SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Anja Horvatinović

SUSTAV ZA UPRAVLJANJE BAZAMA PODATAKA POSTGRESQL 12

ZAVRŠNI RAD

Varaždin, 2020.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Anja Horvatinović

Matični broj: 44925/16-R

Studij: Poslovni sustavi

SUSTAV ZA UPRAVLJANJE BAZAMA PODATAKA POSTGRESQL 12

ZAVRŠNI RAD

Mentor:

Prof. dr. sc. Kornelije Rabuzin

Varaždin, lipanj 2020.

Anja Horvatinović

Izjava o izvornosti

Izjavljujem da je moj završni/diplomski rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor/Autorica potvrdio/potvrdila prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

Radom će biti prikazan i opisan sustav za upravljanje bazama podataka PostgreSQL 12. Na početku je bitno teorijski opisati što su to relacijske baze podataka kao i kako izraditi ERA model i relacijsku shemu. Bit će opisane karakteristike i novosti koje donosi nova verzija sustava 12.1.

Također, kako bi se bolje prikazale funkcionalnosti samog sustava na primjeru baze podataka za imaginarno skladište, u PostgreSQL-u će biti implementirana baza podataka. Svaki korak izrade i naredbe koje su korištene bit će opisane kroz primjer i njihovu glavnu sintaksu.

Na kraju će se uz pomoć alata pgAdmin prikazati kako baza izgleda i da je kroz sučelje moguće raditi razne promjene nad bazom na jednostavniji način bez korištenja SQL Shell-a (psql).

Ključne riječi: baza podataka; sustav za upravljanje bazama podataka; PostgreSQL 12; relacijska baza podataka; pgAdmin

Sadržaj

Uvod

U današnjem svijetu svakim danom broj podataka raste. Podaci su zapravo činjenice koje se bilježe. Bitno je raspoznati koji su podaci korisni i bitni za svaku osobu ili sustav te oni koji nemaju neku svrhu. Pojavom velike količine podataka dolazi do pojave da se ti podaci na pravilan način zapisuju i koriste. Također, nije bitno samo imati zapisane podatke, vrlo je važno s njima pravilno manipulirati kako bi oni bili od važnosti za one koji ih koriste.

Upravo iz tih razloga neophodno je poznavanje sustava za upravljanje bazama podataka (SUBP) za sve one koji se tim podacima bave. Sustavi na brz i efikasan način rade s podacima ono što želimo, naravno ukoliko se SUBP koristi na pravilan način. Postoje razni sustavi, oni koji se temelje na SQL jeziku i tzv. NoSQL koji podatke ne prikazuju u tablicama i ne koriste SQL jezik.

Ovaj rad govori o sustavu za upravljanje bazama podataka PostgreSQL 12, četvrtim najkorištenijim sustavom za brojnim naprednim značajkama. Verzija je izašla krajem 2019. godine i donosi neka poboljšanja u samim performansama. Osim teorijskog dijela o samom sustavu, bit će dan primjer izmišljene baze podataka te naredbe koje su korištene pri samoj implementaciji.

.

Metode i tehnike rada

Rad se temelji na izradi izmišljene baze podataka u psql konzoli instaliranoj na OS Windows 10. Također, bitno je prvo opisati teorijski dio i sintaksu naredbi kako bi se pravilno implementirala baza. Osim korištenja psql konzole, ukratko je pokazano kako izgleda baza u grafičkom alatu pgAdmin 4. Sve naredbe su opisane pomoću korištene literature, a primjer baze i podaci unutar tablica su izmišljeni.

1. Osnovni pojmovi

Da bi se prikazao i opisao SUBP PostgreSQL te kasnije implementirala baza, potrebno je definirati osnovne pojmove koji su bitni za lakše povezivanje i shvaćanje cjelokupne teme. Dakle, bitno je shvatiti pojam baza podataka, relacijskih baza podataka i osnovnih koncepata vezanih uz njih.

1.1. Baza podataka

„Baza podataka je kolekcija podataka, ograničenja i operacija koja reprezentira neke aspekte realnog svijeta.“ (Maleković, Rabuzin, 2016.)

To je zbirka podatkovnih zapisa koji su pohranjeni na računalu i koji su dostupni korisnicima kao i drugim aplikacijama. Podaci se pohranjuju zajedno bez suvišnog ponavljanja. No, bitno je naglasiti kako nije svaka zbirka podatkovnih zapisa baza podataka. Da bi ona to bila mora imati određena svojstva, a neka od njih su sljedeća (Kaštelan, 2010.):

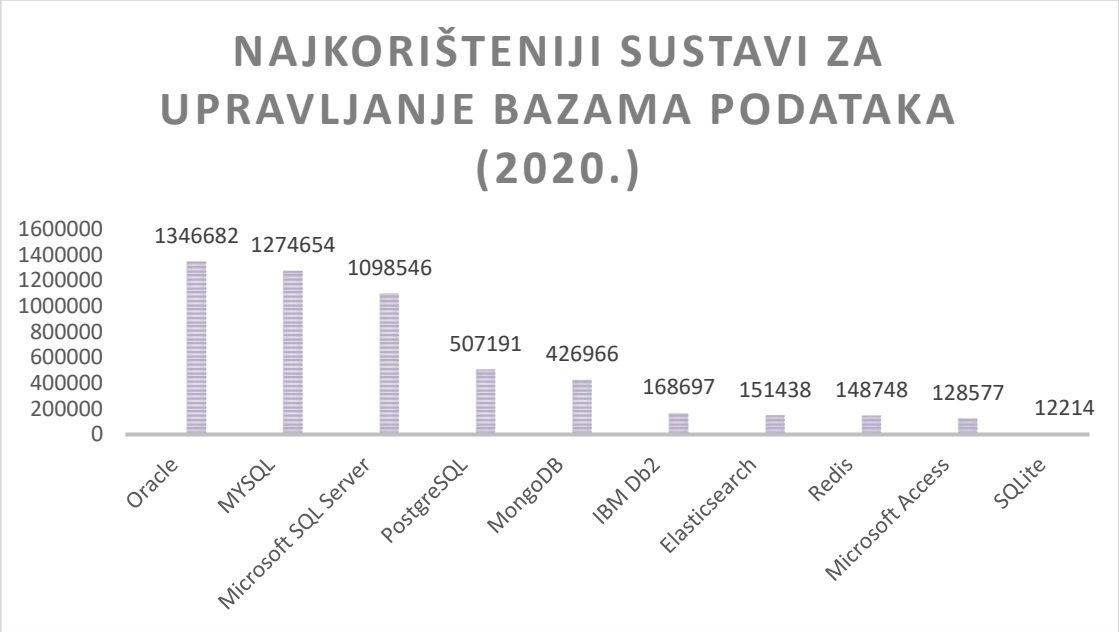
- Integritet domene
- Referencijalni integritet
- Jasna podatkovna shema
- Standardni upitni jezik
- Omogućen zaštićeni zajednički pristup podacima skupini korisnika

Baza sadrži strukturirani opis vrste činjenica koji se nalazi u njoj, a taj opis se zove shema. Ona opisuje predmete iz baze podataka kao i njihove međusobne odnose. Modeli baza podataka predstavljaju način organiziranja sheme, odnosno modeliranja strukture. Postoji mnogo modela, ali najčešće korišteni je relacijski model koji sadrži informacije zapisane u više tablica koje se sastoje od redaka i stupaca. Relacijski model prikazuje odnose uporabom vrijednosti koje su zajedničke za više tablica, a hijerarhijski i mrežni modeli koriste prikaze i odnose koji su eksplicitniji. (Kaštelan, 2010.)

1.2. Sustav za upravljanje bazom podataka (SUBP)

Sustav za upravljanje bazama podataka (*engl. DBMS – Database Management System*) je programska podrška koja izvodi operacije nad bazom podataka. Pod operacijama se misli na kreiranje strukture, brisanje i mijenjanje podataka, dohvaćanje podataka, administracija i drugo. (Kaštelan, 2010.)

Oni se dijele prema modelu podataka kojima upravljaju i podržavaju pa tako mogu biti relacijski, objektno/relacijski, mrežni i dr. Većina današnjih sustava koji su u upotrebi su zasnovani na relacijskom modelu kojeg je uveo E.F.Codd, a osim relacijskih postoje i sustavi koji su zasnovani na objektnom modelu.



Slika 1: Najkorišteniji SUBP (siječanj, 2020.), izrada autora prema: https://db-engines.com/en/ranking_trend

Iako PostgreSQL zauzima četvrto mjesto na listi najkorištenijih, on nije nimalo lošiji. Štoviše ako usporedimo mogućnosti koje on nudi dolazi se do zaključka da je u nekim funkcionalnostima napredniji. Ako se napravi usporedba četiri najkorištenija, MySQL i PostgreSQL su slobodni softveri otvorenog koda, no MYSQL nema većinu osobina koje posjeduje PostgreSQL. Osobine MySQL – a su očuvanje referencijalnog integriteta, otvoreno sučelje za aplikacije, podrška za SSL i indeksi. Oracle koji zauzima prvo mjesto nije otvorenog koda no posjede skoro sve osobine kao i PostgreSQL kao što su: pravila, pogledi, ACID uvjeti,

okidači, nasljeđivanje, proceduralni jezici, ANSI SQL kompatibilnost, replikacija, podrška za Unicode, vanjsko spajanje i mnoge druge. (Medak, 2005.)

1.3. Relacijske baze podataka

Relacijska baza podataka je baza za čiji skup podataka vrijedi da su u njoj podaci vezani relacijama i strukturirani tako da se osigura:

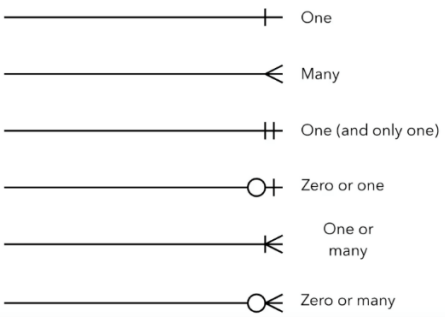
- ažurnost pohranjenih podataka
- sigurnost i nadzor pristupa podacima
- mogućnost dnevnog sigurnosnog arhiviranja
- najmanja zalihost (redundancija)
- postojanost pohranjenih podataka
- trajno očuvanje integriteta pohranjenih podataka

(Relacijska baza podataka wiki, Izvor: https://hr.wikipedia.org/wiki/Relacijska_baza_podataka)

Relacijska struktura se velikim dijelom podudara sa događajima u stvarnom svijetu i zbog te jednostavnosti, podaci su jasni ljudima za shvaćanje. Upravo zbog toga je relacijski model u prednosti nad ostalima i relacijska baza podataka je najčešće korišteni model baze podataka.

1.3.1. ERA model

ERA model *(engl. Entity Relationship Attribute*) je konceptualni model podataka koji prikazuje veze i odnose između pojedinih entiteta. U ERA modelu bitno je očuvanje veza pa tako veze prikazane u modelu smiju biti jedino 1:1 i 1:M. Dakle, ukoliko se pojavljuje veza M:N potrebno je uvesti slabi entitet tako da nastanu veze 1:M i N:1. Model je temelj za izradu baze podataka i definiranje veza između relacija.



Slika 2: Kardinalnost veza

2. PostgreSQL

PostgreSQL je besplatni sustav za upravljanje relacijsko/objektnim bazama podataka otvorenog koda. To je zadana baza podataka za macOS Server i također je dostupna za Linux, FreeBDS, OpenBSD i Windows. Sustav zadovoljava sve zahtjeve ACID-a, proširljiv je i drži se većine SQL:2011 standarada. PostgreSQL Global Development Group je raznolika skupina mnogih tvrtki i pojedinačnih suradnika koja je razvila ovaj SUBP. (PostgreSQL wiki, Izvor: https://en.wikipedia.org/wiki/PostgreSQL)

Slika 3: Logo

Uz uobičajena svojstva koja nudi, PostgreSQL pruža i mnogo mogućnosti za sistem administratore: backup, nadgledanje rada, replikacija i druge. Podupire razne SQL standarde i nudi mnogo modernih karakteristika poput složenih upita, vanjskih ključeva, triggera, pogleda.. Također, PostgreSQL može biti proširen od strane korisnika na mnogo načina, naprimjer dodavajući nove tipove podataka, funkcije, operacije, agregirajuće funkcije, indekse i proceduralne jezike. (PostgreSQL wiki, Izvor: https://en.wikipedia.org/wiki/PostgreSQL)

2.1.Povijest i razvoj

Sustav za upravljanje bazom podataka PostgreSQL razvio se iz projekta Ingres na Sveučilištu Berkley u Kaliforniji sedamdesetih godina prošloga stoljeća. Glavni začetnik je bio profesor Michael Stonebraker koji je konstantno radio na poboljšanju sustava i nakon što je Ingres preuzela tvrtka Relational Technologies/Ingres Corporation, Michael ne odustaje na usavršavanju postojećeg sustava. Osamdesetih godina Ingres dobiva objektno – orijentirana svojstva i mijenja naziv u Postgres. Sredinom devedesetih godina Postgres počinje podržavati SQL jezik te mijenja ime u PostgreSQL koji se koristi i danas. Danas PostgreSQL razvija grupa stručnjaka i profesora ujedinjenih u PostgreSQL Global Development Group. (PostgreSQL – službena stranica, 2020., Izvor: https://www.postgresql.org/docs/12/history.html)

Zadnja verzija sustava izdana je 14. studenog 2019. godine i to je verzija 12.1. Donosi neka poboljšanja u odnosu na ranije verzije što se tiče performansi i optimizacije. Ova verzija ne sadrži nove značajke, nego je težnja bila na podešavanju i dobro uglađenoj implementaciji postojećih PostgreSQL mogućnosti.

