Oblikovanje baze podataka za praćenje zaliha u skladištu sustavom PostgreSQL

Kahriman, Emilija

Undergraduate thesis / Završni rad

2015

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University North / Sveučilište Sjever**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:122:189638

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-11-30**

Repository / Repozitorij:

University North Digital Repository

Sveučilište
Sjever

Završni rad br. 255/TGL/2015

Oblikovanje baze podataka za praćenje zaliha u skladištu sustavom PostgreSQL

Emilija Kahriman, 4628/601

Varaždin, rujan 2015. godine

Sveučilište
Sjever

Odjel za tehničku i gospodarsku logistiku

Završni rad br. 255/TGL/2015

Oblikovanje baze podataka za praćenje zaliha u skladištu sustavom PostgreSQL

Student

Emilija Kahriman, 4628/601

Mentor

dr.sc. Ladislav Havaš, dipl.ing.

Varaždin, rujan 2015. godine

PREDGOVOR

Zadatak završnog rada pod naslovom "Oblikovanje baze podataka za praćenje zaliha u skladištu sustavom PostgreSQL" nastao je u dogovoru s mentorom. Za ovu temu sam se odlučila jer je usko vezana uz dva područja koja me najviše zanimaju, a to su logistika i informatika. Iako sam iz kolegija "Baze podataka" koji sam uspješno položila na prvoj godini studija mnogo naučila, svoje znanje iz područja programiranja i izgradnje aplikacije proširila sam proučavajući dodatnu literaturu dostupnu na internetu.

Zahvaljujem se mentoru Ladislavu Havašu koji je pratio cijeli proces nastajanja završnog rada i koji mi je puno pomogao svojim znanjem i savjetima. Zahvaljujem se i ostalim profesorima koji su svoje znanje prenijeli na mene. Posebno bih se zahvalila svojem kolegi Anti Adžagi koji mi je svojim velikim znanjem iz područja informatike pomogao pri stvaranju aplikacije i koji mi je čitavo vrijeme bio podrška. Također bih spomenula svoje kolege koji su mi uvijek bili spremni pomoći kada je trebalo i koji su mi studentske dane učinili puno zabavnijima i ljepšima.

SAŽETAK

Cilj izrade ovog rada je prikaz izrade i primjene sustava za praćenje zaliha u skladištu nekog poduzeća. U radu je ukratko opisana povijest SQL-a koji je najkorišteniji jezik za rad bazama podataka te je detaljno opisan PostgreSQL sustav koji je korišten za izradu programskog rješenja. U radu se također detaljno opisuju koraci u izgradnji baze podataka koja služi kao potpora radu aplikacije za praćenje zaliha u skladištu kreirane u alatu Microsoft Visual Studio. Također je prikazan rad aplikacije te su opisane njezine mogućnosti. Na temelju opisanih koraka oblikovanja baze podataka i aplikacije kreiran je odgovarajući sadržaj i napisan tekst završnog rada.

Ključne riječi:

- Baza podataka
- ER model
- PostgreSQL
- Microsoft Visual Studio

Popis korištenih kratica

SQL - Structured Query Language

DBMS - Database Managment System

ER - Entity - relationship

QBE - Query By Example

IBM - International Business Machines

ANSI - American National Standradrs Institute

ISO - International Organization for Standardization

DDL - Data Definition Language

PSQL - Postgre SQL

DML - Data Manipulation Language

IEEE - Institute of Electrical and Electronics Engineers

IDE - Integrated development environment

CIL - Common Intermediate Language

CLR - Common Language Runtime

ODBC - Open Database Conectivity

SADRŽAJ

1. UVOD 1

2. RAZVOJNI CIKLUS BAZA PODATAKA 3

2.1. Konceptualno oblikovanje baza podataka 4

2.1.1. ER dijagram (engl. Entity – relationship) 4

2.1.2. Entiteti, atributi i veze 6

2.1.3. Entiteti i njihovi atributi 7

2.1.4. Veze i njihovi atributi 7

2.1.5. Funkcionalnost veze, obaveznost članstva, kardinalnost 8

2.1.6. Koraci u oblikovanju entiteta, veza i atributa 8

2.2. Relacijski model – logičko oblikovanje baza podataka 9

2.2.1. Općenito o relacijskom modelu 9

2.3. Tipovi podataka 11

2.3.1. Numerički tipovi 12

2.3.2. Znakovni tipovi 13

2.3.3. Datum i vrijeme 14

2.3.4. Ostali tipovi podataka 14

2.3.5. Null znak 14

3. OPIS I IZGRADNJA SUSTAVA 17

3.1. Oblikovanje baze podataka 17

3.1.1. Create table 18

3.1.2. Primarni ključ 21

3.1.3. Vanjski ključ 22

3.1.4. Naredbe select i insert 22

3.1.5. Naredba alter table 25

3.1.6. Klauzule from, where, order by 26

3.1.7. Klauzule limit, offset i dinstict 27

3.2. Izgradnja aplikacije 28

3.2.1. Rad u alatu Microsoft Visual Studio 29

4. SLOŽENI UPITI 44

5. USPOREDBA RAZVIJENOG SUSTAVA S POSTOJEĆIM RJEŠENJIMA NA TRŽIŠTU 46

6. ZAKLJUČAK 48

1. UVOD

Informacijski sustav je podsustav nekog poslovnog sustava i služi mu kao podrška pri izvođenju svakodnevnih poslovnih procesa. Informacijski sustavi mogu tvrtkama olakšati poslovanje te ubrzati poslovni proces, a oni napredniji pomažu i menadžmentu u donošenju važnih odluka. Dobro osmišljeni i implementirani informacijski sustavi iziskuju izdvajanje velikih novčanih sredstava, a sama implementacija sustava nije jednostavan zadatak. Osim cijene, izrada informacijskog sustava nosi sa sobom i rizik od neuspjeha te eventualno nanošenje nepopravljive štete. Ukoliko bi poduzeće uložilo velike količine novca u izradu informacijskog sustava, a sustav bude spor, neefikasan ili težak za korištenje, poduzeće će pretrpjeti veliku štetu. Zbog toga se informacijski sustavi izgrađuju, odnosno nadograđuju jedino u slučaju kada je dobit veća od troška izrade pa i samoga rizika. U ovome radu opisat će se izrada jednog dijela informacijskog sustava za praćenje zaliha u skladištu u fiktivnoj tvrtki Juicy d.o.o. Dio koji će biti implementiran sastojat će se od baze podataka koja je osnova informacijskog sustava te jednostavne aplikacije koja će raditi nad samom bazom. U teorijskom dijelu rada govori se općenito o tome što su baze podataka, zatim o sustavu za upravljanje bazama podataka PostgreSQL, opis samog sustava i funkcije koje će se koristiti i koje se mogu koristiti. Tijekom opisivanja samoga sustava, korak po korak će se implementirati potrebne stavke za rad baze i aplikacije. Praktični dio rada uključuje fizičku implementaciju baze podataka na lokalno računalo u sustavu PostgreSQL te izrada jednostavne aplikacije u alatu Microsoft Visual Studio. SQL (engl. *Structured Query Language*) je najkorišteniji jezik za rad s bazama podataka koji omogućuje kreiranje, brisanje te dohvaćanje podataka iz relacija (u daljnjem tekstu koristit će se riječ tablica), odnosno baze. Baza podataka je zapravo skup povezanih tablica gdje svaka tablica predstavlja jedan entitet o kojem se govori, npr. studenti, zaposlenici, profesori i slično. Ključno je kod baza podataka da ne postoji redundancija, odnosno da se podaci ne ponavljaju u više od jedne tablice. Kako bi se moglo raditi sa bazom podataka, nužno je poznavanje SQL-a koji je postao standard u svijetu kod korištenja baza podataka.

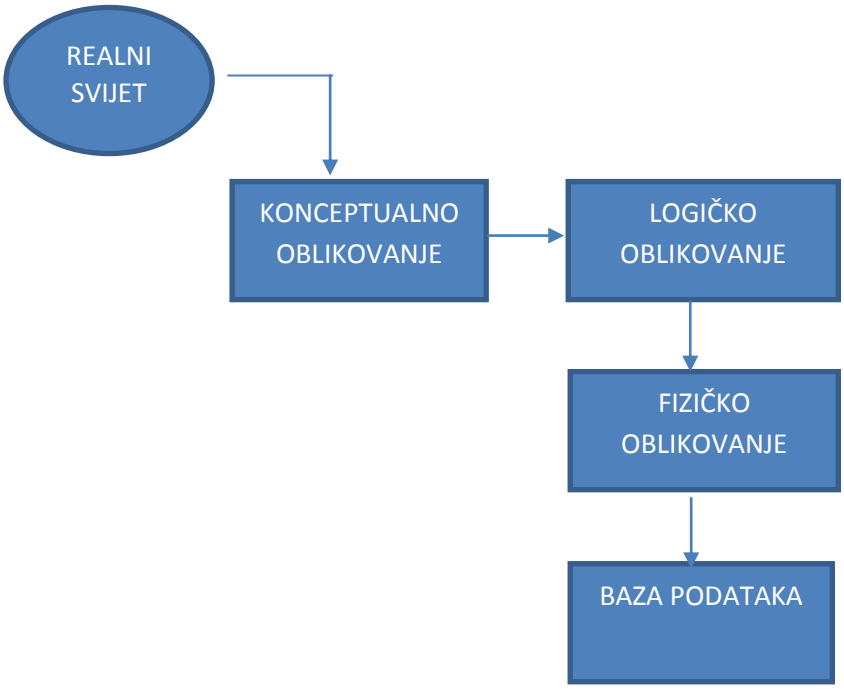
„Pod pojmom baze podataka danas se podrazumijeva kompjuterizirana baza podataka. Podaci su smješteni na disku u obliku koji nije razumljiv krajnjem korisniku te se za korištenje tim podacima koristi neki od sustava za upravljanje bazom podataka (DBMS).“[1] Sustavi za upravljanje bazama podataka su proizvodi koji omogućuju većem broju korisnika

manipuliranje podacima, upisivanje podataka, brisanje i slično, ovisno o pravima koja su im dodijeljena za rad nad tom bazom. Danas su relacijske baze podataka najkorištenije i najpopularnije baze podataka. Kako je osnova takve baze tablica, u bazi će se pronaći veći broj međusobno povezanih tablica. U prošlosti su se koristili mrežni modeli gdje je baza bila predočena mrežom koja se sastoji od čvorova i lukova, hijerarhijski modeli gdje je baza predočena stablom ili više njih, no ti modeli nisu više toliko u upotrebi. Kako bi povezali dvije tablice, koristimo redundantne podatke, odnosno primarne i vanjske ključeve. Primarni ključ tablice je stupac koji jednoznačno određuje vrijednost nekog atributa, npr. kod neke tablice studenata, prikladan ključ bio bi JMBAG studenta jer je on različit za svakog studenta. Datum rođenja bio bi primjer lošeg primarnog ključa jer postoji više osoba koji su rođeni istog dana te se javljaju kolizije kod baze. SQL se sastoji od određenog broja naredbi, a mogu se podijeliti u nekoliko skupina, to su *Data Definition Language*, odnosno naredbe za kreiranje objekata. U ovu vrstu pripadaju naredbe *create, alter i drop*. Zatim postoje *Data Manipulation Language* naredbe u koje pripadaju *insert, update i delete*. Zatim *Data Query Language* u koju pripada naredba *select*. Također se negdje spominju i druge grupe naredbi, kao, npr. *Data Control Language*, no nije toliko važno koliko ima grupa naredbi, koliko je važno shvatiti način rada pojedine naredbe.

2. RAZVOJNI CIKLUS BAZA PODATAKA

Kako bi se kreirala neka baza podataka koja će biti podrška pri radu nekog sustava, potreban je timski rad, niz stručnjaka te primjena alata i metoda potrebnih za samo kreiranje baze. Projekt izrade baze podataka se može podijeliti u nekoliko aktivnosti, a to su: utvrđivanje i analiza zahtjeva, projektiranje, implementacija, testiranje i na kraju održavanje. Kako bi se utvrdili zahtjevi nekog sustava potrebno je proučiti sve informacijske tokove i materijale te uočiti veze među njima. Također se provodi razgovor sa korisnicima kako bi se pobliže sve moglo shvatiti. Nakon što se utvrde zahtjevi za bazu, slijedi oblikovanje baze. Cilj oblikovanja je kreirati bazu u skladu sa zahtjevima koji su definirani u prethodnom koraku. Oblikovanje baze podataka predlaže način na koji se podaci grupiraju, povezuju i strukturiraju. Samo oblikovanje je dosta složen proces te zahtjeva također neke standardne faze kreiranja baze, a to su: konceptualno oblikovanje, logičko oblikovanje te fizičko oblikovanje.

Implementacija je sljedeći korak pri izradi baze podataka, a sastoji se od fizičke realizacije oblikovane baze podataka. Pri korištenju sustava za upravljanje podataka pokreću se naredbe, ovisno o jeziku koji se koristi u nekom sustavu za upravljanje bazama podataka. Nakon što su tablice kreirane i povezane, slijedi popunjavanje tablica podacima.

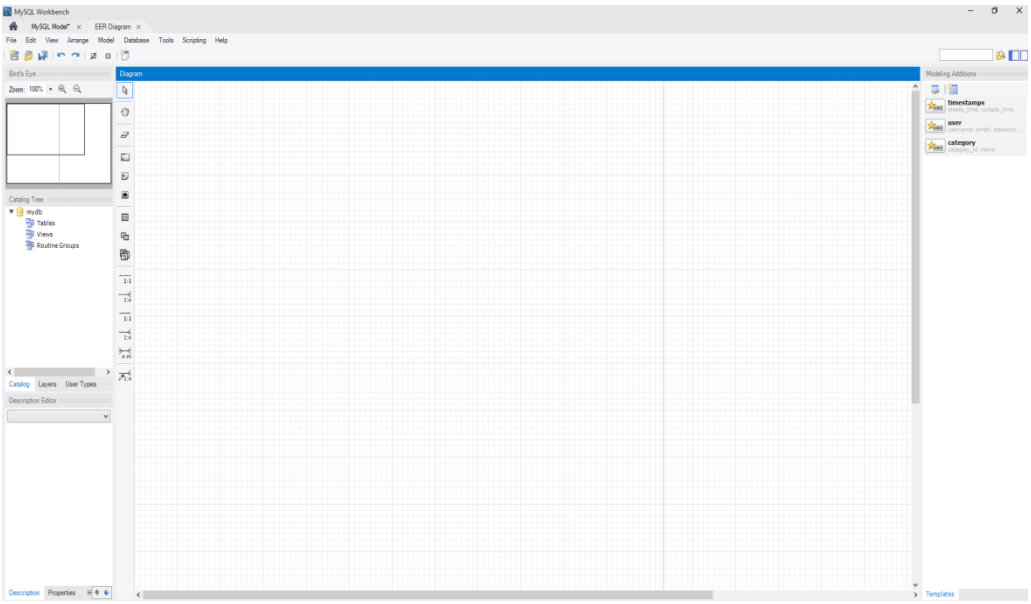


Slika 2.1. *Oblikovanje baze podataka*

2.1. Konceptualno oblikovanje baza podataka

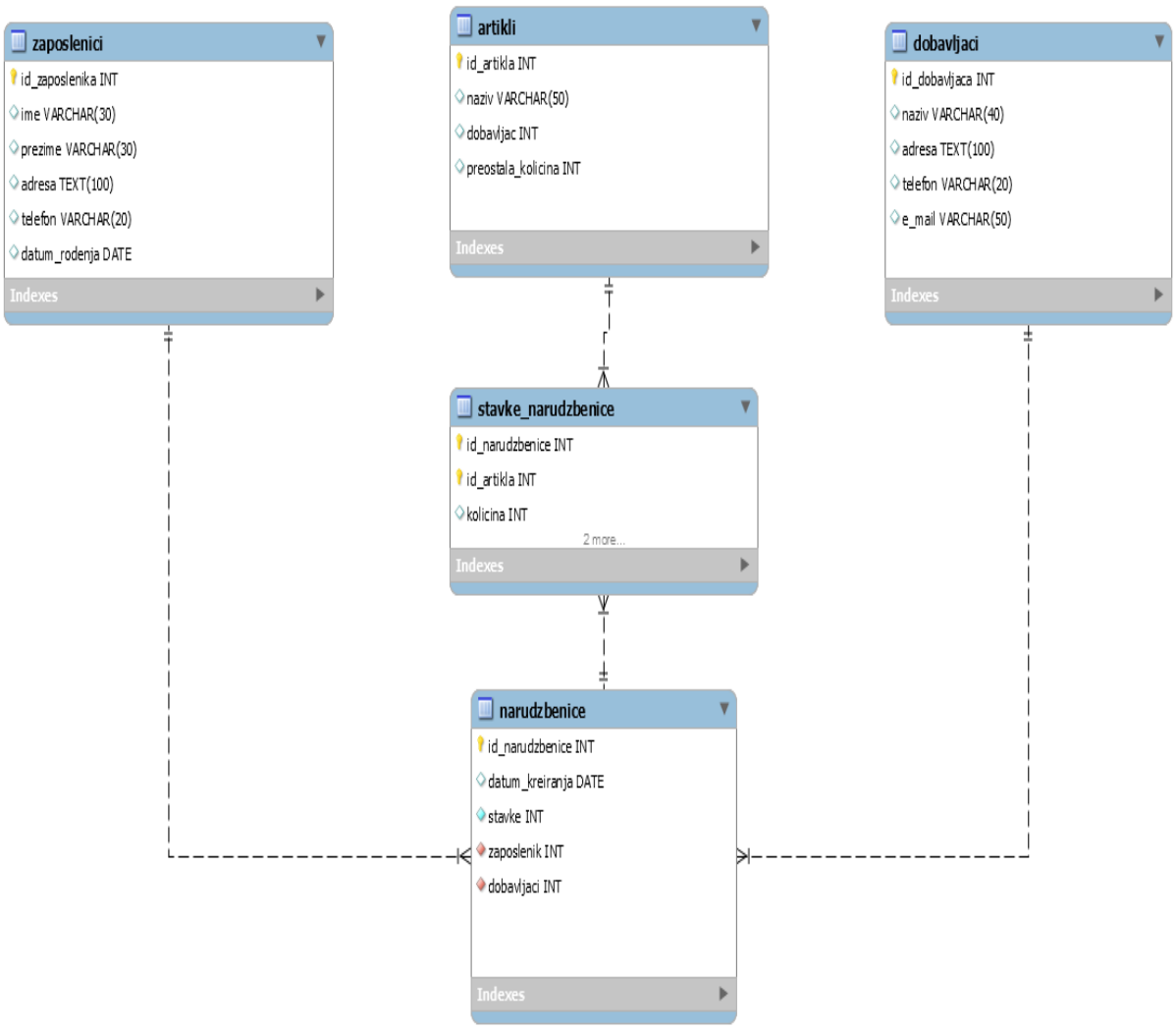
2.1.1. ER dijagram (engl. Entity – relationship)

