row = st.executeUpdate(del_query);
            if (row==1)   { %><b>Törölve: </b>#<%= msgs[i] %><br><% }
            else
            {
               %><b><font color="red">Nem sikerült törölni: </font></b>#<%= msgs[i] %><br><%
            }
         }
      } catch (NullPointerException npe) { %>
         Nincs mit törölni!<br>
         <%
        }
        catch (java.sql.SQLException sqle) { %>
         <b>SQL hiba: </b><%= sqle.toString() %>
         <%
        }
        catch (ClassNotFoundException cnfe) { %>
         <%= cnfe.toString() %>
         <%
        }
%>
<%   }
%>
<br>
<a href="DBA_login.jsp"><b>Vissza</b></a>
</html></body>
```

Összegzés

Dolgozatomban a ma használatos – és igen népszerű – web-technológiák közül mutattam be néhányat. Mivel a bemutatott szoftverek mindegyike ingyenesen használható, ezért a leírtakat bárki ki tudja próbálni.

Az Esettanulmány című fejezetben egy konkrét webes alkalmazást készítettem el, melyben egy internetes fórum teljes szerkezetét ismertetem az adatbázis létrehozásától egészen a felhasználóval való kommunikációhoz szükséges oldalak elkészítéséig. Az itt bemutatott rendszer – melyet a jelenleg egyik legmodernebbnek tartott technológiával (JSP – Java Server Pages) készítettem el – alapul szolgálhat mindazon szakemberek számára, akik website-jukat hasonló funkcióval szeretnék kibővíteni.

Mivel a WWW népszerűsége egyre növekszik, ezért ma már csak azok a site-ok versenyképesek, amelyek a megnövekedett információmennyiséget dinamikus módon tudják kezelni, ehhez pedig elengedhetetlen az adatok könnyű tárolását és elérhetőségét biztosító adatbázis-szerverek alkalmazása. A dolgozatban szereplő MySQL, PostgreSQL adatbázisszerverek kis- és középvállalkozások számára nyújthatnak megoldást.

Az említett két adatbáziskezelő-rendszert folyamatosan bővítik új tulajdonságokkal, így várható, hogy ezeket a szoftvereket a jövőben nagyobb vállalkozások is szívesen alkalmazzák. A közeljövőben a Linux, és így a szabad szoftver egyre nagyobb térnyerésére számíthatunk.

1. sz. Melléklet

ApacheJServ installálása

A program installálását Linux Mandrake 7.2 rendszeren próbáltam ki a következő csomagokkal:

apache-1.3.14-2mdk.i586.rpm

ApacheJServ-1.1.2-5mdk.i586.rpm

Természetesen más disztribúció alatt, más verziójú csomagokkal is működnie kell.

/etc/httpd/conf/httpd.conf ez az Apache konfigurációs file-ja,

ennek a végére a JServ berakja magát automatikusan, így a JServ az Apache-csal együtt fog majd indulni

/etc/httpd/conf/jserv itt vannak az ApacheJServ konfigurációs file-jai

Az egyes felhasználókhoz hozzunk létre külön szervletzónákat, így az egyes szervletek jól elkülöníthetőek egymástól, s minden felhasználónak csak a saját zónáját kell rendben tartania.

Tegyük fel, hogy létezik egy jabba nevű felhasználó; HOME-könyvtára: /home/jabba. Hozzunk létre számára egy szervletzónát! Ehhez a következőket kell beállítani:

jserv.conf itt kell felmount-olni az egyes szervletzónákat. Kettő alapból létezik, ezek:

(1)
```
ApJServMount /servlets /root

ApJServMount /servlet /root
```

Ez azt jelenti, hogy a zóna neve root, s a böngészőben majd így hivatkozhatunk rá:
http://localhost/servlet/...
http://localhost/servlets/...

Ehhez a listához adjuk hozzá jabba-t is:

```
ApJServMount /jabba/servlets /jabba1
```

jserv.properties

(2)
```
zones=root,jabba1
```

itt kell felsorolni a szervletzónák neveit

(3)
```
root.properties=/etc/httpd/conf/jserv/zone.properties
```

```
jabba1.properties=/home/jabba/servlets/zone.properties
```

a szervleteket NE tegyük a public_html-be, mert onnan bárki letöltheti és visszafordíthatja a .class file-okat!!! Inkább hozzunk létre a HOME könyvtárunkban egy külön *servlets* könyvtárat.

zone.properties (4) másoljuk át a (3)-as pont alatti "minta" zone.properties file-t

a szervlet-könyvtárunkba (/home/jabba/servlets). Más is lehet a neve a file-nak, de egyezzen azzal a file-névvel, amit a (3)-as pont alatt megadtunk. Ebben a file-ban módosítsuk a következő sort:

```
repositories=/home/jabba/servlets
```

Vagyis megadjuk, hogy a szervleteink hol találhatóak. Ha vannak .jar file-jaink, akkor azokat is itt kell felsorolni.

Miután ezzel készen vagyunk, még be kell állítani a jogokat!

`/home/jabba`	`drwxr-xr-x` jogok ($HOME könyvtár)
`/home/jabba/servlets`	`drwxr-xr-x` jogok (itt tároljuk a szervleteket)
`/home/jabba/servlets/*`	`-rw-r--r--` jogok (644)

Az ApacheJServ telepítéskor felrak két példaprogramot (Hello, IsItWorking). Ezek megtalálhatóak a root szervletzónában. Másoljuk át őket a mi $HOME/servlets könyvtárunkba, állítsuk be a jogokat, s hozzáláthatunk a kísérletezéshez.

A konfigurációs file-ok módosítása esetén a változtatások érvényesítése érdekében indítsuk újra az Apache-ot: /etc/init.d/httpd restart

Ezután először nézzük meg, hogy a root zóna megy-e:

http://localhost/servlet/Hello
http://localhost/servlets/IsItWorking

Ha megy, akkor nézzük meg a mi szervletzónánkat:

http://localhost/jabba/servlets/Hello

az (1)-es pont alapján tehát a mi zónánk így van bemount-olva, így tudunk rá a böngészőből hivatkozni

Ha mégsem menne, akkor nézzünk át mindent elölről:

- jserv.conf-ban 1 bejegyzés
- jserv.properties-ben 2 bejegyzés
- jogok a könyvtárakra, file-okra

Ha minden rendben, és mégsem működik, akkor próbálkozzunk a következőkkel:

az (1),(2),(3)-mal jelölt részek valamelyikében cseréljük fel a sorrendet! Például az (1)-esben a mount-olási sorrendet, stb. Érdekes módon néha csak ennyin múlik.

Ha "Internal Server Error" a válasz:

ez már félsiker: a log file-ban nézzük meg, hogy mi volt a gond:
/var/log/httpd/jserv/jserv.log

2. sz. Melléklet

Az Apache beállítása

Az Apache webszerver a mai Linux disztribúciók részét képezi, így telepítését már az operációs rendszer installálásakor elvégezhetjük. A webszervert Linux Mandrake 7.2 alatt próbáltam ki. Ez a disztribúció a Red Hat Linuxra épül, átvéve annak RPM (Red Hat Package Manager) csomagkezelőjét is. Ennek az az óriási előnye, hogy miután letöltöttük egy program bináris változatát, egyetlen paranccsal telepíteni tudjuk (`rpm -ivh csomag_neve`), ill. később ugyanilyen egyszerűen el is tudjuk távolítani a rendszerből (`rpm -e csomag_neve`).

Az Apache beállításait tartalmazó konfigurációs file a `httpd.conf` (ezt az `/etc/httpd/conf/` könyvtárban találjuk meg). Ha módosítjuk ezt a file-t, akkor a változtatások érvénybeléptetéséhez újra kell indítani a webszervert (root felhasználóként):

```
/etc/init.d/httpd restart
```

A kiszolgáló működésének a tesztelésére a telnet a legalkalmasabb. Ezzel nem csak a 23-as telnet portra lehet csatlakozni, hanem tetszőleges kapura. A httpd.conf file-ban a

```
Port 80
```

például azt határozza meg, hogy a webszervert a 80-as porton keresztül lehet elérni. Lépjünk be a saját gépünk 80-as portjára:

```
telnet localhost 80
```

A legegyszerűbb lekérdezés a GET /, melyet ENTER-rel zárunk. Ezt a formát kompatibilitási okokból a legtöbb korszerű kiszolgáló támogatja, de már nem használatos. Az ENTER leütése után megjelenik a webszerver kezdőlapja. Azt, hogy melyik oldalt hozza be ilyenkor a kiszolgáló, a következőképpen lehet beállítani:

```
DocumentRoot /var/www/html
```

Összetettebb lekérdezéseket a HTTP/1.0 használatával intézhetünk. Ez a HTTP első olyan változata, melynél a lekérdezésekben és válaszokban fejléc is található. A fenti egyszerű lekérdezést a GET / HTTP/1.0 alakban is megadhattuk volna, ezzel jelezve, hogy a kliens a HTTP 1.0-s változatát is megérti. Ezt két ENTER-rel kell zárni – az első a sor végét jelöli, a másik pedig azt, hogy a parancssor és a HTTP-kérelem között nem akarunk fejlécet elküldeni.

Hány kiszolgálót futtassunk?

Még a kevésbé látogatott weboldalak üzemeltetői is több Apache-folyamatot futtatnak egyszerre. A régebbi kiszolgálók általában addig várnak, amíg egy új kapcsolat szükségessé teszi az új folyamat indítását. Az Apache készítői azonban nem javasolják ezt a módszert, így az Apache már induláskor több alfolyamatot is elindít. Egy alfolyamat egyszerre csak egy HTTP-kapcsolatot kezel. A határértéket a **MaxClients** paranccsal állíthatjuk be. Ennek alapértéke 150. Ha ezt az értéket túl alacsonyra választjuk, akkor az újonnan kapcsolódó látogatóknak várniuk kell, míg valamelyik alfolyamat felszabadul.

Az Apache a httpd.conf file-ban megadott elvek alapján állandóan változtatja a kiszolgálók számát a bejövő kérelmeknek megfelelően. A **MinSpareServers** és a **MaxSpareServers** parancsokkal szabályozhatjuk, hogy hány tartalék kiszolgáló fusson állandóan a háttérben, melyek feladata a friss kérelmek azonnali kiszolgálása. Ha a szabad tartalékok száma a MinSpareServers-ben megadott szintre csökken, akkor az Apache több új kiszolgálót indít. Ha a folyamatok száma eléri, vagy meghaladja a MaxSpareServers paranccsal meghatározott szintet, az Apache azonnal leállítja a feleslegesen futó folyamatokat. Ha a webszerverhez való kapcsolódás túl sok időt vesz igénybe, akkor valószínűleg a fenti értékekkel lesz gond. Ilyenkor növeljük meg a MaxSpareServers, vagy a MaxClients értékét azért, hogy minél kevesebb felhasználónak kelljen várnia a kapcsolódásra.

Természetesen az új folyamatok indítása nagyon leterheli a számítógépet – a processzor kevesebb időt tud szánni egy-egy folyamatra, és a memória is vészesen fogy. A Linux **free** parancsával le tudjuk ellenőrizni az elérhető fizikai és virtuális memóriát, a **top** paranccsal pedig az egyes folyamatok által elfogyasztott processzoridőt jeleníthetjük meg.

Adatbázis használata

Ha a honlap és az adatbázis (pl. MySQL, PostgreSQL) ugyanazon a gépen található, akkor komolyabb gondok is előfordulhatnak. A dinamikusan létrehozott oldalak látogatottságának növekedésével természetesen egyre több Apache-folyamat fut a gépen, de a látogatók kiszolgálása céljából az adatbázist kezelő kapcsolatok számának is növekednie kell. Egy ponton a honlap saját népszerűségének az áldozatává válik, ui. az Apache és az adatbázis egyre nagyobb küzdelmet folytat a rendszererőforrásokért. A nagy forgalmú, adatbázist is

használó honlapok esetében érdemes tehát a két feladatot különválasztani: a webkiszolgáló külön gépéhez egy vagy több, az adatbázisok kezeléséért felelős gép csatlakozzon.

Az Apache pillanatnyi állapota

Az Apache állapotáról a **mod_status** modullal készíthetünk pillanatfelvételt. A mod_status alapértelmezés szerint az Apache része. Ezt az információt általában csak egyetlen URL-lel használjuk. Például létrehozunk egy „/server-status" nevű URL-t a kiszolgálón, melyet megtekintve a látogató a kiszolgáló állapotáról tájékozódhat. Érdemes mindig a teljes állapotjelzést megjeleníteni (ExtendedStatus On):

