PostgreSQL BAZA PODATAKA
NA LINUX OPERATIVNOM SISTEMU
- Seminarski rad -

Predmet: Baze podataka

Kandidat
Goran Maleš

SADRŽAJ

POGLAVLJE 1 – *UVOD*

PostgreSQL je objektno-relacioni sistem za upravljanje bazama podataka (object-relational database management system (ORDBMS)), baziranog na POSTGRES-u verzija 4.2[1] razvijenog na Univerzitetu Kalifornija u odeljenju za računarske nauke Berkli. POSTGRES je bio pionir u nekim konceptima koje su postale dostupne u komercijalnim bazama tek dosta kasnije.
PostgreSQL je open-source izdanak originalnog berklijevog koda. Podržava veliki deo SQL standarda i obezbeđuje mnoge savremene karakteristike:

- kompleksne upite
- foreign keys
- trigere
- views
- transakcioni integritet

PostgreSQL može biti slobodno korišćen, modifikovan i distribuiran od strane svakog korisnika za bilo kakvu upotrebu, bila ona privatna, komercijalna ili akademska.

Istorija
POSTGRES projekat je bio vođen od strane profesora Majkla Stonbrakera i implementacija je započela 1986. godine. Prva demo verzija je postala operativna 1987 godine i bila je prikazana 1988 godine na ACM-SIGMOD konferenciji. Verzija 1 je bila objavljena u junu 1989 godine, verzija 2 u junu 1990, verzija 3 1991 godine. Berklijev POSTGRES je zvanično završen sa verzijom 4.2.
1994, Endrju Ju i Džoli Čen su POSTGRES-u implementrirali interpreter za SQL jezik. Pod novim imenom, Postgres95 je bio izdan na webu kao open-source verzija originalnog POSTGRES Berkli koda. Pored ostalih, izvršena su sledeća poboljšanja:

- Upitni jezik PostQUEL je zamenjen SQL-om. Podupiti nisu bili podržani sve do PostgreSQL. Dodata je podrška za klauzulu GROUP BY.
- Novi program, psql, je omogućen radi interaktivnih SQL upita
- Kratki tutorijal za SQL je dolazio zajedno sa izvornim kodom

1996 je bilo jasno da naziv Postgres95 neće izdržati test vremena. Odabrano je novo ime, PostgreSQL, koje ukazuje na povezanost originalnog POSTGRESA i mogućnosti korišćenja SQL–a.

U seminarskom radu će biti predstavljene osnove PostgreSQL-a, gotovo u formi tutorijala, za operativni sistem Linux. Biće prikazan postupak instaliranja, osnove SQL-a, administracije servera, upravljanje bazama podataka i

[1] Više na internet adresi http://db.cs.berkeley.edu//postgres.html

postupak bekapa. Predstavljeni su terminalski program psql i grafički program PgAdmin III, koji postoje i za druge operativne sisteme (Windows i MacOS).

Tema seminarskog rada je značajna iz nekoliko razloga:

- PostgreSQL je jedna od najpopularnijih i najkvalitetnijih open-source baza podataka
- Sve je češća upotreba open-source programa te GNU/Linux operativnih sistema, naročito kao servera zbog svoje stabilnosti i sigurnosti. Za ilustraciju ove teze može poslužiti kompanija Oracle koja je nedavno (neuspešno) pokušala kupovinu konkurentske open-source sitema za upravljanje bazama podataka MySQL, objavila open-source verziju Oracla i poslednje najave kompanije su da želi pokretanje svoje sopstvene Linux distribucije ili kupovinu Novell-a, odnosno jedne od najpopularnijih Linux distribucija – SuSE Linuxa (na kojoj je i rađen ovaj seminarski). Oracle, po rečima svog prvog čoveka Larija Elisona ovim želi da postigne “kvalitativnu prednost u odnosu na svoje rivale - Microsoft i IBM”
- Na našem jeziku ne postoje knjige, uputstva, tutorijali za PostgreSQL
- Veći deo prikazanog psql-a i PgAdmin-a je upotebljiv i na operativnim sistemima Windows i MacOS

POGLAVLJE 2 – *INSTALACIJA*

Prvo je potrebno skinuti program sa interneta sa adrese http://www.postgresql.org/download/
Postupak instalacije je sledeći:

```
./configure
gmake
su
gmake install
adduser postgres
mkdir /usr/local/pgsql/data
chown postgres /usr/local/pgsql/data
su - postgres
/usr/local/pgsql/bin/initdb -D /usr/local/pgsql/data
/usr/local/pgsql/bin/postmaster -D /usr/local/pgsql/data >logfile 2>&1 &
/usr/local/pgsql/bin/createdb test
/usr/local/pgsql/bin/psql test
```

2.1. Kreiranje baze

Za kreiranje baze, koja će u ovom slučaju biti nazvana mydb, se koristi sledeća komanda:

```
# createdb mydb
```

Odgovor da je sve urađeno korektno će izgledati

```
CREATE DATABASE
```

Ukoliko je odgovor drugačiji, na primer:

```
createdb: command not found
```

tada PostgreSQL nije korektno instaliran ili nije instaliran uopšte ili je putanja komande pogrešno postavljena. Komanda se može pozvati i upisivanjem čitave putanje:

```
# /usr/local/pgsql/bin/createdb mydb
```

Odgovor može biti i ovakav:

```
createdb: could not connect to database postgres: could not connect to
server:
No such file or directory
        Is the server running locally and accepting
        connections on Unix domain socket "/tmp/.s.PGSQL.5432"?
```

Što znači da server nije pokrenut, ili nije pokrenut tamo gde ga je komanda createdb očekivala. U tom slučaju je potrebno proveriti uputstva za instalaciju.

Moguć je i sledeći odgovor:

```
createdb: could not connect to database postgres: FATAL:  user "joe" does
not exist
```

U ovom slučaju administrator nije napravio PostgreSQL nalog za korisnika “joe”. Bitno je razlikovati PostgreSQL naloge od naloga na samom operativnom sistemu. Naknadno će biti prikazano pravljenje naloga, koje može da pravi samo administrator. Potrebno je ulogovati se kao korisnik pod kojim je instalirana baza (najčešće postgres) da bi se kreirao prvi korisnički nalog.

Ukoliko postoji korisnički nalog, ali nalog koji nema dozvolu za kreiranje baze, odgovor će biti:

```
createdb: database creation failed: ERROR:  permission denied to create
database
```

PostgreSQL ne ograničava broj baza. Ime baze mora početi sa alfabetnim slovom i može imati još 63 karaktera. Zgodno je kreirati bazu pod istim imenom kao što je i korisničko ime. Mnogi alati posmatraju takvo ime baze za pretpostavljenu, tako da se naknadno olakšava rad. Za kreiranje takve baze dovoljno je otkucati:

```
# createdb
```

Ukoliko se više ne želi koristiti određena baza (npr mydb), jednostavno se može ukloniti komandom

```
# dropdb mydb
```

Sve to naravno ne može da se uradi ukoliko korisnik nije ujedno i kreator baze, odnosno ako nema privilegije super korisnika.

Sve prethodno se radi tek kada se uloguje kao korisnik postgres (ili kako drugacije, u zavisnosti od naziva vlasnika baze) sa komandom:

```
su - postgres
```

2.2. Pristupanje bazi

Jednom kreiranoj bazi može se pristupiti na sledeća tri načina:

a) *Korišćenjem PostgreSQL interaktivnog terminal programa* ***psql****, koji omogućava otvaranje, editovanje i izvršavanje SQL komandi.*

Seminarski se bazira na ovoj opciji, mada će biti prikazani i drugi načini za pristupanje bazi. Pristupanje iz terminalskog programa je najzgodnije iz nekoliko razloga – laka mogućnost administracije sa udaljene mašine pomoću SSH ili nekog drugog protokola, korišćenje svih mogućnosti baze (što u ostalim načinina pristupa nije moguće) i najvažnija je prodiranje u samu srž problematike PostgreSQL baze što eventualno naknadno korićenje grafičkih programa čini znatno lakšim. Pokreće se sa:

```
psql mydb
```

dobija se ovakav odgovor:

```
Welcome to psql 8.0.1, the PostgreSQL interactive terminal.

Type:  \copyright for distribution terms

       \h for help with SQL commands

       \? for help with psql commands

       \g or terminate with semicolon to execute query

       \q to quit

mydb=#
```

sada se moze isprobati recimo ovakva komanda:

```
mydb=# SELECT version();
```

dobijamo otpilike ovakav odgovor:

```
                                 version

-----------------------------------------------------------------------------
PostgreSQL 8.1.3 on i686-pc-linux-gnu, compiled by GCC gcc (GCC) 3.3.5
20050117 (prerelease) (SUSE Linux)

(1 row)
```

ili

```
mydb=# SELECT current_date;
```

dobijamo otpilike ovakav odgovor:

```
    date

------------

 2006-03-26

(1 row)
```

ili

```
mydb=# SELECT 2 + 2;
```

dobijamo ovakav odgovor:

```
?column?

----------

        4

(1 row)
```

Program psql ima veliki broj internih komandi koje nisu SQL komande. Počinju sa backslash karakterom “ \ ”. Neke od tih komandi su izlistane u pozdravnoj poruci koja se javila nakon pokretanja psql-a. Na primer, moguće je dobiti pomoć za razne PostgreSQL komande kucajući:

```
mydb=> \h
```

Za izlazak iz psql, kuca se

```
mydb=> \q
```

i psql prestaje sa radom i vraća Linuksov terminal.

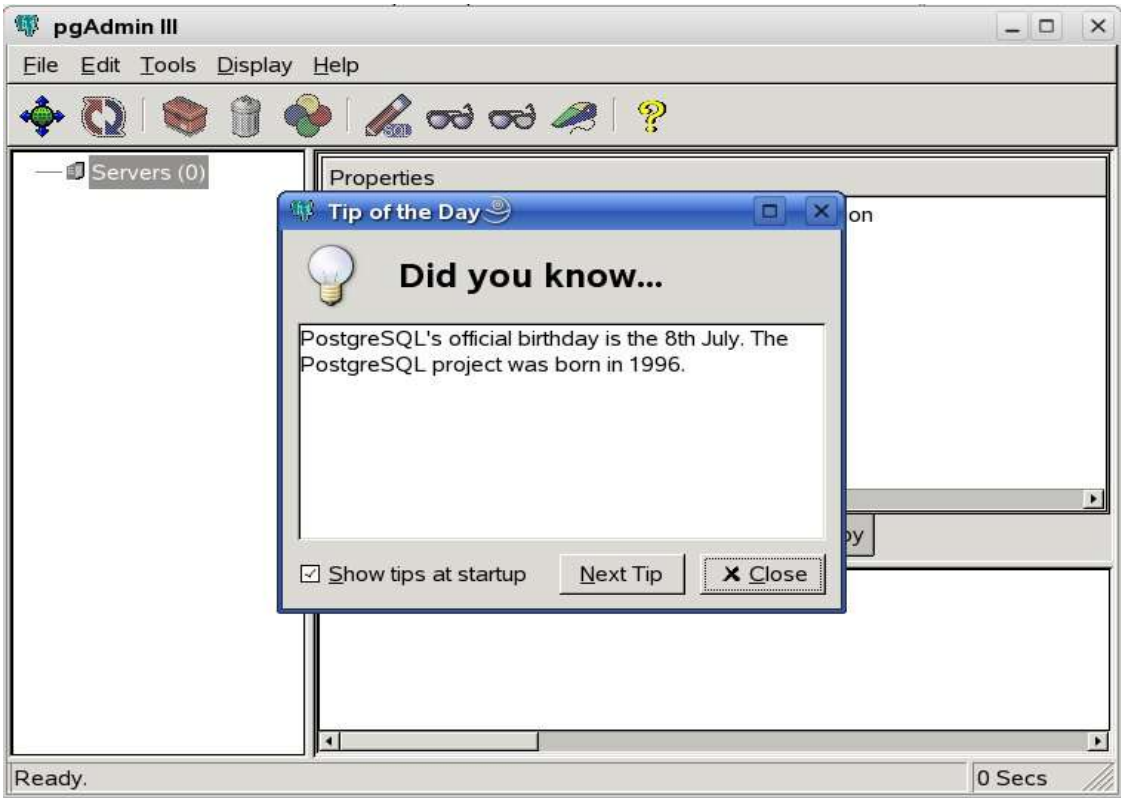
b) *Koristeći postojeće grafičke alatke kao što su* ***PgAdmin III*** *ili* ***PgAccess****.*

PgAdmin će biti predstavljen uporedo sa psql-om budući da je pretpostavljeni upravljački grafički program PostgreSQl-a za neke operativne sisteme (Windows), čak i dolazi sa PostgreSQL-om u verziji za Windows. PgAccess će biti prikazan na kraju seminarskog kao dodatak. Oba ova programa sadrže alate koji drugi program ne sadrži.

PgAdmin III se pokreće komandom:

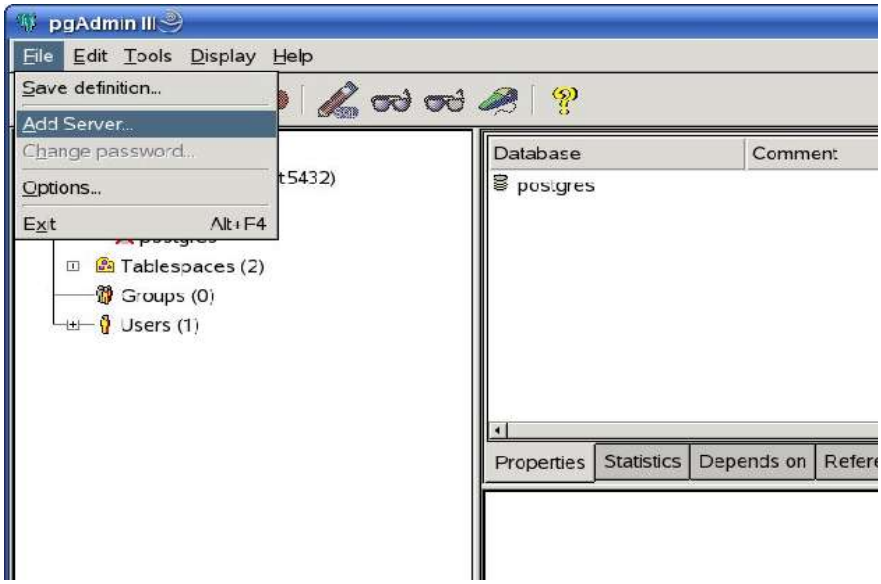
```
pgadmin3
```

Početni prozor izgleda kao na slici 1:



slika 1.

Nemamo nikakvih mogućnosti dok god ne definišemo na kom serveru želimo da radimo, zato je potrebno dodati server sa **File→Add Server...** kao na slici 2.



slika 2.

Dobijamo prozor kao na slici 3. Potrebno je popuniti sledeća polja:

slika 3.

Address – upisujemo adresu mašine na kojoj se nalazi server. U ovom slučaju, radimo na pomenutoj mašini te je adresa *localhost*.

Description – opis servera, u ovom slučaju *seminarski*.

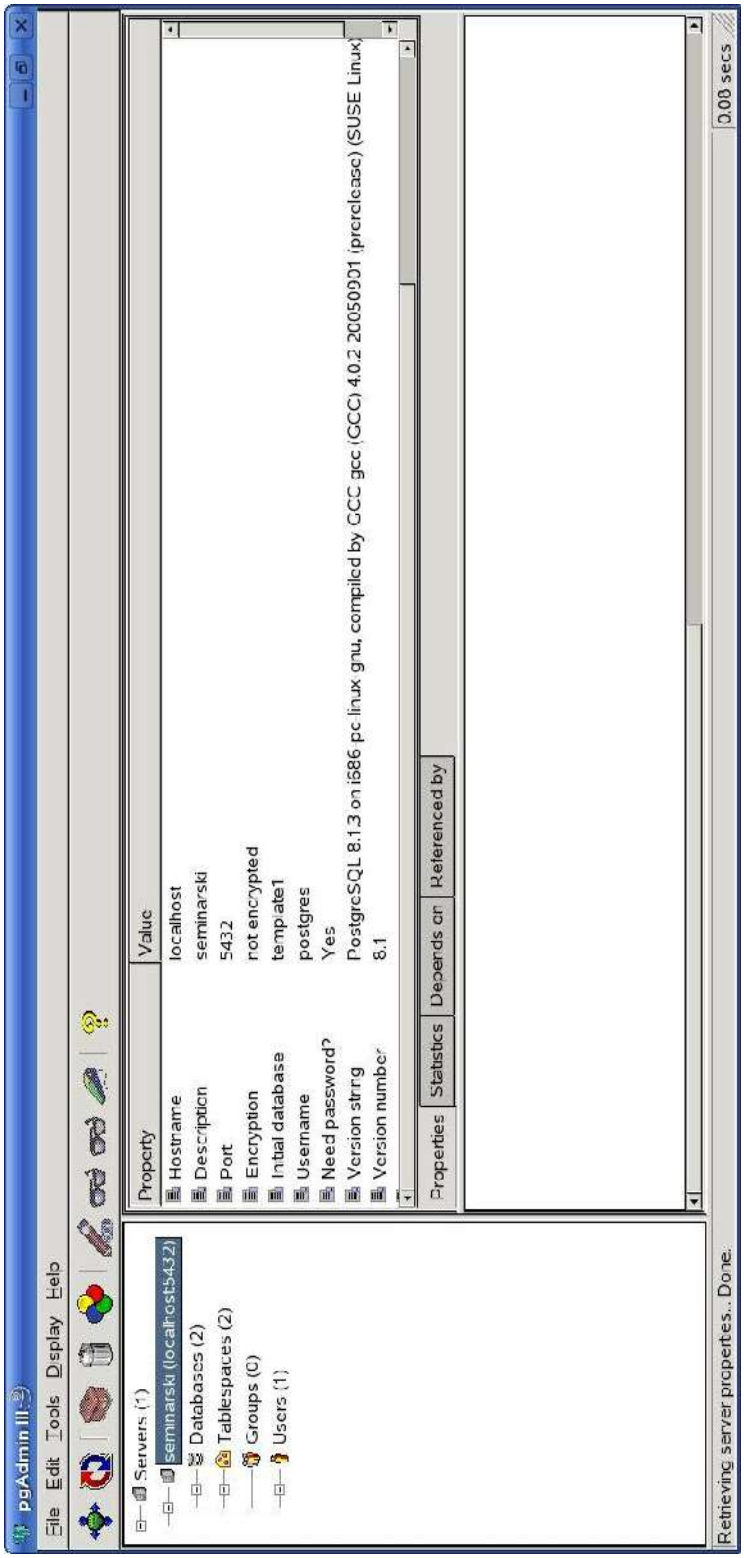
Initial DB – odabira se opcija *template 1*, naknadno će biti pojašnjeni šabloni baza podataka.

