Napredne funkcionalnosti na bazah

Napredne funkcionalnosti na bazah

- Namestitev Linuxa, PostgreSQL
- Uporabniki, vloge ter dostop
- Indeksi
- Transakcije
- Določanje dostopa
- Shranjene funkcije
- Prožilci (*triggers*)
- Pogledi (*views*)
- Ogledali si bomo na RDBMS PostgreSQL

Namestitev Linuxa

- Virtualizacija: zaganjanje drugega operacijskega sistema (celotnega računalnika) kot programa.
- VirtualBox - program za virtualizacijo (Oracle).
- Iz spleta naložimo program VirtualBox.
- Iz spleta naložimo sliko `.iso` za namestitev operacijskega sistema Linux Ubuntu Server.
- Po navodilih nastavimo virtualni računalnik in namestimo Linux.
- POZOR: za uporabnika nastavimo uporabniško ime `ubuntu` in dogovorjeno geslo.

Osnovno delo v terminalu

- Namestili smo *Ubuntu Server*, ki omogoča samo terminalski dostop.
- Lahko bi si namestili *Ubuntu Desktop*, ki ponuja tudi uporabo preko ukazne lupine.
- V terminal se po zagonu prijavimo kot uporabnik `ubuntu`.
- Uporabnik `ubuntu` je član skupine `sudoers` in lahko preko ukaza `sudo` izvaja operacije s pravicami administratorja.
- Spoznamo se z nekaj osnovnimi ukazi v ukazni lupini `bash`.

Pomembni ukazi v bash

- `pwd` - v kateri mapi se nahajamo
- `ls` - izpis vsebine mape (`ls -l`, `ls -lt`)
- `cd` - spremeni mapo (`cd podmapa`, `cd ..`, `cd /home/ubuntu`, `cd`, ...)
- `cp` - kopiranje (`cp datoteka1 datoteka2`, `cp -r pot/do/mape pot/do/cilja`)
- `mv` - premikanje oz. preimenovanje (`mv datoteka1 datoteka2`, `mv mapa1 pot/do/mape2`)
- `rm` - briši (`rm datoteka`, `rm -r mapa`)
- `nano` - preprost tekstovni urejevalnik (`nano pot/do/datoteke.txt`)
- `sudo` - izvajanje ukazov z administratorskimi pravicami

Namestitev PostgreSQL

- Nameščanje na Ubuntu izvajamo preko paketnega sistema in ukaza `apt`.

```
sudo apt update
sudo apt install postgresql postgresql-contrib
```

- Ob namestitvi se na sistemu ustvari uporabnik `postgres`, ki je tudi administrator na bazi.
- Vstop v vlogo (uporabnika) `postgres`:

```
sudo -i -u postgres
```

Program `pqsl`

- Program za priklop na bazo - `psql`. Priklop na bazo kot uporabnik `postgres`.

```
psql
```

- Poln ukaz za priklop (glej tudi `man psql`).

```
psql -h baza.fmf.uni-lj.si -U student banka
```

- `\q` - izhod
- `\h` - pomoč glede ukazov v SQL
- `\conninfo` - parametri priklopa na bazo
- `\du` - pregled vlog na bazi
- `\d` - pregled tabel
- `\?` - pomoč z izpisom vsah ukazov v `psql`

Uporabniki na bazi

- Na bazo se priklopimo z zadostnimi pravicami.
- Npr. na Linuxu pod uporabniškim imenom `postgres` zaženemo klienta `psql`.
- Ustvarjanje uporabnika

```
CREATE USER uporabnisko_ime
  WITH ENCRYPTED PASSWORD '********';
```

- PostgreSQL pozna uporabnike in skupine, ampak jih enovito uporablja preko posplošitve (vloge) - `ROLE`.
- Uporabnika definiramo lahko tudi takole:

```
CREATE ROLE ime_vloge WITH LOGIN;
```

- Vlogam dodeljujemo pravice, vlogo lahko obravnavamo kot uporabnika (če ima pravico `WITH LOGIN`) ali kot skupino.

