SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Tomislav Bobinac

APLIKACIJA ZA UPRAVLJENJE SKLADIŠTEM TEMELJENA NA AKTIVNIM BAZAMA PODATAKA

DIPLOMSKI RAD

Varaždin, 2017.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Tomislav Bobinac

Matični broj: 42619/13-R

Studij: Baze podataka i baze znanja

APLIKACIJA ZA UPRAVLJENJE SKLADIŠTEM TEMELJENA NA AKTIVNIM BAZAMA PODATAKA

DIPLOMSKI RAD

Mentor:

Izv.prof.dr.sc. Markus Schatten

Varaždin, rujan 2017.

Sadržaj

1. Uvod

Diplomski rad se sastoji od tri dijela. Prvi dio se sastoji od teorije vezane uz baze podataka i sustave za upravljanje bazama podataka. Naglasak je stavljen na aktivne baze podataka koje će se kasnije koristiti u praktičnom dijelu diplomskog rada. Drugi dio sadrži teoriju o skladištu podataka i njegovom upravljanju, te zadnji dio je prikaz aplikacije za upravljanje skladištem podataka temeljene na aktivnim bazama podataka.

Skladište je određeni prostor namijenjen čuvanju i smještaju robe koja je predmet poslovanja preduzeća. Ubrzani razvoj tržišta doveo je do različitih shvaćanja uloge skladištenja, te ono preuzima sve veću ulogu u cjelokupnom opskrbnom lancu, te se osim osnovnih skladišnih operacija poput prijema, uskladištenja, otpreme obavlja i niz drugih usluga. Da bi skladište ispunilo zahjeve menadžmenta i zadovoljilo zahtjeve korisnika treba koristiti tehnologije koje omogućuju jednostavniju kontrolu i rukovanje skladištem, a to sve omogućuje sustav za upravljanje skladištem, o kojem je i riječ u ovom radu.

Sustav za upravljenje skladištem treba omogućiti konstanto praćenje cjeloukupnog stanja skladišta, te pravovremeno reagirati na novonastale situacije. Takvo ponašanje nam omogućuju aktivne baze podataka. One automatski reagiraju na događaje koji se odvijaju unutar ili izvan baze podataka. Primjerice sustav za upravljanje zalihama treba pratiti količinu robe na skladištu, tako da se kada količina određene stavke padne ispod praga aktivira određena aktivnost. Ukratko, aktivne baze podataka proširuju baze podataka sa sposobnošću reaktivnog ponašanja.

U sljedećim poglavljima detaljno je pojašnjena i uloga aktivnih baza podataka i sustava za upravljenje skladištem.

2. Baza podataka

Baze podataka i sustavi baza podataka su bitan sastavni dio života u modernom društvu. Većina nas se svakodnevno susreće s aktivnostima koje uključuju neku interakciju s bazom podataka. Bilo da idemo u banku uplatiti ili povući sredstva s računa , ili prilikom rezervacije hotela ili aviona, postoji velika šansa da će ta akcija uključiti i sudjelovanje računalnog programa koji pristupa bazi podataka (Elmasri & Navathe, 2010).

Baza podataka je dobro organizirana, računalno čitana, najčešće relacijski povezani skup podataka. Primarne zadaće sutava baze podataka su pohrana, integritet i interpretacija podataka. Baze podataka pohranjuju podatke poslovanja neke organizacije i kao rezultat pohrane sadrže veliki broj podataka koje je potrebno dugoročno skladištiti. Podaci su pohranjeni na način neovisan o programima koji ih koriste.

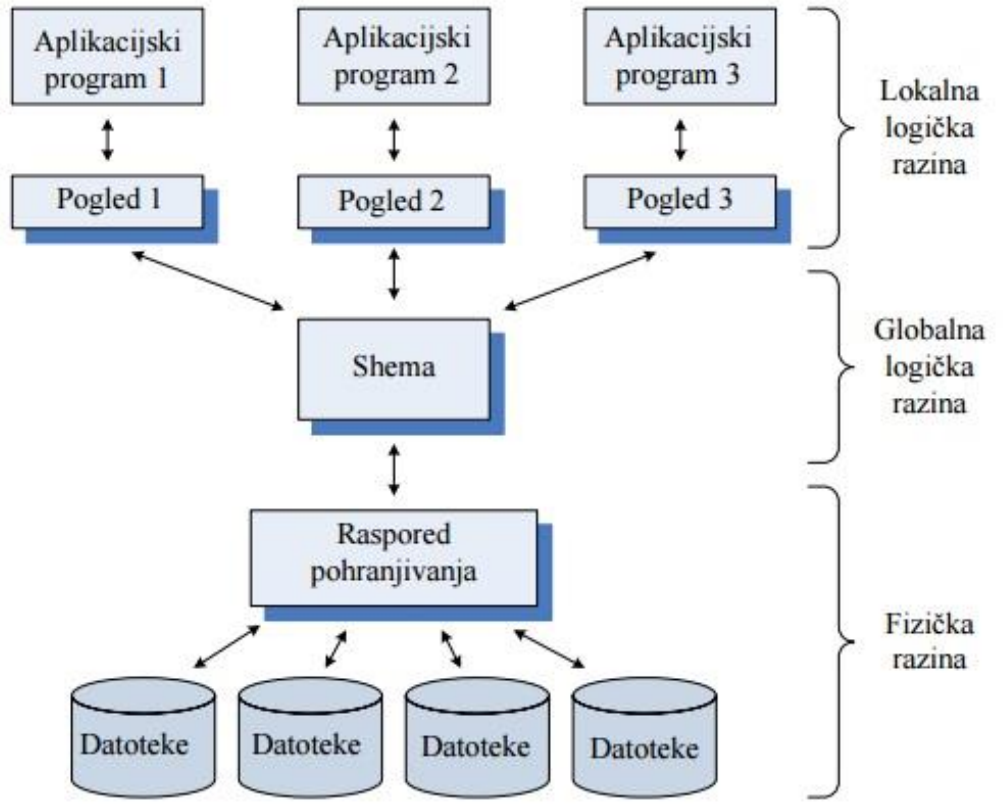
Od svog dolaska, baze podataka su među najistraživanijim područjima u informatici. Baza podataka je repozitorij podataka, dizajnirana da podupire unčikovitu pohranu, dohvaćanje i održavanje podataka. Postoji više tipova baza podataka koji odgovaraju različitim zahtjevima poslovanja. Baza može biti specijalizirana za spremanje binarnih datoteka, dokumenata, slika, videa, transakcijskih podataka, analitičkih podataka , geografskih podataka itd.

Podaci mogu biti pohranjeni u različitim oblicima, a to su tablični, hijerarhijskih i mrežni oblik. Ako se podaci pohranjuju u tabličnom obliku onda se radi o relacijskoj bazi podataka. Kada se podaci pohranjuju u obliku strukture stabla onda se radi o hijerarhiskoj bazi podataka, a mrežna baza podataka je kada su pohranjeni podaci grafovi koji predstavljaju odnose između objekata (Sharma, et al., 2010).

3. Arhitektura baze podataka

Arhitektura baze podataka sastoji se od tri razine apstrakcije. Razine apstrakcije su (Baranović & Zakošek, 2012):

- Fizička razina
- Globalna logička razina
- Lokalna logička razina



Slika 1: Arhitektura baze podataka (Izvor: Robert Manger, 2010)

3.1. Fizička razina

Fizička razina je najniža razina apstrakcije koja opisuje kako su podaci zapravo pohranjeni i na koji način im se može pristupati. Ona detaljno opisuje strukturu podataka. Fizička razina je sakrivena ispod globalne logičke razine, i obično se može jednostavno mijenjati bez da utječe na aplikacijske programe. Kaže se da aplikacijski programi posjeduju fizičku neovisnost podataka, te zbog toga se podaci ne trebaju prepravljati ukoliko dođe do promjene fizičke razine odnosno izmjena fizičke razine ne utječe na globalnu logičku razinu (Silberschatz, et al., 2010).

3.2. Globalna lokalna razina

Ova razina sadrži opis svih entiteta i veza, atributa i domena i integritetska ograničenja. Opis globalne logičke razine naziva se shema. Shema je tekst ili dijagram koji definira logičku strukturu baze i u skladuje sa zadanim modelom. Ona opisuje kakvi podaci su pohranjeni u bazi, te njihovu međusobnu povezanost. Logička razina opisuje cijelu bazu podataka u smislu manjega broja jednostavnih struktura. Premda implementacija jednostavnih struktura može uključivati složene fizičke razine strukture, korisnik logičke razine ne mora biti svjestan te složenosti. Kao što smo već spomenuli, to se naziva fizička neovisnost podataka (Silberschatz, et al., 2010).

Administratori baza podataka koriste logičku razinu apstrakcije. Logička shema je ujedno i daleko najvažnija u pogledu njenog utjecaja na aplikacijske programe iz razloga što programeri razvijaju apikacije korsteći upravo nju (Silberschatz, et al., 2010).

Izmjena globalne logičke razine ne mora izazvati izmjenu lokalne logičke razine. Ona ne utječe na korisnike i aplikacijske programe koji ih koriste (Silberschatz, et al., 2010).

3.3. Lokalna logička razina

Lokalna logička razina je najviša razina apstrakcije koja opisuje samo dio cjeloukupne baze podataka. Iako koristi jednostavnije strukture, složenost ostaje zbog raznovrsnosti informacija koje su pohranjene u bazi podataka. Mnogi korisnici sustava baze podataka ne trebaju sve informacije, već trebaju pristupiti samo određenom dijelu baze podataka s određenim informacijama. Ova razina postoji kako bi pojednostavila njihovu interakciju sa sustavom. Sustav može pružiti različite poglede za istu bazu podataka. Ona opisuje pogled na dio baze podataka koji je namijenjen specifičnoj grupi korisnika (Silberschatz, et al., 2010).

4. Model i modeliranje

Model je apstrakcija ili prikaz stvarnog svijeta koji otkriva sve značajke koje su od interesa korisnicima podataka u tom modelu. Modeli se stvaraju kako bi se postiglo bolje razumijevanje procesa, pojava ili aktivnosti (Sharma, et al., 2010).

Koriste se za prikaz podataka, događaja i relacijskih odnosa između njih, sve kako bi osigurali osnovne koncepte i pravila za dobru komunikaciju između programera i korisnika. Modeliranje stvarnog svijeta predstavlja njegovo preslikavanje u oblik pogodan za računalnu obradu. Baza podataka nekog informacijskog sustava predstavlja sliku stvarnog organizacijskog svijeta. Stvarni svijet, zbog njegove složenosti, ne možemo prikazati sa svim detaljima. On se predstavlja pojednostavljenim, nadomjesnim modelom. Model stvarnog svijeta predstavlja se uz pomoć nekog formalnog sustava (Baranović & Zakošek, 2012).

Modeliranje je važno jer uzima u obzir fleksibilnost koja je potrebna za buduće promjene bez da one značajno utječu na korištenje. Modeliranje omogućuje kompatibilnost sa svojim prethodnim modelom i omogućuje moguća buduća proširenja. Ono definira sve potrebne podatke iz stvarnog svijeta. To je prvi korak procesa razvoja baze podataka.

4.1. Model podataka

Baze podataka se mijenjaju tijekom vremena onako kako se informacije ažuriraju i brišu. Skup podataka pohranjenih u bazi podataka u određenom trenutku zove se instanca baze podataka, dok se cjeloukupni dizajn baze podataka naziva shema baze podataka. Shema baze podataka je formalni opis svih podataka baze podataka i svih odnosa koji postoje između njih. Postoje različiti načini organiziranja sheme. Oni se nazivaju modeli podataka (ili modeli baze podataka) (Silberschatz, et al., 2010).

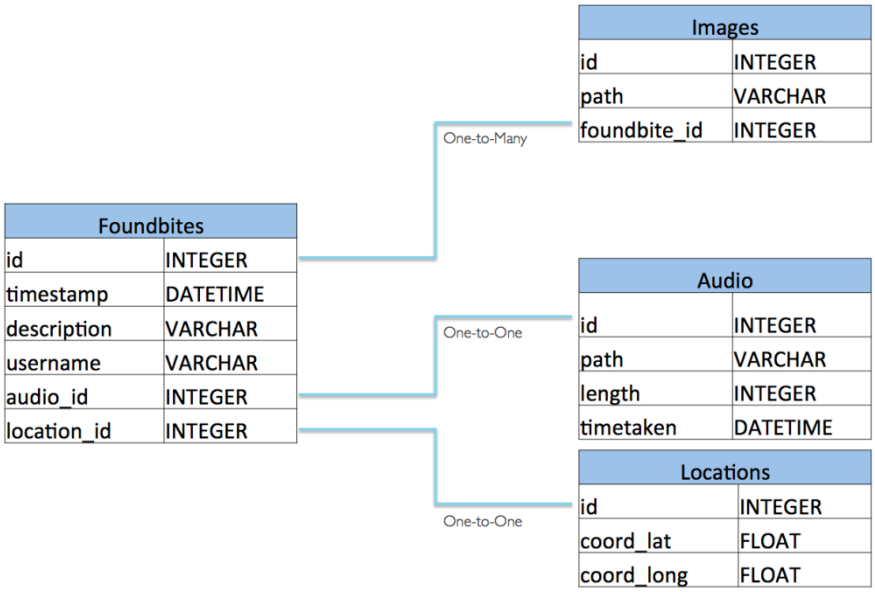
Model podataka je skup pravila koja određuju kako sve može izgledati logička struktura baze podataka. To je je apstraktni model čija je svrha opisati kako se podaci mogu učinkovito koristiti i predstavljati. Model čini osnovu za projektiranje i implementiranje baze. Točnije rečeno, podaci u bazi moraju biti logički organizirani u skladu s modelom koji podržava odabrani sustav za upravljanje bazom podataka.

Najčešći i najrasprostranjeniji modeli baza podataka su relacijski model, hijerarhijski model i mrežni model.

4.1.1. Relacijski model

Relacijski model koristi zbirku tablica za opisivanje podataka i odnosa između njih. Svaka tablica ima više stupaca, a svaki stupac ima jedinstveni naziv. Tablice su još poznate kao i veze. Model ima čvrste matematičke temelje bazirane na teoriji skupova, predikatnom računu i relacijskoj algebri. Relacijski model podataka je najkorišteniji model podataka, a velika većina postojećih sustava baza podatka temelji se upravo na relacijskom modelu podataka. Ciljevi relacijskog modela podataka (Sharma, et al., 2010):

- Osigurati visoki stupanj nezavisnosti podataka
- Postaviti temelje za rješavanje problema semantike, konzistentnosti i redudancije podataka
- Omogućiti razvoj DML[1] (eng. *Data Manipulation Language*)

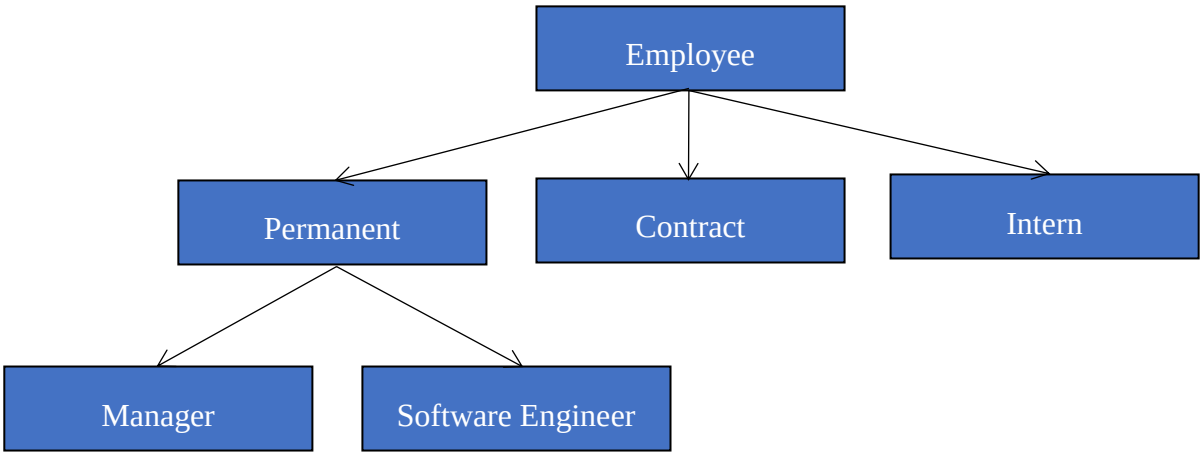


Slika 2: Relacijski model (Izvor: Anders Ramsay, 2013)

[1] DML je dio SQL jezika za manipulaciju podacima

4.1.2. Hijerarhijski model

Hijerarhijski model organizira svoje podatke pomoću strukture stabla. Korijen stabla je roditelj, a zatim sljede djeca, pri čemu dijete ne može imati više od jednog roditelja, dok roditelj može imati više djece. To je temeljno pravilo hijerarhijske baze. Prikazuje se skupom hijerarhijskih dijagrama strukture podataka (za prikaz nekih problema, zbog ograničenosti hijerarhijskog prikaza, potreban više nego jedan dijagram).



Slika 3: Hijerarhijski model

Prvi hijerarhijski sustav za upravljanjem bazom podataka zove se IMS (*Information Management System*) izdan 1968 od strane IBM-a. Prvobitno je izrađen kao baza podataka za Apollo svemirski program za sljetanje prvih ljudi na mjesec (Sharma, et al., 2010).

4.1.3. Mrežni model

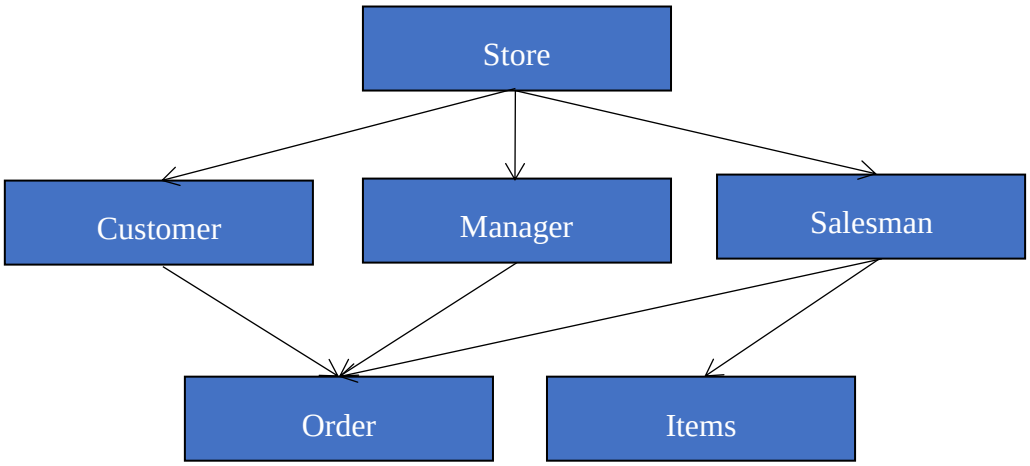
Mrežni model podataka definiran je krajem šezdesetih godina prošlog stoljeća. Najreprezentativniji model CODASYL [2] definiran je 1971.g., a na osnovu njega razvijeni su najpoznatiji mrežni sustavi za upravljanje bazama podataka. Jedan od njih je i NDBMS[3]. To je fleksibilan način prikazivanja objekata i veza među njima.

Mrežni model predstavlja proširenje hijerarhijskog modela, a osnovno poboljšanje je moguće da jedan podređeni slog može imati više korijen slogova odnosno moguća je implementacija veze i tipa M:N. Mrežna baza podataka sastoji se od skupa slogova i skupa

[2] CODASYL - mrežni model podataka predložen od strana CODASYL konsortijuma (Conference/Committee on Data Systems Languages). Konsotrijum je osnovan 1959. godine kako bi vodio razvoj standarnog programskog jezika koji bi se mogao koristiti na mnogim računalima.
[3] NDBMS (Network Database Management System) - mrežni sustav za upravljanje bazom podataka; temeljen na mrežnom modelu podataka.

veza, pri čemu jedan slog sadrži podatke jedne pojave entiteta i sastoji se od polja koja odgovaraju atributima.

Prednost mrežnog modela pa i hijerarhijskog modela podataka jest jednostavnost u prikazivanju i upravljanju podacima. Međutim nedostaci su pretraživanje po unaprijed definiranim putovima i unaprijed definiranim vezama između objekta. Prikazivanje veze M:N ni u mrežnom modelu nije najbolje riješeno već se veze razbijaju na tipove 1:M i 1:N. Slično hijerarhijskim bazama podataka i mrežne baze su u novije vrijeme sve manje u uporabi.



Slika 4: Mrežni model

Povijesno gledano, mrežni model podataka i hijerarhijski model podatka prethodili su relacijskom modelu podataka. Ti modeli su bili usko vezani za početnu implementaciju, što je kompliciralo modeliranje podataka. Kao rezultat toga rijetko ih možemo pronaći u primjeni. To su uglavnom sustavi koji koriste zastarjele baze podataka (Sharma, et al., 2010).

5. Entiteti

Već smo nekoliko puta spomenuli entitete, a da ih nismo pojasnili. Pod pojmom entiteta podrazumijeva se sve što se može jednoznačno odrediti, identificirati i razlikovati. Entitet je bilo što, što ima suštinu ili bit i posjeduje značajke s pomoću kojih se može razlučiti od svoje okoline. Predstavljaju nešto što je u stanju postojati ili nepostojati, te se može identificirati, u ovom slučaju to su tablice u bazama podataka (Sharma, et al., 2010).

Svojstvo entiteta uključuje dva elementa - atribut i vrijednost atributa. Npr. entitet auto je opisan atributima: marka, naziv, motor, vrsta.... Ukoliko neki od atributa, sam za sebe zahtijeva opis atributima, potrebno ga je definirati kao novi entitet. Isto vrijedi ako atribut može istovremeno imati više vrijednosti. Slični entiteti se svrstavaju u skupove entiteta, a slični entiteti su oni kojima se promatraju ista svojstva. Pojedini entitet naziva se pojavom entiteta. Sve pojave entiteta istog tipa (npr. OSOBA) imaju jednake atribute (npr. Matični_broj, Prezime, Adresa), a razlikuju se po vrijednosti atributa pojedinih pojava. Atributi fizički predstavljaju stupce u tablicama u bazi podataka.

Baza podataka ne modelira samo pojedinačne entitete nego i odnose tj. veze između entiteta. Veze su uspostavljene između dva ili više entiteta. Tipovi veza između entiteta:

- jedan-naprema-jedan 1:1
- jedan-naprema-mnogo 1:M
- mnogo-naprema-jedan M:1
- M : N

6. Relacijska baza podataka

Relacijska baza podataka je vrsta baze podataka kod koje se organizacija podataka zasniva na relacijskom modelu podataka. Podaci se u ovakvim bazama organiziraju u skup relacija između kojih se definiraju određene veze (Kramberger, 2011).

Relacija predstavlja imenovanu dvodimenzionalnu tablicu, pri čemu je atribut imenovani stupac relacije, domena skup dopuštenih vrijednosti atributa, a n-torka redak relacije. Relacija se definira kao skup n-torki sa istim atributima, definiranih nad istim domenima iz kojih se mogu uzimati vrijednosti (Kramberger, 2011).

Relacijska shema R mijenja se relativno rijetko, dok instanca relacije r predstavlja trenutnu vrijednost relacije i često se mijenja.

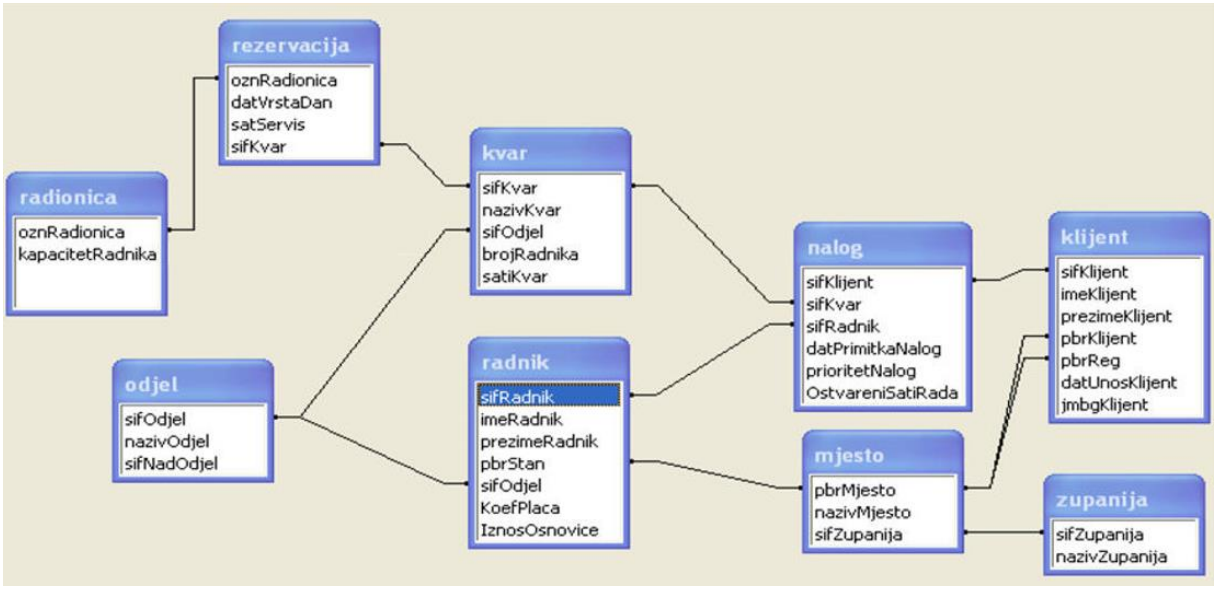
Svojstva relacije (Baranović & Zakošek, 2012):

- Relacija posjeduje ime koje je jedinstveno unutar sheme baze podataka
- Atributi unutar relacije imaju jedinstvena imena
- Jedan atribut može poprimiti vrijednost iz samo jedne domene
- U jednoj relaciji ne postoje dvije jednake n-torke
- Redoslijed atributa unutar relacije je nebitan
- Redoslijed n-torki unutar relacije je nebitan

Relacijsku algebru dijelimo na unarne i binarne operacije:

- Unarne operacije su: projekcija, selekcija, preimenovanje, agregacija, grupiranje.
- Binarne operacije dijelimo na:
    - Skupovske operacije - temelje se na relacijama kao skupovi n-torki: unija, presjek, razlika
    - Ostale binarne operacije - kartezijev produkt, djeljenje, spajanje

Najpopularniji sustavi za upravljanje su: Microsoft SQL Server, Oracle Database, MySQL i drugi. Većina tih sistema koristi upitni jezik SQL za manipulaciju podacima.



Slika 5: Primjer relacijske baze podataka (Izvor: Kramberger, 2011)

6.1. Primarni ključ

U relacijskim bazama podataka, svaka relacija mora imati definirani primarni ključ, koji predstavlja atribut pomoću kojeg se jedinstveno identificira svaka n-torka. Relacija može imati i strani ključ, preko kojeg ostvaruje vezu sa drugim relacijama. Primarni ključ smije biti samo jedan po tablici, ali može sadržavati više od jednog atributa u sebi (kompozitni ili složeni primarni ključ). Primarni ključ ne smije sadržavati NULL vrijednosti ni duplikate. Kod većine SUBP-a[4] (uključujući i MySQL) prilikom kreiranja primarnog ključa automatski se stvara indeks po kojem se sortiraju podaci (eng. *clustered indeks*) (Kramberger, 2011).

6.2. Strani ključ

Strani ključ se koristi za povezivanje određene kolone s primarnim ključem na način da onemogućava upis vrijednosti koju ne sadrži kolona s primarnim ključem. Za razliku od primarnog ključa on može omogućavati NULL vrijednosti (u tom slučaju nema spoja). Moguće je kreirati više stranih ključeva po tablici, a MySQL SUBP prilikom kreiranja stranog ključa automatski kreira indeks (Kramberger, 2011).

