SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
VARAŽDIN

Filip Gatarić

APLIKACIJA ZA VOĐENJE STATISTIKE SKLADIŠTA I PLANIRANJE ZALIHA

PROJEKT

Varaždin, 2020.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Filip Gatarić

Matični broj: 42676/13–R

Studij: Informacijsko i programsko inženjerstvo

APLIKACIJA ZA VOĐENJE STATISTIKE SKLADIŠTA I PLANIRANJE ZALIHA

PROJEKT

Mentor/Mentorica:

Izv. prof. dr. sc. Markus Schatten

Varaždin, lipanj 2020.

Filip Gatarić

Izjava o izvornosti

Izjavljujem da je moj projekt 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

Ovaj rad prikazuje kreiranje stolne aplikacije za vođenje statistike i planiranje zaliha skladišta pomoću temporalnih i aktivnih baza podataka. Objašnjavaju se principi temporalnih i aktivnih baza podataka te se prikazuje implementacija istih. Automatizacija procesa je postignuta pomoću okidača u bazi podataka, tako da je proces naručivanja artikala sa skladišta u potpunosti automatiziran. Stolna aplikacija je rađena u Microsoft Visual Studio 2019 u C# jeziku kao Windows Forms aplikacija, a PostgreSQL je korišten za bazu podataka. Rad s aktivnim bazama podataka uvelike olakšava programiranje aplikacija pošto je većina logike pohranjena u ECU pravilima na bazi podataka.

Ključne riječi: baza podataka, PostgreSQL, aktivne baze podataka, temporalne baze podataka, ECA pravila, okidači, funkcije, C#

Sadržaj

1. Uvod

Primjer vođenja skladišta je dosta česti primjer u domeni baza podataka i kako god da je aplikacija za vođenje skladišta dobro, uvijek postoji neko poboljšanje. Jedno od elegantnijih rješenja za automatizaciju vođenja zaliha na skladištu jest pretvoriti bazu podataka u aktivnu bazu podataka. Aktivne baze podataka uključuju pravila poznata i kao ECU(event-condition-action) pravila. Korištenjem takvih pravila možemo definirati određene događaje i pravila koja određuju koje će se akcije provesti, ukoliko se zadovolji određeno pravilo. Kod vođenja skladišta jedan od najbitnijih zahtjeva je vođenje evidencije zaliha artikala kroz vrijeme. Tu nam u igru dolaze temporalne baze podataka koje rade upravo to, pohranjuju informacije o stanjima iz stvarnog svijeta kroz vrijeme.

U ovom radu prikazati će se proces kreiranja stolne aplikacije koja služi za vođenje skladišta. Za bazu podataka odabrana je PostgreSQL baza podataka, a za kreiranje baze podataka, funkcija i okidača korišteni su pgAdmin i DBeaver. Za kreiranje stolne aplikacije odabrani je alat Microsoft Visual Studio 2019, a aplikacija je napravljena u C# programskom jeziku kao Windows Forms aplikacija. Komponente koje su dostupne su Windows Forms su vrlo jednostavni za korištenje i učinkoviti u vizualnom prikazu podataka.

2. Metode i tehnike rada

Za izradu aplikacije za vođenje skladišta koristiti će se ECU pravila u bazama podataka. Korištenjem ECU pravila baza podataka postaje aktivnom bazom podataka. Dok će se aktivne baze podataka brinuti za obavljanje određenih akcija temeljem događaja i uvjeta, temporalne baze podataka će voditi evidenciju zaliha artikala kroz vrijeme. Za prioritetizaciju narudžbi se koristi FIFO strategija, tako da one narudžbe koje su se prve naručile, prve će se i ulaziti u proces isporuke.

pgAdmin je platforma korištena za upravljanje i razvoj PostgreSQL. U ovom projektu korišten je gotovo isključivo samo za hostanje PostgreSQL poslužitelja. DBeaver je aplikacija za SQL klijent i alat za upravljanje bazama podataka i po meni je jako dobar alat za upravljanje bazama podataka. U DBeaver je kreirana cijela baza i sve funkcije i okidači. Alat također podržava i vizualizaciju baze, tako da je vrlo jednostavno kreirati ERA model baze podataka. Za kreiranje stolne aplikacije korišten je alat Microsoft Visual Studio 2019, a apliklacija je kreirana kao Windows Forms, pisano u C# programskom jeziku.

3. Baze podataka

3.1. Aktivne baze podataka

Aktivna baza podataka je baza podataka koja uključuje arhitekturu zasnovanu na događajima koja može odnositi na uvjete unutar i izvan baze podataka. U većini modernih relacijskih baza podataka, aktivnu komponentu baza podataka čine okidači ili aktivna pravila. Okidač je posebna procedura koja se izvodi kada se u bazi podataka dogodi određeni događaj. Okidači se mogu definirati na tablici, pogledu ili bazi podataka.[1] Pravila se u aktivnima bazama podataka najčešće sastoje od tri dijela: događaj, uvjet i akcija. Kada se dogodi određeni događaj, provjerava se uvjet i ako je istinit, provodi se radnja. Takva pravila su poznata kao ECA (event-condition-action) pravila. Prema tome, aktivne baze podataka možemo definirati kao baze podataka koje uključuju aktivna pravila najčešće u obliku ECA. Aktivne baze podataka obogaćuju tradicionalne baze podataka s mehanizmima za procesiranje pravila.[2]