Username – korisničko ime.

need password – odabiranjem ove opcije će se nakon svakog pokretanja PgAdmina zahtevati password.

Password – lozinka korisnika.

Konačno, nakon dodavanja servera dobijamo prozor PgAdmina kao na slici 4. koji nam pruža mogućnost početka rada nad bazom.



Slika 4.

c) *Napisati svoju sopstvenu aplikaciju, koristeći jedan od mnogih podržanih programskih jezika*

POGLAVLJE 3 – *OSNOVE SQL-a*

3.1. Uvod

Potrebno je napomenuti da su neke PostgreSQL funkcije proširenje standardnih. U sledećim primerima oslanjamo se na pretpostavku da se nastavlja rad na prethodno napravljenoj mydb bazi i da je pokrenut program psql.

3.2. Koncepti

PostgreSQL je relacioni sistem za upravljanje bazama podataka. To znači da je u pitanju sistem za upravljanje podacima koje su smešteni u relacijama – esencijalno matematički izraz za tabelu.

Svaka tabela se sastoji od imenovanih redova. Svaki red ima iste kolone, a svaka kolona sadrži specifične tipove podataka.

Tabele su grupisane u baze podataka, a više baza podataka kojima se upravlja jednim PostgreSQL serverom čine klaster (en. cluster – grozd) baze podataka.

3.3. Kreiranje tabele

Tabela se može kreirati odabirom imena tabele, zajedno sa nazivima kolona i tipovima podataka:

```
CREATE TABLE vreme (
    grad            varchar(80),
    temp_nis        int,            -- najniza temperatura
    temp_vis        int,            -- najvisa temperatura
    padv            real,           -- padavine
    datum           date
);
```

Kako dolazi do preloma redova, u psql-u će kreiranje ove tabele izgledati ovako:

```
mydb=# CREATE TABLE vreme (
mydb(#     grad            varchar(80),
mydb(#     temp_nis        int,           -- najniza temperatura
mydb(#     temp_vis        int,           -- najvisa temperatura
mydb(#     padv            real,          -- padavine
mydb(#     datum           date
mydb(# );
```

psql prepoznaje da komanda nije završena, sve do poslednje linije.
Razmaci se mogu slobodno koristiti u SQL komandama. To znači da je moguće kucati komande u novom redu drugačije poravnate nego one u gornjim redovima, čak je sve moguće napisati u jednoj liniji. Dve crte ("--") najavljuju komentar. Šta god da iza njih sledi biće ignorisano sve do kraja linije. SQL je "casse insensitive" - komande se mogu pisati velikim ili malim slovima ali je elegantnije pisati ih velikim slovima zbog preglednosti

`varchar(80)` označava tipove podataka koje mogu biti dugi do 80 karaktera.

`int` predstavlja tip integer.

`real` je tip podataka za skladištenje single precision floating-point numbers.

`date` je tip podataka koji označava datum

PostgreSQL podržava standardne SQL tipove `int`, `smallint`, `real`, `double precision`, `char(N)`, `varchar(N)`, `date`, `time`, `timestamp`, i `interval`, kao i druge tipove i bogati set geometrijskih tipova. PostgreSQL može biti prilagođen za proizvoljan broj tipova podataka definisanih od strane korisnika. Posledica toga je da imena tipova nisu ključne reči (en. key words) osim kada se to zahteva za podršku u specijalnim slučajevima SQL standarda.

Sledeći primer predstavlja pravljenje tabele sa gradovima i njihovom geografskom lokacijom:

```
CREATE TABLE gradovi (
    naziv           varchar(80),
    lokacija        point
);
```

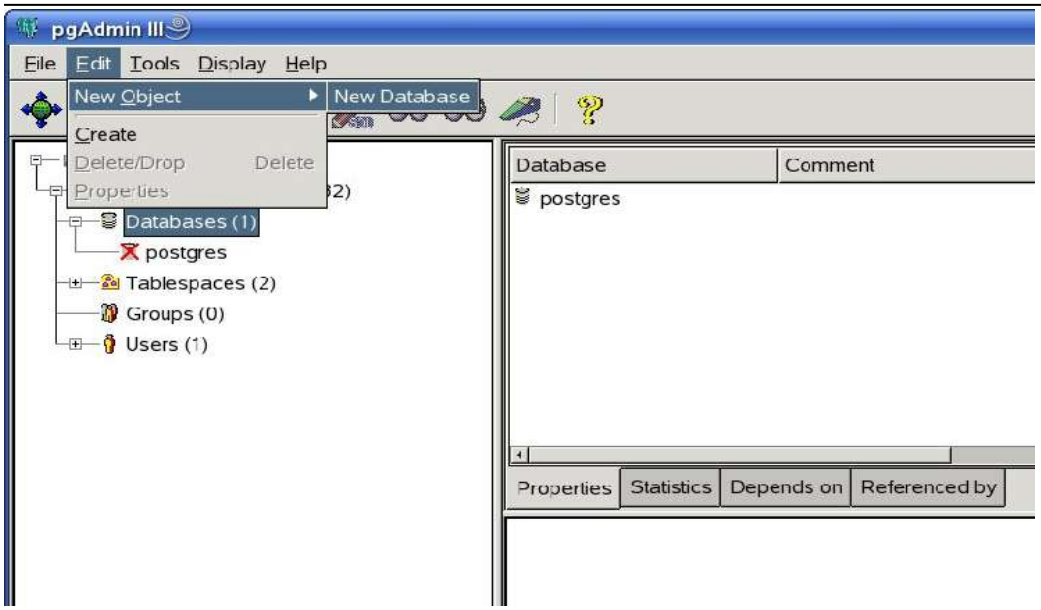
`point` je primer specifičnog PostgreSQL tipa podataka.

Na kraju, treba spomenuti da ukoliko određena tabela više nije potrebna ili ukoliko se ona želi ponovo kreirati, može se ukloniti komandom:

```
DROP TABLE imetabele;
```

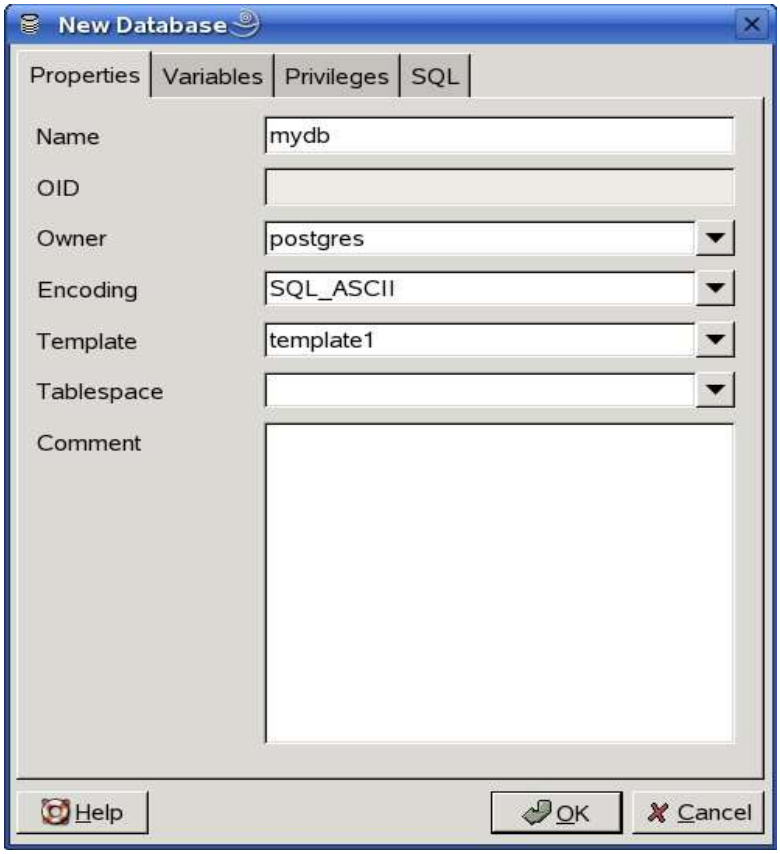
3.3.1. Kreiranje baze i tabele sa PgAdmin

Ukoliko baza već nije kreirana, možemo je napraviti odabirom opcije Edit→New Object→New Database, kao na sledećoj slici:



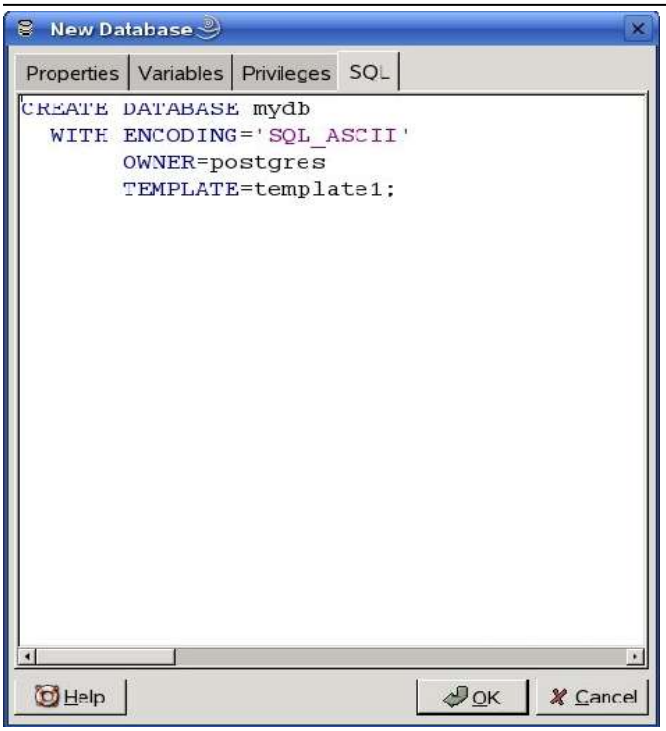
slika 5.

Sledeći korak je popunjavanje prozora kao na slici 6. Potrebno je dati ime bazi (i ovaj put je to mydb) i dodeliti joj vlasnika. Budući da je do sada napravljen samo jedan korisnički nalog, postgres, nikakva druga opcija pod «Owner» i ne može stajati. I ovaj put, kao i prilikom dodavanja servera koristimo template 1.

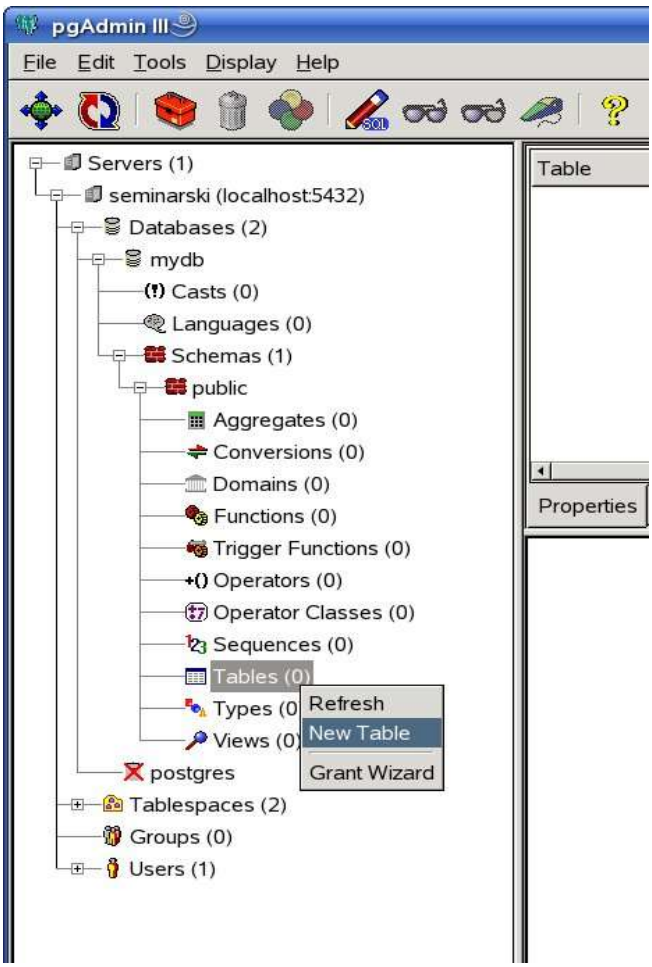


slika 6.

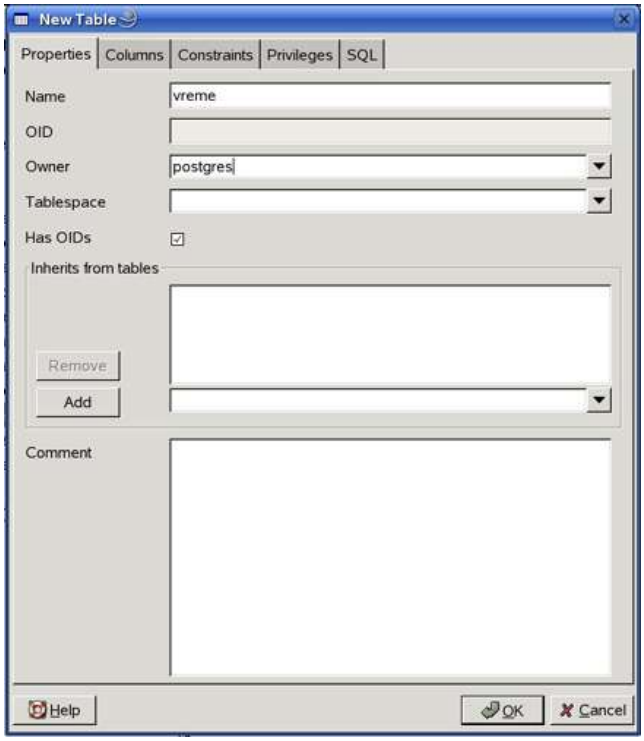
Tab SQL nam omogućava uvid kako sve to izgleda u SQL kodu (slika 7):



slika 7.



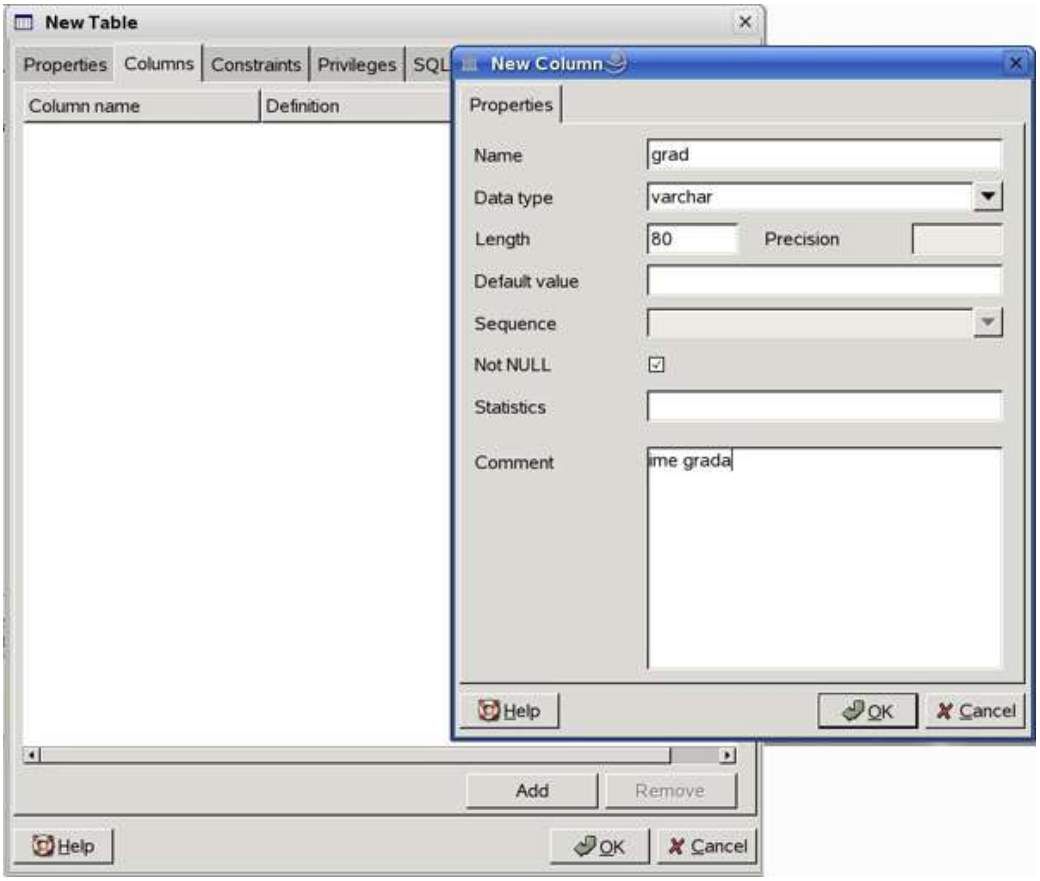
slika 8.



slika 9.

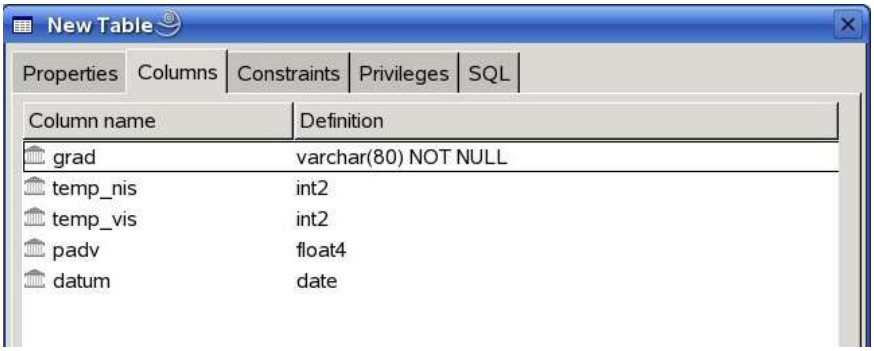
Na slici 8 je prikazano kreiranje tabele. Odabira se Servers→seminarski→Databases→mydb→Schemas→ public gde desnim klikom na Tables dobijamo opciju za kreiranje nove tabele.

Na slici 9, pod tabom Properties, upisuje se ime tabele, “vreme” u ovom slučaju i dodeljuje se vlasnik postgres. Odabiramo i opciju “Has OIDs” jer u suprotnom će tabela biti samo “read-only”, neće se moći editovati, znači ni popunjavati redovima.



