A PostgreSQL és a PHP (2. rész)

Varázsoljuk PostgreSQL-lel tárolt adatbázisainkat a Webre, és egyúttal kóstoljunk bele napjaink egyik leggyorsabban fejlődő programozási nyelvébe, a PHP-ba.

Sorozatunk előző részében telepítettük a PHP-t és beállítottuk a PostgreSQL-t, hogy PHP-ból elérhessük az adatbázisokat. Most megismerkedünk a PostgreSQL biztonsági beállításaival, a PHP postgreses adatbázisokat támogató függvényeivel, írunk egy rövid webes alkalmazást, és a cikk végén megnézzük, hogy miként kell eljárni, ha a PostgreSQL egy újabb változatára szeretnénk átállni.

Az előző cikk végén lévő példában (test.php3) kapcsolódni próbáltunk a php_db adatbázishoz a php_user felhasználó nevében, a heureka jelszóval:

```
<?php
$connection = pg_Connect (
  "dbname=php_db port=5432 user=php_user
  password=heureka");
?>
```

A PostgreSQL biztonsági beállításai

Mielőtt tovább ismerkednénk a PHP-val, egy kis kitérőt kell tennünk a PostgreSQL felhasználóinak jelszavai kapcsán. Próbáljunk rossz jelszót megadni a test.php3 fájlban vagy változtassuk meg a php_user jelszavát psql-ben az `ALTER USER php_user WITH PASSWORD kukucs;` paranccsal. Elképzelhető, hogy a PostgreSQL így is engedi a hozzáférést az adatbázishoz. Ha ezt nem akarjuk, módosítsuk a `/etc/postgresql/pg_hba.conf` fájlt. Itt eltérő biztonsági szinteket (felhasználó-ellenőrzési módszereket) állíthatunk be a különböző helyekről érkező kérésekhez. Ha egy helyben megbízunk (`trust` értéket állítunk be) – ez az alapértelmezett biztonsági szint helyi hálózatunkra, – nem történik semmilyen jelszóellenőrzés. Ha `password` a beállított érték, a PostgreSQL ellenőrzi a jelszót, de a jelszavak sima szövegként haladnak a hálózaton. Ha `crypt`-et adunk meg, a jelszavak titkosítva utaznak a hálózaton, de vigyázzunk, a `/var/lib/postgres/data/pg_pwd` fájlban (*10. lista*) a jelszavak kódolatlanul tárolódnak. Ilyenkor a `pg_pwd`-ben levő jelszót kódolja a rendszer, majd ezt a kódolt jelszót hasonlítja össze a belépni kívánó felhasználó által megadott jelszó kódolt változatával. A rendszer további hiányosságairól bővebben a `postgresql-doc/README.passwords` fájlban olvashatunk.

Ehhez kapcsolódik még néhány fontos dolog: ha nem `trust`-ot állítottunk be a helyi hálózatunkra, a psql-t `psql -u` paranccsal kell indítanunk, különben nem kér jelszót, így azután nem férhetünk hozzá az adatbázisokhoz. Ekkor azok a felhasználók, akiknek nem állítottunk be jelszót, többé nem tudnak bejelentkezni. Ilyen a Postgres is, tehát mindig állítsunk be jelszót a felhasználónak. Van egy *do.maintenance* karbantartó parancsfájl, amit a cron démon futtat a Postgres-felhasználó nevében (Debianon a /etc/cron.d/postgres fájl segítségével), ezt kell kibővíteni a `-u postgres -p jelszó` szöveggel.

A másik említésre méltó dolog, hogy azok a felhasználóink, akik maguk is létrehozhatnak felhasználókat, teljes jogú urai a PostgreSQL-nek, azaz minden felhasználó jelszavát (így a Postgresét is) megváltoztathatják, sőt, megnézhetik a *pg_pwd* fájlban (10. lista).