3.2. Temporalne baze podataka

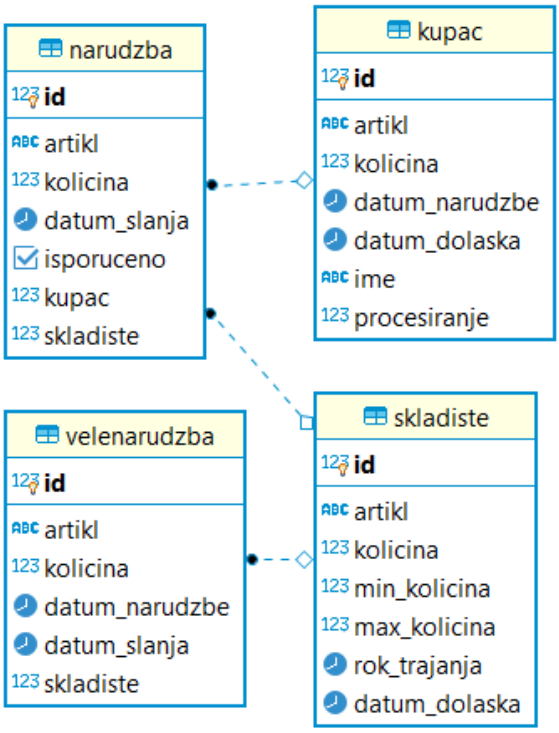
Temporalne baze podataka su one baze podataka koje pohranjuju informacije o stanjima iz stvarnog svijeta kroz vrijeme. Temporalne baze podataka u najširem smislu obuhvaćaju sve baze podataka za koje je potreban određeni aspekt vremena prilikom organiziranja njihovih podataka. Vremenski aspekti obično uključuju valjano vrijeme, vrijeme transakcije ili vrijeme odluke. Valjano vrijeme je skup vremenskih intervala tijekom kojih je činjenica istinita u stvarno svijetu. Vrijeme transakcije je vremenski interval tijekom kojeg je činjenica trenutno u sustavu baze podataka. Vrijeme odluke je vrijeme u kojem je donesena odluka o činjenici.[3]

3.3. Model baze podataka

Model i arhitektura baze podataka kreirana je u alatu DBeaver. DBeaver je aplikacija za SQL klijent i alat za upravljanje bazama podataka. Alat je jako intuitivan i svakako bih ga preporučio i drugima za korištenje. Pošto je projekt više orijentiran automatizaciji vođenja skladišta, model se sastoji od samo 4 relacije koje su dovoljne za prikaz koncepata rada, a atribute i veze između njih mogu se vidjeti na Slika 1. Model se sastoji od sljedećih relacija: kupac, narudžba, skladište i velenarudžba.

3.3.1. Kupac

Relacija kupac sastoji se od primarnog ključa id koji služi kao identifikator, u atribut artikl se pohranjuje naziv željenog artikla, a u količinu količina željenog artikla, datum_narudzbe definira kada je narudžba kreirana, a datum_dolaska kada je artikl isporučen tj. narudžba završena, u ime se pohranjuje naziv kupca, a atribut procesiranje nam koristi kao zastavica kako bi znali u kojoj se fazi isporuke artikla narudžba kupca trenutno nalazi.



Slika 1: ERA model (Izvor: rad autora, 2020)

3.3.2. Narudžba

Relacija narudzba sastoji se od primarnog ključa id koji služi kao identifikator, u atribut artikl se pohranjuje naziv naručenog artikla, a u količinu količina naručenog artikla, datum_slanja definira kada se narudžba šalje u proces realizacije iste, atribut isporuceno nam služi kao zastavica kako bi znali u kojoj se fazi isporuke artikla narudžba kupca trenutno nalazi (slično kao atribut procesiranje u relaciji kupac), a kupac i skladište su vanjski ključevi na relacije kupac i skladište kako bi znali koja narudžba se veže uz kojeg kupca i iz kojeg će se skladišta povlačiti artikli. Narudžba se kreira tako da se uzima prvi slog iz relacije kupac koji zadovoljava određena pravila.

3.3.3. Skladište

Relacija skladiste sastoji se od primarnog ključa id koji služi kao identifikator, u atribut artikl se pohranjuje naziv artikla koji se nalazi na skladištu, a u količinu trenutna količina artikla na skladištu, min_kolicina označava kolika je minimalna dopustiva količina artikla na skladištu, max_kolicina označava kolika je maksimalna dopustiva količina artikla na skladištu, rok_trajanja je rok valjanosti artikla na skladištu, a datum_dolaska označava kada je artikl stigao na skladište.

3.3.4. Velenarudžba

Relacija velenarudzba sastoji se od primarnog ključa id koji služi kao identifikator, u atribut artikl se pohranjuje naziv naručenog artikla, a u količinu količina naručenog artikla, datum_narudzbe definira kada je velenarudžba kreirana, datum_slanja kada je artikl isporučen tj. velenarudžba završena, a skladiste označava skladište koje je naručilo artikl i kojem se isti isporučuju.

4. Implementacija

U ovom će se poglavlju prikazati izvorni kod za kreiranje relacija(tablica) koje su prikazanje u modelu baze podataka, kreiranje funkcija i okidača, spajanje PostgreSQL baze podataka sa stolnom aplikacijom te komunikaciju između baze podataka i stolne aplikacije.

