BAZE PODATAKA 1

VODIČ ZA PRIPREMU USMENOG DELA ISPITA

POPIS MATERIJALA PO TEMAMA

Osnovni pojmovi
[1], [4] 1. glava, [6] 1. glava, [7] 1. Glava

- Podatak/informacija/metapodaci - definicije, primer
- Fajl sistem u ulozi sistema za čuvanje i obradu podataka – nedostaci
- Baza podataka, SUBP, Sistem baza podataka – definicije
- Zadaci Baza podataka
- Okruženje baza podataka
- Uloga SUBPa
- Osnovne komponente SUBPa

Modeli podataka
[1]
[5] 3. glava i to 3.1.1 – 3.2.1.2, 3.2.1.5, 3.1.1

- ANSI troslojna arhitektura baze podataka [3]-1.1
- Relacioni model
  - Struktura
  - Operacije
  - Null vrednosti
  - Operacije sa null vrednostima

SQL (Structured Query Language)

SKRIPTA U NASTAVKU

Normalizacija
DATA SKRIPTA = [5] 17.4.1-17.4.7

SQL (Structured Query Language)

Struktuirani upitni jezik

- najšire korišćeni komercijalni jezik relacionih baza podataka.
- nastao kao rezultat standardizacije[1] organizacije i manipulacije podataka u relacionim bazama podataka; standard je prvi put definisan 1986. i nakon toga dopunjavan 1989., 1992., 1995., 1999., 2003., 2006., 2008., 2011. godine
  SQL89 (ili SQL1) , SQL92 (SQL2), SQL99 (SQL3)
- U osnovi **neproceduralan** jezik.

```
SELECT *
FROM Zaposleni
WHERE Godine > 40 OR Zarada > 60000
```

  Verzija SQL:2003 unela karakteristike proceduralnih jezika kao što su BEGIN blokovi, IF naredbe, funkcije i procedure.

Tri osnovne funkcije koje podržava SQL su:

- definicija baze podataka: pre početka rada sa bazom podataka neophodno je definisati njenu strukturu - tabele, atributi, tipovi, ograničenja unutar tabela i između njih, pomoćne strukture (indeksi) za ubrzanje pristupa podacima postoje i za koje tabele;
  ova komponenta jezika odgovara **DDL-jeziku baze podataka** (Data Definiton Language);
- manipulacija bazom podataka: pored upita nad bazom podataka, kojima dobijamo željene informacije, neophodno je obezbediti i ažuriranje baze podataka, odnosno unos, izmenu i brisanje podataka;
  ova komponenta je u stvari **DML-jezik baze podataka** (Data Manipulation Language);
- kontrola pristupu podacima: u svakoj bazi podataka neophodno je ostvariti kontrolu koji korisnici imaju pristup kojim podacima i šta mogu da rade sa tim podacima;
  ova komponenta predstavlja **DCL-jezik baze podataka** (Data Control Language).

SQL jezik podržava više režima rada sa bazom podataka:

- **interaktivni**: korisnik zadaje jednu po jednu SQL naredbu interaktivno; pristup bazi podataka je ograničen jedino pravima korisnika;
- **programski**: korisnik pokreće program u kome se nalaze "ugrađene" SQL naredbe; pristup bazi podataka ograničen je pravima korisnika i sadržajem programa; pri tome, ugrađene naredbe mogu biti:
  - **statičke** (fiksirane u vreme prevođenja programa) ili
  - **dinamičke** (konstruisane tokom izvršavanja programa).

Skup svih komponenata koje čine okruženje i koje rade međusobno povezano sa ciljem čuvanja i operisanja podacima uz podršku SQL opearacija čini **SQL okruženje**. SQL okruženje predstavlja u osnovi neki relacioni SUBP. Komponente formiraju model na kome se RDBMS sistem bazira, pa je model specifičan za svaku RDBMS platformu.

[1] Motivacija za uvođenje standarda je bila činjenica da se najveća vrednost u oblasti informatike nalazi u ogromnom broju programa raznovrsne namene napisanih na raznim programskim jezicima. Sa prvim standardima usvojeno je načelo "vertikalne kompatibilnosti", po kome svaki novi standard treba da sadrži i sve mogućnosti prethodnog standarda. Zahvaljujući tome, jednom razvijeni programi mogu su se mogli relativno lako prenositi sa jedne platform na drugu.