```
<Location /server-status>
      SetHandler server-status
</Location>
ExtendedStatus On
```

Illesszük be ezt a négy sort a httpd.conf file-ba, majd indítsuk újra a webszervert. Ezután a http://localhost/server-status URL-t megtekintve az alábbihoz hasonló üzenetet kapunk:

```
Server Version: Apache-AdvancedExtranetServer/1.3.14 (Linux-Mandrake/2mdk)
PHP/4.0.4pl1 mod_ssl/2.7.1 OpenSSL/0.9.5a ApacheJServ/1.1.2
Server Built: Oct 24 2000 10:55:49

Current Time: Sunday, 08-Apr-2001 10:11:44 /etc/localtime
Restart Time: Sunday, 08-Apr-2001 07:49:20 /etc/localtime
Parent Server Generation: 0
Server uptime: 2 hours 22 minutes 24 seconds
Total accesses: 63 - Total Traffic: 159 kB
CPU Usage: u.26 s.17 cu.02 cs.07 - .00609% CPU load
.00737 requests/sec - 19 B/second - 2584 B/request
1 requests currently being processed, 9 idle servers
```

Az állapotadat a kiszolgáló verziószámát, moduljait, indításának időpontját, a kapcsolatok számát és forgalmát tartalmazza. Megjeleníti az eddigi teljes forgalmat, ill. azt is kijelzi, hogy hány folyamat várakozik (így ki lehet deríteni, hogy megfelelően állítottuk-e be a MaxSpareServers értékét).
A mod_status ezután a következő alakban jelenít meg adatokat:

```
_W_____.....................................................
.....................................................
.....................................................
.....................................................
```

Minden „." karakter egy feladat nélkül várakozó Apache-folyamatot jelöl. Az új kapcsolatra váró folyamatok jele „_", a kérelmet beolvasóké „R", a választ küldőké „W". A jelek természetesen állandóan változnak. Ezt követően az egyes működő folyamatok állapotáról

tájékozódhatunk. Láthatjuk például, hogy mely kapcsolatok feldolgozása tart sokáig, melyek a legnépszerűbb kapcsolatok, stb.
Természetesen nem túl bölcs dolog a webszerver állapotát az egész világ elé tárnunk. Szerencsére az „order”, „deny”, „allow” parancsokkal szabályozhatjuk a hozzáférést:

```
<Location /server-status>
SetHandler server-status
order deny,allow
deny from all
allow from localhost
</Location>
ExtendedStatus On
```

A fenti beállítások hatására a mod_status kimenete csak a webszervert futtató gépről lesz elérhető. Egyéb kérelmekre az „Access forbidden” (hozzáférés megtagadva) üzenettel válaszol.

CGI-programok futtatásának az engedélyezése

Ahhoz, hogy egy felhasználó CGI-programokat tudjon futtatni, ezt a httpd.conf file-ban külön engedélyezni kell a számára. Tegyük fel, hogy van egy *jabba* nevű felhasználónk (HOME könyvtára: */home/jabba*). Az ő számára engedélyezzük, hogy a *$HOME/public_html/cgi-bin* könyvtárban CGI-ket futtathasson. Ehhez a következő négy sort kell beilleszteni:

```
<Directory /home/jabba/public_html/cgi-bin>
AllowOverride None
Options ExecCGI
</Directory>
```

Az adott felhasználó ezután már futtathat CGI programokat, de csak ebben a könyvtárban.

Modulok használata

A `httpd -l` paranccsal tudjuk megnézni, hogy mely modulok vannak az Apache-ba statikusan belefordítva. A legjobb megoldás az, ha az Apache-ba csupán két modul kerül be statikusan: az egyik a **mod_core** (ez az alapvető szolgáltatásokat tartalmazza), a másik pedig a **mod_so** (ezzel tölthetjük be a DSO-kat (Dynamic Shared Objects)). A többi modult csak szükség esetén kell betöltenünk. Mivel így a modulok a programon kívül helyezkednek el, ezért frissítésük is egyszerűbben végezhető el, a program újrafordítása nélkül. S pontosan ez az, ami a folyamatos működést igénylő, forgalmas webkiszolgálók esetén rendkívül fontos.

Minden modult a LoadModule paranccsal kell betölteni, majd az AddModule paranccsal engedélyeznünk. Például:

```
LoadModule info_module        modules/mod_info.so
AddModule mod_info.c
```

A LoadModule-nak két értéke van: az egyik a modul neve, a másik az .so file. A fenti példában a DSO modulok az /usr/lib/apache könyvtárban helyezkednek el. A LoadModule parancsnak meg kell előznie az AddModule parancsot.

Összegzés

Az Apache egy megbízható webkiszolgáló, bár eléggé összetett, s így használatához, telepítés utáni beállításához elkél némi tapasztalat. Szerencsére az Apache-ot úgy is telepíthetjük, hogy a modulok telepítése és frissítése egyszerűen végrehajtható legyen. Az Apache további előnye még ingyenessége, így használata mindenkinek ajánlható.

3. sz. Melléklet

HTML kód Java-sztringgé alakítása

