REPUBLIKA SLOVENIJA
MINISTRSTVO ZA IZOBRAŽEVANJE, ZNANOST IN ŠPORT

Univerza *v Ljubljani*

EVROPSKA UNIJA
EVROPSKI
SOCIALNI SKLAD
NALOŽBA V VAŠO PRIHODNOST

Management podatkovnih baz v resničnem/poslovnem okolju - 1. del

Tadej Borovšak, XLAB d.o.o.

Uvod

Na današnji delavnici se bomo posvetili prvima dvema področjema uporabe podatkovnih baz v produkcijskem okolju. Pričeli bomo s pripravo varne namestitve PostgreSQL sistema za upravljanje podatkovnih baz in enostavnega programa, ki nam bo omogočil oddaljen dostop do baze. V drugem delu pa si bomo s pregledom izbranih primerov analize izvajanja poizvedb na prej postavljenem sistemu ogledali postopke za optimizacijo ter se spotoma seznanili z nekaterimi teoretičnimi temelji delovanja indeksov.

Postavitev produkcijskega PostgreSQL sistema

Namestitev PostgreSQL bomo pričeli s pripravo (virtualnega) računalnika, ki uporablja operacijski sistem CentOS 7[1]. Na sami delavnici bomo uporabili virtualizirano instanco, ki gostuje na OpenStack[2]-u, kar precej dobro posnema običajno produkcijsko okolje. Ker so navodila za pripravo instance specifična za OpenStack, ki ga bomo uporabljali, bo priprava predstavljena le praktično, dokumentirali pa bomo soroden postopek za pripravo instance s pomočjo VirtualBox[3]-a, ki pa ga lahko namesti in uporabi vsak.

Namestitev CentOS-a s pomočjo VirtualBox-a

Namestitev VirtualBox-a poteka kot vsaka druga namestitev programa, tako da je na tem mestu ne bomo opisali. Si bomo pa podrobno ogledali, kako ustvarimo novo virtualni instanco računalnika in kako nanjo namestimo operacijski sistem CentOS 7.

Ko odpremo VirtualBox, najprej v zgornjem levem kotu kliknemo na ikono **Nov ...**, kar nam odpre okno, v katerem izberemo ime in vrsto operacijskega sistema (slika 1).

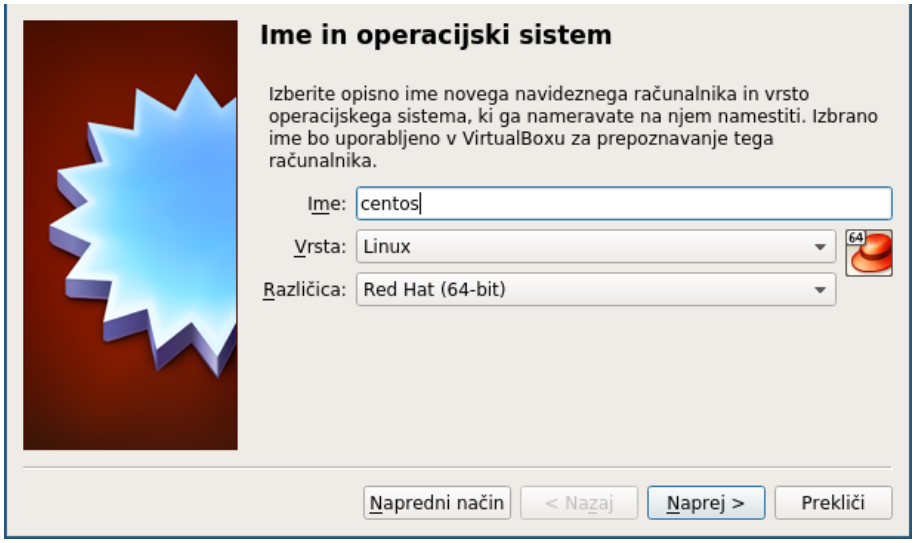

Figure 1: Izbira operacijskega sistema.

Na naslednjem koraku izberemo količino pomnilnika, ki ga želimo dodeliti instanci. Priporočena vrednost je spodnja meja, tako da rajši izberemo malce večjo vrednost, če nam naš gostiteljski

[1] https://www.centos.org/
[2] https://www.openstack.org/
[3] https://www.virtualbox.org/

sistem to dopušča (slika 2).