2.2. Nove značajke verzije 12.1.

Poboljšanja u verziji 12.1. koja je izdana 14.studenog 2019. godine uključuju poboljšanja performansi upita, posebno u pogledu većih skupova podataka i ukupne iskorištenosti prostora. Izdanje pruža programerima nove mogućnosti kao što su podrška za izraze SQL/JSON, optimizacije za izvršenje uobičajenih upita tablice i generirane stupce. U ovom izdanju uvodi se i prilagodljivo sučelje za pohranu tablica koje programerima omogućuje izradu vlastitih metoda za pohranu podataka. Ovo izdanje nastavlja trend olakšavanja upravljanja velikim i malim radnim opterećenjima baza podataka, istovremeno gradeći PostgreSQL-ovu reputaciju koja se temelji na fleksibilnosti, pouzdanosti i stabilnosti u proizvodnim okruženjima. Nakon što su rubni slučajevi postali podržani, performanse bolje, a nedostaci uklonjeni, slijedi osam najznačajnijih novosti koje donosi PostgreSQL 12. (PostgreSQL – službena stranica, 2020. Izvor: https://www.postgresql.org/about/news/1976/)

Particioniranje nije nova značajka koja se pojavljuje u ovoj verziji, ali je znatno poboljšano. Za korisnike koji se „premještaju“ iz drugih sustava i baza podataka s tisućama particija, PostgreSQL 12 sada donosi prednosti performansi pružajući mogućnosti koje istovremeno mogu učinkovito obraditi tisuće particija. Poboljšanja performansi particija mogu poboljšati izvedbu upita, posebno one sa INSERT i COPY klauzulama. Pored toga, korisnici sada mogu mijenjati particionirane tablice bez blokiranja upita i koristiti vanjske ključeve za referencirane tablice.
(Enterprisedb, 2020. Izvor: https://www.enterprisedb.com/blog/8-major-improvements-postgresql-12)

B-stablo jedna je od najkompliciranijih značajki napravljeno u PostgreSQL-u posljednjih godina. Veličine indeksa u više stupaca sada se smanjuju za do 40% učinkovitijom upotrebom prostora čime se štedi na diskovnom prostoru. Učinkovitost se poboljšava za indekse s duplikatima i učinkovitije se uklanjaju redovi, odnosno slogovi (*engl. tuples* – podatkovni objekti koji se sastoje od dvije i više komponenti) iz indeksa. Uz to, smanjuje se potreba za zaključavanjem tijekom ažuriranja indeksa. (https://www.enterprisedb.com/blog/8-major-improvements-postgresql-12)

Multi-Column Most-Common Value (MCV) - ovo ažuriranje, koje se razvija već nekoliko godina, ima za cilj riješiti probleme na koje se žalilo godinama – rubni slučaj povezanih stupaca

u upitu. PostgreSQL je do ove značajke bilježio samo jednu korelacijsku vrijednost za više stupaca. Sada je moguće usporediti više stupaca i usporediti kombinacije kako bi se optimizirali indeksi za upite. (https://www.enterprisedb.com/blog/8-major-improvements-postgresql-12)

Sada je moguće pokrenuti upite preko JSON dokumenata pomoću JSON izraza definiranih u SQL/JSON standardu i oni mogu koristiti postojeće mehanizme indeksiranja dokumenata pohranjenih u JSONB formatu za učinkovito dohvaćanje podataka. Uvode se generirani stupci koji izračunavaju vrijednost iz sadržaja drugih stupaca u istoj tablici. (Severalnines 2020. Izvor: https://severalnines.com/database-blog/whats-new-postgresql-12)

2.3. Naredbe

U SQL jeziku se naredbe mogu podijeliti u četiri osnovne grupe:

- DDL (*engl. Data Definition Language*) – naredbe za kreiranje i definiranje objekata. U nju spadaju CREATE, ALTER, DROP i RENAME
- DML (*engl. Data Manipulation Language*) – naredbe za manipuliranje podacima. To su naredbe INSERT, UPDATE i DELETE
- DQL (*engl. Data Query Language*) – naredba za postavljanje upita, SELECT
- DCL (*engl. Data Control Language*) – naredbe za kontroliranje, GRANT i REVOKE

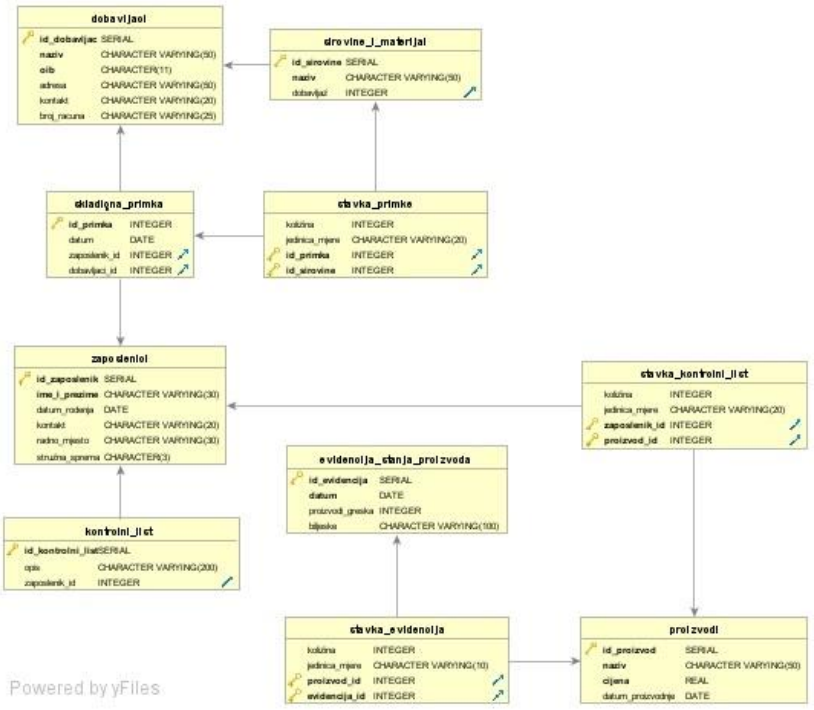
3. Implementacija baze podataka u PostgreSQL-u

Kako bi se najefikasnije prikazale funkcionalnosti i značajke koje sustav nudi, na primjeru baze podataka za skladište drvne industrije će korak po korak biti opisana cijela izrada kao i neke značajke koje postoje, a za izradu ove baze neće biti potrebne.

Kako PostgreSQL radi sa relacijama, odnosno tablicama, prvo je potrebno osmisliti korištene tablice i njihove međusobne veze. Za to će nam najbolje poslužiti ERA model čije su značajke već opisane na prethodnim stranicama rada. Entiteti koji će se koristiti kod implementacije baze su sljedeći:

- Dobavljači
- Zaposlenici
- Proizvodi
- Kontrolni list
- Evidencija stanja proizvoda
- Sirovine i materijal
- Skladišna primka

Slijedi prikaz ERA modela sa zadanim atributima koji je generiran u alatu DbVisualizer na temelju tablica iz PostgreSQL-a nakon kojeg je izrađen relacijski model u kojem su prikazani primarni i vanjski ključevi za svaku relaciju:



Slika 4: ERA model skladišta (izrada autora)

U relacijskom modelu su prikazane relacije i njihovi atributi. Primarni ključ svake relacije je jedinstven i podebljan, a vanjski je osjenčan sivom bojom i kod nekih relacija je dvokomponentni jer se radi o asocijativnom entitetu koji je dodan između tablica koje su bile spojene vezom M:N.

Dobavljači (**ID_dobavljač**, Naziv, OIB, Adresa, Kontakt, Broj_računa)
Zaposlenici (**ID_zaposlenik**, Ime_i_prezime, Datum_rođenja, Kontakt, Radno_mjesto, Stručna_sprema)
Proizvodi (**ID_proizvod**, Naziv, Cijena, Datum_proizvodnje)
Kontrolni list (**ID_kontrolni_list**, Opis, Zaposlenik_ID)
Stavka_kontrolni_list (Količina, Jedinica_mjere**, ID_proizvoda, ID_zaposlenika**)
Evidencija stanja proizvoda **(ID_evidencija**, Datum, Proizvodi_greška, Bilješke)
Stavka_evidencija (Količina, Jedinica_mjere, **ID_proizvoda, ID_evidencije**)
Sirovine i materijal (**ID_sirovine**, Naziv, Količina, Jedinica_mjere, ID_dobavljača)
Skladišna primka (ID_primka, Datum, **ID_zaposlenika, ID_dobavljača**)
Stavka_primke (Količina, Jedinica_mjere, **ID_dobavljača, ID_sirovine**)

3.1. Kreiranje korisnika

Naredbom *CREATE ROLE* se kreiraju korisnici i njihove ovlasti. Sintaksa naredbe glasi ovako:

```
CREATE ROLE name [ [ WITH ] option [ ... ] ]

where option can be:

      SUPERUSER | NOSUPERUSER
    | CREATEDB | NOCREATEDB
    | CREATEROLE | NOCREATEROLE
    | INHERIT | NOINHERIT
    | LOGIN | NOLOGIN
    | REPLICATION | NOREPLICATION
    | BYPASSRLS | NOBYPASSRLS
    | CONNECTION LIMIT connlimit
    | [ ENCRYPTED ] PASSWORD 'password' | PASSWORD NULL
    | VALID UNTIL 'timestamp'
    | IN ROLE role_name [, ...]
    | IN GROUP role_name [, ...]
    | ROLE role_name [, ...]
    | ADMIN role_name [, ...]
    | USER role_name [, ...]
    | SYSID uid
```

Naredbom se stvara novi PostgreSQL korisnik (ili točnije, ulogu). Samo korisnici sa privilegijom *CREATEROLE* mogu stvarati nove korisnike. Korisniku se može dati privilegija da može kreirati novu bazu podataka (*CREATEDB*), da se može logirati (LOGIN) i imati svoju lozinku za prijavu (*PASSWORD*). Također, korisnik može biti *SUPERUSER* koji ima sve ovlasti nad bazom. Zadano je da je kod kreiranja *NOSUPERUSER*.

Kreiranje nove uloga naziva *anja* koja ima ovlast kreiranja nove baze podataka:

```
SQL Shell (psql)
Server [localhost]:
Database [postgres]:
Port [5432]:
Username [postgres]:
Password for user postgres:
psql (12.1)
WARNING: Console code page (852) differs from Windows code page (1250)
         8-bit characters might not work correctly. See psql reference
         page "Notes for Windows users" for details.
Type "help" for help.

postgres=# create role anja createdb;
CREATE ROLE
postgres=#
```

Slika 5: CREATE ROLE

Ako se želimo prebaciti u tu ulogu koristit ćemo naredbu *set role* <naziv>, a ako nismo sigurni u kojoj se ulozi trenutno nalazimo koristit ćemo *select current_user*

3.2. Kreiranje baze podataka

Za kreiranje baze podataka koristi se naredba *CREATE DATABASE*. Za brisanje baze koristi se *DROP DATABASE*. Ako se želimo spojiti na odabranu bazu koristit ćemo *connect* <naziv_baze>. Sintaksa naredbe je sljedeća:

```
CREATE DATABASE name
    [ [ WITH ] [ OWNER [=] user_name ]
           [ TEMPLATE [=] template ]
           [ ENCODING [=] encoding ]
           [ LC_COLLATE [=] lc_collate ]
           [ LC_CTYPE [=] lc_ctype ]
           [ TABLESPACE [=] tablespace_name ]
           [ ALLOW_CONNECTIONS [=] allowconn ]
           [ CONNECTION LIMIT [=] connlimit ]
           [ IS_TEMPLATE [=] istemplate ] ]
```

Kreiranje baze podataka naziva *skladiste*:

```
postgres=> create database skladiste;
CREATE DATABASE
postgres=> \connect skladiste;
You are now connected to database "skladiste" as user "postgres".
skladiste=# _
```