4.1. Baza podataka

4.1.1. Kreiranje tablica

4.1.1.1. Kupac

```
-- DROP TABLE public.kupac;

CREATE TABLE public.kupac (
        id serial NOT NULL,
        artikl varchar NULL,
        kolicina int4 NULL,
        datum_narudzbe timestamp NULL,
        datum_dolaska timestamp NULL,
        ime varchar NULL,
        procesiranje int4 NULL DEFAULT 0,
        CONSTRAINT kupac_pk PRIMARY KEY (id)
);
```

4.1.1.2. Narudžba

```
-- DROP TABLE public.narudzba;

CREATE TABLE public.narudzba (
        id serial NOT NULL,
        artikl varchar NULL,
        kolicina int4 NULL,
        datum_slanja timestamp NULL,
        isporuceno bool NULL DEFAULT false,
        kupac int4 NULL,
        skladiste int4 NULL,
        CONSTRAINT narudzba_pk PRIMARY KEY (id),
        CONSTRAINT narudzba_fk FOREIGN KEY (kupac) REFERENCES kupac(id),
        CONSTRAINT narudzba_fks FOREIGN KEY (skladiste) REFERENCES skladiste(id)
);
```

4.1.1.3. Skladište

```
-- DROP TABLE public.skladiste;

CREATE TABLE public.skladiste (
        id serial NOT NULL,
        artikl varchar NULL,
        kolicina int4 NULL,
```

```
        min_kolicina int4 NULL DEFAULT 3,
        max_kolicina int4 NULL,
        rok_trajanja date NULL,
        datum_dolaska timestamp NULL,
        CONSTRAINT skladiste_pk PRIMARY KEY (id)
);
```

4.1.1.4. Velenarudžba

```
-- DROP TABLE public.velenarudzba;

CREATE TABLE public.velenarudzba (
        id serial NOT NULL,
        artikl varchar NULL,
        kolicina int4 NULL,
        datum_narudzbe timestamp NULL,
        datum_slanja timestamp NULL,
        skladiste int4 NULL,
        CONSTRAINT velenarudzba_pk PRIMARY KEY (id),
        CONSTRAINT velenarudzba_fk FOREIGN KEY (skladiste) REFERENCES skladiste(id)
);
```

4.1.2. Funkcije i okidači

Pošto se aktivne baze podataka temelje na ECA pravilima, okidači su ono što ovu bazu podataka čini aktivnom bazom podataka i iz tog razloga su iznimno bitni u ovom projektu. U nastavku će se prikazati izvorni kod funkcija i okidača koji pozivaju te funkcije na određeni događaj.

4.1.2.1. Dodaj količinu postojećem artiklu

Prije svakog novog dodavanja u tablicu skladište, poziva se funkcija koja provjerava da li artikl koji dolazi na skladište već postoji. Ako postoji i ako dodavanjem nove količine količina ne prelazi maksimalnu količinu, zbrajaju se trenutna i nova količina. Ako postoji i ako se dodavanje nove količine prelazi maksimalna količina, diže se obavijest a količina se postavlja na maksimalnu. Ukoliko artikl već ne postoji na skladištu, kreira se i pohranjuje sa svim navedenim podacima.

```
create or replace
function public.dodaj_kolicinu_postojecem_artiklu() returns trigger language plpgsql
    as $function$
declare
        ukupno integer;
        stara_kolicina integer;
        maksi integer;

begin
select  count(*) into ukupno from skladiste
where artikl = new.artikl;

select  kolicina into stara_kolicina from skladiste
```

```
where artikl = new.artikl;

select  max_kolicina into maksi from skladiste
where artikl = new.artikl;

if ukupno > 0
and (stara_kolicina + new.kolicina) <= maksi then raise notice 'Postoji! Kolicina
    manja ili jednaka max';

update  skladiste set kolicina = stara_kolicina + new.kolicina
where artikl = new.artikl;
return null;

elsif ukupno > 0
and (stara_kolicina + new.kolicina) > maksi then raise notice 'Postoji! Kolicina
    veca od max';

update  skladiste set kolicina = maksi
where artikl = new.artikl;
return null;
else raise notice 'NE postoji!  %',
new.artikl;
return new;
end if;
end;
$function$ ;

-- DROP TRIGGER dodaj_kolicinu_artiklu ON public.skladiste;
create trigger dodaj_kolicinu_artiklu before
insert on public.skladiste for each row execute function
    dodaj_kolicinu_postojecem_artiklu();
```

4.1.2.2. Dohvati najprioritetniju narudžbu

Nakon dodavanja novog zapisa u tablicu kupac, dohvaća se prvi slog kojem je datum dolaska NULL i dodaje se u tablicu narudzba. Na ovaj se način poštuju pravila FIFO strategije.

```
CREATE OR REPLACE FUNCTION public.dohvati_najprioritetniju_narudzbu()
 RETURNS trigger LANGUAGE plpgsql AS $function$
 begin
raise notice 'Dohvati';

insert  into narudzba(artikl,kolicina, datum_slanja, kupac)
select  artikl, kolicina,datum_narudzbe,id from kupac
where   datum_dolaska is null
order by datum_narudzbe fetch first 1 rows only;

return null;
end;
$function$
;
```

```
-- DROP TRIGGER create_narudzba ON public.kupac;
create trigger create_narudzba after
insert on public.kupac for each row execute function
    dohvati_najprioritetniju_narudzbu();
```