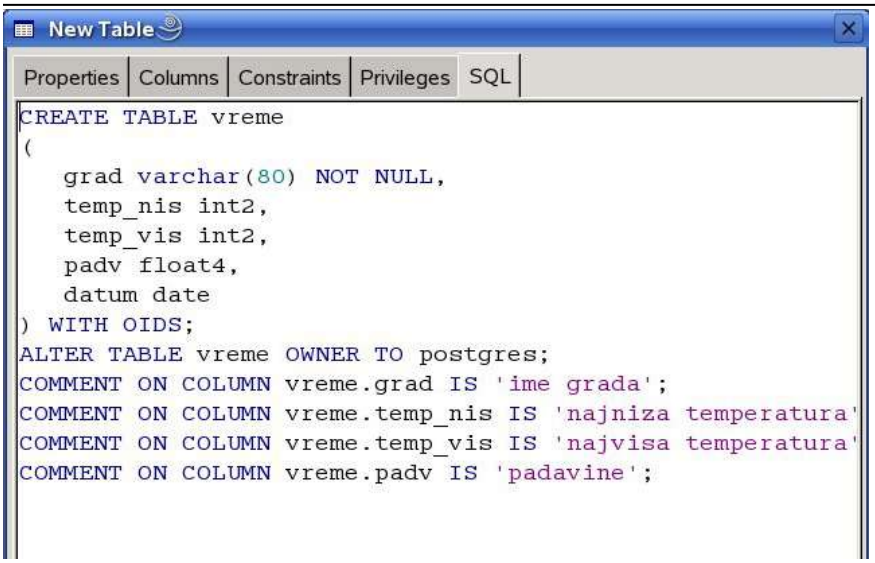
slika 10.

Na drugom tabu, Columns (kolone), vrši se kreiranje kolona. Klikom na dugme Add dobija se prozor u kom se upisuje ime kolone ("grad"), tip podatka ("varchar(80)"), opciono "Not NULL" i eventualno komentar ("ime grada").



slika 11.

Postupak se ponovi onoliko puta koliko je kolona predviđeno i na kraju tab Columns izgleda kao na slici 11. Na slici 12 je prikazan tab SQL:



slika 12.

3.4. Popunjavanje tabele redovima

Za popunjavanje tabele redovma se koristi komanda `INSERT`:

```
INSERT INTO vreme VALUES ('Novi Sad', 12, 21, 0.0, '2006-03-28');
```

Treba primetiti da svi tipovi podataka koriste očigledne ulazne oblike. Konstante koje nisu jednostavne numeričke vrednosti obično moraju biti pod navodnicima ('), kao u ovom primeru. Tip date je prilično fleksibilan u prihvatanju upisanih podataka.

Tip `point` zahteva par koordinata za ulazni podatak:

```
INSERT INTO gradovi VALUES ('Novi Sad', '(19.51, 45.20)');
```

Ili

```
INSERT INTO gradovi VALUES ('Zrenjanin', '(20.25, 45.22)');
```

Sintaksa korišćena do sada zahteva pamćenje poretka kolona. Alternativa je sintaksa koja omogućava ekspicitno navođenje kolona:

```
INSERT INTO vreme (grad, temp_nis, temp_vis, padv, datum)
    VALUES ('Novi Sad', 5, 12, 0.6, '2006-03-29');
```

Kolone se mogu izlistati u redosledu drugačijem nego što su u samoj tabeli, ako se ne želi popuniti neka kolona ili ako su npr. “padavine” nepoznate:

```
INSERT INTO vreme (datum, grad, temp_vis, temp_nis)
    VALUES ('2006-03-29', 'Zrenjanin', 14, 7);
```

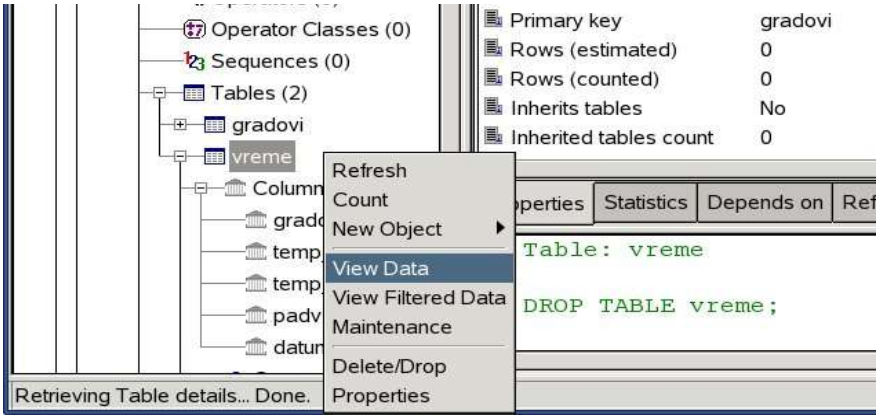
Popunjavanje je takođe moguće korišćenjem komande `COPY` za učitavanje velikih količina podataka iz tekstualne datoteke. Na ovaj način se popunjavanje obično izvede brže zato što je `COPY` komanda optimizovana za datu aplikaciju, ali je manje fleksibilna nego komanda `INSERT`.

Na primer:

```
COPY vreme FROM '/home/user/vreme.txt';
```

gde ime datoteke za izvornu datoteku mora biti dostupna serveru, ne klijentu, budući da server čita fajl direktno.

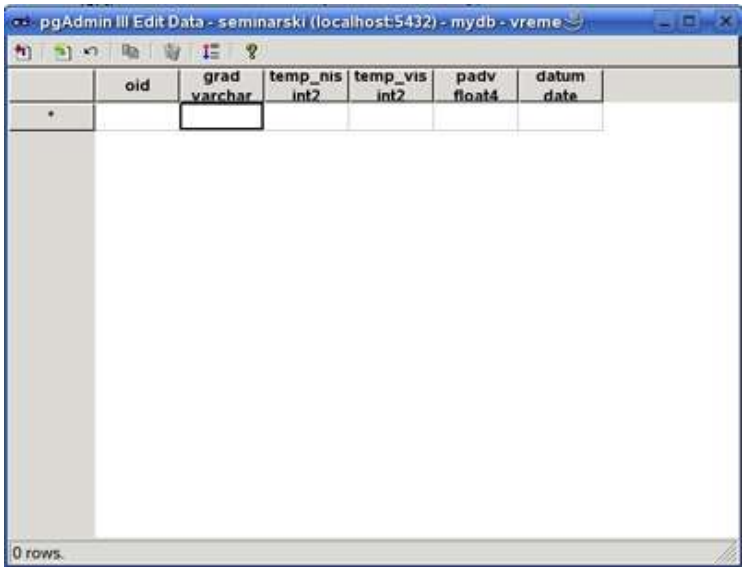
3.4.1. Popunjavanje tabele redovima sa PgAdmin



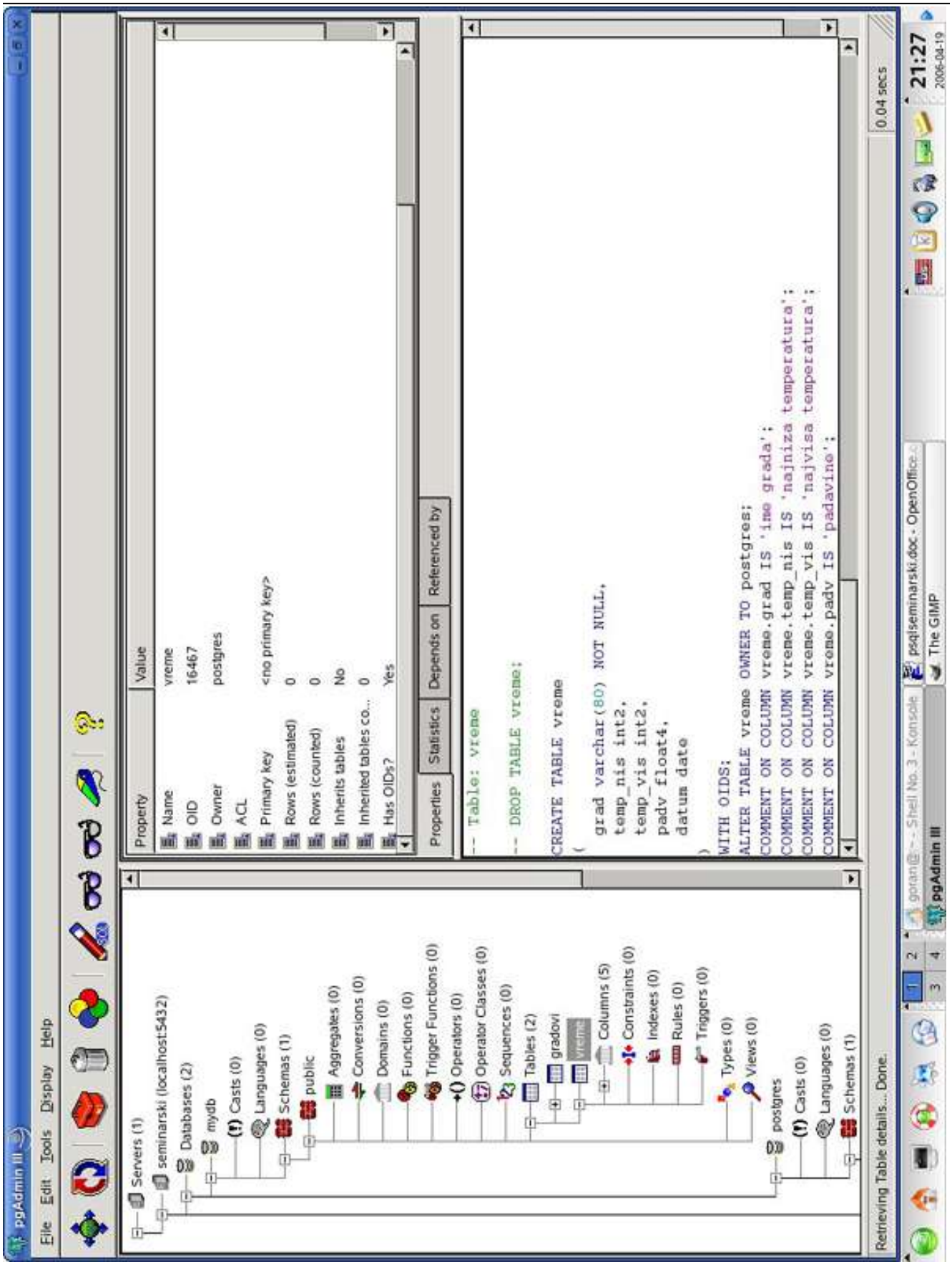
Desnim klikom na tabelu "vreme" i odaborim opcije "View Data" se može izvršiti editovnaje tabele.

slika 13.

Jednostavnim klikom u željnu ćeliju se upisuju potrebni podaci. Novi red se dobija klikom na "*" ili kombinacijom tastera Ctrl+Enter. Konačno, na slici 15 je prikazan kompletan prozor PgAdmina.



slika 14.


pgAdmin III
File Edit Tools Display Help
Servers (1)
seminarski (localhost:5432)
Databases (2)
mydb
Casts (0)
Languages (0)
Schemas (1)
public
Aggregates (0)
Conversions (0)
Domains (0)
Functions (0)
Trigger Functions (0)
Operators (0)
Operator Classes (0)
Sequences (0)
Tables (2)
gradovi
vreme
Columns (5)
Constraints (0)
Indexes (0)
Rules (0)
Triggers (0)
Types (0)
Views (0)
postgres
Casts (0)
Languages (0)
Schemas (1)
Property
Value
Name
vreme
OID
16467
Owner
postgres
ACL
Primary key
<no primary key>
Rows (estimated)
0
Rows (counted)
0
Inherits tables
No
Inherited tables co...
0
Has OIDs?
Yes
Properties
Statistics
Depends on
Referenced by
-- Table: vreme
-- DROP TABLE vreme;
CREATE TABLE vreme
(
grad varchar(80) NOT NULL,
temp_nis int2,
temp_vis int2,
padv float4,
datum date
)
WITH OIDS;
ALTER TABLE vreme OWNER TO postgres;
COMMENT ON COLUMN vreme.grad IS 'ime grada';
COMMENT ON COLUMN vreme.temp_nis IS 'najniza temperatura';
COMMENT ON COLUMN vreme.temp_vis IS 'najvisa temperatura';
COMMENT ON COLUMN vreme.padv IS 'padavine';
Retrieving Table details... Done.
0.04 secs
21:27
2006-04-19

3.5. Upiti

Da bi se dobili podaci iz tabele mora se postaviti upit. Za to se koristi SQL komanda `SELECT`. Primer:

```
SELECT * FROM vreme;
```

U ovom primeru (*) predstavlja sve kolone. Potpuno isti rezultat se dobija komandom:

```
SELECT grad, temp_nis, temp_vis, padv, datum FROM vreme;
```

Izlaz će izgledati ovako:

```
   grad    | temp_nis | temp_vis | padv |   datum
-----------+----------+----------+------+------------
 Novi Sad  |       12 |       21 |    0 | 2006-03-28
 Novi Sad  |        5 |       12 |  0.6 | 2006-03-29
 Zrenjanin |        7 |       14 |      | 2006-03-29
(3 rows)
```

Moguće je pisanje izraza, ne samo jednostavno upućivanje na kolone. Primer:

```
SELECT grad, (temp_vis+temp_nis)/2 AS temp_pros, datum FROM vreme;
```

Izlaz:

```
   grad    | temp_pros |   datum
-----------+-----------+------------
 Novi Sad  |        16 | 2006-03-28
 Novi Sad  |         8 | 2006-03-29
 Zrenjanin |        10 | 2006-03-29
(3 rows)
```

`AS` je korišćen za dodeljivanje imena izlaznoj koloni.

Upit može biti određen dodavanjem `WHERE` odredbe koja specificira koji redovi su traženi.

Odredba `WHERE` sadrži boolean izraz i samo redovi za koje je boolean izraz zadovoljen će biti izlistani. Najčešči boolean operatori (`AND`, `OR`, i `NOT`) su dozvoljeni. Na sledećem primeru će biti izlistani kišni dani u Novom Sadu:

```
SELECT * FROM vreme
    WHERE grad = 'Novi Sad' AND padv > 0.0;
```

Rezultat:

```
   grad   | temp_nis | temp_vis | padv |   datum
----------+----------+----------+------+------------
 Novi Sad |        5 |       12 |  0.6 | 2006-03-29
(1 row)
```

Takođe je moguće zahtevati da rezultati upita budu izlistani u željenom redosledu:

```
SELECT * FROM vreme
    ORDER BY grad;
```

Rezultat:

```
   grad    | temp_nis | temp_vis | padv |   datum
-----------+----------+----------+------+------------
 Novi Sad  |       12 |       21 |    0 | 2006-03-28
 Novi Sad  |        5 |       12 |  0.6 | 2006-03-29
 Zrenjanin |        7 |       14 |      | 2006-03-29
(3 rows)
```

U ovom primeru, redosled ispisa nije potpuno specificiran. Sledi primer koji ispisuje rezultat po tačno traženom redosledu:

```
SELECT * FROM vreme
    ORDER BY grad, temp_nis;
```

Rezultat:

```
   grad    | temp_nis | temp_vis | padv |   datum
-----------+----------+----------+------+------------
 Novi Sad  |        5 |       12 |  0.6 | 2006-03-29
 Novi Sad  |       12 |       21 |    0 | 2006-03-28
 Zrenjanin |        7 |       14 |      | 2006-03-29
(3 rows)
```

Može se zahtevati i da dupirani redovi budu uklonjeni iz rezultata upita:

```
SELECT DISTINCT grad
    FROM vreme;
```

Rezultat:

```
   grad
-----------
 Novi Sad
 Zrenjanin
(2 rows)
```

I u ovom slučaju, ispis redova može da varira. Da bi redosled bio tačno onakav kakav želimo da vidimo, zajedno

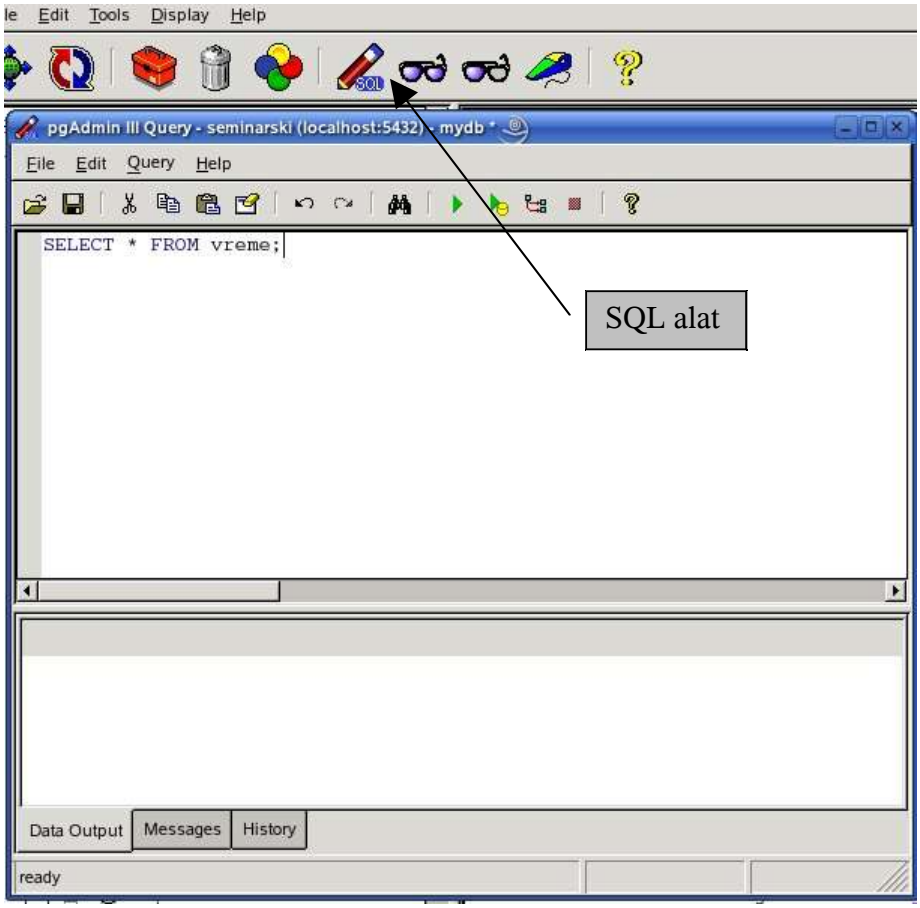
se koriste `DISTINCT` i `ORDER BY`:

```
SELECT DISTINCT city
    FROM weather
    ORDER BY city;
```

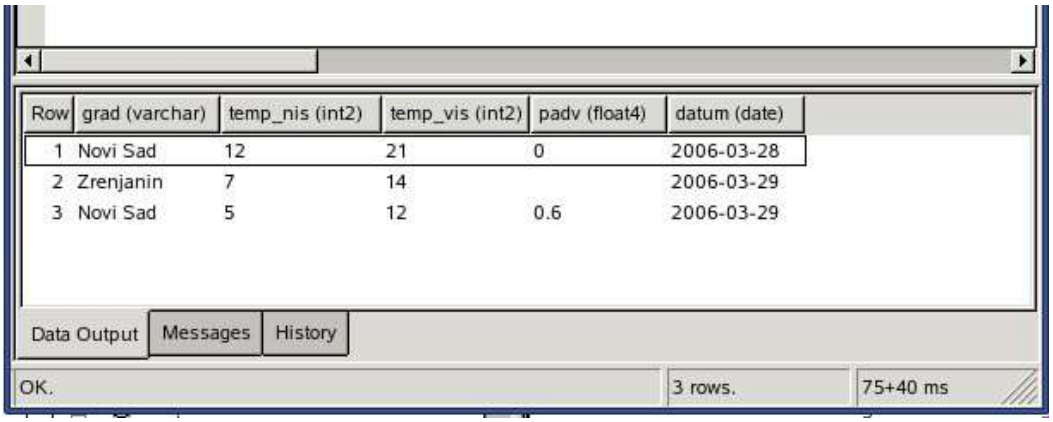
U nekim sistemima za upravljanje bazama podataka, uključujući i starije verzije PostgreSQL-a, korišćenje **DISTINCT** automatski ispisuje redove po redosledu i **ORDER BY** je nepotrebno. Ali ovo nije zahtevano SQL standardom i PostgreSQL ne garantuje da će samo korišćenje **DISTINCT** ispisati redove po željenom redosledu.