Slika 6: CREATE DATABASE

3.3. Kreiranje tablica

Kreiranje tablica radi se naredbom CREATE TABLE i sintaksa je sljedeća:

```
CREATE [ [ GLOBAL | LOCAL ] { TEMPORARY | TEMP } | UNLOGGED ] TABLE [ IF NOT EXISTS ] table_name ( [
  { column_name data_type [ COLLATE collation ] [ column_constraint [ ... ] ]
    | table_constraint
    | LIKE source_table [ like_option ... ] }
    [, ... ]
] )
[ INHERITS ( parent_table [, ... ] ) ]
[ PARTITION BY { RANGE | LIST | HASH } ( { column_name | ( expression ) } [ COLLATE collation ] [ opclass ] [, ... ] ) ]
[ USING method ]
[ WITH ( storage_parameter [= value] [, ... ] ) | WITHOUT OIDS ]
[ ON COMMIT { PRESERVE ROWS | DELETE ROWS | DROP } ]
[ TABLESPACE tablespace_name ]
```

Dakle, prilikom kreiranja tablice potrebno je navesti ime tablice, atribute (kolone) tablice i pripadajući tip podataka. Isto tako prilikom kreiranja tablice definira se primarni ključ tablice kao i vanjski ključevi, odnosno na koju se tablica referencira.
Naredba za brisanje tablice glasi:

DROP TABLE naziv;

Kreiranje tablica započinje sa kreiranjem tablica koje nemaju vanjske ključeve. *Id_dobavljac* je primarni ključ tablice te se on definira nakon tipa podataka upisivanjem *primary key.* Atribut *naziv* ima znakovni tip podataka *varchar (50)* što označuje da naziv ima varijabilnu dužinu do 50 znakova, a *not null* označava da ta kolona ne smije biti prazna. Isto tako je definiran *OIB* samo što je kod njega *char (11)* što označuje da je broj znakova fiksni i iznosi 11. Ako je tablica uspješno kreirana ispisat će se CREATE TABLE.

```
skladiste=# create table dobavljaci(
skladiste(# id_dobavljac serial primary key,
skladiste(# naziv varchar (50) not null,
skladiste(# OIB char (11) not null,
skladiste(# adresa varchar (50),
skladiste(# kontakt varchar (20),
skladiste(# broj_racuna varchar (25));
CREATE TABLE
```

Slika 7: Tablica DOBAVLJAČI

Na isti način bit će kreirane ostale tablice koje nemaju vanjske ključeve, dakle potrebno je kreirati tablice *zaposlenici*, *proizvodi* i *evidencija stanja proizvoda*.

Kod definiranja atributa ***stručna_sprema*** u tablicu *zaposlenici* pomoću *check* provjerava se da prilikom unosa podataka u tablicu vrijednost može biti jedino NSS, SSS ili VSS.

```
skladiste=# create table zaposlenici(
skladiste(# id_zaposlenik serial primary key,
skladiste(# ime_i_prezime varchar(30) not null,
skladiste(# datum_rodenja date,
skladiste(# kontakt varchar(20),
skladiste(# radno_mjesto varchar(30),
skladiste(# stručna_sprema char (3) check(stručna_sprema IN ('NSS','SSS','VSS')));
CREATE TABLE
```

Slika 8: Tablica ZAPOSLENICI

```
skladiste=# create table proizvodi(
skladiste(# id_proizvod serial primary key,
skladiste(# naziv varchar (50) not null,
skladiste(# cijena real not null check(cijena>0),
skladiste(# datum_proizvodnje date);
CREATE TABLE
```

Slika 9: Tablica PROIZVODI

```
skladiste=# create table evidencija_stanja_proizvoda(
skladiste(# id_evidencija serial primary key,
skladiste(# datum date not null,
skladiste(# proizvodi_greska int,
skladiste(# biljeske varchar (100));
CREATE TABLE
```

Slika 10: Tablica EVIDENCIJA STANJA PROIZVODA

Potrebno je još kreirati tablice *kontolni list*, *stavka kontrolni list*, *stavka evidencija*, *sirovine i materijal*, *skladišna primka* i *stavka skladišna primka*. Kreirat će se prvo tablice koje imaju samo primarni i vanjski ključ, dakle one kod kojih ne postoji dvokomponentni primarni ključ koji

je ujedno i vanjski. To su tablice kontrolni list, sirovine i materijal i skladišna primka. Vanjski ključ se definira tako što se nakon upisivanja atributa i tipa podataka (koji mora biti jednak tipu podataka primarnog ključa na koji se tablica referencira) upiše *references* i tablica na koju se povezuje.

```
skladiste=# create table kontrolni_list(
skladiste(# id_kontrolni_list serial primary key,
skladiste(# opis varchar (200),
skladiste(# zaposlenik_id integer references zaposlenici);
CREATE TABLE
```

Slika 11: Tablica KONTROLNI LIST

```
skladiste=# create table sirovine_i_materijal(
skladiste(# id_sirovine serial primary key,
skladiste(# naziv varchar(50) not null,
skladiste(# količina integer,
skladiste(# jedinica_mjere varchar(10) check(jedinica_mjere IN('komad','metar')),
skladiste(# dobavljac_id integer references dobavljaci);
CREATE TABLE
```

Slika 12: Tablica SIROVINE I MATERIJAL

```
skladiste=# create table skladišna_primka(
skladiste(# id_primka serial primary key,
skladiste(# datum date,
skladiste(# zaposlenik_id integer references zaposlenici,
skladiste(# dobavljaci_id integer references dobavljaci);
CREATE TABLE
```

Slika 13: Tablica SKLADIŠNA PRIMKA

Kreiranje tablica *stavka kontolni list*, *stavka evidencija* i *stavka skladišna primka* kreiraju se na sličan način osim što je potrebno definirati i dvokomponentni primarni ključ koji se radi na sljedeći način:

```
skladiste=# create table stavka_kontrolni_list(
skladiste(# količina integer,
skladiste(# jedinica_mjere varchar (20),
skladiste(# zaposlenik_id integer references zaposlenici,
skladiste(# proizvod_id integer references proizvodi,
skladiste(# primary key(zaposlenik_id,proizvod_id));
CREATE TABLE
```

Slika 14: Tablica STAVKA KONTROLNI LIST

```
skladiste=# create table stavka_evidencija(
skladiste(# količina integer,
skladiste(# jedinica_mjere varchar(10),
skladiste(# proizvod_id integer references proizvodi,
skladiste(# evidencija_id integer references evidencija_stanja_proizvoda,
skladiste(# primary key(proizvod_id, evidencija_id));
CREATE TABLE
```

Slika 15: Tablica STAVKA EVIDENCIJA

```
skladiste=# create table stavka_primke(
skladiste(# količina integer,
skladiste(# jedinica_mjere varchar(20),
skladiste(# id_primka integer references skladišna_primka,
skladiste(# id_sirovine integer references sirovine_i_materijal,
skladiste(# primary key(id_primka,id_sirovine));
CREATE TABLE
```

Slika 16: Tablica STAVKA PRIMKE

3.3.1. Ograničenja podataka

Prilikom definiranja tablica vidljivo je da je bitno koristiti određena ograničenja nad podacima koji će se kasnije unositi. Slijede neka bitna ograničenja bez kojih nema smisla raditi kreirati tablice.

NULL ili NOT NULL – sprječavanje unosa NULL vrijednosti u stupac koji je definiran kao NOT NULL. Primjerice, primarni ključ ne smije poprimiti NULL vrijednost pa je bitno postaviti ograničenje da ono bude NOT NULL.

CHECK – definiranje intervala vrijednosti koji sustav može poprimiti. Dakle, moguće je staviti interval brojeva od 1 do 10 ili primjerice da godina ne smije biti niža od 2015.

UNIQUE – sprječavanje unosa duplih vrijednosti u jednom stupcu, tj. koristi se kada je bitno da svi podaci u stupcu imaju različite vrijednosti. Primjerice, ne možemo unijeti isti broj mobitela za dvije osobe jer svaka osoba ima različit.

DEFAULT – omogućava da stupac sa ovim ograničenjima tijekom unosa vrijednosti poprimi definiranu početnu vrijednost ukoliko se ne unese neka druga vrijednost.

PRIMARY KEY – jedan ili više atributa koji jedinstveno identificiraju svaki redak u tablici.

FOREIGN KEY - jedan ili više atributa koji se koriste za uspostavljanje veza između tablica. Vanjski ključ je stupac (ili više njih) u drugoj (ili istoj) tablici u koji pohranjujemo vrijednost (i) primarnog ključa prve tablice.

ON DELETE <akcija>

ON UPDATE <akcija>

- SET NULL – vrijednost vanjskog ključa postaje NULL

- SET DEFAULT – vrijednost vanjskog ključa postaje DEFAULT
- CASCADE – ista naredba (DELETE ili UPDATE) će se izvršiti i u tablici potomaka
- NO ACTION – ne dozvoljava izvršavanje UPDATE ili DELETE naredbe kojima bi se prekršio referencijalni integritet
- RESTRICT – isto kao i NO ACTION, s dodatnim ograničenjem da se akcija ne može dogoditi

3.3.2. Tipovi podataka

PostgreSQL podržava sve standardne SQL tipove podataka. Postoji ih mnogo i postoje razne varijante, a najčešće korištene dane su u tablici:

NAZIV	OPIS
NUMERIČKI	
Smallint	cijeli brojevi
Integer, int	cijeli brojevi
Bigint	cijeli brojevi
Decimal	brojevi do 1000 znamenaka
Numeric	brojevi do 1000 znamenaka
Real	decimalni brojevi, preciznost do 6 decimala
Double precision	decimalni brojevi, preciznost do 15 decimala
Smallserial	sekvenca
Serial	sekvenca
Bigserial	sekvenca
NOVČANI	
Money	novčani iznos
ZNAKOVNI	
Varchar (n)	niz znakova varijabilne dužine
Char (n)	niz znakova fiksne dužine
Text	niz znakova varijabilne dužine bez limita
VREMENSKI	
Timestamp	datum i vrijeme
Time	vrijeme
Date	datum

Tablica 1: Tipovi podataka

3.4. INSERT naredba

Nakon kreiranja tablica one su prazne pa je potrebno unijeti podatke u njih. Za dohvaćanje tablice i podataka u njima koristi se naredba SELECT. Za dohvaćanje svih redova koristi se *, a za definiranje tablice iz koje se podaci dohvaćaju klauzula FROM. Cijela naredba će detaljnije biti opisana u poglavlju o jednostavnim upitima, a sada je potrebna samo kako bi se vidjela struktura tablice u koju će se unositi podaci. Dakle, cijela naredba za dohvaćanje podataka iz tablice *proizvodi* glasi:

```
skladiste=# select *from proizvodi;
 id_proizvod | naziv | cijena | datum_proizvodnje
-------------+-------+--------+-------------------
(0 rows)
```

Slika 17: Struktura tablice PROIZVODI

Kod popunjavanja tablica prvo se kreće sa popunjavanjem onih u kojima nema vanjskih ključeva, a zatim se popunjavaju ostale. Kod INSERT naredbe podaci se upisuju u jednostruke navodnike, osim ako je riječ o *default* vrijednostima ili *integeru* onda se oni navode bez njih.

```
INSERT INTO table_name [ AS alias ] [ ( column_name [, ...] ) ]
    [ OVERRIDING { SYSTEM | USER} VALUE ]
    { DEFAULT VALUES | VALUES ( { expression | DEFAULT } [, ...] ) [, ...] | query }
    [ ON CONFLICT [ conflict_target ] conflict_action ]
    [ RETURNING * | output_expression [ [ AS ] output_name ] [, ...] ]
```

Primjer popunjavanja tablica sa podacima pomoću INSERT naredbe:

```
skladiste=# insert into proizvodi values(default,'Stol OLYMP',700,'10-02-2020');
INSERT 0 1
skladiste=# insert into proizvodi values(default,'Stolica MOON',300,'12-02-2020');
INSERT 0 1
skladiste=# insert into proizvodi values(default,'Ormar KING',1500,'14-02-2020');
INSERT 0 1
skladiste=# insert into stavka_evidencija values(3,'komad',1,1);
INSERT 0 1
skladiste=# insert into stavka_evidencija values(4,'komad',2,1);
INSERT 0 1
skladiste=# insert into stavka_evidencija values(2,'komad',2,2);
INSERT 0 1
skladiste=# insert into stavka_evidencija values(5,'komad',3,3);
INSERT 0 1
skladiste=# insert into stavka_evidencija values(3,'komad',1,3);
INSERT 0 1
skladiste=# insert into stavka_primke values(100,'metar',1,2);
INSERT 0 1
skladiste=# insert into stavka_primke values(1,'komad',1,4);
INSERT 0 1
skladiste=# insert into stavka_primke values(10,'metar',2,3);
INSERT 0 1
skladiste=# insert into kontrolni_list values(default, 'Obavljena kontrola proizvoda',2);
INSERT 0 1
skladiste=# insert into kontrolni_list values(default, 'Obavljena kontrola proizvoda',2);
INSERT 0 1
skladiste=# insert into kontrolni_list values(default, 'Obavljena kontrola proizvoda',4);
INSERT 0 1
skladiste=# insert into sirovine_i_materijal values(default,'Hrast slavonski','100','metar',6);
INSERT 0 1
skladiste=# insert into sirovine_i_materijal values(default,'Ariš','50','metar',7);
INSERT 0 1
skladiste=# insert into sirovine_i_materijal values(default,'Šarafi 200KOM','1','komad',8);
INSERT 0 1
skladiste=# insert into sirovine_i_materijal values(default,'Tračna pila','1','komad',6);
INSERT 0 1
```