Vloge

- Ustvarjanje vloge:

```
CREATE ROLE ime_vloge;
```

- Brisanje vloge:

```
DROP ROLE ime_vloge;
```

- Brisanje neobstoječe vloge vrne napako. Lahko uporabnimo tole:

```
DROP ROLE IF EXISTS ime_vloge;
```

Določanje pravic na vloge

- Ustvarjanje vloge z dodelitvijo pravice prijave:

```
CREATE ROLE demo_role WITH LOGIN;
```

- Popravek pravice prijave:

```
ALTER ROLE ime_vloge WITH NOLOGIN;
```

Ustvarjanje podatkovnih baz

- Na bazo moramo biti priklopljeni kot uporabnik z dovolj pravicami (npr. administrator `postgres`)

```
CREATE DATABASE ime_baze;
```

- Ustvarimo novo bazo, se priklopimo nanjo in preverimo priklop.

```
CREATE DATABASE nova;
\c nova
\conninfo
```

- Ustvarimo novo tabelo in nekaj vnosov.

```
CREATE TABLE oseba (emso TEXT PRIMARY KEY,
                    ime TEXT, priimek TEXT);
INSERT INTO oseba (emso, ime, priimek)
  VALUES ('1234', 'Janez', 'Novak');
```

Dodelitev pravic na vloge

- Splošen ukaz za dodelitev pravice.

```
GRANT tip_pravice ON tabela TO ime_vloge;
```

- Dodelitev pravice popravljanja na tabeli uporabniku.

```
GRANT UPDATE ON tabela TO uporabnik;
```

- Dodelitev vseh pravic na tabeli uporabniku.

```
GRANT ALL ON tabela TO uporabnik;
```

Odvzemanje pravic vlogam

- Onemogočanje pravic.

```
REVOKE tip_pravice ON tabela FROM ime_vloge;
```

- Primer: onemogočanje vstavljanja vsem uporabnikom (ki nimajo ustrezne pravice dodeljene na drug način).

```
REVOKE INSERT ON tabela FROM PUBLIC;
```

Sheme

- Uporabniki in vloge so vidne preko celotnega Postgres RDBMS.
- S `psql` ali kakim drugim klientom (npr. pgAdmin 4) se lahko priklopimo na točno določeno bazo.
- Sheme so imenski prostori znotraj ene podatkovne baze, v katerih živijo tabele in drugi logični objekti (npr. števci, shranjene funkcije).
- Uporabljamo jih za lažje določanje pravic na tabelah, boljšo logično organizacijo objektov in razreševanje imenskih konfliktov.
- Ko ustvarimo neko podatkovno bazo, se ustvari tudi shema `public`, v kateri so nove tabele.
- Na tabele na podatkovni bazi se sklicujemo v obliki `ime_sheme.ime_tabele`.

Primer

- Ustvarimo uporabnika `janez`.

```
CREATE USER janez WITH ENCRYPTED PASSWORD 'pomlad';
```

- Poglejmo obstoječe uporabnike.

```
\du
```

- Odjavimo se in poskusimo priklopiti na bazo.

```
psql -h localhost -U janez nova
```

- Poizvedba ni možna:

```
SELECT * FROM oseba;
```

Primer

- Odjavimo se in prijavimo kot uporabnik `postgres`.

```
psql nova
```

- Izvedemo naslednje:

```
REVOKE ALL ON DATABASE nova FROM PUBLIC;
GRANT CONNECT ON DATABASE nova TO janez;
GRANT USAGE ON SCHEMA public TO janez;
GRANT ALL ON ALL TABLES IN SCHEMA public TO janez;
GRANT ALL ON ALL SEQUENCES IN SCHEMA public TO janez;
```

- Preverimo pravice na bazi, shemi in tabelah na bazi.

```
\l
\dn+
\dp
```

Vloge kot skupine

- Ustvarjanje skupine in dodajanje uporabnikov.

```
CREATE ROLE skupina;
GRANT skupina TO uporabnik1;
GRANT skupina TO uporabnik2;
```

- V resnici dajamo lahko “vloge na vloge” - tako kot bi dajali skupine v skupine, uporabnike pa si predstavljamo kot elementarne skupine s pravico priklopa.
- Uporabnik v neki skupini se začne obnašati skladno z vlogo, če to vklopimo:

```
SET ROLE skupina;
```

Skupine

- V osnovno vlogo se vrnemo:

```
RESET ROLE;
```

- Lahko pa uporabniku posebej nastavimo, da avtomatično podeduje pravice iz skupin/vlog (privzeto).

```
ALTER ROLE uporabnik INHERIT;
```

- To pomeni: če uporabniku nastavimo lastnost `INHERIT`, mu ni treba vsakič "skakati" v vloge s pomočjo `SET ROLE` ampak se pravice iz vseh vlog, katerih član je, avtomatično upoštevajo.