ER dijagram ili ER model je model podataka za opis podataka ili informacijskih aspekata poslovne domene ili njezinih poslovnih zahtjeva prikazan na način da prikazuje entitete, njihove atribute i veze među njima. Glavne komponente ER modela su entiteti i veze. ER modeliranje je razvio Peter Chen i objavio rad o tome 1976. godine pod nazivom *"The Entity – Relationship Model – Toward A Unified View of Data"*. To je sistematski način objašnjavanja i definiranja poslovnih procesa. Proces je modeliran kao komponenta (entitet) koji su međusobno povezani vezama koje prikazuju ovisnosti među njima i zahtjeve. ER model se obično implementira kao baza podataka. U slučaju relacijske baze podataka, koja pohranjuje podatke u tablice, svaki red svake tablice predstavlja jednu instancu entiteta. Slika 2.3. prikazuje ER dijagram sustava opisanog u ovome radu. Dijagram prikazuje pet tablica koje će se kasnije kreirati u sustavu za upravljanje bazama podataka PostgreSQL, a upravo ovaj prikaz služi kao osnova za implementaciju navedenih tablica. Kao što se može vidjeti, u svakoj tablici napisan je naziv tablice na samome vrhu. Žuti ključ pored atributa *id_zaposlenika, id_artikla* i drugih sugerira da su ti atributi primarni ključevi tablica. Crvene točkice pored atributa *zaposlenici i dobavljači* u tablici *narudzbenice* sugeriraju da su to vanjski ključevi koji se referenciraju na druge tablice. Također se iz modela može iščitati kardinalnost, odnosno opcionalnost veza. Kod tablice *zaposlenici* postoji veza jedan naprama više što znači da samo jedan zaposlenik kreira narudžbenicu, ali da može kreirati više narudžbenica (dio kod tablice *narudzbenice* također je poznat kao „vranina noga“ (engl. *crows foot*). ER dijagram koji će se koristiti kreiran je u sustavu MySQL Workbench 6.2. [2]



Slika 2.2. *Sučelje MySQL Workbench programa*

Slika 2.2. prikazuje sučelje MySQL Workbench programa koji je jako jednostavan za korištenje. Klikom na ikonu *Place a new table* dodaje se nova tablica u dijagram, a dvostrukim klikom na dodanu tablicu otvaraju se opcije tablice te se unose atributi koje želimo imati u tablici. Također se pored atributa nalaze opcije kojima se određuje tip podataka atributa, hoće li atribut biti vanjski ili primarni ključ, unikatan i drugo. Nakon kreiranja svih tablica, povezujemo ih klikom na *Place a new relationship* te kliknemo na atribut koji će nam biti vanjski ključ i zatim na atribut na kojeg se referencira vanjski ključ. Nakon što je model gotov, klikne se na File->Export->Export as JPG i dobije se prikaz modela kojeg smo napravili u formatu JPG. Valja naglasiti kako ovaj alat ne služi samo za grafički prikaz modela, već omogućuje izravnu fizičku realizaciju ovih tablica. Način na koji se to radi je da se spoji na bazu podataka opcijom *Connect to database* te se nakon spajanja nude opcije *forward i reverse engineer. Forward engineer* znači da se podaci sa modela koji je napravljen preslikavaju u bazu podataka na koju smo se spojili. Nakon što se sve napravi, dobije se ER model (koji je kombinacija ER i relacijskog sustava jer prema Chenu nema stranih ključeva) kako je prikazano na slici 2.3. U radu se koristi ER dijagram koji je drugačiji od dijagrama s entitetima na slici 2.4.



Slika 2.3. *ER dijagram razvijenog sustava*

2.1.2. Entiteti, atributi i veze

Glavni cilj konceptualnog oblikovanja baze podataka je kreiranje sheme baze podataka koja će služiti za daljnji razvoj same baze, odnosno kod logičkog i kasnije kod fizičkog oblikovanja baze. Konceptualna shema daje prikaz same srži baze i jasan uvid u entitete koji će se kasnije koristiti. Moglo bi se reći da je konceptualno oblikovanje oslobođeno nekih tehničkih detalja. Ovakva shema nam opisuje stvarni svijet o kojemu se pohranjuju podaci u tablice. Konceptualna shema mora biti razumljiva svima, a ne samo ljudima koji se bave bazama podataka, jer bi trebala poslužiti kao sredstvo komunikacije između projektanta i korisnika i sl. Korisnici u komunikaciji sa projektantima određuju jesu li svi procesi uključeni

u bazu i jesu li ispravno kreirani odnosi među procesima ili entitetima. Konceptualna shema se ne implementira direktno uz pomoć sustava za upravljanje bazama podataka jer nije detaljno razrađena, ali je svakako važna za daljnje razvijanje same baze.

Modeliranje entiteta o kojima se spremaju podaci u tablice, zahtjeva shvaćanje osnovnih pojmova vezano za baze podataka, a to su entiteti, veze i atributi. Da bi se shvatilo što su to entiteti, najlakše ih se može objasniti kao objekte o kojima se govori u bazi podataka, odnosno stvari, bića, pojave ili događaji koji nas zanimaju. Kao što se može pretpostaviti, veze su odnosi između entiteta. Atributi su svojstva entiteta koji nas zanimaju.

2.1.3. Entiteti i njihovi atributi

Entitet je nešto što možemo identificirati i o čemu želimo spremati podatke. Entitet može biti kuća, student, profesor, automobil, film itd. Entitet također može biti pojava, npr. nogometna utakmica. Entitet opisujemo njegovim atributima, npr. ako imamo studenta, opisat ćemo ga atributima JMBAG, ime, prezime, godina upisa studija itd. Kandidat za primarni ključ nekog entiteta je atribut ili više njih pomoću čije vrijednosti možemo jednoznačno odrediti primjerak entiteta nekog tipa. Tako ne mogu postojati dva različita entiteta istog tipa sa istim vrijednostima. Ako neki entitet ima više kandidata koji mogu biti primarni ključ, tada biramo neki koji nam se čini prikladniji. Dobar primjer ključa za entitet tipa *student* je JMBAG studenta. Mogli bismo koristiti i kombinaciju atributa ime i prezime, no moguće je da imamo dva studenta sa istim imenom i prezimenom te nam se tu javljaju kolizije. Svi atributi koji opisuju isti entitet moraju imati različite nazive. Moguće je imati dva entiteta koji imaju iste atribute i iste nazive atributa.

2.1.4. Veze i njihovi atributi

Kada je uspostavljena veza između nekih entiteta, tada se ti entiteti nalaze u nekom odnosu. „Veza se uvijek definira na razini tipova entiteta, no realizira se povezivanjem primjeraka entiteta nekog tipa. Najčešći tip veze je binarna veza koja povezuje dva entiteta. Ako, npr. imamo entitete *student* i *predmet*, između njih možemo uspostaviti vezu *upisao* koja

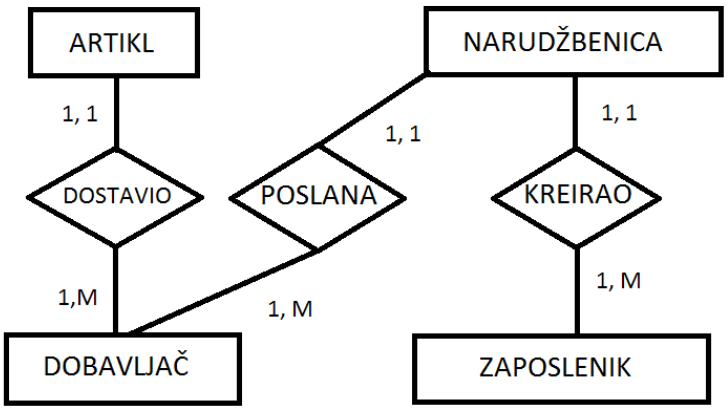
nam sugerira da je student upisao neki predmet.“[1] Stanje veze se prikazuje kao skup parova primjeraka entiteta koje promatramo. Kao što kod tipova entiteta moramo imati različita imena, tako i kod veza unutar iste sheme moramo imati različita imena.

2.1.5. Funkcionalnost veze, obaveznost članstva, kardinalnost

Načini na koji veza može povezivati primjerke nekog entiteta su određeni svojstvima funkcionalnosti, obaveznosti članstva, odnosno kardinalnosti. Poznavanje ovih svojstava je važno jer se veza mora ispravno prikazati unutar relacijske sheme. Prije spomenuta veza *upisao* između tipova entiteta *student* i *predmet* ima funkcionalnost M:M. To bi značilo da jedan student može upisati više predmeta, a jedan predmet može upisati više studenata.

2.1.6. Koraci u oblikovanju entiteta, veza i atributa

U ovome dijelu će se pokušati objasniti koraci kojima se na osnovi specifikacije dolazi do projektne dokumentacije. Prvi korak u oblikovanju konceptualne sheme je otkrivanje samih elemenata od kojih se shema sastoji. U pravilu, elementi se trebaju prepoznati čitanjem specifikacije. Imenice upućuju na atribute i entitete, a glagoli upućuju na veze. Kod prepoznavanja elemenata sheme, projektant često ne zna treba li imenicu shvatiti kao entitet ili kao atribut. Nakon što smo otkrili entitete, veze i atribute, potrebno je za svaki entitet utvrditi koji ga atributi opisuju. Također je moguće da neki atributi pripadaju vezi između entiteta, a ne pojedinim entitetima. Također za svaku vezu treba odrediti njezinu funkcionalnost, obaveznosti članstva te kardinalnost. Za svaki entitet potrebno je odabrati primarni ključ koji je ranije objašnjen. Specifikacija mora biti potpuno jasna kako bi se mogli otkriti entiteti, atributi i veze. Kod oblikovanja naše sheme postojat će tipovi entiteta *artikli, dobavljaci, narudzbenica, zaposlenici*. Veze između entiteta bit će *kreirao, dostavio* itd. Kako bi bilo jasnije, za realiziranu bazu podataka kreirat će se dijagram Chenovom grafičkom notacijom.



Slika 2.4. *Dijagram s entitetima i veze među njima*

2.2. Relacijski model – logičko oblikovanje baza podataka

Druga faza oblikovanja baze podataka je logičko oblikovanje. Cilj ove faze oblikovanja je stvoriti relacijsku shemu baze, odnosno shemu koja će opisati logičku strukturu baze u skladu s pravilima relacijskog modela podataka. U relacijskoj shemi nalaze se entiteti i veze među njima koje su pretvorene u relacije. Uz pomoć današnjih sustava za upravljanje bazama podataka moguće je implementirati relacijsku shemu izravno. Jedan od korisnih alata kojim bi to mogli postići, a koji koristi PostgreSQL sustav za upravljanje podataka je *MicroOlap database designer for PostgreSQL*.

2.2.1. Općenito o relacijskom modelu

Relacijski model se prvi put spominje u radovima Edgara Codda. Sredinom 80-ih godina model postaje najkorišteniji i standard u primjeni. Relacijski model zahtjeva da se baza podataka sastoji od skupa tablica. Svaka tablica ima svoje ime, a nemoguće je da u istoj shemi postoje dvije tablice sa istim imenom. Jedan stupac tablice sadrži vrijednost nekog

atributa, a atribut također ima ime koje bi se u tablici trebalo pojaviti samo jednom. Dozvoljeno je da se atributi u različitim tablicama zovu jednako, npr. atribut *ime* u tablici *zaposlenik* i atribut *ime* u tablici *dobavljač*. Ne postoji neko pravilo kojim redoslijedom se atributi kreiraju u nekoj tablici, iako je nekako običaj da se kao prvi atribut u tablici navede kao primarni ključ.

Kandidati za ključ, primarni ključ

Ključ K relacije R je podskup skupa atributa od R sa sljedećim svojstvima:

A) Vrijednosti atributa iz K jednoznačno određuju n-torku u R. Znači, ne mogu u R postojati dvije n-torke s istim vrijednostima atributa iz K.
B) Ako bi se izbacilo iz K bilo koji atribut, tada se narušava svojstvo pod A.

Npr. u tablici koja će se prikazati:

Zaposlenici id_zaposlenika čini primarni ključ, a kombiniranjem imena i prezimena nije dobro uzeti za primarni ključ jer se mogu pojaviti dva zaposlenika sa istim imenom. Budući da su sve n-torke u R međusobno različite, K uvijek postoji.

Relacijska shema, načini njezina zapisivanja

Građu relacije kratko se opisuje shemom tablice. To je redak koji se sastoji od imena tablice te popisa imena atributa odvojenih zarezima i zatvorenih u zagrade. Obično je primarni ključ podvučen kako bi se lakše shvatila relacijska shema.

Relacijska shema za primjer koji će se prikazati bi onda izgledala ovako:

Zaposlenici (id_zaposlenika, ime, prezime, telefon)

Dobavljaci (id_dobavljaca, naziv, adresa)

Artikli (id _artikla, naziv, preostala_kolicina, dobavljac)

Narudzbenica (id_narudzbenice, zaposlenik, datum_kreiranja, stavka, kolicina)

Navedeni prikaz relacijske sheme je vrlo pregledan i sažet, no nije dovoljan kako bi se shvatilo kojeg su tipa atributi. Za to je potrebno nadopuniti shemu rječnikom podataka, odnosno popisom svih atributa, s pripadnim tipovima vrijednosti i neformalnim opisom.

2.3. Tipovi podataka

Kada se kreira neka tablica (naredbom *create table*), potrebno je navesti i tip podataka za svaki stupac kojeg se kreira. U SQL standardu imamo sljedeće tipove podataka.

CHARACTER	BIGINT
CHARACTER VARYING	FLOAT
CHARACTER LARGE OBJECT	REAL
BINARY	DOUBLE PRECISION
BINARY VARYING	BOOLEAN
BINARY LARGE OBJECT	DATE
DECIMAL	TIME
NUMERIC	TIMESTAMP
SMALLINT	INTERVAL
INTEGER	

Tablica 2.1. *Tipovi podataka u SQL standardu*

Ovisno o tipu podataka koji je dodijeljen određenom stupcu, takav tip podataka se može i unositi u taj stupac. Nije moguće kreirati tip podataka *date* te upisivati podatke tipa *character*. Svaki od navedenih tipova podataka podržava i *null* vrijednost koja se upisuje u tablicu ako se trenutno ne zna koju vrijednost unijeti (napomena: *null* nije jednak nuli već je zamjena za praznu ćeliju).