Adatelérés PHP3-mal

Most már minden készen áll ahhoz, hogy elérjük a PostgreSQL adattáblákat, és megjelenítsük a tárolt adatokat a weboldalunkon. Ahhoz, hogy ezt kipróbáljuk, hozzunk létre táblákat és engedélyezzük egy felhasználónak az elérést! Hozzuk létre az allatok táblát – amelyben az állatok és tulajdonosaik neve található – a

```
create table allatok (allatnev char(8), tulaj
       char(8));
```

utasítással, majd tegyünk bele egy sort az

```
insert into allatok values ('Méhecske','Szilvia');
```

paranccsal. Ezután adjunk jogot a php_user felhasználónak a táblához:

```
grant select on allatok to php_user;
```

(A lemezmellékletben szereplő `php_proba.sql` fájlban megtalálhatók ezek az utasítások, elég a psql-ben kiadni a `\i php_proba.sql` utasítást.) A grant utasítás általános alakja `GRANT engedélyek ON táblák TO felhasználó;` ahol az engedélyek ALL, SELECT, INSERT, UPDATE, DELETE, illetve RULE. Ezek a szavak megfelelnek az azonos nevű SQL utasítás végrehajtásához való joggal, kivéve az ALL-t – ami minden jogot megad, és a RULE-t – amely a szabályok létrehozását engedi, és nem szabványos SQL utasítások. A felhasználó lehet PUBLIC, ez az összes felhasználót takarja, vagy egy csoportnév, amely a felhasználók egy csoportjára utal. Erről bővebb tájékoztatást a PostgreSQL leírásban találhatunk.

Egy adatbázis tábláira adott jogosultságokat a `\z psql` parancs kiadásával nézhetünk meg (*11. lista*). A jobb oldali oszlop egy sorában levő értéksorozat jelzi, kinek milyen jogokat adtunk felhasználó/csoport=jogok formában. A jogokat karaktersorozat jelképezi, ahol r:SELECT, w:UPDATE/DELETE, a:INSERT, r:RULE lehet. Ha az egyenlőségjel bal oldalán nincs semmi, akkor a sor mindenkire érvényes jogokat mutat.

Miután elkészült a tábla, tovább próbálgathatjuk a PHP3-at. A test2.php3 (*12. lista*) program kapcsolódik az adatbázishoz. A `pg_Connect()` függvény eredménye egy szám, amely azonosítja a kapcsolatot. Több adatbázishoz is kapcsolódhatunk, ekkor mindegyik kapcsolatnak más-más száma lesz. A további PHP-PostgreSQL függvényekben ezt a kapcsolatazonosítót kell használnunk.

A `pg_Exec (kapcsolat_azonosító, SQL-szöveg);` függvény az SQL-szövegben levő SQL utasítást hajtatja végre. A visszaadott szám azonosítja a lekérdezést. Hiba esetén 0, eredményt nem szolgáltató SQL utasítások (INSERT, DELETE, UPDATE stb.) esetén

10. lista A /var/lib/postgres/data/pg_pwd fájlban normál szövegként tárolódnak a jelszavak

```
billgates     1002  t   t   t   t   \N   \N
proba         1003  t   t   t   t   \N   \N
phpuser       1004  f   f   f   f  heureka \N
jeno          1006  f   f   f   f   \N   \N
php_user      1005  f   f   f   f  heureka \N");
```

11. lista A \z psql utasítással a táblákhoz rendelt jogosultságokat nézhetjük meg

```
php_db=>    \ z
Database      =    php_db
+————————+————————————————————————+
| Relation |  Grant/Revoke Permissions  |
+————————+————————————————————————+
| allatok  |     !"=","php_user=r"}     |
| szemely  |     {"="}                  |
+————————+————————————————————————+
```

12. lista A test2.php3 program. Kiírja egy SQL tábla első sorának első mezőjét

```
<?php
 $connection = pg_Connect (
      "dbname=php_db port=5432 user=php_user
       ➥password=heureka");

 $result = pg_Exec($connection,
                   "SELECT * FROM allatok;");
 $row = pg_fetch_row ($result, 0);
 print "A tábla elso sorának elso eleme: <B>"
       . $row[0] . "</B>\n";
 pg_Close($connection);
?>
```

1, lekérdezéseknél egynél nagyobb ez az érték. A későbbiekben, ha egy lekérdezés eredményét el akarjuk érni, ezt az azonosítót kell használnunk.