```
#!/usr/bin/perl -w
#
# HTML kód Java-sztringgé alakítása
#

(@ARGV==1) || die("$0: I need one and only one parameter!\n");
open (F1, "<$ARGV[0]") || die("$0: cannot open $ARGV[0]!\n");
$_ = <F1>;
while (1)
{
   if (!$_) { last; }
   chomp;
   $next = <F1>;

   s#\"#\\\"#g;
   s#\$([\d\w\.()]+)#\"+$1+\"#g;
   $continue = ($next ? "+" : "");
   $_ = "\"$_\\n\"".$continue;
   print $_.($continue ? "\n" : ";");
   $_ = $next;
}
print "\n";
close F1;
```

Ezzel a kis szkripttel kész HTML kódokat (kódrészleteket) lehet Java-beli String objektumokká átalakítani. Egy szervlet a kliens számára egy HTML forrást állít elő, melyet egy *print* utasítással nyomtat ki (s ezt a webszerver eljuttatja a kliensnek). Ezt a HTML kódot pedig sztringek segítségével lehet felépíteni. Például a következő kódot szeretnénk elküldeni a kliensnek:

```
<td>
<form method="post" action="Lista">
   <input type="submit" value="Töröl">
   <input type="hidden" name="name" value="$name">
   <input type="hidden" name="password" value="$password">
   <input type="hidden" name="nev" value="$rc.getString(1)">
</form>
<form method="post" action="Lista">
<input type="submit" value="Módosít">
   <input type="hidden" name="name" value="$name">
   <input type="hidden" name="password" value="$password">
   <input type="hidden" name="nev" value="$rc.getString(1)">
</form>
</td>
```

A *$változó_név* jelzésű helyeken a *változó_név* nevű String objektumok értéke kerül. Ha ezt egy print utasítással ki szeretnénk nyomtatni, akkor a következőt kellene írnunk:

```
"<td>\n"+
"<form method=\"post\" action=\"Lista\">\n"+
"   <input type=\"submit\" value=\"Töröl\">\n"+
"   <input type=\"hidden\" name=\"name\" value=\""+name+"\">\n"+
"   <input type=\"hidden\" name=\"password\" value=\""+password+"\">\n"+
"   <input type=\"hidden\" name=\"nev\" value=\""+rc.getString(1)+"\">\n"+
"</form>       \n"+
"<form method=\"post\" action=\"Lista\">\n"+
"<input type=\"submit\" value=\"Módosít\">\n"+
"   <input type=\"hidden\" name=\"name\" value=\""+name+"\">\n"+
"   <input type=\"hidden\" name=\"password\" value=\""+password+"\">\n"+
"   <input type=\"hidden\" name=\"nev\" value=\""+rc.getString(1)+"\">\n"+
"</form>       \n"+
"</td>\n";
```

Pontosan ez lesz az az eredmény, amit a szkript előállít. Látható, hogy még egy ilyen egyszerű kis HTML kód esetén is milyen nagy figyelmet igényelne a sok escape-szekvencia helyes alkalmazása.

A program egyetlen argumentumot vár (a HTML kódot tartalmazó file nevét), s a kimenetet a standard output-ra küldi (melyet át tudunk irányítani egy tetszőleges file-ba).

4. sz. Melléklet

A PHP konfigurálása

A PHP-t szintén előre lefordított bináris csomagból telepítettem. A munka írásakor a legfrissebb verzió a 4.0.4pl1 volt.

A program fő konfigurációs állománya a `/etc/php.ini` file. A PHP működését itt lehet beállítani. Az egyes lehetőségek:

- nyelvi kapcsolók
- erőforrás-korátozás
- hibakezelés, hibák naplózása
- adatkezelés
- elérési utak, könyvtárak
- dinamikus kiterjesztések
- modulbeállítások

Ha a MySQL, ill. PostgreSQL adatbáziskezelőkhöz is szeretnénk kapcsolódni PHP-ból, akkor még két további csomagot kell feltelepíteni: php-pgsql, ill. php-mysql. Ezek csupán egy-egy file-t (pgsql.so, ill. mysql.so) tesznek be a PHP futásidejű modulokat tartalmazó könyvtárába (ez a /usr/lib/php/extensions címen található meg). Ahhoz, hogy ezeket a PHP is használni tudja, a következő két sor elől törölni kell a megjegyzésjelet:

```
extension=pgsql.so
extension=mysql.so
```

A php.ini -ben még ellenőrizzük le, hogy a futásidejű modulokat valóban az előbb említett könyvtárban keresi-e:

```
extension_dir      =     /usr/lib/php/extensions
```

5. sz. Melléklet

A MySQL telepítése, kezdeti beállítása

Linux alatt a legelterjedtebb adatbáziskezelő-rendszer. A Linux Journal 2000. évi olvasói közvéleménykutatása alapján a MySQL közel kétszer annyi szavazattal büszkélkedhet, mint a PostgreSQL. A harmadik helyen az Oracle 8i állt. A többi adatbázis egyike sem érte el a 3%-ot, ezért csak az első két helyezettel foglalkozunk.

A dolgozat írásakor a legfrissebb verzió a 3.23.31 volt. Ahhoz, hogy a felhasználók csak a nekik kiszabott feladatokat tudják elvégezni, a szervernek a felhasználóihoz jogokat kell rendelnie. Ezeket a jogosultságokat az SQL szerverek ugyanolyan táblákban tárolják, mint a többi adatot. A MySQL a mysql nevű adatbázis öt táblájában tárolja el ezeket az információkat: *user*, *db*, *host*, *table_priv*, *column_priv*.

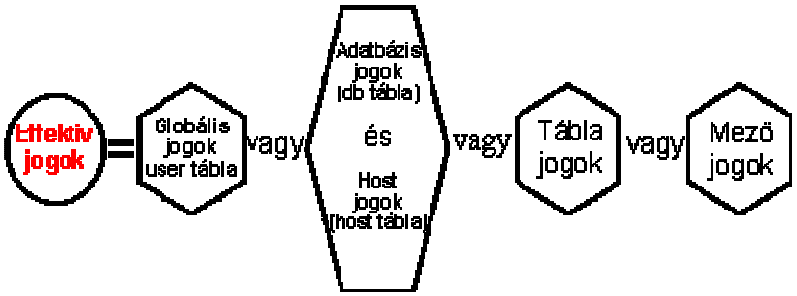
A **user** táblában adatbázisszintű jogokat tárol a szerver. Tehát ha itt jogosultságot adunk valamihez egy felhasználónak, akkor az a szerver minden adatbázisára és táblájára érvényes lesz. Általában itt csak a szerver gazdájának szoktak jogosultságokat engedélyezni, így nem kell neki minden egyes adatbázisra külön elvégezni azt, ezért ezeket a jogokat globális jogoknak is hívják.

A **db** táblában szereplő jogok már csak egy adatbázisra vonatkoznak, azon belül viszont az összes táblára, illetve mezőre. A db táblához szorosan kapcsolódik a host tábla, melyet akkor használ a szerver, ha a db táblában kitöltetlenül marad a host mező.

A **host** táblában szereplő bejegyzésekkel gépenkét korlátozhatjuk a felhasználók jogait, mivel az előzőleg a db táblában kikeresett rekordban szereplő és a host táblában a gépre illő rekord metszetét veszi a szerver. Így a db táblában megadhatunk megengedőbb, általánosabb jogokat, melyeket a host táblával szűkíthetünk.

A **tables_priv** táblában a tábla azonosítására, míg a **columns_priv** táblában a tábla és az adott mező azonosítására is szerepel egy-egy adat.

Ha csatlakozott a felhasználó, akkor kiadható az SQL parancs. Ezután a program sorra elkezdi nézni az előbb említett táblákat, hogy végrehajthatja-e az adott felhasználó (az adott gépről) a parancsot. Ezt a következő ábra szemlélteti:



A MySQL mindaddig halad balról jobbra, amíg a kifejezés igaz nem lesz. Ha a végére ér a sornak és még mindig nem lesz igaz a feltétel, akkor hibaüzenetet ad (Acces denied).

A MySQL telepítése

A MySQL-t előre lefordított csomagból feltelepíteni nagyon egyszerű. Telepítés után próbáljuk meg elindítani, ill. leállítani a rendszert:

```
root$ safe_mysqld --log &
mysqladmin -u root shutdown
```

Ha sikerült, akkor rendben lezajlott a telepítés. Ezután érdemes azonnal foglalkozni a szerverben szereplő felhasználókkal, mert alapértelmezésben Linux alatt például a root felhasználónak (akinek mindenhez van joga a users táblában) nincs semmilyen jelszava, és a localhost-ról, azaz a saját gépünkről léphet be, ami még nem is olyan nagy gond, ha egyedüli használói vagyunk a gépnek. Először adjunk valamilyen jelszót a root felhasználónak (ne felejtsük elindítani a szervert):

```
mysqladmin -u root password "uj_jelszo"
```

Csatlakozzunk a szerverhez és töröljük a root kivételével az összes felhasználót (a jelszó nélküli és a nem localhost-ról csatlakozó root-ot is töröljük). (Figyelem: ha hoztunk már létre felhasználókat, és nem akarjuk töröni őket, akkor **ne** hajtsuk végre a következő parancsokat!)

Csatlakozunk a szerverhez:

```
mysql -u root -p mysql
Enter password: ******
```

Nézzük meg hogy mit fogunk törölni:

```
mysql>SELECT host, user, password FROM user
      WHERE host!='localhost' or user!='root' or
      password='';
```

Ha jól meggondoltuk és tényleg törölni akarjuk a kilistázott felhasználókat, akkor gépeljük be a következő parancsot:

```
mysql>DELETE FROM user WHERE host!='localhost'
      or user!='root' or password='';
```

A telepítés után még egy anonymous (név nélküli) felhasználó kerül a rendszerbe, aki csak a db táblában szerepel, mivel globális jogai nincsenek. Csupán a *test* adatbázisra rendelkezik minden jogosultsággal, viszont ő is jelszó nélkül léphet be bármilyen gépről, ami szintén nem felel meg igényeinknek, ezért törölhetjük őt is, pontosabban a db tábla egész tartalmát. (Az előző figyelmeztetés természetesen most is érvényes.)

Nézzük meg, hogy kit (kiket) fogunk törölni:

```
mysql>SELECT host, db, user FROM db;
```

Majd ha jól meggondoltuk töröljünk mindent:

```
mysql>DELETE FROM db;
```

Még egy valamiről kell gondoskodnunk és tökéletes lesz szerverünk biztonság szempontjából: Ha gyalogos módszerrel, azaz GRANT és REVOKE helyett INSERT, UPDATE, DELETE parancsokkal módosítjuk ezeket a privilégiumokat tároló táblákat, akkor tudatnunk kell a MySQL-el, hogy frissítse a memóriába betöltött privilégiumokat. Ehhez adjuk ki a következő parancsot:

```
FLUSH PRIVILEGES;
```

Ezután bátran hozzá lehet kezdeni a MySQL-lel való munkához.

6. sz. Melléklet

A PostgreSQL telepítése, használatba vétele

Linux alatt a MySQL mellett a PostgreSQL a legnépszerűbb adatbáziskezelő-rendszer. Mandrake 7.2-es operációs rendszert feltételezve telepítéséhez a következő csomagok szükségesek:

- postgresql-7.0.2-6mdk (maga a szerver)
- postgresql-server-7.0.2-6mdk (kliens)
- postgresql-perl-7.0.2-6mdk (Pg modul Perl-hez)
- postgresql-jdbc-7.0.2-6mdk (JDBC meghajtóprogramok Java-hoz)

A továbbiakban a következőket nézzük meg: adatbázis-felhasználó létrehozása, jelszavának beállítása, adatbázis létrehozása, tábla létrehozása, tábla feltöltése adatokkal.