Slika 18: INSERT naredba

3.4.1. Naredba ALTER TABLE

Za brisanje, odnosno za promjenu strukture tablice koristi se naredba ALTER TABLE. Ovom naredbom moguće je dodavati, brisati ili preimenovati stupce. Moguće je mijenjanje tipa podataka postojećeg stupca kao i preimenovanje postojeće tablice. Isto tako naredbom se može dodati ograničenje, a neke od najkorištenijih sintaksi su sljedeće:

```
ALTER TABLE [ IF EXISTS ] [ ONLY ] name [ * ]
    action [, ... ]
ALTER TABLE [ IF EXISTS ] [ ONLY ] name [ * ]
    RENAME [ COLUMN ] column_name TO new_column_name
ALTER TABLE [ IF EXISTS ] [ ONLY ] name [ * ]
    RENAME CONSTRAINT constraint_name TO new_constraint_name
ALTER TABLE [ IF EXISTS ] name
    RENAME TO new_name
ALTER TABLE [ IF EXISTS ] name
    SET SCHEMA new_schema
ALTER TABLE ALL IN TABLESPACE name [ OWNED BY role_name [, ... ] ]
    SET TABLESPACE new_tablespace [ NOWAIT ]
ALTER TABLE [ IF EXISTS ] name
    ATTACH PARTITION partition_name { FOR VALUES partition_bound_spec | DEFAULT }
ALTER TABLE [ IF EXISTS ] name
    DETACH PARTITION partition_name

where action is one of:

    ADD [ COLUMN ] [ IF NOT EXISTS ] column_name data_type [ COLLATE collation ] [ column_constraint [ ... ] ]
    DROP [ COLUMN ] [ IF EXISTS ] column_name [ RESTRICT | CASCADE ]
    ALTER [ COLUMN ] column_name [ SET DATA ] TYPE data_type [ COLLATE collation ] [ USING expression ]
    ALTER [ COLUMN ] column_name SET DEFAULT expression
    ALTER [ COLUMN ] column_name DROP DEFAULT
    ALTER [ COLUMN ] column_name { SET | DROP } NOT NULL
    ALTER [ COLUMN ] column_name ADD GENERATED { ALWAYS | BY DEFAULT } AS IDENTITY [ ( sequence_options ) ]
    ALTER [ COLUMN ] column_name DROP IDENTITY [ IF EXISTS ]
    ALTER [ COLUMN ] column_name SET STATISTICS integer
    ALTER [ COLUMN ] column_name SET ( attribute_option = value [, ... ] )
    ALTER [ COLUMN ] column_name RESET ( attribute_option [, ... ] )
    ALTER [ COLUMN ] column_name SET STORAGE { PLAIN | EXTERNAL | EXTENDED | MAIN }
```

Naredbom ALTER TABLE u tablici sirovine i materijal obrisane su suvišne kolone količina i jedinica mjere na način:

```
skladiste=# select *from sirovine_i_materijal;
 id_sirovine |      naziv      | količina | jedinica_mjere | dobavljac_id
-------------+-----------------+----------+----------------+--------------
           2 | Hrast slavonski |      100 | metar          |            6
           3 | Ariš            |       50 | metar          |            7
           4 | Šarafi 200KOM   |        1 | komad          |            8
           5 | Tračna pila     |        1 | komad          |            6
(4 rows)


skladiste=# alter table sirovine_i_materijal drop column količina;
ALTER TABLE
skladiste=# alter table sirovine_i_materijal drop column jedinica_mjere;
ALTER TABLE
skladiste=# select *from sirovine_i_materijal;
 id_sirovine |      naziv      | dobavljac_id
-------------+-----------------+--------------
           2 | Hrast slavonski |            6
           3 | Ariš            |            7
           4 | Šarafi 200KOM   |            8
           5 | Tračna pila     |            6
(4 rows)
```

Slika 19: ALTER TABLE naredba (1)

Za promjenu naziva stupca koristi se sljedeća naredba:

```
skladiste=# alter table sirovine_i_materijal rename column dobavljac_id to dobavljač;
ALTER TABLE
skladiste=# select *from sirovine_i_materijal;
 id_sirovine |      naziv       | dobavljač
-------------+------------------+-----------
           2 | Hrast slavonski |         6
           3 | Ariš            |         7
           4 | Šarafi 200KOM   |         8
           5 | Tračna pila     |         6
(4 rows)
```

Slika 20: ALTER TABLE naredba (2)

3.4.2. Naredbe UPDATE i DELETE

UPDATE i DELETE su naredbe koje služe za promjenu/ažuriranje i brisanje podataka u tablicama. Dakle, ukoliko je potrebno promijeniti podatak unutar stupca koristi se naredba koja glasi ovako:

UPDATE *tablica* SET *stupac*=nova_vrijednost WHERE *uvjet* ;

U tablici proizvodi mijenja se cijena sa 300 na 400 za proizvod čiji je id=2.Ažuriranje tog podatka radi se na sljedeći način:

```
skladiste=# select *from proizvodi;
 id_proizvod |    naziv     | cijena | datum_proizvodnje
-------------+--------------+--------+-------------------
           1 | Stol OLYMP   |    700 | 2020-02-10
           2 | Stolica MOON |    300 | 2020-02-12
           3 | Ormar KING   |   1500 | 2020-02-14
(3 rows)


skladiste=# update proizvodi set cijena=400 where id_proizvod=2;
UPDATE 1
skladiste=# select *from proizvodi;
 id_proizvod |    naziv     | cijena | datum_proizvodnje
-------------+--------------+--------+-------------------
           1 | Stol OLYMP   |    700 | 2020-02-10
           3 | Ormar KING   |   1500 | 2020-02-14
           2 | Stolica MOON |    400 | 2020-02-12
(3 rows)
```

Slika 21: UPDATE naredba (1)

Naredbu UPDATE možemo koristiti i ako povežemo dvije tablice. U tablici skladišna primka ažuriranje datuma pomoću imena i prezimena zaposlenika, ako je u tablici skladišna primka poznat id zaposlenika:

```
skladiste=# select *from zaposlenici;
 id_zaposlenik |  ime_i_prezime  | datum_rodenja |  kontakt   |    radno_mjesto    | stručna_sprema
---------------+-----------------+---------------+------------+--------------------+----------------
             2 | Marko Janić     | 1967-07-25    | 0912335887 | Skladištar         | SSS
             3 | Ivan Marić      | 1985-04-14    | 0987745563 | Voditelj skladišta | VSS
             4 | Hrvoje Modrić   | 1978-02-10    | 0995632887 | Skladištar         | SSS
             5 | Matej Ivanković | 1978-09-10    | 916778995  | Skladištar         | SSS
(4 rows)


skladiste=# select *from skladišna_primka;
 id_primka |   datum    | zaposlenik_id | dobavljaci_id
-----------+------------+---------------+---------------
         1 | 2020-02-10 |             3 |             6
         2 | 2020-02-10 |             3 |             7
         3 | 2020-02-10 |             3 |             8
         6 | 2020-02-26 |             3 |             9
         7 | 2020-03-20 |             3 |             9
         5 | 2020-05-10 |             2 |            10
         8 | 2020-05-10 |             2 |             7
         9 | 2020-05-10 |             2 |             8
        10 | 2020-05-10 |             2 |             9
(9 rows)

skladiste=# update skladišna_primka set datum=current_date from zaposlenici where zaposlenici.ime_i_prezime='Marko Janić'
skladiste-# and zaposlenici.id_zaposlenik=skladišna_primka.zaposlenik_id;
UPDATE 4
skladiste=# select *from skladišna_primka;
 id_primka |   datum    | zaposlenik_id | dobavljaci_id
-----------+------------+---------------+---------------
         1 | 2020-02-10 |             3 |             6
         2 | 2020-02-10 |             3 |             7
         3 | 2020-02-10 |             3 |             8
         6 | 2020-02-26 |             3 |             9
         7 | 2020-03-20 |             3 |             9
         5 | 2020-05-17 |             2 |            10
         8 | 2020-05-17 |             2 |             7
         9 | 2020-05-17 |             2 |             8
        10 | 2020-05-17 |             2 |             9
(9 rows)
```

Marko Janić, id=2
Skladišna_primka.datum=current_date

Slika 22: UPDATE naredba (2)

Naredba DELETE služi za brisanje redova iz tablice i njegova je sintaksa slična UPDATE naredbi i glasi:

DELETE FROM *tablica* WHERE *uvjet* ;

U tablicu proizvodi bit će dodan proizvod koji će se *DELETE* naredbom obrisati iz tablice i pritom je bitno uočiti kako uvjet ne mora biti zadan samo za id proizvoda, nego može biti za bilo koji stupac unutar tablice.

```
skladiste=# insert into proizvodi values (default,'Krevet SKY',1000,'15-02-2020');
INSERT 0 1
skladiste=# select *from proizvodi;
 id_proizvod |    naziv     | cijena | datum_proizvodnje
-------------+--------------+--------+-------------------
           1 | Stol OLYMP   |    700 | 2020-02-10
           3 | Ormar KING   |   1500 | 2020-02-14
           2 | Stolica MOON |    400 | 2020-02-12
           4 | Krevet SKY   |   1000 | 2020-02-15
(4 rows)


skladiste=# delete from proizvodi where naziv='Krevet SKY';
DELETE 1
```

Slika 23: DELETE naredba (1)

Ako se pokuša obrisati zaposlenik, to neće biti moguće ukoliko se neka druga tablica referencira na tablicu zaposlenici. Dakle, nije moguće obrisati zaposlenika ako je on primjerice izdao neki kontrolni list:

```
skladiste=# select *from zaposlenici;
 id_zaposlenik | ime_i_prezime | datum_rodenja |  kontakt   |    radno_mjesto    | stručna_sprema
---------------+---------------+---------------+------------+--------------------+----------------
             2 | Marko Janić   | 1967-07-25    | 0912335887 | Skladištar         | SSS
             3 | Ivan Marić    | 1985-04-14    | 0987745563 | Voditelj skladišta | VSS
             4 | Hrvoje Modrić | 1978-02-10    | 0995632887 | Skladištar         | SSS
(3 rows)


skladiste=# select *from kontrolni_list;
 id_kontrolni_list |             opis             | zaposlenik_id
-------------------+------------------------------+---------------
                 1 | Obavljena kontrola proizvoda |             2
                 2 | Obavljena kontrola proizvoda |             2
                 3 | Obavljena kontrola proizvoda |             4
                 4 | Obavljena kontrola proizvoda |             4
(4 rows)


skladiste=# delete from zaposlenici where id_zaposlenik='4';
ERROR:  update or delete on table "zaposlenici" violates foreign key constraint "kontrolni_list_zaposlenik_id_fkey" on table "kontrolni_list"
DETAIL:  Key (id_zaposlenik)=(4) is still referenced from table "kontrolni_list".
skladiste=#
```

Slika 24: DELETE naredba (2)

No, postoje klauzule koja omogućavaju da se zaposlenik i kontrolni listovi ipak obrišu. Prisjetimo se definiranja vanjskog ključa i klauzula CASCADE i RESTRICT koji su tada spoemnute. Ako prilikom kreiranja tablice kontrolni listovi i definiranja vanjskog ključa dodamo klauzulu ON DELETE CASCADE, to prouzrokuje da se brisanjem zaposlenika brišu i svi kontrolni listovi koje je on izdao. Klauzulom ON DELETE RESTRICT to onemogućujemo.

ON DELETE CASCADE ON UPDATE RESTRICT bi u prethodnom primjeru značilo da se brisanjem zaposlenika brišu svi kontrolni listovi, a zabranjuje se promjena primarnog ključa. ON DELETE RESTRICT ON UPDATE RESTRICT znači da nije moguće obrisati zaposlenika ukoliko je izdao barem jedan kontrolni list i zabranjuje se promjena primarnog ključa.

3.5. Indeksi

Indeksi se primarno koriste za poboljšanje performansi baze podataka iako neprimjerena upotreba može rezultirati sporijom izvedbom. Može se kreirati nad jednim ili više stupaca u tablici i služe za brže pretraživanje podataka u tablici.

Naredba za kreiranje indeksa glasi

CREATE [UNIQUE] INDEX *naziv* ON *tablica* ;

Kako PostgreSQL podržava B-tree, hash, GiST i GIN indekse, naredbu je moguće proširiti i odrediti vrstu indeksa koju želimo, a default je B-tree indeksiranje jer odgovara

najčešćim situacijama. Svaka vrsta indeksa koristi drugačiji algoritam koji je najprikladniji za različite vrste upita.