1. DEFINISANJE PODATAKA

U standardnom SQL-jeziku manipulacija definicijama podataka se postiže upotrebom tri fraze:

- **CREATE** služi za kreiranje nekog objekta (tabele, indeksa itd.) u bazi podataka;
- **DROP** služi za uklanjanje definicije nekog objekta iz baze podataka;
- **ALTER** služi za izmenu definicije nekog objekta u bazi podataka

Neki od objekata koji se mogu definisati su:

- baza podataka (nije predviđena SQL standardom)
- shema
- tabela
- ograničenje
- indeks
- pogled

Pošto DROP komanda nema nekih posebnih opcija u osnovnom obliku navešćemo je pre svih. Njen oblik je

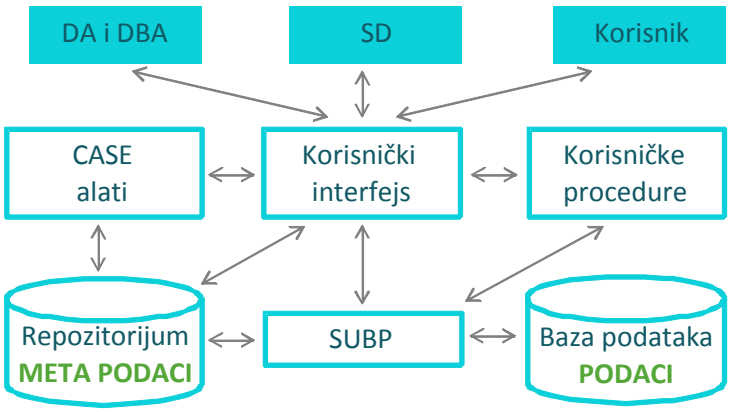
```
DROP tip naziv
DROP VIEW [ schema_name . ] view_name [ ...,n ] [ ; ]
DROP DATABASE { database_name } [ ,...n ] [;]
```

Primeri.

```
DROP TABLE Studenti;
DROP DATABASE STUDIJE;
```

Katalog, Šema, INFORMATION_SCHEMA

U uobičajenim postavkama okruženja baze podataka (kao kolekcije logički povezanih podataka), kao i SQL okruženja meta podaci se mogu posmatrati kao posebna komponenta koja se najčešće naziva **repozitorijumom** ili **katalogom**. Baza podataka sadrži objekte koji su logički grupisani po šemama, a jedna ili više logički povezanih šema čine katalog.



Katalog je imenovana kolekcija šema baze podataka u SQL okruženju. Sam katalog obezbeđuje hijerarhijsku strukturu za organizovanje podataka u okviru šema.

Šema je kontejner za objekte kao što su tabele, pogledi i domeni.

SQL okruženje sadrži nula ili više kataloga, a katalog sadrži jednu ili više šema.

