© Kiskapu Kft. Minden jog fenntartva

A PostgreSQL adatbázis-kezelő használata

Egy programozó szemszögéből vizsgálva…

A PostgreSQL (továbbiakban PG) korszerű objektum-relációs adatbázis-kezelő rendszer, amit a Berkeley Egyetem Számítástudományi tanszékének vezetésével fejlesztenek. A program támogatói között olyan híres szervezetek is részt vállalnak, mint az amerikai DARPA, ARO és NSF. A PG elődjének – az Ingresnek – a fejlesztése 1988 körül kezdődött. A jelenlegi PG 7.2-es változat támogatja az SQL92/SQL99 szabványokat, valamint lehetővé teszi az öröklődést, SQL-adattípusok, tárolt eljárások, kioldók (trigger), megszorítások, számlálók használatát, valamint a tranzakciókezelést (BEGIN WORK, COMMIT, ROLLBACK). A cikk célja, hogy az általános SQL-és programozási ismeretekre építve belépőt nyújtson a PG használatának megkezdéséhez.

Indulás a PG világába…

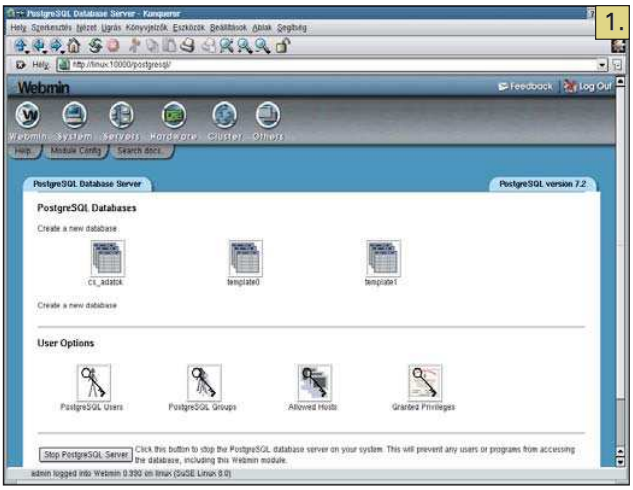
Az összes jelentős Linux-terjesztés tartalmazza a PG-t. Tekintettel arra, hogy ez egy nagyméretű alkalmazás, több csomagra szét van osztva, hiszen nem biztos, hogy minden összetevőre szükségünk lesz. A cikk mintapéldái SusSE Linux 8.0 alatt készültek, ahol az `rpm -qa | grep postgres` parancsot kiadva listázhatjuk ki a feltelepített PG-csomagokat:

```
postgresql-libs-7.2-86
postgresql-7.2-86
postgresql-contrib-7.2-86
postgresql-devel-7.2-86
postgresql-server-7.2-86
postgresql-tcl-7.2-86
postgresql-test-7.2-86
postgresql-tk-7.2-86
postgresql-jdbc-7.2-86
postgresql-odbc-7.2-86
```

Maga a PG-kiszolgáló és annak teljes leírása a `postgresql-7.2-86` csomagban található. A csomagok telepítése után javaslom megnézni, hogy engedélyezve van-e a TCP/IP-n keresztüli hozzáférés, illetve hogy az adatbázis-elérésre mely gépek jogosultak. A legegyszerűbb módszer erre a `su postgres` paranccsal átmenetileg Postgres-felhasználóvá válni, majd egy kapcsolók nélküli `cd` paranccsal belépni a Postgres ***saját*** könyvtárába. Itt találhatóak meg a ***pg_hba.conf*** és ***postmaster.opts*** fájlok (Linux-változattól függően ezek eltérőek lehetnek!).
Először nézzük meg a második fájl tartalmát, ami az alábbi egyetlen sorból áll:

```
/usr/bin/postmaster '-i' '-D/var/lib/pgsql/data'
```