```
[jabba@localhost sql]$ su postgres
Password: ********
[postgres@localhost sql]$ createuser
Enter name of user to add: proba
Shall the new user be allowed to create databases? (y/n) y
Shall the new user be allowed to create more new users? (y/n) n
CREATE USER
[postgres@localhost sql]$ exit
exit
```

A postgres nevű felhasználó root jogosultságokkal rendelkezik a PostgreSQL adatbázis-környezetben. Az ő nevében létrehozunk egy új felhasználót. (Ez az új felhasználó csupán egy adatbázis-felhasználó, nem muszáj léteznie az /etc/passwd állományban). Az új felhasználónak jogokat adunk: létrehozhat új adatbázisokat, de nem hozhat létre új felhasználókat.

```
[jabba@localhost sql]$ psql -U proba
Welcome to psql, the PostgreSQL interactive terminal.

Type:  \copyright for distribution terms
       \h for help with SQL commands
            \? for help on internal slash commands
            \g or terminate with semicolon to execute query
            \q to quit

Null display is "NULL".
proba=> select current_user;
 getpgusername
---------------
 proba
(1 row)
```

```
proba=> alter user "proba" with password 'proba_jelszo';
ALTER USER
```

Csatlakozzunk az adatbázis-szerverhez proba felhasználóként, majd változtassuk meg a jelszavunkat (alapértelmezésben még nincs jelszava egy új felhasználónak).

```
proba=> create database proba_db;
CREATE DATABASE
proba=> \l
List of databases
Database  |  Owner   | Encoding
----------+----------+-----------
areas     | jabba    | SQL_ASCII
mydb      | jabba    | SQL_ASCII
postgres  | postgres | SQL_ASCII
proba_db  | proba    | SQL_ASCII
template1 | postgres | SQL_ASCII
(5 rows)

proba=> \c proba_db
You are now connected to database proba_db.
proba_db=> \d
No relations found.
```

Hozzunk létre egy proba_db nevű adatbázist. A \l paranccsal le tudjuk kérdezni a már létező adatbázisok listáját. Egy adatbázishoz a \c paranccsal lehet kapcsolódni. A \d –vel egy adatbázis tábláit lehet megjeleníteni.

```
proba_db=> create table codes(code char(3) not null,
proba_db(> name char(30));
CREATE
proba_db=> \copy codes from stdin
hu      hungary
fr      france
de      germany
\.
proba_db=> select * from codes;
 code |              name
------+--------------------------------
 hu   | hungary
 fr   | france
 de   | germany
(3 rows)