3.5.1. Upiti sa PgAdmin

Ikonica SQL alata je prikazana na slici 16. Rezultat upita je prikazan na slici 17.



slika 16.



slika 17.

3.6. Udruživanje tabela

Upiti mogu da pristupe u više tabela istovremeno ili da pristupe jednoj tabeli tako da više redova tabele budu obrađeni u isto vreme. Upit koji pristupa ka više redova jedne tabele ili ka različitim tabelama istovremeno se naziva *join* upit. Recimo da se žele izlistati svi vremenski zapisi zajedno sa lokacijom grada. Da bi se to uradilo, potrebno je porediti kolonu "grad" svakog reda tabele "vreme" sa imenom kolone svih redova u tabeli "gradovi" i odabrati par/kombinaciju redova tamo gde se vrednosti podudaraju.

Upit će izgledati ovako:

```
SELECT *
    FROM vreme, gradovi
    WHERE grad = naziv;
```

Rezultat:

```
   grad     | temp_nis | temp_vis | padv |   datum    |   naziv   |   lokacija
-----------+----------+----------+------+------------+-----------+---------------
 Novi Sad  |       12 |       21 |    0 | 2006-03-28 | Novi Sad  | (19.51,45.2)
 Novi Sad  |        5 |       12 |  0.6 | 2006-03-29 | Novi Sad  | (19.51,45.2)
 Zrenjanin |        7 |       14 |      | 2006-03-29 | Zrenjanin | (20.25,45.22)
(3 rows)
```

Potrebno je obratiti pažnju na dve mogućnosti u rezultatu:

- U slučaju da jedan od ova dva grada nije postojao u tabeli "gradovi", ne bi se pojavio ni u zajedničkoj tabeli pošto bi join ignorisao redove koji nemaju svog para.
- Izlistan je rezultat sa dve kolone koje sadrže ime grada. To je ispravno, zato što su liste kolona tabela vreme i gradovi povezani. U praksi to nije narošito poželjno, zato je mnogo elegantnije korititi imena kolona umesto (*):

```
SELECT grad, temp_nis, temp_vis, padv, datum, lokacija
         FROM vreme, gradovi
         WHERE grad = naziv;
```

Rezultat:

```
   grad     | temp_nis | temp_vis | padv |   datum    |   lokacija
-----------+----------+----------+------+------------+---------------
 Novi Sad  |       12 |       21 |    0 | 2006-03-28 | (19.51,45.2)
 Novi Sad  |        5 |       12 |  0.6 | 2006-03-29 | (19.51,45.2)
 Zrenjanin |        7 |       14 |      | 2006-03-29 | (20.25,45.22)
(3 rows)
```

Budući da sve kolone imaju različita imena automatski se prepoznaje kojoj tabeli kolona pripada, ali je dobro samostalno definisati imena kolona:

```
SELECT vreme.grad, vreme.temp_nis, vreme.temp_vis,
       vreme.padv, vreme.datum, gradovi.lokacija
    FROM vreme, gradovi
    WHERE gradovi.naziv = vreme.grad;
```

Istovetan rezultat se dobija i ovakvom formom:

```
SELECT *
    FROM vreme INNER JOIN gradovi ON (vreme.grad = gradovi.naziv);
```

Na sledećem primeru će biti prikazano rešenje za prethodnu situaciju kada u tabeli “gradovi” ne postoji zapis sa gradom koji se pojavljuje u tabeli “vreme”, a želimo da ga izlistamo u join upitu. Prvo je potrebno napraviti takvu situaciju unošenjem još jednog grada u tabelu “vreme”

```
INSERT INTO vreme VALUES ('Beograd', 4, 11, 0.5, '2006-03-29');
```

Ono što želimo jeste skeniranje tabele vreme i za svaku kolonu pronaći par u tabeli gradovi. Ako par ne postoji, želimo da “prazne vrednosti” zamenjene za kolene tabele gradovi. Takva vrsta upita se naziva outer join.

Komanda:

```
SELECT *
    FROM vreme LEFT OUTER JOIN gradovi ON (vreme.grad = gradovi.naziv);
```

Rezultat:

```
   grad     | temp_nis | temp_vis | padv |   datum    |   naziv   |   lokacija
-----------+----------+----------+------+------------+-----------+---------------
 Novi Sad  |       12 |       21 |    0 | 2006-03-28 | Novi Sad  | (19.51,45.2)
 Novi Sad  |        5 |       12 |  0.6 | 2006-03-29 | Novi Sad  | (19.51,45.2)
 Zrenjanin |        7 |       14 |      | 2006-03-29 | Zrenjanin | (20.25,45.22)
 Beograd   |        4 |       11 |  0.5 | 2006-03-29 |           |
(4 rows)
```

Ovaj upit se naziva *left outer join* zbog toga što će u tabeli na levoj strani u ispisu biti prikazane sve kolone najmanje jednom, a u ispisu tabele na desnoj strani će biti prikazane samo oni redovi koji odgovaraju redovima leve tabele.

Takođe je moguće join jedne tabele i to se naziva *self join*. Na primer, žele se videti svi zapisi vreme koji su u temperaturnoj zavisnosti od drugih vremenskih zapisa. Znači, potrebno je porediti `temp_nis` i `temp_vis` kolone od svakog vreme reda sa `temp_nis` i `temp_vis` kolonama drugih vreme redova.

Upit:

```
SELECT W1.grad, W1.temp_nis AS niska, W1.temp_vis AS visoka,
    W2.grad, W2.temp_nis AS niska, W2.temp_vis AS visoka
    FROM vreme W1, vreme W2
    WHERE W1.temp_nis < W2.temp_nis
    AND W1.temp_vis > W2.temp_vis;
```

Ovakav način preimenovanja dosta može da pomogne u brzini rada i dosta se često koristi. Npr:

```
SELECT *
    FROM vreme w, gradovi c
    WHERE w.grad = c.naziv;
```

3.7. Agregatne funkcije

Kao i većina drugih realacionih baza podataka, PostgreSQL podržava agregatne funkcije. Agregatne funkcije računaju rezultat više ulaznih redova. Na primer, postoje agregatne funkcije za `count`, `sum`, `avg` (prosek), `max` (maximum) i `min` (minimum).

Kao primer, može se pronaći najviša niska temperatura upitom:

```
SELECT max(temp_nis) FROM vreme;
```

Rezultat:

```
max
-----
  12
(1 row)
```

Ako se želi dobiti koji je grad kod kog se javlja prethodna temperatura, može se pokušati sledeće:

```
SELECT grad FROM vreme WHERE temp_nis = max(temp_nis);
```

Rezultat:

```
ERROR:  aggregates not allowed in WHERE clause
```

Greška se pojavila pošto agregatna funkcija max ne može biti korišćena u WHERE odredbi. Ova restrikcija postoji zato što WHERE odredba određuje redove koji će ići u agregaciju, tako da mora biti razvijena pre nego se izračuna agregatna funkcija. Ovakva situacija se može rešiti korištenjem podupita:

```
SELECT grad FROM vreme
    WHERE temp_nis = (SELECT max(temp_nis) FROM vreme);
```

Rezultat:

```
   grad
----------
 Novi Sad
(1 row)
```

Ovakav upit je u redu i ne prijavljuje grešku pošto je podupit nezavisan.

Agregatne funkcije su takođe korisne u kombinaciji sa `GROUP BY` uslovima. Na primer, može se dobiti maksimalna niska temperatura posmatajući svaki grad ponaosob

```
SELECT grad, max(temp_nis)
    FROM vreme
    GROUP BY grad;
```

Rezultat:

```
   grad    | max
-----------+-----
 Novi Sad  |  12
 Zrenjanin |   7
 Beograd   |   4
(3 rows)
```

Rezultat daje po jedan izlazni red po gradu. Ovi grupisani redovi se mogu filtrirati koristeći `HAVING`:

```
SELECT grad, max(temp_nis)
    FROM vreme
    GROUP BY grad
    HAVING max(temp_nis) < 10;
```

Rezultat:

```
   grad    | max
-----------+-----
 Zrenjanin |   7
 Beograd   |   4
(2 rows)
```

Dobijamo rezultate samo za gradove koji imaju vrednosti temp_nis manje od 10.

Konačno, ako nas interesuju samo gradovi čije ime počinje sa “Z”:

```
SELECT grad, max(temp_nis)
    FROM vreme
    WHERE grad LIKE 'Z%'
    GROUP BY grad
    HAVING max(temp_nis) < 10;
```

Rezultat:

```
   grad    | max
-----------+-----
 Zrenjanin |   7
(1 row)
```

Još je bitno napomenuti razliku između SQL WHERE and HAVING uslova. WHERE selektuje redove pre grupe redova i agregati se računaju, dok HAVING selektuje grupe redova i agregati se računaju. Zato WHERE uslov ne sme sadržati agregatne funkcije.

3.8. Osvežavanje

Postojeći redovi se mogu osvežiti koristeći komandu UPDATE. Pretpostavka je da je primećeno da su 29.03.2006. temperature pogrešno unesene i da je temperatura bila veća za 2 stepena:

```
UPDATE vreme
    SET temp_vis = temp_vis + 2,  temp_nis = temp_nis + 2
    WHERE datum > '2006-03-29';
```

Rezultat:

```
   grad    | temp_nis | temp_vis | padv |   datum
-----------+----------+----------+------+------------
 Novi Sad  |       12 |       21 |    0 | 2006-03-28
 Novi Sad  |        5 |       12 |  0.6 | 2006-03-29
 Zrenjanin |        7 |       14 |      | 2006-03-29
 Beograd   |        4 |       11 |  0.5 | 2006-03-29
(4 rows)
```

3.9. Brisanje

Redovi se mogu uklanjati koristeći DELETE komandu. Pretpostavka je da više nisu potrebni podaci o vremenu u Beogradu:

```
DELETE FROM vreme WHERE grad = 'Beograd';
```

Rezultat:

```
SELECT * FROM vreme;
```

```
   grad    | temp_nis | temp_vis | padv |   datum
-----------+----------+----------+------+------------
 Novi Sad  |       12 |       21 |    0 | 2006-03-28
 Novi Sad  |        5 |       12 |  0.6 | 2006-03-29
 Zrenjanin |        7 |       14 |      | 2006-03-29
(3 rows)
```

Jedino čega se pri brisanju treba čuvati jeste komanda

```
DELETE FROM imetabele;
```

Ukoliko se ne napiše koji se red želi obrisati i komanda izgleda kao prethodna, biće uklonjeni svi redovi iz tabele, bez ikakvog traženja potvrde od strane sistema.

3.10. Napredne funkcije

3.10.1. Pogledi (Views)

Ukoliko su određeni podaci od naročitog interesa za aplikaciju i ne želi se svaki put iznova kucati upit, može se kreirati pogled nad upitom, koji upitu daje ime:

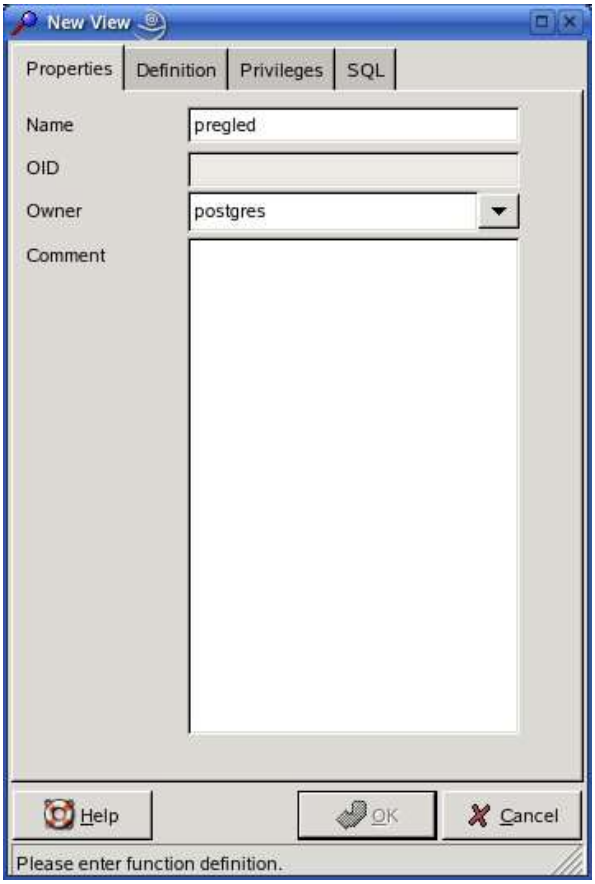
```
CREATE VIEW pregled AS
    SELECT grad, temp_nis, temp_vis, padv, datum, lokacija
        FROM vreme, gradovi
        WHERE grad = naziv;
```

Upit:

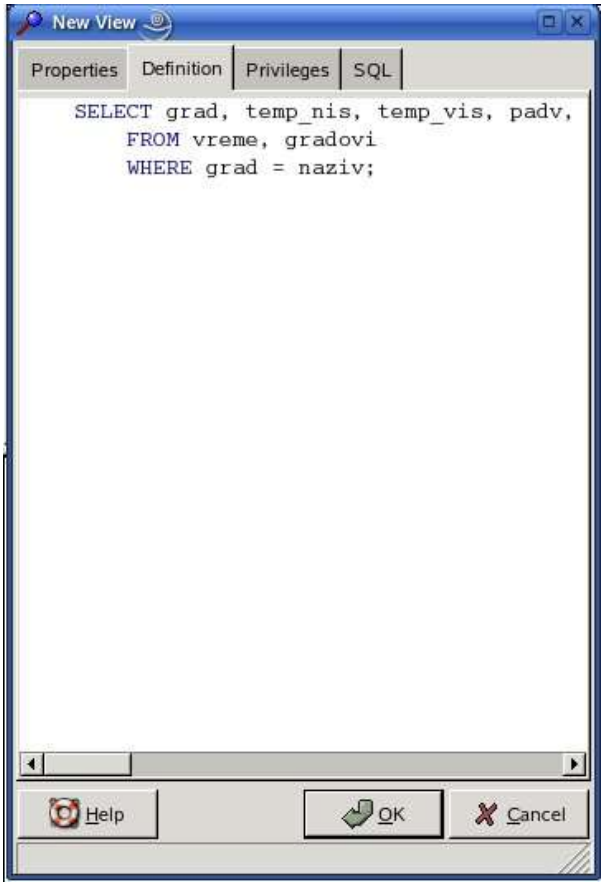
```
SELECT * FROM pregled;
```

3.10.1.1. Pogledi sa PgAdmin

Analogno prethodnom kreiranju nove tabele. kreira se i novi view. Odabirom opcije Servers→seminarski → Databases→mydb→Schemas→public dobija se mobućnost kreiranja novog pogleda. Dobija se prozor u kom je potebno naznačiti ime pogleda, u ovom slučaju "pregled" i odrediti vlasnika. Na sledećem tabu, Definition, prikazan je SQL kod.



slika 18.



slika 19.

3.10.2. Foreign Keys

Ukoliko se javi potreba da se ne dozvoljava unošenje redova u tabelu vreme ukoliko se ne željeni podaci ne javljaju i u tabeli gradovi problem se rešava referencijalnim integritetom podataka. Prilikom kreiranje tabele koristi se sledeći SQL:

```
CREATE TABLE gradovi (
        grad     varchar(80) primary key,
        lokacija point
);
```

```
CREATE TABLE vreme (
        grad      varchar(80) references gradovi(grad),
        temp_nis   int,
        temp_vis   int,
        padv       real,
        datum      date
);
```

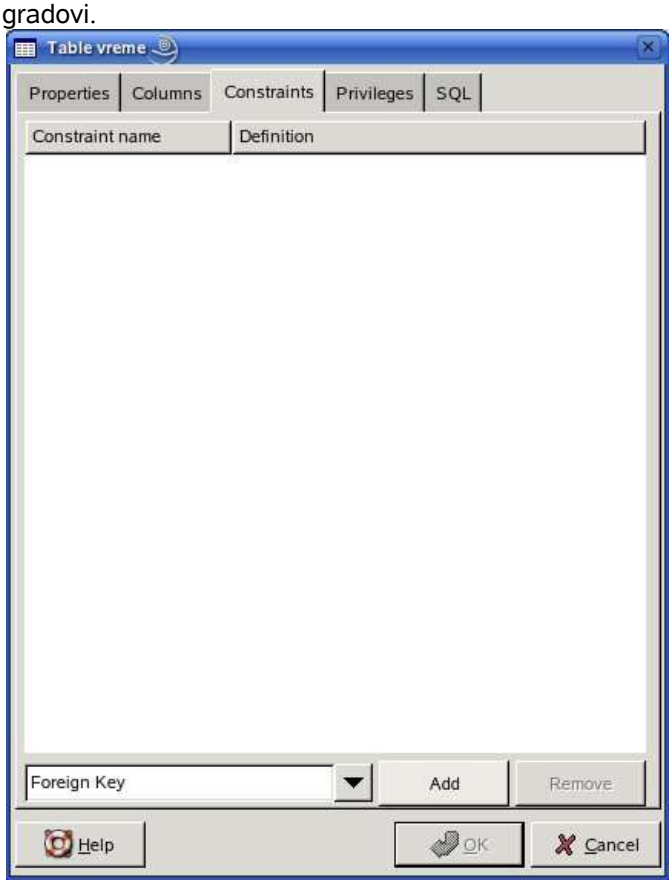
Sada, ako se pokuša uneti neodgovarajući zapis:

```
INSERT INTO vreme VALUES ('Novi Sad', 12, 21, 0.0, '2006-03-28');
```

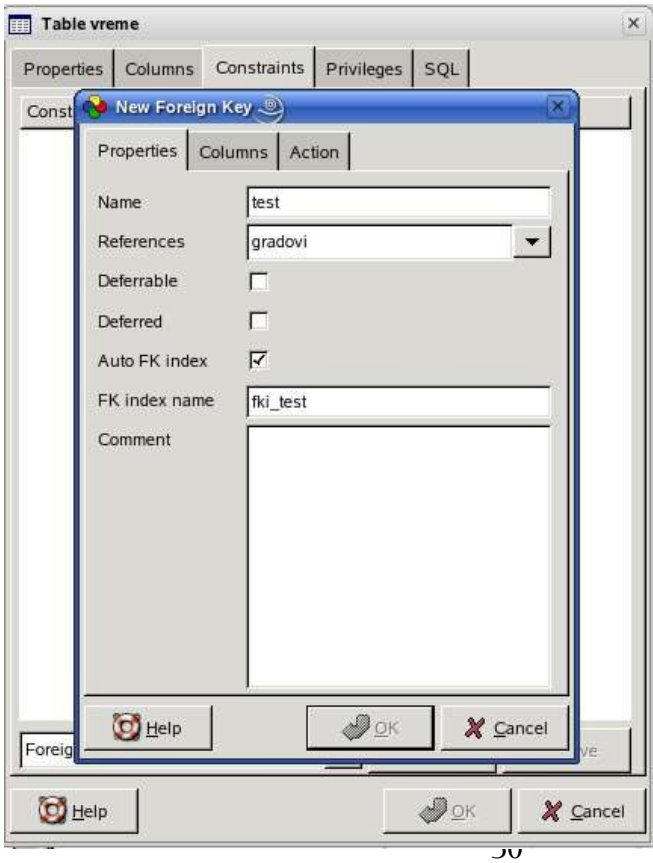
```
ERROR:  insert or update on table "vreme" violates foreign key constraint
"vreme_grad_fkey"
DETAIL:  Key (grad)=(Novi Sad) is not present in table "gradovi".
```

3.10.2.1. Foreign Keys sa PgAdmin

Odabirom opcije Servers→seminarski→Databases→mydb→Schemas→public, desnim klikom na tabelu vreme i odabirom opcije Properties, prelazimo na treći tab, Constraints (slika 20). Klikom na dugme Add se dobija situacija sa slike 21, gde je potrebno odrediti ime za Foreign Key, odrediti referentnu tabelu, u ovom slučaju gradovi.