2.3.1. Numerički tipovi

Sustav za upravljanje bazama podataka PostgreSQL podržava sljedeće numeričke tipove:

Ime	Zauzima
SMALLINT	2 B
INTEGER	4 B
BIGINT	8 B
DECIMAL	ovisi o unosu
NUMERIC	ovisi o unosu
REAL	4 B
DOUBLE PRECISION	8 B
SERIAL	4 B
BIGSERIAL	8 B

Tablica 2.2. *Numerički tipovi podržani sustavom PostgreSQL*

Razlika između tipova *smallint, integer i bigint* je u rasponu koje vrijednosti mogu poprimiti. Tako *smallint* poprima vrijednosti od -32768 do +32768, *integer* od -214783648 do + 214783648 itd. Ponekad je mudro odabrati *smallint* ako znamo da se u stupac neće unositi puno vrijednosti te će zbog manjeg zauzimanja memorijskog prostora sama baza podataka imati bolju funkcionalnost. Također pri definiranju tipova *smallint* često se piše INT2, a umjesto *bigint* INT8. Tipovi podataka *serial i bigserial* koriste se kada korisnik unosi u tablicu vrijednosti i želi da se vrijednost u ćeliji definiranom kao *serial ili bigserial* sama popunjava od broja 1 pa na dalje. Zapravo *serial i bigserial* nisu pravi tipovi podataka već su to sekvence.

2.3.2. Znakovni tipovi

Znakovni tipovi koje podržava sustav PostgreSQL su:

Naziv	Opis
VARCHAR	Niz znakova varijabilne dužine
CHAR	Niz znakova fiksne dužine
TEXT	Niz znakova varijabilne dužine

Tablica 2.3. *Znakovni tipovi podržani sustavom PostgreSQL*

Varchar i char mogu se pronaći u literaturi kao *character varying i character*, no uobičajeno je pisati samo *varchar i char*. Nakon što se definira tip podataka kao *char ili varchar*, u zagradi se navodi koliko se maksimalno znakova može upisati na mjesto gdje smo ih definirali. Tako ime *varchar(30)* znači da se može upisati 30 znakova u svaku ćeliju u stupcu *ime*. Prikladno je definirati dovoljan broj znakova, ali ne previše niti premalo. Ako bismo definirali previše, zauzimali bi mjesto u memoriji bez razloga te usporili performansu baze podataka, a ako bismo definirali premalo mogla bi se dogoditi situacija da netko ima dugačko ime, prezime ili adresu, a mi nemamo dovoljno mjesta za unos. Razlika između *char i varchar* tipa podataka je ta da je *varchar* varijabilne dužine, što znači, ako će se staviti 30 znakova, može se unijeti i manje od 30 (ali ne i više), no kod tipa *char* to nije slučaj. Kad bi se definirao stupac tipa *char(10)*, to bi značilo da se u svaku ćeliju stupca mora unijeti točno 10 znakova, niti više niti manje. Ovakav tip podataka je dobar za definiranje unosa nekih standardiziranih zapisa kao, npr. JMBAG studenta, OIB neke osobe i slično. Pošto se zna da se takvi zapisi sastoje od točno određenog broja znakova, tip podatka *char* sprječava krivi unos i kada netko unosi u tablicu, ne može pogriješiti, barem što se dužine znakova tiče. Ukoliko baza sadrži krive podatke, to se može odraziti i na funkcioniranje aplikacije koju baza podržava ili na rezultate poslovanja, ukoliko se radi o nekom poduzeću.

Varchar i text su nizovi znakova varijabilne dužine, no razlika između njih je da pri definiranju tipa *varchar* u zagradi navodimo maksimalnu dužinu, dok kod tipa *text* dužina upisa je bez ograničenja. Dakle, u tablicu se sprema onoliko znakova koliko korisnik unese.

Kod uspoređivanja tipova podataka *char i varchar* potrebna je konverzija naredbom *cast*, no o tome više kasnije.

2.3.3. Datum i vrijeme

Naziv	Opis
TIMESTAMP	Datum i vrijeme
DATE	Datum
TIME	Vrijeme
INTERVAL	Vremenski interval

Tablica 2.4. *Vremenski tipovi podataka u sustavu PostgreSQL*

Timestamp tipom podataka definira se datum i vrijeme. Također postoje verzije *timestampa* sa definiranom vremenskom zonom te bez definirane vremenske zone. Isto vrijedi i za tip podataka *time*. Interval tip podataka služi za zapis nekog intervala (od-do).

2.3.4. Ostali tipovi podataka

Od ostalih tipova podataka najčešće korišteni su *boolean i money*. *Boolen* tip podataka kao i u drugim jezicima može poprimiti vrijednost *true* i *false* ili *null*. Tipom podataka *money* spremamo novčane vrijednosti.

2.3.5. Null znak

Ukoliko se unose podaci u tablicu, a vrijednost nekog stupca nije poznata ili još nije definirana, koristit će se *null* znak. Valja odmah napomenuti kako *null* nije isto što i '0', već se jednostavno sa *null* znakom govori da vrijednost još nije poznata. Kako vrijednost neke ćelije može biti nula, nije prikladno stavljati nulu na mjesto gdje se ne zna vrijednost te se stoga

koristi *null*. *Null* je dakle specijalni marker koji se koristi u SQL-u i indicira da vrijednost podataka ne postoji u bazi. E.F. Codd, otac relacijskih baza podataka je prvi predstavio *null* znak. *Null* se u teoriji o bazama podataka predočava kao grčki znak omega (ω).

p	*q*	*p* OR *q*	*p* AND *q*	*p* = *q*
True	True	True	True	True
True	False	True	False	False
True	Unknown	True	Unknown	Unknown
False	True	True	False	False
False	False	False	False	True
False	Unknown	Unknown	False	Unknown
Unknown	True	True	Unknown	Unknown
Unknown	False	Unknown	False	Unknown
Unknown	Unknown	Unknown	Unknown	Unknown

Slika 2.5. *Logički operatori i 3VL (Three valued logic)* [3]

U sustavu PostgreSQL postoje neke funkcije za rad sa *null* znakom. Jedna od njih je funkcija *nullif* (vrijednost1,vrijednost2). Vrijednost1 i vrijednost2 su parametri koji se prenose funkciji ili se unose izravno, a funkcija uspoređuje dva argumenta i ako su im vrijednosti jednake, vraća *null*.

Ako argumenti koji su upisani nisu jednaki, funkcija vraća vrijednost prvog proslijeđenog parametra. Sljedeća funkcija za rad s *null* znakom je funkcija *coalesce*. Ova funkcija je korisna ako se želi tablicu popuniti sa podacima, a u tablici su već upisani *null* znakovi. *Coalesce* funkcijom mijenjaju se *null* znakovi sa vrijednostima koje želimo.

Tako, npr. u primjeru kojeg se prikazuje, zamislimo situaciju da imamo zaposlenika koji nema definiranu adresu te umjesto toga upišemo tekst *Nepoznata adresa*.

To se radi upravo funkcijom *coalesce* na način prikazan na slici. Najprije se dodaje zaposlenika bez definirane adrese te će se umjesto *null* upisati *Nepoznata adresa*.

```
skladiste=# select ime, prezime, telefon, datum_rodenja from zaposlenici where ime ='Mario' or ime = 'Zdenko' order by prezime, ime;
  ime   |  prezime  |  telefon   | datum_rodenja
--------+-----------+------------+---------------
 Mario  | Devcic    | 0996579899 | 1990-05-11
 Zdenko | Josipović | 0985478844 | 1981-11-24
(2 rows)


skladiste=# insert into zaposlenici values (default,'Josip','Solic',null,'0925781145','25.07.1970.');
INSERT 0 1
skladiste=# select * from zaposlenici;
 id_zaposlenika |  ime   |  prezime  |     adresa      |  telefon   | datum_rodenja
----------------+--------+-----------+-----------------+------------+---------------
              1 | Mario  | Devcic    | Trakoscanska 21 | 0996579899 | 1990-05-11
              2 | Ivan   | Maric     | Brace Radic 17  | 0915798814 | 1988-02-16
              3 | Ivan   | Maric     | Uska 2          | 0956581243 | 1985-04-18
              4 | Zdenko | Josipović | Masarykova 3    | 0985478844 | 1981-11-24
              5 | Josip  | Solic     |                 | 0925781145 | 1970-07-25
(5 rows)


skladiste=# select ime, prezime, coalesce(adresa,'Nepoznata adresa') from zaposlenici;
  ime   |  prezime  |     coalesce
--------+-----------+------------------
 Mario  | Devcic    | Trakoscanska 21
 Ivan   | Maric     | Brace Radic 17
 Ivan   | Maric     | Uska 2
 Zdenko | Josipović | Masarykova 3
 Josip  | Solic     | Nepoznata adresa
(5 rows)


skladiste=#
```

Slika 2.6. *Dodavanje novog zaposlenika s nepoznatom vrijednosti „adresa“ i zamjena null vrijednosti funkcijom coalesce*

3. OPIS I IZGRADNJA SUSTAVA

3.1. Oblikovanje baze podataka

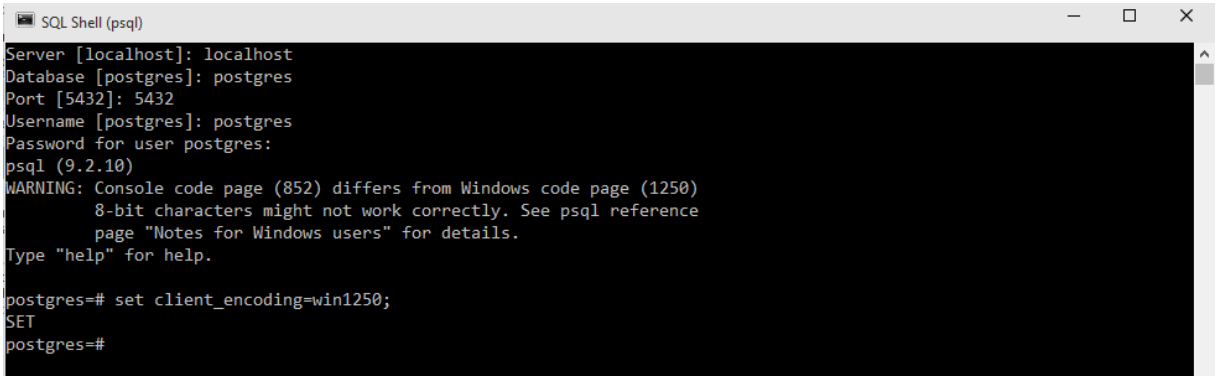
„Iako je SQL najčešće korišteni jezik za kreiranje i rad sa bazama podataka svakako nije i jedini, MS Access podržava QBE jezik, Ingres podržava QUEL itd. U ovom radu govorit će se o PostgreSQL sustavu za upravljanje bazama podataka.

Povijest jezika SQL vezana je za IBM laboratorije u San Joseu, California. 1970. Edgar Frank Ted Codd objavljuje članak pod nazivom *"A relation model of data for large shared data banks"* u kojemu kao osnovu za pohranjivanje podataka u bazu predlaže tablicu. Bez obzira što je Edgar Frank Ted Codd radio za IBM, *Relational Software, Inc.* razvijaju prvi sustav za upravljanje podataka koji koristi SQL. ANSI (engl. *American National Standards Institute*) objavljuje prvu verzija SQL standarda 1986. godine, dok ISO (engl. *International Organization for Standardization*) prihvaća standard 1987. Prva službena verzija, poznata kao i SQL1 izlazi 1989. godine, a zadnja verzija je SQL 2008.“ [4] Jednostavno rečeno, pod pojmom baze podataka podrazumijeva se skup povezanih tablica. Baza podataka se može pojaviti kao sastavni dio neke aplikacije ili može davati podršku aplikaciji. Razvoj baze podataka odvojen je od razvoja aplikacije. Prije nego što se počne sa kreiranjem baze podataka, potrebno je poznavati faze oblikovanja baze podataka jer bi se bez osnovnih znanja o oblikovanju baza podataka vrlo brzo našli u problemima. Oblikovanje baze podataka se sastoji od tri faze: konceptualnog, logičkog i fizičkog oblikovanja. Rezultat prve faze oblikovanja je logička shema baze koja se sastoji od entiteta, atributa i veza. Ovakav prikaz je neformalan i lako razumljiv. Logičkim oblikovanjem se popravlja struktura samih tablica te se provodi normalizacija baze podataka. Fizičko oblikovanje kao posljednja faza izgradnje baze podataka kao rezultat daje fizičku shemu cijele baze. Fizička shema je zapravo niz SQL naredbi kojima se relacije iz logičke sheme realiziraju kao tablice. Daljnji postupak sastoji se od punjenja tablica podacima. Takvi podaci obično postoje u nekom obliku, npr. kao tekstualni dokumenti i slično. Međutim, vrijednost baze podataka ne očituje se u samim podacima, već u njihovoj strukturiranosti. Ciljevi koji se nastoje postići uporabom baza podataka su: fizička nezavisnost podataka, logička nezavisnost podataka, fleksibilnost pristupa podacima, istovremeni pristup do podataka i očuvanje integriteta. Način kreiranja nove baze podataka razlikuje se od sustava do sustava, no obično je to naredba *create database*. Nakon kreiranja baze, korisnik se na nju može spojiti naredbom *connect*.

Na početku dizajniranja baze podataka potrebno je odrediti entitete, atribute te veze među njima. Entitet je objekt o kojemu prikupljamo podatke, npr. student. Atributi koji pobliže opisuju neki entitet su ime, prezime, JMBAG studenta, datum rođenja i slično. Na temelju entiteta nastaju tablice tako da za svaki entitet imamo po jednu tablicu. Svaki red tablice sadrži podatke o konkretnoj instanci pojedinog entiteta, dok stupci sadrže vrijednosti istog tipa za svaki redak u tablici. Kako bi se baza podataka dobro dizajnirala, potrebno je kreirati tablice na način da se u tablici nalaze samo relevantni podaci koji će se koristiti u nekoj aplikaciji koju podržava baza podataka. Tako, npr. nas ne zanima koja je boja voća u tablici, jer nam je taj podatak irelevantan za naš sustav. Nakon što se odrede entiteti potrebni za kreiranje baze te veze među njima, potrebno je kreirati same tablice. Kod povezivanja tablica, obično jednu tablicu zovemo *tablica roditelj*, a druga se naziva *tablica potomak*. Za potrebe ovog završnog rada kreirat će se baza podataka *skladiste* te će se svi daljnji primjeri pokazivati nad tom bazom. Naredba za kreiranje tablice je naredba *create table*.

3.1.1. Create table

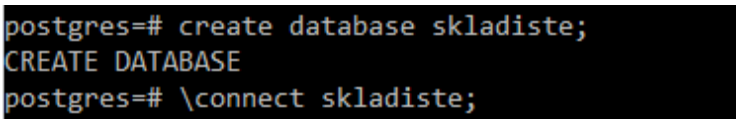
Naredba *create table* jedna je od DDL (engl. *Data definition language*) naredbi te omogućuje kreiranje tablice. Često se kaže da životni ciklus baze započinje naredbom *create*. Također postoje naredbe *create view*, *create index* za kreiranje pogleda i indeksa. Vlasnik tablice je korisnik koji je izvršio naredbu *create table*. Nakon što se tablica kreira, ona je prazna te se za umetanje podataka u tablicu koristi naredba *insert*. Sada će se kreirati tablica *zaposlenici* koja će sadržavati osnovne podatke o zaposlenicima u poduzeću. Podaci koje će se unositi biti će *id_zaposlenika, ime, prezime, adresa, telefon i datum_rodenja*. Kako bi se moglo raditi sa sustavom PostgreSQL potrebno je preuzeti instalaciju, instalirati sustav i sustav je spreman za rad. Nakon pokretanja SQLShell (psql) pojavljuje se početni prozor gdje sustav korisnika traži da unese podatke za prijavu. Početni prozor izgleda kao što je prikazano na slici 3.1.



Slika 3.1. *Početni prozor*

Nakon što su uneseni podaci, ispisuje se upozorenje kako 8-bitni znakovi možda neće raditi ispravno. Kako bi se to promijenilo, postavlja se *client_encoding=win1250* kako bi se moglo raditi ispravno sa specifičnim znakovima hrvatske abecede (č,ć,dž,đ,š,ž).

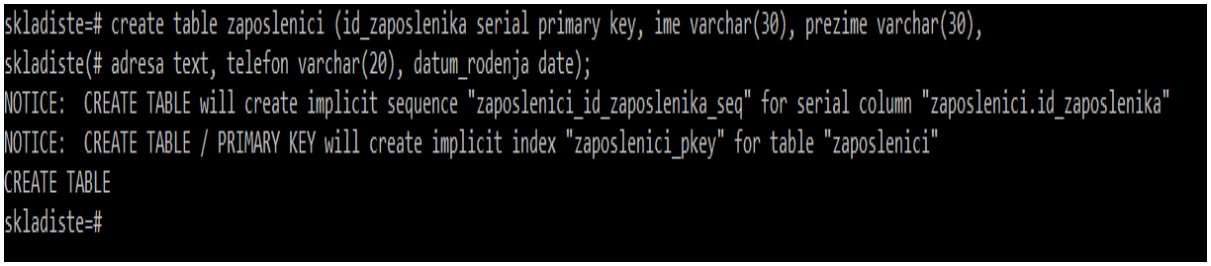
Prvo će se kreirati baza podataka naredbom *create database*, a spajanje na nju će se izvršiti naredbom *\connect*.