B-tree indeks će PostgreSQL koristiti kad god je indeksirani stupac uključen u usporedbu, npr. ako se koriste operatori uspoređivanja $<, <=, =, >, >=$ u SELECT naredbi. Konstrukcije ekvivalentne kombinacijama operatora BETWEEN i IN također se mogu implementirati u pretraživanju indeksa B-tree. Indeks se može koristiti za upite koji uključuju operator podudaranja LIKE. Hash indeksi mogu podnijeti jednostavne usporedbe jednakosti. Planer upita će razmotriti korištenje hash indeksa kad god je indeksirani stupac uključen u usporedbu koristeći operator = u SELECT naredbi. GiST indeksi se najčešće koriste za optimiziranje tablica koje reprezentiraju dvodimenzionalno geometrijsko tijelo. GIN indeksi koriste se za optimiziranje pretrage po objektima koji mogu sadržavati više od jedne vrijednosti, npr. polje. (PostgreSQL – službena stranica, 2020., Izvor: https://www.postgresql.org/docs/12/indexes-types.html)

Kreiranje jedinstvenog indeksa imena *naziv_indeks* nad stupcima *naziv* i *oib* u tablici *dobavljači* radi se na sljedeći način:

```
skladiste=# create unique index naziv_indeks on dobavljaci (naziv,oib);
CREATE INDEX
skladiste=#
```

Slika 25: B-TREE indeks

Kreiranje hash indeksa imena *cijena_indeks_hash* nad stupcem *cijena* u tablici *proizvodi* glasi ovako:

```
skladiste=# create index cijena_indeks_hash on proizvodi using hash (cijena);
CREATE INDEX
```

Slika 26: HASH indeks

3.6. Grupovne funkcije

Najčešće korištene grupovne funkcije u SQL-u su:

- COUNT
- SUM
- MIN
- MAX
- AVG

Funkcija COUNT broji broj redaka ili ne-null vrijednosti u odnosu na određeni stupac iz tablice. Kada se koristi zvjezdica (*) s funkcijom brojanja, vraća se ukupni broj redaka. Dakle, COUNT(*) broji sve redove, a COUNT (stupac) broji samo poznate vrijednost, tj. ne broji NULL vrijednosti.

```
skladiste=# select *from zaposlenici;
 id_zaposlenik | ime_i_prezime | datum_rodenja |  kontakt   |    radno_mjesto    | stručna_sprema
---------------+---------------+---------------+------------+--------------------+----------------
             2 | Marko Janić   | 1967-07-25    | 0912335887 | Skladištar         | SSS
             3 | Ivan Marić    | 1985-04-14    | 0987745563 | Voditelj skladišta | VSS
             4 | Hrvoje Modrić | 1978-02-10    | 0995632887 | Skladištar         | SSS
(3 rows)


skladiste=# select count(*) from zaposlenici;
 count
-------
     3
(1 row)
```

Slika 27: COUNT funkcija

Funkcijom SUM() dobiva se zbroj svih vrijednosti u stupcu, MIN() i MAX() služe za određivanje najmanje i najveće vrijednosti stupca, a AVG() daje prosjek svih vrijednosti. SUM() i AVG() se koriste kod numeričkih tipova podataka, dok se MIN() i MAX() mogu koristiti i kod znakovnih.

Određivanje najmanje i najveće cijene iz tablice proizvodi se pomoću funkcija MIN() i MAX() radi na sljedeći način:

```
skladiste=# select *from proizvodi;
 id_proizvod |    naziv     | cijena | datum_proizvodnje
-------------+--------------+--------+-------------------
           1 | Stol OLYMP   |    700 | 2020-02-10
           3 | Ormar KING   |   1500 | 2020-02-14
           2 | Stolica MOON |    400 | 2020-02-12
(3 rows)


skladiste=# select MIN(cijena), MAX(cijena) from proizvodi;
 min | max
-----+------
 400 | 1500
(1 row)
```

Slika 28: MIN() i MAX() funkcije

3.7. Upiti

Upiti služe za dohvaćanje redova iz tablice ili iz više njih. Ako se radi o dohvaćanju podataka iz jedne tablice radi se o jednostavnim upitima, a ako se povezuje dvije ili više tablica takvi upiti su složeni. Ispravan redoslijed klauzula za postavljanje upita je sljedeći:

SELECT – FROM – WHERE – GROUP BY – HAVING – ORDER BY

Klauzula SELECT služi za dohvaćanje stupca ili više njih iz tablice. FROM označava tablicu iz koje se podaci uzimaju, tj. tom klauzulom specificiraju se izvori podataka. WHERE služi za postavljanje uvjeta kada se nisu potrebni svi redovi iz tablice, nego samo određeni koji su u tom trenutku potrebni. GROUP BY služi za grupiranje podataka po određenom stupcu. HAVING klauzulom se eliminiraju podaci nastali grupiranjem koji ne zadovoljavaju neki uvjet. ORDER BY služi za sortiranje podataka. Ako se ne navede redoslijed kojim se želi sortirati zadano je da se sortira uzlazno (ASC), a dodavanjem DESC sortiranje je silazno.

3.7.1. Jednostavni upiti

U nekoliko sljedećih primjera opisani su najčešći jednostavni upiti i njihove sintakse. Dohvaćanje svih podataka iz tablice vrši se pomoću zvjezdice (*) koja slijedi nakon SELECT, a nakon FROM klauzule specificira se tablica iz koje se podaci uzimaju.

```
skladiste=# select *from proizvodi;
 id_proizvod |    naziv     | cijena | datum_proizvodnje
-------------+--------------+--------+-------------------
           1 | Stol OLYMP   |    700 | 2020-02-10
           2 | Stolica MOON |    300 | 2020-02-12
           3 | Ormar KING   |   1500 | 2020-02-14
(3 rows)
```

Slika 29: Jednostavni upit (1)

Nazivi i cijene proizvoda ako je cijena veća od 300:

```
skladiste=# select naziv, cijena from proizvodi where cijena>300;
   naziv    | cijena
------------+--------
 Stol OLYMP |    700
 Ormar KING |   1500
(2 rows)
```

Slika 30: Jednostavni upit (2)

Nazivi proizvoda koji počinju slovom S:

```
skladiste=# select naziv from proizvodi where naziv like 'S%';
    naziv
--------------
 Stol OLYMP
 Stolica MOON
(2 rows)
```

Slika 31: Jednostavni upit (3)

Imena i prezimena zaposlenika čija je stručna sprema SSS:

```
skladiste=# select *from zaposlenici;
 id_zaposlenik | ime_i_prezime | datum_rodenja |  kontakt   |    radno_mjesto    | stručna_sprema
---------------+---------------+---------------+------------+--------------------+----------------
             2 | Marko Janić   | 1967-07-25    | 0912335887 | Skladištar         | SSS
             3 | Ivan Marić    | 1985-04-14    | 0987745563 | Voditelj skladišta | VSS
             4 | Hrvoje Modrić | 1978-02-10    | 0995632887 | Skladištar         | SSS
(3 rows)


skladiste=# select ime_i_prezime from zaposlenici where stručna_sprema='SSS';
 ime_i_prezime
---------------
 Marko Janić
 Hrvoje Modrić
(2 rows)
```

Slika 32: Jednostavni upit (4)

Imena i prezimena i radno mjesto zaposlenika sortirani prema imenu i prezimenu uzlazno, a zatim silazno:

```
skladiste=# select ime_i_prezime, radno_mjesto from zaposlenici order by ime_i_prezime;
 ime_i_prezime |    radno_mjesto
---------------+--------------------
 Hrvoje Modrić | Skladištar
 Ivan Marić    | Voditelj skladišta
 Marko Janić   | Skladištar
(3 rows)


skladiste=# select ime_i_prezime, radno_mjesto from zaposlenici order by ime_i_prezime desc;
 ime_i_prezime |    radno_mjesto
---------------+--------------------
 Marko Janić   | Skladištar
 Ivan Marić    | Voditelj skladišta
 Hrvoje Modrić | Skladištar
(3 rows)
```

Slika 33: Jednostavni upit (5)

Za rad sa upitima korisno je spomenuti i LIMIT I OFFSET. LIMIT se koristi za limitiranje broja redova koje želimo dobiti upitom. Klauzula OFFSET služi za preskakanje prvih redova, ovisno o tome koji broj se definira.

Naziv iz tablice proizvodi koji prikazuje dva reda (*limit 2*) s tim da se prvi red (*offset 1*) preskoči:

```
skladiste=# select *from proizvodi;
 id_proizvod |    naziv     | cijena | datum_proizvodnje
-------------+--------------+--------+-------------------
           1 | Stol OLYMP   |    700 | 2020-02-10
           2 | Stolica MOON |    300 | 2020-02-12
           3 | Ormar KING   |   1500 | 2020-02-14
(3 rows)


skladiste=# select naziv from proizvodi limit 2 offset 1;
    naziv
--------------
 Stolica MOON
 Ormar KING
(2 rows)
```

Slika 34: LIMIT i OFFSET klauzule

3.7.2. Složeni upiti

Složenim upitima se dohvaćaju podaci iz dvije ili više tablica i to iz onih koje su povezane ključevima. Dakle, pri definiranju upita u WHERE klauzuli bitno je povezati primarni ključ jedne i pripadajući vanjski ključ druge tablice.

```
postgres=# \connect skladiste;
You are now connected to database "skladiste" as user "postgres".
skladiste=# select *from sirovine_i_materijal;
 id_sirovine |      naziv      | dobavljač
-------------+-----------------+-----------
           2 | Hrast slavonski |         6
           3 | Ariš            |         7
           4 | Šarafi 200KOM   |         8
           5 | Tračna pila     |         6
(4 rows)

skladiste=# select *from dobavljaci;
 id_dobavljac |    naziv     |     oib     |        adresa         |    kontakt     | broj_racuna
--------------+--------------+-------------+-----------------------+----------------+--------------
            6 | DI Čazma     | 26545874551 | Zagrebačka 20         | +385912447884 | HR4210210012
            7 | DI Bjelovar  | 65445210018 | Bilogorska 70         | +385984474120 | HR6632120121
            8 | DI Novoselec | 54774410210 | Ulica braće Radića 9  | +385995332001 | HR0032026654
(3 rows)
```

Slika 35: Povezivanje tablica

Dohvaćanje podataka iz obje tablice moguće je ako se te tablice povežu na ispravan način. Pri tome treba izjednačiti ime tablice i stupca koji je primarni ključ sa drugom tablicom i stupcem koji je vanjski ključ.

Ako se upitom želi ispisati naziv sirovina i materijala i naziv dobavljača to se radi na sljedeći način:

```
SELECT sirovine_i_materijal.naziv, dobavljaci.naziv
FROM sirovine_i_materijal, dobavljaci
WHERE sirovine_i_materijal.dobavljač=dobavljaci.id_dobavljac;
```

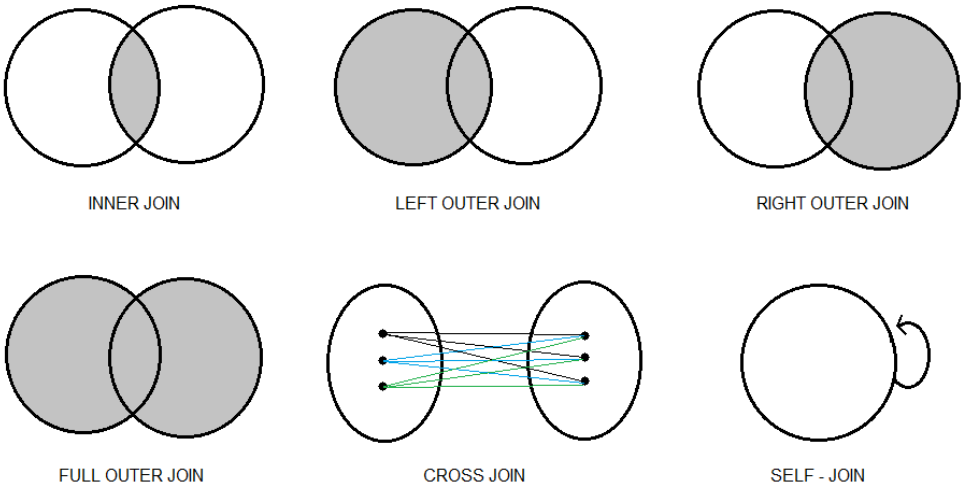
```
skladiste=# select sirovine_i_materijal.naziv, dobavljaci.naziv from sirovine_i_materijal,dobavljaci
skladiste-# where sirovine_i_materijal.dobavljač=dobavljaci.id_dobavljac;
      naziv      |    naziv
-----------------+--------------
 Hrast slavonski | DI Čazma
 Ariš            | DI Bjelovar
 Šarafi 200KOM   | DI Novoselec
 Tračna pila     | DI Čazma
(4 rows)
```

Slika 36: Složeni upit

3.7.3. JOIN sintaksa

U PostgreSQL-u sintaksa JOIN koristi se za kombiniranje stupaca iz jedne ili više tablica na temelju vrijednosti sličnih stupaca između tablica. Slični stupci su uglavnom stupac primarnog ključa iz prve tablice i stupac vanjskog ključa iz druge tablice. JOIN je novija sintaksa koja je zamjenjuje WHERE klauzulu jer je na taj način spajanje tablica brže. PostgreSQL podržava sljedeća spajanja:

- INNER JOIN
- LEFT JOIN
- RIGHT JOIN
- FULL OUTER JOIN
- CROSS JOIN
- NATURAL JOIN
- SELF-JOIN



Slika 37: Spajanje tablica (Izrada autora prema: https://stackoverflow.com/questions/45987384/using-and-in-an-inner-join)

INNER JOIN spaja tablice na način da spoji samo ono što je zajedničko u obje tablice, dakle, odabire sve retke iz obje tablice sve dok postoji podudaranje između stupaca odabranih tablica.