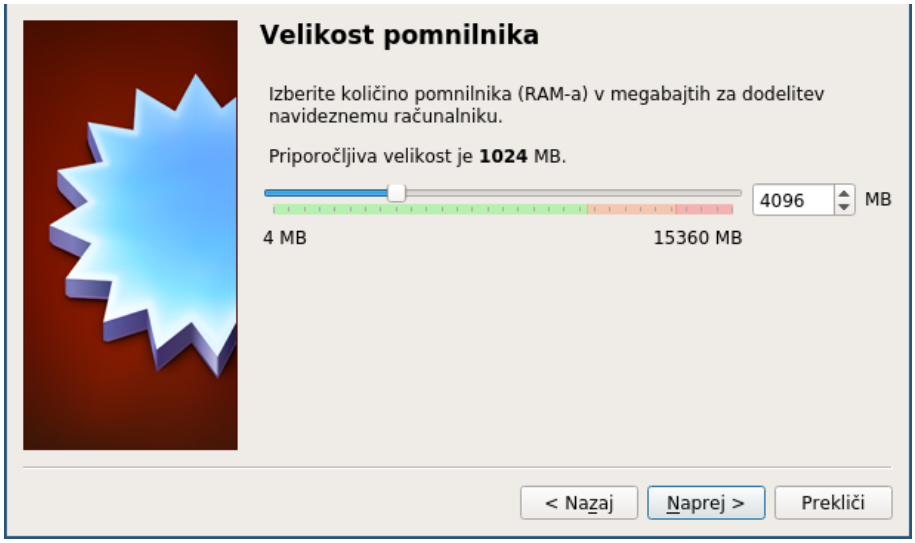

Figure 2: Dodelitev pomnilnika.

Za našo instanco moramo ustvariti tudi nov disk. Podobno kot prej, priporočena velikost je spodnja meja, tako da če se le da, uporabimo višjo vrednost (slike 3-6).

Figure 3: Ustvarjanje novega diska.

Ko dokončamo ustvarjanje diska, se bo v osnovnem pogledu pojavil virtualni računalnik z izbranim imenom, ki pa še nima nameščenega operacijskega sistema (slika 7). Preden pa lahko namestimo operacijski sistem, ga moramo prenesti s CentOS strani s prenosi[4]. Ker želimo namestiti strežniški sistem, izberemo minimalno ISO sliko in jo prenesemo na disk.

Ko imamo ISO sliko prenešeno, v VirtualBox-u izberemo možnost **Pomnilniške naprave** in prenos priklopimo v virtualni CD-ROM pogon kot je prikazano na sliki 8.

S klikom na gumb **Zaženi** bomo zagnali računalnik. Na prvem zaslonu izberemo **Install CentOS 7**, kar nam bo najprej odprlo okno z izbiro jezika, kjer izberemo slovenski jezik, da se nam

[4]https://www.centos.org/download/

Figure 4: Izbira vrste diska.

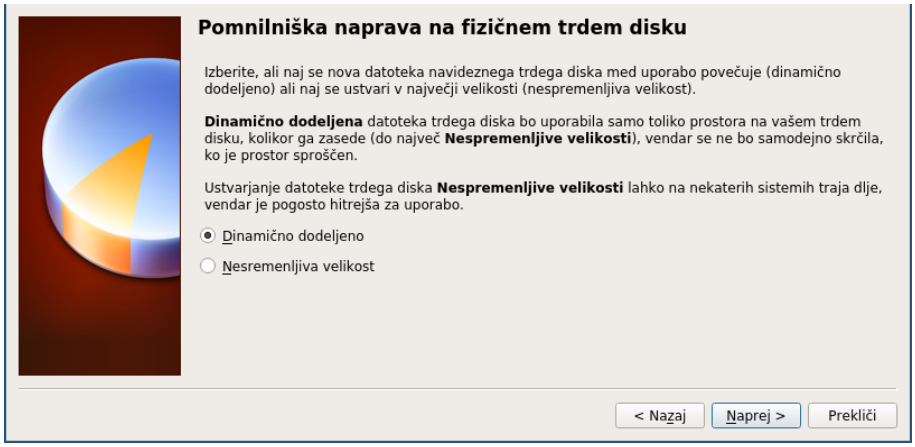

Figure 5: Pomnilniška naprava.

Figure 6: Določitev velikosti diska.