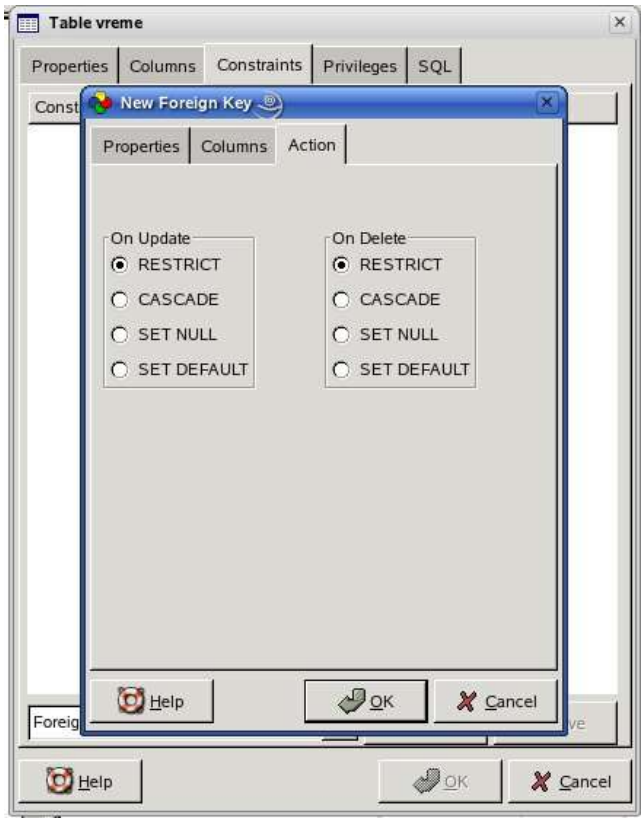
slika 20.



slika 21.

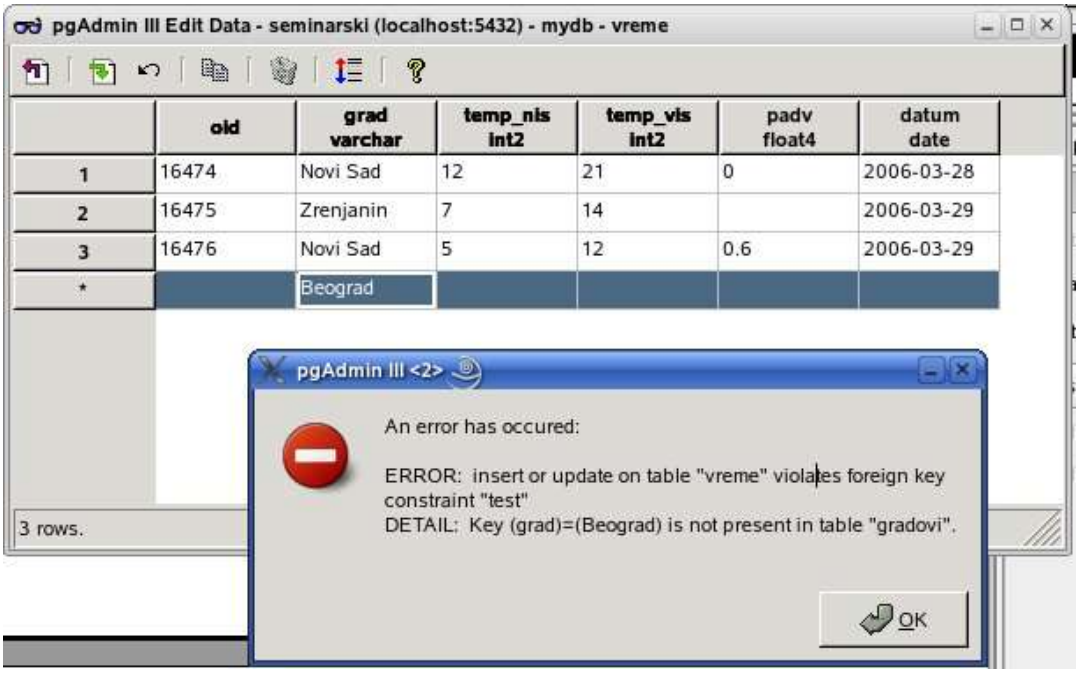
Na sledećem tabu se definiše zavisnost kolona između dve tabele, u ovom slučaju grad i naziv (slika 22). Na slici 23 su prikazane sve moguće kombinacije vezane za Foreign Key.

slika 22.



slika 23.

Ukoliko sada pokusamo da unesemo Beograd u tabelu vreme, neće nam biti dozvoljeno budući da u tabeli gradovi ne postoji odgovarajući zapis (slika 24).



slika 24.

3.10.3. Transakcije

Transakcije su fundamentalni koncept svih sistema baza podataka. Smisao transakcija je da poveže više koraka u jednu, sve-ili-ništa operaciju. Središnje postavke između koraka nisu vidljive drugim konkurentnim transakcijama, i ukoliko se desi neka greška upravo to sprečava kompletiranje transakcije tako da nijedan korak neće uticati na samu bazu.

Na primer, baza neke banke sadrži bilanse za različite klijentske naloge, kao i totalne bilanse za filijale. Pretpostavka je da se želi snimiti uplata od 100000 dinara sa Alisinog računa na Bobanov račun. SQL komanda može da izgleda ovako:

```
UPDATE nalozi SET bilans = bilans - 100.00
    WHERE ime = 'Alisa';
UPDATE filijala SET bilans = balance - 100.00
    WHERE ime = (SELECT filijala_NS FROM nalozi WHERE ime = 'Alisa');
UPDATE nalozi SET bilans = bilans + 100.00
    WHERE ime = 'Boban';
UPDATE filijala SET bilans = bilans + 100.00
    WHERE ime = (SELECT filijala_ZR FROM nalozi WHERE ime = 'Boban');
```

Detalji ovih komandi trenutno nisu bitni, bitno je da je izvršeno nekoliko odvojenih osvežavanja da bi se izvršila ova operacija. Bankovni službenici su svakako zainteresovani da li su se sve ove operacije izvršile ili se nije izvršila nijedna. Tek će grupisanje svih navedenih update-ovanja u transakciju moći to da garantuje.

Takođe je potebna i garancija da će jednom izvršena i u bazu zabeležena transakcija biti zaista snimljena i da neće biti izgubljena čak i ukoliko se nakon toga desi pad sistema. Transakciona baza zato treba da je garantuje da su sva update-vanja uzrokovana transakcijom backup-ovana, sačuvana (npr na disk) pre nego što je izveštaj o završetku transakcije gotov.

Još je jedna stvar od velilke važnosti za transakcione baze je da kada se više transakcija odvija istovremeno, svaka od njih ne sme da vidi nekompletirane promene na bazi drugih transakcija. Znači, transakcija mora biti orijentisana na sve-ili-ništa ne samo kao posledica njenog trajnog uticaja na bazu već i kao uslov da ne bude vidljiva dok se ne zavrsi.

U PostgreSQL-u, transakcije su određene SQL komandama BEGIN i COMMIT. Tako će prethodna bankarska transakcija izgledati ovako:

```
BEGIN;
UPDATE nalozi SET bilans = bilans - 100.00
    WHERE ime = 'Alisa';
-- itd
COMMIT;
```

Ukoliko ne želimo da se transakcija izvrši (iz recimo razloga što je Alisin račun prešao u minus), koristi se komanda ROLLBACK umesto COMMIT.

PostgreSQL i inače tretira svaku SQL komandu kao da je izvršena u okviru transakcije. Ako se i ne krene sa BEGIN svaka individualna komanda implicitno počinje sa BEGIN i ako je uspešna završava sa COMMIT. Grupe komandi uokvirene sa BEGIN i COMMIT se ponekad nazivaju transakcioni blokovi.

Moguće je kontrolisati komande u transakciji još detaljnije koristeći savepoints. Oni omogućavaju selektivno uklanjanje delova transakcije, dok će ostatak biti izvršen. Nakon definisanja savepointa sa komandom SAVEPOINT, može se vratiti sa komandom ROLLBACK TO. Sve promene baze transakcijom između te dve komande su uklonjene. Budući da se sve dešava pod transakcionim blokom, ništa neće biti vidljivo za ostale sesije.

Primer savepointa se može izvesti na prethodnim primerima – pretpostavka je da je potrebno prebaciti određenu sumu novca sa Alisinog na Bobanov račun, ali se naknadno primeti da je bilo potrebno kreditirati Perin nalog:

```
BEGIN;
UPDATE nalozi SET bilans = bilans - 100.00
    WHERE ime = 'Alisa';
SAVEPOINT moj_savepoint;
UPDATE nalozi SET bilans = bilans + 100.00
    WHERE name = 'Boban';
-- greska... trebalo je koristiti Perin nalog
ROLLBACK TO moj_savepoint;
UPDATE nalozi SET bilans = bilans + 100.00
    WHERE ime = 'Pera';
COMMIT;
```

3.10.4. Nasleđivanje

Nasleđivanje je koncept iz objektno orijentisanih baza podataka i otvara interesantne nove mogućnosti prilikom dizajniranja baza podataka.

Kreiramo dve tabele – gradovi i prestonice. Naravno, prestonice su takođe gradovi ali se žele istaći prilikom izlistavanja svih gradova. U principu, mogu se napraviti ovakve dve tabele:

```
CREATE TABLE prestonice (
  ime        text,
  populacija real,
  nadm_vis   int,    -- (u metrima)
  drzava     char(2)
);
```

```
CREATE TABLE ne_prestonice (
  ime        text,
  populacija real,
  nadm_vis   int     -- (u metrima)
);
```

```
CREATE VIEW gradovi AS
  SELECT ime, populacija, nadm_vis FROM prestonice
    UNION
  SELECT ime, populacija, nadm_vis FROM ne_prestonice;
```

I ovakva postavka će funkcionisati korektno ali će update-ovanje redova biti prilično neelegantno. Zato je bolja solucija:

```
CREATE TABLE gradovi (
  ime        text,
  populacija real,
  nadm_vis   int     -- (u metrima)
);
```

```
CREATE TABLE prestonice (
  drzava      char(2)
) INHERITS (gradovi);
```

U ovom slučaju, redovi tabele `prestonice` *nasleđuju* sve kolone `(ime, populacija, i nadm_vis)` od svoje starije tabele `gradovi.` Tip kolone ime je `text,` prirodni PostgreSQL tip za dužinu varijabli. Tabela prestonice ima jednu kolonu više, drzava, koja će prikazati u kojoj državi je data prestonica. U PostgreSQL-u, tabela može naslediti od nule do više drugih tabela. U tabelu gradovi se ne unose upisi prestonica.
Na primer, sledeći upit pronalazi imena gradova, uključujući prestonice koje se nalaze na nadmorskoj visini većoj od 5 metara.

```
SELECT ime, nadm_vis
  FROM gradovi
  WHERE nadm_vis > 5;
```

Rezultat:

```
    ime     | nadm_vis
-----------+----------
 Novi Sad  |       80
 Zrenjanin |       80
 Petrograd |       10
 Beograd   |      116
 Moskva    |      200
 London    |       15
(6 rows)
```

U sledećem primeru, upit pronalazi sve gradove koji nisu prestonice a nalaze se na nadmorskoj visini većoj od 50 metara:

```
SELECT ime, nadm_vis
    FROM ONLY gradovi
    WHERE nadm_vis > 50;
```

Rezultat:

```
    ime     | nadm_vis
-----------+----------
 Novi Sad  |       80
 Zrenjanin |       80
(2 rows)
```

Ili, samo prestonice:

```
SELECT ime, nadm_vis
    FROM ONLY prestonice
    WHERE nadm_vis > 50;
```

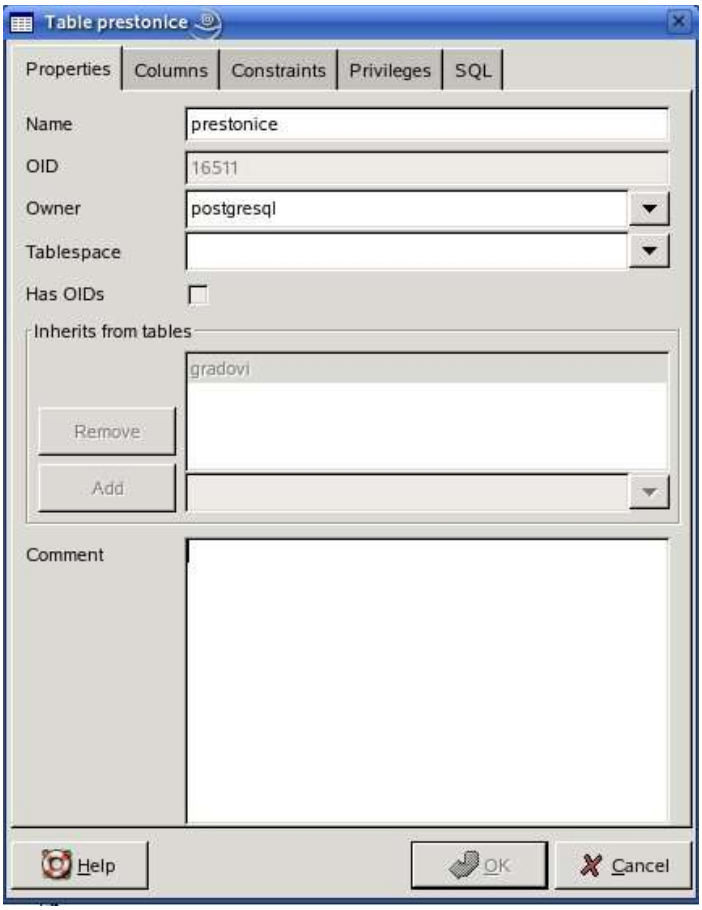
Rezultat:

```
   ime    | nadm_vis
---------+----------
 Beograd |      116
 Moskva  |      200
(2 rows)
```

Ovde komanda ONLY pre gradovi zahteva da se upit vrši samo nad tabelom gradovi, ne i u tabelama koje su zavisne od nje.

3.10.4.1. Nasleđivanje sa PgAdmin

Odabirom opcije Servers ✇seminarski ✇Databases ✇mydb ✇Schemas ✇public, desnim klikom na tabelu prestonice i odabirom opcije Properties, dobijamo stanje sa sledeće slike:



Nasleđivanje se vrlo lako rešava samo u ovom prozoru jednostavnim odabirom željene tabele i klikom na dugme Add.

slika 25.

POGLAVLJE 4 – *ADMINISTRACIJA SERVERA*

4.1. PostgreSQL korisnički nalozi

Kao i drugi deamon serveri koji su dostupni iz "spoljnjeg sveta" preporučljivo je pokretati PostgreSQL pod odvojenim korisničkim nalogom. Takav nalog treba da je vlasnik samo podataka kojima upravlja server, i ne treba biti deljen sa ostalim daemonima.

Za dodavanje korisnika na Unix sistemima koristi se komanda useradd or adduser kako je već prethodno navedeno.

4.2. Kreiranje klastera baze podataka

Prvo je potrebno definisati prostor na disku za bazu. SQL koristi termin katalog klaster a PostgreSQL klaster baze podataka. Klaster baze podataka je skup baza kojima se upravlja jednim pokretanjem date baze. Nakon inicijalizacije klaster baze podataka će pokretati bazu pod imenom postgres, koja je default baza. Sam server baze podataka ne zahteva postojanja pomenute baze, ali mnogi drugi pomoćni programi pretpostavljaju da ona postoji.

Klaster baze podataka će biti jedan direktorijum u kom će biti pohranjivani svi podaci. Naziv je *direktorijum podataka* ili *oblast podataka*. Po osnovnim postavkama nema tačno određenog direktorijuma koji će služiti za to, mada se najčešće koriste `/usr/local/pgsql/data` ili `/var/lib/pgsql/data`. Da bi se inicirao klaster baze podataka koristi se komanda initdb. Odabir lokacije u fajl sistemu se određuje komandom:

```
# initdb -D /usr/local/pgsql/data
```

initdb će pokušati da kreira direktorijum ukoliko već ne postoji. Ukoliko se ova komanda pokrenula pod neprivilegovanim korisnikom, direktorijum neće moći biti kreiran. U tom slučaju potrebno ga je "ručno" kreirati ulogovan kao root a potom PostgreSQL korisničkom nalogu dodeliti sva ovlaštenja. Komande će izgledati ovako:

```
su
root# mkdir /usr/local/pgsql/data
root# chown postgres /usr/local/pgsql/data
root# su postgres
postgres$ initdb -D /usr/local/pgsql/data
```

initdb neće prihvatiti pokretanje ukoliko direktorijum podataka već postoji.