[4] SUBP – sustav za upravljanje bazom podataka

6.3. Primjer primarnog i stranog ključa

Primjer stvaranja tablice sa primarnim ključem:

```
CREATE TABLE mjesto (
  pbrMjesto INT,
  naziv VARCHAR(50),
  PRIMARY KEY pbrMjesto (pbrMjesto)
)
```

Proglašavajući kolonu odnosno atribut primarnim ključem, nemoguće je unijeti dvije n-torke s istim poštanskim brojem. Prilikom pokušaja unosa, SUBP će javiti grešku.

Kako bi se stvorila relacija i održao referencijalni integritet osim primarnih ključeva potrebno je koristiti strane ključeve. Strani ključ neće dozvoliti da se unese podatak s vrijednosti koja ne postoji u referenciranoj tablici s primarnim ključem. To ukratko znači da ukoliko u tablici „student“ pokuša unijeti vrijednost poštanskog broja „12345“, a on ne postoji u tablici „mjesto“ upis podatka se neće obaviti (Kramberger, 2011).

Primjer kreiranja tablice sa stranim ključem:

```
CREATE TABLE student (
  sifStudent INT,   jmbag VARCHAR(10) DEFAULT NULL,
  ime VARCHAR(255) COLLATE cp1250_croatian_ci DEFAULT NULL,
  prezime  VARCHAR(255)  COLLATE  cp1250_croatian_ci  DEFAULT
NULL,
  pbrRodjenja INT(11) DEFAULT NULL,
  datUnos DATE DEFAULT NULL,
  PRIMARY KEY sifStudent (sifStudent),
  CONSTRAINT FK_Student_Mjesto FOREIGN KEY (pbrRodjenja)
REFERENCES `mjesto` (`pbrMjesto`) ON DELETE SET NULL ON UPDATE
SET NULL
)
```

Osim dodavanja constrainta stranog ključa, u ovom primjeru mogu se primijetiti i neke druge opcije vezane uz kolone tj. atribute same tablice. Naredba `COLLATE` određuje jezik i postavke koje će se koristiti prilikom pohranjivanja i pretraživanja zapisa. Naredba `DEFAULT NULL` označava da ukoliko se podatak ne upiše u kolonu, on iznosi „NULL“ vrijednost.

7. SQL

Neophodni dijelovi SUBP-a su jezik za definiciju podataka i jezik za rukovanje podacima. SQL (eng. *Structured Query Language*) objedinjuje funkcije oba jezika, te je proglašen standardnim jezikom za relacijske sustave. SQL je temeljen na relacijskom modelu podataka. Važna značajka jezika je neproceduralnost. Opisuje se što se želi dobiti kao rezultat, ali ne i kako se do tog rezultata dolazi. U SUBP-u ugrađeni optimizator upita pronalazi nejefikasniji način obavljanje upita. Objekti u SQL-u su tablice, a ne relacije (Kramberger, 2011).

7.1. DDL

DDL (eng. *Data Definition Language*) je dio SQL jezika za definiranje struktura baze podataka. DDL omogućava kreiranje, mijenjanje i brisanje entiteta (tablica), atributa (kolona), baza podataka, relacija, procedura, funkcija itd (Kramberger, 2011). Najvažnije naredbe su:

1. CREATE – omogućava kreiranje novih struktura (baze podataka, tablica, funkcija...)

Osnovna sintaksa za kreiranje baze podataka:

```
CREATE {DATABASE | SCHEMA} [IF NOT EXISTS] db_name
[create_specification]
create_specification:
[DEFAULT] CHARACTER SET [=] charset_name
| [DEFAULT] COLLATE [=] collation_name
```

Primjer:

```
CREATE DATABASE NazivBaze
CHARACTER SET = cp1250
COLLATE = cp1250_croatian_ci;
```

Osnovna sintaksa za kreiranje tablice:

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name
(create_definition,...) [table_options]
```

Primjer:

```
CREATE TABLE mjestoBezKljuca (
     pbrMjesto INT,
     naziv VARCHAR(50)
)
```

2. ALTER – omogućava promjenu prethodno kreiranih struktura

Osnovna sintaksa za `ALTER` baze podataka:

```
ALTER {DATABASE | SCHEMA} [db_name]       alter_specification
...

alter_specification:            [DEFAULT]  CHARACTER  SET  [=]
charset_name   | [DEFAULT] COLLATE [=] collation_name
```

Primjer:

```
ALTER DATABASE NazivBaze
CHARACTER SET = cp1250
COLLATE = cp1250_croatian_ci;
```

Osnovna sintaksa za `ALTER` tablice:

```
ALTER [ONLINE | OFFLINE] [IGNORE] TABLE tbl_name
    [alter_specification [, alter_specification] ...
```

Primjer:

```
ALTER TABLE student
DROP COLUMN datUnos,
ADD COLUMN visina DECIMAL(5,2) DEFAULT NULL AFTER pbrRodjenja,
CHANGE COLUMN ime imeStudent VARCHAR(50) NOT NULL;
```

3. RENAME – mijenja naziv entitetu ili bazi podataka

Osnovna sintaksa `RENAME` baze podataka:

```
RENAME {DATABASE | SCHEMA} db_name TO new_db_name;
```

Primjer:

```
RENAME DATABASE NazivBaze TO NoviNazivBaze;
```

Osnovna sintaksa `RENAME` tablice

```
RENAME TABLE tbl_name TO new_tbl_name
    [, tbl_name2 TO new_tbl_name2] ...
```

Primjer:

```
RENAME TABLE student TO noviStudent;
```

4. DROP – briše strukturu i sve podatke unutar iste

Osnovna sintaksa `DROP` baze podataka:

```
DROP {DATABASE | SCHEMA} [IF EXISTS] db_name
```

Primjer:

```
DROP DATABASE NazivBaze;
```

Osnovna sintaksa `DROP` tablice:

```
DROP [TEMPORARY] TABLE [IF EXISTS]
    tbl_name [, tbl_name] ...
```

Primjer:

```
DROP TABLE NekaTablica;
```

7.2. DML

DML (eng. *Data Manipulation Language*) je dio SQL jezika za manipulaciju podacima koji se nalaze u strukturi kreiranoj od strane DDL-a. DML omogućava umetanje podataka u tablice, ispravak podataka, brisanje podataka te projekcije podataka (Kramberger, 2011).Popis DML naredbi:

1. INSERT – unosi podatke u tablicu

Osnovna sintaksa (3 načina):

```
INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]
     [INTO] tbl_name [(col_name,...)]
     {VALUES | VALUE} ({expr | DEFAULT},...),(...),...
     [ ON DUPLICATE KEY UPDATE
         col_name=expr
            [, col_name=expr] ...
```

```
INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]
     [INTO] tbl_name
     SET col_name={expr | DEFAULT}, ...
     [ ON DUPLICATE KEY UPDATE
        col_name=expr
           [, col_name=expr] ...
```

```
INSERT [LOW_PRIORITY | HIGH_PRIORITY] [IGNORE]
     [INTO] tbl_name [(col_name,...)]
     SELECT ...
     [ ON DUPLICATE KEY UPDATE
        col_name=expr
           [, col_name=expr] ... ]
```

Primjer za svaki od načina:

```
INSERT INTO rezervacija (oznRadionica, satServis, sifKvar)
VALUES('s55', 2, 33);
```

```
INSERT INTO rezervacija
SET oznRadionica = 's55',
satServis = 2,
sifKvar = 33;
```

```
INSERT INTO rezervacija (oznRadionica, satServis, sifKvar)
SELECT oznRadionica, satServis, sifKvar FROM rezervacija
```

2. UPDATE – mijenja postojeće podatke u tablici

Osnovna sintaksa `UPDATE` naredbe:

```
UPDATE [LOW_PRIORITY] [IGNORE] table_reference |
table_references
     SET col_name1={expr1|DEFAULT} [,
col_name2={expr2|DEFAULT}] ...

     [WHERE where_condition]
```

Primjer:

```
UPDATE klijent
SET imeKlijent = 'Ivan',
prezimeKlijent = 'Horvat'
```

3. SELECT – prikazuje podatke

Osnovna sintaksa `SELECT` naredbe:

```
SELECT
     [ALL | DISTINCT | DISTINCTROW ]
        [HIGH_PRIORITY]
        [STRAIGHT_JOIN]
        [SQL_SMALL_RESULT] [SQL_BIG_RESULT] [SQL_BUFFER_RESULT]
        [SQL_CACHE | SQL_NO_CACHE] [SQL_CALC_FOUND_ROWS]
     select_expr [, select_expr ...]
     [FROM table_references
     [WHERE where_condition]
     [GROUP BY {col_name | expr | position}
        7[ASC | DESC], ... [WITH ROLLUP]]
     [HAVING where_condition]
     [ORDER BY {col_name | expr | position}
        [ASC | DESC], ...]
     [LIMIT {[offset,] row_count | row_count OFFSET offset}]
     [PROCEDURE procedure_name(argument_list)]
     [INTO OUTFILE 'file_name'
```

```
        [CHARACTER SET charset_name]
        export_options
      | INTO DUMPFILE 'file_name'
      | INTO var_name [, var_name]]
    [FOR UPDATE | LOCK IN SHARE MODE]]
```

Primjeri:

```
SELECT klijent.* FROM klijent;
SELECT         klijent.jmbgKlijent,         klijent.imeKlijent,
klijent.prezimeKlijent
FROM klijent;
```

4. DELETE – briše podatke

Naredba `DELETE` služi za brisanje podataka iz tablice, te se nikako ne smije pomiješati s naredbom `DROP` koja se koristi za brisanje strukture. `DELETE` naredba briše podatke a strukturu ostavlja netaknutom, `dok DROP` naredba briše i podatke i strukturu zajedno. `DELETE` naredbom nije moguće brisati po kolonama već samo po stupcima. Po kolonama je moguće brisati vrijednost pomoću `UPDATE` naredbe (Kramberger, 2011).

Osnovna sintaksa:

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE] FROM tbl_name
     [WHERE where_condition]
     [ORDER BY ...]
     [LIMIT row_count]
```

Primjer brisanja svih podataka iz tablice klijent:

```
DELETE FROM klijent;
```

U stvarnom životu jako rijetko je potrebno sve podatke iz tablice obrisati. Naredbu `DELETE` potrebno je oprezno koristiti kako se podaci nebi nepovratno obrisali.

7.3. Nepotpune informacije i NULL vrijednosti

Ponekad se dešava da informacije koje treba unijeti u bazu podataka nisu informacije. Razlozi zbog kojeg može doći do toga su: neke informacije trenutno nisu poznate, neke informacije uopće ne postoje (nisu primjenjive), neke informacije postoje ali do njih nije moguće doći. Informacije koje nedostaju prikazuju se kao poseban oblik podatka NULL vrijednost. NULL vrijednost se interno pohranjuje drugačije od bilo koje druge dopuštene vrijednosti. Sam način pohrane je nebitan, jer je NULL vrijednost neovisna od tipa podatka kojeg predstavlja. U SQL naredbama, bez obzira na tip podatka, koristi se „konstanta“ NULL (Baranović & Zakošek, 2012).

8. Zaštita baze podataka

Svaka baza podataka mora biti zaštićena od nedopuštenih radnji u svrhu očuvanja sigurnosti i integriteta baze. Pojmovi integritet i sigurnost baze podataka se često spominju zajedno, međutim radi se o dva različita aspekta zaštite podataka (Baranović & Zakošek, 2012).

Integritet baze podataka - operacije nad podacima koje korisnici obavljaju su ispravne tj. uvijek rezultiraju konzistentnim stanjem baze podataka („podaci se štite od ovlaštenih korisnika“).

Sigurnost baze podataka - korisnici koji obavljaju operacije nad podacima su ovlašteni za obavljanje tih operacija („podaci se štite od neovlaštenih korisnika“).

Među tim pojmovima postoje i sličnosti. U oba slučaja moraju biti definirana pravila koja korisnici ne smiju narušiti, pravila se pohranjuju u rječnik podataka te SUBP nadgleda rad korisnika (osigurava poštivanje pravila).

8.1. Integritet baze podataka

Pojam integriteta baze podataka odnosi se na konzistentnost (ispravnost) podataka sadržanih u bazi podataka. Integritet baze podataka može biti narušen zbog: slučajne greške korisnika ili kod izmjene podataka i pogreške aplikacijskog programa ili sustava. Integritetska ograničenja osiguravaju da izmjene podataka koje obavljaju korisnici ne rezultiraju narušavanjem konzistetnosti podataka. O problemu djelovanja neautoriziranih korisnika, diverzije ili sabotaže brine poseban dio SUBP-a koji je zadužen za sigurnost baze podataka.

Shema baze podataka sastoji se od skupa relacijskih shema i skupa integritetskih ograničenja, a ispravna instanca baze podataka je ona instanca koja zadovoljava sva definirana integritetska ograničenja.

Definicije integritetskih ograničenja su sastavni dio sheme podataka te se pohranjuju u rječnik podataka baze podataka. Na taj način pravila definirana integritetskim ograničenjima postaju nezaobilazna za svakog korisnika sustava.

Riječnik podataka sadrži opise podataka (metapodatke) koji su prikazani na isti način kao i „obični“ podaci te im se može i pristupati na isti način. On sadrži opis relacijskih shema,

opis pravila integriteta, opis pravila pristupa, opis pohranjenih procedura i poslovnih pravila, opis okidača itd.

SUBP provjerava integritetska ograničenja pri obavljanju svake operacije koja mijenja sadržaj baze podatake. U trenutku završetka operacije nad podacima, baza podataka mora biti u stanju u kojem su zadovoljena sva integritetska ograničenja. SUBP odbija operacije koje nemaju to svojstvo ili obavlja kompenzacijske akcije koje osiguravaju da su u konačnici sva integritetska ograničenja zadovoljena.

Integritetska ograničenja (Baranović & Zakošek, 2012):

Entitetski integritet - niti jedan atribut primarnog ključa ne smije poprimiti null vrijednost
Integritet ključa: u relaciji ne smiju postojati dvije n-torke s jednakim vrijednostima ključa (vrijedi za sve moguće ključeve).

Domenski integritet - atribut može poprimiti samo jednu vrijednost iz domene atributa
Ograničenja NULL vrijednosti: za određene atribute se može definirati ograničenje prema kojem vrijednost atributa ne smije poprimiti NULL vrijednost.

Referencijski integritet - referencijski integritet odnosi se na konzistentnost među n-torkama dviju relacija (ili n-torkama iste relacije). To je sustav pravila kojima se osigurava
da veze između zapisa u povezanim tablicama budu važeće i da ne bi slučajno obrisali ili izmjenili podatke. Dakle, referencijalni integritet osigurava postojanost svih podataka u bazi.

Opća integritetska ograničenja - ograničenja općeg oblika.

8.2. Sigurnost baze podataka

Oblici narušavanja sigurnosti baze podataka su:

- Neovlašteno čitanje podataka
- Neovlaštena izmjena podataka
- Neolašteno učitavanje podataka

Moduće posljedice su:

- Krađa ili prijevara
- Gubitak tajnosti
- Gubitak privatnosti
- Gubitak raspoloživosti

Sigurnost baze podataka osigurava se zaštitom na nekoliko razina (Baranović & Zakošek, 2012):

- **Zaštita na razini SUBP**: spriječiti pristup bazama podataka ili onim dijelovima baza podataka za koje korisnici nisu ovlašteni
- **Zaštita na razini operacijskog sustava**: spriječiti pristup radnoj memoriji računala ili datotekama u kojima SUBP pohranjuje podatke
- **Zaštita na razini računlne mreže**: spriječiti presretanje poruka na internetu i intranetu
- **Fizička zaštita**: fizički zaštititi lokaciju računalnog sustava
- **Zaštita na razini korisnika**: spriječiti da ovlašteni korisnici nepažnjom ili namjerno omoguće pristup podacima neovlaštenim osobama

8.3. Autentičnost i autorizacija

Administrator sustava omogućuje korisniku pristup sustavu definiranjem jedinstvenog identifikatora korisnika i pripadne lozinke koja je poznata samo dotičnom korisniku i sustavu. Korisnik koji pristupa sustavu poznavanjem lozinke ovjerava svoju autetičnost.

Autorizacija je postupak kojim se određenom korisniku dodjeljuje dozvola za obavljanje određenih vrsta operacija (čitanje, izmjena, brisanje itd.) nad određenim objektima baze podataka (relacija, pogled, atribut itd.). Podaci o dodijeljenim dozvolama pohranjaju se u

rječnik podataka. Prije obavljanja svake operacije, SUBP provjerava ima li korisnik dozvolu za obavljanje operacije nad objektom (kontrola pristupa).

Današnji SUBP podržavaju dva različite modela kontrole pristupa podacima (Baranović & Zakošek, 2012):

- **Mandatna kontrola pristupa** (eng. *MAC – Mandatory Access Control*)
- **Diskrecijska kontrola pristupa** (eng. *DAC – Discretionary Access Control*)

8.3.1. Mandatna kontrola pristupa

Mandatnu kontrolu pristupa podržava manji broj SUBP-ova te se koristi relativno rijetko u odnosu na diskrecijsku kontrolu pristupa. Ona je primjenjiva u sustavima u kojima se dozvole dodjeljuju na temelju pozicije korisnika u hijerarhiji neke organizacije (vojska, državna uprava itd.). Djeluje na način da se svakom objektu dodjeljuje oznaka klasifikacijske razine (npr. povjerljivo, tajno, vrlo tajno itd.), a svakom korisniku oznaka razine ovlasti. Na taj način korisnici mogu obavljati operacije nad onim objektima za koje imaju odgovarajuću razinu ovlasti (Baranović & Zakošek, 2012):

8.3.2. Diskrecijska kontrola pristupa

Većina današnjih SUBP podržava diskrecijsku kontrolu pristupa. Ona je podrđana SQL standardom. Određenom korisniku se eksplicitno dodjeljuje dozvola za obavljanje određene operacije nad određenim objektom (relacija, atribut, virtualna relacija, baza podataka). Dozvole su opisane trojkama `<korisnik, objekt, vrsta operacije>` (Baranović & Zakošek, 2012).

Primjer:

```
<ivan, fakultet, čitanje>
```

Kada korisnik ivan pokuša obaviti operaciju čitanja objekta (relacije) fakultet, SUBP provjerava postoji li dozvola u obliku trojke.

Pojam vlasnik objekta odnosi se na korisnika koje je kreirao objekt. Npr. vlasnik baze podataka je korisnik koji je kreirao bazu podataka, a vlasnik relacije je korisnik koji je kreirao relaciju. Vlasnik objekta implicitno dobiva dozvole za obavljanje svih vrsta operacija nad objektom, uključujući dozvole za: dodjeljivanje svih vrsta dozvola nad tim objektom drugim korisnicima te uništavanje objekta.

8.3.3. Stvaranje korisnika i dodjeljivanje dozvola

Svaki korisnik u bazi podataka je definiran korisničkim imenom te oznakom lokacije (adrese klijenta) s koje može pristupati podacima.

Osnovna sintaksa za kreiranje korisnika:

```
CREATE USER imeKorisnika [IDENTIFIED BY [PASSWORD] 'lozinka']
           [,imeKorisnika2     [IDENTIFIED     BY     [PASSWORD]
'lozinka2']] …
```

Primjer:

```
CREATE USER 'ihorvat'@'localhost' IDENTIFIED BY '1234';
```

Osnovna sintaksa za brisanje korisnika:

```
DROP USER imeKorisnika
```

Primjer:

```
DROP USER 'ihorvat'@'localhost'
```

Dozvole korisnicima za rad s podacima u MySQL bazi moguće je dodijeliti na dva načina. Prvi i preporučljiv način je koristeći naredbu `GRANT`. Sintaksa je čista i jednostavna, te se na brz način može dodijeliti odgovarajuća dozvola nekom korisniku. Kao što je moguće stvoriti novog korisnika izmjenom odnosno unosom podataka u tablici mysql.user, moguće je i dodijeliti odgovarajuće dozvole manipulacijom podataka u odgovarajućim tablicama u bazi mysql. To je nešto složeniji način jer je potrebno poznavati strukturu tablica u kojima su dozvole pohranjene. Ta se metoda preporuča samo u delikatnijim zahtjevima i situacijama (Duk, 2012).

Osnovna sintaksa korištenja naredbe `GRANT`:

```
GRANT
     priv_type [(column_list)]
       [, priv_type [(column_list)]] ...
     ON [object_type] priv_level
     TO user_specification [, user_specification] ...
     [REQUIRE {NONE | ssl_option [[AND] ssl_option] ...}]
     [WITH with_option ...]
```

Primjer:

Dodijeliti sve dozvole nad svim podacima u sustavu korisniku admin sa svih adresa.

```
GRANT ALL ON *.* TO 'admin'@'%';
```

Primjer:

Dodijeliti sve dozvole nad svim podacima u sustavu korisniku admin sa svih adresa. Neka navedeni korisnik ima mogućnost dodjeljivanja dozvola drugim korisnicima.

```
GRANT ALL ON *.* TO 'admin'@'%' WITH GRANT OPTION;
```

Primjer:

Dodijeliti dozvole čitanja, dodavanja i ažuriranja podataka u tablici nalog (baza autoradionica) korisniku korisnik25 sa bilo kojeg poslužitelja u domeni servis.com. Korisnik korisnik25 se spaja koristeći lozinku mojaLozinka. Neka korisnik25 može ostvariti najviše 25 konekcija u satu.

```
GRANT SELECT, INSERT, UPDATE
        ON autoradionica.nalog
        TO 'korisnik25'@'%.servis.com'
        IDENTIFIED BY 'mojaLozinka'
        WITH MAX_CONNECTIONS_PER_HOUR 25;
```

Ako se naredbom grant pokušavaju dodijeliti dozvole korisniku koji ne postoji, korisnik će automatski biti kreiran.

Kao i kod dodjeljivanja, dozvole se mogu ukinuti na dva načina. Prvi je način naredbom `REVOKE`, a drugi ažuriranjem odgovarajućih tablica u bazi MySQL. Sve što je oko izbora načina navedeno za dodjeljivanje dozvola, vrijedi i za ukidanje dozvola (Duk, 2012).

Osnovna sintaksa korištenja naredbe `REVOKE`:

```
REVOKE ALL PRIVILEGES, GRANT OPTION
     FROM user [, user] ...
```

Primjer:

Ukinuti dozvole ažuriranja podataka u tablici nalog (baza autoradionica) korisniku korisnik25 sa bilo kojeg poslužitelja u domeni servis.com.

```
REVOKE UPDATE ON autoradionica.nalog
     FROM 'korisnik25'@'%.servis.com';
```

Ako je određenom korisniku potrebno ukinuti određene dozvole, a nismo sigurni koje trenutno ima dodijeljene, potrebno je prvo s naredbom `SHOW GRANTS` provjeriti trenutno stanje korisnikovih dozvola (Duk, 2012).

Osnovna sintaksa

```
SHOW GRANTS FOR imeKorisnika
```

Primjer:

Provjeriti dozvole za ranije stvorenog korisnika ihorvat.

```
SHOW GRANTS FOR 'ihorvat'@'localhost
```

Promjena korisničkog imena

Osnovna sintaksa:

```
RENAME USER stariKorisnik TO noviKorisnik
     [,stariKorisnik2 TO noviKorisnik2] …
```

Kod promjene korisničkog imena treba biti oprezan jer će MySQL ažurirati samo podatke u tablici mysql.user, odnosno neće promijeniti ostale zapise koji se odnose na korisnikove dozvole. Posljedica toga je da stare dozvole ostaju aktivne bez obzira što stari korisnik više ne postoji. Postoji opasnost da kada bi se u budućnosti stvorio novi korisnik istog imena kao što je bio stari, odjednom bi imao dodijeljene (a potencijalno i opasne) dozvole. Iz tog se razloga preporuča prvo koristiti naredbu `REVOKE` pa nakon toga `GRANT`, odnosno trajno obrisati starog korisnika ako više nije potreban (Duk, 2012).

8.4. Praćenje rada korisnika

Kako bi se povećala sigurnost same baze podataka potrebno je evidentirati svaki pristup osjetljivim podacima u posebnoj datoteci za praćenje rada korisnika (Audit Trail). Tipičan zapis datoteke sadrži sljedeće informacije (Duk, 2012):

- SQL naredba koja se izvršava
- Mjesto s kojeg je upućen zahjtev
- Indentifikator korisnika koji je pokrenuo operaciju
- Datum i vrijeme operacije
- N-torke, atributi na koje se zahtjev odnosi
- Stara vrijednost n-torke
- Nova vrijednost n-torke

8.5. Obnova baze podataka

Obnova baze podataka predstavlja dovođenje baze podataka u najnovije stanje za koje se pouzdano zna da je bilo ispravno. Velike baze podataka nužno moraju posjedovati mehanizme obnove. Manje jednokorisničke baze podataka obično imaju malu ili uopće nemaju potporu obnovi. Ona se prepušta korisnikovoj odgovornosti. Podrazumijeva se da korisnik periodički stvara arhivsku kopiju pomoću koje u slučaju potrebe obnavlja bazu podataka (Anon., n.d.).

Obnovu baze podataka omogućuje redudancija (zalihost). To je općenito pravilo koje govori da se svaki podatak mora moći rekonstruirati iz nekih drugih informacija redudantno pohranjenih negdje drugdje u sustavu. Ona se postiže zrcaljenjem podataka (eng. *mirroring*), sigurnosnim kopijama (eng. *backup*) ili dnevnicima izmjena (eng. *logical log*) (Baranović & Zakošek, 2012).

9. Sustav za upravljanje bazom podataka (SUBP)

Baza podataka je repozitorij podataka, a sustav za upravljanje bazom podataka, ili jednostavnije SUBP (eng. *DBMS - database management system*), je skup softverskih alata koji kontroliraju pristup, organiziraju, pohranjuju, upravljaju, dohvaćaju i održavaju podatke u bazi podataka. To je skup međusobno povezanih podataka i niz programa za pristup tim podacima (Sharma, et al., 2010).

Primaran cilj SUBP-a je pružiti efikasan način za pohranu i dohvat podataka. Sustavi baza podataka su dizajnirani za upravljanje velikom količinom podataka. Upravljanje podacima uključuje i definiranje strukture za pohranu informacija i osiguravanje mehanizma za upravljanje podacima. Osim toga, sustav baze podataka mora osigurati sigurnost pohranjenih podatka, i u slučaju pada sustava i neovlaštenog upada. Ako se podaci dijele između više korisnika, sustav mora izbjeći moguće nepravilne rezultate. Zbog velike važnosti informacija i podataka znanstvenici su počeli razvijati veliki broj koncepata i tehnika za upravljanje podacima (Silberschatz, et al., 2010).

SUBP sakriva od korisnika detalje fizičke pohrane podataka, osigurava logičku i fizičku nezavisnost podataka, omogućuje definiciju i rukovanje, obavlja funkciju zaštite podataka (integritet podataka, pristup podacima – autorizacija, sigurnost, kontorla paralelnoga pristupa, obnova u slučaju razrušenja), obavlja optimiziranje upita (Baranović & Zakošek, 2012).

Korisnik ili korisnički program postavlja zahtjev za obavljanjem neke operacije s podacima, a SUBP ga analizira, provjerava, optimizira, transformira u niz operacija koje je potrebno obaviti na fizičkoj razini, obavlja operacije i vraća rezultat.

SUBP se temelji na odabranom modelu podataka. Prema modelu podataka na kojem se temelj SUBP-ove dijelimo na: hijerarhijske, mrežne, relacijske, objektno-relacijske, objektno orijentirane itd (Baranović & Zakošek, 2012).

Zašto nam je uopće potrebna baza podataka i sustav za upravljanje njome?

Odgovor leži u načinu na koji korisnik pristupa podacima i kako rukuje određenim zahtjevima. Prvo, potrebna je mogućnost umetanja, ažuriranja i brisanja podataka od strane većeg broja korisnika bez da dolazi do međusobnog konfilikta. To znači da različiti korisnici neće prouzročiti nestalnost podataka, te da se podaci neće nehotice izgubiti tijekom tih operacija. Također je potrebno imati standarno sučelje za pristup podacima, alate za povrat podataka i oporavak, kao i sposobnost za rad s velikim količinama podataka i korisnika. Sustav za upravljanje bazom podataka je dizjaniran za izvršavanje svih tih akcija (Sharma, et al., 2010).