A `pg_fetch_row (lekérdezés_azonosító, sorszám);` függvény az eredmény *sorszám*-adik sorát adja vissza tömbként. A tömb elemei a sor mezői. Az első eredménysor a 0-ik sorszámú. A `print` paranccsal vastagon szedve kiírjuk a `$row` tömb nulladik elemét. Végül bezárjuk a kapcsolatot a `pg_Close (kapcsolat_azonosító);` függvénnyel.

A test3.php3 program (*13. lista*, a sorszámok csak a magyarázat kedvéért szerepelnek) végigolvassa a táblát és HTML táblázatként kiírja. A PHP-program a 3. sornál kezdődik. A kapcsolat létesítése és az SQL utasítás kiadása után (4–6. sorok) kiírjuk a táblázat fejlécét, piros hátterű cellákban. A `pg_NumFields (lekérdezés_azonosító)` függvény megadja, hány oszlopa van az eredménytáblázatnak. A 9. sorban indított számlálásos ciklussal végigmegyünk az oszlopokon és a `pg_fieldname (lekérdezés_azonosító, oszlop_sorszám)` függvénnyel – ez az oszlopok nevét adja vissza – a 10–11. sorokban kiírjuk a fejlécet. A `pg_numrows (lekérdezés_azonosító);` függvény

13. lista A test3.php3 program. Egy teljes eredménytábla kiírása

```
1 A tábla sorai:<P>
 2 <TABLE BORDER=1 BGCOLOR=YELLOW>
 3 <?php
 4  $connection = pg_Connect
 5    ("dbname=php_db port=5432 user=php_user
       ➥password=heureka");
 6  $result = pg_Exec($connection,
       "SELECT * FROM allatok;");
 7
 8  print '<TR>';
 9  for($i=0; $i<pg_numfields($result);$i++) {
10    print '<TD BGCOLOR=RED><B>'.
11       pg_fieldname($result,$i)."</B></TD>";
12  };
13  print '</TR>';
14
15  for($i=0; $i<pg_numrows($result); $i++) {
16    $row = pg_fetch_row ($result, $i);
17    print '<TR><TD>'
   ➥.implode('</TD><TD>',$row)."</TD></TR>\n";
18  };
19
20  pg_Close($connection);
21 ?>
22 </TABLE>
```

14. lista A test4.php3 program. Az include() parancs bemutatása

```
<HTML>
<HEAD>
<TITLE>test4</TITLE>
</HEAD>
<BODY bgcolor=black text=white>
<?php
 include("test4.inc");
 print "<H1><CENTER>Helló</CENTER></H1>\n";
 include("test4.inc");
?>
</BODY>
</HTML>
```

15. lista A test4.inc fájl. Példa include-fájlra

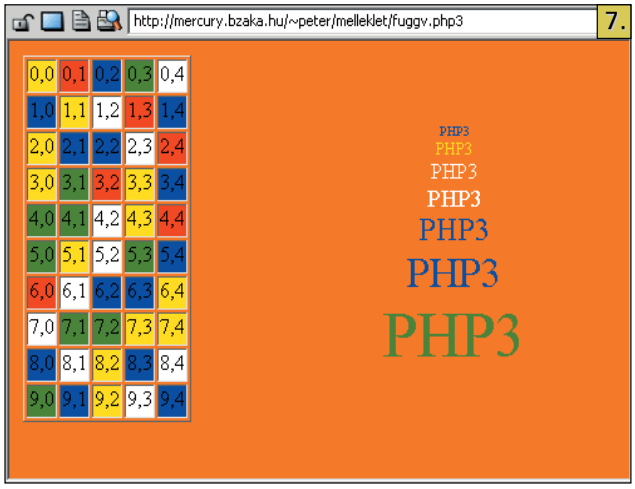
```
<TABLE bgcolor=red width=100%>
<TR><TD><CENTER>Szia</CENTER></TD></TR>
</TABLE>
```

visszaadja az eredménytáblázat sorainak a számát. Ennek segítségével (15. sor) egy számlálásos ciklussal olvassuk be a sorokat a $row tömbbe (16. sor), ezt az `implode()` függvénnyel alakítjuk át karakterláncca és írjuk ki (17. sor).