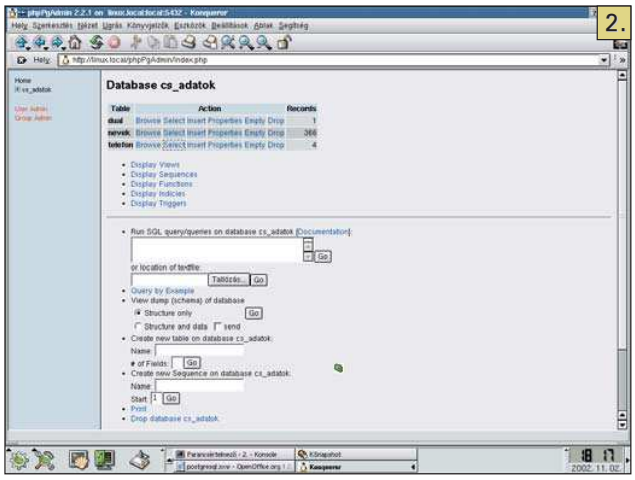
Ebből megtudhatjuk, hogy adatbázisaink a `-D` által jelölt könyvtárban helyezkednek el (az új adatbázisok is ide jönnek majd létre). A kis `"-i"` jelzi, hogy a postmaster kiszolgálófolyamat a TCP/IP-n keresztüli (azaz internet-) kapcsolódási kérelmeket is elfogadja, így ha ez a kapcsoló hiányzik, szúrjuk be a `-i`-t, majd a `/etc/init.d/postgresql restart` paranccsal indítsuk újra a PG-kiszolgálót.
A ***pg_hba.conf*** fájl tartalma határozza meg azokat a gépeket, amelyek adatbázisainkra bejelentkezni jogosultak. A fájlban megjegyzésbe tett mintasorok szerepelnek, így egy-egy új beállítósor beszúrása nem okozhat gondot.
Azok számára, akik a napi felügyeleti feladatokhoz jobban szeretik a grafikus felületet, a Webmin alkalmazást ajánlom, ami PG-adatbázis-kezelőnk Weben keresztüli felügyeletét is lehetővé teszi. A Webmin telepítése után böngészőnkben a következő címet adjuk meg: ➲ http://gépnév:10000. A ***felhasználónév és a jelszó*** megadása után bejelentkezik a Webmin indulóképe, ahol a ***Servers*** fülre, majd a PostgreSQL ikonra kattintva betöltődik a PG-t kezelő alkalmazás – ezt mutatja az *1. kép*.

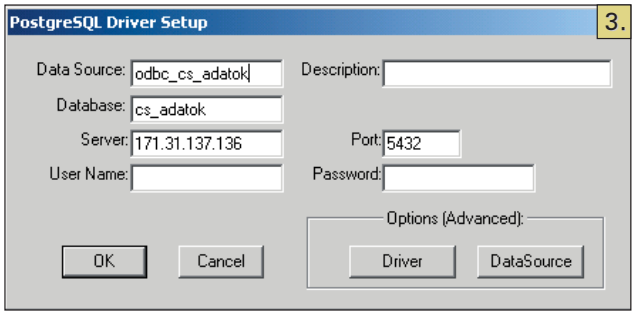


Ez a cikk nem a PG-kiszolgáló felügyeletéről, hanem annak programozásáról szól, ezért ezen a ponton abba is hagyjuk a telepítéssel és üzemeltetéssel kapcsolatos kérdések megválaszolását, és a következő lépésként inkább egy új adatbázist hozunk létre.