Glavna svrha sustava baze podataka je pružiti korisnicima apstraktni prikaz podataka. To znači da sustav skriva određene detalje o tome kako se podaci pohranjuju. Budući da mnogi korisnici sustava baze podataka nisu računalno obučeni, programeri skrivaju složenost sustava od korisnika kroz nekoliko razina apstrakcije, u svrhu podjednostavljenja interakcije sa sustavom (Silberschatz, et al., 2010).

9.1. Razvoj sustava za upravljanje bazom podataka

1960., mreže i hijerarhijski sustavi kao što su CODASYL i IMSTM[5] su smatrani kao napredna tehnologija za upravljanje bankarstvom, računovodstvom i obradom narudžbi, što je omogućeno uvođenjem komercijalnih računala. Arhitektura sustava se temeljila na povezanosti fizičke manipulacije podataka sa njihovom logičkom manipulacijom. Kada bi se fizička lokacija podataka promijenila, npr. premještanje podataka sa jednog diska na drugi, programi bi se trebali ažurirati kako bi mogli pokazivati na novu lokaciju (Sharma, et al., 2010).

Revolucionarna studija E.F.Gord-a , zaposlenika IBM-a u San Jose istraživačkom laboratoriju, 1970. godine je to sve promijenila. U radu pod naslovom „*Relacijski model podataka da dijeljenje velikog broja podataka*“ uveo je pojam nezavisnosti podataka, koji razdvaja fizički prikaz podataka od logičkog prikaza podataka koji su predstavljeni

[5] IMS (Information Management System) - hijerarhijski sustav baze podataka i sustav za upravljanje informacijama s velikom mogućnošću obrade transakcija

programima. Podaci se mogu premještati iz jednog dijela diska na drugi dio ili drugi disk te mogu biti pohranjeni u drugom formatu bez da je potrebno ažurirati aplikaciju. Na taj način programeri su bili oslobođeni vođenja brige oko fizičkih detalja manipulacije podataka te su se mogli posvetiti logičkoj manipulaciji podataka (Sharma, et al., 2010).

Danas, sustavi za upravljanje relacijskim bazama podataka su najkorišteniji sustavi za upravljanje bazama podatala. Kako su one postajale sve popularnijima nastala je potreba za pružanjem visokih performansa upita.

9.2. Fizička i logička organizacija podatka

Važna posljedica primjene SUBP-a jest razdvajanje fizičke i logičke organizacije podataka. Dok logička organizacija podataka predstavlja organizaciju sa gledišta korisnika baze podataka ili programera te je koncentrirana na vrste podataka i njihove međusobne logičke veze, fizička organizacija predstavlja organizaciju fizičke pohrane podataka unutar računala. Oblik i organizacija pohranjenih podataka tu su često potpuno različiti od njihovog logičkog oblika i organizacije. U okviru toga, zadaća je SUBP-a omogućiti korisniku ili programeru upravljanje podacima uz poznavanje samo logičkog opisa baze podataka, a ne nužno i poznavanja načina fizičke pohrane podataka. Fizička organizacija podataka ne utječe na rezultate operacija s podacima ali ima vrlo veliki utjecaj na učinkovitost sustava za upravljanje bazom podataka (Baranović & Zakošek, 2007).

10. Aktivne baze podataka

Pojam ASUBP[6] (*eng. ADBMS - Active Database Management System*) se definira kao sposobnost da se automatski reagira na nastale situacije u bazi podataka i izvan nje.

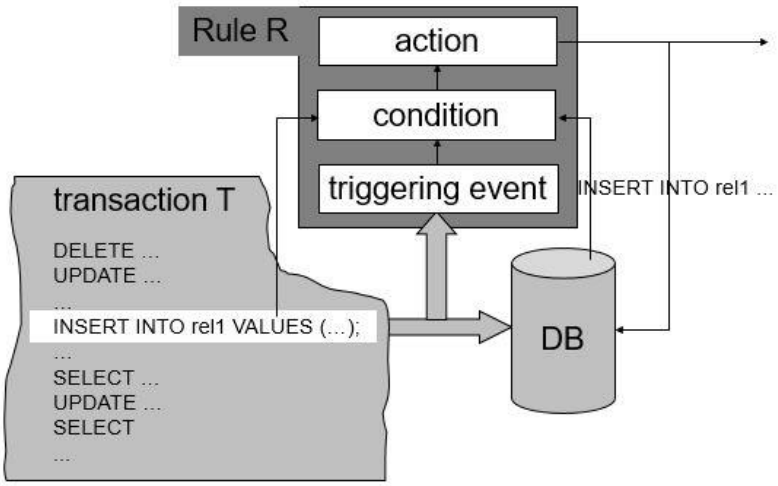
Tradicionalni sustavi za upravljanje bazom podataka su pasivni u smislu da baze podataka izvršavaju naredbe (upiti, umetanja, ažuriranja, brisanja) kada su one zadane od strane određenog korisnika ili aplikacije. Oni su promatrani kao spremišta koja pohranjuju podatke koji su potrebni za određene aplikacije i kojima se pristupa ili putem koriničkih programa ili preko interaktivnih sučelja. Međutim, današnji sustavi baze podataka primjenjuju se na raznim područjima povezanim s vrlo složenom obradom informacija, sa sve većim količinama podataka, ili vrlo strogim uvjetima izvedbe, što nije moguće podržati sa tradicionalnim sustavima. Za mnoge primjene potrebno je praćenje cjeloukupnog stanja sustava kako bi se pravovremeno aktivirao određeni odgovor na nastalu situaciju. Primjerice sustav za upravljanje zalihama treba pratiti količinu robe na skladištu, tako da se kada količina određene stavke padne ispod praga aktivira određena aktivnost. Takvo ponašanje se može ugraditi u pasivne sustave baza podataka (Paton & Diaz, 1999).

Aktivni sustavi baza podataka, za razliku od pasivnih, prate stanje sustava te pravovremeno reagiraju na novonastalu situaciju. Oni posjeduju mehanizme koji im omogućavaju da automatski reagiraju na događaje koji se odvijaju unutar ili izvan baze podataka. Ukratko, ASUBP proširuje SUBP sa sposobnošću reaktivnog ponašanja. Aktivna baza mora osigurati model znanja i model izvođenja strategija kako bi mogla omogućiti takvo reaktivno ponašanje.

Omogućavanjem reaktivnog ponašanja baze podataka, veliki dio semantike, koja se obično programira unutar aplikacije, se može izraziti aktivnim pravilima, te na taj način aplikacijama pruža novu dimenziju neovisnosti. To znači da su aplikacije oslobođene znanja o reaktivnom ponašanju te je ono smješteno u oblik aktivnih pravila. Prema tome, pravilo koje jednom napisano jednom vrijedi za sve elemente scheme i zajedničko je svim korisnicima. Promjena ponašanja se modificira promjenom pravila baze podataka, a ne promjenama u aplikaciji, te time jasno pruža, već spomenutu, novu dimenziju neovisnoti (Zaniolo, et al., 1997).

[6] ASUBP – Aktivni sustav za upravljanje bazom podataka

Aktivna pravila se koriste kao mehanizam za: kontrolu integriteta baze podataka, kontrolu prava pristupa, održavanje pogleda, generiranje izvedenih podataka, prikupljanje statističkih podataka, praćenje funkcioniranja baze podataka itd.



Slika 6: Koncepti aktivnih baza podataka (Izvor: http://slideplayer.com/slide/775887/)

Najpoznatiji predstavnici aktivnih sustava koji su se razvijali kao nadogradnja relacijskih sustava su Postgres[7] i Starburst[8], dok najpoznatiji aktivni sustav koji se zasniva na objektno orijentiranom modelu podataka je HiPAC [9].

[7] Postgres - razvijao se od 1985. godine na sveučilištu Berkeley u Kaliforniji.
[8] Starburst - razvijao se od 1985. godine u IBM-ovom istraživačkom centru u San Jose-u u Kaliforniji.
[9] HiPAC - *High Performance Active Database System*

11. ECA pravila

Većina aktivnih sustava baze podataka podržava pravila koja koja mogu imati do tri komponente: događaj, uvjet i akcija. Takvo pravilo je poznato pod nazivom ECA (*eng. event-condition-action*) pravilo. ECA pravila samostalno reagiraju na događaje koji utječu na podatke. Kada određeni događaj nastane pravilo vrednuje uvjet i ako je uvjet zadovoljen izvršava akciju. U nekim situacijama komponente pravila događaj ili uvjet mogu nedostajati. Ukoliko nedostaje događaj onda je riječ o condition-action pravilu, u drugom slučaju riječ je o event-action pravilu. Neka pravila su ograničena na praćenje jednog događaja. Međutim, u većini slučaja pravila prate zbirku događaja te se aktiviraju ako se bilo koji od njih pojavi (Zaniolo, et al., 1997).

U nekim sustavima pravila se mogu dinamički isključivati i uključivati. Međutim, isključivanje pravila može biti vrlo opasno pošto je jedan od tipičnih primjera aktivnih pravila održavanje integriteta. U tom slučaju, pravila očuvanja integriteta baze podataka ne smiju biti samovoljno isključena od strane baš bilo kojeg korisnika baze podataka. Iz tog razloga, aktivna pravila (kao i svi objekti baze podataka) podliježu autorizacijskim mehanizimima. Privilegije za pravila stvaranja, mijenjanja, brisanja, aktiviranja i deaktiviranja mogu biti dane samo administratorima baza podataka koji ih mogu delegirati drugim korisnicima koristeći `GRAND PRIVILEGE` naredbe.

Pravila se mogu svrstati u skupine. Formirana skupina može se odvojeno izvršavati, aktivirati i isključivati. Korisne su i za dodjeljivanje privilegija nad skupom pravila.

11.1. Model znanja

Model znanja aktivnog sustava baza podataka opisuje aktivna pravila u tom sustavu. Ustaljeni pristup je korištenje modela znanja koji koristi već spomenuta pravila koja mogu imati do tri komponente: događaj, uvjet i akcija. On zapravo predstavlja semantiku ECA pravila.

11.2. Događaj

Događaj je nešto što nastane u određenom trenutku. Navođenje događaja u pravilu pruža opis događaja koji se treba pratiti. Opis događaja i način na koji se on može detektirati u velikoj mjeri ovisi o njegovom izvoru.

Mogući izvori (Paton & Diaz, 1999):

- Operacije nad strukturom baze
- Operacije definirane od strane korisnika
- Transakcije
- Izuzetci (pokušaj neovlaštenog pristupanja podacima)
- Vremenski (u određenom trenutku)
- Vanjski (događaj izvan baze podataka)

Razlikujemo primitivne (jednostavne) i složene događaje. U primitivne događaje spada ažuriranje podataka, prikaz podataka, vrijeme itd. Složeni događaji se formiraju kombinacijom primitivnih i prethodno definiranih složenih događaja. Kombinacije mogu biti složene korištenjem logičkih operatora (AND, OR, NOT itd), sekevnci (pravilo se može pokrenuti kada se dva ili više događaja pojave u određenom, definiranom redoslijedu) i pomoću vremenskih kompozicija (10sek nakon određenog događaja) (Paton & Diaz, 1999).

Zrnastost događaja prikazuje da li je događaj definiran za svaki objekt u grupi objekata (npr. svaka instanca klase), za pojedine podskupine (npr. svi zaposlenici osim profesora) ili za pojedine članove skupina (npr. točno određeni zaposlenik), sve u skladu s tim spriječi neovlašteni pristup.

11.3. Uvjet

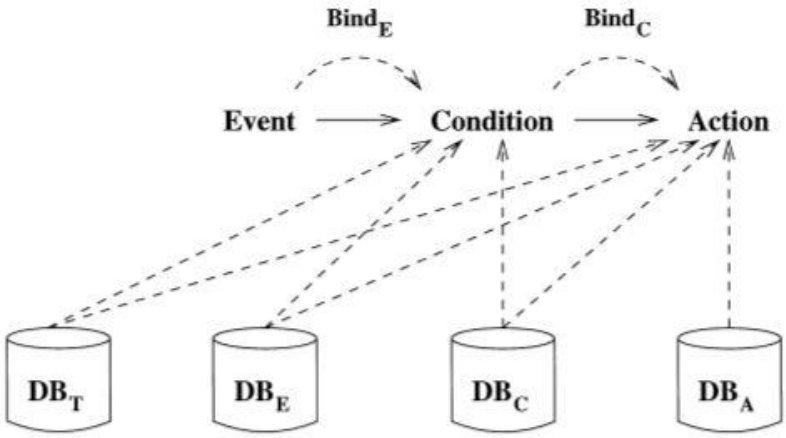
Kontekst komponente uvjet prikazuje situacije u kojima se uvjet evaluira odnosno vrednuje. Različite komponente pravlia se ne evaluiraju u izolaciji od baze podataka ili jedna od druge. Kao rezultat toga, obrada jednog pravila potencijalno može biti povezana sa najmanje četiri različitih stanja baze podataka (Paton & Diaz, 1999):

- DBT - stanje baze podataka na početku transakcije,
- DBE - stanje baze podataka u trenutku događaja,
- DBC - stanje baze podataka kad je uvjet evaluiran i
- DBA - stanje kada je akcija izvršena.

11.4. Akcija

Raspon zadataka koje komponenta akcija može obavljati je definiran kao njezine mogućnosti. Akcije mogu ažurirati strukturu baze podataka ili set pravila, izvršavati određeno ponašanje unutar baze, obavijesiti korisnika ili sistem administratora o određenoj situaciji, mogu prekinuti izvršavanje transakcije itd. Kontekst akcije je sličan kontekstu uvjeta. On

prikazuje informaciju koja je dostupna akciji (prikazano na slici 7). Moguće je informaciju iz uvjeta pravila proslijediti njegovoj akciji. Na primjer spomenuto pravilo se može koristiti za prepravljanje podataka pohranjenih u nekom atributu koji su zahvaćeni promjenom podataka nekog drugog atributa (Paton & Diaz, 1999).



Slika 7: Prikaz konteksta u kojem se pravilo izvršava (Izvor: Paton & Diaz, 1999)

11.5. Model izvršavanja

Model izvršavanja je u suprotnosti sa modelom znanja. On koji određuje kako se postupa sa skupom pravila za vrijeme izvršavanja. Model izvršavanja predstavlja obradu ECA pravila. Obradu pravila obavlja mehanizam aktivnih pravila koji prati događaje uzrokovane transakcijama i ostalim promjenama. Ta obrada nam jamči reaktivno ponašanje baze podataka, što je u suprotnosti sa ponašanjem pasivnih baza podataka.

Način razmatranja aktivnih pravila u odnosu na događaj koji se prati može biti **neposredan (trenutan)**, **odgođen** ili **nezavisan** (Zaniolo, et al., 1997).

Neposredno razmatranje – može se dogoditi prije (BEFORE), poslije (AFTER) ili umjesto (INSTEAD OF) događaja koji se prati.

Odgođeno razmatranje – može se dogoditi na kraju transakcije (pokrenuto transakcijskom naredbom COMMIT WORK) ili nakon korisnički definiranih naredbi.

Nezavisno razmatranje – javlja se u kontekstu odvojene transakcije koja je proizašla iz inicijalne transakcije nakon nastanka određenog događaja.

Izvršavanje akcija relativnih načinu razmatranja može biti **neposredno (trenutno)**, **odgođeno** ili **nezavisno** (Zaniolo, et al., 1997).

Neposredno izvršavanje – izvršavanje akcije odmah slijedi nakon razmatranja uvjeta, koristi se u gotovo svim slučajevima.

Odgođeno izvršavanje – izvršavanje akcije se odgađa do kraja transakcije ili korisnički definirana naredba započinje izvršavanje pravila.

Nezavisno izvršavanje – odvija se u kontekstu zasebne transakcije koju pokreće inicijalna transakcija nakon razmatranja uvjeta. U tom slučaju može doći do klauzalnih zavisnoti između početne i odvojene transakcije.

11.5.1. Prioriteti izvršavanja pravila

U mnogim sustavima, više pravila se mogu aktivirati u isto vrijeme i kažemo da ta pravila pripadaju skupu sukoba. Zbog toga i algoritam pravila za obradu pravila mora imati određenu strategiju za odabir pravila koje će se izvršiti. U aktivnim bazama podataka, odabir sljdećeg pravila kojeg treba obraditi obično se javlja nakon svakog razmatranja pravila i njegovog mogućeg izvršavanja. Na ovaj način, sljedeće pravilo koje se treba izvršiti može biti aktivirano akcijom prethodno izvršenog pravila. Alternativno, odabir pravila može se riješiti i dodavanjem liste pravila algoritmu za obradu pravila, koja se zatim razmatraju i izvršavaju jedno za drugim sve dok se lista ne isprazni. Ovakav način odabira obično se koristi u ekspertnim sustavima u kontekstu primjene umjetne inteligencije. Odabir pravila iz skupa sukoba je pod utjecajem prioriteta (Zaniolo, et al., 1997).

Načini određivnja prioriteta (Zaniolo, et al., 1997)**:**

- Pravila se mogu odrediti zajedno sa numeričkim prioritetima koji definiraju njihov konačan redoslijed
- Pravila se mogu odrediti zajedno sa numeričkim ili relativnim prioritetima koji definiraju njihov djelomičan redoslijed. U ovom slučaju, ili sustav čuva sistemski definiran konačan redoslijed koji je u skladu sa korisnički definiranim djelomičnim redoslijedom, ili sustav sam nedeterministički odabire pravila s višom prioritetnom razinom.
- Pravila ne moraju imati eksplicitne prioritetne mehanizme. U ovom slučaju isto, ili sustav čuva sistemski definiran ukupna redoslijedi, ili nedeterministički odabire sva pravila.

U većini slučajeva sustavi imaju sistemski definirane prioritete. U takvim slučajevima izvršenja su ponavljajuća: dva izvršenja iste transakcije na istoj instanci baze podataka sa istom grupom okidača kao rezultat daju isti redoslijed izvršavanja.

11.6. Tranzicijske vrijednosti

Tranzicijske vrijednosti su privremeni podaci koji opisuju promjene stanja uzrokovane transakcijom na čiji sadržaj utječe razina zrnatosti pravila, o čemu ovisi i način njihovog dohvaćanja.

Kada se promjene prate na razini instance, tranzicijske vrijednosti se odnose na samo jedan objekt te se one obično dohvaćaju pomoću ključnih riječi OLD i NEW.
Kada se promjene prate na razini skupine objekata, tranzicijske vrijednosti se kolektivno mijenjaju te se prikupljaju u privremene strukturue kao što su tablice INSERTED i DELETED. Promjene se mogu tretirati eksplicitno (npr. pomoću dvije tablice OLD-UPDATE i NEW-UPDATE) ili implicitno (npr. dodavanjem njihovog prijašnjeg stanja u tablicu DELETED, a stanja poslije u tablicu INSERTED).

11.7. Eksterna i interna pravila

Aktivna pravila mogu zadovoljiti različite potrebe primjene. Općenito govoreći razlikujemo interne i eksterne primjene aktivnih pravila.

Interne primjene podržavaju klasnične značajke za upravljanje bazama podataka, kao što su održavanje intregriteta baze podataka, održavanje podataka, održavanje sigurnosti, praćenje događaja itd. Ta pravila su često sistemski generirana i skrivena za korisnike.

Primjeri eksterne primjene aktivnih pravila su: pravila za upravljanje zalihama, za prikazivanje različitih upozorenja ili poruka, bez promjene sadržaja podataka, pravila za praćenje vrijednosnih promjena (trgovanje obveznicama) itd. Takva pravila su najprikladnija za korištenje u aplikacijama koje imaju slične ili iste strategije. Programski kod koji bi imala svaka od tih aplikacija se premješta u aktivno pravilo, i na taj način se ono centralizira, odnosno pravilo više ne o ovisi o aplikacijama te postiže neovisnot.

12. Transakcije

„Transakcija je niz logički povezanih operacija koje se izvršavaju kao cjelina i prevode bazu podataka iz jednog u drugo konzistentno stanje.“[10]

Transakcija je slijed operacija (čitanja i pisanja podataka) koje jedan korisnik ili aplikacija provodi nad bazom podataka. Kažemo da je transakcija jedinica rada nad bazom podataka. Da bi se očuvao integritet baza podataka svaka jedinica provedena nad bazom podataka mora biti provedena u cijelosti ili ne smije biti provedena uopće. Transakcija koja iz bilo kojeg razloga nije do kraja bila obavljena morala bi biti neutralizirana, odnosno svi podaci koje je ona do trenutka prekida promijenila morali bi natrag dobiti svoje polazne vrijednosti. Jedan od najčešćih slučajeva pogreške u bazi je upravo neizvršavanje transakcije u potpunosti i neizvršavanje neutralizacije. Kod ulaska u transakciju otpuštaju se sva zaključavanja ako su pokrenuta (Anon., n.d.).

Početak transakcije se obilježava ključnim riječima `START TRANSACTION` ili `BEGIN WORK`, a završetak sa `COMMIT WORK` ili `ROLLBACK`. `COMMIT WORK` označava uspješan završetak i sve promjene učinjene prilikom transakcije mogu se pohraniti i tada postaju vidljive. Potvrđena izmjena nikad ne može biti poništena. Sustav garantira da će njezine izmjene biti permanentno pohranjene u bazi podataka, čak i ako kvar nastane već u sljedećem trenutku. `ROLLBACK WORK` označava neuspješan završetak i sve promjene učinjene prilikom transakcije, do ključne riječi `START` ili `BEGIN`, se poništavaju, kao da se nisu ni dogodile. Koristi se za prekid transakcije unutar koje je uočena pogreška ili se obavlja automatski kada je transakcija prekinuta izvana (Anon., n.d.).

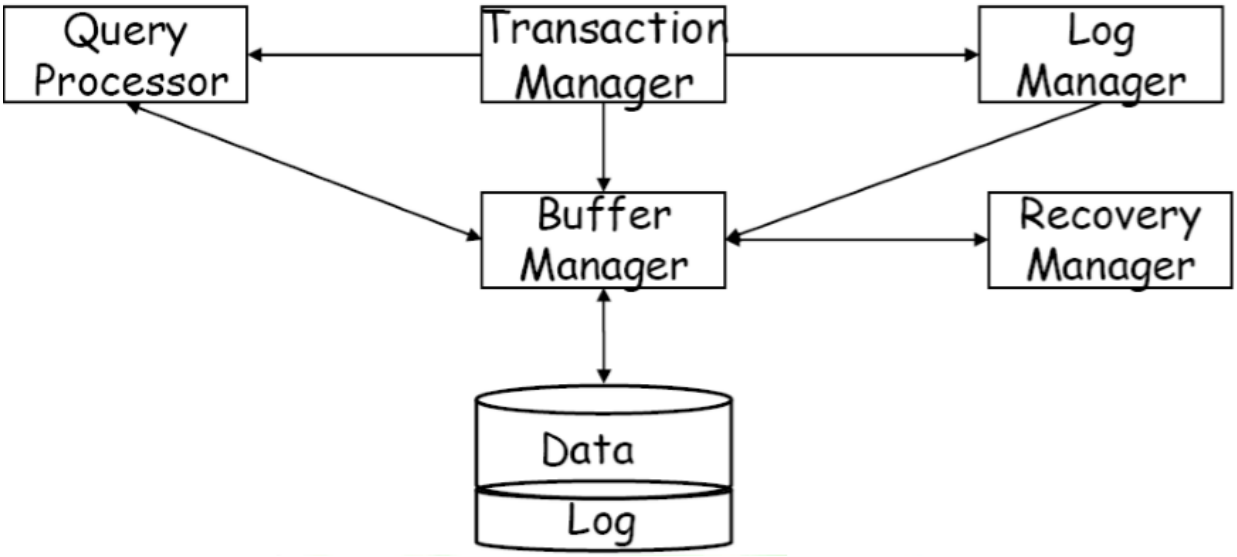
Svaka operacija izmjene u bazi podataka predstavlja transakciju, njezin uspješan završetak predstavlja `COMMIT`, a neuspješan `ROLLBACK`. Transakcije se ne mogu ugnježđivati te jedna aplikacija u jednom trenutku može imati aktivnu samo jednu transakciju. One ne podržavaju opoziv (rollback) DDL naredbi (Anon., n.d.).

Ako želimo da više operacija (SQL naredbi) predstavlja jednu transakciju, potrebno je postaviti varijablu `AUTOCOMMIT` na 0. Vrijednost `AUTOCOMMIT` inicijalno je postavljena na 1 (MySQL). Pretpostavljeni (eng. *default*) način rada je takav da se svaka naredba poziva i izvršava posebno. Svaka sesija se pokreće s postavljenim autocommit načinom rada. U

[10] Baranović, M. i Zakošek, S., (valjača 2012.) Baze podataka.

slučaju kada se onemogući autocommit način (`SET AUTOCOMMIT=0;`) i seseija se nenadano prekine, MySQL automatski napravi opoziv transakcije.

Dio sustava koji brine o obavljanju transakcija naziva se upravitelj transakcijama (eng. *transaction processing monitor* – TP monitor). On osigurava zadovoljavanje svih poznatih pravila integriteta (Anon., n.d.).



Slika 8: Transaction manager (Izvor: Anon., n.d.)

12.1. Točke pohranjivanja

Postavljanje točke pohranjivanja služi kako bi se kasnije mogao napraviti opoziv do te točke.

Primjer za postavljanje točke pohranjivanja:

```
SAVEPOINT imeTockePohranjivanja
```

Primjer opoziva svih naredbi nakon imenovane točke opoziva:

```
ROLLBACK TO SAVEPOINT imeTockePohranjivanja
```

Primjer otpuštanja postavljene točke pohranjivanja:

```
RELESE SAVEPOINT
```

Primjer: Potrebno je radniku Petru Kruljcu povećati koeficijent plaće za 0.5, te istovremeno osigurati smanjenje koeficijenta za 0.5 radniku Dini Parlovu.

```
SET AUTOCOMMIT = 0;
BEGIN;
UPDATE radnik
     SET koefPlaca=koefPlaca+0.5
```

```
     WHERE prezimeRadnik = 'Kruljac' AND imeRadnik='Petar';
SAVEPOINT tocka;
UPDATE radnik
     SET koefPlaca=koefPlaca-0.5
     WHERE prezimeRadnik = 'Parlov' AND imeRadnik='Dino';
ROLLBACK TO SAVEPOINT tocka;
SELECT * FROM radnik
     WHERE (prezimeRadnik = 'Kruljac' AND imeRadnik='Petar') OR
(prezimeRadnik = 'Parlov' AND imeRadnik='Dino');
COMMIT;
SET AUTOCOMMIT = 1
```

12.2. Stanje transakcije

Jedna transakcija s korisničkog stanovišta predstavlja jednu nedjeljivu cjelinu koja se obično realizira kao niz od nekoliko elementarnih zahvata u samoj bazi. Glavno svojstvo transakcije je da prevodi bazu iz jednog konzistentnog stanja u drugo. Međustanja koja nastaju nakon pojedinih operacija unutar transakcije mogu biti nekonzistentna.

Stanja transakcije (Anon., n.d.):

- **Aktivna** (active) – tijekom izvođenja
- **Djelomično završena** (partially committed) – nakon što je obavljena njezina posljednja operacija
- **Neispravna** (failed) – nakon što se ustanovi da nije moguće nastaviti njezino normalno izvođenje
- **Neuspješno završena** (aborted) – nakon što su poništeni njezini efekti i baza podataka vraćena u stanje kakvo je bilo prije nego što je započela
- **Potvrđena** (committed) – uspješno završena
- **Točka potvrđivanja** (commit point) – sve izmjene koje je transakcija napravila postaju permanentne; sve izmjene koje je transakcija načinila prije točke potvrđivanja mogu se smatrati pokušajima; u točki potvrđivanja otpuštaju se svi ključevi

12.3. Svojstva transakcije

Svojstva transakcije još se nazivaju i ACID (eng. *atomicity, consistency, isolation, durability*) svojstva (Anon., n.d.).

Atomarnost (*eng. Atomicity*)

- transakcija se mora obaviti u cijelosti ili se uopće ne smije obaviti.
- transakcija može biti neizvršena u cijelosti u tri slučaja: prekinuta ili neuspješno završena, prilikom pada sustava, nepredviđene situacije.