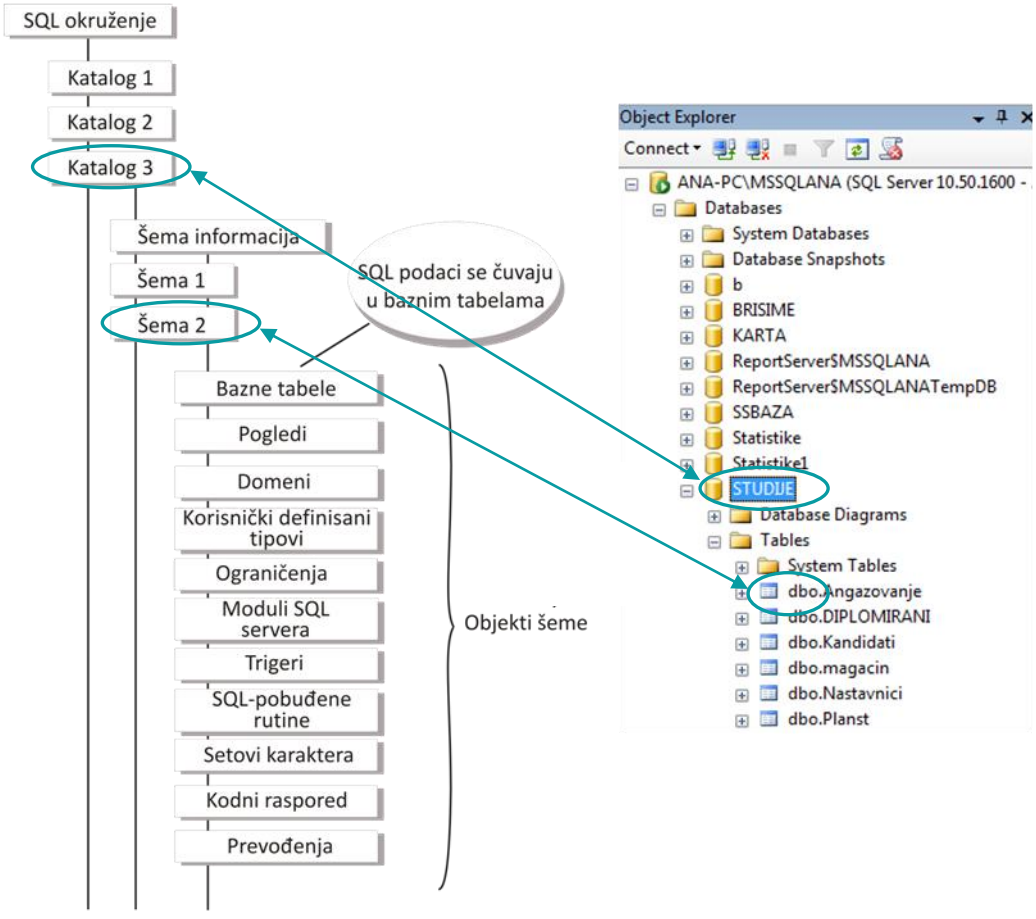
Za svaki objekat se za kojoj šemi pripada, kao i kom katalogu pripada šema, pa je logično što SQL standard podržava konvenciju definisanja trodelnog imena objekata u tekućem okruženju, tj. na tekućem server i to:

`[naziv_kataloga].[naziv_seme].[naziv_objekta]`

Ovakvo ime se naziva punim kvalifikovanim imenom objekta. U slučaju da je ime oblika

`[naziv_objekta]`

ono je nekvalifikovano i tada se uzima da objekat pripada podrazumevanoj šemi.
Svaki katalog sadrži jednu specifičnu šemu pod nazivom **INFORMATION_SCHEMA** (informaciona šema), koja predstavlja **rečnik podataka**. Nju čini skup pogleda nad sistemskim tabelama, koje sadrže sve bitne informacije o SQL okruženju.



SQL standard ne predviđa naredbe za kreiranje i uništavanje kataloga. Način njihovog kreiranja i uništavanja je implementaciono definisan, odnosno prepušten je vlasnicima softverskih proizvoda koji implementiraju SQL okruženje.

Kada je u pitanju MS SQL Server objekat koji odgovara katalogu jeste baza podataka (database objekat).

1.1 Kreiranje i izmena baze podataka

SQL2003 standard ne sadrži naredbe za kreiranje baze podataka i njenu izmenu. Komercijalni SUBP, ipak, imaju specifične naredbe **CREATE/ALTER DATABASE**.

Naredba **CREATE DATABASE** za SQL Server ima sledeći oblik:

```
CREATE DATABASE database_name
    [ ON    [ PRIMARY ] [ <filespec> [ ,...n ]
            [ , <filegroup> [ ,...n ] ]
            [ LOG ON { <filespec> [ ,...n ] } ] ]
    [ COLLATE collation_name ]
] [;]
```

Database_name
Ime baze podataka; mora da bude jedinstveno u okviru instance SQL servera.

ON
Služi za navođenje definisane datoteke na disku za smeštanje sekcija sa podacima baze podataka.

PRIMARY
Specificira da odgovarajuća <filespec> lista definiše primarnu datoteku.

Prva datoteka koja se specificira u <filespec> u primarnoj filegroup postaje primarna datoteka. Baza podataka može da ima samo jednu primarnu datoteku.