16. lista A fuggv.php3 program

```
<?php
include("fuggv.inc");
?>
<HTML><HEAD></HEAD>
<BODY BGCOLOR=ORANGE>
<TABLE BORDER=0 WIDTH=100%>
 <TR>
  <TD WIDTH=50%>
    <TABLE BORDER=1>
       <?php MyTable(); ?>
    </TABLE>
  </TD>

  <TD>
    <CENTER>
    <?php PHP3_Caption(1,7); ?>
    </CENTER>
  </TD>
 <TR>
</TABLE>
</BODY>
</HTML>
```



Hasznos PHP-trükkök

Include-fájlok, függvények, űrlapok

Mielőtt megírnánk egy „komolyabb” alkalmazást, meg kell ismerkednünk a nyelv néhány fontos tulajdonságával, ezért átmenetileg szakadjunk el az adatbázisoktól.

Elsőként megemlíteném, hogy PHP-programunkat lehetőségünk van több fájlban tárolni. Így a különálló részeket egyszerűen használ-

17. lista A fuggv.inc fájl. Példa függvényekre

```
<?php

srand((double)microtime()*1000000);

function random_color() {
  $colors=array('white','red','green',
                'yellow','blue');
  return $colors[rand(0,count($colors)-1)];
};

function MyTable() {
  for($i=0; $i<10; $i++) {
    print '<TR>';
    for($j=0; $j<5; $j++) {
       print '<TD BGCOLOR='
             ➥.random_color().">$i,$j</TD>";
    };
    print "</TR>\n";
  };
};

function PHP3_Caption($minsize,$maxsize) {
   for($i=$minsize; $i<=$maxsize; $i++) {
     print "<BR><FONT SIZE=$i COLOR="
           ➥.random_color().
            ">PHP3</FONT>\n";
   };
};
?>
```

hatjuk későbbi programjainkban, és kisebb mérete következtében a forrás is olvashatóbb lesz. A részeket beilleszteni az `include (fájlnév);` utasítással lehet. Erre példa a test4.php3 (*14. lista*) program és a test4.inc (*15. lista*) fájl. A test4.inc fájl piros hátterű táblázatot rajzol egyetlen cellával, melynek közepén a *Szia* szó szerepel. Ezt a fájlt fűzzük be kétszer test4.php programunk szövegébe, a *Helló* szó kiírása előtt és után (*6. kép*).

Az include-fájlok nevének nem kötelező .INC-re végződni, az bármi lehet, akár .PHP is. A php-végződés előnye, hogy az Internetről láthatatlan a tartalma, ugyanis amikor valaki megpróbálja letölteni, a webkiszolgáló értelmezi a fájlt, és csak az abban lévő kimenet tartalma fog megjelenni a böngészőben. Azért használok mégis .INC végződést, hogy könnyen megkülönböztethetők legyenek az include-fájlok a programfájloktól.

Függvények írása sem nehéz PHP-ban. A fuggv.php3 program (*16. lista*) meghívja a `MyTable ()` és a `PHP3_Caption ()` függvényeket (a program eredménye a *7. képen* látható). Mindkét függvény megvalósítása a fuggv.inc fájlban (*17. lista*) található. Az `srand ()` paranccsal beállítjuk a véletlenszám-előállító kezdőértékét. A `function random_color ()` részben egy véletlen színt választó függvény van, ami a return utasítással ad vissza értéket a hívó programnak. A `function MyTable ()` kezdetű blokkban két számlálásos ciklussal írjuk ki a 7. kép bal oldalán látható táblázatot. Itt látunk példát a `random_color ()` függvény használatára is. A `PHP3_Caption ($minsize,$maxsize)` függvény készíti el a 7. kép jobb oldalát. A két átadott érték a PHP3 szöveg méretének alsó és felső korlátja. A PHP-nyelv megengedi a változók cím és érték szerinti használatát, alapértelmezett (default) értékének megadását és változó számú átadott érték kezelését is.