Konzistentnost (*eng. Consistency*)

- transakcijom baza podataka prelazi iz jednog konzistentnog stanja u drugo konzistentno stanje.
- korisnik koji potvrđuje transakciju mora osigurati da će transakcija ostaviti bazu podataka u ispravnom stanju. Npr. ako definiramo prebacivanje određenog iznosa s računa klijenta A klijentu B, nakon što oduzmemo iznos s računa A, moramo ISTI iznos dodati računu B (SUBP se neće sam za to pobrinuti).

Izolacija (*eng. Isolation*)

- kada se paralelno obavljaju dvije ili više transakcija, njihov učinak mora biti jednak kao da su se obavljale jedna iza druge.

Izdržljivost (*eng. Durability*)

- ako je transakcija obavila svoj posao, njezini efekti ne smiju biti izgubljeni ako se dogodi kvar sustava, čak i u situaciji kada se kvar dogodi neposredno nakon završetka transakcije.
- ako je do kvara došlo za vrijeme izvođenja transakcije, SUBP se mora pobrinuti za vraćanje u konzistentno stanje pomoću logova. Prije nego se odradi, svaka promjena na disku evidentirana je u logovima.

12.4. Serijabilnost

Neka se u višekorisničkoj bazi podataka izvodi nekoliko transakcija paralelno tako da se pojedini dijelovi tih transakcija izvode vremenski izmiješano. Ako je konačni učinak njihovog izvođenja isti kao da su se one izvršavale serijski (sekvencijalno), kažemo da se radi o serijabilnom (ili serijalizabilnom) izvršavanju transakcija.

12.5. Paralelni pristup podacima

Paralelni pristup podacima je jako bitan jer on povećava broj obavljenih transakcija u jedinici vremena, te se smanjuje prosječno vrijeme između aktiviranja i završetka transakcije.

Jako je bitno kontroliratati paralelni pristup podacima i **ne dozvoliti istovremenu izmjenu podataka** - može dovesti do izgubljenih podataka, posljednja izmjena podataka se sprema (Oracle, 2017).

Primjer – Rezervacija kino ulaznica
Prodavač A pročita da ima 20 slobodnih mjesta, isti broj slobodnih mjesta pročita i prodavač B. Prodavač rezervira 3 mjesta, a sustav promijeni broj slobodnih mjesta sa 20 na 17. Prodavač B u istom trenutku rezervira 2 mjesta, a sustav kod njega promijeni broj slobodnih mjesta sa 20 na 18. Na kraju ovih transakcija sustav će zabijležiti da u kinu još ima 18 slobodnih mjesta umjesto 15. Iz primjera vidimo da dolazi do izgubljnih podatka. Posljednja izmjena pobjeđuje!

Ne dozvoliti čitanje podataka koje netko drugi mijenja i time spriječiti prljavo čitanje, neponovljivo čitanje i sablasti .

Primjer – Prljavo čitanje
U primjeru koristiti ćemo dva programa, A i B. Prvi program A izvršava transakciju koja ažurira vrijednosti određene tablice. Nakon što su vrijednosti ažurirale program B izvršava upit čitanja podataka iz ažurirane tablice. Upit će vratiti sve podatke zajedno i sa onim ažuriranim. Zatim program A pomoću naredbe ROLLBACK poništava sve napravljene promjene, tako da kada program B ponovo izvrši isti upit rezultat će biti drugačiji od prvotnog.

Primjer – Neponovljivo čitanje

Imamo dvije transakcije koje se paralelno izvršavaju. Prva transakcija A na početku izvršava upit čitanja podatka iz određene tablice. Nakon toga transakcija B ažurira taj isti podatak u toj tablici. Na kraju, transakcija A ponavlja upit čitanja tog istog podatka te dobije rezultat koji se razlikuje od onog sa početka transakcije. Na taj način više ne možemo dobiti rezultat iz prvog čitanja.

Primjer – Sablasne n-torke

Opet imamo dvije transakcije koje se paralelno izvršavaju. Prva transakcija na početku prebrojava broj studenta na fakultetu te dobija rezultat koji kaže da postoji 300 studenata. Nakon prebrojavanja druga transakcija briše 2 studenta iz sustava. Na kraju izvršavanja prve transakcije opet se izvršava upit čitanja broja studenata koji ovaj put vraća rezultat od 298 studenata, što znači da su izgubljena dva retka u tablici.

Ovi problemi se rješavaju zaključavanjem podataka. Transakcija može zaključati podatke, te time spriječiti pristupanje tim podacima od strane drugih transakcije sve dok ga ta ista transakcija ne otključa. Podaci koji su se mijenjali tijekom transakcije ostaju zaključani do kraja transakcije. Dio SUBP koji upravlja time naziva se locking manager (Oracle, 2017).

Primjer – Rješenje problema izgubljenih izmjena (rezervacija kino ulaznica)

Prodavač A zaključa tablicu za sebe i pročita da još ima 20 slobodnih mjesta u kinu. Za to vrijeme prodavač B ne može izvršiti čitanje tablice. Prodavač A rezervira 3 mjesta i zapiše da je broj preostalih mjesta sada 17, te zatim otključa tablicu. Nakon toga prodavač B zaključa tablicu i ovaj put izvršava uspješno čitanje tablice i dobija rezultat od 17 slobodnih mjesta. Zatim prodavač B rezervira 2 slobodna mjesta, te sustav zapisuje da je preostalo 15 slobodnih mjesta. Nakon izvršene promjene pordavač B otključa tablicu. Na kraju ažuriranja u sustavu je zapisano da je broj slobodnih mjesta 15.

DBMS treba omogućiti paralelno izvođenje u što je moguće većoj mjeri. Efekt transakcija koje izvršavaju paralelno mora biti isti kao da su se izvršavale jedna iza druge (**serijabilnost**). Serijabilnost je osigurana ako sve transakcije poštuju protokol dvofaznog zaključavanja (2PL – two phase locking protocol). Prije obavljanja operacije nad objektom (npr. n-torkom iz baze), transakcija mora za taj objekt zatražiti ključ, a nakon otpuštanja ključa transakcija više ne smije zatražiti nikakav ključ (Oracle, 2017).

Transakcije koje poštuju 2PL protokol imaju dvije faze:

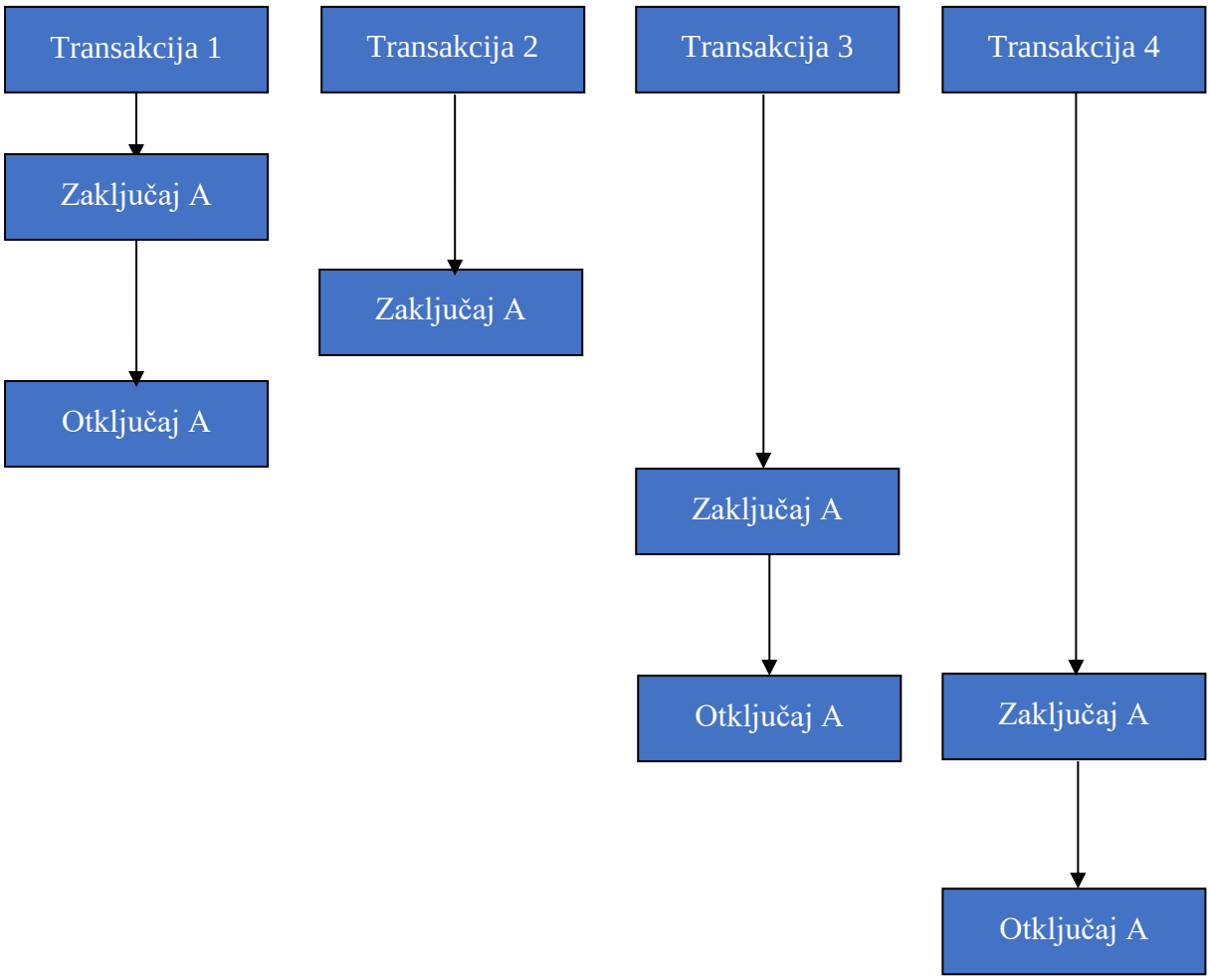
- **Faza pribavljanja ključeva**
- **Faza otpuštanja ključeva** - ova faza je najčešće izvedena kroz jednu operaciju (COMMIT ili ROLLBACK na kraju transakcije)

Vrste zaključavanja:

- **Ključ za pisanje/izmjenu – WRITE LOCK**
    - Još se naziva i ekskluzivno zaključivanje. Transakcija zaključa objekt za pisanje te niti jedna druga transakcije ne može zaključati objekt dok ga prva transakcija ne otključa. Svaka operacija izmjene podataka postavlja ključ za pisanje.
- **Ključ za čitanje – READ LOCK**
    - Još se naziva i zajedničko čitanje. Transakcija zaključa objekt za čitanje. Bilo koja druga transakcija može ga zaključati za čitanje, ali niti jedna ga ne može zaključati za pisanje.
- **Unaprijedivi ključ – PROMOTABLE LOCK**
    - Ovaj ključ izvršava namjeru izmjene podataka. Na početku transakcije postavlja se ključ za čitanje, te bilo koja druga transakcija može postaviti ključ za čitanje, dok niti jedna transakcija ne može postaviti ključ za pisanje. Prije izmjene podataka pomoću prve transakcije ključ se unaprijeđuje u ključ za pisanje. On omogućuje DBMS-u predviđanje budućih akcija. Unprijedivi ključ za sada ne postoji u MySQL bazi podataka.

Problemi koji se mogu pojaviti kod zaključavanja su **nepotpuni zastoj** (LIVE LOCK) i **potpuni zastoj** (DEAD LOCK).

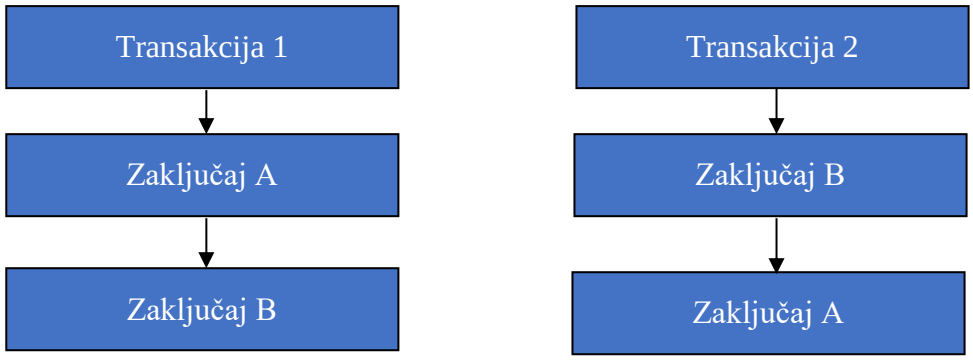
Nepotpuni zastoj



Slika 9: Nepotpuni zastoj

Iz ovog primjera je vidljivo da je moguće da transakcija zavijek čeka na zaključavanje, iako je mnogo puta imala priliku. Rješenje ovog problema je korištenje strategije **FIRST-COME-FIRST-SERVED**.

Potpuni zastoj



Slika 10: Potpuni zastoj

Potpuni zastoj se može izbjeći na nekoliko načina. Jedan od njih je da transakcija zatraži sva zaključavanja odjednom ili da transakcije zaključavaju podatke u nekom određenom poretku. Moguće je postaviti i da ako transakcija ne dobije ključ tijekom zadanog vremenskog odsječka menadžer transakcija pretpostavi da se ključ ne može postaviti upravo zbog pojave potpunog zastoja te poništi transakciju. To se naziva vremenski bazirana prevencija poptunog zastoja. U tom slučaju, može se dogoditi da će se poništiti transakcije koje zapravo ne sudjeluju u potpunom zastoju. Za detekciju poptunog zastoja koristi se i wait-for graf.

Kod zaključvanja podataka moguće je odrediti granulaciju zaključavanja odnosno razinu koja će biti zaključana: baza podataka, tablica (relacija), redak (n-torka) ili indeks. Koristi se uglavnom za stvaranje backupa (budući da zaustavlja trenutni rad baze). Da bi u potpunosti spriječili pojavljivanje sablasnih n-torki trebamo se pridržavati pravila sljedećih pravila (Oracle, 2017):

- Svaka relacija mora imati barem jedan indeks
- Transakcija Ti može pristupati n-torkama relacije isključivo preko jednog ili više indeksa relacije
- Ako transakcija Ti pregledava podatke – mora postaviti dijeljeni ključ na sve indeksne listove kojima pristupa
- Transakcija Ti ne može dodavati, mijenjati ili brisati n-torku ti u relaciji r bez izmjene svih indeksa na r
- Transakcija mora pribaviti ekskluzivne ključeve na svim indeksnim listovima na koje utječe unos, izmjena ili brisanje
- Moraju biti primijenjena pravila protokola dvofaznog zaključavanja

12.5.1. Razina izolacije

Razina izolacije određuje pristup podacima u višekorisničkom okruženju (Oracle, 2017):

- **Prljavo čitanje - DIRTY READ, READ UNCOMMITTED**
  - čitaju se svi traženi podaci bez zaključavanja i bez provjere jesu li podaci možda zaključani
  - problemi koji s mogu pojaviti: n-torke koje nikad stvarno nisu postojale u bazi podataka, neponovljivo čitanje, sablasne n-torke
- **Čitanje potvrđenih n-torki - READ COMMITTED**
  - proces koji provjerava je li podatak koji se trenutno čita već zaključan za pisanje
  - problemi koji s mogu pojaviti: neponovljivo čitanje, sablasne n-torke
- **Ponovljivo čitanje - REPEATABLE READ**
  - dohvat n-torke iz djelatnog skupa uzrokuje zaključavanje (za čitanje) odgovarajuće n-torke u bazi podataka
  - dohvatom nove n-torke prethodna ostaje zaključana
  - kod ponovnog dohvata istog djelatnog skupa unutar iste transakcije, prethodno zaključane n-torke i dalje su zaključane
  - zapisi/stranice ostaju zaključane do kraja transakcije
  - problemi koji s mogu pojaviti: sablasne n-torke
- **Serijabilnost– SERIALIZABLE**
  - isto kao i REPEATABLE READ
  - ostao je radi kompatibilnosti sa InnoDB mehanizmom
  - problemi koji s mogu pojaviti: sablasne n-torke

12.6. Pogreške u transkaciji, uzroci razrušenja i obnova baze

Tipovi pogrešaka (Oracle, 2017)**:**

- Pogreške koje otkriva sama aplikacija.
- Pogreške unutar transakcije kojima aplikacija ne rukuje na eksplicitan način (npr. dijeljenje s nulom).
- Kvar računalskog sustava - baza nije fizički uništena.
- Kvar medija za pohranu - fizički uništena baza.

Uzroci razrušenja (Oracle, 2017)**:**

- pogreške opreme,
- pogreške operacijskog sustava,
- pogreške SUBP-a,
- pogreške aplikacijskog programa,
- pogreške operatera,
- kolebanje izvora energije,
- sabotaža,
- prirodne katastrofe.

Obnova baze u slučaju razrušenja (Oracle, 2017)**:**

- Dovesti bazu podataka u najnovije stanje za koje se pouzdano zna da je bilo ispravno
- Velike baze podataka, dijeljene i višekorisničke moraju obavezno posjedovati mehanizme obnove.
- Male jednokorisničke baze često nemaju mehanizme obnove već se u nekom vremenskom intervalu radi sigurnosna kopija podatka na čije se stanje vraća sustav u slučaju razrušenja

12.7. Način zapisivanja podataka u bazu

Podaci se u bazu podataka zapisuju pomoću međuspremnika. To može biti međuspremnik dnevnika ili međuspremnik baze podataka. Sadržaj tih međuspremnika zapisuje se u dnevnik odnosno bazu podataka kad se popuni ili kada SUBP za to izda nalog.

Ako kvar nastane nakon potvrđivanja i prije nego što se izmjene iz memorijskih spremnika prebace u bazu podataka, procedura za ponovno pokretanje provest će te promjene u bazi podataka. Vrijednosti koje je potrebno zapisati u bazu podataka nalaze se u odgovarajućim zapisima u dnevniku izmjena. U suprotnom slučaju izmjene bi u trenutku kvara bile izgubljene (Oracle, 2017).

12.8. Dnevnik baze podataka

Dnevnik baze podatka, jedan od naziv je i transaction log, sadrži sve zaspise o transakcijama i izmjene u bazi podataka uzrokovane transakcijama. Dnevnik je ključna komponenta baze podataka. U slučaju razrušenja, ti logovi će biti potrebni za vraćanje baze u konzistentno stanje. Postupak pretraživanja dnevnika predugo traje te kao rješenje tog problema nudi se uvođenje kontrolnih točaka ili kako se nazivaju u MySQL bazama broj dnevnika, koje predstavljaju trenutak u kojem je baza bila u konzistentnom stanju (Oracle, 2017).

12.9. Proces obnove

Na početku obnove u dnevniku baze podataka pretražuju se sve transakcije koje su bile aktivne od kontrolne točke te se svaka transakcija za koju se pronađe `BEGIN` dodaje u listu za poništavanje, dok se transakcija za koju se pronađe `COMMIT` prebacuje iz liste za poništavanje u listu za ponovno obavljanje. Na kraju procesa sve transakcije iz liste za ponovno obavljanje se ponovo obavljaju, dok se transakcije sa liste za poništavanje poništavaju (Oracle, 2017).

SUBP ne može prihvatiti ni jedan zahtjev dok se ne završi proces obnove!

13. Pohranjeni zadaci

Pohranjene zadatke promatramo kao korisnički definirane funkcije koje nam omogućuju slijedno izvršavanje većeg broja SQL naredbi u obliku samo jedne naredbe. To su neovisni dijelovi koda namijenjeni višekratnoj uporabi koja enkapsulira određene radnje, a mogu se pozivati (unutar aplikacija) prema potrebi. Pohranjeni podaci proširuju mogućnosti SQL jezika uporabom globalnih i lokalnih varijabli, naredbi za kontrolu toka i naredbi za rukovanje iznimkama (Anon., n.d.).

Prilikom porasta složenosti poslovne logike aplikacija koje se vežu na bazu podataka, česta pojava je repetitivno pisanje niza istih SQL naredbi. Pohranjeni zadaci se pohranjuju na serveru čime se izbjegava ponovno pisanje SQL naredbi. Primjerice, više puta piše se ista `INSERT` naredba za unos n-torke u tablicu ili isti izračuni nad određenim setom podataka. Tada je poželjno pohraniti niz ovakvih naredbi u pohranjeni zadatak koji će se nalaziti na poslužitelju baze podataka i izvršavati na način da aplikacija na klijentskoj strani ne mora prenositi niz naredbi do poslužitelja baze, već samo ime zadatka koji će obaviti upite nad bazom.

Prednosti (Duk, 2012):

- **Pohranjeni zadaci nalaze se na poslužitelju baze podataka** umjesto u samoj aplikaciji. U kontekstu klijent-poslužitelj arhitekture, brže je i manje se opterećuje prijenosni kanal ako se prenosi samo naziv pohranjenog zadatka, u odnosu na prijenos niza naredbi koje se trebaju izvršiti.
- **Izbjegava se repetitivno pisanje istih naredbi** čime se smanjuje mogućnost pogreške. Time se također dobiva na preglednosti i efikasnosti samoga koda aplikacije.
- **Povećana je prenosivost aplikacije na drugu platformu**. Razna sučelja i aplikacije napisane primjerice u jeziku Java, VB, Excel, pozivaju se primjerice pomoću JDBC, JDBC/ODBC ili ODBC programskih sučelja istog pohranjenog zadatka.
- **Povećana je sigurnost i izolacija podataka** jer postoji mogućnost dodjeljivanja dozvola (GRANT) za pristup podacima na razini pohranjenih zadataka. SPL (Stored Procedure Language) omogućava zaštitu podataka od neovlaštene uporabe na razini pohranjenih zadataka. Korisniku se pridijeli dozvola za obavljanje

definiranog pohranjenog zadatka, umjesto dozvole za pristup podacima. Time je precizno određen način na koji korisnik smije obaviti operacije nad podacima.

- **Logika baze podataka ostaje u samoj bazi**, čime se povećava robusnost aplikacija na razini konzistentnosti podataka i izbjegava nedosljednost podataka.
- **Nadograđene su osnovne funkcionalnosti** baze podataka aritmetikom, varijablama i petljama.

Nedostaci (Duk, 2012):

- Umjesto dohvata podataka za što su baze optimizirane, od pohranjenog zadatka očekuje se rad s logičkim operacijama i poslovnom logikom. Posljedica je opterećenje poslužitelja odnosno opterećenje CPU-a i memorije.
- Postoje ograničenja u pohranjenim zadacima u obliku ograničenosti naredbi koje se mogu koristiti. SPL nije kompleksan kao C++, VB, Java i ostali.
- Upravljanje pogreškama (error handling) je loše što uzrokuje složenost razvoja pohranjenog zadatka jer je potrebno dobro znanje SQL jezika.
- Nedostatkom se može smatrati što je poslovna logika u bazi umjesto u aplikaciji. (Ovo je prednost u slučajevima operacija nad kompleksnim setom rezultata, kada je bolje koristiti pohranjeni zadatak kako se složenost entiteta ne bi prenosila u aplikacijski sloj.)
- Dostatno odstupanje od SQL standarda među različitim tipovima baza kako bi se osigurala prednost te baze

Pohranjene zadatke dijelimo na:

- **Pohranjene rutine** (pohranjene procedure i pohranjene funkcije)
- **Okidači**
- **Događaji**

13.1. Pohranjene rutine

Pohranjene rutine djelimo: **pohranjene procedure** i **pohranjene funkcije**.

Deklaracija procedura i deklaracija funkcija međusobno su vrlo slične te je sličan i način njihovog izvođenja. Razlike između procedura i funkcija mogu se svesti na sljedeće (Duk, 2012):

- Funkcije ne mogu vratiti skup rezultata.
- Funkcije ne mogu koristiti SQL naredbe koje podrazumijevaju transakcijsku obradu (COMMIT, ROLLBACK).
- Funkcija ne može sama sebe rekurzivno pozivati.
- Funkcije moraju vratiti rezultat.

Nakon što se procedure i funkcije pohrane, više ne možemo mijenjati njhova tijela. Pohranjene rutine mogu se backupirati zajedno s bazom.

13.2. Varijable

Pohranjeni zadaci sa bazom podataka komuniciraju putem varijabli. Razlikujemo:

- **GLOBALNE VARIJABLE**
  - varijable u MySQL-u koje počinju znakom @,
  - vrijede u jednoj sesiji (spoju na poslužitelj),
  - nije ih potrebno deklarirati.
- **LOKALNE VARIJABLE**
  - varijable deklarirane unutar bloka BEGIN … END pohranjenog zadatka,
  - ne moraju početi sa znakom @,
  - vidljive su unutar bloka pohranjenog zadatka,
  - deklaracija se mora nalaziti uvijek na vrhu procedure, poslije BEGIN i obavezno prije početka nekoga drugoga koda.

Varijabli je moguće dodijeliti vrijednost ključnom riječi **SET**:

```
SET @var2 = 5678;
```

U varijablu je moguće pohraniti (samo jednu) vrijednost izvođenja upita koristeći ključnu riječ **INTO**:

```
SELECT AVG(koefPlaca) INTO @prosjek2 FROM radnik;
```

14. Pohranjene procedure

Za pohranjene procedure karakteristično je da obavljaju niz SQL naredbi te da mogu vratiti skup rezultata. Rezultat ne mogu vratiti direktno (kao što će to kasnije moći funkcija) već rezultat vraćaju preko varijabli ili koristeći `SELECT` naredbu. Procedura može primiti varijable koje mogu biti modificirane unutar tijela procedure (Duk, 2012).

Procedura se definira prema sljedećim koracima:

1. Izradi proceduru (ključna riječ CREATE PROCEDURE).
2. Definiraj ime procedure.
3. Definiraj ulazne i izlazne parametre.
4. Definiraj tijelo procedure (ako se procedura sastoji od više naredbi, potrebno ih je definirati unutar BEGIN … END bloka).

Osnovna sintaksa:

```
CREATE PROCEDURE ime_procedure ([parametri[,...]])
        tijelo_procedure
```

Pohranjenom zadatku može se dodijeliti ime od najviše 64 znakova, pri čemu se ime ne smije sastojati samo od brojki. Ime može sadržavati i posebne znakove, ali u tom slučaju moraju se označiti s unazadnim navodnikom (`). Preporuča se izbjegavati nazivanje pohranjenih podataka imenima baze, tablice ili atributa. Procedura ne smije imati ime rezervirano za neku ugrađenu funkciju (ne može se „overrajdati“).

Procedura se poziva ključnom riječi `CALL` nakon čega slijedi ime procedure te navođenje vrijednosti parametara koje se proceduri predaje i varijabli u koje se rezultat pohranjuje.

Osnovna sintaksa:

```
CALL ime_procedure ([parametri[,...]]
```

MySQL omogućava naredbom ALTER PROCEDURE izmijeniti samo određene karakteristike procedure. Ako postoji potreba za izmjenom tijela procedure, s obzirom da DBMS ne dozvoljava više procedura istog imena, potrebno je proceduru obrisati te je nakon toga ponovno izraditi sa izmijenjenim tijelom. (Duk, 2012).

Brisanje procedure

Osnovna sintaksa:

```
DROP PROCEDURE    [IF EXISTS] ime_procedure
```

Klauzula IF EXISTS sprečava pojavu pogreške prilikom izvođenja naredbe brisanja ako procedura s danim imenom ne postoji. U tom se slučaju neće dogoditi pogreška već će se samo ispisati upozorenje o nepostojanju procedure koja se pokušava obrisati. (Duk, 2012).