Datoteke su grupisane u jednu ili više grupa datoteka (filegroup). Obavezno je postojanje najmanje jedne grupe za podatke koja se naziva osnovna (PRIMARY) grupa, koja ima ekstenziju **mdf** i jedna grupa za dnevnik transakcija sa ekstenzijom **ldf**. Mogu da postoje sekundarne grupe koje se mogu koristiti i za podatke i za dnevnik sa ekstenzijom **ndf**.

LOG ON
Specificiranje datoteke na disku koja će se koristiti za smeštanje dnevnika transakcija baze podataka. Korišćenjem LOG ON klauzule dnevnik transakcija se može smesti na lokaciju koja je odvojena od podataka sa ciljem poboljšanja ulazno/izlaznih performansi. Ako se LOG ON ne specificira dnevnik transakcija će automatski biti kreiran sa veličinom koja iznosi jednu četvrtinu od zbira svih datoteka baze podataka sa podacima.

COLLATE collation_name
Specificira podrazumevani kodni raspored za bazu podataka. Collation_name može da bude Windows collation_name ili SQL collation_name. Ako se ne specificira ova klauzula podrazumeva se collation_name instance SQL Servera.

Primeri kreiranja

```
CREATE DATABASE STUDIJE
ON PRIMARY
      (NAME = PODACI1, FILENAME = 'C:\podaci\primarna.mdf', SIZE = 5,
      MAXSIZE = 20, FILEGROWTH = 5),
   FILEGROUP GRUPA2
      (NAME = PODACI2, FILENAME = 'C:\podaci\sekundarna1.ndf', SIZE = 5,
      MAXSIZE = 25, FILEGROWTH = 5)
   LOG ON
      (NAME = fakultet_log, FILENAME = 'C:\fak_dnevnik.ldf', SIZE = 5,
      MAXSIZE = 20, FILEGROWTH = 5)
```

Izmene u definiciji baze se postižu pozivom komande **ALTER DATABASE**.

Primeri:

1. Dodavanje nove datoteke za dnevnik.

```
ALTER DATABASE STUDIJE
ADD FILE
(NAME=PODACI3, FILENAME='C:\podaci\sekundarna2.ndf',
    SIZE = 5MB, MAXSIZE = 50MB, FILEGROWTH = 5MB)
TO FILEGROUP GRUPA2
```

2. Brisanje prethodno dodate datoteke za dnevnik.

```
ALTER DATABASE STUDIJE
REMOVE FILE PODACI3
```

1.2 Schema

Shema je kolekcija imenovanih objekata baze podataka, koja obezbeđuje logičku klasifikaciju tih objekata.

```
CREATE SCHEMA [schema_name] [AUTHORIZATION owner_name]
   [ ANSI CREATE statements [...] ]
   [ ANSI GRANT statements [...] ]
```

[schema_name]
Ako se izostavi, baza podataka će sama kreirati ime šeme korišćenjem imena korisnika koji poseduje šemu.

AUTHORIZATION owner_name
Postavlja owner_name kao korisnika koji poseduje šemu. Kada se izostavi, tekući korisnik se postavlja kao korisnik.

ANSI CREATE statements [...]
Sadrži jednu ili više naredbi za kreiranje.

ANSI GRANT statements [...]
Sadrži jednu ili više naredbi GRANT za prethodno definisane objekte.

```
CREATE SCHEMA NastOdr AUTHORIZATION Savic
    CREATE TABLE NastOdr.Nastavnici (Snast int not null, Imen char(25))
    CREATE VIEW NastOdr.NastPogl AS SELECT * FROM Nastavnici
    GRANT SELECT TO Panic;
```

Specifičnost uklanjanja šeme je u opciji RESTRICT

```
DROP SCHEMA ime-sheme RESTRICT
```

RESTRICT zabranjuje uklanjanje sheme ako ona sadrži bilo koji objekat baze podataka.

Default (podrazumevana) šema je šema koja se pretražuje kada se koriste nekvalifikovani objekti (objekti kojima nije navedeno puno ime), npr.

```
SELECT *
FROM Studenti
```

Podrazumevana šema može biti eksplicitno definisana za svakog korisnika pojedinačno, pri kreiranju ili promeni definicije korisnika (CREATE/ALTER USER).
MS SQL Server: Ako se ne naglasi drugačije dafult schema koja se definiše za svakog korisnika je **dbo**, pa prethodni upit ima isto značenje kao i

```
SELECT STUDIJE.dbo.*
FROM STUDIJE.dbo.Studenti
```

1.3 Kreiranje i izmena tabele

SQL održava tri tipa tabela:

- **bazne** tabele
  Mogu da se podele u dve kategorije:
  - **stalne** (persistent) i
  - **privremene** (temporary).