proba_db=> \q
[jabba@localhost sql]$
```

Ezután létrehozunk egy táblát, melynek szerkezete megegyezik az 1.2.3. fejezetben már használt tábláéval. Egy táblát nem csak insert utasításokkal lehet feltölteni, hanem egy file-ból is, melyben az egyes oszlopok értékeit tabulátorok választják el. Most file helyett billentyűzetről olvassuk az adatokat, s <CTRL-D>-vel jelezzük a bevitel végét. A psql-ből (mely egy karakteres kliens a szerverhez) a \q paranccsal lehet kilépni.

Irodalomjegyzék

Könyvek:

1. Balogh Judit – Rutkovszky Edéné: SQL példatár. Debrecen: CODEX-3V, 1994.
2. Bócz Péter – Szász Péter: A világháló lehetőségei: Interaktív Weblapok készítése. 2. jav. kiad. Bp.: ComputerBooks K., 2000.
3. Eckel, Bruce: Thinking in Java 2nd Edition. New Jersey: Prentice Hall, 2000.
4. Jamsa, Kris: A WEB programozása 2. / ford.: Inotai László. Bp.: Kossuth K., 1997.
5. Java: Das Grundlagen-Buch / Daniel Becker. Düsseldorf: Data Becker, 1999.
6. Java 2. útikalauz programozóknak. / szerk.: Nyékiné Gaizler Judit et al. 6. jav. kiad. Bp.: ELTE, 2000. 1–3. köt. Mell.: 1 db CD-ROM
7. Manger, Jason J.: A JAVA programozási nyelv. / ford.: Werner Zsolt. Bp.: Panem, 1996.
8. McMillan, Michael: Perl 2. / ford.: Morvai Gábor. Bp.: Panem, 1999. (WEB Világ)
9. Momjian, Bruce: PostgreSQL: introduction and concepts. Boston: Addison–Wesley, 2001.
10. MySQL and mSQL. / Editor: Andy Oram. Beijing: O'Reilly, 1999.
11. Schwartz, Randal L. – Christiansen, Tom: A PERL programozási nyelv. Bp.: Kossuth K., 1998.
12. Stolnicki Gyula: SQL kézikönyv. 2. átd. bőv. kiad. Bp.: ComputerBooks K., 1995.

Folyóiratcikkek:

13. Az Apache fejlődik = Linuxvilág, 2. évf. 2001. 2–3. sz. 25.p.
14. Klapcsik Péter: Ismerkedés a PHP-vel: I. Telepítés és az első lépések = Új Alaplap, 18. évf. 2000. 12. sz. 58–59.p.
15. Klapcsik Péter: Ismerkedés a PHP-vel: II. Telepítés Windowsra és a vendégkönyv elkészítése = Új Alaplap, 19. évf. 2001. 1. sz. 63–64.p.
16. Klapcsik Péter: Ismerkedés a PHP-vel: III. Telepítés: MySQL Windowsra, PostgreSQL Linuxra = Új Alaplap, 19. évf. 2001. 2. sz. 61–62.p.
17. Klapcsik Péter: Ismerkedés a PHP-vel: IV. Adatbázisok – A PHP és a MySQL kapcsolata = Új Alaplap, 19. évf. 2001. 3. sz. 61–62.p.

18. Korsós István: A PHP programozási nyelv I. = PC World, 9. évf. 2000. 12. sz. 130–133.p.
19. Korsós István: A PHP programozási nyelv II.: Változók = PC World, 10. évf. 2001. 1. sz. 76–79.p.
20. Korsós István: A PHP programozási nyelv III.: Űrlapok = PC World, 10. évf. 2001. 2. sz. 98–100.p.
21. Korsós István: A PHP programozási nyelv IV.: Áttekintés = PC World, 10. évf. 2001. 3. sz. 86–87.p.
22. Lerner, Reuven M.: Az Apache beállítása, trükkjei és hibakeresés = Linuxvilág, 1. évf. 2000. 1. sz. 54–58.p.
23. Lerner, Reuven M.: Merre tovább, melyik úton?: Nézzünk körbe a webes világban: milyen módszerek és eszközök vannak, melyek a legígéretesebbek = Linuxvilág, 2. évf. 2001. 1. sz. 60–63.p.
24. Mead, Heather: A 2000. évi olvasói díjak = Linuxvilág, 1. évf. 2000. 2. sz. 15–17.p.
25. Novák Áron: Szerveroldali bűvészkedések = Internet Kalauz. 5. évf. 2000. 12. sz. 55–56.p.
26. Perdue, Tim: PHP4 és PostgreSQL: komoly webes alkalmazások készítése nyílt forráskódú programeszközökkel. = Linuxvilág, 1. évf. 2000. 2. sz. 71–75.p.

WWW címek:

27. http://www.idaya.co.uk/news/newsdesk/press_releases.phtml (felmérés a Linux térnyeréséről)
28. http://www.netcraft.com/survey (felmérés a webszerverekről)
29. http://www.linux-mandrake.com (Linux Mandrake)
30. http://www.mysql.com
31. http://www.postgresql.org
32. http://www.postgresql.org/docs/awbook.html (PostgreSQL on-line könyv)
33. http://java.sun.com (a Java hivatalos oldala)
34. http://java.sun.com/products/jdk/1.2/docs/api/index.html (JDK API referencia)
35. http://java.sun.com/products/jdbc
36. http://java.sun.com/j2se (Java2 platform)
37. http://java.sun.com/products/servlet

38. http://java.apache.org (Apache JServ)
39. http://java.sun.com/products/jsp
40. http://java.sun.com/products/products.a-z.html (lista az összes termékről)
41. http://www.apl.jhu.edu/~hall/java/Servlet-Tutorial (JSP 1.0)
42. http://www.javacaps.com/jsp/java_jsp.html
43. http://www.big-boys.com/jsp
44. http://java.sun.com/products/servlet/download.html (Tomcat letöltése)
45. http://java.sun.com/docs/books/tutorial/index.html (Java oktató)
46. http://www.mindview.net/Books/TIJ (Bruce Eckel – Thinking in Java, letölthető könyv)
47. http://hu.php.net (a PHP hivatalos oldala)
48. http://hu.php.net/manual/hu
49. http://www.phpbuilder.com/columns/david20000512.php3 (PHPLib)
50. http://phplib.netuse.de (PHPLib)
51. http://www.prog.hu
52. http://www.prog.hu/cikkek/prog_langs/web_langs/php/01/index.htm
53. http://www.prog.hu/cikkek/prog_langs/web_langs/php/03/index.htm
54. http://www.prog.hu/cikkek/prog_langs/web_langs/php/04/index.htm
55. http://www.prog.hu/cikkek/prog_langs/web_langs/php/05/index.htm
56. http://www.prog.hu/cikkek/prog_langs/web_langs/php/06/index.htm
57. http://www.prog.hu/cikkek/prog_langs/web_langs/php/07/index.htm
58. http://www.perl.com (a Perl hivatalos oldala)
59. http://www.flensburg-treff.de/thema/januar00/perltut.html (rövid bevezető a nyelvbe)
60. http://www.perlreference.com (referenciafüzetek)
61. http://www.solutionsoft.com/DL_PerlBuilder.htm (egy IDE környezet Perl-hez)
62. http://hoohoo.ncsa.uiuc.edu/cgi/security.html (a CGI biztonsága)
63. http://www.improving.org/paulp/cgi-security/safe-cgi.txt (biztonságos CGI)
64. http://Stars.com/Authoring/Scripting/Security (szkriptek biztonsága)
65. http://www.perl.com/CPAN-local/doc/FAQs/cgi/perl-cgi-faq.html
66. http://www.apache.org (az Apache hivatalos oldala)
67. http://httpd.apache.org (dokumentációk)