4.1.2.3. Dohvati minimalnu količinu artikla

Prije unosa u tablicu velenarudzba poziva se funkcija dostavi_min_kolicinu_artikla(). Funkcija kreira temporalnu tablicu koja se sastoji od artikl, kolicina, datum_narudzbe, datum_slanja, skladiste iz tablice velenarudzba gdje datum slanja nije NULL, sortira prema datum_narudzbe i uzima prvi redak. Ažurira se tablica velenarudzba tako da se datum_slanja postavi kao trenutni, a isto tako se ažurira i tablica skladiste gdje se nova količina dodaje trenutnoj količini artikla na skladištu.

```
CREATE OR REPLACE FUNCTION public.dostavi_min_kolicinu_artikla()
 RETURNS trigger LANGUAGE plpgsqlAS $function$
declare kolicina_artikla integer;
declare broj_narudzbe integer;
declare datumnarudzbe timestamp;
declare ime varchar;

begin
drop table if exists temporalna;
create temp table temporalna as
select artikl, kolicina,datum_narudzbe, datum_slanja, skladiste from velenarudzba
where not (datum_slanja is not null)
order by datum_narudzbe asc fetch first 1 rows only;

select  kolicina from temporalna
into    kolicina_artikla ;

select skladiste from temporalna
into    broj_narudzbe;

select artikl from temporalna
into    ime;

select  datum_narudzbe from     temporalna
into datumnarudzbe;

raise notice 'Dostavljeno␣%␣%.', kolicina_artikla, ime;

update  velenarudzba set datum_slanja = now()
where artikl = ime and datum_narudzbe = datumnarudzbe;

update  skladiste set kolicina = kolicina + kolicina_artikla
where skladiste.id = broj_narudzbe;
return null;
end;
$function$
;
```

```
-- DROP TRIGGER dostavi_kolicinu_skladistu ON public.velenarudzba;
create trigger dostavi_kolicinu_skladistu after
insert on public.velenarudzba for each row execute function
    dostavi_min_kolicinu_artikla();
```

4.1.2.4. Isporuka narudžbe

Nakon unosa u tablicu narudzba poziva se funckcija isporuka_narudzbe(). Ukoliko je polje isporuceno u tablici narudzba jednako true, tada promjeni procesiranje na false u tablci kupac.

```
CREATE OR REPLACE FUNCTION public.isporuka_narudzbe()
 RETURNS trigger
 LANGUAGE plpgsql
AS $function$ begin

update kupac set procesiranje = 1
from narudzba
where narudzba.kupac = kupac.id and narudzba.isporuceno = true and kupac.
    datum_dolaska is null;

raise notice 'Narudzba isporucena';
return null;
end;
$function$
;

-- DROP TRIGGER isporuci_narudzbu ON public.narudzba;
create trigger isporuci_narudzbu after
insert on public.narudzba for each row execute function isporuka_narudzbe();
```

4.1.2.5. Količina na skladištu

Nakon unosa u tablicu narudzba poziva se funckcija kolicina_na_skladistu(). Kreira se temporalna tablica u kojoj se nalaze id skladišta, naziv artikla, naručena količina iz tablice narudzba i trenutna količina artikla iz tablice skladište. Ako je naručena količina manja ili jednaka trenutnoj količini, kreira se narudžba. Ažurira se tablica skladiste i trenutna količina se smanjuje za količinu naručenog artikla. Ažurira se i tablica narudzba gdje se isporuceno postavlja na true, a u skladiste se pohranjuje id skladišta iz kojeg dolazi roba. Ukoliko je pak naručena količina artikla veća od trenutne količine artikla na skladištu, kreira se velenarudžba. U novoj velenarudžbi naručuje se artikl u naručenoj količini, tj. koliko je robe naručeno, toliko se i naručuje u velenarudžbi.

```
CREATE OR REPLACE FUNCTION public.kolicina_na_skladistu()
 RETURNS trigger
 LANGUAGE plpgsql
AS $function$
declare narucena_kol integer;
```

```
declare trenutna_kol integer;
declare minimalna integer;
declare artiklN varchar;
declare skladiste_id integer;
begin
drop table if exists privremena;
create temp table privremena as
select skladiste.id, narudzba.artikl , narudzba.kolicina as narucena , skladiste.
    kolicina as trenutna from skladiste,   narudzba
where skladiste.artikl = narudzba.artikl and narudzba.isporuceno = false
order by narudzba.datum_slanja fetch first 1 rows only;

select narucena from privremena into narucena_kol;
select trenutna from privremena into trenutna_kol;
select min_kolicina from skladiste into minimalna;
select artikl from privremena into artiklN;
select id from  privremena into skladiste_id;

if(narucena_kol <= trenutna_kol) then update skladiste set kolicina = trenutna_kol-
    narucena_kol where artikl = artiklN;
update narudzba set     isporuceno = true, skladiste = skladiste_id where artikl =
    artiklN      and kolicina = narucena_kol;
raise notice 'Kreirana narudzba';
end if;

if narucena_kol > trenutna_kol then insert into         velenarudzba(artikl ,
    kolicina , skladiste ) values (artiklN,narucena_kol,skladiste_id);

raise notice 'Manjak kolicine narucene robe. Kreirana velenarudzba. Naruceno % %.',
    narucena_kol, artiklN;
end if;
return null;
end;
$function$
;

-- DROP TRIGGER insert_provjera_kolicine ON public.narudzba;

create trigger insert_provjera_kolicine after
insert on public.narudzba for each row execute function kolicina_na_skladistu();
```