LEFT OUTER JOIN vraća sve retke iz lijeve tablice te odgovarajuće redove iz desne tablice i pri tome ispisuje NULL vrijednosti ako nema podudaranja.

RIGHT OUTER JOIN vraća sve redove iz desne tablice te odgovarajuće redove iz lijeve tablice i pri tome ispisuje NULL vrijednost ako nema podudaranja.

FULL OUTER JOIN vraća sve redove iz lijeve i desne tablice i tako objedinjuje rezultat spojeva LEFT i RIGHT JOIN.

CROSS JOIN spajanje spaja svaki podatak i prve tablice sa svakim podatkom iz druge tablice, dakle, svaki podatak iz druge tablice je povezan sa svakim iz prve.

SELF – JOIN omogućava samostalno spajanje tablice, tj. dozvoljava spajanje tablice sa samom sobom. Korisno je kada se žele usporediti redovi unutar iste tablice.

Prikaz naziva dobavljača i naziva sirovina i materijala tako da se prikazuju svi podaci iz prve tablice (dobavljači) i vrijednosti iz druge tablice. Pri tome se ispisuju NULL vrijednosti za podatke iz desne tablice koje nemaju podudarajuće vrijednosti iz prve lijeve tablice.

```
skladiste=# select dobavljaci.naziv, sirovine_i_materijal.naziv
skladiste-# from dobavljaci left join sirovine_i_materijal on
skladiste-# sirovine_i_materijal.dobavljač=dobavljaci.id_dobavljac;
    naziv     |      naziv
--------------+-----------------
 DI Čazma     | Hrast slavonski
 DI Bjelovar  | Ariš
 DI Novoselec | Šarafi 200KOM
 DI Čazma     | Tračna pila
 DI Ilok      |
 DI Osijek    |
(6 rows)
```

Slika 38: LEFT JOIN

CROSS JOIN spajanje koje svakom podatku iz prve tablice pridružuje svaki podatak iz druge tablice. Tablica dobavljači ima pet redova, a sirovine i materijal četiri reda što spajanjem daje ukupno $5 * 4 = 20$ redova:

```
skladiste=# select dobavljaci.naziv, sirovine_i_materijal.naziv
skladiste-# from dobavljaci cross join sirovine_i_materijal;
    naziv     |      naziv
--------------+-----------------
 DI Čazma     | Hrast slavonski
 DI Bjelovar  | Hrast slavonski
 DI Novoselec | Hrast slavonski
 DI Osijek    | Hrast slavonski
 DI Ilok      | Hrast slavonski
 DI Čazma     | Ariš
 DI Bjelovar  | Ariš
 DI Novoselec | Ariš
 DI Osijek    | Ariš
 DI Ilok      | Ariš
 DI Čazma     | Šarafi 200KOM
 DI Bjelovar  | Šarafi 200KOM
 DI Novoselec | Šarafi 200KOM
 DI Osijek    | Šarafi 200KOM
 DI Ilok      | Šarafi 200KOM
 DI Čazma     | Tračna pila
 DI Bjelovar  | Tračna pila
 DI Novoselec | Tračna pila
 DI Osijek    | Tračna pila
 DI Ilok      | Tračna pila
(20 rows)
```

Slika 39: CROSS JOIN

3.8. Pogledi

Pogledi u SQL-u su virtualne, nematerijalizirane relacije ili pohranjeni upiti. „View“ ili pogled predstavlja način prikazivanja određenih podataka iz baze. Podaci iz upita se mogu mijenjati, ažurirati i brisati kao i u radu s običnim upitima.
(Krstarica, 2020. Izvor: https://www.krstarica.com/zivot/tehnika/sql-pogled-view/)

Naredba za kreiranje pogleda je CREATE VIEW i cijela sintaksa glasi:

```
CREATE [ OR REPLACE ] [ TEMP | TEMPORARY ] [ RECURSIVE ] VIEW name [ ( column_name [, ...] ) ]
    [ WITH ( view_option_name [= view_option_value] [, ... ] ) ]
    AS query
    [ WITH [ CASCADED | LOCAL ] CHECK OPTION ]
```

Za brisanje pogleda koristi se naredba DROP VIEW *naziv*;

CREATE OR REPLACE VIEW ima slično značenje, ali ako pogled sa istim nazivom već postoji, bit će zamijenjen. Novi upit mora generirati iste stupce koji su bili generirani postojećim upitom u pogledu.

Jednostavni se prikazi automatski ažuriraju. Sustav će omogućiti da se izrazi INSERT, UPDATE i DELETE koriste u prikazu na isti način kao i na uobičajenoj tablici. Prikazi se automatski ažurira ako zadovoljava uvjete:

- Pogled mora imati točno jedan unos s popisa FROM, što mora biti tablica ili drugi prikaz koji je moguće ažurirati
- Definicija pogleda ne smije sadržavati klauzule WITH, DISTINCT, GROUP BY, HAVING, LIMIT ili OFFSET na gornjoj razini
- Definicija pogleda ne smije sadržavati zadane operacije (UNION, INTERSECT ili EXCEPT) na gornjoj razini
- Popis za odabir vlasničkog pregleda ne smije sadržavati agregate, funkcije prozora ili funkcije koje vraćaju postavke

(PostgreSQL – službena stranica, 2020. Izvor: https://www.postgresql.org/docs/12/sql-createview.html)

Kreiranje jednostavnog pogleda naziva *pogled_proizvodi* koji dohvaća sve podatke iz tablice proizvodi i brisanje pogleda:

```
skladiste=# create view pogled_proizvodi
skladiste-# as select *from proizvodi;
CREATE VIEW
skladiste=# select *from pogled_proizvodi;
 id_proizvod |    naziv     | cijena | datum_proizvodnje
-------------+--------------+--------+-------------------
           1 | Stol OLYMP   |    700 | 2020-02-10
           3 | Ormar KING   |   1500 | 2020-02-14
           2 | Stolica MOON |    400 | 2020-02-12
(3 rows)


skladiste=# drop view pogled_proizvodi;
DROP VIEW
```

Slika 40: Pogled (1)

Dakle, za poglede je bitno poznavati sintaksu jednostavnih i složenih upita jer se pogledi rade na temelju njih.

Pogled koji dohvaća podatke iz tablica *dobavljači* i *sirovine i materijal* te ispisuje naziv i adresu dobavljača te naziv sirovina i materijala radi se na temelju složenog upita na sljedeći način:

```
skladiste=# create view pogled2
skladiste-# as select dobavljaci.naziv "Dobavljači", dobavljaci.adresa "Adresa", sirovine_i_materijal.naziv "Sirovine i materijal"
skladiste-# from dobavljaci, sirovine_i_materijal
skladiste-# where sirovine_i_materijal.dobavljač=dobavljaci.id_dobavljac;
CREATE VIEW
skladiste=# select *from pogled2;
  Dobavljači  |        Adresa        | Sirovine i materijal
--------------+----------------------+----------------------
 DI Čazma     | Zagrebačka 20        | Hrast slavonski
 DI Bjelovar  | Bilogorska 70        | Ariš
 DI Novoselec | Ulica braće Radića 9 | Šarafi 200KOM
 DI Čazma     | Zagrebačka 20        | Tračna pila
(4 rows)
```

Slika 41: Pogled (2)

Prilikom kreiranja pogleda moguće je imenovati nazive stupaca po želji na način da se prilikom odabira stupaca (nakon SELECT) u dvostruke navodnike navede ime koje želimo dati pojedinom stupcu. Pogled koji sadrži sve dobavljače neovisno o tome jesu li dobavili koju sirovinu ili ne kao i nazivi sirovina i materijala

3.8.1. Materijalizirani pogledi

Materijalizirani prikaz upita nastaje naredbom CREATE MATERIALIZED VIEW, a cijela sintaksa glasi:

```
CREATE MATERIALIZED VIEW [ IF NOT EXISTS ] table_name
    [ (column_name [, ...] ) ]
    [ USING method ]
    [ WITH ( storage_parameter [= value] [, ... ] ) ]
    [ TABLESPACE tablespace_name ]
    AS query
    [ WITH [ NO ] DATA ]
```

Upit se izvršava i koristi se za popunjavanje pogleda u trenutku kada je naredba izdana i može se kasnije osvježiti pomoću naredbe REFRESH MATERIALIZED VIEW. Materijalizirani pogled je sličan naredbi CREATE TABLE AS, osim što pamti i upit koji je korišten za inicijaliziranje pogleda, tako da se kasnije na zahtjev može osvježiti. Materijalizirani pogled ima mnoga ista svojstva kao i tablica, ali ne postoji podrška za privremene materijalizirane poglede.

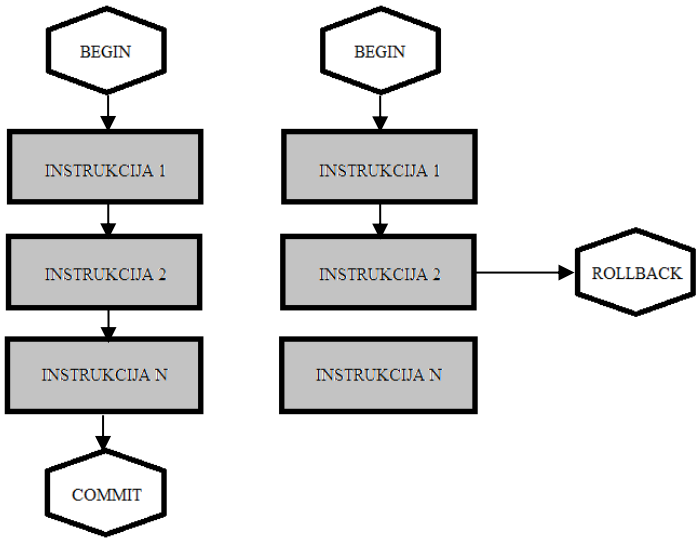
Materijalizirani pogled koji sadrži sve dobavljače (naziv i OIB), neovisno o tome jesu li dostavili koju sirovinu i materijal kao i nazivi sirovina i materijala radi se na sljedeći način:

```
skladiste=# create materialized view pogled_materialized as
skladiste-# select dobavljaci.naziv "Dobavljač", dobavljaci.oib "OIB", sirovine_i_materijal.naziv "Sirovine i materijal"
skladiste-# from dobavljaci left join sirovine_i_materijal on
skladiste-# sirovine_i_materijal.dobavljač=dobavljaci.id_dobavljac;
SELECT 5
skladiste=# select *from pogled_materialized;
  Dobavljač   |     OIB     | Sirovine i materijal
--------------+-------------+----------------------
 DI Čazma     | 26545874551 | Hrast slavonski
 DI Bjelovar  | 65445210018 | Ariš
 DI Novoselec | 54774410210 | Šarafi 200KOM
 DI Čazma     | 26545874551 | Tračna pila
 DI Osijek    | 86372637777 |
(5 rows)
```

Slika 42: Materijalizirani pogled

3.9. Transakcije

Transakcija je skup instrukcija koje se izvršavaju u cijelosti ili se ne izvršavaju uopće. Svaka transakcija započinje operacijom *BEGIN*, a završava sa *COMMIT* ako je uspješno izvršena. Operacija *ROLLBACK* vraća bazu u stanje u kojem je bila prije izvođenja transakcije, dakle, u bazi se ne događaju promjene jer transakcija nije bila uspješna.



Slika 43: Izvršavanje transakcije (izrada autora prema: https://vladmihalcea.com/a-beginners-guide-to-acid-and-database-transactions/)

Razlozi zbog kojih ne dolazi do potpunog izvođenja transakcije mogu biti razni, primjerice nestanak struje, kvar hardvera, nemogućnost promjene podataka i mnogi drugi. Zbog toga je bitno da transakcija ima određena svojstva, a ona su:

- Atomnost (*engl. **A**tomicity*)
- Konzistentnost (*engl. **C**onsistency*)
- Izolacija (*engl. **I**solation*)
- Trajnost (*engl. **D**urability*)

Atomnost jamči da se svaka transakcija tretira kao nevidljiva jedinica koja se ili izvršava u potpunosti ili ne izvršava uopće. Ako se instrukcija koja sačinjava transakciju ne dovrši, cijela transakcija tada ne uspijeva i baza ostaje nepromijenjena. Sustav mora jamčiti atomnost u svakoj situaciji.

Konzistentnost transakcije osigurava da transakcija može dovesti bazu podataka iz jednog valjanog stanja u drugo.

Izolacija osigurava da istodobno izvršavanje transakcija ostavlja bazu podataka u istom stanju koje bi bilo dobiveno da su transakcije izvršene sekvencijalno.