Egy új adatbázis létrehozása

A PG több segédprogramot (helyük általában a ***/usr/bin*** könyvtár) tartalmaz, amelyekkel minden feladatot el lehet végezni. Az alapértelmezett adatbázisgazda a `postgres`.
Új adatbázist a `createdb` paranccsal tudunk létrehozni, ennek neve legyen most `cs_adatok`. A szövegek alapértelmezésben SQL_ASCII kódban tárolódnak, de az adatbázis létrehozása során az ISO-8859-2 (Latin 2) karakterkódok használatát javaslom. Ez a beállítás elsősorban a Java-programok számára fontos, mert ezzel válik lehetővé az Unicode-ról való hibátlan oda-vissza történő karakterátalakítás. Itt az a gond, hogy az ő, ű karakterek nem képeződnek le az ISO-8859-1 kódlapra, hiszen ott nincs ilyen. Ez fordítva is igaz, azaz a kalapos ékezeteknek is lesz Unicode-beli párjuk, így azok nem a mi ő, ű karakte-reinkhez fognak rendelődni. A Latin 2 használatának további előnye, hogy ilyenkor nem kell kódátalakítást végezni az





ISO-8859-2-es kódlapon dolgozó ügyfél és kiszolgáló között. Nézzük meg adatbázis-létrehozó parancsunkat:

```
createdb -U postgres cs_adatok -E latin2
```

Érdemes még megjegyezni, hogy a `latin2` helyett az `unicode` is használható.
Ez a parancs a `-U` kapcsolóban tartalmazza azt a felhasználót, akinek jelenleg joga van PG-adatbázis-műveleteket végezni (természetesen további felhasználókat is felvehetünk, de ez már üzemeltetési kérdés). A jelszót azért nem kell megadnunk, mert éppen rendszergazdaként dolgozunk. Amennyiben a `su postgres` paranccsal egy új héjat kérünk, a `-U` kapcsoló minden parancsból elmaradhat, hiszen ezzel az operációs rendszer ekkor már azonosította a `postgres` felhasználót. Általában is igaz, hogy a PG elfogadja az operációs rendszer által már elvégzett felhasználói azonosításokat.
A PG több programozási nyelven (SQL, C/C++, plpgsql, pltcl, pltclu, plperl, plperlu, plpython) is biztosítja a tárolt eljárások és kioldók írását. Az adatbázis alapértelmezetten csak az SQL nyelvet ismeri.
Az Oracle-ben megszeretett PL/SQL nyelv alapján született meg a plpgsql nyelv (mindkettő az ADA leegyszerűsített változata), így ezt a lehetőséget mindenképpen érdemes kihasználni. Egy új tárolt eljárás készítését lehetővé tévő nyelvi modul telepítése a `createlang` paranccsal lehetséges. Tegyük elérhetővé a `cs_adatok` adatbázis számára a plpgsql nyelvet a következő parancs kiadásával:

```
createlang -U postgres -d cs_adatok plpgsql
```

Ezt követően az Oracle-programozók is azonnal ismerős programozási környezetben találják magukat.

A PSQL terminál felfedezése

A PG SQL konzolt a `psql` program valósítja meg, ami hasonló célokat szolgál, mint az Oracle *Plus. Kapcsolódjunk a most létrehozott `cs_adatok` adatbázishoz:

```
psql -U postgres cs_adatok
```

A sikeres kapcsolódást a következő képernyőtartalom igazolja vissza:

```
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

cs_adatok=#
```

A `psql` terminálon kétféle parancsot használhatunk. Az egyik egy bármilyen, közvetlenül kiadott SQL-parancs, amit a `psql` parancsjele után lehet beírni, és pontosvesszővel (;) kell lezárni. A másik parancscsoportot „slash” parancsoknak hívják, mert mindig visszaperjel (\) karakterrel kezdődnek, és olyan utasításokat fogalmaznak meg, amelyek magának a `psql` programnak szólnak.
A végrehajtható SQL-parancsokról a `\h`, míg a „slash” parancsokról a `\?` „slash” utasítások adnak részletes tájékoztatást.
Gyakorlásképpen hozzunk létre egy ***dual*** nevű táblát, hogy az Oracle-programozók még otthonosabban érezzék magukat a PG adatbázis-kezelőben:

```
cs_adatok=# create table dual (
cs_adatok(#    o1 varchar(10),
cs_adatok(#    o2 varchar(10)
cs_adatok(# );
CREATE
```