4.1.2.6. Minimalna količina

Nakon promjene na tablici skladiste poziva se funkcija min_kolicina(). Ukoliko je trenutna količina artikla na skladištu manja od minimalne količine, kreira se velenarudžba i naruči određeni artikl (maksimalna količina - trenutna).

```
CREATE OR REPLACE FUNCTION public.min_kolicina()
 RETURNS trigger
 LANGUAGE plpgsql
AS $function$
declare minimalna integer;
```

```
declare trenutna integer;
declare ime varchar;
declare id_artikla integer;
begin
drop table if exists temporalna;
create temp table temporalna as
select  id,     artikl, kolicina, min_kolicina from     skladiste
where kolicina <min_kolicina order by id asc fetch first 1 rows only;
select kolicina from temporalna into trenutna;
select  min_kolicina from temporalna into minimalna;
select artikl from temporalna into ime;
select id from temporalna into id_artikla;

if trenutna<minimalna then
insert into     velenarudzba (id, artikl, kolicina,     datum_narudzbe ,
    datum_slanja , skladiste ) values(default,
ime, minimalna, now(), null, id_artikla);

raise notice 'Naruceno␣%␣%.', minimalna, ime;
end if;
return null;
end;
$function$
;

-- DROP TRIGGER update_kolicina ON public.skladiste;
create trigger update_kolicina after
update on public.skladiste for each row execute function min_kolicina();
```

4.1.2.7. Postavi datum i vrijeme

Prije unosa novog sloga u tablicu kupac i velenarudzba u atribut datum_narudzbe postavlja se trenutni datum i vrijeme.

```
CREATE OR REPLACE FUNCTION public.postavi_datum_i_vrijeme()
 RETURNS trigger
 LANGUAGE plpgsql
AS $function$
        BEGIN
        NEW.datum_narudzbe = NOW();
    raise notice 'Datum␣je␣%.', now();
        RETURN NEW;
        END;
$function$
;

-- DROP TRIGGER insert_datum ON public.velenarudzba;
create trigger insert_datum before
insert on public.velenarudzba for each row execute function postavi_datum_i_vrijeme
    ();

-- DROP TRIGGER insert_kupac_datum ON public.kupac;
```

```
create trigger insert_kupac_datum before
insert on public.kupac for each row execute function postavi_datum_i_vrijeme();
```

4.1.2.8. Zatvori narudžbu

Nakon promjene na tablici narudzba poziva se funckija zatvori_narudzbu(). Odabire se id kupca kojemu je kupac.artikl = narudzba.artikl, kupac.kolicina = narudzba.kolicina, narudzba.skladiste nije null, kupac.procesiranje = 1 i kupac.datum_dolaska je jednak null. Prema tom id-u kupca ažurira se redak u tablici kupac i postavlja se procesiranje na 2, a datum_dolaska na trenutno vrijeme i datum.

```
CREATE OR REPLACE FUNCTION public.zatvori_narudzbu()
 RETURNS trigger
 LANGUAGE plpgsql
AS $function$
declare sifra_kupca integer;
begin
select kupac.id from kupac,     narudzba into sifra_kupca
where kupac.artikl = narudzba.artikl
and kupac.kolicina = narudzba.kolicina
and narudzba.skladiste is not null
and kupac.procesiranje = 1
and kupac.datum_dolaska is null
order by kupac.datum_narudzbe fetch first 1 rows only;
update  kupac set procesiranje = 2 , datum_dolaska = now()
where id = sifra_kupca;

raise notice 'Narudzba zatvorena %', sifra_kupca;
return null;
end;
$function$
;

-- DROP TRIGGER zatvori_narudzbu ON public.narudzba;
create trigger zatvori_narudzbu after
update on public.narudzba for each row execute function zatvori_narudzbu();
```

4.1.3. Povezivanje PosgreSQL baze podataka i aplikacije

Za povezivanje PostreSQL baze podataka i stolne aplikacije koja je rađena u C#, koristio se Npgsql library. Metoda GetConection nam vraća NpgsqlConnection preko koje komuniciramo sa bazom podataka. ExecuteSQL je metoda koju koristimo za izvršavanje upita nad bazom podataka.

```
using Npgsql;

        public static NpgsqlConnection GetConenction()
        {
            string server, username, password, database;
            int port;
```

```
    server = "127.0.1.1";
    port = 5432;
    username = "postgres";
    password = "admin";
    database = "postgres";

    string connString = String.Format("Server={0};Port={1};" +
           "User␣Id={2};Password={3};Database={4};",
           server, port, username, password, database);

    NpgsqlConnection connection = new NpgsqlConnection(connString);
    connection.Open();

    return connection;
}

public static void ExecuteSQL(string query, NpgsqlConnection con)
{
    var cmd = new NpgsqlCommand();
    cmd.Connection = con;

    cmd.CommandText = query;
    cmd.ExecuteNonQuery();
}
```