Primer - dovoljevanje dostopa preko skupine

- Odjavimo se in se prijavimo kot uporabnik `postgres`.

```
psql nova
```

- Izvedemo naslednje:

```
CREATE ROLE novaadmini;
REVOKE ALL ON DATABASE nova FROM PUBLIC;
GRANT CONNECT ON DATABASE nova TO novaadmini;
GRANT USAGE ON SCHEMA public TO novaadmini;
GRANT ALL ON ALL TABLES IN SCHEMA public
  TO novaadmini;
GRANT ALL ON ALL SEQUENCES IN SCHEMA public
  TO novaadmini;
\l
\dn+
\dp
```

Primer - dovoljevanje dostopa preko skupine

- Uporabniku dodelimo pravico.

```
CREATE USER metka WITH ENCRYPTED PASSWORD 'pomlad';
ALTER ROLE metka NOINHERIT;
GRANT novaadmini TO metka;
\du
```

- Če želimo ohranjanje privzetih pravic na novih objektih:

```
ALTER DEFAULT PRIVILEGES
  FOR ROLE metka IN SCHEMA public
  GRANT ALL ON TABLES TO novaadmini;
ALTER DEFAULT PRIVILEGES
  FOR ROLE metka IN SCHEMA public
  GRANT ALL ON SEQUENCES TO novaadmini;
```

Lažji oddaljen priklop na bazo

- Tipično imamo podatkovno bazo na nekem strežniku.
- Lahko se sicer priklopimo nanjo preko programa `psql` (ukazna vrstica).
- Tipkanje v terminalu `psql` je velikokrat nepriročno.
- Najbolj priročno je do baze dostopati preko ustreznih aplikacij z grafičnim vmesnikom.
- Primer take aplikacije: phpPgAdmin (https://baza.fmf.uni-lj.si), ki se uporablja na vajah.

Konfiguracija strežnika za oddaljen dostop

- Najprej popravimo datoteko `postgresql.conf`.
- Lokacijo najdemo z ukazom

```
psql -U postgres -c 'SHOW config_file'
```

- S pomočjo urejevalnika `nano` popravimo ustrezno vrstico na:

```
listen_addresses = '*'
```

- Urejevalnik zaženemo npr. kot

```
nano /etc/postgresql/14/main/postgresql.conf
```

Oddaljen dostop - konfiguracija

- Lokacijo konfiguracijske datoteke `pg_hba.conf` najdemo z ukazom

```
psql -U postgres -c 'SHOW hba_file'
```

- S pomočjo urejevalnika `nano` popravimo vrstico oblike

```
host    all     all     127.0.0.1/32    scram-sha-256
```

v vrstico

```
host    all     all     0.0.0.0/0       scram-sha-256
```

- Urejevalnik zaženemo npr. kot

```
nano /etc/postgresql/14/main/pg_hba.conf
```

Oddaljen dostop - konfiguracija

- Da uveljavimo nove nastavitve, ponovno zaženemo strežnik (kot uporabnik `ubuntu`)

```
service postgresql restart
```

- Za ustavljanje in zagon RDBMS PostgreSQL lahko uporabimo podoben ukaz (namesto parametra `restart` lahko uporabimo parametra `stop` oz. `start`).

PgAdmin 4

- Namizna alternativa: program `PgAdmin 4` (namestimo iz: https://www.pgadmin.org/download/).
- V programu konfiguriramo povezavo podobno kot v `psql`.
- Namesto ukazov v `psql` oblike `\ukaz` imamo grafične preglede in ukaze preko menijev.
- Poizvedbe izvajamo preko orodja *Query tool*.