Látható, hogy az SQL-parancsokat több soron keresztül is gépelhetjük. Szúrjunk be egy új rekordot a ***dual*** táblába:

```
cs_adatok=# insert into dual values
            ('Valami1', 'Valami2');
INSERT 17045 1
```

Az egyes SQL-parancsokkal a továbbiakban nem foglalkozunk, hiszen az SQL-nek hatalmas irodalma van, és írásunknak nem az SQL ismertetése a célja. Nézzük meg inkább, hogy milyen további meglepetéseket tartogatnak a \ parancsok!
Sokszor szükség lehet rá, hogy az adatbázis-objektumok leírását megtekinthessük. Erre a szolgál a `\d` *`obj_nØv`* parancs (d=describe). Nézzük meg, mit mond ez az utasítás a most létrehozott ***dual*** táblára!

```
cs_adatok=# \d dual
                Table "dual"
 Column  |          Type          | Modifiers
---------+------------------------+-----------
 o1      | character varying(10)  |
 o2      | character varying(10)  |
```

© Kiskapu Kft. Minden jog fenntartva

© Kiskapu Kft. Minden jog fenntartva

1. lista

```
-----------------------------------------
-- Visszaadja az  sszesen fizetett alapd j
--  sszegØt,amelynek telefononkØnti d ja a
-- f ggvØny ØrtØke forintban
-----------------------------------------
create or replace function MyFunc(integer)
returns integer as '
declare
 s integer := 0;
begin
 // LekØrdezi Øs s-ben tÆrolja a telefonok
 // szÆmÆt
 select count(*) into s from telefon;
 // KiszÆm tja az  sszesen beszedheti
 // alapd jat
 s := ($1) * s;
 return s;
end;
' language 'plpgsql';


-----------------------------
-- BeszØr egy Øj telefonszÆmot
-----------------------------
drop function MyFunc2(varchar);
create or replace function
MyFunc2(varchar,varchar) returns integer as
'
declare
 s varchar;
 p varchar;
begin
  s := $1; p := $2;
  insert into telefon values ( s,  p);
  return 1;
end;
' language 'plpgsql';
```

Alapvető igény, hogy egy előre elkészített SQL-parancsfájlt végrehajthassunk. Legyen egy ***nevek_db.sql*** parancsfájlunk, aminek az első pár sora így néz ki:

```
create table nevek (
        honap    int2,
        nap      int2,
        nevnap   varchar(50)
);

insert into nevek values ('1','1','Fruzsina');
insert into nevek values ('1','2',''bel');
insert into nevek values ('1','3','GenovØva,
BenjÆmin');
...
```

Ezt a fájlt a `\i /opt/imre/sql/`*`nevek_db`*`.sql` paranccsal futtathatjuk le, ennek hatására létrejön a ***nevek*** tábla és töltődik fel adattal.
A psql-ben a pillanatnyi könyvtár fogalma is létezik, amit a `\cd` *`k nyvtÆr`* alakban adhatunk meg, elkerülve, hogy min-

2. lista

```
CREATE FUNCTION uj_telefon(varchar, varchar)
returns integer  AS '
INSERT INTO telefon VALUES ($1, $2);
select 1;
' LANGUAGE 'SQL';
```

3. lista

```
#include "postgres.h"
//  rtØk szerint
int duplazo(int i)
{
  return 2 * a;
}

// C m szerint
float8 *fduplazo(float8 *f)
{
  return 2.0 * a;
}
```

den parancsfájlt a teljes neve megadásával kelljen futtatni.
A psql programból a \q slash paranccsal léphetünk ki. Tegyük meg! Miért? Azért, hogy kipróbálhassuk a psql parancs -c kapcsolójának használatát, ami egy bármilyen SQL- vagy slash parancsot végrehajt, majd a psql program futása befejeződik. Ezen a ponton elérkeztünk oda, hogy héjprogramjainkba PG SQL-részleteket is betehetünk. Kezd izgalmassá válni, ugye? Nézzünk egy egyszerű héjparancsot, ami kilistázza a ***dual*** táblát:

```
linux:~ # psql -U postgres cs_adatok -c
"select * from dual"
   o1    |    o2
---------+---------
 Valami1 | Valami2
(1 row)
linux:~ #
```