4.1.4. Komunikacija PosgreSQL baze podataka i aplikacije

4.1.4.1. Dodavanje zapisa u bazu

Kada smo povezali stolnu aplikaciju i PostreSQL bazu podataka, napokon možemo vršiti upite na bazu putem aplikacije. U sljedećem bloku koda prikazan je unos novih podataka u tablicu kupac. Metoda insertKupacRand podatke preuzima iz nedefiniranih lista podataka i metoda koje vraćaju količine, dok insertKupac dodaje nove podatke koji su mu proslijeđeni kao parametri.

```
public static void insertKupacRand(NpgsqlConnection con)
{
    var cmd = new NpgsqlCommand();
    cmd.Connection = con;

    string sql = "INSERT␣INTO␣kupac(id,␣ime,␣artikl,␣kolicina)␣" +
                 "VALUES(default,␣'" + Data.getName() + "',␣'" + Data.
                     getItem() + "',␣" + Data.getQuantity() + ")";

    Console.WriteLine(sql);
    infoMessage = sql;
    try
    {
        Database.ExecuteSQL(sql, con);
    }
```

```
        catch (NpgsqlException npqe)
        {
            MessageBox.Show(npqe + "\nError␣on␣query:" + sql, "ERROR");
        }
        Thread.Sleep(delay);
    }

    public static void insertKupac(NpgsqlConnection con, string ime, string
        artikl, int kolicina)
    {
        var cmd = new NpgsqlCommand();
        cmd.Connection = con;

        string sql = "INSERT␣INTO␣kupac(id,␣ime,␣artikl,␣kolicina)␣" +
                     "VALUES(default,␣'" + ime + "',␣'" + artikl+ "',␣" +
                         kolicina + ")";

        Console.WriteLine(sql);
        infoMessage = sql;
        try
        {
            Database.ExecuteSQL(sql, con);
        }
        catch (NpgsqlException npqe)
        {
            MessageBox.Show(npqe + "\nError␣on␣query:" + sql, "ERROR");
        }
        Thread.Sleep(delay);
    }
```

4.1.4.2. Ažuriranje zapisa u bazi

Kako bismo ažurirali određeni slog u tablici, moramo znati koji je to slog. Prilikom odabiranja sloga za ažuriranje, spremamo njegov identifikator(id) kojega kasnije šaljemo metodi kao parametar prema kojem tražimo slog za ažuriranje. Metodi se kao parametri šalju nove vrijednosti atributa određenog sloga. U sljedećem bloku koda prikazano je ažuriranje zapisa u tablici kupac.

```
public static void updateKupac(NpgsqlConnection con, string ime, string artikl,
    int kolicina, int id)
    {
        var cmd = new NpgsqlCommand();
        cmd.Connection = con;

        string sql = "UPDATE␣kupac␣set␣ime='" + ime + "',␣artikl='" + artikl + "
            ',␣kolicina=" + kolicina + "␣where␣id=" + id;

        Console.WriteLine(sql);
        infoMessage = sql;
        try
        {
```

```
            Database.ExecuteSQL(sql, con);
        }
        catch (NpgsqlException npqe)
        {
            MessageBox.Show(npqe + "\nError␣on␣query:" + sql, "ERROR");
        }
        Thread.Sleep(delay);
    }
```

4.1.4.3. Brisanje zapisa iz baze

Kao što je zapise potrebno dodavati i ažurirati u bazi podataka, te iste zapise je ponekad potrebno i brisati. Kako bi omogućili brisanje zapisa uz baze preko stolne aplikacije, potrebno je za to napisati metode. Metode u narednom bloku koda prikazuju kako se brišu podaci iz baze podataka na primjeru tablice kupac. Metoda deleteKupacAll je metoda koja će obrisati sve zapise u tablici kupac. Metodi deleteKupac moramo proslijediti identifikator kupca kojeg želimo obrisati.

```
public static void deleteKupacAll(NpgsqlConnection con)
    {
        var cmd = new NpgsqlCommand();
        cmd.Connection = con;

        string sql = "DELETE␣FROM␣kupac";

        try
        {
            Database.ExecuteSQL(sql, con);
        }
        catch (NpgsqlException npqe)
        {
            MessageBox.Show(npqe + "\nError␣on␣query:" + sql, "ERROR");
        }
        Thread.Sleep(delay);
    }

    public static void deleteKupac(NpgsqlConnection con, int id)
    {
        var cmd = new NpgsqlCommand();
        cmd.Connection = con;

        string sql = "DELETE␣FROM␣kupac␣where␣id=" +id;
        infoMessage = sql;
        try
        {
            Database.ExecuteSQL(sql, con);
        }
        catch (NpgsqlException npqe)
        {
            MessageBox.Show(npqe + "\nError␣on␣query:" + sql, "ERROR");
        }
```

```
        Thread.Sleep(delay);
    }
```