Budući da su u direktorijumu podataka smešteni svi podaci neophodno je da on bude zaštićen od neautorizovanog pristupa. Zbog toga će initdb odbijati pristup svakome osim PostgreSQL korisniku.

Kako god, dok je sadržaj direktorijuma osiguran, po osnovnim postavkama autorizacije klijenta dozvoljeno je konektovati se na bazu podataka i čak postati superkorisnik. Zato se preporučuje korišćenje jedne od dve opcije: `initdb`'s `-W`, `--pwprompt` ili `--pwfile` za dodeljivanje lozinke superkorisniku baze podataka.

4.3. Pokretanje servera baze podataka

Pre nego što se pristupi bazi podataka potrebno je pokrenuti server. Program servera baze podataka se zove postmaster i on mora znati gde se nalaze podaci koje treba da koristi. Komanda

```
# postmaster -D /usr/local/pgsql/data
```

ostavlja server da radi vidljiv. Ova komanda se može koristiti samo ukoliko je ulogovan PostgreSQL korisnički nalog. Bolje je koristiti postmaster tako da radi u pozadini:

```
# postmaster -D /usr/local/pgsql/data >logfile 2>&1 &
```

Program postmaster poseduje veliki broj drugih opcija u komandnoj liniji, što brzo može postati naporno za korišćenje. Zato se može koristiti program pg_ctl, na primer komanda:

```
pg_ctl start -l logfile
```

pokreće server u pozadini i postavlja izlaz u unapred određeni log fajl.

Najbolje je naravno da se server startuje prilikom podizanja operativnog sistema. Skripte za autostart su specifične za svaki operativni sistem, a uz PostgreSQL dolazi nekoliko skripti u `contrib/start-scripts` direktorijumu. Instaliranje novih zahteva root privilegije.

Različiti operativni sistemi imaju različite konvencije za startovanje daemona prilikom podizanja sistema. Mnogi sistemi imaju fajl `/etc/rc.local` ili `/etc/rc.d/rc.local`. Drugi koriste `rc.d` direktorijume, što je slučaj sa SuSE Linuxom. Kako god, server mora biti pokrenut pod PostgreSQL korisničkim nalogom a nikako pod root ili nekim drugim lokalnim nalogom. Komanda će biti oblika su `-c '...' postgres` na primer:

```
su -c 'pg_ctl start -D /usr/local/pgsql/data -l serverlog' postgres
```

Najzgodnije je pogledati u source skritpte `contrib/start-scripts/linux` u PostgreSQL distribuciji.

4.4. Moguće greške prilikom podizanja servera

Nekoliko grešaka se dosta često ponavljaju. Prilikom podizanja servera je moguće dobiti poruku:

```
LOG:  could not bind IPv4 socket: Address already in use
HINT:  Is another postmaster already running on port 5432? If not, wait a
few seconds and retry.
FATAL:  could not create TCP/IP listen socket
```

Najčešće ovo znači da se pokušano pokretanje još jednog postmastera na već zauzetom portu (od strane drugog postmastera).

Ili, ukoliko je prijavljena `Address already in` use ili neka varijanta te greške od strane kernela, problem je druge prirode. Na primer, ukoliko želimo pokrenuti postmaster na nekom rezervisanom portu:

```
# postmaster -p 222
LOG:  could not bind IPv4 socket: Permission denied
HINT:  Is another postmaster already running on port 222? If not, wait a
few seconds and retry.
FATAL:  could not create TCP/IP listen socket
```

Ili greška:

```
FATAL:  could not create shared memory segment: Invalid argument
DETAIL:  Failed system call was shmget(key=5440001, size=4011376640, 03600).
```

najverovatnije znači da je limitirana veličina deljene memorije od strane kernela manja od radne oblasti koju PostgreSQL želi da kreira (4011376640 bitova u ovom primeru).

4.5. Moguće greške prilikom konektovanja klijenata

Mada su ovakve greške najčešće uzrokovane samom aplikacijom, moguće su i greške uzrokovane načinom kako je server startovan. Greška:

```
psql: could not connect to server: Connection refused
        Is the server running on host "server.joe.com" and accepting
        TCP/IP connections on port 5432?
```

Najčešće je uzrok ovakve greške neomogućavanje TCP/IP konekcije prilikom konfiguracije servera.

Ili:

```
psql: could not connect to server: No such file or directory
        Is the server running locally and accepting
        connections on Unix domain socket "/tmp/.s.PGSQL.5432"?
```

Poslednja linija je korisna pošto pokazuje da li klijent pokušava da se konektuje na pravo mesto. Ukoliko na traženom mestu zaista nije pokrenut server, tipična poruka će biti ili `Connection refused` ili `No such file or directory`, kako je i prethodno ilustrovano. Pri tome, treba imati na umu da `Connection refused` ne

znači da je server prihvatio konekciju a zatim je odbio.

Druge poruke, tipa `Connection timed` out najverovatnije znači da je mrežna veza prekinuta.

4.6. Gašenje servera

Postoji nekoliko načina za gašenje servera. Način gašenja se može kontrolisati slanjem različitih signala postmaster procesima:

SIGTERM

Nakon primanja ovog signala, server ne dozvoljava nove konekcije ali dozvoljava postojećim sesijama da završe svoj posao bez ometanja. Ovo je takozvano *pametno gašenje* (*Smart Shutdown*).

SIGINT

Server ne dozvoljava nove konekcije i šalje svim postojećim procesima SIGTERM, što će izvršiti momentalni prekid svih transakcija. Tada se čeka završavanje procesa samog servera i nakon toga se ovaj gasi. Ovo je *brzo gašenje* (*Fast Shutdown*).

SIGQUIT

Ovo je momentalno gašenje (*Immediate Shutdown*). Preporučuje se samo u najhitnijim slučajevima, server se ne gasi pravilno što uzrokuje oporavak prilikom sledećeg startovanja.

4.7. Sigurne TCP/IP konekcije sa SSH tunelima

Prvo je potrebno uveriti se da je podignut SSH server na istoj mašini na kojoj je i PostgreSQL server i da je moguće ulogovati se na koristeći SSH kao neki od postojećih korisnika (npr. root - u krajnjem slučaju).

Tada se uspostavlja tunel sa klijentske mašine komandom:

```
ssh -L 3333:neki_server.co.yu:5432 neko@neki_server.co.yu
```

Prvi broj, 3333 je broj porta na klijentskom kraju tunela – on može biti odabran slobodno. Sledeći broj, 5432 je broj porta udaljenje mašine – broj porta koji koristi server. Ime neki_server.co.yu ili IP adresa npr 212.62.xx.xx je ime mašine na kojoj se nalazi PostgreSQL server. neko je naziv korisničkog naloga na udaljenoj mašini. Prvo se konektuje na port 3333 na lokalnoj mašini:

```
psql -h localhost -p 3333 postgres
```

Server proverava da li je zaista u pitanju korisnik neko@neki_server.co.yu Ukoliko je dozvoljena sigurna ssh konekcija za pomenutog korisnika, podešavanje tunela će biti uspešno.

4.8. Uloge i privilegije baze podataka

PostgreSQL upravlja dozvolama za pristup bazi podataka koristeći koncept *uloga* (*roles*). Uloga može biti organizovana kao korisnik baze podataka ili kao grupa korisnika. Uloge mogu biti vlasnici objekata baze podataka (tabela, npr) i mogu dodeljivati privilegije za navedene objekte drugim ulogama, čak je moguće dodeljivanje *članstva* drugim ulogama dozvoljavajući drugim ulogama privilegije koje i sami imaju.
Koncept uloga obuhvata koncept “korisnika” i “grupa”. U verzijama PostgreSQL-a starijim od verzije 8.1. korisnici i grupe su bili odvojene vrste entiteta, ali od pomenute verzije u pitanju su samo uloge. Bilo koja uloga se može ponašati kao korisnik, grupa ili oboje.

4.8.1. Uloge

Za kreiranje uloge koristi se komanda CREATE ROLE :

```
CREATE ROLE ime;
```

Za uklanjenje uloge koristi se komanda DROP ROLE:

```
DROP ROLE ime;
```

Udobnosti radi, programi createuser i dropuser su omogućeni za korišćenje kao omoti oko SQL komandi koje mogu biti pozvane iz komandne linije terminala:

```
createuser ime
```

```
dropuser ime
```

Da bi se utvrdio set postojećih uloga potrebno je istražiti pg_roles katalog, na primer:

```
SELECT rolname FROM pg_roles;
```

Novopodignuti server uvek ima jednu već definisanu ulogu, ulogu superkorisnika – po osnovnim postavkama ime je ime korisnika operativnog sistema, koji je inicijalizirao klaster baze podataka. Najčešće je to ime postgres. Da bi se oformile druge uloge prvo je potrebno konektovati se kao navedenii superkorisnik.

4.8.2. Osobine uloga

Uloga može imati mnogo osobina koje će definisati njene privilegije i komunicirati sa sistemom za autorizaciju klijenta.

privilegija za logovanje

Samo uloge koje imaju LOGIN osobinu mogu biti korišćene kao inicijalne uloge za konekciju na bazu podataka. Takve uloge se mogu smatrati kao “korisnicima baze podataka”. Dve moguće komande:

```
CREATE ROLE ime LOGIN;
```

```
CREATE USER ime;
```

(CREATE USER je ekvivalnetno sa CREATE ROLE s tim da prvi prihvata LOGIN po početnim postavkama, dok drugi ne)

status superkorisnika

Superkorisnik baze podataka zaobilazi sve provere dozvola. U principu, to je opasna privilegija i ne bi je trebalo koristiti bez brige, uvek se preporučuje da se najveći deo posla obavlja sa ulogom koja nije superkorisnik. Za kreiranje novog superkorisnika, koristi se komanda

```
CREATE ROLE ime SUPERUSER
```

kreiranje baze podataka

Uloga mora imati eksplicitnu dozvolu za kreiranje baze (osim za superkorisnika). Komanda:

```
CREATE ROLE ime CREATEDB.
```

kreiranje uloge

Uloga mora imati eksplicitnu dozvolu za kreiranje drugih uloga (osim za superkorisnika). Ovako kreirana uloga može izmeniti ili poništiti druge uloge. Komanda

```
CREATE ROLE ime CREATEROLE
```

lozinka

Lozinka je značajna samo kada je metod za autorizaciju klijenta zahteva prilikom konektovanja na bazu. Ove lozinke nemaju nikakvu vezu sa lozinkama za operativni sistem. Komanda:

```
CREATE ROLE ime PASSWORD 'string'
```

4.8.3. Privilegije

Kada je neki objekat kreiran, dodeljen mu je vlasnik, najčešće je to uloga koja ga je kreirala. Za najveći broj objekata, ta uloga je jedini vlasnik (ili superkorisnik) i jedino ona može da vrši bilo kakve operacije nad njom. Da bi se dozvolilo drugim ulogama da je koriste, moraju im se dodeliti privilegije. Postoji nekoliko vrsta privilegija: SELECT, INSERT, UPDATE, DELETE, RULE, REFERENCES, TRIGGER, CREATE, TEMPORARY, EXECUTE, i USAGE.

Da bi se privilegija dodelila, koristi se komanda GRANT. Tako da, ako je na primer pera postojeća uloga, a nalozi postojeća tabela, privilegija za apdejtovanje tabele se dodeljuje sa:

```
GRANT UPDATE ON nalozi TO pera;
```

Specijalno ime PUBLIC može biti korišćeno za dodeljivanje privilegije svakoj ulozi u sistemu. Pisanjem ALL umesto neke specifične privilegije dodeljuje privilegije za sve uloge koje su nad datim objektom.

Za opoziv svih privilegija se koristi REVOKE komanda:

```
REVOKE ALL ON nalozi FROM PUBLIC;
```

Specifične privilegije nad objektom koje ima njen vlasnik (npr modifikacija ili brisanje objekta) su samo u njegovom vlasništvu i ne mogu se dodeliti niti opozvati. No, vlasnik može sam sebi opozvati neke privilegije – na primer, da tabelu učini read-only za sebe.

4.8.4. Članstvo u grupi uloga

Često je zgodno napraviti grupe korisnika zbog olakšavanja posla sa dodeljivanjem privilegija. Na taj način je moguće dodeliti ili opozvati privilegiju svim članovima grupe odjednom. U PostgreSQL-u ovo je učinjeno tako što

se kreira uloga koja predstavlja grupu a onda se dodeli članstvo u grupi individualnoj korisničkoj ulozi.

Dakle, da bi se podesila grupna uloga, prvo se kreira uloga:

```
CREATE ROLE ime;
```

Zatim se komandama GRANT i REVOKE dodeljuju ili uklanjaju članovi:

```
GRANT grupna_uloga TO uloga1, ... ;
```

```
REVOKE grupna_uloga FROM uloga1, ... ;
```

Budući da ne postoji neka suštinska razlika između grupnih uloga i ne-grupnih uloga, moguće je dodeliti članstvo i u drugim grupnim ulogama.

Pripadnici uloge koji imaju INHERIT osobinu automatski koriste privilegije uloge čiji su članovi. Na primer, urađeno je sledeće:

```
CREATE ROLE pera LOGIN INHERIT;
CREATE ROLE admin NOINHERIT;
CREATE ROLE centar NOINHERIT;
GRANT admin TO pera;
GRANT centar TO admin;
```

Odmah nakon konektovanja kao korisnik pera, sesija baze podataka će koristiti privilegije dodeljene direktno peri a isto tako i privilegije dodeljene adminu, budući da pera nasleđuje adminove privilegije. No, privilegije dodeljene centru nisu dostupne iako je pera indirektno član centra, budući da je članstvo preko admina, koji ima osobinu nenasledljivosti.

Nakon:

```
SET ROLE admin;
```

sesija će imati samo one privilegije koje su odobrene adminu (ne i peri).

Nakon:

```
SET ROLE centar;
```

sesija će imati samo one privilegije dodeljene centru (ne peri ili adminu). Početne privilegije mogu biti vraćene sa bilo kojom od sledećih komandi:

```
SET ROLE pera;
SET ROLE NONE;
RESET ROLE;
```

Osobine uloga LOGIN, SUPERUSER, CREATEDB, i CREATEROLE se mogu smatrati specijalnim privilegijama ali se nikada ne nasleđuju

Za uklanjanje uloge, koristi se:

```
DROP ROLE ime;
```

POGLAVLJE 5 – *UPRAVLJANJE BAZAMA PODATAKA*

5.1. Uvod

Baza podataka je imenovana kolekcija SQL objekata. Generalno, svaki objekat baze podataka (tabele, funkcije itd) pripadaju samo jednoj bazi (osim svega nekoliko sistem kataloga, npr `pg_database`, koja pripada celom klasteru i dostupna je svakoj bazi u okviru klastera). Preciznije, baza podataka je skup šema a šeme sadrže tabele, funkcije i sl. Potpuna hijerarhija je: server, baza podataka, šema, tabela (ili neki drugi objekat, npr funkcija).

Prilikom konektovanja na server baze podataka, klijent mora da specificira naziv baze kojoj želi da pristupi. Nije moguće pristupiti na više baza istovremeno prilikom iste konekcije (ali aplikacije nisu ograničene u broju konekcija). Baze su fizički razdvojene.

Baza se kreira sa CREATE DATABASE komandom a briše DROP DATABASE komandom. Da bi se utvrdio broj postojećih baza, treba istražiti `pg_database` sistem katalog, na primer:

```
SELECT datname FROM pg_database;
```

5.2. Kreiranje baze podataka

Da bi se kreirala baza, PostgreSQL server mora biti pokrenut. Kao što je već prethodno navedeno, komanda je:

```
CREATE DATABASE ime;
```

Pošto je potebno biti konekovan na server da bi se izvršila CREATE DATABASE komanda, postavlja se pitanje kako kreirati *prvu* bazu podataka? Prva baza je uvek kreirana sa initdb komandom kada je oblast podataka inicijalizovana. Ta baza se zove `postgres` i zato je za kreiranje prve "obične" baze potrebno konektovati se na nju.

Druga baza, `template1`, je takođe kreirana sa `initdb`. Kada god se kreira nova baza, klonira se `template1`.

To u principu znači da svaka promena nad bazom template1 utiče na sve buduće kreirane baze. Stoga, template1 ne bi trebala da služi za stvaran rad.

Kao olakšanje postoji program koji se može pokrenuti iz terminala a kojim se kreira baza - `createdb`.

```
createdb bpime
```

Ponekad je potrebno kreirati bazu za nekog drugog. Ta uloga bi trebala da bude vlasnik nove baze. Komanda za SQL okruženje:

```
CREATE DATABASE bpime OWNER ulogaime;
```

Ili u terminalu

```
createdb -O ulogaime bpime
```

Naravno, potrebno je biti supekorisnik da bi se mogla kreirati baza za nekog drugog.

5.3. Šabloni baza podataka

CREATE DATABASE funkcioniše tako što kopira postojeću bazu. Po početnim pretpostavkama, kopira standardnu sistemsku bazu `template1`. Ukoliko se bilo šta izmeni na navedenoj bazi, to će uticati na sve ostale baze koje se budu kreirale. Na primer, ukoliko se instalira proceduralni jezik PL/pgSQL u `template1`, on će automatski biti dostupan korisnikovim bazama.
Postoji i druga standardna sistemska baza po imenu template0. Ta baza sadrži iste podatke kao i `template1`, samo se nikada ne menja te je korisna za naknadna kreiranja baza podataka koja nemaju potrebe za izmenama učinjenim u `template1`.

Za kreiranje baze kopiranjem `template0` koristi se komanda za SQL okruženje:

```
CREATE DATABASE bpime TEMPLATE template0;
```

Ili u terminalu

```
createdb -T template0 bpime
```

Moguće je kreirati i sopstveve šablone baza podataka.

5.4. Tablespaces

Tablespaces u PostgreSQL-u dozvoljavaju administratoru da definiše lokacije u fajl sistemu gde fajlovi koji reprezentuju objekte baze podataka mogu biti pohranjeni. Prva prednost korišćenja tablespaces je da ukoliko na određenoj particiji hard diska ponestane prostora, može se napraviti tablespaces na drugoj particiji dok se sistem ne prekofiguriše. Druga prednost jeste optimizacija performasni. Na primer, moguće je smestiti određene “teške” podatke na brže diskove a retko korišćene podatke na obične diskove.