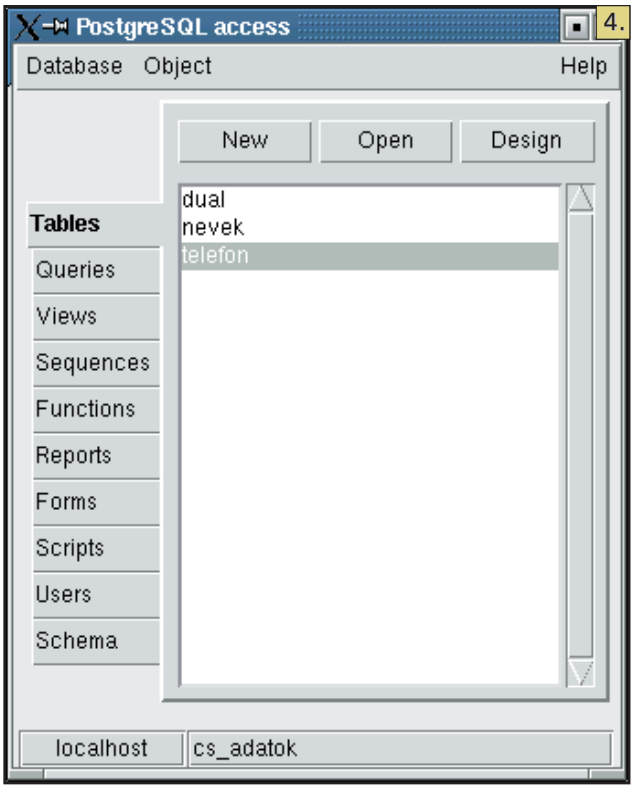
Természetesen ezzel a módszerrel egy teljes SQL-parancsfájl is lefuttatható. Az előbbi select parancsunkat írjuk be egy ***listdual.sql*** nevű fájlba, majd adjuk ki a következő parancsot:

```
psql -U postgres cs_adatok -c
➥"\i /opt/imre/sql/listdual.sql"
```

Ez az előző parancséval teljesen megegyező kimenetet fog eredményezni.
A psql tartalmaz néhány On/Off jellegű kapcsolóparancsot is. A \t parancs segítségével ki- és bekapcsolhatjuk, hogy egy select parancs sorai tartalmazzák-e a fejlécet. A \H-val a html-, illetve nem html-mód között kapcsolhatunk. Nézzünk erre egy példát!

```
linux:/opt/imre/sql# psql -U postgres
cs_adatok -H -c "select * from telefon"

<table border=1>
  <tr>
```




```
    <th align=center>nev</th>
    <th align=center>szam</th>
  </tr>
  <tr valign=top>
    <td align=left>Nyiri Imre ıı   </td>
    <td align=left>123456789</td>
  </tr>
  <tr valign=top>
    <td align=left>Kiss JÆnos</td>
    <td align=left>123456789</td>
  </tr>
  <tr valign=top>
    <td align=left>PØntek TamÆs</td>
    <td align=left>123456789</td>
  </tr>
</table>
(3 rows)<br>
```

Adjuk ki parancssorból a következő utasítást, ami a fejléckiírás-letiltást is tartalmazza:

```
linux:/opt/imre/sql# psql -U postgres
cs_adatok  -t  -c "select * from telefon"
 Nyiri Imre ıı    | 123456789
 Kiss JÆnos       | 123456789
 PØntek TamÆs     | 123456789
```

E parancs eredménye már olyan formátumú, amit bármelyik Unix-parancsfájlban kiválóan alkalmazni tudunk. Beállítható az is, hogy az oszlopelválasztó karakter ne a csőkarakter (|), hanem valamely más karakter legyen; például vessző (,) esetén olyan kimenetet kapunk, ami az Excel vagy OpenOffice által ismert formátum (csv). A psql további lehetőségeinek izgalmas felfedezését olvasóinkra hagyjuk.