14.1. Delimiter (Graničnik)

Proceduru kao vrstu pohranjenog zadatka želimo izvesti na način da se izvedu sve naredbe od `CREATE PROCEDURE` do `END`. U pretpostavljenom načinu rada DBMS prekida izvođenje upita kada dođe do kraja trenutnog upita. Kraj upita označava graničnik (`DELIMITER`) čija je pretpostavljena vrijednost „ ;“ (točka zarez). Graničnik je znak ili niz znakova koji govori MySQL-u da se na tom mjestu nalazi kraj SQL naredbe. Da bi se izbjeglo zaustavljanje izvođenja pohranjenog zadatka prilikom nailaska na prvi „;“, potrebno je privremeno izmijeniti vrijednost graničnika na neku drugu vrijednost. Graničnik može poprimiti bilo koju vrijednost, ali se ne preporuča korištenje oznaka koje bi se mogle pronaći u tijelu procedure. Po završetku procedure graničnik je potrebno vratiti na njegovu inicijalnu vrijednost naredbom „`DELIMITER ;`“ (Duk, 2012).

Primjer procedure:

```
DELIMITER //
CREATE PROCEDURE ispisiStudente()
BEGIN
     SELECT * FROM studenti;
END; //
DELIMITER ;
```

```
CALL ispisiStudente();
```

14.2. Definiranje parametara

Proceduri se može definiraii ulazni parametar u odnosu na kojeg je u mogućnosti vratiti skup rezultata koji će ovisiti o tom parametru. Parametar nije ništa drugo već varijabla pomoću koje procedura komunicira s bazom podataka (Duk, 2012).

Postoje sljedeće vrste parametara:

- **Ulazni parametri – IN**
  - Koriste se kao ulaz u proceduru
  - Procedura ne može mijenjati vrijednost ove varijable
  - Ako se ne navede tip ulaza, podrazumijeva se IN
- **Izlazni parametri – OUT**
  - Koristi se samo kao izlaz iz procedure
  - Procedura može mijenjati vrijednost ove varijable
- **Ulazno-izlazni parametri – IN/OUT**
  - Koristi se kao izlaz i ulaz u proceduru

Osnovna sintaksa:

```
CREATE PROCEDURE imeProcedure ([IN] x tipPodatka,  INOUT y tipPodatka, OUT z tipPodatka)  ...tijelo procedure...
```

Primjer:

```
DELIMITER //
CREATE PROCEDURE prebrojiStudente(IN zadaniFakultet VARCHAR(50), OUT broj INT)
BEGIN
    SELECT COUNT(*) INTO broj FROM studenti WHERE nazivFakultet=zadaniFakultet;
END //
DELIMITER ;
```

```
CALL prebrojiStudente ('FOI,@n);
SELECT (@n);
```

Naredbom `CALL` rezultat neće biti ispisan, već će biti upisan u @a. Sadržaj globalne varijable @a ispisuje se naredbom `SELECT`.

15. Pohranjene funkcije

Pohranjena funkcija vrsta je pohranjene rutine koja direktno vraća rezultat te ne može vratiti skup rezultata. Za nju je karakteristično da s bazom podataka komunicira isključivo putem ulaznih parametara, dok izlazne parametre ne podržava. Podatke prima preko ulaznih varijabli koje može modificirati. S obzirom da rezultat ne može biti vraćen preko parametra, prilikom deklaracije funkcije nužno je navesti tip podatka kojeg će funkcija (obavezno) vratiti. Za razliku od procedura, funkcija mora vratiti rezultat i ne može ispisivati podatke unutar `BEGIN … END` bloka (Duk, 2012).

Funkcija se definira prema sljedećim koracima (Anon., n.d.):

1. Izradi funkciju (ključna riječ CREATE FUNCION)
2. Definiraj ime funkcije
3. Definiraj ulazne i izlazne parametre - svi su parametri ulaznog tipa (drugi tipovi nisu dozvoljeni) te se iz tog razloga ne navodi ključna riječ IN prije imena parametra
4. Ključna riječ RETURNS i tip parametra koji funkcija vraća
5. Definirati je li funkcija DETERMINISTIC ili NOT DETERMINISTIC
6. Definirati tijelo procedure (ako se procedura sastoji od više naredbi, potrebno ih je definirati unutar BEGIN … END bloka)
7. Unutar tijela funkcije naredbom RETURN definirati izlaz iz funkcije

Funkcija ne smije imati ime kao neka već ugrađena funkcija u MySQL-u.

Pravila koja su vrijedila za procedure, a tiču se definiranja graničnika, izmjene i brisanja procedura vrijede i za pohranjene funkcije.

Osnovna sintaksa:

```
CREATE FUNCTION ime_funkcije ([parametri[,...]]) RETURNS
tip_podatka
     [NOT] DETERMINISTIC [karakteristike]
     tijelo_funkcije
```

Funkcija se poziva ključnom riječi `SELECT` nakon čega slijedi ime funkcije te navođenje vrijednosti parametara koje se funkciji predaje.

Pozivanje funkcije

Osnovna sintaksa:

```
SELECT ime_funkcije ([parametri[,...]]);
```

MySQL omogućava naredbom `ALTER FUNCTION` izmijeniti samo određene karakteristike funkcije. Ako postoji potreba za izmjenom tijela funkcije, s obzirom da DBMS ne dozvoljava više funkcija istog imena, potrebno je funkciju obrisati te je nakon toga ponovno izraditi s izmijenjenim tijelom (Duk, 2012).

Brisanje funkcije

Osnovna sintaksa:

```
DROP FUNCTION [IF EXISTS] ime_funkcije;
```

Primjer funkcije:

```
DELIMITER //
CREATE FUNCTION vratiNajvecuPlacu() RETURNS INT
DETERMINISTIC
BEGIN
   DECLARE maxPlaca INT DEFAULT NULL;
   SELECT MAX(placaRadnik) INTO maxPlaca FROM radnik;
   RETURN maxPlaca;
END; //
DELIMITER ;
```

```
SELECT vratiNajvecuPlacu();
```

15.1. Definiranje parametara

Funkcija podržava isključivo ulazne parametre. Pri tome se ne smije navoditi IN kao oznaka tipa parametra jer druga opcija ne postoji.

Osnovna sintaksa:

```
CREATE PROCEDURE imeFunkcije (x tipPodatka)
      ...tijelo procedure...
```

Primjer:

```
DELIMITER //
CREATE FUNCTION prebrojiStudente(zadaniFakultet INT) RETURNS
INT
DETERMINISTIC
BEGIN
     RETURN (SELECT COUNT(*) FROM student WHERE nazivFakulteta=
     zadaniFakultet);
END; //
DELIMITER ;
```

```
SELECT brojiNaloge(456);
```

15.2. Deterministic & not determenistic

Prilikom deklaracije funkcije nužno je definirati je li funkcija deterministička ili je nedeterministička. Ako funkcija ima deterministično obilježje, tada se kaže da je strogo određena. Za iste će ulazne parametre uvijek vratiti isti rezultat. Navedeno se podrazumijeva u slučaju replikacije i u situacijama kada se unutar funkcije pozivaju ugrađene funkcije tipa RAND() i NOW(). Prenosi se naredba, a ne podaci koje generira funkcija (Duk, 2012).

Ako je funkcija non-deterministic i primjerice generira nasumične brojeve, kod prijenosa u repliciranu bazu dobile bi se različite vrijednosti te time dolazi do **nekonzistentnosti podataka** (Anon., n.d.).

16. Okidači

Okidač (eng. *trigger*) je objekt baze podataka koji se asocira s tablicom i aktivira kada se dogodi neki određeni događaj nad tablicom (umetanje, ažuriranje, brisanje) (Anon., n.d.). Čini ga niz definiranih SQL naredbi koje se automatski izvode prilikom unosa, izmjene ili brisanja podataka. Omogućava manipuliranje podacima prije nego što stvarno budu uneseni, izmijenjeni ili obrisani (Duk, 2012). Obzirom da se aktivira automatski, nije ga moguće zaobići putem aplikacije koja koristi tu istu bazu podataka. Kaže se da se okidač stvara ili definira nad objektom koji može biti tablica, pogled, schema ili baza podataka. Također se može specificirati vremenska oznaka koja određuje hoće li se okidač aktivirati prije ili poslije određenog događaja. Po defaultu, okidač se kreira u omogućenom (ENABLED) stanju.

Ako je okidač kreiran nad tablicom ili pogledom, onda se aktivirajuća operacija (eng. *triggering event*) sastoji od DML naredbi, a okidač se naziva **DML okidač**. Ako je okidač kreiran nad schemom ili nad bazom podataka, onda se aktivirajuća operacija sastoji ili od DDL naredbi ili od operacijskih naredbi baza podataka te se naziva okidač naziva **DDL okidačem**. Postoji i takozvani uvjetni okidač koji ima WHEN klauzulu.

Svi okidači u bazi se mogu pronaći u **INFORMATION_SCHEMA**.

Dvije osnovne karakteristike okidača su: ne primaju parametre ili argumente, ne mogu izvršiti commit ili rollback neke transakcije.

Okidači imaju tri osnovna dijela:

- **Triggering event/statement** - aktivirajuća operacija/događaj/naredba
- **Triggering restriction** - ograničenje, uvjet
- **Triggering action** - aktivirana operacija, aktivnost

Aktivirajuća operacija/naredba je SQL naredba, događaj baze podataka ili događaj korisnika koji uzrokuje da se okidač aktivira.

Aktivirajuće operacije (događaji) nad kojima se može stvoriti okidač mogu biti:

- **DML naredbe**: `INSERT`, `UPDATE` ili `DELETE` naredba nad određenom tablicom (ili pogledom)
- **DDL naredbe**: `CREATE`, `ALTER` ili `DROP` naredba nad schema objektom
- **Događaji baze podataka**:
  - `STARTUP`, `SHUTDOWN` (Pokretanje baze podataka ili gašenje)
  - `LOGON`, `LOGOFF` (Prijava ili odjava korisnika)
  - `SERVERERROR` (Određena poruka o grešci)

Ograničenje ili uvjet okidača navodi Boolean izraz koji mora biti TRUE kako bi se okidač aktivirao. Neće se aktivirati ako je ograničenje FALSE ili UNKNOWN.

Aktivirana operacija je procedura koja sadrži SQL naredbe i kôd koji se treba pokrenuti kada se određeni događaji pojave:

- aktivirajuća operacija je izdana
- ograničenje okidača je TRUE

Osnovna sintaksa za kreiranje okidača:

```
CREATE TRIGGER ime vrijeme događaj
     ON tablica
     FOR EACH ROW izjava
```

Klauzula `FOR EACH ROW` je obavezna po MySQL sintaksi. Označava da će okidač biti izvršen za svaki redak na kojeg se odnosi DML naredba koja je aktivirala okidač. Obzirom da će okidač u većini slučajeva izvršavati više SQL naredbi, one će nakon `FOR EACH ROW` izjave biti definirane unutar `BEGIN … END` bloka.

Okidač je vezan isključivo uz pravu tablicu, i ne može biti vezan uz privremenu tablicu. Svaki okidač je objekt u jednoj bazi podataka i vrijedi samo za tu bazu. Prilikom brisanja baze podataka, bit će obrisani i svi njeni okidači. Da bi korisnik u bazi mogao izraditi okidač, potrebno je da su mu dodjeljene odgovarajuće privilegije (Duk, 2012).

Osnovna sintaksa za brisanje okidača:

```
DROP TRIGGER imeOkidaca
```

16.1. Razlozi za korištenje okidača

Razlozi za korištenje okidača:

- Okidači omogućuju prilagođavanje sustava za upravljanje bazama podataka (SUBP). Automatski generiraju virtualne vrijednosti stupaca
- Log-ovi događaja
- Skupljaju statistike nad pristupom tablicama
- Modificiraju tablice kada su DML naredbe izvršene nad pogledima
- Provode referencijalni integritet kada su dijete I roditelj tablice na različitim čvorovima distribuirane baze podataka
- Objavljuju informacije o događajima u bazi, o događajima korisnika i o SQL izrazima
- Spriječavaju DML operacije nad tablicama nakon radnog vremena
- Spriječavaju nevažeće transakcije
- Provode složena poslovna pravila I pravila referencijalnog integriteta koja se ne mogu definirati s ograničenjima
- Prate rad korisnika

Iako su okidači korisni za prilagodbu baze podataka, treba ih koristiti samo kada je to potrebno. Pretjerana upotreba okidača može rezultirati složenim međusobnim ovisnostima, što može biti teško održavati u velikoj aplikaciji. Na primjer, kada se okidač okine, SQL naredba unutar okidača može potencijalno okinuti druge okidače što može dovesti do neželjenih učinaka.

16.2. Razlika između ograničenja i okidača

I okidači i ograničenja mogu ograničiti unos podataka, ali se znatno razlikuju. Okidač se uvijek odnosi samo na nove podatke. Na primjer, okidač može spriječiti DML naredbu od umetanja NULL vrijednosti u stupac, ali stupac može sadržavati NULL vrijednosti koje su umetnute prije nego što je okidač definiran ili dok je okidač bio onemogućen (DISABLED).

Ograničenja se mogu odnositi na samo na nove podatke (kao okidač) ili na nove i postojeće podatke. Ponašanje ograničenja ovisi o stanju ograničenja. Ograničenja su lakša za pisanje i manje su sklona pogreškama nego okidači koji provode ista pravila. Međutim, okidači mogu provoditi neka složena poslovna pravila koja ograničenja ne mogu.

16.3. Ograničenja okidača

Okidač ne smije koristiti naredbe koje eksplicitno označavaju početak i kraj transakcije. Potrebno je izbjeći primjerice izazivanje neželjenog završetka transakcije u trenutku kada je transakcija izvršila određenu DML naredbu koja je aktivirala okidač (Anon., n.d.).

Ako postoji pogreška u okidaču, aktivirajuća operacija neće se izvesti. U slučaju da na tablici postoje i `BEFORE` i `AFTER` okidač te ako dođe do pogreške prilikom izvođenja `BEFORE` okidača, tada se niti `AFTER` okidač neće izvesti.

MySQL do sada nije razvio mogućnost izazivanja pogreške (eng. *raise exception*) kako bi obavijestio aplikaciju pod kojom radi da nešto nije u redu. Ako je potrebno direktno prekinuti izvođenje određene radnje, rješenje je prouzrokovati pogrešku što će rezultirati prekidom rada okidača te prekidom aktivirajuće operacije. To se može napraviti na nekoliko načina (Duk, 2012):

- Unosom podatka u nepostojeću kolonu
- Unosom NULL vrijednosti u kolonu za koju je definirano da ne smije poprimiti NULL vrijednost
- Pozivom nepostojeće pohranjene procedure

```
DELIMITER $$
CREATE TRIGGER trigger_name BEFORE DELETE
     ON student
     FOR EACH ROW
           BEGIN
              CALL nepostojeca_procedura();
           END;
$$
DELIMITER ;
```

Primjer naredbe koja će aktivirati okidač:

```
DELETE FROM odjel WHERE sifStudent=2;
```

16.4. Okidači i procedure

Okidači su posebna vrsta pohranjene procedure. Posebni su iz razloga što se ne pozivaju direktno kao pohranjena procedura. Glavna razlika između okidača i pohranjene procedure je ta da se okidač poziva automatski kad se dogodi neka promjena nad podacima u tablici, dok se pohranjena procedura mora pozvati eksplicitno. Jedno od obilježja okidača je da osim izvođenja seta naredbi može pozvati i proceduru (Duk, 2012).

Primjer:

Potrebno je napraviti okidač koji će se pobrinuti za osiguravanje referencijalnog integriteta vezanog uz tablicu mjesto. Prilikom brisanja zapisa u tablici mjesto, potrebno je provjeriti da li je to mjesto referencirano u tablici student ili fakultet. Ako je referencirano, potrebno je postaviti vrijednosti odgovarajućih atributa u tablicama student i fakultet na NULL.

```
DROP TRIGGER mjesto;

DELIMITER $$
CREATE TRIGGER mjesto AFTER DELETE
     ON mjesto
     FOR EACH ROW
       BEGIN
         UPDATE student SET pbrStudent=NULL
                 WHERE pbrStudent=OLD.pbrMjesto;
         UPDATE fakultet SET pbrFakultet=NULL
                 WHERE pbrFakultet=OLD.pbrMjesto;
       END;
$$
DELIMITER ;
```

Primjer naredbe koja će aktivirati okidač:

```
DELETE FROM mjesto WHERE pbrMjesto=10000;
```

Kažemo da su okidači "čuvari" referencijalnog integriteta baze podataka. Isti je okidač moguće realizirati i pozivom procedure. U tom se slučaju procedura brine za osiguravanje integriteta, a okidač će pozvati proceduru i predati joj odgovarajući argument.

```
DROP PROCEDURE IF EXISTS radi_mjesto;

DELIMITER $$
CREATE PROCEDURE radi_mjesto(IN pbr INT)
   BEGIN
     UPDATE klijent SET pbrreg=NULL
        WHERE pbrreg=pbr;
     UPDATE klijent SET pbrKlijent=NULL
        WHERE pbrKlijent=pbr;
    END;
$$
DELIMITER ;
```

```
DROP TRIGGER mjesto;

DELIMITER $$
CREATE TRIGGER mjesto BEFORE DELETE
ON mjesto
FOR EACH ROW
     BEGIN
         CALL radi_mjesto(OLD.pbrMjesto);
     END;
$$
DELIMITER ;
```

Primjer naredbe koja će aktivirati okidač:

```
DELETE FROM mjesto WHERE pbrMjesto=10000;
```

16.5. DML okidač

DML okidač se može stvoriti ili nad tablicom ili nad pogledom, a aktivirajuća naredba je sastavljena od DML naredbi `INSERT`, `UPDATE` i `DELETE`. DML okidač može biti ili jednostavan ili složen. Podjela DML okidača:

1. DML okidači se dijele na one koji se pokreću na razini:

Naredbe (eng. *statement triggers*)

Okidač na razini naredbe se aktivira jednom u ime aktivirajuće naredbe, bez obzira na broj redova u tablici na koje aktivirajuča naredba utječe, čak i kad niti jedan red nije pogođen tom naredbom. Na primjer, ako `DELETE` naredba briše nekoliko redova iz tablice, okidač na razini naredbe će se okinuti samo jednom.

Reda (eng. *row triggers*)

Retčani okidač se aktivira svaki put kada aktivirajuća naredba utječe na tablicu. Na primjer, ako UPDATE naredba ažurira više redaka tablice, retčani okidač će se okinuti jednom za svaki red koji je pogođen s tom `UPDATE` naredbom. Ako aktivirajuća naredba ne utječe niti na jedan red, retčani okidač se neće pokrenuti.

2. Također se dijele i prema vremenu okidanja:

BEFORE okidači

Događaju se prije nego podatak dođe do tablice. SUBP u trenutku unosa kreira dodatnu tablicu u memoriji – NEW. Struktura te tablice identična je strukturi originalne tablice. Ona sadrži samo podatke koje unosimo ili mijenjamo. Podaci se prije unosa u originalnu tablicu unose u privremenu tablicu NEW, te se unutar nje može korisnički manipulirati podacima na potreban način. Po završetku okidača, SUBP će podatke iz tablice NEW automatski prenijeti u originalnu tablicu.

Primjer:

```
CREATE TRIGGER provjeraImena BEFORE INSERT
     ON klijent
     FOR               EACH               ROW               SET
new.imeklijent=CONCAT(new.imeKlijent,'test');
```

AFTER okidači

Aktiviraju se nakon unosa podataka u tablice. Kod AFTER okidača nije moguće mijenjati vrijednosti koje će biti unesene ili ažurirane u tablici na koju se okidač odnosi. U tom je slučaju kasno pokušavati mijenjati radnju aktivirajuće operacije. Podaci NEW.imeKolone biti će uneseni bazu nepromijenjeni, a pokušaj njihovog mijenjanja rezultirat će pogreškom.

Iako se većina potrebnih funkcionalnosti može riješiti sa BEFORE okidačima, AFTER okidače će je korisno upotrijebiti za radnje koje se logički trebaju izvršiti tek po uspješnom završetku aktivirajuće operacije. Primjerice, revizijske aktivnosti najbolje je izvršiti tek po uspješnom završetku DML naredbe (aktivirajuće operacije). Prilikom izrade sigurnosne kopije (backup) također se najčešće koriste AFTER okidači (Duk, 2012).

Primjer:

Za svaki unos novog imena u tablicu, unijeti isti podatak i u backup tablicu naziva backupTablica.

```
DROP TRIGGER provjeraImena;

DELIMITER $$
CREATE TRIGGER provjeraImena AFTER INSERT
     ON mojaTablica
     FOR EACH ROW
           BEGIN
                    INSERT INTO backupTablica VALUES (NEW.ime);
           END;
$$
DELIMITER ;
```

BEFORE i AFTER okidači korišteni u DML naredbama mogu biti definirani samo nad tablicama, ne i nad pogledima.

BEFORE i AFTER okidači korišteni u DDL naredbama mogu biti definirani samo nad bazom podataka ili nad schemom, ne i nad tablicama.

Ne mogu postojati dva ista okidača za istu tablicu, npr:

- BEFORE INSERT i BEFORE INSERT- ne može
- BEFORE INSERT i AFTER INSERT - može
- BEFORE INSERT i BEFORE UPDATE - može

Za stvaranje (ili zamjenu) DML okidača koristi se sljedeća sintaksa:

```
CREATE [OR REPLACE] TRIGGER trigger_name
{BEFORE | AFTER}
{INSERT | DELETE | UPDATE | UPDATE OF column_list } ON
table_name
[FOR EACH ROW]
[WHEN (...)]
[DECLARE ... ]
BEGIN
...executable statements...
[EXCEPTION ... ]
END [trigger_name];
```

Jednostavan DML okidač se aktivira točno nad jednoj od ovih vremenskih točaka:

- Prije nego što se obavi operacija koja je aktivirala okidač - **BEFORE**
- Nakon što se obavi operacija koja je aktivirala okidač - **AFTER**
- Prije/poslije svakog reda (n-torke) na kojeg je okidač utječe - **FOR EACH ROW**

Složeni DML okidač stvoren nad tablicom ili nad pogledom može biti aktiviran na jednoj, nanekim ili na svim vremenskim točkama.

Jednostavni ili složeni DML okidač koji se aktivira na razini reda, može pristupiti podacima u redu kojeg obrađuje.

16.5.1. Privremene tablice old i new

U tijelu okidača moguće je referencirati se na stare vrijednosti koje će biti promijenjene ili obrisane odnosno na nove koje će biti unesene u tablicu. Navedeno je moguće ključnim riječima NEW odnosno OLD nakon kojih se navodi naziv kolone na koju će se podatak odnositi. NEW.imeKolone odnosi se na podatak koji će tek biti unesen u bazu, odnosno OLD.imeKolone odnosi se na postojeći redak u tablici koji će biti izbrisan ili promijenjen. Navedene oznake postoje i kod BEFORE i kod AFTER okidača. Ključne riječi OLD i NEW nisu osjetljive na mala i velika slova.

Aktivirajuća naredba	OLD.field vrijednost	NEW.field vrijednost
INSERT	NULL	Post-insert value
UPDATE	Pre-update value	Post-update value
DELETE	Pre-delete value	NULL

Tablica 1: Razlike između old i new tablica

NEW.imeKolone

- Sadrži podatke koji će biti uneseni u tablicu prilikom izvođenja naredbe `INSERT` ili `UPDATE`
- Moguće je korisnički manipulirati s ovim podacima prije nego se unesu u originalnu tablicu uz koju je okidač vezan -> sintaksa: SET NEW.imeKolone=novaVrijednost
- Navedena mogućnost promjene vrijednosti u retku NEW moguća je isključivo kod BEFORE okidača
- Po završetku rada okidača, podaci koji imaju prefiks NEW, automatski će biti uneseni u originalnu tablicu

OLD.imeKolone

- Sadrži podatke koji će biti izbrisani u tablici izvođenjem naredbe `DELETE` ili promijenjeni izvođenjem naredbe `UPDATE`
- Ove je podatke moguće isključivo čitati, ali ne i mijenjati

16.6. DDL okidači

DDL okidači se stvaraju ili nad schemom ili nad bazom podataka. To je posebna vrsta okidača koji se aktiviraju kao odgovor na DDL (Data Definition Language) naredbe. Mogu se koristiti za obavljanje administrativnih zadataka kao što su revizija i reguliranje poslovanja baza podataka (Oracle, 2014).

DDL okidači, baš kao i DML okidači, aktiviraju pohranjene procedure kao odgovor na određeni događaj, ali reagiraju samo na na DDL događaje. To su naredbe koje počinju sa ključnim riječima `CREATE`, `ALTER` i `DROP`. Određene sistemske pohranjene procedure koje obavljaju operacije slične DDL-ovim, također mogu aktivirati DDL okidače.

DDL okidač se aktivira točno na jednoj od ovih vremenskih točaka (Oracle, 2014):

- Prije nego što se aktivirajuća naredba pokrene
- Nakon što se aktivirajuća naredba pokrene
- Umjesto okidanja `CREATE` naredbe

DDL okidače treba koristiti kada:

- želimo spriječiti određene promjene nad schemom baze podata
- želimo da se dogodi neka promjena u bazi kao odgovor na promjenu u schemi bazi podataka
- želimo snimiti promjene ili događaje nad schemom baze podataka

Za stvaranje (ili zamjenu) DDL okidača koristi se sljedeća sintaksa:

```
CREATE [OR REPLACE] TRIGGER trigger name
{BEFORE | AFTER } { DDL event} ON {DATABASE | SCHEMA}
[WHEN (...)]
DECLARE
Variable declarations
BEGIN
...some code...
END;
```

Primjer DDL okidača koji se može koristiti kako bi se spriječilo da se tablice u bazi modificiraju ili obrišu.

```
CREATE TRIGGER safety
ON DATABASE
FOR DROP_TABLE, ALTER_TABLE
AS
PRINT 'You must disable Trigger "safety" to drop or alter
tables!'
ROLLBACK ;
```

DDL i DML okidači se koriste u različite svrhe. DML okidači rade nad `INSERT`, `UPDATE` i `DELETE` naredbama kako bi pomogli i osigurali provođenje poslovnih pravila i integriteta. DDL okidači rade nad `CREATE`, `ALTER` i `DROP` naredbama i koriste se za obavljanje administrativnih poslova i za provedbu poslovnih pravila koji utječu na bazu podataka.

U sljedećoj tablici su izlistani samo neki DDL događaji koji se mogu koristiti za aktiviranje DDL okidača:

CREATE_FUNCTION	ALTER_FUNCTION	DROP_FUNCTION
CREATE_INDEX	ALTER_INDEX (Applies to the ALTER INDEX statement andsp_indexoption.)	DROP_INDEX
CREATE_MASTER_KEY	ALTER_MASTER_KEY	DROP_MASTER_KEY
CREATE_PROCEDURE	ALTER_PROCEDURE (Applies to the ALTER PROCEDURE statement andsp_procoption.)	DROP_PROCEDURE
CREATE_REMOTE_SERVICE_BINDING	ALTER_REMOTE_SERVICE_BINDING	DROP_REMOTE_SERVICE_BINDING
CREATE_ROLE (Applies to the CREATE ROLE statement, sp_addrole, and sp_addgroup.)	ALTER_ROLE	DROP_ROLE (Applies to the DROP ROLE statement, sp_droprole, andsp_dropgroup.)
ADD_ROLE_MEMBER	DROP_ROLE_MEMBER	
CREATE_TABLE	ALTER_TABLE (Applies to the ALTER TABLE statement andsp_tableoption.)	DROP_TABLE

CREATE_TRIGGER	ALTER_TRIGGER (Applies to the ALTER TRIGGER statement andsp_settriggerorder.)	DROP_TRIGGER
CREATE_USER (Applies to the CREATE USER statement, sp_adduser, and sp_grantdbaccess.)	ALTER_USER (Applies to ALTER USER statement andsp_change_users_login.	DROP_USER (Applies to the DROP USER statement, sp_dropuser, andsp_revokedbaccess.)

Tablica 2: DDL događaji za aktiviranje DDL okidača (Izvor: **https://docs.microsoft.com/en-us/sql/relational-databases/triggers/ddl-events**)

16.7. Okidači događaja baze podataka

Aktivirajuća operacija je jedan od 6 događaja baze podataka: `STARTUP`, `SHUTDOWN`, `SERVERERROR`, `LOGON`, `LOGOFF`, `DB_ROLE_CHANGE`. Svaki administrator baze podataka će odmah vidjeti kako ovi okidači nude odlične mogućnosti za automatsku administraciju i vrlo zrnatu kontrolu.