Za definisanje tablespace, koristi se komanda CREATE TABLESPACE na primer:

```
CREATE TABLESPACE fastspace LOCATION '/mnt/sda1/postgresql/data';
```

Inicijalno kreiranje tablespaces je dozvoljeno samo superkorisniku, ali nakon toga se može omogućiti privilegija bilo kom korisniku za njeno korišćenje. Da bi se to uradilo potebno je dodeliti CREATE privilegiju.

Tabele, indeksi, i cele baze mogi biti dodeljene posebnim tablespaces. Da bi se to uradilo, korisnik sa CREATE privilegijom nad datim tablespaces mora da prosledi njegovo ime kao parametar za relevantnu komandu. Na primer, sledeća komanda kreira tabelu u tablespace space1:

```
CREATE TABLE foo(i int) TABLESPACE space1;
```

Za uklanjanje praznog tablespace, koristi se komanda DROP TABLESPACE

Da bi se utvrdio broj postojećih tablespaces, treba istražiti pg_ tablespace sistem katalog, na primer:

```
SELECT spcname FROM pg_tablespace;
```

POGLAVLJE 6 – *BACKUP* i *RESTORE*

Kao i sve što sadrži dragocene podatke, i PostgreSQL bi trebalo bekapovati (praviti rezervnu, sigurnosnu kopiju) redovno. Postoje tri fundamentalno različita pristupa za bekapovanje PostgreSQL podataka.

- SQL dump
- Bekap fajl sistem nivoa
- On-line bekap

Svaki od ova tri pristupa ima svoje snage i slabosti.

6.1. SQL Dump

Ideja metoda jeste generisanje tekstualnog fajla sa SQL komandama tako da nakon uvođenja na server ponovo kreira bazu onakva kakva je bila u trenutku izvršenja SQL Dumpa. PostgreSQL za ovu svrhu obezbeđuje pomoćni program pg_dump. Osnovna upotreba ove komande je :

```
pg_dump bpime > outfile
```

Dakle. pg_dump zapisuje rezultat u standardni izlazni fajl.

pg_dump je redovna PostgreSQL klijentska aplikacija, što znači da se bekap može izvršiti sa bilo kog računara koji ima pristup bazi. Bitno je razumeti da pg_dump ne funkcioniše sa specijalnim dozvolama, te se najčešće mora pokretati kao superkorisnik baze.

6.1.1. Obnavljanje dump-a

Tekstualni fajl kreiran uz pomoć pg_dump se čita u psql programu. Osnovna komanda za obnavljanje, ponovno učitavanje dump-a je:

```
psql bpime < infile
```

gde je `infile` ono što se koristilo kao `outfile` u pg_dump komandi. Baza *bpime* neće biti kreirana ovom komandom, baza se mora kreirati iz template0 pre pokretanja psql-a (na primer komandom

```
createdb -T template0 bpime
```

).

Ne samo da željena baza mora postojati pre obnavljanja, nego moraju postojati i svi korisnici koji su bili vlasnici objekata u bekapovanoj bazi. Naravno, ukoliko neki od tih korisnika ne bude postojao, obnavljenje baze se neće izvršiti zbog greške u zapisu o vlasništvu i dozvolama nad objektima.

Kada je baza obnovljena, savetuje se pokretanje ANALYZE. Komanda:

```
vacuumdb -a -z to VACUUM ANALYZE all databases;
```

pg_dump i psql imaju mogućnost dump-a baze direktno sa jednog servera na drugi, na primer

```
pg_dump -h host1 bpime | psql -h host2 bpime
```

Za bekapovanje celog klastera koristi se program pg_dumpall, program koji bekapuje svaku bazu u klasteru i ujedno čuva podatke o korinicima i grupama. Komanda:

```
pg_dumpall > outfile
```

Obnavljanje:

```
psql -f infile postgres
```

6.1.2. Upravljanje velikim bazama

Kako PostgreSQL dozvoljava tabele veće od maksimalne veličine datoteke na operativnom sistemu javlja se problem damp-ovanja takve tabele u datoteku. Budući da pg_dump može da zapisuje u standardni izlaz, dovoljno je koristiti standardne Unix alatke:

Kompresovanje dump-a – može se koristiti bilo koji program za kompresovanje, na primer gzip

```
pg_dump bpime | gzip > imefajla.gz
```

Ponovo se pokreće sa

```
createdb bpime
gunzip -c imefajla.gz | psql bpime
```

ili

```
cat imefajla.gz | gunzip | psql bpime
```

Deljenje datoteka sa split. Datoteka se može podeliti na više delova željene veličine koje se prihvatljive za fajl sistem. U sledećem primeru datoteka je podeljena u n delova veličine 1mb

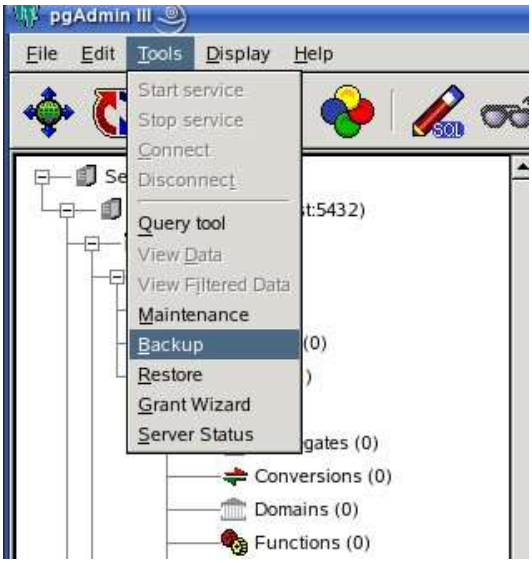
```
pg_dump bpime | split -b 1m - imefajla
```

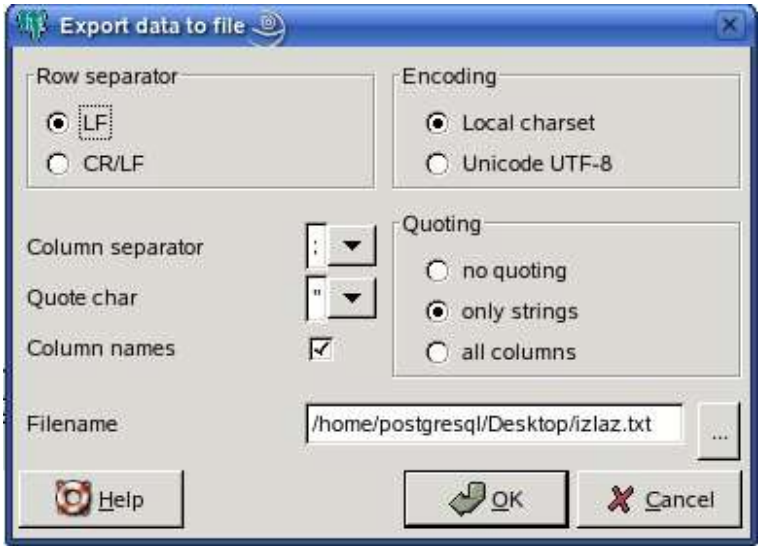
Ponovo se pokreće sa

```
createdb bpime
cat imefajla* | psql bpime
```

6.1.3. Backup sa PgAdmin

Backup je krajnje jednostavan, baš kako je prikazano na slikama 26 i 27.





slika 26. **6** *slika 27.*

6.2. Bekap fajl sistem nivoa

Alternativna bekap strategija je direktno kopiranje fajlova koje PostgreSQL koristi za smeštanje podataka u bazu. Na primer:

```
tar -cf backup.tar /usr/local/pgsql/data
```

Dva su ograničavajuća faktora koja ovaj metod čine nepraktičnim ili bar lošijim od pg_dump metoda:

1. Server *mora* biti ugašen da bi se dobio upotrbljiv bekap.
2. Nemoguće je bekapovati samo određene delove baze.

6.3. On-line bekap i point-in-time recovery (PITR)

Sve vreme, PostgreSQL održava *write ahead log* (WAL) u `pg_xlog/` poddirektorijumu klasterovog data direktorijuma. Log opisuje svaku promenu nad bazom i primarno postoji zbog sigurnosnih razloga: ukoliko se desi

pad sistema, baza može biti podignuta "ponavljanjem" događaja.
Moguće je kombinovati bekap fajl sistem nivoa i bekap WAL datoteka. Ukoliko je bekap potreban, uradi se restore bekapa a onda se ponavljenjem događaja baza dovodi do poslednjeg stanja. Ovaj pristup je najsloženiji, ali ima neke važne prednosti:

- Nije potreban potpuno dosledan bekap u samom startu. Svaka nedoslednost, nekonzistentnost će biti ispravljena iz postojećeg loga.
- Nije neophodno ponoviti WAL datoteke od starta do samog kraja. Samim tim, ova tehnika podržava *point-in-time recovery* – moguće je obnoviti bazu na stanje kakvo je bilo u bilo kom vremenskom periodu.
- Ukoliko se kontinualno WAL datoteke pohranjuju na neku drugu mašinu, postojaće još jedna kopija baze.

Uz pomoć ovog metoda moguće je obnoviti celi klaster baze podataka ali ne i podgrupe. Takođe, zahtevan je i po pitanju veličine podataka i količini mrežnog saobraćaja, no i dalje je to najpreporučljivija metoda u situacijama kada se zahteva visoka pouzdanost.

Korišćenje metode on-line bekapa zahteva neprekinutu sekvencu WAL datoteka koje se vremenski prostiru najmanje od početka bekapa. Dakle, pre nego se počne sa bekapom, neophodno je podesiti i testirati proceduru za arhiviranje WAL datoteka.

6.3.1. Podešavanje WAL arhiviranja

PostgreSQL sistem proizvodi neograničeno dugačku sekvencu WAL zapisa. Sistem fizički deli sekvencu u WAL segment datoteke koje su veličine 16MB. Imena segmenata su numerička i ona definišu njihovu poziciju u WAL sekvenci.

Arhiviranje WAL podataka je moguće na CD-u, prebacivanjem na drugu mašinu ili nešto tome slično. PostgreSQL pruža administratoru veliku fleksibilnost i dozvoljava mu izbor komande za kopiranje kompletnog segmenta gde god poželi. Komanda može biti sasvim jednostavna kao cp (copy).

Komanda može da izgleda ovako (gde će svaki %p biti zamenjen sa apsolutnom putanjom fajla koji će biti arhiviran a svaki %f sa imenom fajla):

```
archive_command = 'cp -i %p /mnt/server/archivedir/%f </dev/null'
```

Ova komanda kopira WAL segment koji će biti arhiviran u direktorijum /mnt/server/archivedir.

Neophodno je komanda za arhiviranje vrati zero exit status i tek tada možemo bitzi sigurni da je ono uspešno obavljeno. Tada će WAL segment biti obrisan. Nonzero status saopštava PostgreSQL-u da fajl nije arhiviran i ovaj će periodično pokušavati da izvrši komandu dok ne uspe.

Komanda za arhiviranje bi trebala da bude tako dizajnirana da odbija pisanje novog fajla preko starog. To je prvi razlog zbog koga treba izvršiti testiranje a drugi je da vraća *nonzero status* u ovom slučaju. Komanda:

```
archive_command = 'test ! -f .../%f && cp %p .../%f'
```

6.3.2. Bekapovanje baze

Procedura:

1. Uveriti se da je WAL arhiviranje omogućeno i da ispravno funkcioniše
2. Konektovati se na bazu, kao superkorisnik dati komadu

```
SELECT pg_start_backup('label');
```

gde je label neko ime pod kojim se želi sačuvati ova operacija bekapovanja (dobro je opredeliti se za

kompletnu putanju do meste gde se želi sačuvati bekap fajl)

3. Izvesti bekap, koristeći bilo koju fajl sistem alatku kao što su tar ili cpio

4. Ponovo se konetktovati na bazu kao superkorisnik i dati komandu

```
SELECT pg_stop_backup();
```

5. Kada su WAL segment fajlovi arhivirani kao deo redovne aktivnosti baze, proces je završen

6.3.3. Obnavljanje sa on-line bekap pristupom

Procedura:

1. Zaustaviti postmaster, ukoliko je pokrenut
2. Ukoliko postoji mesta, kopirati celi klaster direktorijuma podataka i tablespaces ne neku privremenu lokaciju za slučaj da zatrebaju naknadno.
3. Očistiti sve postojeće fajlove i poddirektorijume u klasteru direktorijuma podataka i pod root direktorijuma za svaki tablespaces koji se koristi.
4. Obnoviti bazu. Biti oprezan da se baza obnavlja pod pravilnim vlasništvom i sa pravilnim dozvolama (nikako kao root već kao korisnik baze podataka) .
5. Ukloniti bilo koji fajl prisutan u `pg_xlog/`; ukoliko `pg_xlog/`nije arhiviran uopšte, potrebno ga je ponovo kreirati i biti siguran da je kreiran i poddirektorijum `pg_xlog/archive_status/` takođe.
6. Ukoliko postoje nearhivirani WAL segment fajlovi koji su sačuvani u koraku 2, kopirati ih u `pg_xlog/`.
7. Kreirati fajl za komandu obnavljanja `recovery.conf` u klasteru direktorijuma podataka.
8. Pokrenuti postmaster. On će preimenovati fajl `recovery.conf` u `recovery.done`.
9. Proveriti sadržaj baze da bi se uverili da je obnavljanje uspešno urađeno. Ukoliko nije, potrebno je vratiti se na korak 1.

POGLAVLJE 7 – *ZAKLJUČAK*

Bilo bi veoma opširno na ovom mestu definisati i razmatrati prednosti (i mane) open source software-a, te će u najkraćim crtama biti navedeni samo pojedini detalji.
Osnovna prednost Linux-a jeste **stabilnost i sigurnost**. Prema kompaniji Symantec postoji oko 120000 virusa koji napadaju Windows i oko 500 koji napadaju Linux – Linux je sigurniji i manje ranjiv. Usled entuzijazma open source zajednice uočeni problem sa ranjivošću operativnog sistema se rešava prosečno za nedelju dana. Kod Windowsa OS se na "zakrpu" čeka prosečno oko 45 dana. U proseku, Windows "nezakrpljena" mašina bude napadnuta nakon jednog sata provedenog na internetu. **Cena OS** – cena od 130$ za (samo) Windows u odnosu na cenu skidanja Linux-a sa interneta ili cenu rezanja nekoliko praznih cd medija. Cene dodatnih programa znatno povećavaju ovu razliku. U zavisnosti od toga koja se distribucija Linuxa koristi, one uključuju i mogućnost instaliranja i do nekoliko hiljada programa tokom same instalacije sistema. **Podesivost i kontrola** Linux-a koji dozvoljava direktan pristup čitavom sistemu. O **kvalitetu** Linux-a govori podatak da oko 60% servera radi pod ovim operativnim sistemom, da ga koristi giganti pout NASA-e, Googla, IBM-a, Novella... Linux se koristi od desktop računara, preko mobilnih telefona i video uređaja do superkompjutera. Veliki broj preduzeća, banaka, univerziteta, internet provajdera i servera, a sad i sve veći broj korisnika koristu baš Linux za OS. Mnoge firme razmišljaju da nije potrebno ulagati hiljade evra na Microsoft-ove operativne sisteme koji nisu bezbedni, sigurni, a ni stabilni poput Linuxa. Sve je veći broj državnih administracija koje prelaze sa Windows-a na Linux.
Linux je zbog svoje cene naročito interesantan u zemljama u razvoju. Konkretno, u našem slučaju, nakon jedne decenije maksimalno rasprostranjene piraterije preduzeća i državne ustanove imaju pred sobom izbor – platiti mnogo za postojeći OS ili preći na Linux. Nedostaci prelaska na Linux su nedovoljna informatička obrazovanost zaposlenih gde je potrebno uložiti u njihovu obuku, i pre svega cena migracije.
Interesantan je i podatak da su UN preporučile svojim članicama korišćenje open source software-a, naročito u područjima zdravstva, prosvete i međunarodne trgovine. Naznačeno je da je prema UN open source najadekvatnije sredstvo za razvoj svojih članica.

PostgreSQL baza podataka, iako nije najbrža, na mnogim testovima je okarakterisana kao najnaprednija baza podataka. Inspirisana je Oraclom i od samog starta podržava transakcije, trigere, referencijalni integritet. Preporuke za njeno korišćenje su više u pravcu proverenog kvaliteta i robustnosti nego samih

performansi. Korisnici PostgreSQL-a su između ostalih UNICEF, Cisco i American Chemical Society.

Za kraj, prikazana je tabela odnosa nekoliko najpopularnijih baza[2]:

Feature	SQL Server	Oracle	MySQL	PostgreSQL
Open Source			X	X
Free / No License Costs			X	X
ACID Compliant		X		X
ANSI SQL Compliant	X	X		X
Referential Integrity	X	X		X
Replication	X	X	X	X
Rules	X	X		X
Views	X	X		X
Triggers	X	X		X
Unicode	X	X		X
Sequences		X	X	X
Inheritance		X		X
Outer Joins	X	X	X	X
Sub-selects	X	X		X
Open API			X	X
Stored Procedures	X	X		X
Native SSL Support	X	X	X	X
Procedural Languages	X	X		X
Indexes	X	X	X	X

[2] http://support.summersault.com/why_postgresql.html

DODATAK – *PgAccess*

PgAccess je još jedan grafički program za upravljanje PostgreSQL bazom. Budući da je PgAdmin III već predstavljan uporedo sa terminalski programom psql, ovde će u veoma kratkim crtama biti naznačene samo razlike PgAccessa u odnosu na PgAdmin, odnosno opcije koje PgAccess ima.