Trajnost podrazumijeva da se dovršene transakcije kasnije ne mogu prekinuti ili da se njihovi rezultati odbace. One se moraju nastaviti kroz, naprimjer, ponovo pokretanje SUBP-a nakon njegova kraha.

Sintaksa operacija *BEGIN, COMMIT* i *ROLLBACK* u PostgreSQL-u je sljedeća:

```
Command:     BEGIN
Description: start a transaction block
Syntax:
BEGIN [ WORK | TRANSACTION ] [ transaction_mode [, ...] ]

where transaction_mode is one of:

    ISOLATION LEVEL { SERIALIZABLE | REPEATABLE READ | READ COMMITTED | READ UNCOMMITTED }
    READ WRITE | READ ONLY
    [ NOT ] DEFERRABLE

URL: https://www.postgresql.org/docs/12/sql-begin.html


skladiste=# \h commit;
Command:     COMMIT
Description: commit the current transaction
Syntax:
COMMIT [ WORK | TRANSACTION ] [ AND [ NO ] CHAIN ]

URL: https://www.postgresql.org/docs/12/sql-commit.html


skladiste=# \h rollback;
Command:     ROLLBACK
Description: abort the current transaction
Syntax:
ROLLBACK [ WORK | TRANSACTION ] [ AND [ NO ] CHAIN ]
```

Transakcija kojom se u tablicu *zaposlenici* dodaje novi radnik i zatim se dodaje u tablicu *kontrolni list* nakon kontrole proizvoda:

```
skladiste=# begin;
BEGIN
skladiste=# insert into zaposlenici values (default, 'Matej Ivanković','10-09-1978',0916778995,'Skladištar','SSS');
INSERT 0 1
skladiste=# insert into kontrolni_list values (default, 'Obavljena kontrola proizvoda',5);
INSERT 0 1
skladiste=# commit;
COMMIT
```

Slika 44: Naredba COMMIT

Transakcija završava naredbom COMMIT što označava da je uspješno izvedena, no slijedi primjer gdje SQL naredbe nisu izvršene u cijelosti te stanje baze ostaje nepromijenjeno te izgleda kao i prije transakcije:

```
skladiste=# select *from zaposlenici;
 id_zaposlenik |  ime_i_prezime  | datum_rodenja |  kontakt   |    radno_mjesto    | stručna_sprema
---------------+-----------------+---------------+------------+--------------------+----------------
             2 | Marko Janić     | 1967-07-25    | 0912335887 | Skladištar         | SSS
             3 | Ivan Marić      | 1985-04-14    | 0987745563 | Voditelj skladišta | VSS
             4 | Hrvoje Modrić   | 1978-02-10    | 0995632887 | Skladištar         | SSS
             5 | Matej Ivanković | 1978-09-10    | 916778995  | Skladištar         | SSS
(4 rows)


skladiste=# begin;
BEGIN
skladiste=# insert into zaposlenici values (default, 'Matej Horvat','30-06-1988',0987896678,'Skladištar','SSS');
INSERT 0 1
skladiste=# rollback;
ROLLBACK
skladiste=# select *from zaposlenici;
 id_zaposlenik |  ime_i_prezime  | datum_rodenja |  kontakt   |    radno_mjesto    | stručna_sprema
---------------+-----------------+---------------+------------+--------------------+----------------
             2 | Marko Janić     | 1967-07-25    | 0912335887 | Skladištar         | SSS
             3 | Ivan Marić      | 1985-04-14    | 0987745563 | Voditelj skladišta | VSS
             4 | Hrvoje Modrić   | 1978-02-10    | 0995632887 | Skladištar         | SSS
             5 | Matej Ivanković | 1978-09-10    | 916778995  | Skladištar         | SSS
(4 rows)
```

Slika 45: Naredba ROLLBACK

3.10. Funkcije

Kao i kod ostalih SUBP i PostgreSQL nudi mogućnost povezivanja niza SQL izraza koji se tretiraju kao jedinica funkcionalnog bloka i ta operacija se naziva – funkcija. Osim objedinjavanja nekoliko SQL izraza, one daju i mogućnost izvršavanja SQL izraza korištenjem proceduralnih jezika. Struktura funkcije je sljedeća:

```
CREATE [ OR REPLACE ] FUNCTION
    name ( [ [ argmode ] [ argname ] argtype [ { DEFAULT | = } default_expr ] [, ...] ] )
    [ RETURNS rettype
      | RETURNS TABLE ( column_name column_type [, ...] ) ]
  { LANGUAGE lang_name
```

Naredba započinje sa CREATE ili REPLACE što označava kreiranje ili zamjenu funkcije pa se zatim stavlja naziv funkcije. Argtype označava vrstu podataka. RETURNS vraća rezultat funkcije, a LANGUAGE označava jezik u kojem je funkcija napisana, u ovom slučaju to je SQL.

Jedna od najatraktivnijih značajki PostgreSQL-a je ta da on nije ograničen samo na SQL jezik. Moguće je izabrati ugradnju naprednih funkcionalnosti, uključujući mogućnost korištenja različitih jezika za izgradnju funkcija, čime se postiže velika fleksibilnost pomoću boljih uvjeta generiranja i prednosti svojstvenih različitih jezicima. (Administracija Portala Sustav 2020. Izvor: https://hr.admininfo.info/postgresql-funciones)

Funkcija koja za naziv dobavljača vraća broj sirovina i materijal koje on dobavlja:

```
skladiste=# create function funkcija1 (varchar) returns bigint as
skladiste-# 'select count(*) from dobavljaci, sirovine_i_materijal where
skladiste'# dobavljaci.id_dobavljac=sirovine_i_materijal.dobavljač
skladiste'# and dobavljaci.naziv=$1 group by dobavljaci.naziv;'
skladiste-# language sql;
CREATE FUNCTION
skladiste=# select *from dobavljaci;
 id_dobavljac |    naziv     |     oib     |          adresa           |    kontakt    | broj_racuna
--------------+--------------+-------------+---------------------------+---------------+--------------
            6 | DI Čazma     | 26545874551 | Zagrebačka 20             | +385912447884 | HR4210210012
            7 | DI Bjelovar  | 65445210018 | Bilogorska 70             | +385984474120 | HR6632120121
            8 | DI Novoselec | 54774410210 | Ulica braće Radića 9      | +385995332001 | HR0032026654
            9 | DI Osijek    | 86372637777 | Ulica kralja Tomislava 89 | +385916679009 | HR3663290909
           10 | DI Ilok      | 93777463742 | Osječka 99                | +385914445886 | HR9890600547
(5 rows)


skladiste=# select *from sirovine_i_materijal;
 id_sirovine |      naziv      | dobavljač
-------------+-----------------+-----------
           2 | Hrast slavonski |         6
           3 | Ariš            |         7
           4 | Šarafi 200KOM   |         8
           5 | Tračna pila     |         6
(4 rows)


skladiste=# select funkcija1 ('DI Čazma');
 funkcija1
-----------
         2
(1 row)
```

Slika 46: Funkcija (1)

Prethodni primjer prikazuje funkciju koja vraća neku vrijednost, no funkcije ne moraju uvijek vraćati nešto. One mogu i unositi, brisati ili ažurirati podatke. Za to se koriste već prije spomenute naredbe INSERT, UPDATE i DELETE.

Funkcija koja za unesenog zaposlenika i dobavljača upisuje novu skladišnu primku sa datumom kada je izdana:

```
skladiste=# create function funkcija2 (int,int) returns void as
skladiste-# 'insert into skladišna_primka(datum,zaposlenik_id,dobavljaci_id) values
skladiste'# (current_date, $1,$2);'
skladiste-# language sql;
CREATE FUNCTION
skladiste=# select funkcija2(3,9);
 funkcija2
-----------

(1 row)


skladiste=# select *from skladišna_primka;
 id_primka |   datum    | zaposlenik_id | dobavljaci_id
-----------+------------+---------------+---------------
         1 | 2020-02-10 |             3 |             6
         2 | 2020-02-10 |             3 |             7
         3 | 2020-02-10 |             3 |             8
         5 | 2020-02-26 |             2 |            10
         6 | 2020-02-26 |             3 |             9
(5 rows)
```

Slika 47: Funkcija (2)

3.11. Particioniranje

Za baze podataka s izuzetno velikim tablicama, particioniranje se koristi za poboljšanje performansi baze podataka i znatno olakšavanje održavanja. Particioniranje dijeli tablicu u više tablica i obično se vrši na način da aplikacije koje pristupaju tablici ne primjećuju nikakvu razliku, osim što brže pristupaju podacima koji su joj potrebni. Podjelom tablice u više tablica, ideja je omogućiti izvršenje upita da bi morale skenirati mnogo manje tablice i indekse kako bi pronašli potrebne podatke. Bitno je da se particioniranje izvrši na odgovarajući način jer u suprotnom može znatno pogoršati performanse.

Tablica prodaja koja sadrži datum, naziv i broj prodanih komada proizvoda:

```
skladiste=# create table prodaja(
skladiste(# broj_prodaje int not null,
skladiste(# datum date not null,
skladiste(# naziv varchar (30),
skladiste(# broj_komada int
skladiste(# ) PARTITION BY RANGE (datum);
CREATE TABLE
```

Slika 48: Tablica PRODAJA

Izrada particioniranih tablica prodaja_2019 i prodaja_2020 koje će sadržavati prodane proizvode prema godini u kojoj su prodani:

```
skladiste=# create table prodaja_2019 PARTITION OF prodaja
skladiste-# FOR VALUES FROM ('2019-01-01') TO ('2019-12-31');
CREATE TABLE
skladiste=# create table prodaja_2020 PARTITION OF prodaja
skladiste-# FOR VALUES FROM ('2020-01-01') TO ('2020-12-31');
CREATE TABLE
```

Slika 47: Kreiranje particioniranih tablica

Nakon unosa podataka u tablicu prodaja moguće je prikazati sve podatke koji se nalaze u particioniranim tablicama. Tablica prodaja_2019 sadrži sve proizvode koji zadovoljavaju postavljeni uvjet da je datum prodaje period između 01.01.2019. i 31.12.2019. godine:

```
skladiste=# select *from prodaja;
 broj_prodaje |   datum    |    naziv     | broj_komada
--------------+------------+--------------+-------------
            1 | 2019-03-06 | Stol OLYMP   |           3
            2 | 2020-02-02 | Stolica MOON |          10
(2 rows)


skladiste=# select *from prodaja_2019;
 broj_prodaje |   datum    |   naziv    | broj_komada
--------------+------------+------------+-------------
            1 | 2019-03-06 | Stol OLYMP |           3
(1 row)
```

Slika 48: Particionirane tablice

Kako bi se ubrzalo pretraživanje, nad particioniranim tablicama moguće je kreirati indekse:

```
skladiste=# CREATE INDEX ON prodaja_2019 (datum);
CREATE INDEX
skladiste=# CREATE INDEX ON prodaja_2020 (datum);
CREATE INDEX
skladiste=#
```

Slika 49: Kreiranje indeksa nad particioniranim tablicama

3.12. Okidači

Naredba za kreiranje novog okidača (engl. Trigger) je CREATE TRIGGER, a cijela sintaksa glasi:

```
CREATE [ CONSTRAINT ] TRIGGER name { BEFORE | AFTER | INSTEAD OF } { event [ OR ... ] }
    ON table_name
    [ FROM referenced_table_name ]
    [ NOT DEFERRABLE | [ DEFERRABLE ] [ INITIALLY IMMEDIATE | INITIALLY DEFERRED ] ]
    [ REFERENCING { { OLD | NEW } TABLE [ AS ] transition_relation_name } [ ... ] ]
    [ FOR [ EACH ] { ROW | STATEMENT } ]
    [ WHEN ( condition ) ]
    EXECUTE { FUNCTION | PROCEDURE } function_name ( arguments )

where event can be one of:

    INSERT
    UPDATE [ OF column_name [, ... ] ]
    DELETE
    TRUNCATE
```

Okidač je programska procedura sustava za upravljanje bazom podataka koja se aktivira određenim događajem, a oni mogu biti:

- Upis novog zapisa u tablicu
- Brisanje određenog zapisa
- Ažuriranje postojećeg zapisa

Okidač u stvari pokreće određenu aktivnost nad bazom uvijek kada se dogodi jedan od navedenih događaja.