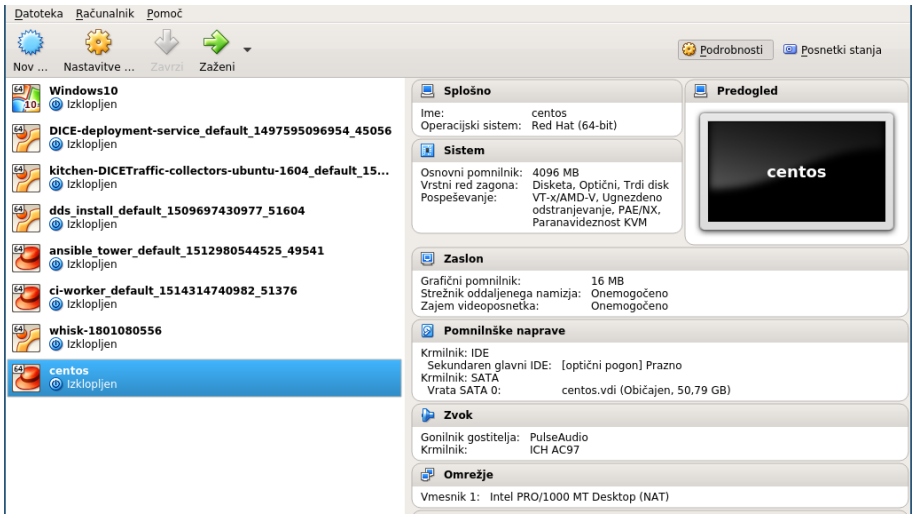

Figure 7: Nov virtualni računalnik.

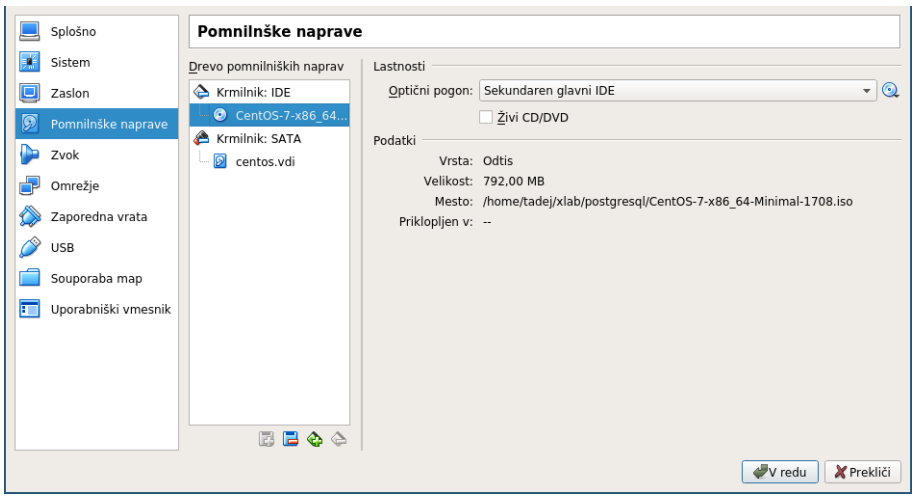

Figure 8: Priklop ISO slike.

ne bo potrebno ukvarjati z iskanjem pravih črk na tipkovnici, nakar se bomo znašli v glavnem namestitvenem programu (slika 9).

Figure 9: Začetek namestitve.

Na tem zaslonu lahko pustimo izbrane privzete vrednosti, poskrbeti moramo le, da imamo aktivirano omrežno povezavo (slika 10). S klikom na gumb **Begin Installation** lahko pričnemo namestitev.

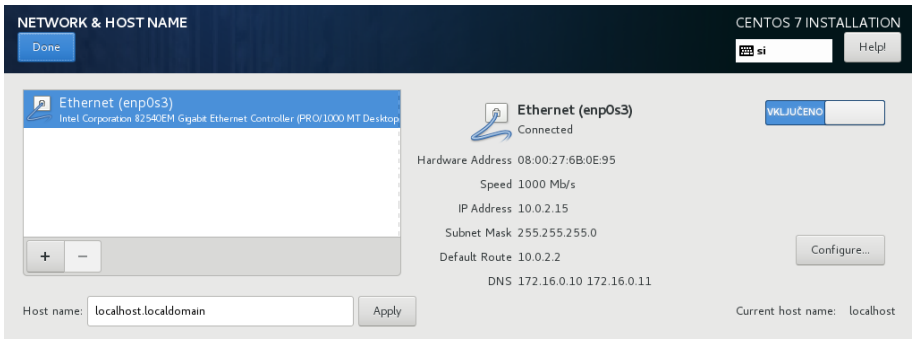

Figure 10: Omrežna povezava.