Készítsünk tárolt eljárásokat!

Ebben a pontban megnézzük, hogyan lehet SQL, PLPGSQL, illetve C/C++ nyelven megfogalmazott tárolt eljárásokat készíteni. A kódot az ***eljarasok.sql*** fájlba gépeljük be (lásd az *1. listát*). Ezután adjuk ki a következő parancsot, ami a `cs_adatok` adatbázisban létrehozza a `MyFunc()` és `MyFunc2()` tárolt eljárásokat:

```
psql -U postgres cs_adatok -c
     ➥"\i /opt/imre/sql/eljarasok.sql"
```

Láthatjuk, hogy az eljárások utolsó sorában a tárolt eljárás nyelvét a `language plpgsql` sor határozta meg, ami az ORACLE PL/SQL-t ismerők számára nem nyújt túl sok újdonságot.
A függvényeknek csak a típusát kell megadni, és az átvett értékekre a `$1, $2...` helyhatározó névvel lehet hivatkozni.
Most nézzünk egy rövid példát egy SQL tárolt eljárásra, amit az ***sqleljaras.sql*** fájl tartalmaz (lásd *2. listánkon*). Psql-ben kiadva a `"select uj_telefon('Janisovszky KÆroly', '11111');"` parancsot a telefontáblába egy új sor szúródik be.
A tárolt eljárások terén végzett kalandozásunk befejezéseként nézzük meg, hogyan lehet C/C++ nyelvű tárolt eljárást készíteni. Először is kell egy C/C++ nyelvű program, amit a ***duplazo.c*** forrásfájlba (*3. lista*) gépeltem be.
Következő lépésként a listán szereplő két függvényt egyetlen ****.so*** (shared object file = megosztott objektfájl) fájlba fordítjuk:

```
gcc -fpic -c duplazo.c
gcc -shared -o libduplazo.so duplazo.o
```

Az `fpic` a position-independent code rövidítése. Ezzel egy so-fájlt kaptunk, a dinamikus összeszerkesztő (linker) számára azonban még meg kell adnunk, hogy hivatkozáskor hol keresse ezt a könyvtárat (azaz a ***libduplazo.so*** fájlt). Ez a `JAVA CLASSPATH` szerkezethez hasonló, de a környezeti változót itt `LD_LIBRARY_PATH`-nak hívják. Másoljuk be a ***libduplazo.so*** fájlt egy ***/opt/pglibs/pg_so*** könyvtárba, majd gondoskodjunk arról, hogy az `LD_LIBRARY_PATH`-ban ez a könyvtár is benne legyen. Ez a módszer nem az egyetlen, hogy a ***libduplazo.so*** keresési helyét a dinamikus összeszerkesztő tudtára adjuk, de most nem ez a fő témánk.
Utolsó lépésként még az adatbázisba is be kell jegyezni egy hivatkozást a `psql` konzol segítségével:

```
create function duplazo( integer )
       returns integer as
       '/opt/pglibs/pg_so/libduplazo.so'
       language 'C'
```

Itt a C-forrásban lévő két függvény közül csak az első használatát tettük elérhetővé. A tárolt eljárás megvalósítása az a C nyelvű függvény lesz, aminek a ***libduplazo.so*** fájlban ***_duplazo*** a neve. Ezek után kiadhatjuk a `select duplazo(3);` parancsot, ami eredményként hatot fog kiírni.
Folytatjuk...

Nyíri Imre (inyiri@mol.hu)
Jelenleg a MOL Rt.-nél dolgozik. Informatikai vállalkozásában az Internet, a Linux, valamint a Java-programozás gyakorlati hasznosításával foglalkozik. Örök szerelme a C++ maradt.

© Kiskapu Kft. Minden jog fenntartva