18. lista A form.html fájl egy egyszerű űrlapot valósít meg

```
<HTML>
<HEAD>
<TITLE></TITLE>
</HEAD>
<BODY>

<FORM action="mutat.php3" method="post">

<P>Írj be valamit:
<INPUT type="text" name="szoveg1">
<INPUT type="text" name="szoveg2">

<P><INPUT type="submit" value="Elküldés">
<INPUT type="reset" value="Törlés">

</FORM>
</BODY>
</HTML>
```

19. lista A mutat.php3 program.
Űrlapmezők neveinek és értékeinek kiírása

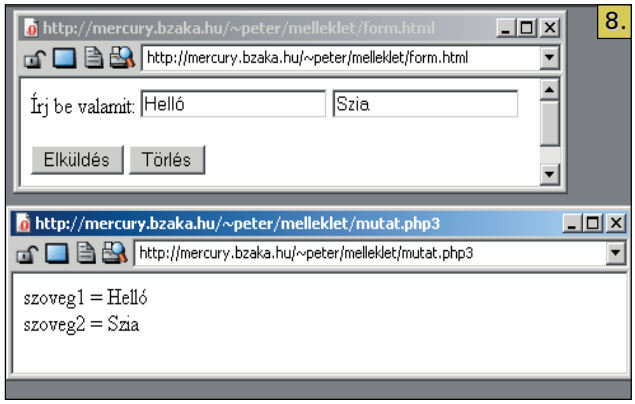
```
<HTML>
<HEAD>
<TITLE></TITLE>
</HEAD>
<BODY>
<?php
 reset ($HTTP_POST_VARS);
 while (list($key, $val)=
each($HTTP_POST_VARS)) {
    print "$key = $val<BR>\n";
 };
?>
</BODY>
</HTML>
```

Erről bővebben a PHP-leírás *functions.arguments.html* oldalán olvashatunk.

Szükségünk lesz űrlapok kezelésére is. Legyen az űrlapunk a form.html fájl (*18. lista*). Ebben két beviteli mező van, szoveg1 és szoveg2 néven, valamint elküldő- és törlőgomb. Elküldéskor a mutat.php3 program kapja meg a beviteli mezők neveit és értékeit. PHP3-ban a POST eljárással kapott név–érték párokat a `$HTTP-POST-VARS` tömbön keresztül érhetjük el; ha GET-et használtunk, akkor a `$HTTP-GET-VARS` tömb áll a rendelkezésünkre. Ezek úgynevezett társított vagy nevesített (asszociatív) tömbök, ezek lényege, hogy a tömb első oszlopa változók neveit, a második pedig az adott változóhoz tartozó értékeket tartalmazza. A tömb egy értékét a hozzá tartozó névvel is indexelhetjük. A tömb adatait emellett elérhetjük a 0, 1, … indexek használatával is, ilyenkor a páros indexekre a neveket kapjuk, a páratlanokra az értéket. Ha például a szoveg1 beviteli mezőbe a *Hello* szót írtuk, elküldés után a mutat.php3 programban a `$HTTP_POST_VARS ["szoveg1"]` kifejezéssel kaphatjuk meg a Hello értéket. A tömböknek van belső mutatójuk is, ennek segítségével az elemeket indexelés nélkül, sorban egymás után is elérhetjük.

20. lista A valaszt.php3 program. Választólistás űrlap

```
 1 <?php
 2  $connection = pg_Connect(
 3       "dbname=php_db port=5432 user=php_user
        ➥password=heureka");
 4  $result = pg_Exec($connection,
              "SELECT * FROM allatok;");
 5  pg_Close($connection);
 6
 7  print '<FORM action="egy.php3"
        ➥method="post">'."\n";
 8
 9  print '<SELECT size="6"
        ➥name="allatok">'."\n";
10  for($row=0; $row < pg_NumRows($result);
              $row++){
11    print '<OPTION VALUE='.pg_Result
        ➥($result, $row, 1).'>';
12    print pg_Result ($result, $row, 0) .
        ➥' gazdája ' .
13          pg_Result ($result, $row, 1) .
                  ➥'</OPTION>' . "\n";
14  }
15  print '</SELECT>'."\n";
16
17  print '<P><INPUT type="submit"
        ➥value="Ezt választom">'.
18        '<INPUT type="reset"
        ➥value="Választás törlése"></FORM>';
19
20 ?>
```



A PHP beállításától függően az űrlap beviteli mezői elérhetők lehetnek egyszerű változóként is, azaz a szoveg1 beviteli mező értéke lekérdezhető a `$szoveg1` globális változón keresztül. Sokan nem szeretik ezt a szolgáltatást, ugyanis könnyű összekeverni a program által egyébként is használt globális változókat az űrlapokon keresztül kapottakkal. Erről bővebben a leírás *configuration.html* oldalán olvashatunk a track_vars és a register_globals részeknél.