Slika 3.2. *Kreiranje nove baze*

Bazu podataka nazvali smo *skladiste* te smo se spojili na nju.

Nakon što je baza kreirana, može se kreirati i prva tablica u njoj naredbom *create table*.



Slika 3.3. *Kreiranje nove tablice*

Može se uočiti kako se ispisuje poruka nakon kreiranja tablice koja kaže kako će *create table* kreirati implicitnu sekvencu koja se odnosi na primarni ključ. Također se može uočiti kako se naziv baze uvijek nalazi na početku svakog reda *(„skladiste=#“).*

Naredba *create table* radi tako da se prvo napiše *create table* te nakon toga ime tablice koja se kreira (u primjeru koji se opisuje *zaposlenici*), zatim se u zagradama navode atributi te im se pridružuje tip podataka kojeg je svaki atribut. Tip podataka *serial* znači da se pri svakom unosu podataka vrijednost automatski povećava od broja 1 do n gdje je n broj unosa u tablicu. ID je odabran za primarni ključ jer će svaki unos u tablicu imati različiti ID, a to je važno za primarni ključ. Naime, primarni ključ mora biti stupac koji će jednoznačno određivati svaku instancu entiteta. Sada će se kreirati ostale tablice koje će se nalaziti u bazi *skladiste*.

```
skladiste=# create table zaposlenici (id_zaposlenika serial primary key, ime varchar(30), prezime varchar(30),
skladiste(# adresa text, telefon varchar(20), datum_rodenja date);
NOTICE:  CREATE TABLE will create implicit sequence "zaposlenici_id_zaposlenika_seq" for serial column "zaposlenici.id_zaposlenika"
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "zaposlenici_pkey" for table "zaposlenici"
CREATE TABLE
skladiste=# create table dobavljaci (id_dobavljaca serial primary key, naziv varchar(40), adresa text, telefon varchar(20), e_mail varchar(50));
NOTICE:  CREATE TABLE will create implicit sequence "dobavljaci_id_dobavljaca_seq" for serial column "dobavljaci.id_dobavljaca"
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "dobavljaci_pkey" for table "dobavljaci"
CREATE TABLE
skladiste=# create table artikli (id_artikla serial primary key, naziv varchar(50), dobavljac bigint references dobavljaci, preostala_kolicina int);
NOTICE:  CREATE TABLE will create implicit sequence "artikli_id_artikla_seq" for serial column "artikli.id_artikla"
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "artikli_pkey" for table "artikli"
CREATE TABLE
skladiste=# create table narudzbenice (id_narudzbenice serial primary key, datum_kreiranja date, artikl bigint references artikli, kolicina int, zaposlenik bigint
skladiste(# references zaposlenici);
NOTICE:  CREATE TABLE will create implicit sequence "narudzbenice_id_narudzbenice_seq" for serial column "narudzbenice.id_narudzbenice"
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "narudzbenice_pkey" for table "narudzbenice"
CREATE TABLE
skladiste=#
```

Slika 3.4. *Kreiranje tablica*

Kreirane su tablice *zaposlenici, dobavljaci, artikli i narudzbenice*. U tablici *zaposlenici* imamo atribute *ime, prezime, adresa, telefon, datum_rodenja*. Primarni ključ u ovoj tablici je *id_zaposlenika*. Sljedeća tablica je tablica *dobavljaci* koja sadrži atribute *id_dobavljaca* koji je primarni ključ tablice, zatim *naziv, adresa, telefon, e_mail*. Sljedeća

tablica je tablica *artikli* u kojoj postoje *id_artikla* kao primarni ključ, *naziv, dobavljac* kao vanjski ključ na tablicu *dobavljaci* te *preostala_kolicina* koja će govoriti koliko je još ostalo količine određenog artikla na skladištu. Posljednja tablica je *narudzbenice* koja se sastoji od atributa *id_narudzbenice* kao primarnog ključa, *datum_kreiranja, artikli* kao vanjski ključ na tablicu *artikli, kolicina,* te *zaposlenik* kao vanjski ključ na tablicu *zaposlenici*.

Primarni ključ tablice može biti i složeni, te bi se mogla uzeti kombinacija imena i prezimena kao primarni ključ. To se postiže naredbom *primary key* (ime,prezime). No postoji mogućnost da imamo zaposlene dvije osobe sa istim imenom i prezimenom što je čest slučaj u velikim poslovnim sustavima te onda sve instance entiteta nisu jednoznačno određene. Tip podataka *varchar ("Variable character")* označava tekstualni tip podataka varijabilne dužine. Pošto je u zagradu stavljen broj 30, maksimalan broj znakova koji se može upisati u polje ime će biti 30. Analogno vrijedi i za polje prezime. Tip podataka *date* služi za definiranje datuma. Tip podataka *char* razlikuje se od *varchar* u tome što *char* mora sadržavati točno onoliko znakova koliko je navedeno u zagradi, ali o samim tipovima podataka već je bilo govora u dijelu gdje je detaljno razrađen i objašnjen svaki tip.

3.1.2. Primarni ključ

"Svaki redak u tablici predstavlja jednu instancu entiteta, a svaka instanca entiteta (student, nastavnik, predmet...) može se jednoznačno identificirati. Primarni ključ (engl. *Primary key*) je ograničenje nad stupcem koje ograničava vrijednosti stupca." [5]. U stupac koji sadrži vrijednost primarnog ključa mora se unijeti podatak (ne može biti *null*), jer mu bez primarnog ključa ne bi mogli pristupiti. Primarni ključ može biti složen (sastojati se od više stupaca) pa se često kaže kako je primarni ključ stupac ili više stupaca pomoću kojeg možemo jednoznačno identificirati svaki red u tablici. Pri kreiranju tablice praktično je staviti primarni ključ u prvi stupac, ponajprije zbog preglednosti tablica.

3.1.3. Vanjski ključ

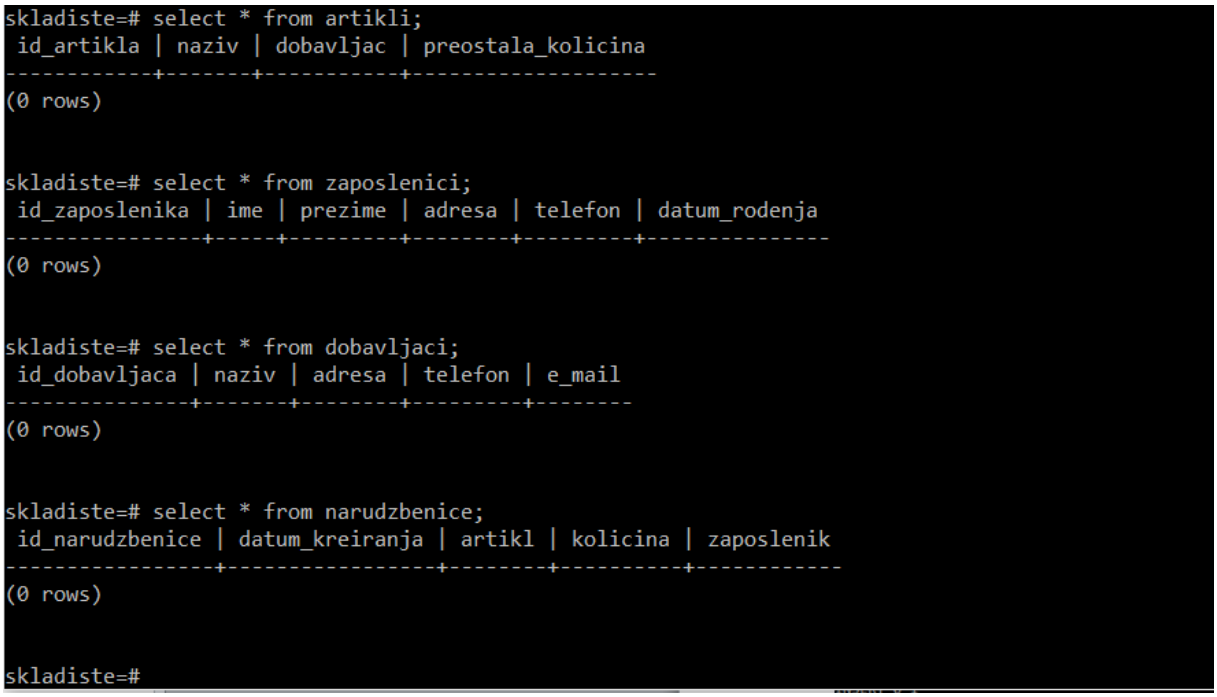
„Atribut (ili skup atributa) u tablici nazivamo vanjskim ključem (engl. *Foreign key*) ako ima funkciju primarnog ključa u nekoj drugoj relaciji, moguće i u istoj. Odnosno, vanjski ključ je vrijednost primarnog ključa pohranjena u drugoj (ili istoj) tablici.“ [6] Kod kreiranja vanjskog ključa i unosa u tablicu mora se paziti da tip i veličina vanjskog ključa odgovara primarnom ključu na kojeg se vanjski referencira. Referencijalni integritet je sustav pravila koji osigurava da veze između zapisa u povezanim tablicama budu važeće i da ne možemo slučajno izmijeniti ili obrisati podatke. Vanjski ključ tablice također može biti jednostavan ili složen. Referencijalni integritet se provjerava kod korištenja neke od DML naredbi ili naredbe *drop*. Sustav neće dozvoliti brisanje *tablice roditelj* na koju se referencira neka druga *tablica potomak*.

Kod kreiranja tablica i određivanja njihovih atributa moguće je definirati da atribut ima vrijednost *not null*. *Not null* vrijednost znači da se taj podatak mora unijeti u tablicu (ne može biti *null*).

Naredbom *references* se određuje na primarni ključ koje tablice se referencira novi stupac kojeg smo kreirali, a naredbe *on update* i *on delete* određuju što će se dogoditi sa vrijednosti vanjskog ključa u *tablici potomak* ukoliko se vrijednost obriše iz *tablice roditelj*.

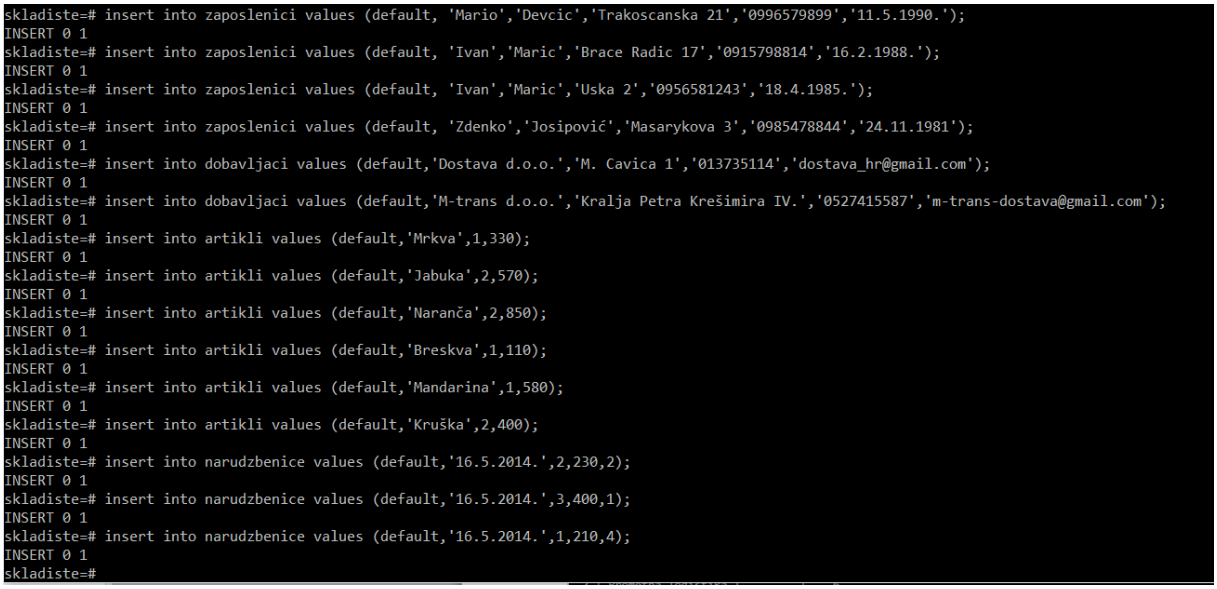
3.1.4. Naredbe select i insert

Sada će se prikazati kako se kreirane tablice pune podacima. Pogledajmo prvo strukturu samih tablica. Struktura tablica može se vidjeti naredbom *select*, konkretno u primjeru "*SELECT * FROM zaposlenici;"*. Zvjezdica (*) označava da se ispiše sve iz tablice. Pošto su tablice prazne, odnosno nisu popunjene podacima, dobiva se sljedeća struktura tablica.



Slika 3.5. *Struktura kreiranih tablica*

Tablice se popunjavaju naredbom *insert*. Za naredbu *insert* važno je ime tablice kao i nazivi stupaca te sami literali (vrijednosti koje se upisuju u tablice). Pošto je definiran tip podataka *serial*, za njega će se upisati *default* vrijednost, jer sustav sam generira brojeve pri svakom unosu. Za neku vrijednost može se postaviti i vrijednost *null* osim ako nije postavljeno ograničenje *not null*. Postoji nekoliko načina korištenja naredbe *insert*. Mogu se eksplicitno navesti stupci u koje se unose podaci, npr. *INSERT INTO zaposlenici (ime, prezime) VALUES ('Mario, Devčić').* Ovom naredbom će se popuniti ime i prezime sa navedenim podacima, dok se *id_zaposlenika* automatski povećava. Na primjeru će se pokazati drugi način i popunit će se sve tablice podacima.



Slika 3.6. *Umetanje podataka u tablice*

Kod korištenja *insert* naredbe, vrijednosti se upisuju unutar jednostrukih navodnika, osim kod upisa *default* vrijednosti ili *integera* za koje nisu potrebni navodnici. Nakon što su popunjene tri tablice sa nasumičnim podacima, dobiva se jednostavna baza podataka te njezine tablice izgledaju ovako. Naredbom *insert* moguće je umetati i više redova odjednom. Ukoliko se želi kreirati tablica koja se sastoji od podataka iz druge tablice, koristit će se naredba *select into*, npr. *SELECT INTO zaposlenici2 * FROM zaposlenici*. Ovom naredbom dobili bi identičnu tablicu sa istim podacima i istim tipovima podataka kao *zaposlenici*, samo drugog naziva *(zaposlenici2).* Isti učinak ima naredba *CREATE TABLE AS. CREATE TABLE zaposlenici2 AS SELECT * FROM zaposlenici* daje jednaku tablicu kao gore opisana naredba.