  Neke su objekti šeme, a neke su smeštene u modulima (npr. dobijenih prevođenjem stornih procedura).
  Sve modularne bazne tabele su privremene tabele.
- **izvedene** tabele - rezultati realizacije SQL upita nad bazom podataka,
- **pogledi (view) -** tip izvedenih tabela čija definicija se čuva u šemi.

SQL podržava četiri tipa baznih tabela:

- **Stalne bazne tabele**. Ove tabele predstavljaju imenovane objekte šeme koji su definisani naredbom `CREATE TABLE`.
- **Globalne privremene tabele**. Ove tabele predstavljaju imenovane objekte šeme koji su definisani naredbom `CREATE GLOBAL TEMPORARY TABLE`.
  Postoje u okviru konteksta jedne SQL sesije. Globalnim privremenim tabelama može da se pristupi svuda u okviru povezane SQL sesije.
- **Kreirane lokalne privremene tabele**. Ove tabele predstavljaju imenovane objekte šeme koji su definisani naredbom `CREATE LOCAL TEMPORARY TABLE`. Postoje u okviru konteksta jedne SQL sesije. Kreiranoj lokalnoj privremenoj tabeli može da se pristupi samo u okviru povezanog modula.
- **Deklarisane privremene tabele**. To su tabele deklarisane kao deo procedure u modulu (npr. storne procedure). Definicija tabele nije smeštena u šemi i ne može da postoji dok se procedura ne pozove. Kao i sve privremene tabele može da bude pozvana samo u okviru konteksta SQL sesije.

Osnovni oblik iskaza za kreiranje bazne tabele je

```
CREATE TABLE ime-bazne-tabele
      ( def-kolone {, def-kolone}
      [, def-prim-kljuca]
      [, def-str-kljuca {, def-str-kljuca}]
      [, uslov-ogranicenja {, uslov-ogranicenja})
      [ drugi-parametri]
```

Definicija kolone `def-kolone` ima oblik:

```
ime-kolone tip-podataka [NOT NULL] [[WITH] DEFAULT [vrednost]]
```

Neki tipovi podataka (T-SQL):

- bit, tinyint, smallint, int, bigint (bit, bajt, 2 bajta, 4 bajta, 8 bajtova)
- numeric, decimal
- float, real
- char, varchar, text
- nchar, nvarchar, ntext
- binary, varbinary, image

Opcija `[NOT NULL] [[WITH] DEFAULT [vrednost]]` odnosi se na kolonu koja ne dozvoljava NULL vrednosti, tj. na mogućnost zamene nedostajuće vrednosti navedenom (DEFAULT) vrednošću iz domena. Ako se vrednost u opciji DEFAULT ne navede, nedostajuća vrednost se sistemski zamenjuje podrazumevanom (DEFAULT) vrednošću za odgovarajući domen (na primer, blank za tip CHAR, 0 za tip INTEGER, itd). Opcija DEFAULT može da se koristi i samostalno (bez NOT NULL) i to u slučaju kolone koja dopušta NULL vrednosti; pri tome se kao podrazumevana vrednost može navesti i NULL.

```
ALTER TABLE table_name
  [ADD [COLUMN] column_name datatype attributes]
| [ALTER [COLUMN] column_name SET DEFAULT default_value]
| [ALTER [COLUMN] column_name DROP DEFAULT]
| [DROP [COLUMN] column_name {RESTRICT | CASCADE}]
| [WITH NOCHECK | CHECK] [ADD table_constraint]
| [DROP CONSTRAINT table_constraint_name {RESTRICT | CASCADE}]
```

1.4 Ograničenja

ANSI standard predviđa četiri tipa ograničenja:

- PRIMARY KEY,
- UNIQUE,
- FOREIGN KEY i
- CHECK.

U ograničenje spada i NOT NULL opcija koja se koristi u definiciji kolone i objašnjena je u prethodnoj sekciji.