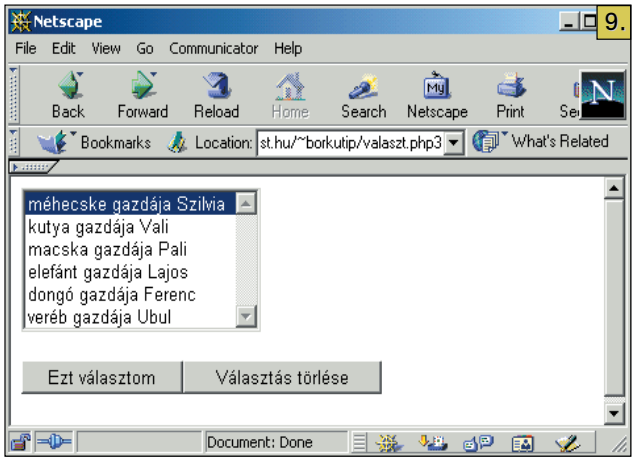
Nézzük meg a mutat.php3 programot (*19. lista*). A `reset ($HTTP_POST_VARS);` sorral a tömb belső mutatóját a tömb elejére állítjuk. A `while (list($key, $val) = each ($HTTP_POST_VARS))` kezdetű blokk egy elöl tesztelős ciklus, amelynek magjában kiírjuk a `$key` és a `$val` változók értékeit. A `$key` tartalmazza az űrlapmező nevét, a `$val` pedig az értékét.

21. lista Az egy.php3 program egyetlen értéket dolgoz fel

```
<?php
 $val = $HTTP_POST_VARS["allatok"];

 $connection = pg_Connect("dbname=php_db
  ➥port=5432 user=php_user password=heureka");
 $result = pg_Exec($connection,
    "SELECT * FROM szemely
          ➥WHERE nev='$val';");
 pg_Close($connection);

 print pg_Result ($result, 0, 0).','.
       pg_Result ($result, 0, 1).','.
       ((pg_Result ($result, 0, 2)=='f') ?
            'nő' : 'férfi') ."\n";
?>
```



Az az izgalmas rész, ahol a `$key` és `$val` megkapja értékét, ez pedig a ciklusfeltételben történik meg. A `list (változólista)` kulcsszó hatására egyben kezelhetünk változókat, úgy, mintha egy tömb lennének. Az `each (tömb)` kiválasztja a tömbből a belső mutatója által mutatott név–érték párt és hátrébb állítja a mutatót. Így a `list ($key, $val) = each ($HTTP_POST_VARS)` értékadásra a `$key` és `$val` változók megkapják az általunk kívánt értékeket. Az elöl tesztelős ciklus miatt az űrlap mezőjét kiírjuk, nem kell tudnunk, hogy pontosan mi volt a mezők neve és hány volt az űrlapon. Futtatásának eredményét láthatjuk a *8. képen*.

Egy alkalmazás

Most már visszatérhetünk az adatbázis-kezeléshez. Legyen a feladatunk az, hogy a példaadatbázis `allatok` táblájában levő állatok gazdáiról tudjunk meg adatokat. A programunk felkínálja az összes bejegyzést a táblából. A felhasználó válasszon innen egy sort, és az ott levő gazdához tartozó adatok jelenjenek meg egy újabb weboldalon.
Elsőként nézzük meg a valaszt.php3 program eredményét (*9. kép*).
A böngészőben egy lista jelenik meg az állatokról és gazdáikról. A képernyő alján két gomb található az elküldésre és a választás törlésére.
A program (*20. lista*) sorait megszámoztam. Tekintsük át a parancsokat! A lista első része (1–5. sorok) csatlakozik az adatbázishoz és lekéri az összes sorát. A 7. sor eredménye lesz a weboldal első sora, megnyit egy űrlapot, amely az adatokat az egy.php3 programnak küldi el. A lényeges rész a 9–15. sorokban van. Itt állítja elő a program a választólistát. A 10. sorban végigmegyünk az eredménytábla összes során és mindegyik sorból készítünk egy HTML sort:

```
<OPTION VALUE=gazda neve>állat gazdája gazda
neve</OPTION>
```