Naredbom *Select * from* [ime tablice]; dobiva se pregled cijele strukture tablice i prikaz podataka. Nakon što je sve napravljeno, može se vidjeti gotova početna verzija baze podataka sa tablicama i podacima u njima.

```
skladiste=# select * from zaposlenici;
 id_zaposlenika |  ime   |  prezime  |     adresa      |  telefon   | datum_rodenja
----------------+--------+-----------+-----------------+------------+---------------
              1 | Mario  | Devcic    | Trakoscanska 21 | 0996579899 | 1990-05-11
              2 | Ivan   | Maric     | Brace Radic 17  | 0915798814 | 1988-02-16
              3 | Ivan   | Maric     | Uska 2          | 0956581243 | 1985-04-18
              4 | Zdenko | Josipović | Masarykova 3    | 0985478844 | 1981-11-24
(4 rows)


skladiste=# select * from dobavljaci;
 id_dobavljaca |     naziv      |          adresa           |  telefon   |         e_mail
---------------+----------------+---------------------------+------------+-------------------------
             1 | Dostava d.o.o. | M. Cavica 1               | 013735114  | dostava_hr@gmail.com
             2 | M-trans d.o.o. | Kralja Petra Krešimira IV. | 0527415587 | m-trans-dostava@gmail.com
(2 rows)


skladiste=# select * from artikli;
 id_artikla |   naziv   | dobavljac | preostala_kolicina
------------+-----------+-----------+--------------------
          1 | Mrkva     |         1 |                330
          2 | Jabuka    |         2 |                570
          3 | Naranča   |         2 |                850
          4 | Breskva   |         1 |                110
          5 | Mandarina |         1 |                580
          6 | Kruška    |         2 |                400
(6 rows)


skladiste=# select * from narudzbenice;
 id_narudzbenice | datum_kreiranja | artikl | kolicina | zaposlenik
-----------------+-----------------+--------+----------+------------
               1 | 2014-05-16      |      2 |      230 |          2
               2 | 2014-05-16      |      3 |      400 |          1
               3 | 2014-05-16      |      1 |      210 |          4
(3 rows)


skladiste=#
```

Slika 3.7. *Prikaz tablice sa unesenim podacima*

3.1.5. Naredba alter table

U konkretnim slučajevima kreiranja baze podataka, može se dogoditi da se želi dodati ili obrisati neki stupac u tablicu koja je već kreirana. Kod takvog slučaja potrebna je naredba *alter table*. Samom naredbom može se dodati stupac, obrisati stupac, postaviti *default* vrijednost ili preimenovati već postojeći stupac. Pokazat će se na primjeru kako izgleda preimenovanje stupca npr. e_mail u email.

```
skladiste=# alter table dobavljaci rename column e_mail to email;
ALTER TABLE
skladiste=#
```

Slika 3.8. *Naredba alter table*

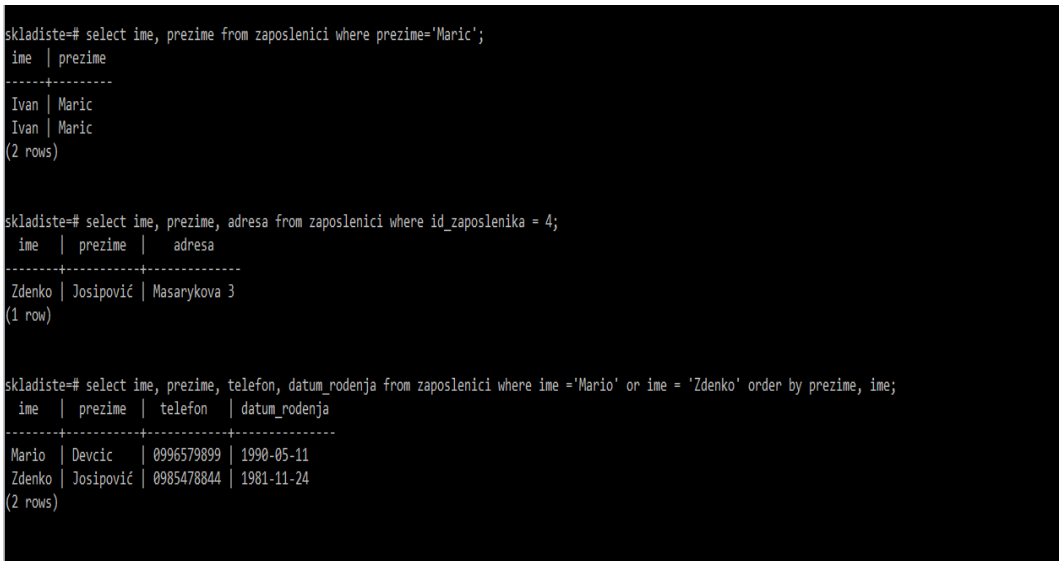
Ukoliko se želi promijeniti samo naziv tablice, to se može učiniti naredbom *ALTER TABLE dobavljaci RENAME to (novo ime).*

3.1.6. Klauzule from, where, order by

Klauzulom *from* specificiramo u naredbi na koju tablicu se odnosi naš upit. Npr. želi li se izlistati sve iz tablice *zaposlenici* napisat će se upit kao na slici 3.7. Ako se želi ispisati, npr. samo ime i prezime iz tablice *zaposlenici*, eksplicitno će se navesti ti stupci na sljedeći način: *SELECT ime, prezime FROM zaposlenici*.

Where klauzula u SQL-u specificira da se *SQL DML* naredba treba izvršavati samo nad redovima koji zadovoljavaju navedeni kriterij. Kriteriji su izraženi jednim ili skupom predikata. Klauzulom *where* dobiva se 0 ili više redova, ovisno o tome koliko redova zadovoljava navedeni uvjet.

Order by je klauzula koja se koristi u SQL-u za sortiranje podataka. Recimo da nam je potreban popis radnika prema abecedi ili prema starosti koristit će se klauzula *Order by*. Sortirati se može prema vrijednostima stupca ili više njih. Ako bi sortirali radnike prema imenu i prezimenu, tada bi se prvo svi sortirali prema imenu, a oni koji imaju jednaka imena bi se još sortirali i po prezimenu. Želi li se sortirati silazno, nakon stupca po kojem se sortira dodaje se ključna riječ DESC (engl. *Descending*). Također je moguće sortirati po jednom stupcu silazno, a po drugom uzlazno unutar istog upita. Sljedećih nekoliko primjera pokazuju korištenje navedenih klauzula.



Slika 3.9. *Korištenje klauzule from, where, order by*

3.1.7. Klauzule limit, offset i dinstict

Kao što se već može zaključiti, klauzula *limit* služi kako bi se ispisao samo određeni broj redova iz tablice. Ukoliko bi željeli ispisati prva dva artikla koji se nalaze u tablici, koristit će se klauzula *limit*. Kod klauzule *limit* također je moguće prvo sortirati redove te onda postaviti limit koliko ih želimo ispisati uz pomoć klauzule *order by*. Ukoliko ne postoje redovi koji zadovoljavaju upit unutar klauzule *limit*, sustav će vratiti onoliko redova koliko ih ima u tablici.

Klauzula *offset* također je vrlo jednostavna klauzula za korištenje. Pomoću nje se definira pomak, odnosno koliko redova se želi preskočiti pri ispisu. Klauzula *distinct* nešto je složenija. Pomoću nje se govori sustavu da samo jednom ispiše svaku vrijednost, što bi značilo, ako se neka vrijednost pojavljuje više puta, sustav će ju samo jednom prikazati. Ako bi, npr. željeli ispisati sva imena zaposlenika iz tablice, ali bez ponavljanja, koristit će se klauzula *distinct*. Korištenje svih ovih klauzula pokazat će se na primjerima koji slijede.

```
skladiste=# select id_artikla,naziv,preostala_kolicina from artikli limit 2;
 id_artikla | naziv  | preostala_kolicina
------------+--------+--------------------
          1 | Mrkva  |                330
          2 | Jabuka |                570
(2 rows)


skladiste=# select id_artikla,naziv,preostala_kolicina from artikli where preostala_kolicina<200 limit 2;
 id_artikla |  naziv  | preostala_kolicina
------------+---------+--------------------
          4 | Breskva |                110
(1 row)


skladiste=# select naziv from artikli offset 1;
   naziv
-----------
 Jabuka
 Naranča
 Breskva
 Mandarina
 Kruška
(5 rows)


skladiste=# select distinct ime from zaposlenici;
  ime
--------
 Mario
 Zdenko
 Josip
 Ivan
(4 rows)
```

Slika 3.10. *Korištenje klauzula limit, offset i distinct*

Može se uočiti kako se iz tablice *zaposlenici* ispisalo samo jedno ime „Ivan“ iako su u tablici navedena dva zaposlenika koji se zovu „Ivan“.

3.2. Izgradnja aplikacije

Razvojni ciklus informacijskog sustava ili podsustava je vremensko razdoblje između donošenja formalne odluke o razvoju i formalne isporuke ili formalnog prekida razvoja ANSI/IEEE(1983a). Rezultat razvoja je ciljni proizvod, tj. određeni tip ili varijanta primjenjivog proizvoda. Za neki poduhvat razvoja, predložak razvojnog ciklusa preuzima se doslovno iz odabrane metodike razvoja ili djelomično prilagođava stvarnim potrebama. Kao što kod izgradnje baze podataka imamo faze izgradnje, tako i kod aplikacija imamo određene faze. Te faze su: analiza i specifikacija zahtjeva, logičko modeliranje, fizičko modeliranje i izrada, isporuka i primjena te održavanje i poboljšavanje.

IDE (engl. *Integrated Development Environment*) je softver koji programeru pruža jednostavniji razvoj softvera jer na jednome mjestu obuhvaća editor u kojemu se piše kod, kompajler/interpreter te debugger. U prošlosti se razvoj softvera odvijao na način da je

programer koristio tekst editor (Notepad++, pico, nano i dr.), napisao kod u editoru te ga spremio u neki direktorij te uz pomoć kompajlera kompilirao napisani kod itd. Moderni alati za kreiranje aplikacija sve navedene specifikacije imaju na jednome mjestu te uz njih imaju i dodatne alate za razvoj softvera kao što su alati za spajanje na bazu podataka, dovršavanje koda prilikom razvoja, strukturiranje koda itd. Korištenje modernih alata za izgradnju aplikacija povećava produktivnost posebno pri izgradnji aplikacija koje sadrže veći broj klasa. Neki od poznatijih IDE alata su: NetBeans, Eclipse, PyCharm, Xcode, Visual Studio i drugi. Nakon što je objašnjen sustav za upravljanje bazom podataka i njegove razne mogućnosti te implementirana baza podataka u sam sustav, može se započeti sa drugom fazom rada – izgradnjom aplikacije. Sama baza podataka nam ne pomaže puno ako moramo „ručno“ svaki put u nju unositi, brisati ili mijenjati podatke. Za to nam služi aplikacija koja radi nad tom bazom. Postoje brojni alati za izradu aplikacija, a u ovome radu koristit će se popularni alat za izgradnju *desktop* i *web* aplikacija Microsoft Visual Studio 2013. [7]

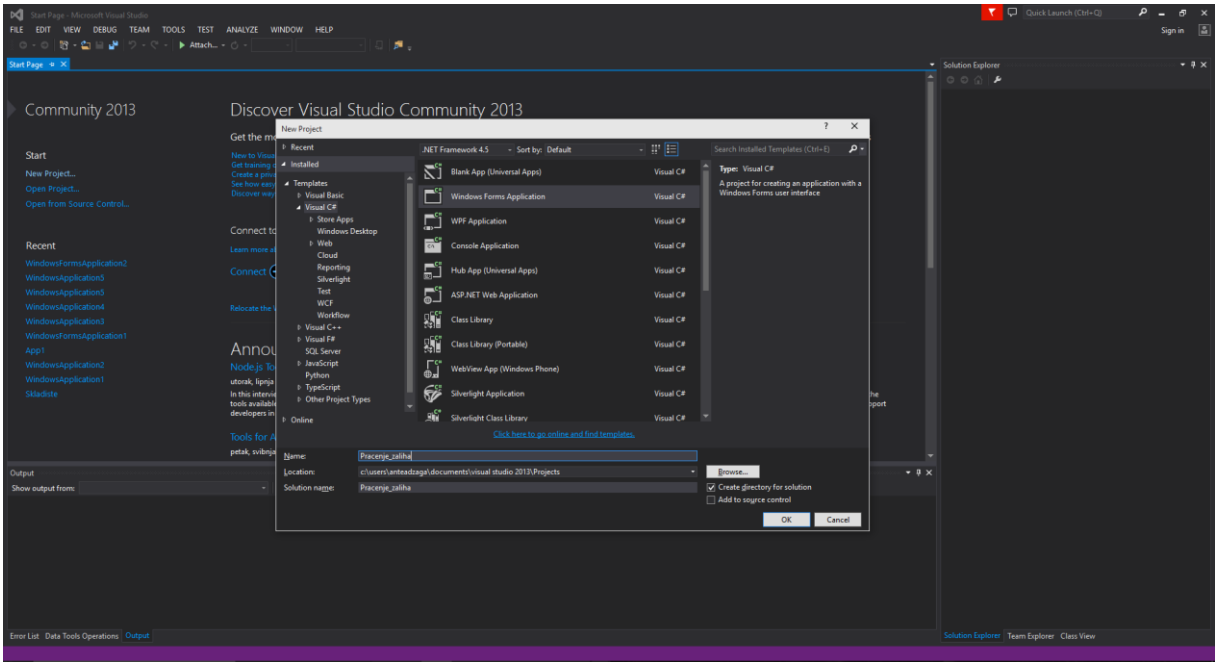
3.2.1. Rad u alatu Microsoft Visual Studio

Microsoft Visual Studio je integrirano razvojno okruženje (IDE) kojeg proizvodi Microsoft. Koristi se za razvoj računalnih programa, izgradnju web stranica, web aplikacija i usluga. Koristi Microsoftove platforme za razvoj poput *application programming interface-a* za *windows, windows forms, windows store* i dr. Program sadrži integrirani debugger koji radi na nivou izvornog i strojnog koda. Također se u njemu nalaze alati dizajnera oblika koji se koriste za pravljenje aplikacija sa grafičkim korisničkim sučeljem, web dizajnera, dizajnera klasa i dizajnera shema baza podataka. Microsoft Visual Studio podržava različite programske jezike i dozvoljava programeru i debuggeru gotovo bilo koji programski jezik. Ugrađeni jezici koji se koriste su C, C++, i C++/CLI, VB.NET, C# i F#. Uz sve navedene podržane programske jezike, podržani su i ostali kao što su Python, Ruby uz instalaciju dodataka koji služe kako bi mogli koristiti sintaksu tih jezika. Ovaj alat je nije besplatan, ali nudi *trial* verziju od 30 dana. Nakon što se preuzme instalacijski paket i instalira program, program je spreman za rad. U ovome radu koristit će se Microsoft Visual Studio uz upotrebu programskog jezika C#.

Nakon što se pokrene alat Microsoft Visual Studio, otvara se korisničko sučelje kod kojeg se može uočiti veliki broj opcija. Klikom na gumb File->New->Project otvara se

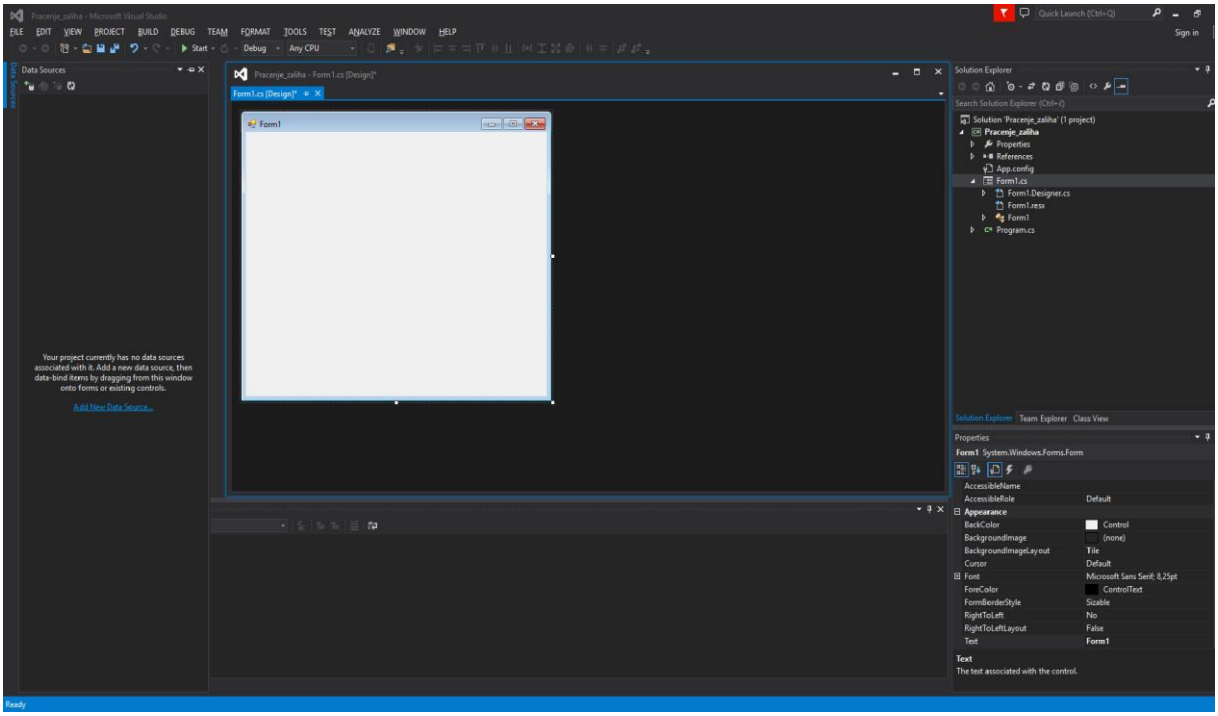
kreiranje novog projekta. Alat nudi veliki izbor programskih jezika koji se mogu koristiti te vrste aplikacija koje se mogu kreirati. Izabire se programski jezik C# te Windows Forms Application. Također na vrhu izbornika koji nam se otvorio može se izabrati .NET framework. Valja napomenuti da Visual Studio služi primarno za razvoj .NET aplikacija. .NET je Microsoftov okvir, odnosno skup biblioteka koji pojednostavljuje razvoj aplikacije namijenjene za tehnologije Microsofta, kao što su Windows, Windows Phone, Windows Azure i dr. Prilikom kompajliranja programa napisanog za okvir .NET (npr. u jeziku C++), koristi se zajednička infrastruktura (engl. *Common Language Infrastructure*) za prijevod na nižu programsku razinu. Svi jezici okvira .NET se prevode u zajednički jezik CIL (engl. *Common Intermediate Language*). CIL je neovisan o platformi. Program preveden u CIL se nakon toga izvršava u CLR (engl. *Common Language Runtime*) koja je zajednička infrastruktura ovisna o platformi (procesor, grafički proceso, operacijski sustav i dr.).