Priklop preko PgAdmin 4

- Virtualni strežnik mora imeti mrežni vmesnik nastavljen na `Bridged Adapter`, t.j., na isti mreži, kot je naš računalnik.
- Za priklop rabimo naslov strežnika (`host`), uporabnika (`user`), njegovo geslo in ime podatkovne baze, na katero se priklapljamo.
- IP številko virtualnega računalnika dobimo z ukazom v terminalu:

```
ip address
```

- Preden omogočimo uporabniku `postgres` (t.j., administratorju) oddaljen priklop, mu moramo nastaviti geslo (dvakrat vnesemo novo geslo):

```
psql
\password
```

- V nadaljevanju bomo komunicirali s podatkovno bazo preko PgAdmin 4.

Indeksi

- Iskanje po splošnem stolpcu v podatkovni bazi zahteva $O(n)$ časa.
- Če je podatkovni tip stolpca linearno urejena množica, lahko nad vrsticami stolpca izgradimo iskalno strukturo, ki omogoča iskanje v $O(1)$ ali $O(\log n)$.
- Strukturo imenujemo *indeks*.

Indeksi - primer

- Ustvarimo tabelo.

```
CREATE TABLE tabela (
    id INTEGER,
    vsebina TEXT
);
```

- Na stolpcu `id` ustvarimo indeks.

```
CREATE INDEX index1 ON tabela (id);
```

- Poglejmo, kakšne indekse imamo.

```
\di
```

Indeksi

- Vzdrževanje indeksov zahteva določeno dodatno delo (ob vsakem popravku baze).
- Odstranjevanje indeksa:

```
DROP INDEX index1;
```

- Tipi indeksov:
  - B-tree: linearno urejene množice
  - Hash: zgoščena tabela. Možno samo iskanje po enakosti.
  - GiST, SPGiST, GIN, ...
- Primer:

```
CREATE INDEX name ON table USING hash (column);
```

Indeksi

- Indekse se lahko naredi za več stolpcev.

```
CREATE INDEX index_name
  ON table_name (column1_name, column2_name);
```

- Primarni in sekundarni ključi so že indeksirani.
- Indeksi na izrazih nad stolpci:

```
CREATE INDEX users_lower_email ON users(lower(email));
```

- Tak indeks omogoča hitro iskanje po pogojih oblike `WHERE lower(email) = ??`

Transakcije

- Lastnosti transakcij - ACID:
  - Atomarnost: vse operacije v transakciji izvedene ali zavrnjene.
  - Konsistentnost: pred in po transakciji je stanje v bazi konsistentno.
  - Izolacija: transakcije so izolirane med seboj in medsebojni vpogled je pregleden.
  - Trajnostnost: rezultat transakcije je trajno shranjen.
- Ukazi:
  - `BEGIN` ali `BEGIN TRANSACTION` - začetek transakcije
  - `COMMIT` ali `END TRANSACTION` - konec transakcije, shrani spremembe
  - `ROLLBACK` - razveljavi spremembe

Primer

```
\c nova
CREATE TABLE blagajna
  (ime TEXT, znesek NUMERIC(10,2));
INSERT INTO blagajna (ime, znesek)
  VALUES ('Janez', 10), ('Metka', 10);
BEGIN;
  UPDATE blagajna SET znesek = znesek - 5
    WHERE ime = 'Janez';
  UPDATE blagajna SET znesek = znesek + 5
    WHERE ime = 'Metka';
COMMIT;
```

Nivo izolacije

- Problemi:
  - *umazano branje* - preberemo nepotrjene podatke.
  - *neponovljivo branje* - v transakciji večkrat beremo iste podatke, a se je nek podatek vmes spremenil.
  - *fantomsko branje* - v transakciji večkrat naredimo isto poizvedbo in zaradi spremembe podatkov dobimo več ali manj vrstic.
- Nivoji izolacije:

Table 13-1. Standard SQL Transaction Isolation Levels

Isolation Level	Dirty Read	Nonrepeatable Read	Phantom Read
Read uncommitted	Possible	Possible	Possible
Read committed	Not possible	Possible	Possible
Repeatable read	Not possible	Not possible	Possible
Serializable	Not possible	Not possible	Not possible

Nivo izolacije

- Privzeti nivo je `READ COMMITED`.
- Lahko ga nastavimo:

```
SET TRANSACTION SERIALIZABLE | REPEATABLE READ |
        READ COMMITTED | READ UNCOMMITTED
```