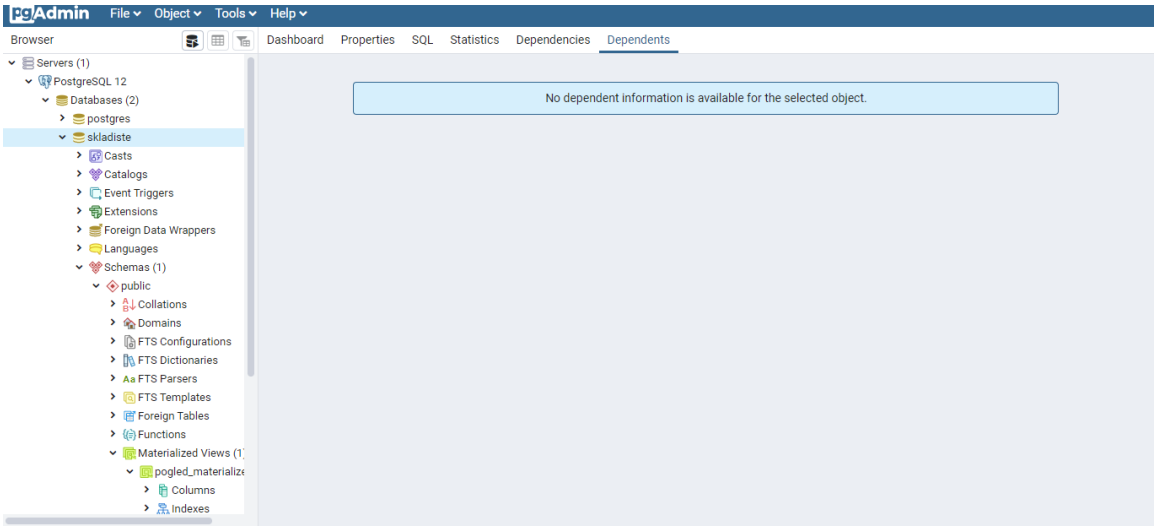
Okidač funkcija ažurirano() koja je ugrađena unutar svake tablice i zapisuje vrijeme tipa timestamp kada je red unutar tablice promijenjen ili nadodan:

```
skladiste=# create function public.ažurirano()
skladiste-# returns trigger
skladiste-# language 'plpgsql' cost 100
skladiste-# volatile not leakproof as $body$
skladiste$# begin
skladiste$# new.ažurirano=current.timestamp;
skladiste$# return new;
skladiste$# end
skladiste$# $body$;
CREATE FUNCTION
```

Slika 50: Okidač funkcija

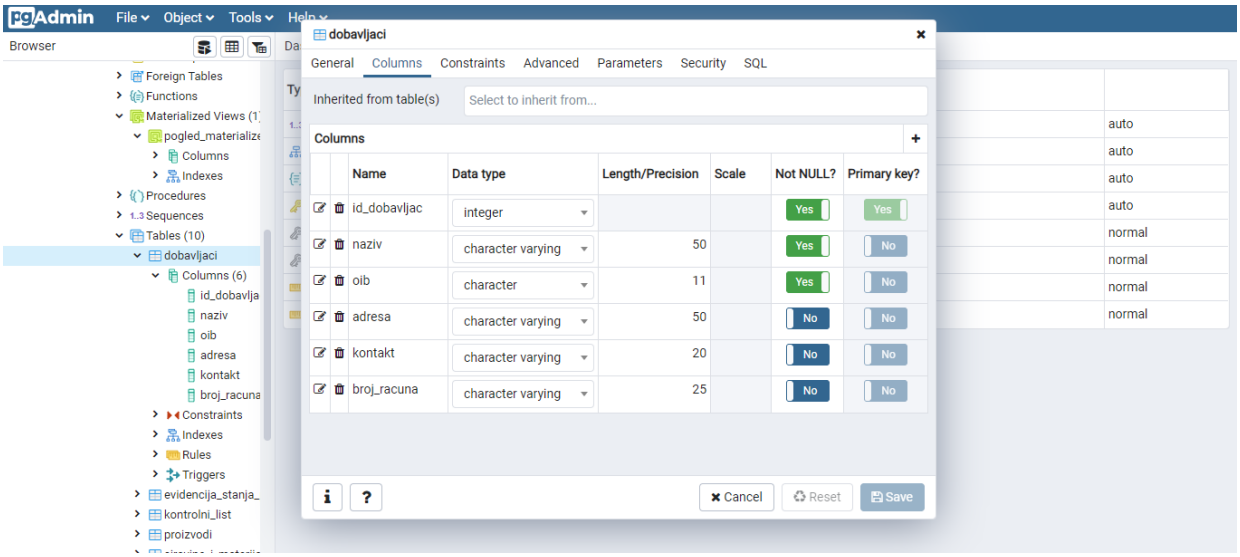
4. pgAdmin 4

pgAdmin je vodeći alat za upravljanje otvorenim kodom za PostgreSQL, najnapredniju svjetsku bazu podataka otvorenih izvora. pgAdmin 4 osmišljen je kako bi zadovoljio potrebe početnika kao i iskusnih Postgres korisnika. Korisniku je pruženo moćno grafičko sučelje koje pojednostavljuje stvaranje, održavanje i upotrebu objekata baze podataka. Zadnja dostupna verzija je 4.17 koja je izdana 9.siječnja 2020. godine i u njoj je otklonjeno mnogo nedostataka i uvedene su dvije značajke koje starija verzija 4.16 nije imala.



Slika 51: Početni zaslon pgAdmin

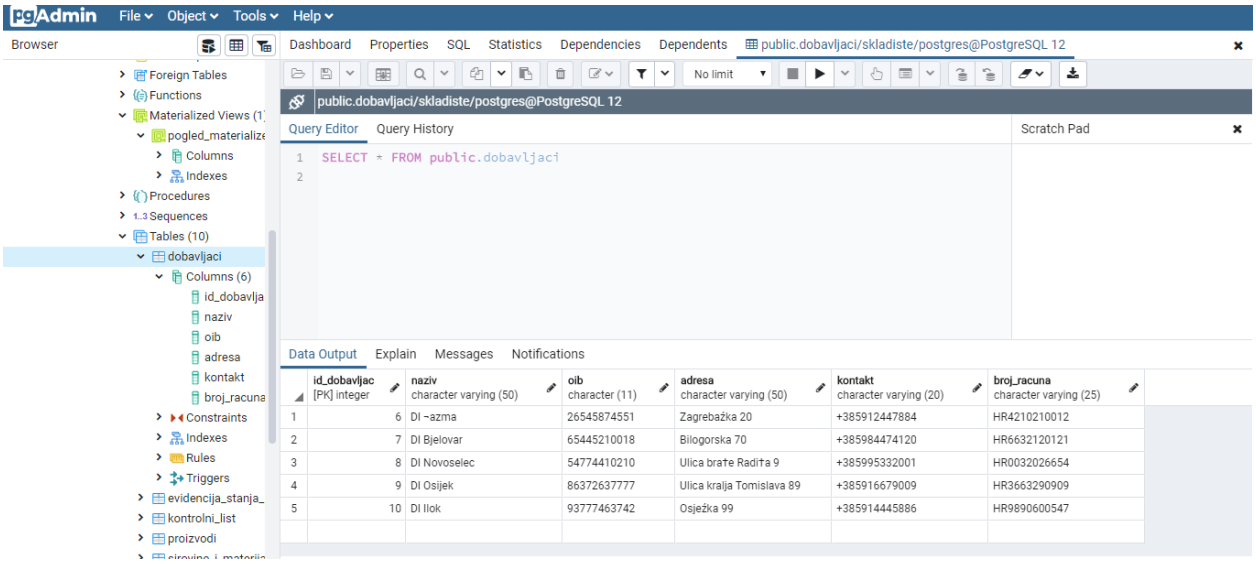
U izborniku s lijeve strane nalaze se baze podataka koje su napravljene u PostgreSQL-u. Moguće je vidjeti sve tablice, poglede, indekse, funkcije i ostale strukture kreirane u SQLShell-u. Moguće je mijenjati strukturu tablica, dodavati atribute, mijenjati tipove podataka i sve što bi se moglo napraviti preko naredbi, samo na jednostavniji način preko sučelja.



Slika 52: Mijenjanje strukture tablice u pgAdmin-u

Jednostavnim klikom na tablicu moguće je vidjeti sva njena svojstva i strukturu i također ju je moguće promijeniti.

Desnim klikom na tablicu i biranjem opcije *View/Edit data* moguće je vidjeti podatke koji su upisani u tablicama:



Slika 53: Prikaz podataka u pgAdmin-u

5. Zaključak

Baza podataka je skup informacija iz realnog svijeta koje su pohranjene, kod relacijskih baza podataka, u tablicama (relacijama). Poznavanje osnovnih i naprednih koncepata sustava za upravljanje bazama podataka je bitno u današnjem modernom svijetu i poslovanju. Baza podataka je osnova svakog programskog proizvoda. Postoje razni SUBP koje te većinom temelje na SQL jeziku pa im je osnova vrlo slična i ukoliko se pozna jedan sustav, ostali se mogu lako naučiti koristiti.

PostgreSQL je sustav koji je relativno lako koristiti te je besplatan i dostupan svima i moguće ga je instalirati na sve operacijske sustave. U radu je korištena psql konzola, a moguće je koristiti i grafički alat pgAdmin koji ima user – friendly sučelje. Također, bazu podataka napravljenu u PostgreSQL-u moguće je spojiti na neki softver poput alata DbVisualizer, kojim je moguće dobiti prikaz ERA modela.

Sustav nudi brojne napredne mogućnosti, a verzija 12.1. upravo nudi poboljšane performanse koje se posebno očituju ukoliko se radi sa velikim skupovima podataka jer znatno olakšavaju i ubrzavaju manipulaciju podataka.

6. Literatura

[1] Kaštelan T., Mesic I., (2010.) Uvod u baze podataka – Priručnik, Algebra

[2] Krstarica (2020.) SQL – Pogled (View) preuzeto 18.veljače 2020. s: (https://www.krstarica.com/zivot/tehnika/sql-pogled-view/

[3] Maleković M., Rabuzin K., (2016.) Uvod u baze podataka, Fakultet organizacije i informatike, Varaždin

[4] Medak i ostali (2005.) Usporedba komercijalnih i slobodnih sustava za upravljanje bazama prostornih podataka, preuzeto 17.veljače 2020. s: https://www.researchgate.net/publication/315407901_Usporedba_komercijalnih_i_slobodnih_sustava_za_upravljanje_bazama_prostornih_podataka

[5] PostgreSQL (2020.) A Brief History of PostgreSQL, preuzeto 01.veljače 2020. s: https://www.postgresql.org/docs/12/history.html

[6] PostgreSQL (2020.) Documentation, preuzeto 15.veljače 2020. s: https://www.postgresql.org/docs/12/sql-createview.html

[7] pgAdmin (2020.) Development, preuzeto 20.veljače 2020. s: https://www.pgadmin.org/docs/pgadmin4/development/index.html

[8] Rabuzin K., (2014.) SQL – Napredne teme, Fakultet organizacije i informatike, Varaždin

[9] Rabuzin K., (2011.) Uvod u SQL, Fakultet organizacije i informatike, Varaždin

[10] Severalnines (2020.) PostgreSQL Features and Improvements, preuzeto 05.veljače 2020. s: https://severalnines.com/database-blog/whats-new-postgresql-12

[11] Wikipedija (2020.) Relacijska baza podataka, preuzeto 15.veljače 2020. s: https://hr.wikipedia.org/wiki/Relacijska_baza_podataka

Popis slika

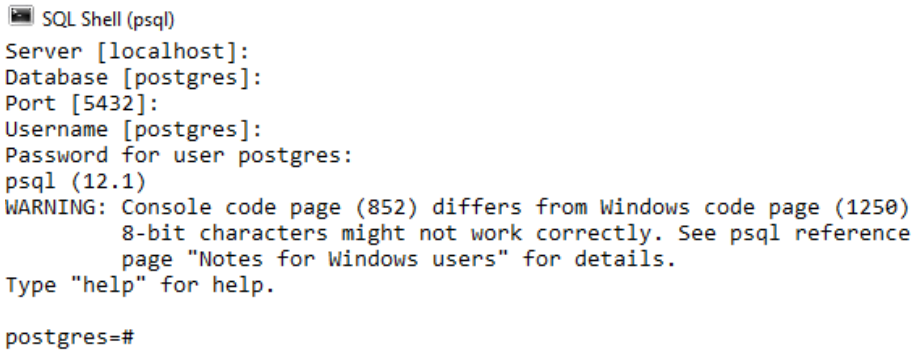
Popis tablica

Prilozi

Kako je PostgreSQL besplatan sustav otvorenog koda, sama instalacija je jednostavna i brza. Na njihovoj službenoj stranici moguće je preuzeti i instalirati verzije PostgreSQL 9.4, 9.5, 9.6, 10, 11 i 12. Također, sustav je dostupan za razne operacijske sustave – BSD, Linux, macOS, Solaris i Windows.

Slijede upute za instalaciju Postgresa na Windows 10:

1. Na stranici www.postgres.org stisnuti *Download* te odabrati operacijski sustav na koji želimo instalirati Postgres u odjeljku *Binary packages.* Zatim kliknuti na link *Download the installer* gdje će se prikazati sve verzije. Ili jednostavnije klikom na link https://www.enterprisedb.com/downloads/postgres-postgresql-downloads dolazi se do istog mjesta za preuzimanje
2. Odabrati 32 ili 64-bitnu verziju te stisnuti *Download*
3. Slijediti upute za instalaciju nakon preuzimanja
4. Kada je instalacija završena na računalu se nalazi SQL Shell (psql) koji je spreman za rad te je za prijavu potrebna lozinka koja je izabrana prilikom instalacije



Slika 54: Početni zaslon SQL Shell (psql)