Projekt će se zvati *Pracenje_zaliha* te je potrebno kliknuti *Ok* kako bi se kreirali svi potrebni predlošci za rad.



Slika 3.11. *Početni zaslon i izbornik Microsoft Visual Studia*

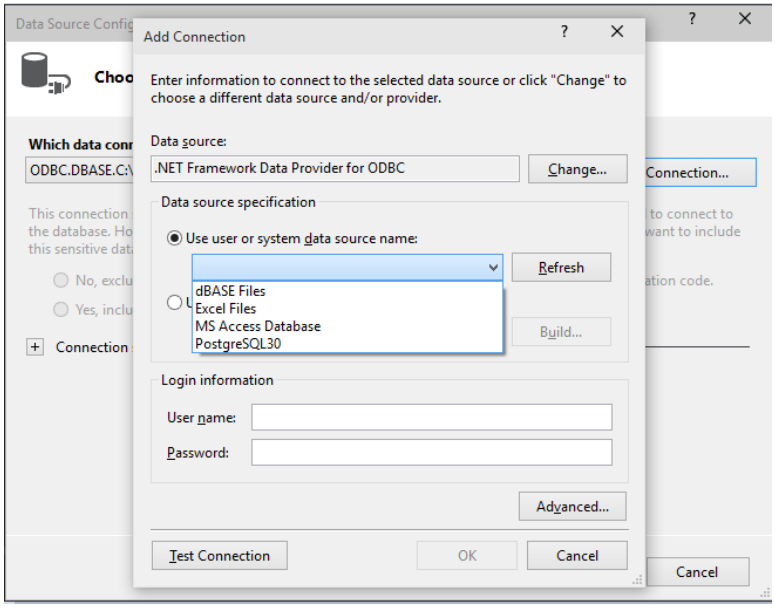
Slika 3.11. prikazuje početni zaslon te izbornik koji se otvara klikom na *new project*. Otvara se forma koja se može vidjeti na slici 3.12. koja će se prikazivati pri pokretanju aplikacije kada bude gotova. S lijeve strane glavnog prozora nalazi se tab „Data Sources“ koji služi za spajanje na bazu podataka koja će se koristiti. S desne strane nalazi se *solution explorer* te *properties*, odnosno opcije objekta na koje se klikne. Tabovi u izbornoj traci kao npr. *edit, project, build, debug* i drugi omogućuju brojne opcije. Iako se možda na prvi pogled korisničko sučelje čini jednostavno, ipak je potrebno poznavati rad sa bazama podataka te objektno orijentirani pristup programiranju za rad sa alatom Microsoft Visual Studio. Klikom na tipku F7 na tipkovnici u bilo kojem trenutku može se otvoriti novi prozor u kojem će biti prikazan kod koji se nalazi u pozadini onoga što vidimo. Taj kod se može po potrebi dodavati, brisati i mijenjati ovisno o zahtjevima aplikacije.



Slika 3.12. *Microsoft Visual Studio Windows form*

Na samom početku kreiranje aplikacije, bilo bi dobro spojiti se sa bazom podataka koja je kreirana. Kako Microsoft Visual Studio ne podržava rad sa PostgreSQL bazom podataka, morat će se preuzeti neki dodaci kako bi se to omogućilo. Sa službene stranice

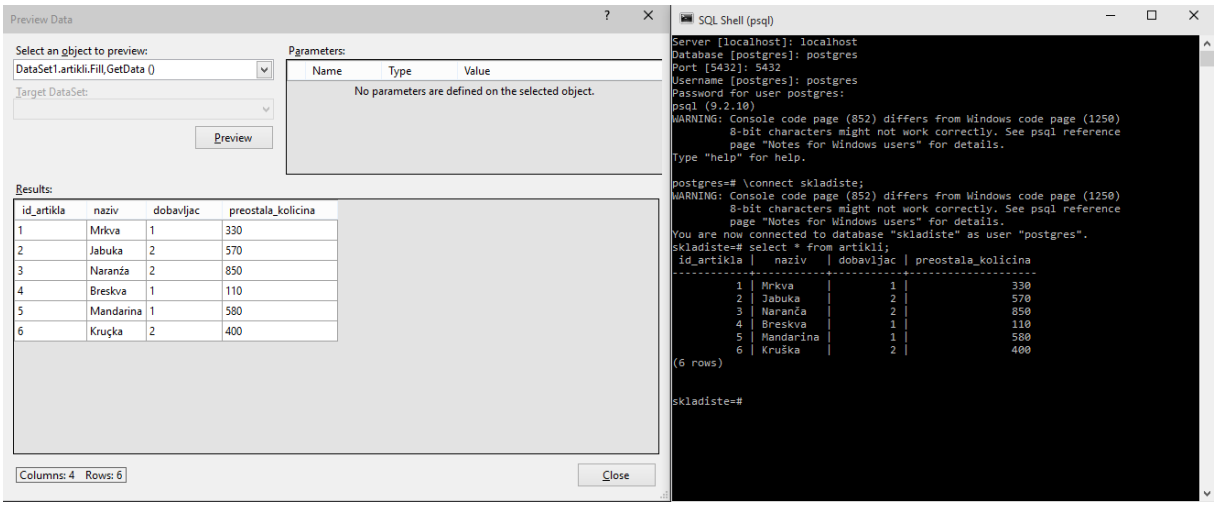
sustava PostgreSQL [8] preuzima se najnoviji dodatak psqlodbc.zip. Nakon što se raspakira .zip dokument, pokreće se *psql-odbc setup* te instalira dodatak. Nakon što se to napravi, može se spojiti sa bazom koja je kreirana. *U data sources* tabu koji se nalazi na lijevoj strani od forme koju se kreira, klikne se na gumb *Add New Data Source*, odabire se Database->Dataset te se klikne na *New Connection*. Kao *data source* odabire se .NET Framework Data Provider for ODBC, a kao *Data source specification* odabire se PostgreSQL te se klikne *ok*. Nakon toga nas sustav traži da se unesu podaci te se popunjavaju podaci kao i kod prijave u sustav PostgreSQL, *username*, *host*, *password*, broj porta i drugo. Nakon toga, testira se konekcija te ako je uspješna, povezivanje sa bazom je obavljeno i može se nastaviti sa radom i dohvaćati podatke iz tablica. Slika 3.13. prikazuje kako izgleda dodavanje nove konekcije na bazu iz koje će se podaci dohvaćati.



Slika 3.13. *Dodavanje nove konekcije na bazu*

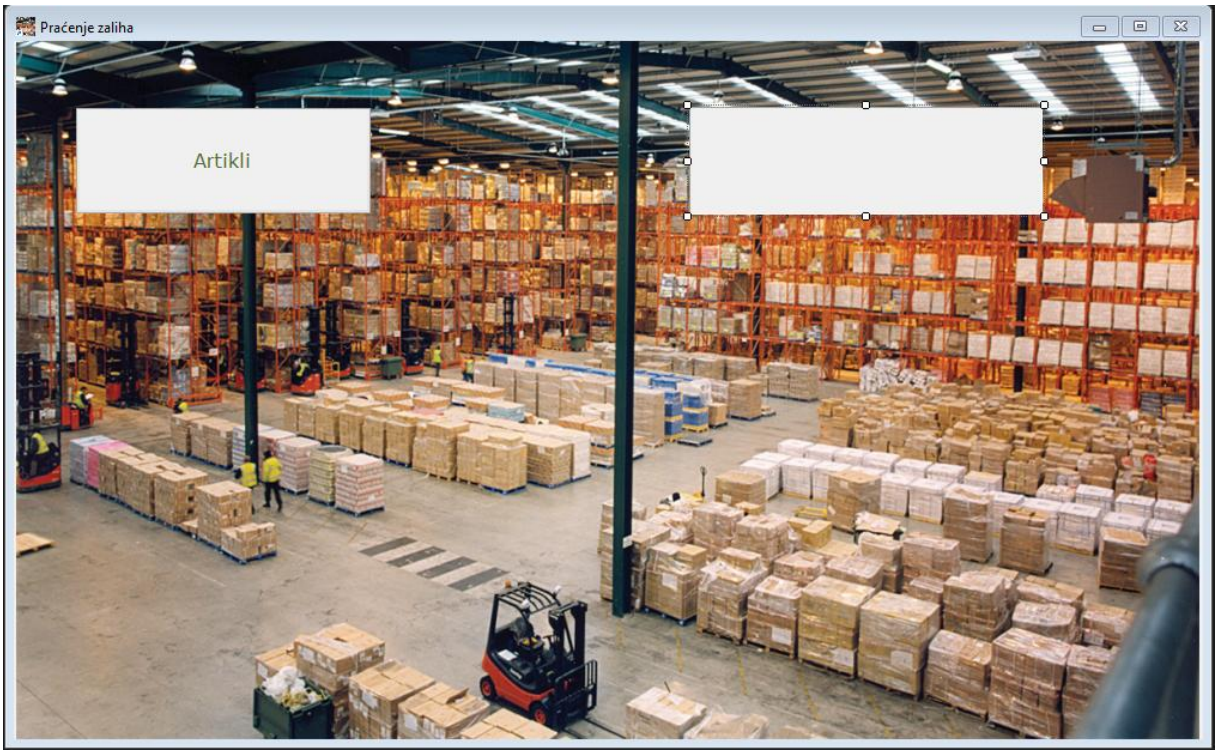
Nakon što se spoji na bazu, u tabu *Data Sources* može se vidjeti naziv konekcije kojoj je dano ime *Dataset* te sve stavke iz kreirane baze. Trenutno se vide samo tablice jer nisu kreirani nikakvi pogledi ili okidači (engl. *Trigger).* Desnim klikom na neku od tablica i odabirom opcije *preview* može se vidjeti da smo stvarno povezani na bazu i iščitavaju se podaci koji su prethodno uneseni. Sada će se prikazati paralelno podaci u PostgreSQL-u te u

preview data prozoru Microsoft Visual Studia kako bi se uvjerili da su podaci koji su dohvaćeni zaista podaci iz baze.



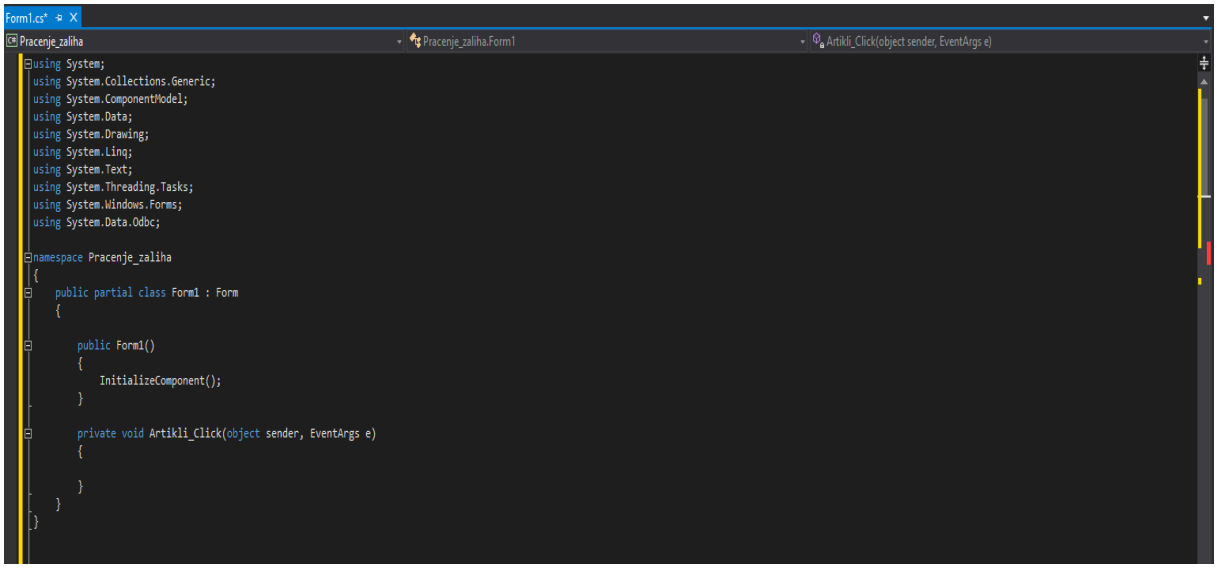
Slika 3.14. *Podaci prikazani u PostgreSQL-u i Microsoft Visual Studiu*

Može se početi sa uređivanjem osnovne forme koja će se prikazati pri pokretanju aplikacije. Za početak će se napraviti jednostavnije stvari, a nakon toga nešto složenije. Dodaje se pozadinska slika na našu formu koja će ju učiniti ugodnijom za rad. Klikne se na formu te u prozoru *properties* pod karticom *BackgrounImage* se klikne na tri točkice i otvara se prozor za dodavanje slike, odabere se *Local Resource* i *Import* te neka od slika lokalno pohranjenih na računalo. Nakon dodavanja pozadinske slike u kartici *properties* mogu se urediti još neke opcije kreirane forme. Postavljaju se *BackgrounImageLayout* na *Zoom*, *Text* će se promijeniti u *Praćenje zaliha*, *Language* se postavlja na „Croatian“, *StartPosition* se postavlja kao *CentralScreen* te će se dodati ikona koja je prethodno izrađena posebno za ovu aplikaciju. U *View* izborniku se odabire *Toolbox* (ili se pritisne Ctrl + Alt + X na tipkovnici) te se prikazuje prozor sa alatima koji se mogu ubaciti u Windows formu. Ubacuje se gumb (*button*) i labela (l*abel*) jednostavnim *drag-and-drop* postupkom. Nakon što se promijene neke od opcija gumba i labele i nakon svih do sad objašnjenih promjena, forma izgleda kao na slici 3.15.



Slika 3.15. *Početni izgled naše forme*

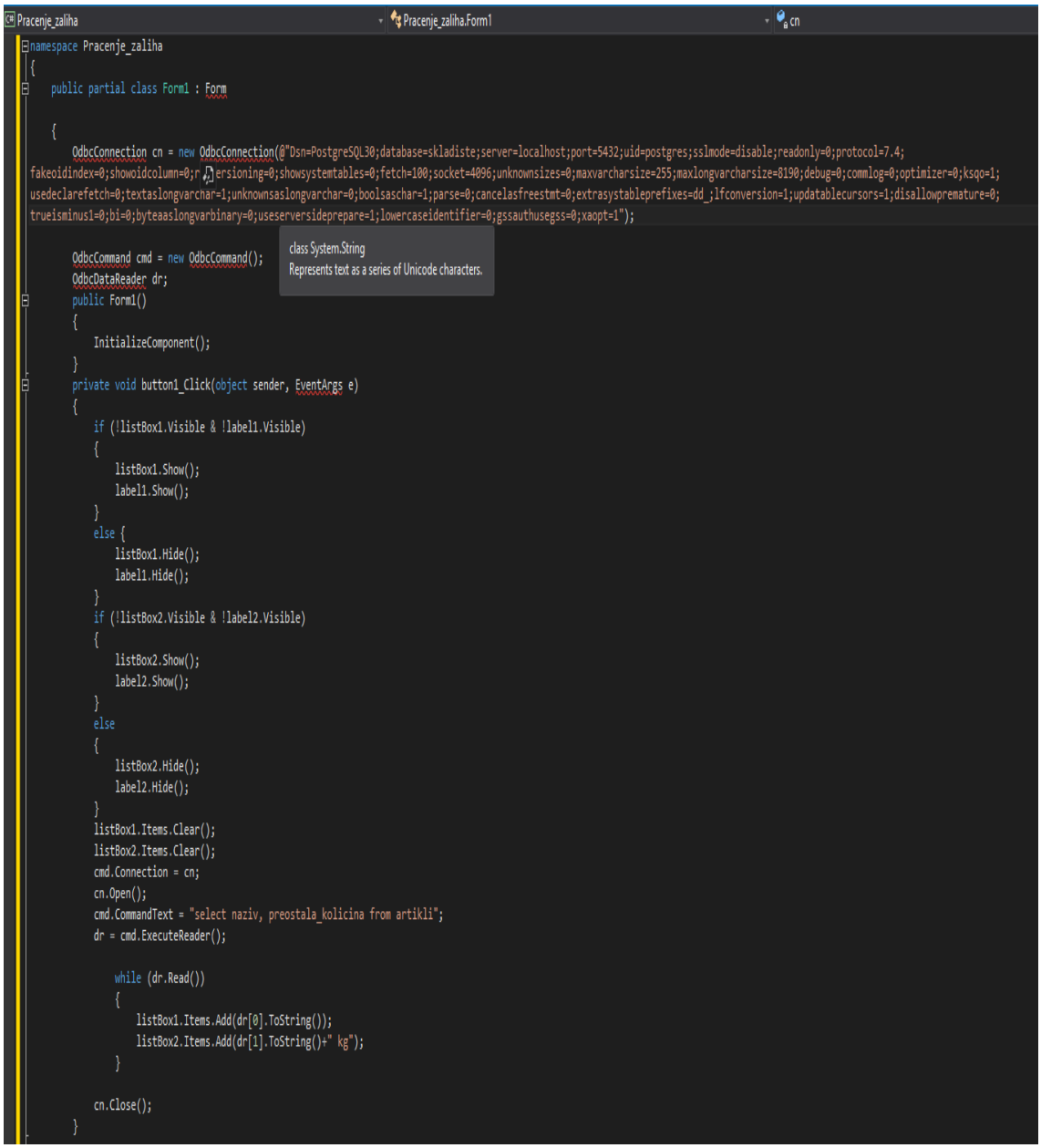
Gumbu se dodaje ime *Artikli* i klikom na taj gumb želimo da se u labelu na desnoj strani pokažu artikli koje imamo na skladištu. Može se uočiti u gornjem lijevom uglu naziv forme koja je nazvana *Pracenje_zaliha*. Želimo da aplikacija funkcionira na način da klik na gumb omogući ispis ažurnih podataka i prikaže stvarno stanje zaliha na skladištu. Kao što je već rečeno, veza sa bazom podataka je uspostavljena i sada je potrebno dohvaćati podatke iz tablice. Klikom na karticu *Form1.cs* prikazuje se cjelokupni kod koji se nalazi u pozadini forme.