Medtem ko se izvaja namestitev, moramo ustvariti tudi novega uporabnika, kar je prikazano na sliki 11. Ime **centos** smo izbrali namenoma. Tako bo sistem zelo podoben tistemu, ki ga bomo uporabljali na sami delavnici.

Ko se osnovna namestitev konča, nas bo namestitveni program pozval še h konfiguraciji nameščenega sistema, ki jo pričnemo s klikom na gumb **Finish Configuration** (slika 12).

Po končanem konfiguracijskem koraku pa lahko sistem ponovno zaženemo, kar nas po postavilo v konzolno okno, iz katerega se lahko prijavimo v sistem, s čimer smo zaključili namestive operacijskega sistema (slika 13).

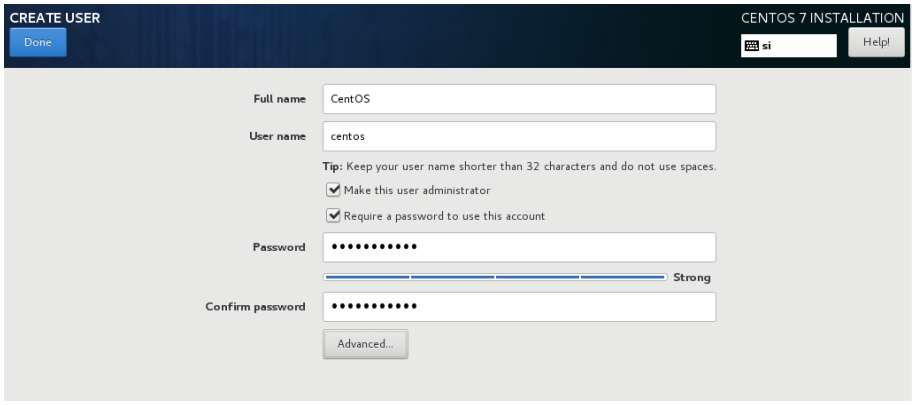

Figure 11: Ustvarjanje uporabnika.

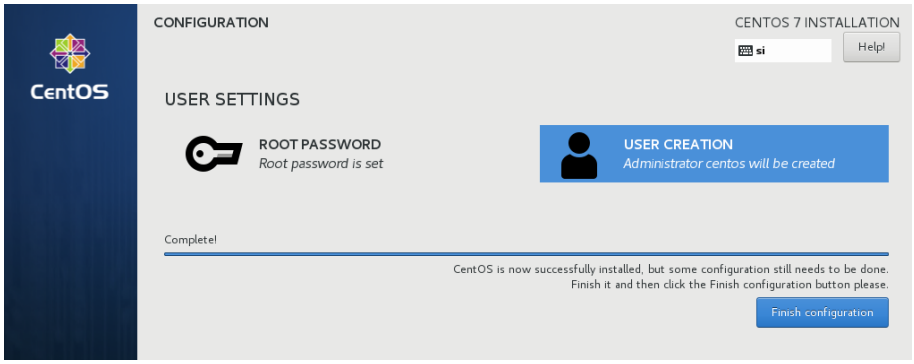

Figure 12: Konfiguracija sistema.

Figure 13: Prijava v sistem.

Namestitev PostgreSQL

Sedaj, ko je namestitev operacijskega sistema za nami, se lahko posvetimo nameščanju PostgreSQL-a. Za začetek moramo dodati nekaj yum repozitorijev, ki vsebujejo želene pakete:

```
$ sudo yum install epel-release
$ sudo yum install https://download.postgresql.org/pub/repos/yum/10/redhat/rhel-7-
    ↪ x86_64/pgdg-centos10-10-1.noarch.rpm
```

Prvi ukaz nas bo najprej povprašal po uporabniškem geslu, nakar bomo morali nekajkrat odgovoriti z **y** (seveda bomo prej prebrali obvestila). Drugi ukaz se bo obnašal precej bolj sproščeno in bo zahteval le eno potrditev.