4.1.4.4. Odabir zapisa iz baze

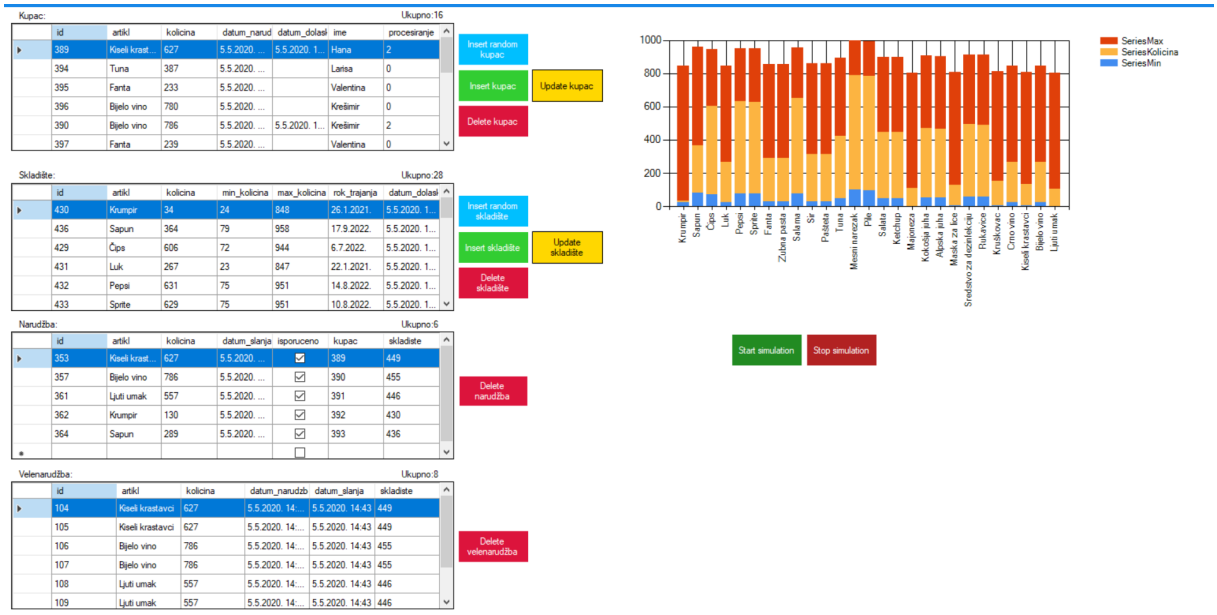
Kako aplikacija ne bi bila prazna, u njoj moramo prikazati određene podatke. Podaci će se prikazivati u DataGridView komponenti. DataGridView prikazuje podatke u tabličnom obliku kojeg je moguće prilagoditi. Podatke iz baze podataka dobivamo jednostavnim SELECT upitom. NpgsqlDataAdapter je adapter za sve tipove upita(select, insert, update, delete) za popunjavanje Dataset-a. Funkcijom Fill() dodajemo retke adaptera u DataSet. Iz DataSet-a uzimamo tablicu tipa DataTable koju vežemo kao izvor podataka za naš DataGridView. Sada će se svi retci koji su zapisani u tablici kupac na bazi podataka prikazati u DataGridView-u u našoj stolnoj aplikaciji.

```
public void showKupac()
{
    DataSet ds = new DataSet();
    DataTable dt = new DataTable();
    NpgsqlConnection con = Database.GetConenction();

    string query = "SELECT␣*␣FROM␣kupac";
    NpgsqlDataAdapter da = new NpgsqlDataAdapter(query, con);

    ds.Reset();
    da.Fill(ds);
    dt = ds.Tables[0];
    dgvKupac.DataSource = dt;
    con.Close();

    outputCountKupac.Text = "Ukupno:" + dgvKupac.Rows.Count.ToString();
}
```

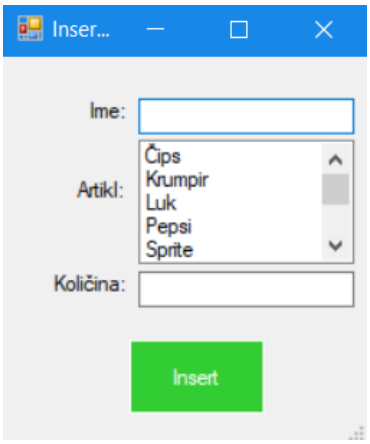
4.1.5. Stolna aplikacija

Stolna aplikacija rađena je u Microsoft Visual Studio 2019, u programskom jeziku C# kao Windows Forms aplikacija.Izgled glavnog ekrana aplikacije prikazan je na Slika2 . Trenutno stanje količine pojedinog artikla na skladištu vizualno je reprezentirano na grafu. Plavom bojom je označena minimalna dopuštena količina artikla na skladištu, žutom bojom je označeno kolika je trenutna zaliha određenog artikla na skladištu, a crvena boja označava koliko zaliha artikla fali kako bi se ispunio maksimalni kapacitet količine artikla na skladištu.

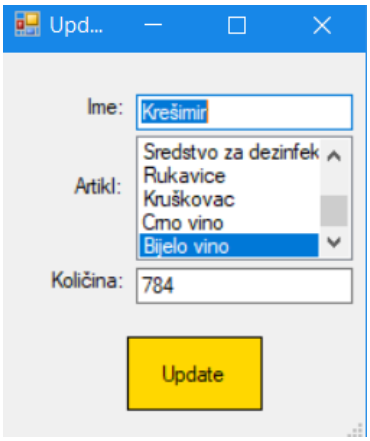
U aplikaciji, korisniku je dopušteno unošenje novih redaka u tablice, uređivanje i brisanje postojećih putem formi. Forma za unos i ažuriranje podataka je ista forma koja mijenja svoj izgled ovisno o odabranoj opciji(insert ili update), a forme su prikazane na Slika 3 i Slika 4. Korisniku je također omogućeno dodavanje novog kupca i skladišta putem gumba Insert random kupac ili Insert random skladiste. Klikom na taj gumb, dodaju se novi retci u tablicu kupac ili skladiste za nasumičnim predefiniranim podacima.