Slika 3.16. *Početni kod forme*

Ovo je osnovni kostur koji se generira pri samom kreiranju nove forme. Ovaj kod će se dalje nadograđivati funkcijama potrebnim za aplikaciju. Ključna riječ *using* znači da uključujemo biblioteke, a „;“ je obavezna nakon završetka naredbe, no nije kod definiranja funkcije. Unutar funkcije *Artikli_Click* definiramo što će se dogoditi kada se klikne na gumb *artikli*.

Sljedeće što se radi je dodavanje funkcionalnosti gumbu. Neke od funkcionalnosti mogu se dodati u kartici opcije (*properties*) kao što je font, veličina gumba, veličina fonta, pozadinska boja itd. Dvostrukim klikom na gumb otvara se kod gumba te će se urediti kako bi gumb imao funkcionalnost koju želimo. Klikom na gumb *artikli* želimo popis svih artikala i preostalu količinu pojedinog artikla. Prvo što će se napraviti je uključiti biblioteku *System.Data.Odbc* gdje kratica *Odbc* označava *Open database conectivity*. Ova datoteka je potrebna kako bi se moglo raditi sa podacima iz baze podataka. Nakon toga će se kreirati konekcija na bazu te unutar koda napisati potrebne naredbe kako bi omogućili funkcionalnost koju želimo. Nakon što je unesen cijeli kod, on izgleda ovako:



Slika 3.17. *Kod forme za funckionalnost gumbova*

Unutar vitičastih zagrada koje slijede nakon *public partial class Form1:Form* definira se što se događa sa formom. Ovdje imamo *OdbcConnection* koja je nazvana *cn*, *operatorom new* kreira se nova konekcija, a u zagradama se navode parametri potrebni za spajanje na bazu. Nakon toga, definira se *Odbc* naredba i *Odbc* čitač podataka. Unutar vitičastih zagrada nakon *private void button1_click(object sender, EventArgs e)* definiraju se naredbe koje će se izvršiti klikom na gumb naziva *button1*. Za početak želimo da se pokažu *listBox1* u kojem će biti prikazan naziv artikla i *label1* za kojeg je tekst u *properties* labela definiran kao *Artikli*,

odnosno kada se ponovno klikne želimo da se *listBox1 i label1* maknu sa ekrana. To će se postići sljedećim kodom:

```
Pracenje_zaliha
if (!listBox1.Visible & !label1.Visible)
            {
                listBox1.Show();
                label1.Show();
            }
            else {
                listBox1.Hide();
                label1.Hide();
            }
```

Slika 3.18. *Kod za prikaz i skrivanje listbox-a i labela*

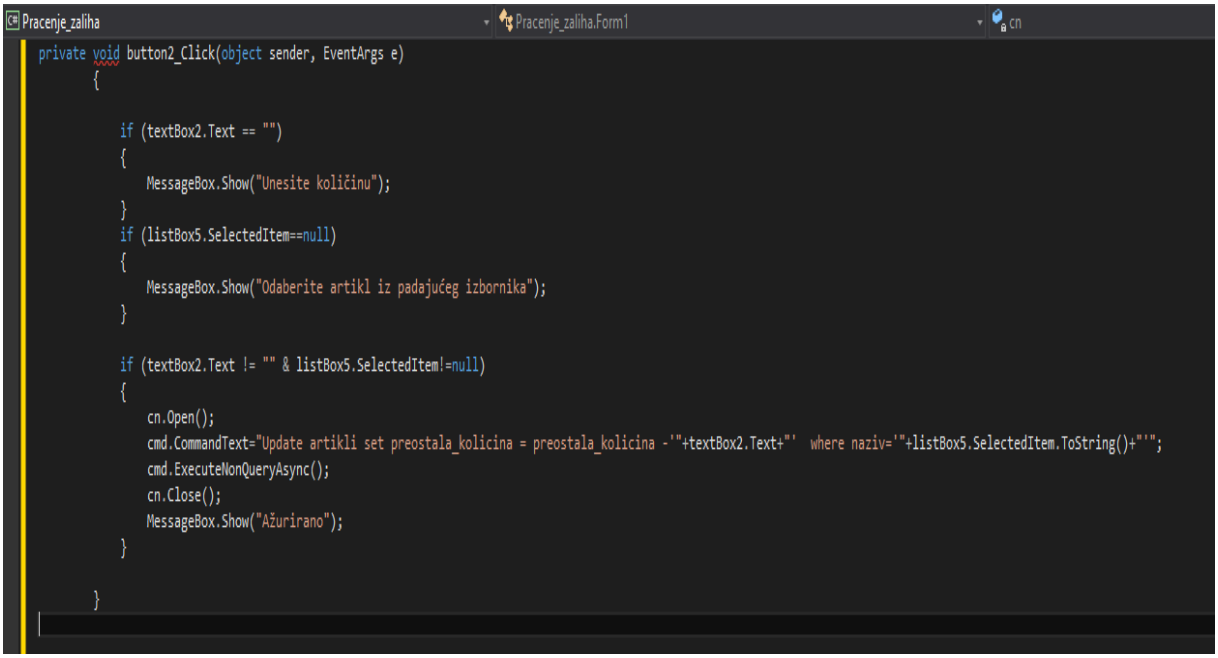
Analogno se radi i za *listBox2 i label2* koji će prikazivati preostalu količinu. Naredbama *listBox1.Items.Clear() i listBox2.Items.Clear()* briše se sadržaj ako bi neki sadržaj već bio prikazan u *listBox1 i listBox2*. Ovaj dio koda se mora unijeti kako bi se sadržaj svaki put osvježio i kako se ne bi prikazivali stari podaci koji su se možda već prije pregledavali.

```
Pracenje_zaliha
cmd.Connection = cn;
 cn.Open();
            cmd.CommandText = "select naziv, preostala_kolicina from artikli";
            dr = cmd.ExecuteReader();
                while (dr.Read())
                {
                    listBox1.Items.Add(dr[0].ToString());
                    listBox2.Items.Add(dr[1].ToString()+" kg");
                }
            cn.Close();
```

Slika 3.19. *Kod za dodavanje podataka na listbox*

Naredbom *cn.Open()* otvara se nova konekcija na našu bazu. U *cmd.CommandText* unosi se SQL upit koji će se izvršiti naredbom *cmd.ExecuteReader()*. Nakon toga, uz pomoć *while* petlje, ispisuje se artikl te njegova preostala količina. *While dr.Read* znači da će se artikli ispisivati dok postoje u tablici, a *listBox1.Items.Add(dr[0].ToString()* pretvara podatke iz tablice u *string* i prosljeđuje *listBox1*, odnosno *listBox2*. Kod ispisa kilograma dodaje se još dodatak „kg“ kako bi ispis bio ljepši.

Nakon što je napravljen gumb koji će ispisivati artikle i preostalu količinu pojedinog artikla na skladištu, napravit će se gumb kojim će se otpremiti artikle u proizvodnju. Klikom na ovaj gumb, radnik će moći odabrati artikl iz padajućeg izbornika te unijeti količinu potrebnu za proizvodnju. Nakon što se klikne na gumb *potvrdi*, u bazi podataka će se artikl koji je odabran iz padajućeg izbornika smanjiti za količinu koja je navedena. Većina koda za gumb kojim će se moći otpremiti artikle u proizvodnju je identičan kodu za popis artikala jer će nam trebati popis kako bi mogli odabrati artikl. Dodajemo *button2* i *textBox* u našu formu, *textBox* će nam služiti za unos količine, a klikom na *button2* će se izvršiti upit koji će u bazi podataka umanjiti količinu artikla koji je odabran iz padajućeg izbornika i to za onu količinu koju ćemo unijeti u *textBox*.



Slika 3.20. *Kod za funkcije button2 gumba*

Sada će se objasniti kod koji će se izvršiti klikom na *button2*. Ukoliko je *textBox2* prazan, ispisat će se poruka *Unesite količinu*. Ukoliko iz *listBox5* nije odabran artikl, ispisat će se *Odaberite artikl iz padajućeg izbornika*. Ove konstrukte potrebno je postaviti unutar *if* uvjeta kako se ne bi upit izvršio bez da smo unijeli sve podatke. Ako su podaci uneseni, otvara se nova konekcija. Pod *CommandText* definira se naredba te izvršava upit i zatvara konekcija uz poruku *Ažurirano*.

Update artikli set preostala_kolicina=preostala_kolicina-'"+textBox2.Text+" where naziv='"+listBox5.SelectedItem.ToString()+"'" je SQL naredba koja će ažurirati podatke u tablici na način da će preostalu količinu umanjiti za količinu koja je unesena u *textBox2* gdje je odabrani artikl jednak nazivu artikla u tablici. Kako bi mogli usporediti artikl iz padajućeg izbornika i artikl iz tablice u bazi, potrebno je izabrati artikl iz padajućeg izbornika i pretvoriti u *string* funkcijom *ToString()*.

Sada se dodaje još jedan gumb koji će ispisati tri artikla koji imaju najmanju količinu zaliha. Kod je gotovo identičan kodu gumba *button1*, jedino se razlikuje SQL naredba te će se rezultat ispisati u drugi *listbox*.

Pracenje_zaliha

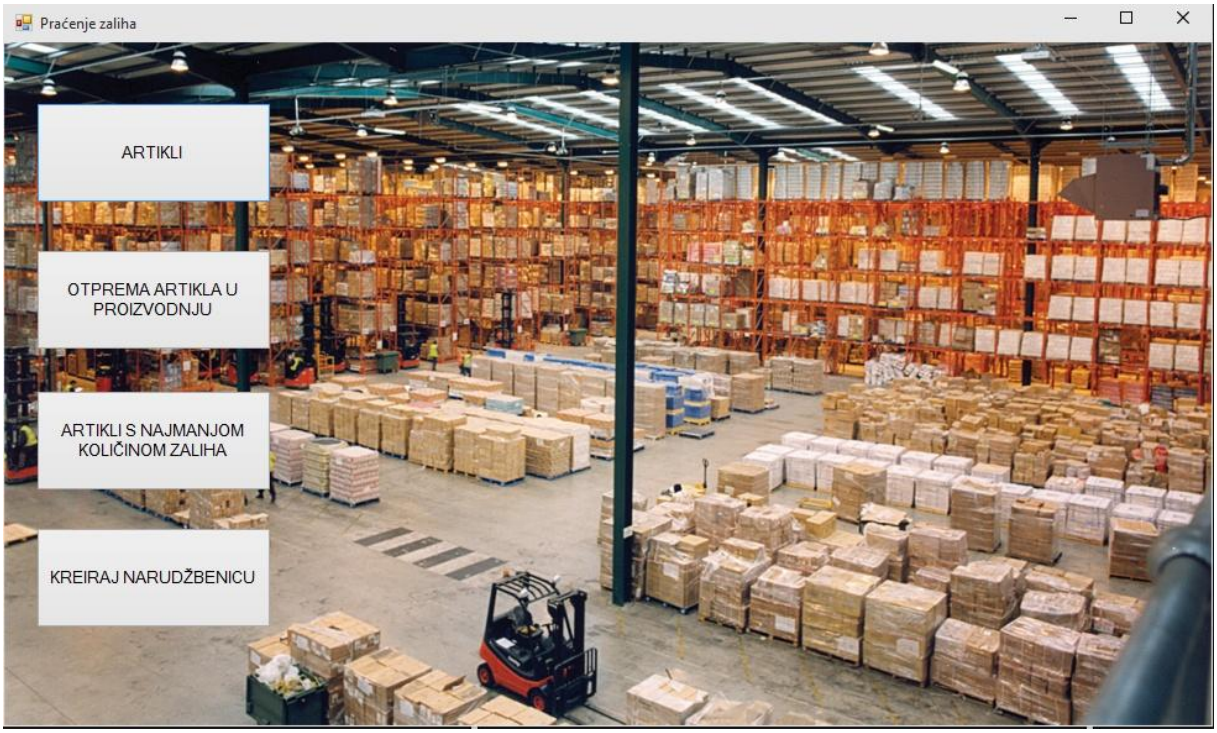
```
listBox3.Items.Clear();
            listBox4.Items.Clear();
            cmd.Connection = cn;
            cn.Open();
cmd.CommandText = "select naziv, preostala_kolicina from artikli order by preostala_kolicina limit 3";
            dr = cmd.ExecuteReader();

            while (dr.Read())
            {
                listBox3.Items.Add(dr[0].ToString());
                listBox4.Items.Add(dr[1].ToString()+" kg");
            }
```

Slika 3.21. *Kod kojim prikazujemo tri artikla s najmanjom količinom zaliha*

Select naziv, preostala_kolicina from artikli order by preostala_kolicina limit 3 će izabrati naziv i preostalu količinu iz tablice *artikli* te ispisati tri artikla sa najmanjom preostalom količinom. Nakon toga popunjava se *listbox3* i *listbox4* sa podacima identično kao i kod gumba za ispis artikala.

Nakon kreiranja svih gumba i funkcionalnosti aplikacije, klikne se na opciju *debug* i zatim *start debugging*. Ukoliko ima nekih grešaka, one će se ispisati te će nas sustav upozoriti u kojem redu u kodu se greška nalazi te ih je tako jednostavno uočiti i ispraviti. Kada se napravi *debug* aplikacije, ona je spremna za korištenje. Forma koja je kreirana nalazi se u *bin folderu* aplikacije na lokaciji koja je odabrana na početku pri kreiranju windows forme. Korisničko sučelje razvijene aplikacije izgleda kao na slici 3.22.



Slika 3.22. *Korisničko sučelje razvijene aplikacije*

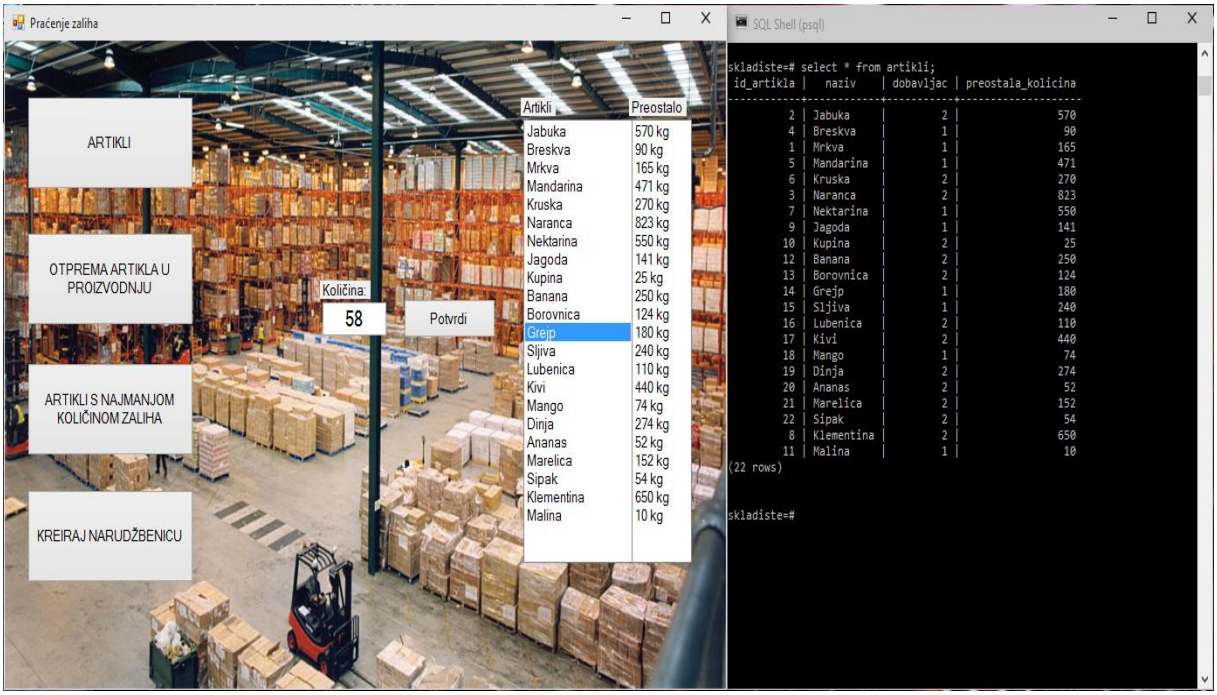
Klikom na gumb *artikli* dobiva se popis artikala kao što se može vidjeti na slici 3.23.