Sedaj pa lahko namestimo in aktiviramo PostgreSQL:

```
$ sudo yum install postgresql10-server
$ sudo /usr/pgsql-10/bin/postgresql-10-setup initdb
$ sudo systemctl start postgresql-10
$ sudo systemctl enable postgresql-10
```

Da je PostgreSQL resnočno aktiven preverimo tako, da povprašamo **systemd**, kaj se dogaja z njim:

```
$ systemctl status postgresql-10
* postgresql-10.service - PostgreSQL 10 database server
   Loaded: loaded (/usr/lib/systemd/system/postgresql-10.service)
   Active: active (running) since sob 2018-02-17 18:05:35 UTC; 19h ago
     Docs: https://www.postgresql.org/docs/10/static/
  Process: 19763 ExecStartPre=/usr/pgsql-10/bin/postgresql-10-check-...
  ...
```

Še bolj neposredno pa se lahko o delovanju prepričamo, če se skušamo neposredno prijaviti:

```
$ sudo su - postgres
(postgres) $ psql
postgres=# \dS
                        List of relations
   Schema   |              Name                | Type  |  Owner
------------+----------------------------------+-------+----------
 pg_catalog | pg_aggregate                     | table | postgres
 pg_catalog | pg_am                            | table | postgres
...
```

Konfiguracija PostgreSQL

Sedaj sicer imamo bazo pripravljeno za uporabo, a na žalost je skoraj neuporabna, ker je dosegljiva le z računalnika, kjer je zagnana, in vsebuje le enega uporabnika (privzeti **postgres** uporabnik je vedno na voljo).

Najprej bomo dodali uporabnika in ustvarili bazo zanj, kar naredimo z naslednjimi ukazi:

```
postgres=# CREATE USER demo WITH PASSWORD 'myPassword';
postgres=# CREATE DATABASE demo;
postgres=# GRANT ALL PRIVILEGES ON DATABASE demo TO demo;
postgres=# \q
```

S tem smo razrešili problem uporabnika, ki pa se še vedno ne more povezati z oddaljene lokacije (npr. iz računalniške učilnice na FMF). V ta namen pa moramo spremeniti konfuguracijo PostgreSQL-a.

Najprej bomo popravili konfiguracijsko datoteko `/var/lib/pgsql/10/data/postgresql.conf` in vrstico

```
listen_addresses = 'localhost'
```

zamenjali z

```
listen_addresses = '*'
```

S tem smo dovolili PostgreSQL-u, da sprejema povezave od vsepovsod. V kolikor bi poznali naslove klientov, s katerih se bodo vzpostavljale povezave, bi jih lahko neposredno našteli v nastavitve, s čimer bi malce povečali varnost namestitve.

Sedaj bomo aktivirali še *Secure Sockets Layer* (SSL[5]), s čimer bomo poskrbeli za varnost podatkov med prenosom s strežnika do klienta. V ta namen bomo poiskali vrstico

```
#ssl = off
```

in jo spremenili v

```
ssl = on
```

Da se bo lahko uporabnik ne le povezal s PostgreSQL-om, ampak tudi uspešno prijavil, pa moramo popraviti še datoteko `/var/lib/pgsql/10/data/pg_hba.conf`, ki vsebuje navodila oz. pravila za avtentikacijo uporabnikov (*hba* v imenu datoteke je kratica za *Host Based Authentication*).

Na konec datoteke bomo dodali vrstico:

```
hostssl all             all             0.0.0.0/0               md5
```

S tem smo dovolili kriptirane povezave iz oddaljenih lokacij, uporabniki pa se morajo avtenticirati s svojim geslom.

Zadnja stvar, ki jo moramo še narediti, pa so strežniški certifikati, ki jih bo PostgreSQL uporabljal za kriptiranje podatkov pred transportom. Za demonstracijsko okolje bomo pripravili samopodpisan certifikat, v produkcijskem okolju pa bi nam certifikat izdal eden izmed avtoritativnih virov certifikatov (za kar bi nam seveda zaračunal konkreten znesek denarja).