Slika 2: Glavni ekran stolne aplikacije (Izvor: rad autora, 2020)



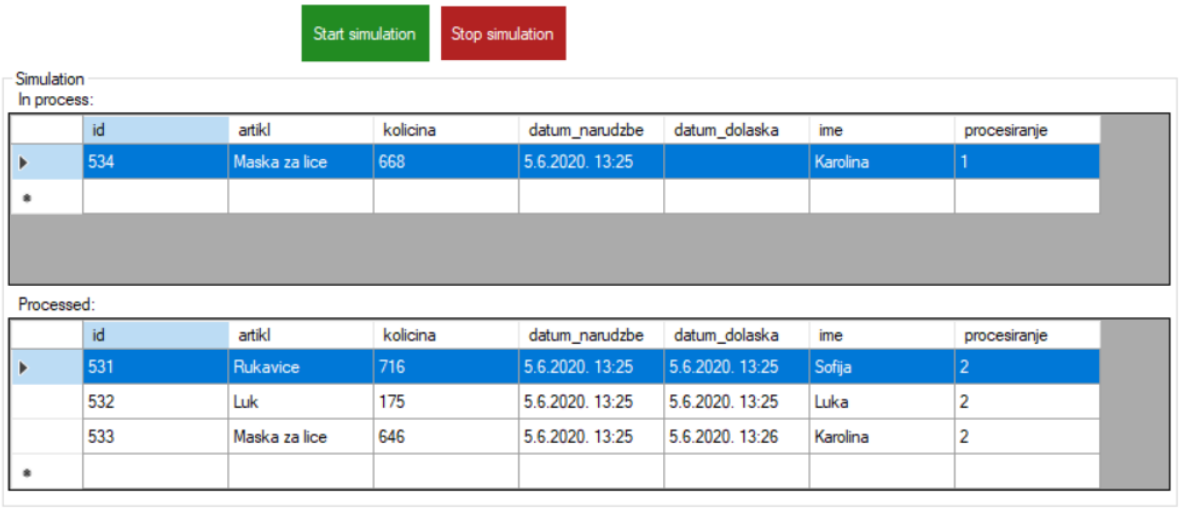
Slika 3: Forma za unos novog kupca (Izvor: rad autora, 2020)



Slika 4: Forma za ažuriranje odabranoga kupca (Izvor: rad autora, 2020)

Još jedna od opcija aplikacije je simulacija rada skladišta. Skladište se automatski po-

puni sa svim artiklima sa definiranim minimalnim, maksimalnim i trenutnim količinama. Svake sekunde se kreira novi zapis u tablicu kupac, koji pokreće okidač. Okidači, koji su opisani u prethodnim odlomcima, dalje sami upravljaju naredbama.Narudžba koja se trenutno nalazi u procesu dostave prikazana je u tablici sa svim svojim atributima, dok su isporučene narudžbe prikazane u zasebnoj tablici. Podaci se mogu pratiti u trenutnom vremenu izvršavanja i u DataGridView-ima na zaslonu aplikacije kako i se vidjelo kako se kreiraju narudžbe, velena-rudžbe, mijenja stanje proizvoda i slično. Grupa funkcionalnosti za simulaciju prikazana je na Slika5.



Slika 5: Grupa funkcionalnosti za simulaciju (Izvor: rad autora, 2020)

5. Zaključak

Aktivne baze podataka pokazale su se kao jedan vrlo elegantan način upravljanja i automatizacije događaja u bazi podataka. Aktivne su baze podataka je teže za održavati zbog složenosti koja nastaje razumijevanjem učinka svih okidača i dosta se lako pogubiti koji će se događaj kada okinuti. Iako bi se svi događaji i uvjeti koje obuhvaćaju ECU pravila mogli rukovati i direktno u aplikacijskom dijelu, ovakvo riješenje je dosta lakše što se tiče procesne snage na korisničkoj strani.

DBeaver se pokazao kao jako jednostavan i intuitivan alat za upravljanje baza podataka. Pregledan smještaj alata na ekranu i logična kategorizacija uvelike olakšavaju rad u alatu. pgAdmin je korišten isključivo samo za hostanje PostgreSQL poslužitelja pošto mi se nije pretjerano svidio način kreiranja tablica i funkcija, a i manjka mu vizualna reprezentacija baze podataka. Kreiranje stolne aplikacije u Microsoft Visual Studio 2019 u C# jeziku kao Windows Forms aplikacija rezultirala je jednostavnim dizajnom i pregledom svih potrebnih elemenata kako bi se dokazali i pokazali sve metode i tehnike korištene u ovom projektu.

Popis literature

[1] M. Mishra, „Active database”, adresa: https://www.slideshare.net/mridulmishra2/active-database (pogledano 15. 6. 2020).

[2] M. Schatten, „Aktivne baze podataka”, adresa: https://files.classcraft.com/game/uploads/zijp9vaaGwK86piNu/1571991887636/tbp2019-003-ABP.pdf (pogledano 15. 6. 2020).

[3] S. Mahara Dabbal, „Temporal databases”, adresa: https://www.slideshare.net/DabalMahara1/temporal-databases-97200226 (pogledano 15. 6. 2020).

Popis slika