68. http://www.amon.hu (webprogramozás)
69. http://www.omikk.hu
70. http://www.wdvl.com (számos leírás, ismertető)
71. http://www.rfc-editor.org (RFC lista)

Bibliográfia

(A diplomamunkában fel nem használt, de a témához kapcsolódó szakirodalmat és WWW címeket tartalmazza válogatva).

Könyvek:

1. Bartók Nagy János – Laufer Judit: UNIX felhasználói ismeretek. Bp.: Openinfo K., 1994.
2. Henczi Béla – Molnár Hajnalka: Tanuljunk Linuxot! Zalaegerszeg: Szerzői kiadás, 2000. Mell.: 1 db CD-ROM
3. Jardin, Cary A.: Java electronic commerce sourcebook: all the software and expert advice you need to open your virtual store. New York: John Wiley & Sons, 1997. Includes 1 CD-ROM
4. Kernighan, B. W. – Pike, R.: A UNIX operációs rendszer. 3. kiad. Bp.: Műszaki Könyvkiadó, 1994.
5. Kupcsikné Fitus Ilona: Adatbázisok: Példatár. Bp.: LSI Oktatóközpont, 1996.
6. Linux-Mandrake felhasználói kézikönyv. / ford.: Bán Szabolcs et al. Bp.: MandrakeSoft, 1999.
7. Nagy Péter: JavaScript. Bp.: Kalibán Bt. és Kiskapu Kft., 1997.
8. Oracle 6.0: Referencia kézikönyv. / Juhász István et al. Bp.: IQSoft, 1992.
9. Szabó Bálint: LINUX az otthoni PC-n. Bp.: LSI Oktatóközpont, 2000.
10. Szabó Bálint: LINUX lépésről lépésre. Bp.: LSI Oktatóközpont, 1999.
11. Tanenbaum, Andrew S.: Számítógéphálózatok. Bp.: Panem Könyvkiadó Kft., 1999.

Folyóiratcikkek:

12. Bagoly József Péter: Az erő vele van!: FreeBSD 4.0 = CHIP, 13. évf. 2001. 2. sz. 105–106.p.
13. Fazekas László: Luxus Web-stúdió = CHIP, 13. évf. 2001. 3. sz. 122–124.p.
14. Horváth Zsolt: Egyszerűen nagyszerű: Webmin – adminisztrátori mindenes = CHIP, 12. évf. 2000. 12. sz. 156–158.p.
15. Horváth Zsolt: Itt a KDE 2.0! = CHIP, 12. évf. 2000. 12. sz. 30.p.
16. Horváth Zsolt: Lássuk, mennyire terhelt a Web-szerver! = CHIP, 12. évf. 2000. 11. sz. 26.p.

17. Kósa Attila: Linux a hazai vállalati felhasználói körben. = Linuxvilág, 2. évf. 2001. 2–3.sz. 12–13.p.
18. Novák Áron: Kereső saját használatra = Internet Kalauz, 6. évf. 2001. 2. sz. 50–51.p.
19. Qualline, Steve: Ez aztán a fejlődés!: Cikkünkben a vi egy okosabb változatát mutatjuk be. = Linuxvilág, 2. évf. 2001. 2–3. sz. 80–83.p.
20. Spányik Balázs: Szegény ember Home Site-ja = CHIP, 13. évf. 2001. 1. sz. 12.p.
21. Sweet, David: Andamooka: A nyílt forrású programfejlesztés hatására nyílt könyvek jelennek meg. = Linuxvilág, 2. évf. 2001. 2–3. sz. 84–85.p.
22. Szendi Gábor: DHTML-iskola (2.) = Internet Kalauz, 6. évf. 2001. 1. sz. 53–55.p.
23. Szendi Gábor: DHTML-iskola (3.) = Internet Kalauz, 6. évf. 2001. 2. sz. 47–49.p.
24. Szendi Gábor: DHTML-iskola = Internet Kalauz, 5. évf. 2000. 12. sz. 52–54.p.
25. Szendi Gábor: Pozicionálás DHTML-ben = Internet Kalauz, 6. évf. 2001. 3. sz. 49–51.p.

WWW címek:

26. http://debian.inf.elte.hu/linux_doksi/dolgozat.htm (mindent a Linuxról)
27. http://developer.netscape.com/docs/manuals/communicator/jsguide/js1_2.htm (JavaScript 1.2)
28. http://freeweb.hu (CGI, PHP futtatása)
29. http://freshmeat.net (szabad szoftverek)
30. http://galaxy.einet.net/galaxy/Engineering-and-Technology/Computer-Technology/Programming-Languages.html (számos programozási nyelv)
31. http://hoohoo.ncsa.uiuc.edu/cgi/env.html (CGI környezeti változók leírása)
32. http://hotwired.lycos.com/webmonkey/programming (webprogramozás)
33. http://java.sun.com/docs/books/jls/index.html (a Java nyelv specifikációja)
34. http://java.sun.com/docs/codeconv/html/CodeConvTOC.doc.html (kódolási konvenciók)
35. http://java.sun.com/j2se/javadoc/index.html (számos dokumentáció)
36. http://leeloo.kiskapu.hu/nil/Sql/index.htm (SQL oktató)
37. http://linux.index.hu (az egyik legjobb magyar Linux-os portál)
38. http://mlf.linux.rulez.org/mailman/listinfo (levelezési listák)
39. http://people.inf.elte.hu/horvaths/craft
(statisztika a Magyarországon használatos webszerverekről)
40. http://php4.x3.hu (PHP)
41. http://slashdot.org (friss hírek)
42. http://sourceforge.net (szabad szoftverek)

43. http://w3.one.net/~jhoffman/sqltut.htm (SQL oktató)
44. http://web.mit.edu/pbh/www/language.html (különböző programozási nyelvek, többek között Java, Perl)
45. http://weblabor.externet.hu (webprogramozás)
46. http://www.dansteinman.com/dynduo (dinamikus HTML)
47. http://www.devshed.com (cikkek többféle témában: MySQL, PHP, stb.)
48. http://www.dhtmlzone.com/index.html (dinamikus HTML)
49. http://www.enteract.com/~lspitz/linux.html (hogyan tegyük biztonságosabbá Linux rendszerünket)
50. http://www.freeweb.hu/e-vilag (Linux-os folyóirat)
51. http://www.geekfinder.hu (napi Linux hírek)
52. http://www.htmlhelp.com/reference/css
53. http://www.hwsw.hu (naponta friss hírek)
54. http://www.ilook.fsnet.co.uk/index/oracle.htm (SQL oktató)
55. http://www.linux.hu (napi Linux hírek)
56. http://www.linuxgazette.com (a Linux Journal havonta megjelenő kiadványa)
57. http://www.linuxguruz.org (cikkek többféle témában: adatbázis, programozás, stb.)
58. http://www.linuxjournal.com (Linux Journal magazin)
59. http://www.linuxnewbie.org (Linux - nem csak kezdőknek)
60. http://www.linuxvilag.hu (az amerikai Linux Journal magyar tarslapja)
61. http://www.linuxvoodoo.com (számos hasznos információ)
62. http://www.mycgiserver.com (szervlet, JSP futtatásának a lehetősége)
63. http://www.nexus.hu/hever (magyar Python honlap)
64. http://www.rpmfind.net (RPM csomagkereső)
65. http://www.tpj.com (The Perl Journal)
66. http://www.w3.org (WWW Consortium)
67. http://www.w3.org/MarkUp (HTML)
68. http://www.w3.org/Protocols (HTTP protokoll)
69. http://www.w3.org/TR/REC-CSS1
70. http://www.w3.org/TR/REC-CSS2
71. http://www.webreference.com (referenciák)
72. http://www.webreference.com/dhtml (dinamikus HTML)
73. http://www.worldwidemart.com/scripts (CGI szkriptgyűjtemény)