Slika 3.23. *Popis artikala u aplikaciji*

Klikom na gumb *otprema artikla* u proizvodnju dobiva se sljedeći prikaz:

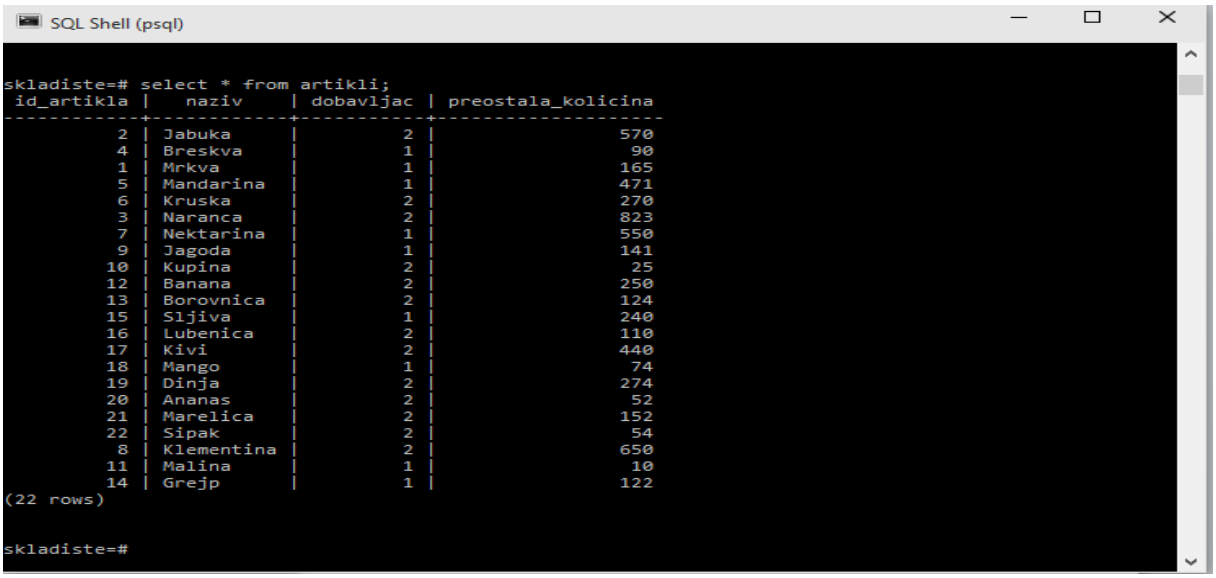
Slika 3.24. *Otprema artikla u proizvodnju*

U polje *količina* unosi se količina koja se želi otpremiti u proizvodnju te se iz padajućeg izbornika izabire artikl kojeg se želi otpremiti. Nakon što se klikne na gumb *potvrdi,* podaci će se ažurirati u aplikaciji, ali i u bazi podataka.



Slika 3.25. *Stanje zaliha prije izvršavanja upita*

Slika 3.25. prikazuje stanje zaliha prije izvršavanja upita. Nakon što se odabere „Grejp“ i unese količina „58“, klikne se na gumb *potvrdi*. Stanje u tablici se mijenja što se vidjeti i na slici 3.26.



Slika 3.26. *Stanje zaliha nakon izvršavanja upita*

Klikom na gumb *Artikli s najmanjom količinom zaliha*, prikazuju se tri artikla koji imaju najmanju količinu te kolika je ta količina.

Slika 3.27. *Artikli s najmanjom količinom zaliha*

Klikom na posljednji gumb odabire se dobavljač te se kreira narudžbenica koja se zatim dostavlja dobavljaču.

4. SLOŽENI UPITI

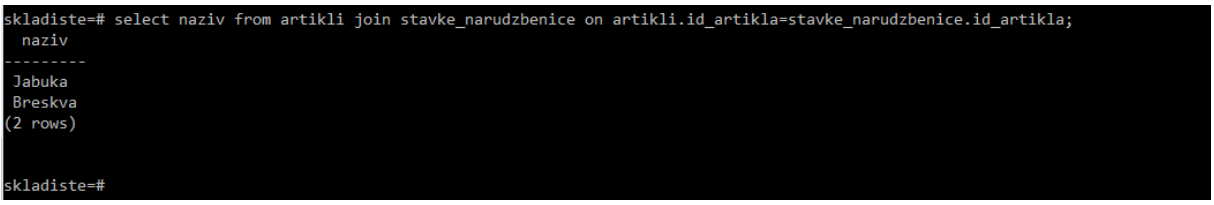
Kao što jednostavni upiti ispisuju podatke iz jedne tablice, tako složeni upiti vraćaju podatke iz više tablica kao rezultat upita. Naravno da su jednostavni upiti važni, no često nam trebaju podaci iz dvije, tri ili više tablica. Postoji nekoliko načina na koji možemo „izvući“ podatke iz više tablica. Korištenjem klauzula *from* i *where* koje su ranije opisane, može se na jednostavan način ispisati podaci iz tablica. Također se koristi i klauzula *join*. Kod korištenja klauzule *join* potrebno je pravilno definirati primarne i vanjske ključeve jer u suprotnome nećemo biti u mogućnosti iščitavati podatke iz tablica. Osnovna naredba *join* ima sljedeću sintaksu: *Select * from tablica1 join tablica2 on tablica1.pk=tablica2.vk* gdje *pk* i *vk* predstavljaju primarne i vanjske ključeve. Prednost ovakvog korištenja je odvojenost uvjeta za spajanje tablica i filtriranje redova. Kako bi prikazali korištenje jednostavne klauzule *join* potrebno je kreirati još jednu tablicu u našem sustavu koji je već prikazan na ER dijagramu, a to je tablica *stavke_narudzbenice*. Tablica se sastoji od složenog primarnog ključa koji se sastoji od atributa *id_narudzbenica* i *id_artikla* te *kolicine*. Ova tablica je potrebna u sustavu kako bi se jednom narudžbenicom moglo naručiti više artikala. Primarni ključevi u tablici *stavke_narudzbenice* su vanjski ključevi na tablice *artikli* i *narudzbenice*. Nakon kreiranja tablice i popunjavanja podacima, tablica izgleda kao na slici 4.1.

```
 id_narudzbenice | id_artikla | kolicina
-----------------+------------+----------
               1 |          2 |      300
               2 |          4 |      200
(2 rows)


skladiste=#
```

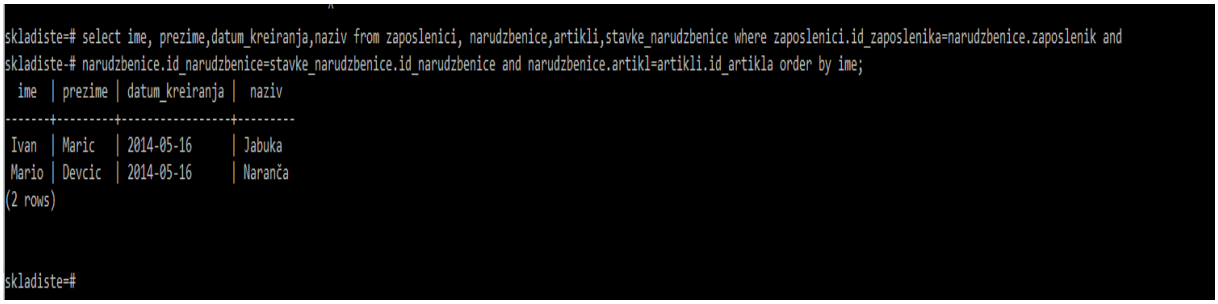
Slika 4.1. *Tablica narudžbenice*

Može se vidjeti da su u tablici podaci koji nisu sasvim jasni. Navedeni su *id_narudzbenice, id_artikla* i *kolicina,* ali ne zna se tko je naručio robu, od kojeg dobavljača i koji je artikl naručen. Kako bi to saznali, koristi se upit *join* prikazan na sljedećoj slici 4.2.



Slika 4.2. *Korištenje join klauzule*

Sada će se kreirati upit kojim će se iz tablice *stavke_narudzbenice* saznati tko je naručio robu, koju robu je naručio i kojeg datuma je roba naručena. Sljedeća slika 4.3. prikazuje kako implementirati ovakav upit.



Slika 4.3. *Složeni upit*

Također je moguće koristiti uz klauzulu *join* uvjet *using*. Kada se koristi *using*, u zagradi se mora navesti koji su stupci zajednički u obje tablice i time se govori sustavu da prema tim stupcima želimo spajati tablice. *Natural join* je način spajanja dvije ili više tablica kod kojeg u svim tablicama iz kojih spajamo podatke postoje stupci istog naziva. Spajanje tablice same sa sobom je također moguće, no rijetko se primjenjuje. Do sada smo uvijek kod spajanja uspoređivali dva stupca tablice znakom jednakosti („=“), no postoji i drugi načini kao što je nejednakosno spajanje. Može se dogoditi situacija da trebamo podatke iz jedne tablice koji ne postoje u drugoj tablici. Moguće je spajanje tablica i po drugim operatorima uspoređivanja. Kada se već govori o spajanjima tablica, spomenimo i Kartezijev produkt koji nam daje rezultat takav da svaki redak jedne tablice spoji sa svakim retkom druge tablice. Kartezijev produkt se također rijetko koristi.

5. USPOREDBA RAZVIJENOG SUSTAVA S POSTOJEĆIM RJEŠENJIMA NA TRŽIŠTU

U današnjem svijetu razvijene tehnologije nije moguće opstati ukoliko se ne prate trendovi. Ako bi samo nekoliko godina zaostajali za našom konkurencijom može se reći da bi vrlo vjerojatno i propali. U takvom okruženju potrebno je pratiti trendove te je nemoguće opstati bez adekvatne potpore poslovnom sustavu. Ogromna skladišta koja posjeduju srednje i velike tvrtke jednostavno ne mogu funkcionirati bez informacijskog sustava koji im pruža potporu. Najveći problem kod takvih sustava je njegova cijena. Iako velikim tvrtkama nije problem uložiti veću količinu novca u razvoj informacijskog sustava, manjim tvrtkama su to ipak preveliki izdaci. Cijene nekih jednostavnih sustava koji služe samo za praćenje zaliha u skladištu kreću se od 400 pa čak i do 1000 dolara mjesečno. Primjer jednog takvog sustava je Warehouse Managment Software tvrtke 3PL Central [9]. Uz standardnu uslugu praćenja zaliha, aplikacija pruža prilagođen softver za tvrtku naručitelja, integraciju elektroničke razmjene podataka te mogućnost naručivanja od nekih dobavljača. Ovakav način rješavanja problema praćenja zaliha u skladištu je vrlo jednostavan, ali zato i skup. Alternativni način je kreirati aplikaciju kakva je kreirana u ovome radu, a glavni razlog je cijena. Aplikacija kreirana u radu može biti prilagođena potrebama bilo kojeg sustava za praćenje zaliha. Baza podataka je kreirana u sustavu za upravljanje bazom podataka PostgreSQL koji je u potpunosti besplatan i jednostavan za korištenje, dok je aplikacija kreirana u Microsoft Visual Studiu koji nudi besplatnu verziju od 30 dana, no moguće je koristiti i druge sustave za kreiranje aplikacije koje možemo preuzeti te instalirati na svoje računalo potpuno besplatno. Ovakve aplikacije preporučene su malim i srednjim poduzećima kako zbog cijene skupih sustava, tako i zbog podrške koju pružaju sustavi koje kupimo i plaćamo svaki mjesec. Ovakvi sustavi nisu u potpunosti pouzdani za velike tvrtke, jer ukoliko bi došlo do gubitka podataka, moglo bi doći do velikih financijskih gubitaka tvrtke.

Drugo postojeće rješenje na tržištu je SAP sustav (engl. *System, Applications, and Products in Data Processing*) [10]. SAP je zapravo kompanija osnovana 1972. godine sa sjedištem u Walldorfu u Njemačkoj, a bavi se razvojem i prodajom integriranih informacijskih rješenja za podršku u poslovanju. SAP je treća najveća softver kompanija u svijetu te vodeći svjetski dobavljač e-business rješenja koja integriraju poslovne procese unutar i među poduzećima i poslovnim zajednicama. SAP nije samo IT sustav ili ERP program, SAP je poslovni sustav tj. "know - how" koji pruža najbolje poslovne prakse i

iskustvo najboljih korporacija. SAP je dizajniran za međunarodno poslovanje i organizacije s velikim ambicijama i rastom. SAP rješenje ne ograničava već podržava. To je vrhunski njemački inženjering koji karakterizira cjelovitost i robusnost te primjena najboljih tehnologija. SAP je apsolutni broj 1 u svijetu poslovnih aplikacija, osmišljen i izveden na totalnu integraciju kupaca, partnera i dobavljača. Također treba napomenuti kako su SAP sustavi pogodniji za veće tvrtke gdje je potrebno pratiti puno veći opseg poslova i poslovnih procesa. Naravno, tu je i pitanje financija pa poduzeće treba donijeti odluku o upotrebi skupih softvera samo onda kad je to nužno za poslovanje kako poduzeće ne bi imalo velike gubitke.

6. ZAKLJUČAK

Kako su se kroz godine razvijala poduzeća, tako se paralelno s njihovim razvojem razvijala potreba za informacijskim sustavom koji će im biti podrška pri odvijanju svakodnevnih procesa. Kao što se može zaključiti iz ovog rada, jedna od osnovnih stavki informacijskog sustava je baza podataka koja sadrži sve relevantne podatke potrebne za poslovanje i rad poslovnog sustava. SQL (engl. *Structured Query Language*) je najpopularniji i najkorišteniji jezik za rad s bazom podataka, a u današnje vrijeme koriste se relacijske baze podataka čija je osnova tablica, odnosno relacija. Tablice u bazi podataka povezane su uz pomoć primarnih i vanjskih ključeva. Također je važno kreirati bazu koja neće sadržavati redundantne podatke, a to se postiže pravilnim kreiranjem baze podataka od konceptualnog pa sve do fizičkog oblikovanja. Podaci koji se nalaze u tablicama odgovaraju podacima u stvarnom poslovnom sustavu te je kvalitetna baza podataka dobra podloga za izgradnju pouzdanog i kvalitetnog informacijskog sustava. Sustav za upravljanje bazom podataka opisan u ovome radu je PostgreSQL koji je u potpunosti besplatan i jednostavan za korištenje. U ovome radu je također kreirana aplikacija koja prati kretanje zaliha u fiktivnom poduzeću. Aplikacija je kreirana u alatu Microsoft Visual Studio koji nije besplatan, no postoje besplatni alati uz pomoć kojih se može kreirati jednaka aplikacija. Iako sustav za upravljanje bazom podataka PostgreSQL i Microsoft Visual Studio nisu kompatibilni, moguće je spojiti se na kreiranu bazu podataka uz instaliranje besplatnog dodatka npgsql. Kako je sustav za upravljanje bazom podataka besplatan, prikazano rješenje je jako dobro i kvalitetno te se može nadograditi za potrebe nekih stvarnih poduzeća. U radu je također objašnjen rad u alatu Microsoft Visual Studio te rad sa gumbima, povezivanje na kreiranu bazu podataka i drugo.

Mjesto i vrijeme: Potpis studenta:

LITERATURA

[1] K. Rabuzin, SQL Napredne teme, FOI Varaždin, 2014.

[2] http://mysqlworkbench.org/, kolovoz 2015.

[3] https://en.wikipedia.org/wiki/Three-valued_logic, svibanj 2015.

[4] K. Rabuzin, Uvod u SQL, FOI Varaždin, 2011.

[5] Z. Skočir, I. Matasić, B. Vrdoljak, Organizacija obrade podataka, Merkur, Zagreb 2007.

[6] R. Manger, Baze podataka, Element, Zagreb 2014.

[7] https://www.visualstudio.com/, kolovoz 2015.

[8] http://www.postgresql.org/, kolovoz 2015.

[9] http://3plcentral.com/, kolovoz 2015.

[10] http://go.sap.com/index.html, kolovoz 2015.