Postoje ograničenja u odnosu na to koji događaj se može kombinirati sa `BEFORE` ili `AFTER` atributima iz razloga što neke situacije jednostavno nemaju smisla. Na primjer ne može se napisati `BEFORE STARTUP` okidač ili `AFTER SHUTDOWN`.

17. MySQL

Kao što je već spomenuto, za potrebe izrade projektnog djela ovog diplomskog rada koristiti će se MySQL sustav za upravljanje bazom podataka.

MySQL je najpopularniji sustav otvorenog koda (open source) za upravljanje bazom podataka. Sa svojim dokazanim performansama, pouzdanošću i jednostavnošću korištenja postao je najkorišteniji sustav za upravljanje bazama podataka za web bazirane aplikacije. Predstavlja jako moćan sustav koji pruža puno fleksibilnosti.

Značajke MySQL-a (Oracle, 2017)**:**

- **Skalabilnost i fleksibilnost**

MySQL pruža maksimalnu skalabilnost, podržavajući kapacitete za rukovanje aplikacijama koje koriste od samo 1MB podataka pa do aplikacija sa masivnim skladištem podataka koje može sadržavati i terabajte podataka. MySQL sustav je otvorenog koda što omogućava veliku fleksibilnost samog sustava. Kao posljedica toga dopuštena je potpuna prilagodba sustava korisnicima koji imaju trebaju dodati jedinstvene zahtjeve sustavu.

- **Visoke performanse**

Jedinstvena arhitektura mehanizma za pohranu omogućuje konfiguriranje MySQL sustava za aplikacije različitih namjena, što pruža odlične rezultate u samim performansama. Bilo da je aplikacija brzi transakcijski sustav za obradu podataka ili web stranica koja pruža milijardu upita dnevno, MySQL može zadovoljiti i najzahtjevnija očekivanja uspješnosti bilo kojeg sustava.

- **Robusna transakcijska podrška**

MySQL pruža jedan od najmoćnijih transakcijskih mehanizma baza podataka. Značajke uključuju kompletnu ACID transakcijsku podršku, neograničenu razinu zaključavanja i više verzijsku transakcijsku podršku gdje čitači transkacija nikad ne blokiraju pisce i obrnuto. Integritet podataka je također osiguran putem referencijalnog integriteta, transakcijskih razina izolacije i trenutno otkrivanje potpunog zastoja (deadlock).

- **Prednosti web-a i skladišta podataka**

MySQL je standard za prometno velike web stranice zbog svojeg upitnog mehanizma visokih performansa, mogućnosti strahovito brzog ažuriranja podataka i snažne podrške funkcijama

za brzo pretraživanje cijelovitog teksta. Iste te značajke se pirmjenjuju i na skladište podataka. Ostale značajke poput glavnih memorijskih tablica, B-stabla i hash indeksa, te komprimiranih arhivskih tablica koje smanjuju potrebnu veličinu skladištenja i do 80% čine MySQL moćnim sustavom.

- **Zaštita podataka**

MySQL pruža iznimne sigurnosne značajke koje osiguravaju potpunu zaštitu podataka. U pogledu autetičnosti baze podataka, MySQL pruža snažne mehanizme za osiguravanje pristupa serveru baze podataka samo ovlaštenim korisnicima. SSH i SSL podrška osigurava sigurno povezivanje. Prisutna je i zrnatost događaja tako da korisnici vide samo sadržaj koji im je dopušten. Snažne enkripcijske i dekripcijske funkcije osiguravaju da su osjetljivi podaci zaštićeni od neovlaštenog pregledavanja. Konačno, omogućuje stvaranje potpune logičke i fizičke kopije sustava kao i potpuni oporavak do određene točke u vremenu.

- **Sveobuhvatni razvoj aplikacija**

MySQL pruža sveobuhvatnu podršku za sve potrebe razvoja aplikacija. Podržava pohranjene procedure, okidače, funkcije, poglede, kursore, SQL, i još mnogo toga.

- **Jednostavno korištenje**

MySQL nudi iznimno brzu i laku instalaciju, neovisno o kojoj platformi se radi. Jednom kad je instaliran omogućuje značajke kao što su automatsko širenje prostora, automatsko ponovo pokretanje, dinamične promjene konfiguracije itd. MySQL pruža i kompletan paket alata za migracije i grafičko upravljanje.

- **Ukupni trošak korištenja**

Prmještanjem trenutne baze aplikacije sa nekog drugog sustava na MySQL, ili korištenje MySQL za nove razvojne projekte, korporacije često mogu smanjiti ukupan trošak za velike iznose. Za cijenu koja je daleko manja od drugih sustava korporacije mogu pomoću MySQL-a postići zadivljujući stupanj skalabilnosti i performansi.

Grafičko sučelje koje će se u ovom radu koristiti za MySQL je SQLyog, i to ponajviše zbog njegove jednostavnosti korištenja.

18. Upravljanje skladištem

Skladište je određeni prostor ili skup prostorija namijenjen za privremeno čuvanje i smještaj robe koja je predmet poslovanja preduzeća. Predstavlja točku gdje tvrtka skladišti sirovine, poluproizvode i gotove proizvode kako bi mogli zadovoljiti potrebe kupaca u svakom trenutku. Ubrzanim razvojem tržišta doveo je i do različitih shvaćanja uloge skladištenja, kao posljedica toga uloga skladištenja je potpuno promijenjena. Skladištenje preuzima sve veću ulogu u cjelokupnom opskrbnom lancu, te se osim osnovnih skladišnih operacija poput prijema, uskladištenja, otpreme obavlja i niz drugih usluga (Goluža, 2016).
U suvremenom načinu upravljanja poslovnim procesima skladište predstavlja točku u logističkoj mreži na kojoj se predmet skladištenja prihvaća ili prosljeđuje u nekom drugom smjeru unutar mreže (Goluža, 2016).

U užem smislu se pod skladištem podrazumijeva mjesto smještaja, čuvanja i izdavanja robe. U širem smislu to je ograđeni ili neograđeni prostor, zatvoreni ili poluzatvoreni (pokriveni) prostor, za uskladištenje robe i svega onog što je u izravnoj vezi sa skladištenjem, te kao takav predstavlja njegov sastavni dio. S toga gledišta, skladište predstavlja prostor u kojem se roba preuzima, čuva od raznih krađa, fizičkih,kemijskih i atmosferskih utjecaja, izdaje i otprema (Goluža, 2016).

Skladištenje je planirana aktivnost kojom se materijal dovodi u stanje mirovanja, a uključuje fizički proces rukovanja i čuvanja materijala te metodologiju za provedbu tih procesa. U industrijskom poduzeću, skladište je uređeno i opremljeno mjesto za privremeno i sigurno odlaganje, čuvanje, pripremu i izdavanje materijala prije, tijekom i poslije njihova trošenja i uporabe u procesu proizvodnje. Ono je usko povezano sa nabavom i distribucijom robe (Goluža, 2016).

Sadašnjost i budućnost uspješnog poslovanja jest što brza isporuka proizvoda odnosno robe prema korisniku. Kako bi ta isporuka bila dovoljno brza i sa minimalnim brojem grešaka, potrebno je adekvatno organizirati skladište i prepoznati ga kao kritični proces u poslovanju. Postoje razna rješenja za upravljanje skladištem i svakako je dobro prepoznati kada je vrijeme za nadogradnju, te koji su sljedeći koraci kako ne bi došlo do problema kod isporuka, zaprimanja, ili samih zaliha (Smolko, 2015)

18.1. Skladišne operacije

Skladišne operacije imaju za zadatak odrediti tok materijala u skladištu. Prva skladišna operacija je prijem i započinje najavom i fizičkim prijemom robe. Najava dostave robe omogućuje usklađivanje prijema i otpreme te učinkovito koordiniranje s drugim aktivnostima unutar skladišta. Zatim slijedi pohrana koja podrazumijeva fizičko premještanje robe iz prijemne zone do skladišne lokacije unutar skladišta. Ovaj proces uključuje identifikaciju proizvoda, skeniranje barkoda proizvoda, pronalazak lokacije unutar skladišta, i premještanje proizvoda na određenu lokaciju. Skladišna operacija pohrana robe je jedna od najvažnijih skladišnih aktivnosti jer ukoliko nije obavljena kako treba sve skladišne aktivnosti poslije nje će duže trajati. Komisioniranje predstavlja središnji dio protoka robe od dobavljača do kupca. To je operacija tijekom koje se prema zahtjevima korisnika prikuplja roba u skladištu i formira pošiljka spremna za otpremu (Goluža, 2016).

18.2. Sustav za upravljanje skladištem

Skladište, koje ima za cilj ispuniti zahtjeve visokog menadžmenta, a da se pritom zadovolje zahtjevi korisnika, treba koristiti alate i tehnologije koje omogućuju olakšanu kontrolu i rukovanje skladišnim aktivnostima. Takvu vrstu tehnologije omogućava računalni sustav upravljanja skladištem, odnosno WMS (eng. *Warehouse Menagement System*) sustav (Bušić, 2015).

WMS sustav je računalni sustav upravljanja skladištem koji za cilj ima kontrolu kretanja i skladištenje materijala unutar skladišta. Sustav obrađuje pripadajuće transakcije, uskladištenje, popunjavanje, komisioniranje te optimizira stanje i količinu zaliha koje temelji na informacijama dobivenim u stvarnom vremenu. WMS prati napredak proizvoda kroz skladište. To uključuje fizičku infrastrukturu skladišta, sustave praćenja i komunikaciju između postaja. Jednostavnije rečeno, računalno upravljanje skladištem uključuje primitak, skladištenje i kretanje robe prema skladišnim mjestima ili prema krajnjem kupcu (Rouse, 2009).

Takav sustav korisnicima omogućuje centralizirani sustav u kojem različitim skladišnim zadacima se upravlja putem sučelja na ručnom uređaju ili nekom drugom računalu. Zbog toga je skladište djelotvorno i jednostavno, a također osigurava minimalni gubitak u različitim procesima skladišta. Stvarni dobitak je u službi za korisnike. Osigurava informacije o tome gdje je svaki proizvod, znajući kada treba ponovo naručiti, i koliko treba ponovo naručiti ili proizvesti. Ove stvari izgledaju kao poslovni cilj, ali za kupca to znači da

mogu dobiti proizvod brže, bez povratnih pogrešaka ili pogrešaka pa će se vjerojatnije vratiti (Bušić, 2015).

Raniji sustavi za upravljanje skladištem mogu su pružiti jednostavne funkcije, uglavnom samo informacije o lokaciji pohrane. Danas, širina WMS funkcionalnosti može se uvelike razlikovati, od osnovnih najboljih praksi u odabiru, pakiranju i dostavi robe do sofisticiranih programa koji koordiniraju napredne interakcije s uređajima za rukovanje materijalom i upravljanjem skladištem. Bez obzira koliko je jednostavna ili složena aplikacija, cilj sustava upravljanja skladištima ostaje isti - pružiti menadžmentu informacije potrebne za učinkovito upravljanje kretanjem materijala unutar skladišta (IQMS, 2016).

Takvi sustavi, osim softverskog dijela skladišnog poslovanja, koriste i strogo namjensku opremu (hardware), koja se koristi u skladišnom poslovanju. To najčešće uključuje skenere, čitače bar kodova, wireless mrežnu infrastrukturu i opremu te prijenosna računala (Rouse, 2009).

Implementacija WMS-a ima za cilj ubrzati procese rada u skladištu, detektirati i otkloniti kritične točke skladišnog poslovanja, povećati točnost zaprimanja, komisioniranja i izdavanja robe te smanjenje potrebne dokumentacije. Tako je danas moguće cijeli operativni posao u skladištu odrađivati bez papira, odnosno nije potreban niti jedan papirnati dokument kako bi se roba uskladištila, premjestila sa jedne na drugu lokaciju unutar skladišta, komisionirala, pripremila za isporuku i isporučila (Murray, 2016).

Prilikom organiziranja skladišnog poslovanja, kao i u toku samog procesa rada u skladištu pojavljuju se neki tipični problemi. To su najčešće nedovoljna iskorištenost skladišnog prostora i povezano s time, nedostatak skladišnog prostora, velika mogućnost pogreške djelatnika koji rade u skladištu (zamjena sličnih artikala jedan za drugi), prevelika potrošnja vremena na traženje određenog artikla u skladištu, relativno spor protok robe i "čepovi" na ulazu ili izlazu iz skladišta, nedostatak informacija o količinama, vrsti i vremenu dolaska/odlaska neke robe iz skladišta i neefikasno korištenje radne snage. Sustav upravljanja skladištem može uvelike smanjiti vjerojatnost tih pogrešaka (Rouse, 2009).

18.3. Vrste sustava za upravljanje skladištem

Postoji nekoliko vrsta sustava upravljanja skladištem, svaki sa svojim vlastitim prednostima i manama. Najpopularnije vrste su:

- Samostalni sustav
- Baziran na oblaku (eng. Cloud based)
- ERP moduli

Samostalni sustav za upravljanje skladištem je tipični sustav implementiran na izvornom hardveru i mreži poslovanja. Većina WMS sustava su treće strane (eng. third part), samostalni paketi koji moraju biti integrirani s ostatkom postojećeg softvera za upravljanje poslovanjem (kao što je ERP). Iako integracija vanjskih programa može funkcionirati, proces je često ispunjen izazovima kao što su dupli unos podataka, kašnjenja informacija, problemi s sučeljem i troškovi prilagodbe. Ti sustavi su često najjeftinija opcija dugoročnog troška, ali nedostaju neke od prednosti koje imaju integrirane WMS opcije (IQMS, 2016).

Sustav upravljanja bazom podataka baziran na cloudu donosi nekoliko prednosti kao što su bolja fleksibilnost, oporavak od katastrofe, skalabilnost i sigurnost. Također nudi korisnicima mogućnost primanja automatskih ažuriranja softvera bez dodatnih kapitalnih izdataka, osiguravajući bolju konkurentnost tehnologije (IQMS, 2016).

ERP modul je sustav upravljanja skladištem koji je ugrađen u ERP[11] (eng. *Enterprise resource planning*) rješenje. Nativan WMS program omogućuje praćenje robe na razini poslovanja, odgovornost zaposlenika i brzinu u stvarnom vremenu (IQMS, 2016).

Postoje još mnogo razlika koje u potpunosti definiraju model ERP hostinga i treba ih se uzeti u obzir prije nego što odlučite koja je implementacija najbolja za vaše poslovanje.

[11] ERP – prema izv.prof.dr.sc. Ruben Picek „ERP sustav predstavlja poslovni informacijski sustav koji prati sve aspekte poslovanja jedne organizacije integrirajući podatke i procese u jedinstvenu cjelinu.“

18.4. Odabir odgovarajućeg WMS-a

Pri odabiru WMS-a, postoje razni čimbenici koje je potrebno razmotriti prije implementacije određenog WMS. Neke od stvari koje treba razmotriti uključuju (QSTOCK, n.d.):

- **Funkcionalnosti** - različiti WMS mogu obavljati različite funkcije i izgrađene su za različite industrije. Treba pronaći sustav upravljanja skladištem koji je podesiv i može se prilagoditi trenutnom stanju poslovanja, a i onom budućem.
- **Veličina skladišta** - veća skladišta zahtijevaju detaljnije sustave od manjih. To je zato što se više aktivnosti i funkcija provodi u velikim skladištima, a sve takve aktivnosti i funkcije zahtijevaju detaljniji sustav. Što je veće skladište, to je veći trošak putovanja između lokacija, a time i važnije detaljno praćenje.
- **Potrebe kupca** - Identificirajući nedostatke u trenutnom sustavu moguće je odrediti potrebne funkcionalnosti sustava za upravljanje skladištem koje će osigurati povećanje zadovoljstva klijenata.
- **Trošak** - trošak instaliranja WMS sustava uvelike se razlikuje ovisno o složenosti sustava i dobavljaču sustava. Potrebno je odabrati WMS sustav koji će omogućiti ispunjavanje svih potrebnih funkcionalnosti, kao i onih koje tvrtka može priuštiti. Odabir sustava koji je cijenom previše skup za određeno poduzeće može ugroziti kvalitetu i učinkovitost koja se upravo njime želi poboljšasti. S druge strane, odabir osnovnog WMS-a možda neće poslužiti dovoljno, posebno ako postoji mnogo funkcija u skladištu. Potrebno je pronaći pravu ravnotežu između troškova i funkcija prilikom odabira pravog WMS-a za svoju tvrtku.

19. Praktični rad

Do sada opisani koncepti, implementirani su unutar aplikacije za upravljanje skladištem. Izrađena aplikacija nije namjenjena za korištenje u relanom sustavu, već je prilagođena ovom diplomskom radu. U ovom slučaju naglasak ja bio na aktivnoj bazi podataka, tako da su ostale funkcionalnosti, poput cijelog poslovnog procesa i dizajna same aplikacije pojednostavljeni i prilagođeni glavnoj svrsi aplikacije.

Kao što i sam nalov diplomskog rada kaže, riječ je o aplikaciji za upravljanje skadištem temeljena na aktivnim bazama podataka. Nije bitno za samu aplikaciju, ali u ovom slučaju kao primjer ćemo koristiti skladište informatičke opreme. Kao što smo prethodno već spomenuli poslovni proces je pojednostavljen i prilagođen. Glavne komponente su narudžbenice i otpremnice. Prvo je potrebno kreirati dobavljače i kategorije, a zatim kreirati i prve proizvode. Nakon kreiranih proizvoda možemo kreirati narudžbenicu sa tim istim proizvodima. Kreirana narudžbenica sve naručene proizvode odmah šalje na skladište, te nakon toga možemo kreirati otpremnicu sa proizvodima koji se nalaze na skladištu. Kreirana otpremnica otprema te proizvode sa skladišta. Svaka narudžbenica i otpremnica imaju svoju detaljnu listu na kojoj se nalazi popis odabranih proizvoda. Još dvije komponenta aplikacije su arhiva u koju se sprema sva uspješno izvršena narudžbenica i otpremnica, te sigurnosna kopija skladišta u kojoj se bilježi trenutno stanje skladišta.

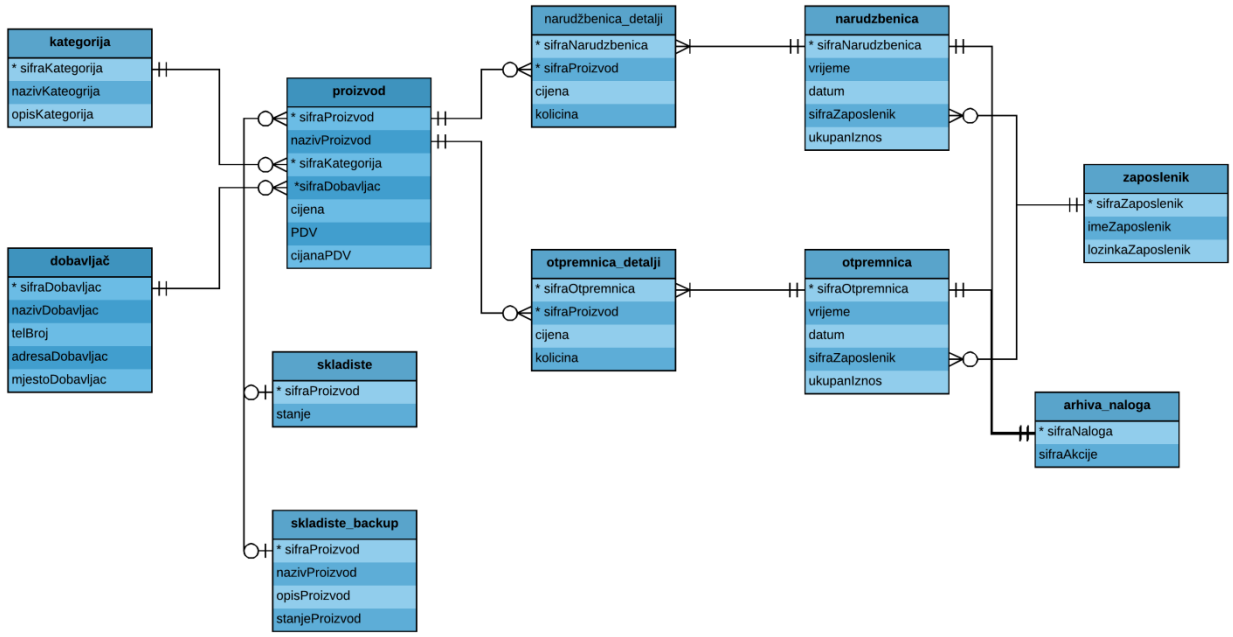
Nakon kratkog uvoda u aplikaciju, trebamo spomenuti i tehnologije koje su korištene prilikom izrade aplikacije. Na strani klijenta (eng. *front-end*) korištene su tehnologije HTML, CSS, Javascript i jQuery[12]. Na strani poslužitelja (eng. *back-end*) korišten je php. Za bazu podataka korišten je MySQL sustav za upravljanje bazama podataka. Što se tiče samog kôda, zbog velikog obujma u nastavku će biti prikazane samo naredbe vezane uz okidače. Cjelokupan kôd se može provjeriti u aplikacijskim datotekama.

[12] jQuery je JavaScript bibilioteka

19.1. Model baze podataka

Kao što smo ranije spomenuli, arhitektura baze podataka se sastoji od 3 razine apstrakcije. U nastavu ćemo bazu podataka promatrati na globalnoj logičkoj razini. Opis te razine naziva se shema baze podataka. Shema baze podataka je formalni opis svih podataka baze podataka i svih odnosa koji postoje između njih. Isto tako već ranije spomenuto, postoji različiti načini organiziranja sheme. Oni se nazivaju modeli baze podataka. Podaci u bazi moraju biti logički organizirani u skladu s modelom koji podržava odabrani sustav za upravljanje bazom podataka. U ovoj aplikciji koristimo relacijski model, što znači da je riječ o relacijskoj bazi podataka. On koristi zbirku tablica za opisivanje podataka i odnosa između njih.

U nastavku je prikazan ER dijagram baze podataka. To je detaljan logički prikaz podataka preko skupa entiteta, njihovih atributa i međusobnih veza. Za prikaz korištena je notacija pod nazivom *Crow's Foot*. Notacija prikazuje enititete u obliku pravokutnika, a odnose linijama između tih pravokutnika. Različiti oblici koji se nalazi na početku i kraju svake linije predstavljaju kardinalnost odnosa.



Slika 11: ER dijagram baze podataka

19.2. Aplikacija

Prije nego što možemo pristupiti aplikaciju, moramo se prijaviti za odgovarajućim podacima. Korisničko ime za prijavu je "tomislav", dok je lozinka "diplomski-rad".

Slika 12: Forma za prijavu korisnika

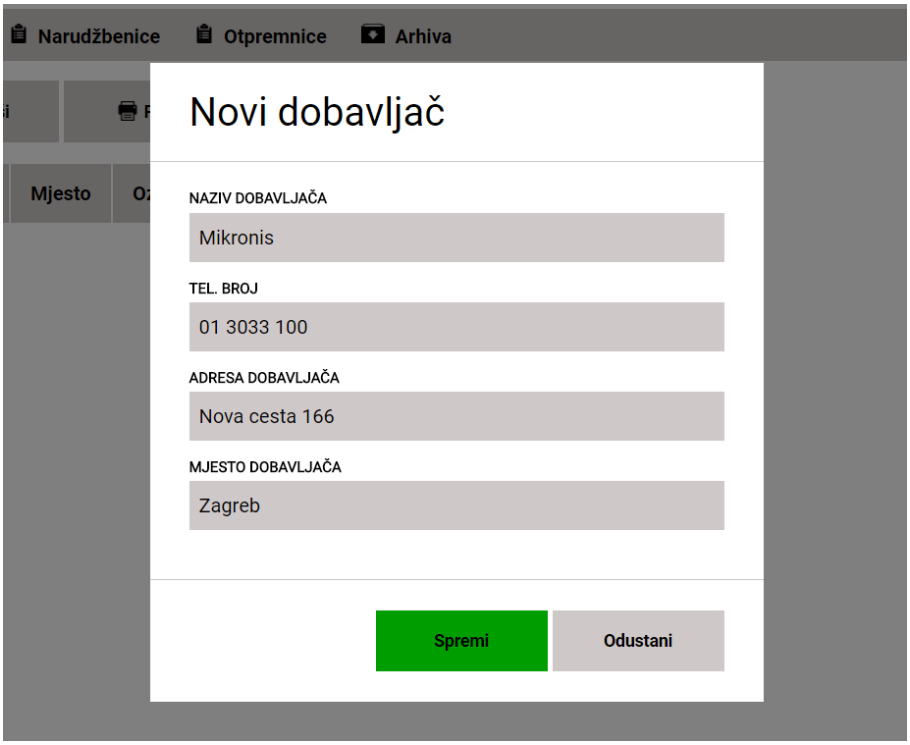
Nakon uspješne prijave prikazuje se početni ekran aplikacije.

Slika 13: Početni ekran aplikacije

Sučelje aplikacije se sastoji od primarne navigacije, sekundarne navigacije i prikaza tablice. Na primarnoj navigaciji se nalazi odabir tablice koju želimo pregledati. Sekundarna navigacija sadrži akcije vezane za pojedinu tablicu, a ispod sekundarne navigacije je prostor rezerviran za prikaz odabrane tablice. Trenutno se nalazimo na podstranici "Skladište", što je ujedno i početni ekran aplikacije. Kao što vidimo na slici, tablica je trenutno prazna. Ranije smo spomenuli da prvo trebamo kreirati dobavljaca, kategoriju i proizvod, da bi mogli naručiti proizvod i dopremiti ga na skladište.

19.2.1. Dobavljači

Prvi korak je kreiranje dobavljača. Na primarnoj navigaciji odaberemo "Dobavljač", a na sekundarnoj „Novi unos“. Nakon toga nam se otvara sučelje za kreiranje dobavljača. Ispunimo prikazanu formu i pritisnemo spremi.



Slika 14: Kreiranje dobavljača

Nakon uspješno kreiranog dobavljača na ekranu se prikazuje poruka o uspješno obavljenoj akciji, te se u tablici prikazuje kreirani dobavljač.