Shranjene funkcije

- Na RDBMS lahko napišemo posebne funkcije v različnih programskih jezikih.
- Prednosti:
  - Zmanjšanje komunikacije z podatkovno bazo
  - Izboljšanje učinkovitosti
  - Souporaba v več bazah
- Slabosti:
  - Počasnejši in težji razvoj programske opreme
  - Težko razhroščevanje, vodenje verzij
  - Težja prenosljivost med različnimi RDBMS
- Razen v posebnih primerih se v veliki večini primerov danes izogibamo uporabe shranjenih funkcij.

Definicija funkcij

- Sintaksa:

```
CREATE FUNCTION function_name(p1 type, p2 type)
 RETURNS INTEGER AS
BEGIN
 -- koda ...
END;
LANGUAGE language_name;
```

Definicija funkcij - primer

```
CREATE FUNCTION povecaj(n INTEGER) RETURNS INTEGER AS
$$
BEGIN
  RETURN n + 1;
END;
$$
LANGUAGE plpgsql;
```

- Uporaba

```
SELECT povecaj(20)
```

Prožilci

- angl. *trigger* - posebna funkcija, ki je povezana s tabelo.
- Sproži se ob dogodkih, povezanih z operacijami: `INSERT`, `UPDATE`, `DELETE`.
- Lahko se proži za vsak stavek posebej ali za vsako vrstico posebej.
- Lahko se proži pred dogodkom ali po njem.

Prožilci - primer

- Ustvarimo tabelo zaposlenih:

```
CREATE TABLE zaposleni (
    id SERIAL PRIMARY KEY,
    ime TEXT NOT NULL,
    priimek TEXT NOT NULL
);
```

Prožilci - primer

- Ustvarimo tabelo sprememb:

```
CREATE TABLE zaposleni_spremembe (
    id SERIAL PRIMARY KEY,
    zaposleni_id INTEGER NOT NULL,
    priimek TEXT NOT NULL,
    spremenjeno TIMESTAMP(6) NOT NULL
);
```

Prožilci - primer

- Prožilna funkcija:

```
CREATE OR REPLACE FUNCTION belezi_spremembe()
  RETURNS trigger AS
$BODY$
BEGIN
  IF NEW.priimek <> OLD.priimek THEN
     INSERT INTO zaposleni_spremembe
       (zaposleni_id, priimek, spremenjeno)
           VALUES (OLD.id, OLD.priimek, now());
    END IF;
    RETURN NEW;
END;
$BODY$ LANGUAGE plpgsql;
```

Prožilci - primer

- Povezava funkcije s prožilcem:

```
CREATE TRIGGER zadnje_spremembe
  BEFORE UPDATE
  ON zaposleni
  FOR EACH ROW
  EXECUTE PROCEDURE belezi_spremembe();
```

```
INSERT INTO zaposleni (ime, priimek)
  VALUES ('Janez', 'Novak');
INSERT INTO zaposleni (ime, priimek)
  VALUES ('Metka', 'Lepše');
SELECT * FROM zaposleni;
UPDATE zaposleni SET priimek = 'Zelenko' WHERE id = 2;
SELECT * FROM zaposleni;
SELECT * FROM zaposleni_spremembe;
```

Pogledi (VIEW)

- Virtualne tabele kot rezultat poizvedbe, s katerimi lahko delamo tako kot s pravimi tabelami (poizvedbe, pravice).
- Pogled lahko predstavlja tabelo, ki določenim uporabnikom bolje predstavi podatke, ki jih pogosto povzamemo iz več tabel (npr. seštevki transakcij po računih).
- Uporabnikom lahko dodelimo pravice samo na (izbrane) poglede.
- Pogledi so definirani kot (shranjene) poizvedbe.

Pogledi (VIEW)

- Ustvarjanje pogleda:

```
CREATE VIEW ime_pogleda AS SELECT ...;
```

- Brisanje pogleda:

```
DROP VIEW ime_pogleda;
```

Pogledi - primer

```
CREATE VIEW samo_priimki_zaposlenih AS
  SELECT id, priimek FROM zaposleni;
SELECT * FROM samo_priimki_zaposlenih;
```