Za začetek bomo ustvarili konfiguracijsko datoteko za OpenSSL, ki opisuje naš certifikat (podatke lahko seveda prilagodimo):

```
[req]
default_bits = 2048
prompt = no
distinguished_name = req_distinguished_name
x509_extensions = v3_req
string_mask = utf8only

[req_distinguished_name]
countryName = SI
stateOrProvinceName = Slovenia
localityName = Ljubljana
0.organizationName = XLAB d.o.o.
organizationalUnitName = Research
commonName = XLAB
emailAddress = tadej.borovsak@xlab.si

[v3_req]
subjectAltName = @alt_names
subjectKeyIdentifier = hash
authorityKeyIdentifier = keyid:always,issuer
basicConstraints = CA:true

[alt_names]
IP.1 = 91.217.255.38
```

[5] https://www.digicert.com/ssl/

Zgornjo vsebino bomo shranili v datoteko `/tmp/ssl.cnf`, nato pa pognali ukaze

```
$ sudo openssl req -new -nodes  -x509 -newkey rsa:2048 \
    -sha256 -config /tmp/ssl.cnf -days 730 \
    -out /var/lib/pgsql/10/data/server.crt \
    -keyout /var/lib/pgsql/10/data/server.key
$ sudo chown postgres:postgres /var/lib/pgsql/10/data/server.*
$ sudo chmod 600 /var/lib/pgsql/10/data/server.*
```

Da zaključimo konfiguracijo, moramo le še znova zagnati PostgreSQL:

```
$ sudo systemctl restart postgresql-10
```

Z zadnjim ukazom smo tudi dosegli cilj, ki smo si ga zadali za prvi del delavnice: pripravo varne namestitve PostgreSQL sistema za upravljanje baz.

Analiza in optimizacija poizvedb

V nadaljevanju bomo našo pozornost preusmerili z administrativnih del (poznavanje katerih pa je nujni in zelo pomemben del programerskega zaledja) na bolj programersko usmerjene dejavnosti. Začeli pa bomo s ponotranjenjem naslednje mantre:

Sistem za upravljanje s podatkovnimi bazami o podatkih ve več kot jaz.

Zato bomo vedno, ko se bomo lotili optimizacije poizvedb, najprej s pomočjo PostgreSQL-a analizirali obstoječe stanje in poiskali vzroke za težave, šele nato bomo glede na vzrok primerno ukrepali.

Natančen izbor poizvedb, ki jih bomo analizirali, bomo prilagajali sproti glede na odzive udeležencev oz. glede na vprašanja, ki se bodo porajala ob izvajanju poizvedb. Tu bomo navedli le nekaj primerov analize poizvedb, ob katerih bomo opisali podatke v izhodu programa.

Primeri analize poizvedb

Za začetek si oglejmo analizo bolj kompleksne poizvedbe, kjer se bomo tudi seznanili z osnovnimi gradniki, ki nam jih vrne analizator.

```
regression=# EXPLAIN SELECT *
regression-# FROM tenk1 t1, tenk2 t2
regression-# WHERE t1.unique1 < 10 AND t2.unique2 < 10 AND t1.hundred < t2.hundred
    ↪ ;
                               QUERY PLAN
------------------------------------------------------------------------
 Nested Loop  (cost=0.00..177.75 rows=3333 width=488)
   Join Filter: (t1.hundred < t2.hundred)
   ->  Seq Scan on tenk1 t1  (cost=0.00..13.75 rows=100 width=244)
         Filter: (unique1 < 10)
   ->  Materialize  (cost=0.00..14.25 rows=100 width=244)
         ->  Seq Scan on tenk2 t2  (cost=0.00..13.75 rows=100 width=244)
               Filter: (unique2 < 10)
```

Zgornji izhod predstavlja drevo operacij, ki se izvedejo. Če si ga poenostavimo in zapišemo samo operacije, dobimo:

```
Nested Loop
  ->  Seq Scan on tenk1 t1
  ->  Materialize
        ->  Seq Scan on tenk2 t2
```

Grafično si lahko to drevo prikažemo tudi z uporabo specializiranih orodij (slika X), kot rečeno pa si lahko to drevo skonstruiramo tudi iz tekstovnega prikaza.