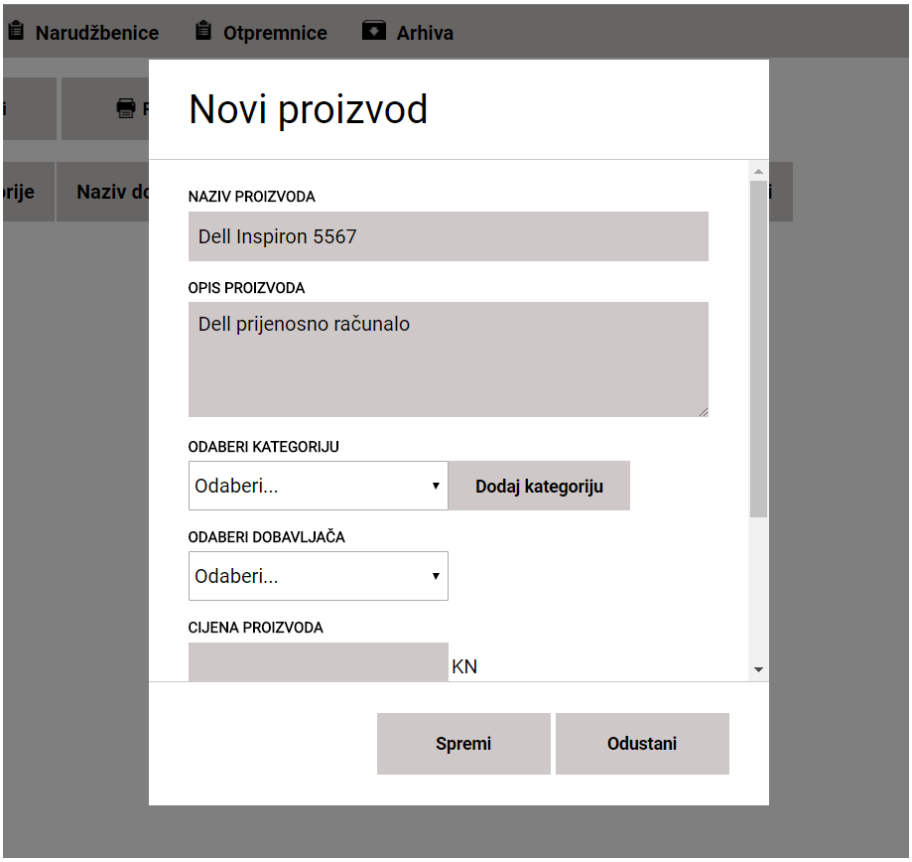
Slika 15: Prikaz tablice "Dobavljači"

Prvi okidač u ovom diplomskom radu služi za osiguravanje referencijalnog integriteta vezanog uz tablicu "dobavljac". Prilikom brisanja zaipisa u tablici "dobavljac" potrebno je provjeriti da li je taj dobavljač referenciran u tablici proizvod. Ako je referenciran postaviti vrijednost odgovarajućeg atributa u tablici „proizvod“ na NULL.

```
DROP TRIGGER referencjalni_integritet;

DELIMITER $$
CREATE TRIGGER referencjalni_integritet
    AFTER DELETE ON dobavljac
    FOR EACH ROW
         BEGIN
          UPDATE proizvod SET sifraDobavljac = NULL
          WHERE sifraDobavljac = OLD.sifraDobavljac;
         END$$
DELIMITER ;
```

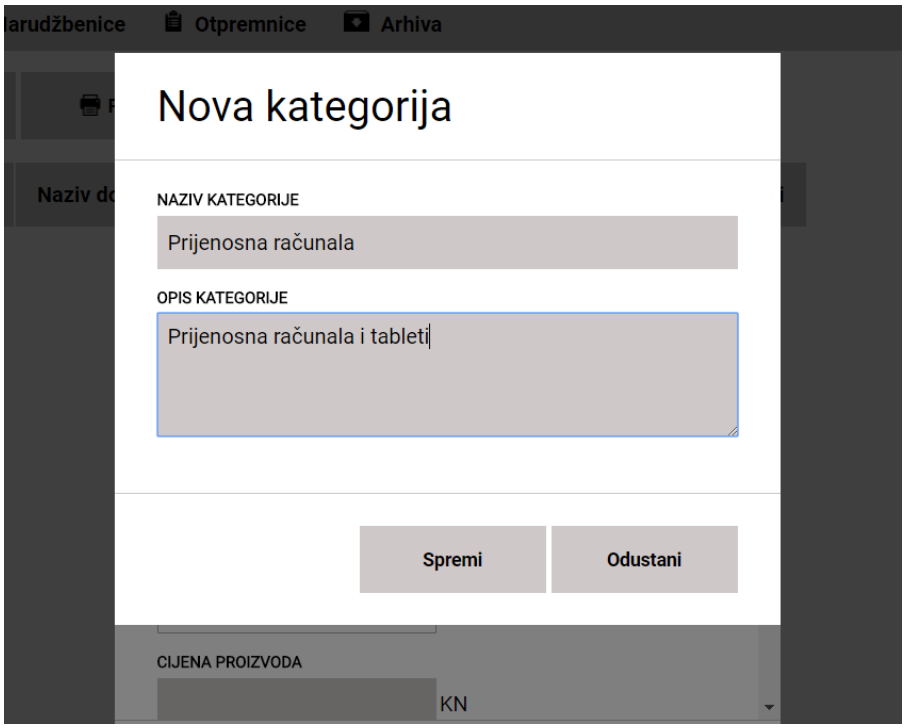
19.2.2. Kategorije i proizvodi

Nakon uspješno kreiranog dobavljača, sljedeći korak je kreiranje katergorije i proizvoda. Kategorija se kreira prilikom kreiranja samog proizvoda. Na primarnoj navigaciji odaberemo "Proizvodi", a zatim na sekundarnoj "Novi unos".



Slika 16: Kreiranje proizvoda

Pritiskom na tipku "Dodaj kategoriju" otvara nam se forma za kreiranje kategorije.



Slika 17: Kreiranje kategorije

Kod padajućeg izbornika odaberemo kreiranu kategoriju i dobavljača kojeg smo malo prije kreirali, te ispunimo preostala polja vezana uz cijenu proizvoda. Nakon toga pritisnemo "Dodaj". Ako je sve uspješno izvršeno, na ekranu bi se trebala ispisati poruka o uspješno izvršenoj akciji i u tablici proizvodi prikazati kreirani proizvod. Ovaj postupak možemo ponoviti još jednom tako da imamo više proizvoda s kojima možemo dalje raditi.

R.br.	Šifra	Naziv	Opis	Naziv kategorije	Naziv dobavljača	Cijena	Iznos pdv-a	Cijena sa pdv-om	Označi
1.	4	Dell Inspiron 5567	Dell prijenosno računalo	Prijenosna računala	Mikronis	1000.00 KN	25 %	1250.00 KN	
2.	5	Toshiba Satellite Pro	Toshiba prijenosno računalo	Prijenosna računala	Mikronis	5000.00 KN	25 %	6250.00 KN	

Slika 18: Prikaz tablice "Proizvodi"

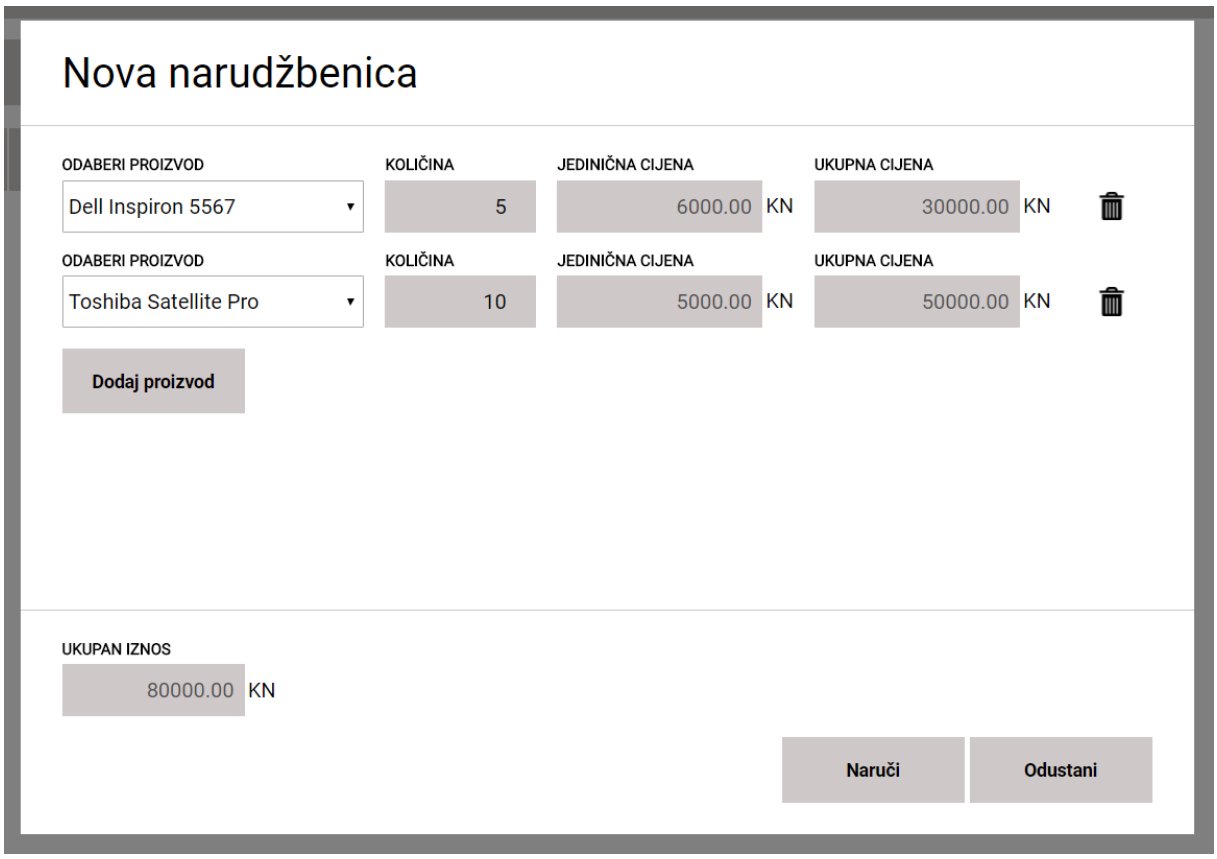
Ranije smo spomenuli da ako postoji pogreška u okidaču aktivirajuća operacija neće se izvesti. Tako da ako je potrebno direktno prekinuti izvođenje određene radnje, rješenje je prouzrokovati pogrešku što će rezultirati prekidom rada i okidača i aktivirajuće operacije. Jedan od načina je pozivom nepostojeće pohranjene procedure.

U sljedećem primjeru je prikazan okidač koji se aktivria na događaj `DELETE` nad tablicom „proizvod". U tijelu okidača nalazi se poziv nepostojeće procedure, čime smo onemogućili brisanje proizvoda iz tablice "proizvod".

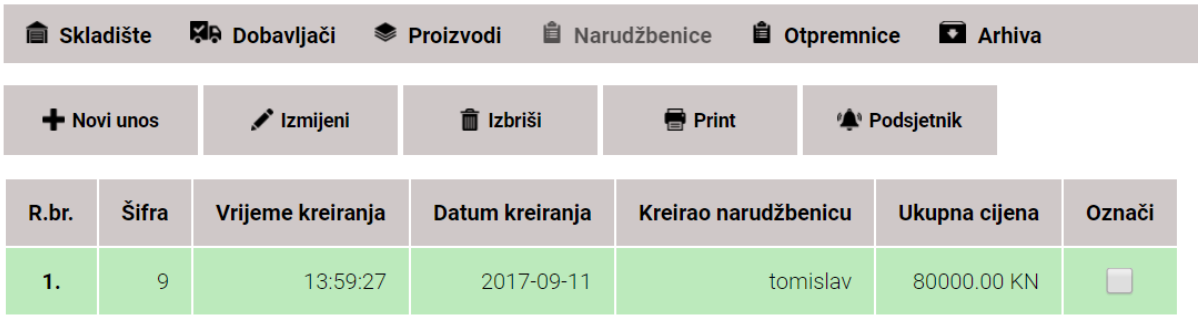
```
DROP TRIGGER nepostojeca_procedura;

DELIMITER $$
CREATE TRIGGER nepostojeca_procedura
    BEFORE DELETE ON proizvod
    FOR EACH ROW
         BEGIN
           CALL nepostojeca_procedura();
         END$$
DELIMITER ;
```

19.2.3. Narudžbenice

Sada kada smo kreirali proizvode, možemo kreirati narudžbenicu. Naručit ćemo 5 Dell i 10 Toshiba laptopa.

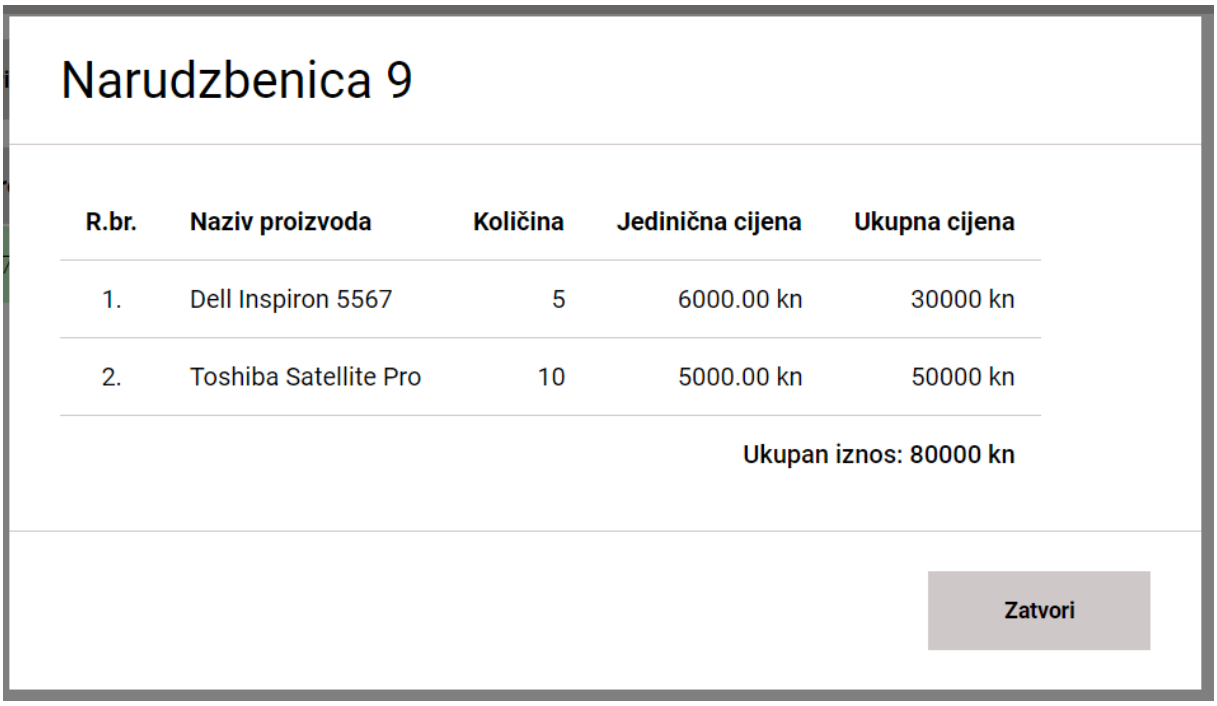


Slika 19: Kreiranje narudžbenice



Slika 20 : Prikaz tablice "Narudžbenice"

Nakon što uspješno kreiramo narudžbenicu, naručeni proizvodi spremaju se u tablicu baze podataka pod nazivom "narudzbenica_detalji". Ukoliko želimo vidjeti te detalje samo pritisnemo redak narudžbenice koju želimo detaljnjije pregledati.



Slika 21: Detalji narudžbenice

Sada dolazimo do prvog okidača. Potrebno je nakon što se proizvod unese u tablicu "narudzbenica_detalji", taj isti proizvod staviti i u tablicu "skladiste", zajedno sa naručenom količinom. Ukoliko taj isti proizvod već postoji na skladištu potrebno je samo mu pribrojiti pristiglu količinu.

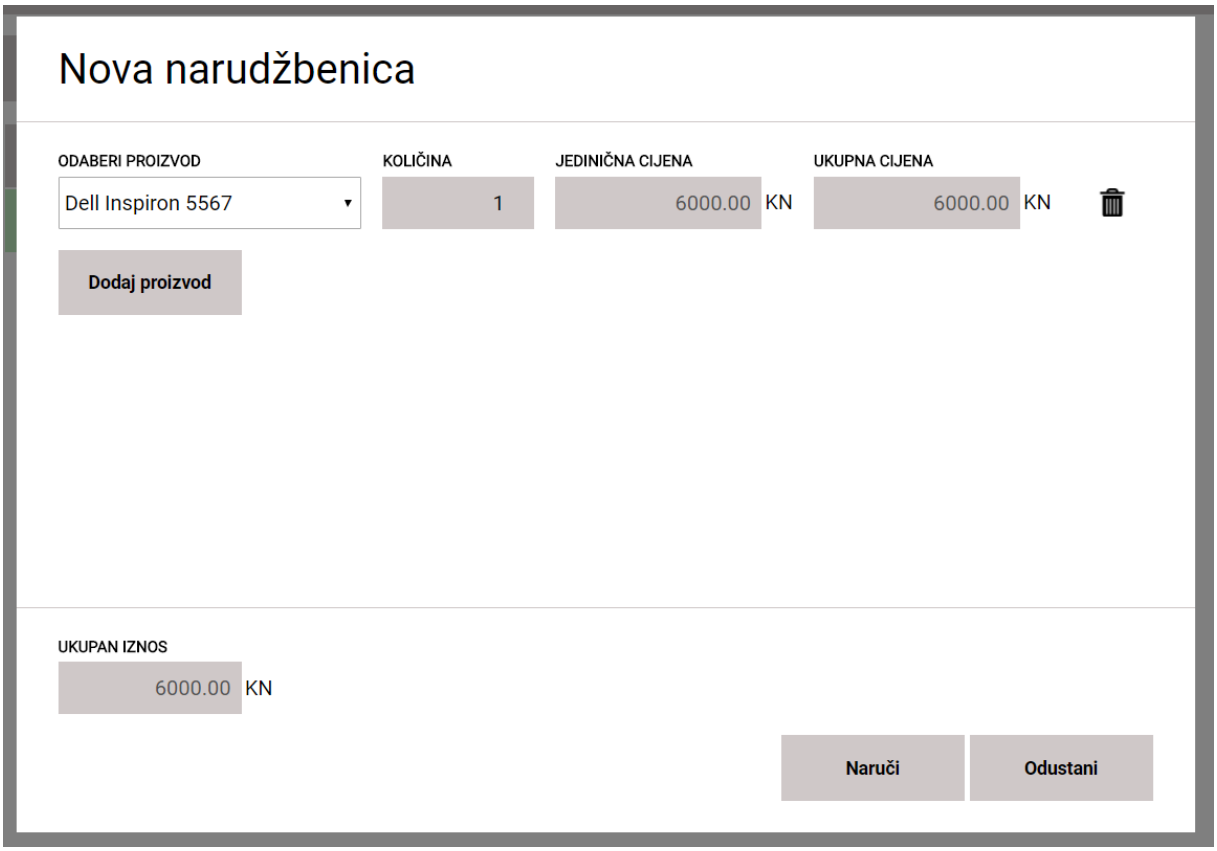
```
DROP TRIGGER narudzba_skladiste;

DELIMITER $$
CREATE TRIGGER narudzba_skladiste
AFTER INSERT ON narudzbenica_detalji
FOR EACH ROW
BEGIN
IF EXISTS (SELECT * FROM skladiste WHERE skladiste.sifraProizvod =
NEW.sifraProizvod)
THEN
  UPDATE skladiste
  SET    skladiste.stanjeProizvod    =    skladiste.stanjeProizvod    +
NEW.kolicina
  WHERE skladiste.sifraProizvod = NEW.sifraProizvod;
ELSE
  INSERT INTO skladiste VALUES (NEW.sifraProizvod, NEW.kolicina);
END IF;
END$$
DELIMITER ;
```

Nakon što se okidač uspješno izvrši možemo provjeriti da li se naručeni proizvodi nalaze na skladištu, samo iz primarne navigacije odaberemo "Skladište".

R.br.	Šifra	Naziv	Opis	Naziv kategorije	Naziv dobavljača	Cijena	Iznos pdv-a	Cijena sa pdv-om	Stanje
1.	4	Dell Inspiron 5567	Dell prijenosno računalo	Prijenosna računala	Mikronis	6000.00 KN	25 %	7500.00 KN	5
2.	5	Toshiba Satellite Pro	Toshiba prijenosno računalo	Prijenosna računala	Mikronis	5000.00 KN	25 %	6250.00 KN	10

Slika 22: Prikaz tablice "Skladište"

Sa priložene slike možemo vidjeti da se okidač zaista uspješno izvršio. Sada ako primjerice naručimo samo laptop Dell i to jedan komad, na skladištu bi trebalo biti 6 takvih laptopa. Pa provjerimo da li je to tako.



Slika 23: Kreiranje narudžbenice

R.br.	Šifra	Naziv	Opis	Naziv kategorije	Naziv dobavljača	Cijena	Iznos pdv-a	Cijena sa pdv-om	Stanje
1.	4	Dell Inspiron 5567	Dell prijenosno računalo	Prijenosna računala	Mikronis	6000.00 KN	25 %	7500.00 KN	6
2.	5	Toshiba Satellite Pro	Toshiba prijenosno računalo	Prijenosna računala	Mikronis	5000.00 KN	25 %	6250.00 KN	10

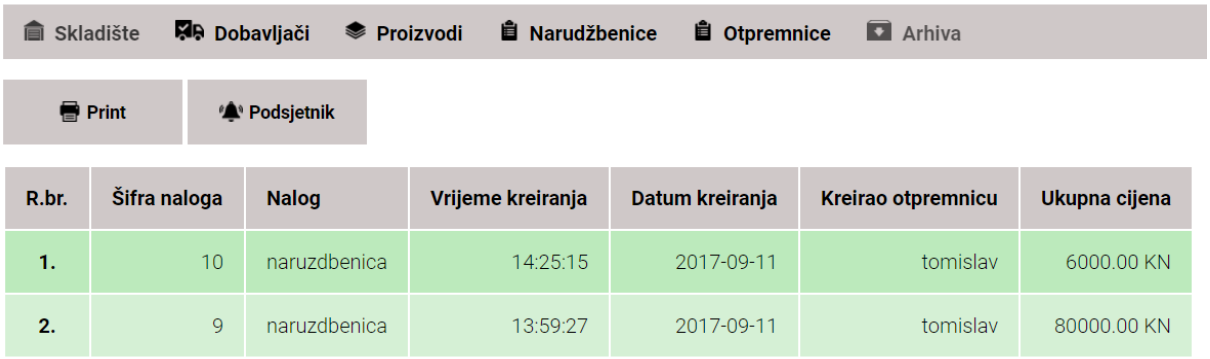
Slika 24: Prikaz tablice "Skladište"

Drugi okidač koji se odnosi na narudžbenicu je arhiviranje kreirane narudžbenice, što znači da se svaka kreirana narudžbenica ujedno sprema i u tablicu "arhiva_naloga". Isto vrijedi i za otpremnice. Prilikom spremanja naloga u tablicu "arhiva_naloga", sprema se i vrijednost 1 ili 2, koje označavaju vrstu naloga (1 – narudžbenica; 2 – otpremnica). Da ne radimo dva okidača, po jedan za svaku tablicu (narudžbenica i otpremnica), ovaj okidač je kreiran uz pomoć procedure. Tijelo okidača se sastoji od poziva procedure i prosljeđivanja odgovarajućih parametara, dok se u proceduri odvija ostatak logike.

```
DROP TRIGGER arhiviranje_narudzbenice;

DELIMITER $$
CREATE TRIGGER arhiviranje_narudzbenice
    AFTER INSERT ON narudzbenica
    FOR EACH ROW
         BEGIN
          CALL arhiviranje_naloga(NEW.sifraNarudzbenica, 1);
         END$$
DELIMITER ;
```

```
DROP PROCEDURE IF EXISTS arhiviranje_naloga;

DELIMITER $$
CREATE
     PROCEDURE    arhiviranje_naloga(IN    sifra_naloga    INT,    IN
sifra_akcije INT)
     BEGIN
          INSERT INTO arhiva_naloga
          SET arhiva_naloga.sifraNaloga = sifra_naloga,
          arhiva_naloga.sifraAkcije = sifra_akcije;
     END;
$$
DELIMITER ;
```

Kada na primarnoj navigaciji odaberamo "Arhiva" možemo vidjeti da se okidač uspješno izvršio i da su izvršene narudžbenice zaista pohranjene i u arhivu. Prikaz naloga pohranjenih u arihvi je sortiran po vremenu izvršavanja.



R.br.	Šifra naloga	Nalog	Vrijeme kreiranja	Datum kreiranja	Kreirao otpremnicu	Ukupna cijena
1.	10	naruzdbenica	14:25:15	2017-09-11	tomislav	6000.00 KN
2.	9	naruzdbenica	13:59:27	2017-09-11	tomislav	80000.00 KN

Slika 25: Prikaz tablice "Arhiva"

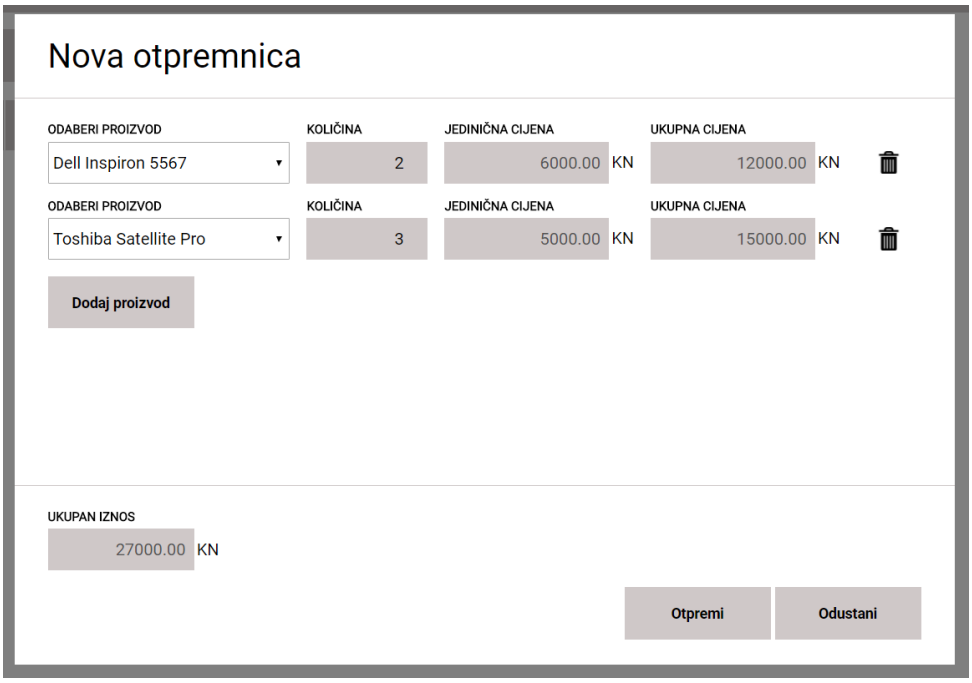
Posljednji okidač koji se odnosi na narudžbenicu okida se prilikom brisanja narudžbenice. Pri svakom brisanju narudžbenice treba se osim brisanja narudžbenice iz tablice "narudzbenica" obrisati i podaci iz tablice "narudzbenica_detalji".

```
DROP TRIGGER narudzbenica_delete;

DELIMITER $$
CREATE TRIGGER narudzbenica_delete
    AFTER DELETE ON narudzbenica
    FOR EACH ROW
         BEGIN
           DELETE FROM narudzbenica_detalji
           WHERE              narudzbenica_detalji.sifraNarudzbenica=
OLD.sifraNarudzbenica;
         END$$
DELIMITER ;
```

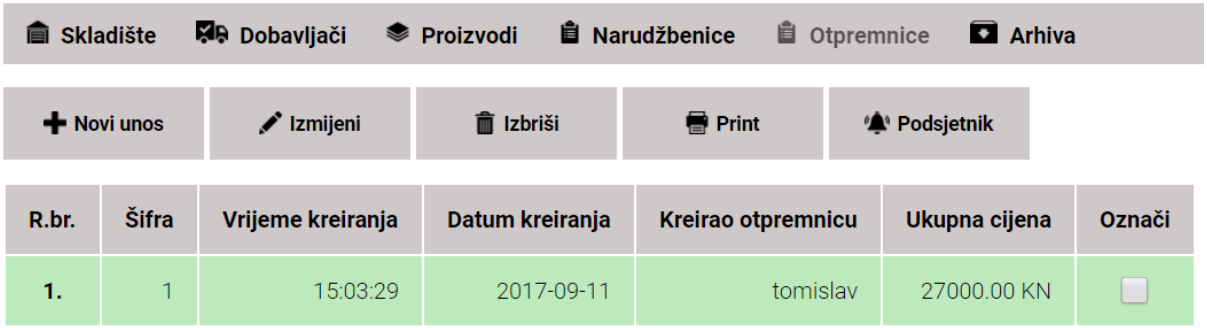
Da bi uspješno izbrisali narudžbnicu ili više njih moramo ih odabrati pritiskom na okvir koji se nalazi u zadnjem stupcu svake narudžbenice, te zatim pritisnuti tipku "Izbriši" koja se nalazi na sekundarnoj navigaciji. Ova operacija ne briše narudžbenice iz arhive, kao što ni ne poništava proizvode na skladištu na koje se narudžbenica odnosila. Isto vrijedi i za otpremnice.