Így az egy.php3 programnak valamelyik állattartó nevét fogjuk elküldeni. A 17. és 18. sorok egy submit és egy reset gombot állítanak elő. Az egy.php3 program (*21. lista*) a $val változóba teszi az *allatok* névhez tartozó értéket a `$HTTP_POST_VARS` tömbből. Ezután kiválasztja azokat a sorokat a *szemely* táblából, ahol a *nev* értéke megegyezik a `$val` változó értékével. Azért kell `$val`-t aposztrófok közé tenni, mert a tömb *nev* oszlopa szöveges típusú adatokat tartalmaz. A program utolsó része kiírja az eredménytábla nulladik sorát. A `(pg_Result ($result, 0, 2) == 'f') ? 'nő' : 'férfi')` eredménye 'nő' vagy 'férfi' – attól függően, hogy a harmadik mező értéke 'f' lesz-e. A program kimenete egyetlen sor a böngészőben, például: Szilvia ,1978-03-14, nő.

Áttérés új PostgreSQL-változatra

Nagyobb váltásoknál módosul az adatbázisok tárolási formátuma. Mivel az újabb változatok nem mindig működnek a régi adatbázisokkal, ki kell menteni az adatbázisokat, letörölni az adatbázis-könyvtárat (általában a `/var/postgres/data` könyvtár), és meg kell szabadulni a régi Postgrestől, telepíteni kell az újat, majd beüzemelni és visszaállítani az adatbázisokat. Ezalatt természetesen meg kell akadályoznunk, hogy az adatbázisok változzanak. Legegyszerűbb, ha leállítjuk a PostgreSQL-t. (A folyamatról olvashatunk a /usr/doc/postgresql-doc/README.Debian.migration.gz fájlban.)
Frissítsünk mi is a 6.3.2-ről (ami a Debian Slink-ben található) a 6.5.3-ra (ez a Potato-változat része).

Az adatbázisok mentése és visszatöltése

Állítsuk le a Postgrest a `/etc/init.d/postgres stop` parancscsal. Rendszergazdaként át tudunk váltani a postgres felhasználóra a `su - postgres` utasítással. Ezután adjuk ki a `postgresql-dump -t db.out` parancsot. A db.out fájl a /var/postgres könyvtárban jön létre. Belenézhetünk: átlagos szöveges fájl, amely a psql számára készült. Másoljuk biztonságos helyre, például a saját könyvtárunkba. Éles rendszereken természetesen mentsünk le mindent a következő lépések végrehajtása előtt!
Töröljük a `/var/postgres/data` könyvtárat.
Töröljük a postgresql csomagot: `dpkg --remove postgresql`
Telepítsük az új PostgreSQL-t. Telepítés alatt beállíthatjuk a használt dátumformátumot és karakterkódolást. Ha nem történt hiba, létrejön a `/var/postgres/data` könyvtár, benne a rendszeradatbázisok és a *template1* adatbázis.
Hozzuk létre régi adatbázisainkat. Lépjünk be postgres néven: `su - postgres`, és adjuk ki a `psql -e elérési_út/db.out` parancsot. Ha valami hibát veszünk észre, elképzelhető, hogy javítani kell a db.out fájlon. Ekkor kezdjük újra a folyamatot: állítsuk le a Postgrest, töröljük a data könyvtárat, lépjünk be postgresként, majd adjuk ki az `initdb` parancsot, ez létrehozza a data könyvtárat a szükséges adatbázisokkal.
A listák megtalálhatók a 14. CD-melléklet Magazin/Cikkekhez könyvtárában.

Borkuti Péter (borkuti@freemail.hu),
matematika-informatika szakos tanár, rendszergazda, informatikus, rendszerépítő és programozó.