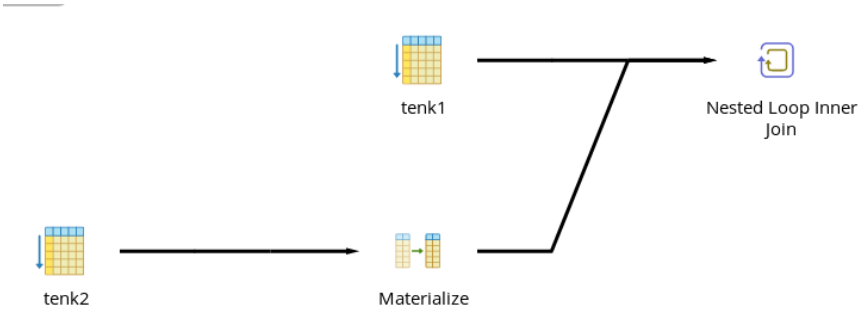

Figure 14: Grafični prikaz plana poizvedbe.

Pri interpretaciji zgornjega zaporedja operacij se tako najprej izvede zaporedno branje iz tabele *tenk1* z izločanjem vozlišč, ki ne ustrezajo pogoju *unique1 < 10*. Za vsak vrnjeni element se izvede zaporedno branje iz tabele *tenk2* z izločanjem elementov, ki ne ustrezajo pogoju *unique2 < 10*. Na koncu pa se izvede spajanje vrstic iz obeh vgnezdenih poizvedb, zopet upoštevajoč pogoj.

Iz tega osnovnega primera kartezičnega produkta z nekaj omejitvami lahko razberemo, da je implementacija same poizvedbe precej zahtevna, obenem pa zna sistem z vzorčenjem podatkov v tabelah sam opraviti marsikatero optimizacijo. V tem specifičnem primeru vidimo, da je drugo zaporedno branje optimizator ovil v materializiran pogled, kar pomeni, da le prva iteracija zanke dejansko bere podatke z diska, vse ostale pa beremo shranjeno kopijo v delovnem pomnilniku.

Če prejšnjo poizvedbo le malenkost spremenimo, pa dobimo povsem drugačen plan.

```
regression=# EXPLAIN SELECT *
regression-# FROM tenk1 t1, tenk2 t2
regression-# WHERE t1.unique1 < 100 AND t1.unique2 = t2.unique2;
                              QUERY PLAN
------------------------------------------------------------------------
 Hash Join  (cost=15.00..34.00 rows=150 width=488)
   Hash Cond: (t2.unique2 = t1.unique2)
   ->  Seq Scan on tenk2 t2  (cost=0.00..13.00 rows=300 width=244)
   ->  Hash  (cost=13.75..13.75 rows=100 width=244)
         ->  Seq Scan on tenk1 t1  (cost=0.00..13.75 rows=100 width=244)
               Filter: (unique1 < 100)
```

V tem primeru se je optimizator ocenil, da je optimalni način izpolnitve poizvedbe naslednji:

1. Najprej vse vrstice iz tabele *tenk1*, ki ustrezajo pogoju *unique1 < 100* postavi v slovar.
2. Nato izvede zaporedno branje elementov iz tabele *tenk2* in za vsak element preveri, če obstaja ustrezni element v prej zgrajenem slovarju.

Za konec pa si oglejmo še, kako lahko pregledamo alternativne plane poizvedb. V naslednjem primeru smo analizatorju prepovedali uporabo slovarja, kar je povzročilo naslednjo spremembo:

```
regression=# set enable_hashjoin = off;
SET
regression=# EXPLAIN SELECT *
FROM tenk1 t1, tenk2 t2
WHERE t1.unique1 < 100 AND t1.unique2 = t2.unique2;
                              QUERY PLAN
------------------------------------------------------------------------
 Merge Join  (cost=42.42..45.92 rows=150 width=488)
   Merge Cond: (t1.unique2 = t2.unique2)
   ->  Sort  (cost=17.07..17.32 rows=100 width=244)
```

```
        Sort Key: t1.unique2
        ->  Seq Scan on tenk1 t1  (cost=0.00..13.75 rows=100 width=244)
              Filter: (unique1 < 100)
->  Sort  (cost=25.34..26.09 rows=300 width=244)
      Sort Key: t2.unique2
      ->  Seq Scan on tenk2 t2  (cost=0.00..13.00 rows=300 width=244)
```

Ker smo prepovedali uporabo slovarja, je analizator sestavil plan, kjer najprej elemente iz tabel uredi po vrednosti iz stolpcev, ki jih uporabljamo za spajanje podatkov, nakar izvede zlivanje podatkov iz obeh urejenih seznamov.