19.2.4. Otpremnice

Sada kada proizvodi postoje na skladištu možemo kreirati otpremnicu. Postupak je sličan kao i kod naručivanja, samo što ovdje postoji ograničenje količine pojedinog proizvoda kojeg možemo otpremiti. Količina otpreme proizvoda ne može i ne smije biti veća od količine koja se trenutno nalazi na skladištu. Na izbor imamo dva proizvoda koja smo prethodno naručili. Otpremit ćemo 2 Dell laptopa i 3 Toshiba laptopa, te nakon toga provjeriti detalje narudžbenice da vidimo da li se podudaraju. Postupak je identičan kao i kod narudžbenica.



Slika 26: Kreiranje otpremnice



Slika 27: Prikaz tablice "Otpremnice"

Slika 28: Detalji otpremnice

Sada dolazimo do prvog okidača koji se odnosi na otpremnice. Isto kao i kod narudžbenica, nakon što smo otpremili proizvode prvo što moramo je ažurirati podatke na skladištu. Ukoliko smo otpremili svu trenutnu količinu pojedinog proizvoda na skladištu, odnosno ako je količina pojedinog proizvoda nakon otpreme na skladištu jednaka nuli, moramo ponoviti zadnju narudžbu tog proizvoda, što znači da moramo kreirati novu narudžbenicu, a zatim i nove detalje narudžbenice.

```
DROP TRIGGER otprema_skladiste;

DELIMITER $$
CREATE TRIGGER otprema_skladiste
    AFTER INSERT ON otpremnica_detalji
    FOR EACH ROW
         BEGIN
         DECLARE quantity INT;
         DECLARE price DECIMAL(10,2);

          UPDATE skladiste
          SET skladiste.stanjeProizvod = skladiste.stanjeProizvod -
NEW.kolicina
          WHERE skladiste.sifraProizvod = NEW.sifraProizvod;
          IF    EXISTS    (SELECT    *    FROM    skladiste    WHERE
skladiste.sifraProizvod        =        NEW.sifraProizvod        AND
skladiste.stanjeProizvod = 0)
          THEN
```

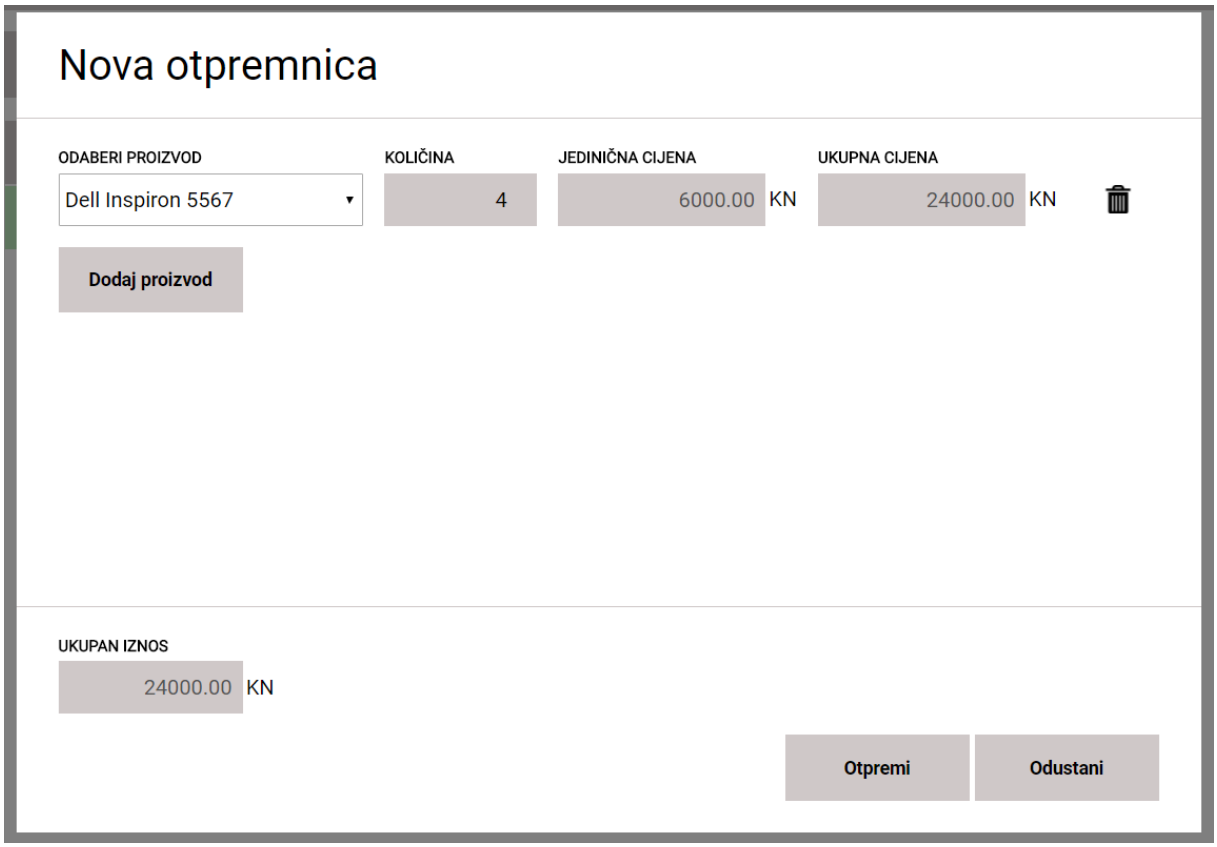
```
                SELECT
narudzbenica_detalji.kolicina,narudzbenica_detalji.cijena        INTO
quantity, price FROM narudzbenica_detalji, narudzbenica
                WHERE      narudzbenica_detalji.sifraProizvod      =
NEW.sifraProizvod
                AND     narudzbenica_detalji.sifraNarudzbenica     =
narudzbenica.sifraNarudzbenica
                ORDER BY datum DESC, vrijeme DESC
                LIMIT 1;

                INSERT INTO narudzbenica
                SET vrijeme = CURTIME(),
                    datum = CURDATE(),
                    sifraZaposlenik` = 0,
                    ukupanIznos = price * quantity;

                INSERT    INTO    narudzbenica_detalji    VALUES    (
LAST_INSERT_ID(), NEW.sifraProizvod, price, quantity);

            END IF;
          END$$
DELIMITER ;
```

Nakon što smo otpremili proizvode sa prehodno kreiranom narudžbenicom, po stanju na skladištu možemo vidjeti da se okidač uspješo izvršio.

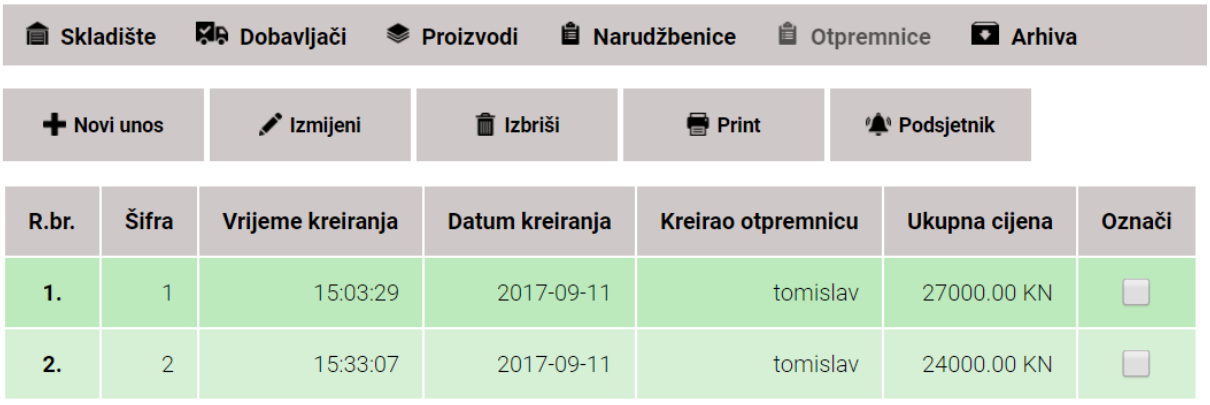
R.br.	Šifra	Naziv	Opis	Naziv kategorije	Naziv dobavljača	Cijena	Iznos pdv-a	Cijena sa pdv-om	Stanje
1.	4	Dell Inspiron 5567	Dell prijenosno računalo	Prijenosna računala	Mikronis	6000.00 KN	25 %	7500.00 KN	4
2.	5	Toshiba Satellite Pro	Toshiba prijenosno računalo	Prijenosna računala	Mikronis	5000.00 KN	25 %	6250.00 KN	7

Slika 29: Prikaz tablice "Skladište"

Sada ćemo kreirati novu otpremnicu kojom ćemo otpremiti sva preostala Dell prijenosna računala. Zatim ćemo provjeriti da li je okidač uspješno kreirao novu narudžbenicu i naručio onoliko Dell prijenosnih računala koliko ih je bilo u zadnjoj narudžbenici koja se odnosila na njih. Podsjetimo se zadnji put smo naručili 1 komad dell prijenosnog računala.

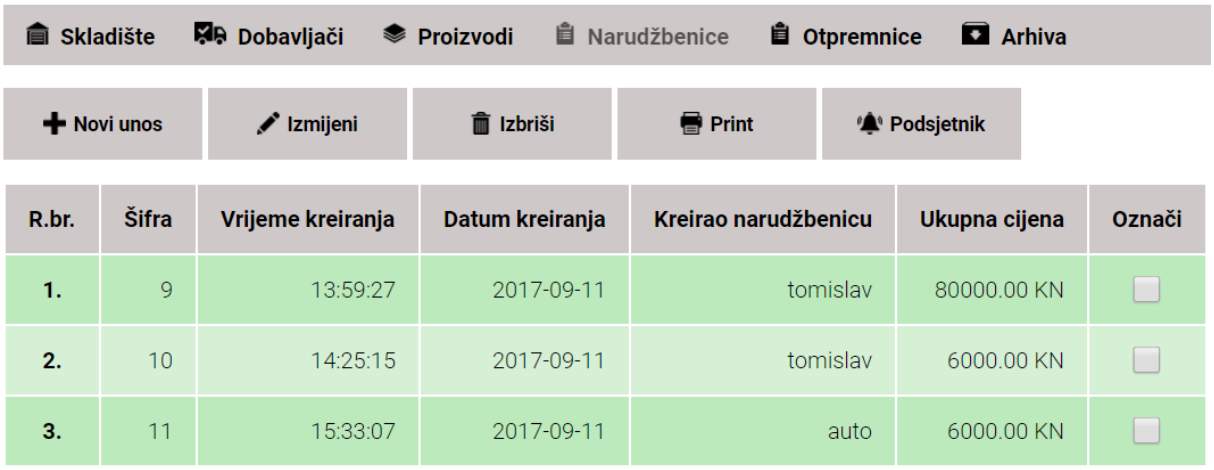


Slika 30: Kreiranje otpremnice



R.br.	Šifra	Vrijeme kreiranja	Datum kreiranja	Kreirao otpremnicu	Ukupna cijena	Označi
1.	1	15:03:29	2017-09-11	tomislav	27000.00 KN	☐
2.	2	15:33:07	2017-09-11	tomislav	24000.00 KN	☐

Slika 31: Prikaz tablice "Otpremnice"



R.br.	Šifra	Vrijeme kreiranja	Datum kreiranja	Kreirao narudžbenicu	Ukupna cijena	Označi
1.	9	13:59:27	2017-09-11	tomislav	80000.00 KN	☐
2.	10	14:25:15	2017-09-11	tomislav	6000.00 KN	☐
3.	11	15:33:07	2017-09-11	auto	6000.00 KN	☐

Slika 32: Prikaz tablice "Narudzbenice"

Sa slike iznad vidljivo je da se okidač uspješno izvršio i da je kreirana nova narudžbenica. Pod kolumnom "Kreirao narudžbenicu" piše vrijednost "auto" iz razloga što je ovo automatski kreirana narudžbenica. Kako bi se dodatno uvjerili i ispravnost okidača možemo provjeriti detalje narudžbenice.

Slika 33: Detalji narudžbenice

U prethodnom poglavlju spomenuli smo okidač koji se odnosi na narudžbenice, a koji prilikom svake kreirane narudžbenice, ažurira proizvode na skladištu, pa provjerimo da li se uspješno izvršio. Prije ovih operacije stanje dell prijenosnih računala je bilo 4. Nakon što smo otpremili ta 4 komada, kreirala se automatska narudžbenica koja je naručila 1 komad, tako da bi trenutno stanje tog proizvoda na skladištu trebalo biti 1.

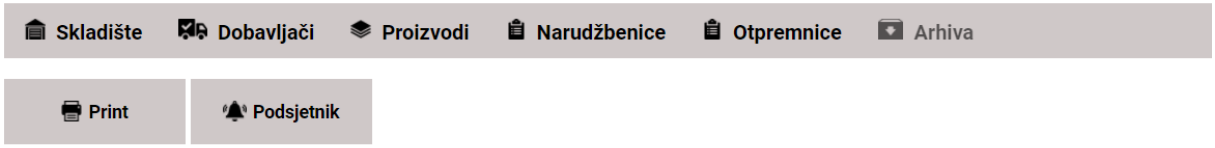
R.br.	Šifra	Naziv	Opis	Naziv kategorije	Naziv dobavljača	Cijena	Iznos pdv-a	Cijena sa pdv-om	Stanje
1.	4	Dell Inspiron 5567	Dell prijenosno računalo	Prijenosna računala	Mikronis	6000.00 KN	25 %	7500.00 KN	1
2.	5	Toshiba Satellite Pro	Toshiba prijenosno računalo	Prijenosna računala	Mikronis	5000.00 KN	25 %	6250.00 KN	7

Slika 34: Prikaz tablice "Skladište"

Sa slike iznad je vidljivo da su sve operacije uspješno izvršene.

Preostala dva okidača koji se odnose na otpremnice su slični kao i kod narudžbenica. Prvi od ta dva odnosi se na arhiviranje kreirane otpremnice. Tijelo tog okidača, kao i kod narudžbenice, se sastoji od poziva procedure, a zatim se u proceduri dalje odvija arhiviranje. Jedina razlika je što u ovom slučaju u proceduru se kao vrsta naloga šalje broj 2, a on označava otpremnicu.

```
DROP TRIGGER arhiviranje_otpremnice;

DELIMITER $$
CREATE TRIGGER arhiviranje_otpremnice
    AFTER INSERT ON otpremnica
    FOR EACH ROW
         BEGIN
           CALL arhiviranje_naloga(NEW.sifraOtpremnica, 2);
         END$$
DELIMITER ;
```



R.br.	Šifra naloga	Nalog	Vrijeme kreiranja	Datum kreiranja	Kreirao otpremnicu	Ukupna cijena
1.	11	naruzdbenica	15:33:07	2017-09-11	auto	6000.00 KN
2.	2	otpremnica	15:33:07	2017-09-11	tomislav	24000.00 KN
3.	1	otpremnica	15:03:29	2017-09-11	tomislav	27000.00 KN
4.	10	naruzdbenica	14:25:15	2017-09-11	tomislav	6000.00 KN
5.	9	naruzdbenica	13:59:27	2017-09-11	tomislav	80000.00 KN

Slika 35: Prikaz tablice "Arhiva"

Kao što smo već spomenuli zadnji okidač se odnosi na brisanje otpremnice i njenih detalja.

```
DROP TRIGGER otpremnica_delete;

DELIMITER $$
CREATE TRIGGER otpremnica_delete
    AFTER DELETE ON otpremnica
    FOR EACH ROW
         BEGIN
          DELETE FROM otpremnica_detalji
          WHERE         otpremnica_detalji.sifraOtpremnica         =
OLD.sifraOtpremnica;
         END$$
DELIMITER ;
```

18.2.5. Skladište

U prethodnim poglavljima već smo opisali njegovu funkciju. Sada ćemo opisati tri okidača koja se odnose na njega. Radi se o okidačima koji svaku promjenu na skladištu spremaju i u tablicu skladiste_backup. Svaki okidač ima jedan od tri događaja koji ga okidaju: `INSERT`, `UPDATE` i `DELETE`. Nažalost, još uvijek u MySQL-u ne postoji opcija za kreiranje jednog okidača koji se koristi za više događaja, tako da smo u ovom slučaju morali kreirati okidač za svaki događaj. Suprotnost je Microsoft SQL koji omogućuje kreiranje jednog okidača za više događaja.

```
DROP TRIGGER skladiste_backup_insert;

DELIMITER $$
CREATE TRIGGER skladiste_backup_insert AFTER INSERT
    ON skladiste
    FOR EACH ROW
         BEGIN
          INSERT      INTO      skladiste_backup      (sifraProizvod,
nazivProizvod, opisProizvod)
          SELECT  sifraProizvod,  nazivProizvod,  opisProizvod  FROM
proizvod
          WHERE  sifraProizvod = NEW.sifraProizvod;
          UPDATE     skladiste_backup     SET     stanjeProizvod     =
NEW.stanjeProizvod
          WHERE  sifraProizvod = NEW.sifraProizvod;
         END$$
DELIMITER ;
```

```
DROP TRIGGER skladiste_backup_update;

DELIMITER $$
CREATE TRIGGER skladiste_backup_update AFTER UPDATE
    ON skladiste
```

```
    FOR EACH ROW
         BEGIN
          UPDATE     skladiste_backup     SET     stanjeProizvod     =
NEW.stanjeProizvod
          WHERE  sifraProizvod = NEW.sifraProizvod;
         END$$
DELIMITER ;
```

```
DROP TRIGGER skladiste_backup_delete;

DELIMITER $$
CREATE TRIGGER skladiste_backup_delete AFTER DELETE
    ON skladiste
    FOR EACH ROW
         BEGIN
          DELETE   FROM   skladiste_backup   WHERE   sifraProizvod   =
OLD.sifraProizvod;
         END$$
DELIMITER ;
```

Sada možemo provjeriti da li su se tijekom svih ovih operacija ova tri okidača uspješno izvršavala. To možemo provjeriti u tablici "skladiste_backup".

sifraProizvod	nazivProizvod	opisProizvod	stanjeProizvod
4	Dell Inspiron 5567	Dell prijenosno računalo	1
5	Toshiba Satellite Pro	Toshiba prijenosno računalo	7

Slika 36: Prikaz tablice "skladiste_backup" iz baze podataka

18.2.6. DDL okidač

Jedan od nedostataka MySQL-a je što još uvijek neomogućava kreiranje okidača nad DDL događajima. Sljedeći okidač napravljen je za Microsoft SQL Server Management. Okidač služi za praćenje svih događaja koji su vezani uz tablicu. Kada se dogodi pojedini događaj nad tablicom pojedine informacije vezane za taj događaj se spremaju u novu tablicu.

```
CREATE TRIGGER pracenje_promjena
ON DATABASE
FOR CREATE_TABLE, ALTER_TABLE, DROP_TABLE
AS
BEGIN
DECLARE @EventData XML
SELECT @EventData = EVENTDATA()

INSERT    INTO    tablicne_promjene    (nazivTablice,    vrstaDogadaja,
nazivKorisnika, SQLNaredba, vrijemeDogađaja)
VALUES
(
@EventData.value('(/EVENT_INSTANCE/ObjectName)[1]', 'varchar(250)'),
```

```
@EventData.value('(/EVENT_INSTANCE/EventType)[1]', 'nvarchar(250)'),
@EventData.value('(/EVENT_INSTANCE/LoginName)[1]', 'varchar(250)'),
@EventData.value('(/EVENT_INSTANCE/TSQLCommand)[1]',
'nvarchar(2500)'),
GetDate()
)
END
```

18.2.7. Pregled okidača

Sve informacije o kreiranim okidačima možemo pronaći u informacijskoj shemi (eng. *Information schema*) ili pomoću naredbe `SHOW TRIGGERS`.

```
SELECT * FROM information_schema.triggers;
```

TRIGGER_CATALOG	TRIGGER_SCHEMA	TRIGGER_NAME	EVENT_MANIPULATION	EVENT_OBJECT_CATALOG	EVENT_OBJECT_SCHEMA	EVENT_OBJECT_TABLE
def	warehouse_manager	referencjalni_integritet	DELETE	def	warehouse_manager	dobavljac
def	warehouse_manager	arhiviranje_narudzbenice	INSERT	def	warehouse_manager	narudzbenica
def	warehouse_manager	narudzbenica_delete	DELETE	def	warehouse_manager	narudzbenica
def	warehouse_manager	narudzba_skladiste	INSERT	def	warehouse_manager	narudzbenica_detalji
def	warehouse_manager	arhiviranje_otpremnice	INSERT	def	warehouse_manager	otpremnica
def	warehouse_manager	otpremnica_delete	DELETE	def	warehouse_manager	otpremnica
def	warehouse_manager	otprema_skladiste	INSERT	def	warehouse_manager	otpremnica_detalji
def	warehouse_manager	nepostojeca_procedura	DELETE	def	warehouse_manager	proizvod
def	warehouse_manager	skladiste_backup_insert	INSERT	def	warehouse_manager	skladiste
def	warehouse_manager	skladiste_backup_update	UPDATE	def	warehouse_manager	skladiste
def	warehouse_manager	skladiste_backup_delete	DELETE	def	warehouse_manager	skladiste

Slika 37: Pregled okidača preko informacijske sheme

```
SHOW TRIGGERS
```

Trigger	Event	Table	Statement		Timing
referencjalni_integritet	DELETE	dobavljac	BEGINUPDATE `proizvod` SET `sifraDobavljac` = NULL WHE...	115B	AFTER
arhiviranje_narudzbenice	INSERT	narudzbenica	BEGINCALL `arhiviranje_naloga`(NEW.`sifraNarudzbenica`, 1...	71B	AFTER
narudzbenica_delete	DELETE	narudzbenica	BEGINDELETE FROM `narudzbenica_detalji` WHERE `narudzbeni...	129B	AFTER
narudzba_skladiste	INSERT	narudzbenica_detalji	BEGINIF EXISTS (SELECT * FROM `skladiste` WHERE skladiste...	366B	AFTER
arhiviranje_otpremnice	INSERT	otpremnica	BEGINCALL `arhiviranje_naloga`(NEW.`sifraOtpremnica`, 2);...	69B	AFTER
otpremnica_delete	DELETE	otpremnica	BEGINDELETE FROM `otpremnica_detalji` WHERE `otpremnica_d...	122B	AFTER
otprema_skladiste	INSERT	otpremnica_detalji	BEGIN DECLARE quantity INT; DECLARE price DECIMAL(1...	1K	AFTER
nepostojeca_procedura	DELETE	proizvod	BEGIN END	18B	BEFORE
skladiste_backup_insert	INSERT	skladiste	BEGININSERT INTO `skladiste_backup` (`sifraProizvod`, `na...	342B	AFTER
skladiste_backup_update	UPDATE	skladiste	BEGINUPDATE `skladiste_backup` SET `stanjeProizvod` = NEW...	135B	AFTER
skladiste_backup_delete	DELETE	skladiste	BEGINDELETE FROM `skladiste_backup` WHERE `sifraProizvod`...	92B	AFTER

Slika 38: Pregled okidača preko naredbe SHOW TRIGGERS

20. Zaključak

Upravljanje skladišnim aktivnostima jedna je od najvažnijih stavki uspješnog skladišnog poslovanja. Uspješno upravljanje tim aktivnostima omogućuje efikasno ispunjavanje svih zahtjeva proizvodnje i kupaca. U suvremeno vrijeme potrebno je konstantno odgovarati na nove zahtjeve tržišta, te se kao jedno od rješenja nudi računalni susatv za upravljenje skladištem podataka. Kod takvog sustava treba voditi računa o više čimbenika. Osim već prethodno nabrojanih, važno je i gledati u budćnost poslovanja, što znači da treba odabrati sustav koje će to poduzeće moći koristiti duže vrijeme.

Što se tiče same aplikacije i korištenje aktivnih baza podataka, mišljenja sam da MySQL u usporedbi s Microsoft SQL Server Management Studiom ima nekoliko nedostataka. Kao što sam ranije u radu naveo Microsoft SQL omogućava izradu jednog okidača koji djeluje na više događaja, što u MySQL još uvijek nije integrirano, no najavljeno je da će upravo tu funkcionalnost uključiti u najnoviju verziju kada je objave. Još jedan nedostatak je nemogućnost kreiranja okidača nad DDL naredbama, što je isto tako već omogućeno u Microsoft SQL-u. Iako mislim da MySQL može poslužiti sasvim dobro za određene aplikacije, ali kada se radi o složenijim aplikacijama i sustavim koji se baziraju na aktivnim bazama podataka onda prednost dajem Microsoft SQL-u.

Literatura

[1] Anon., n.d. *Napredne baze podataka: Okidači,* Zagreb: Tehničko veleučilište u Zagrebu.

[2] Anon., n.d. *Napredne baze podataka: Procedure i funkcije,* Zagreb: Tehničko veleučilište u Zagrebu.

[3] Anon., n.d. *Napredne baze podatka: Transakcije,* Zagreb: Tehničko veleučilište u Zagrebu.

[4] Baranović, M. & Zakošek, S., 2007. *Baze podataka.* Zagreb: Sveučilište u Zagrebu Fakultet elektronike i računarstva.

[5] Baranović, M. & Zakošek, S., 2012. *Baze podataka.* Zagreb: Sveučilište u Zagrebu Fakultet elektronike i računarstva.

[6] Bušić, D., 2015. *Prikaz i analiza skladišnog sustava poduzeća.* Zagreb: Sveučilište u Zagrebu, Fakultet strojarstva i brodogradnje.

[7] Duk, S., 2012. *Okidači,* Zagreb: Tehničko veleučilište u Zagrebu.

[8] Duk, S., 2012. *Pohranjeni podaci,* Zagreb: Tehničko veleučilište u Zagrebu.

[9] Duk, S., 2012. *Zaštita od neovlaštenog pristupa,* Zagreb: Tehničko veleučilište u Zagrebu.

[10] Elmasri, R. & Navathe, S., 2010. *Fundamentals of Database Systems.* 6th ur. s.l.:Fundamentals of Database Systems.

[11] Goluža, A., 2016. *Analiza skladišnog sustava u farmaceutskoj industriji.* Zagreb: Sveučilište u Zagrebu, Fakultet prometnih znanosti.

[12] IQMS, 2016. *IQMS.* [Mrežno]
Dostupno na: https://erpblog.iqms.com/what-is-warehouse-management-system/
[Preuzeto 06/09/2017].

[13] Kramberger, T., 2011. *Skripta iz kolegija „Baze podataka“,* Zagreb: Tehničko veleučilište u Zagrebu.

[14] Murray, M., 2016. *the balance.* [Mrežno]
Dostupno na: https://www.thebalance.com/implementing-a-warehouse-management-system-wms-2221330
[Preuzeto 06/09/2017].

[15] Oracle, 2014. *Oracle Docs.* [Mrežno]
Dostupno na:
http://docs.oracle.com/cd/E11882_01/appdev.112/e25519/triggers.htm#LNPLS99888
[Preuzeto 20/03/2017].

[16] Oracle, 2017. *MySQL.* [Mrežno]
Dostupno na: https://www.mysql.com/
[Preuzeto 20/03/2017].

[17] Paton, N. & Diaz, O., 1999. *Active Database Systems.* s.l.:ACM Computing Surveys.

[18] QSTOCK, n.d. *QStock Inventory.* [Mrežno]
Dostupno na: http://www.qstockinventory.com/blog/warehouse-management-system/
[Preuzeto 06/09/2017].

[19] Rouse, M., 2009. *TechTarget.* [Mrežno]
Dostupno na: http://searcherp.techtarget.com/definition/warehouse-management-system-WMS
[Preuzeto 06/09/2017].

[20] Sharma, N. i dr., 2010. *Database Fundamentals.* Markham: IBM Corporation.

[21] Silberschatz, A., Korth, H. & Sudarshan, 2010. *Database system concepts.* 6th ur. New York: McGraw-Hill Education.

[22] Smolko, T., 2015. *NOS poslovno savjetovanje.* [Mrežno]
Dostupno na: http://www.nos.hr/tema-tjedna-upravljanje-skladistem/
[Preuzeto 06/09/2017].

[23] Zaniolo, C. i dr., 1997. *Advanced Database Systems.* San Francisco: Morgan Kaufmann Publishers.

Popis slika i tablica