Prvo je potrebno skinuti ga sa interneta, npr sa adrese:

http://switch.dl.sourceforge.net/sourceforge/pgaccess/pgaccess-0_99_0_20040219.tgz

Postupak instalacije je sledeći:

```
tar xvfz pgaccess-0.99.0.20040219.tar.gz
cd pgaccess-0.99.0.20040219
su
make clean
make all
```

Pokreće se komandom

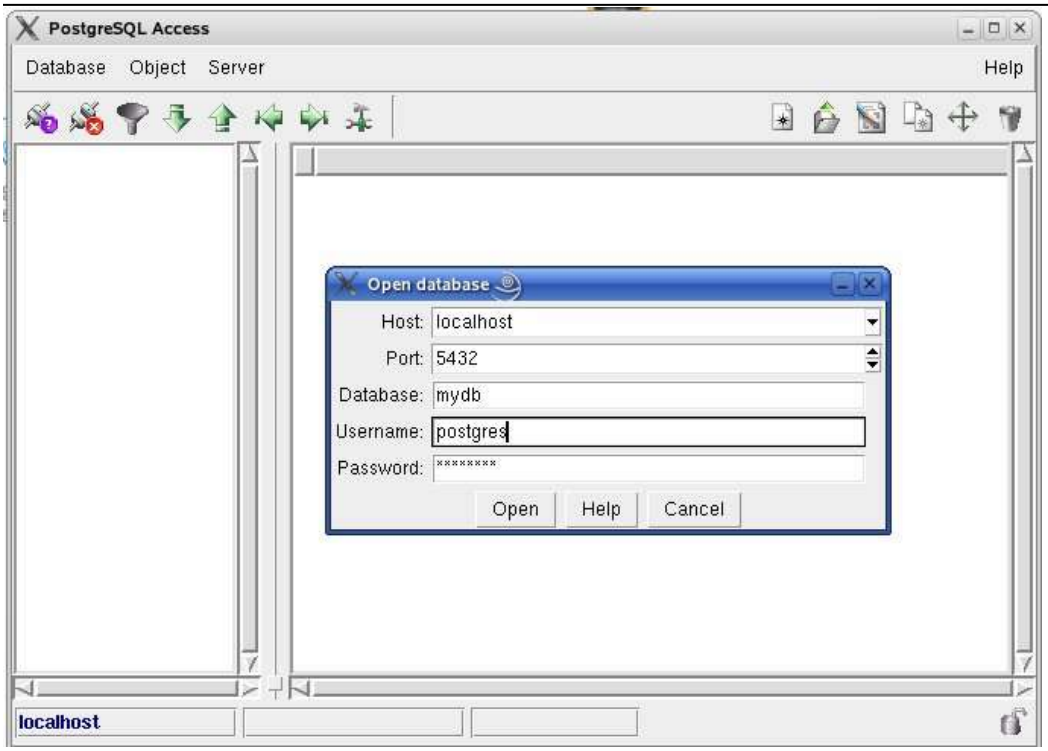
```
pgaccess
```

Gde je ime baze je npr `mydb`

Korisnik/user `postgres`

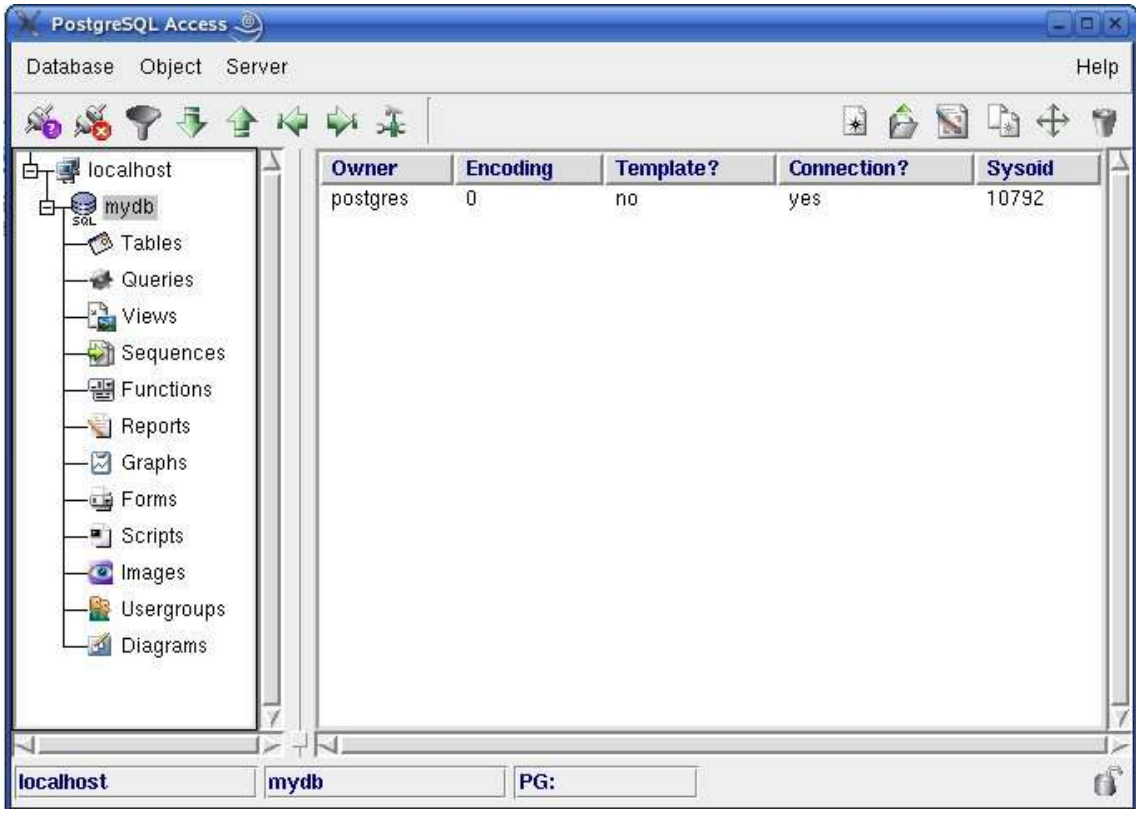
Lozinka/password – lozinka usera postgres

Dakle, kao na slecećoj slici:



slika 28.

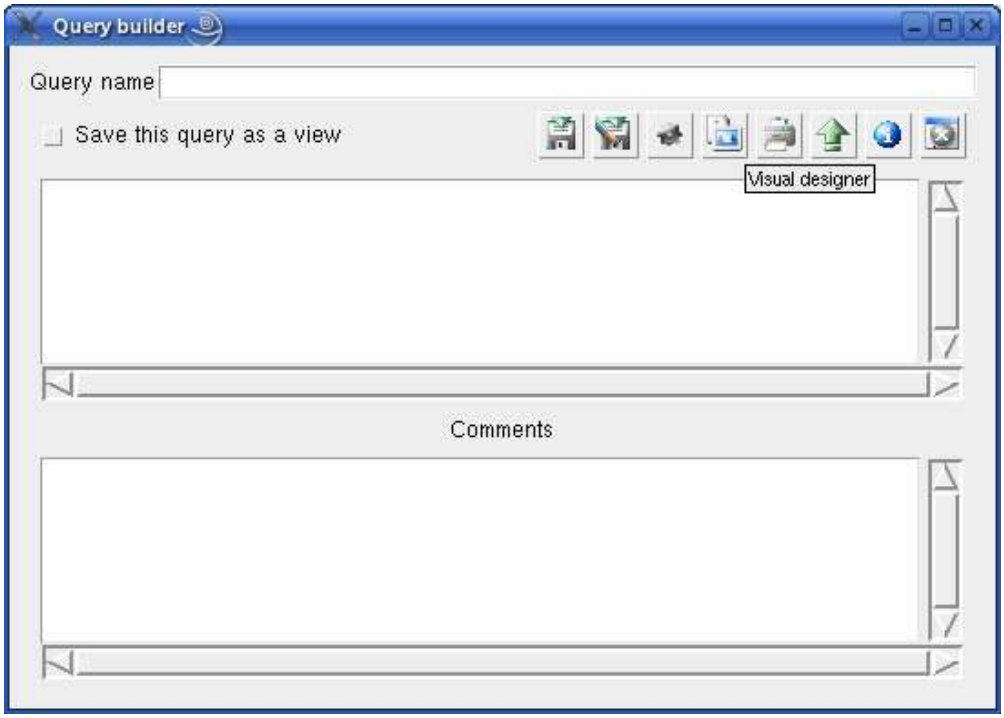
Nakon konektovanja na bazu, dobijamo prozor sa sledeće slike. U samom startu su očigledne razlike dva ovde spomenuta grafička programa za postgres. Ovde će samo slikom biti prikazane mogućnosti PgAccessa koje PgAdmin nema.



slika 29.

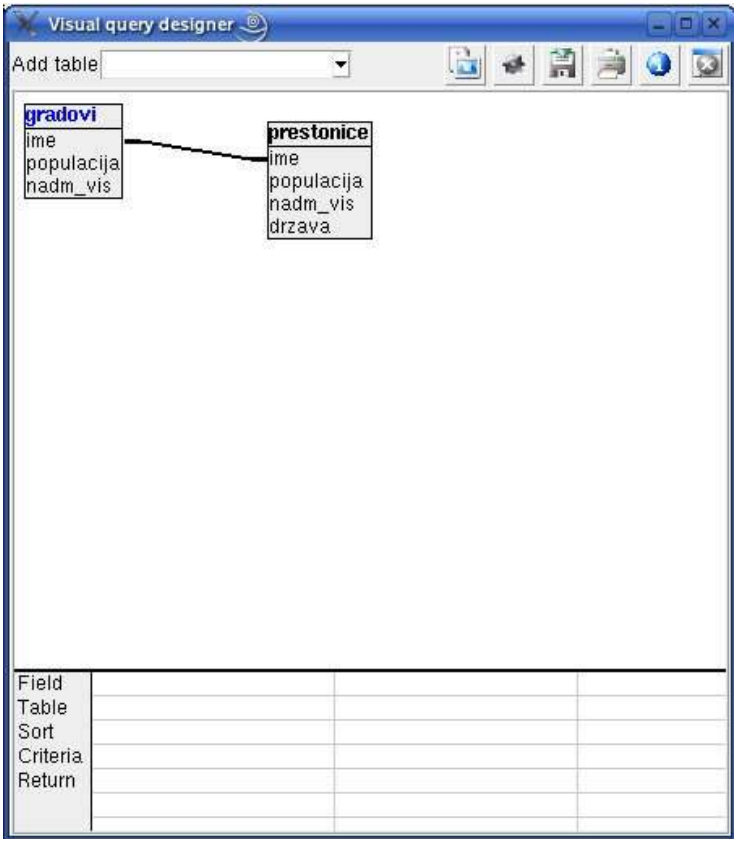
Visual designer

SQL upite je moguće izvršavati uz pomoć Query buildera,



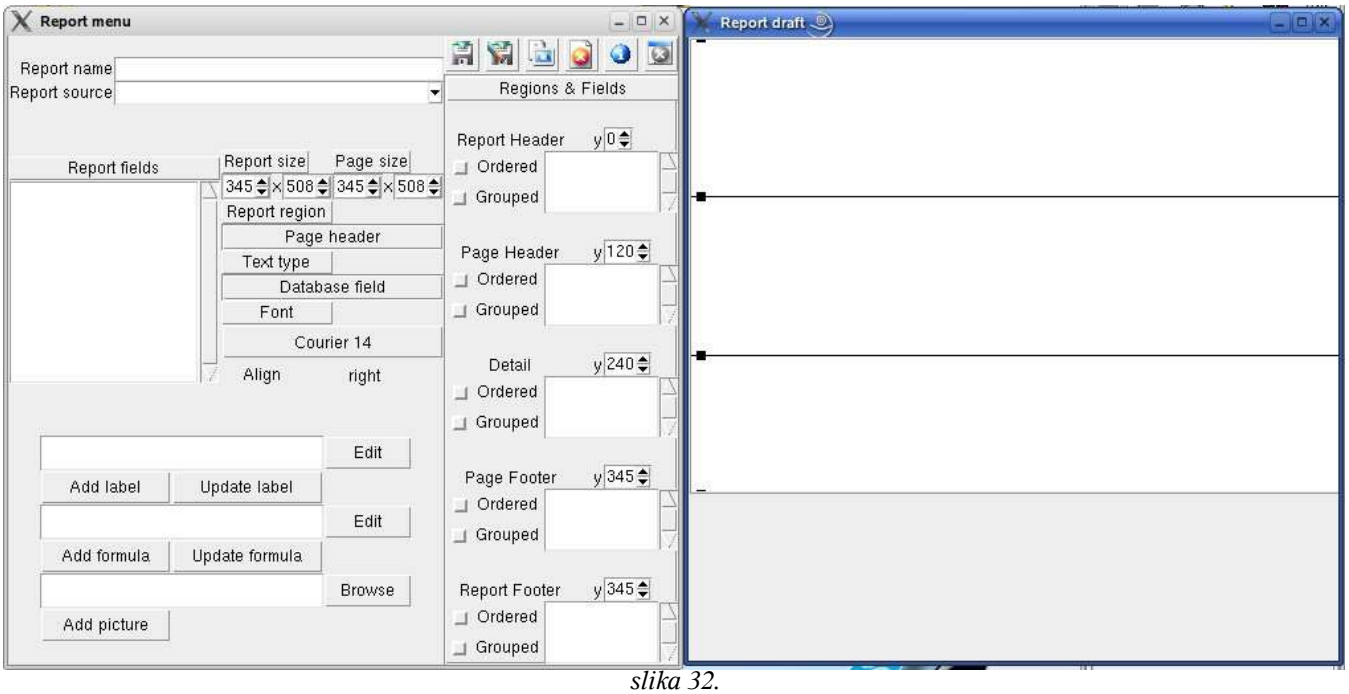
slika 30.

ali je iz Query buildera moguće pokrenuti Visual designer:



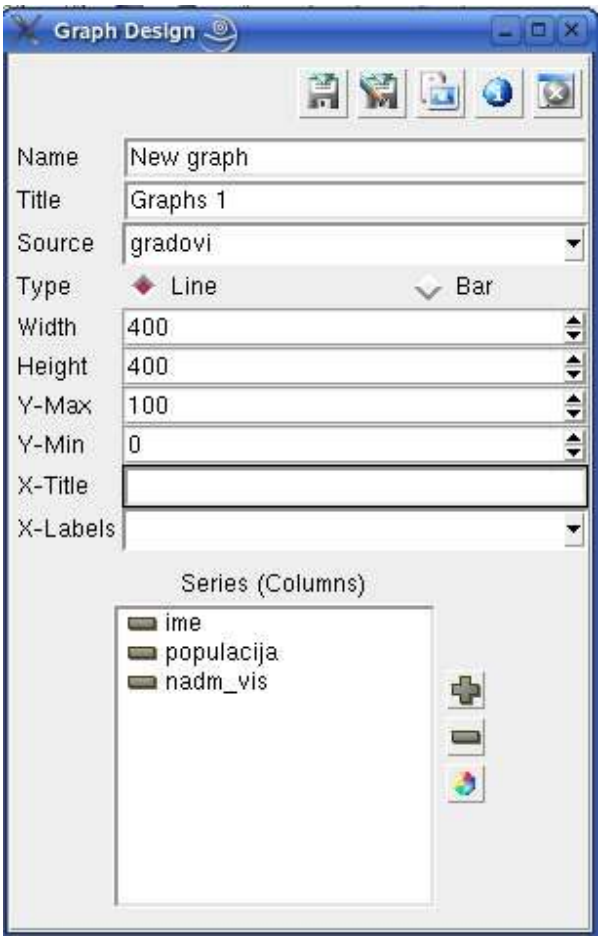
slika 31.

Reports



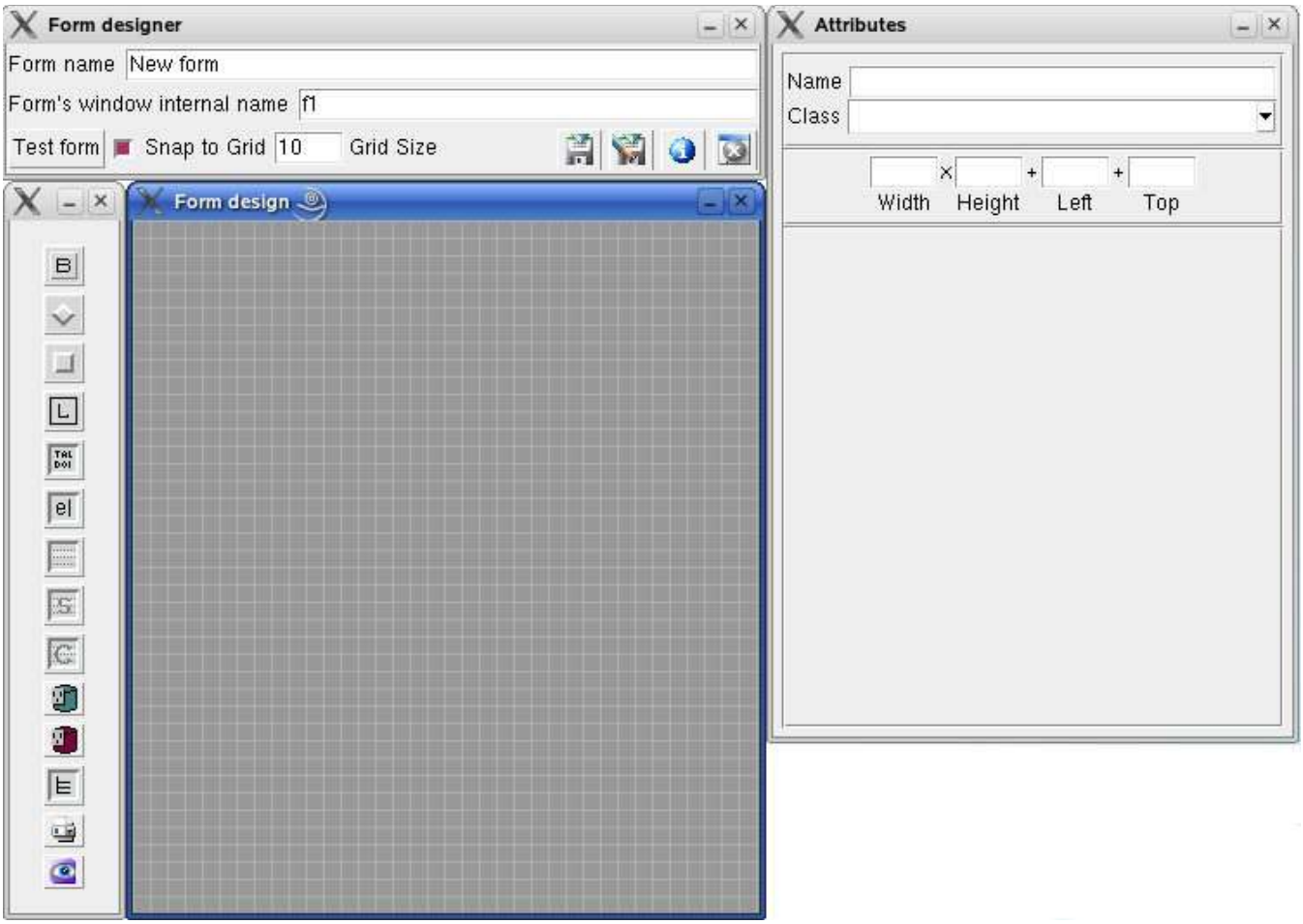
slika 32.

Grafikoni



slika 33.

Form design



slika 34.

LITERATURA

1. PostgreSQL 8.1.3 Documentation
2. Addison Wesley - PostgreSQL - Introduction and Concepts
3. O'Reilly -- SQL in Nutshell
4. O'Reilly - Practical PostgreSQL
5. Internet izvori:
    - www.postgreslq.org
    - http://gborg.postgresql.org/
    - www.pgaccess.org
    - http://www.felixgers.de/teaching/sql/index.html
    - http://pgadmin.postgresql.org/
    - http://www.flex.ro/pgaccess/tutorial/tut.html