Ograničenja mogu da se primene na:

- **nivou kolone**
  Deklarisane kao deo definicije kolone i primenjuju se samo na tu kolonu;
- **nivou tabele**
  Deklarisane nezavisno od bilo koje definicije kolone (uobičajeno, na kraju naredbe CREATE) i mogu da se primene na jednu ili više kolona u tabeli.

Pri izmeni definicije tabele dodavanjem ograničenja

```
ALTER TABLE table_name
 [WITH NOCHECK | CHECK] [ADD table_constraint]
```

opcijom CHECK se obezbeđuje provera zadovoljenosti ograničenja kod podataka koji se već nalaze u tabeli čija se definicija menja. Ukoliko se koristi opcija NOCHECK, provere nad postojećim podacima nema.

Ograničenje primarnog ključa

Definicija ograničenja **primarnog ključa** ima oblik:

```
[CONSTRAINT ime] PRIMARY KEY (ime-kolone {, ime-kolone})
```

Svaka kolona čije je ime navedeno u ovoj definiciji mora biti eksplicitno definisana kao NOT NULL.

```
CREATE TABLE Nastavnici
   (Snast SMALLINT NOT NULL,
    Imen CHAR(25) NOT NULL,
    CONSTRAINT pk_nast PRIMARY KEY CLUSTERED(Snast));
```

Primarni ključ koji se sastoji od jedne kolone može se zadati i u samoj definiciji te kolone, navođenjem opcije [CONSTRAINT ime] PRIMARY KEY.

```
CREATE TABLE Predmeti
(Spred SMALLINT PRIMARY KEY,
 Nazivp CHAR(25) NOT NULL);
```

Ograničenje spoljašnjeg ključa (foreign key constraint)

Definicija ograničenja **spoljašnjeg ključa** ima oblik:

```
[CONSTRAINT ime] FOREIGN KEY ( kolona {, kolona} )
REFERENCES referencirana-tabela [(referencirane-kolone[,...])]
  [ON DELETE {NO ACTION | CASCADE | SET NULL | SET DEFAULT}]
  [ON UPDATE {NO ACTION | CASCADE | SET NULL | SET DEFAULT}]
```

`referencirana-tabela`

je ***bazna*** tabela u kojoj ovde navedene kolone čine **primarni** ili **jedinstveni** ključ. Tabela u kojoj je definisan strani ključ je ***zavisna*** od referencirane tabele.

`[ON DELETE efekat] / [ON UPDATE efekat]`

Definiše akciju koju baza podataka izvodi sa ciljem održavanja (čuvanja) integriteta podataka spoljašnjeg ključa kada je vrednost ograničenja pozvanog primarnog ili jedinstvenog ključa promenjena ili izbrisana.

`NO ACTION`

Omogućuje brisanje/ažuriranje vrste u baznoj (referenciranoj) tabeli samo ako ne postoji vrsta u zavisnoj tabeli koja zavisi od vrste koja se briše/menja (tj. ako ne postoji vrsta u zavisnoj tabeli čija je vrednost stranog ključa jednaka vrednosti primarnog ključa vrste koja se briše). U suprotnom, sistem odbija da izvrši brisanje iz referencirane tabele i javlja grešku. Ovo je i podrazumevani efekat.

`CASCADE`

Omogućuje brisanje/izmenu naznačene vrste referencirane tabele i brisanje/izmenu svih zavisnih vrsta zavisne tabele.

`SET NULL`

Govori bazi podataka da postavi vrednost u spoljašnjem ključu na NULL kada je vrednost primarnog ključa ili jedinstvenog ključa promenjena ili izbrisana. Efekat SET NULL se ne može primeniti ako je neka od kolona stranog ključa NOT NULL kolona.

`SET DEFAULT`

Govori bazi podataka da postavi vrednost u spoljašnjem ključu na podrazumevanu (korišćenjem podrazumevanih vrednosti koje su specificirane za svaku kolonu) kada je vrednost primarnog ključa ili jedinstvenog ključa promenjena ili izbrisana.

Primer.

```
CREATE TABLE Prodavci
    (prod_id            CHAR(4)    NOT NULL,
     prod_ime           VARCHAR(40),
     prod_adresa1       VARCHAR(40),
     prod_adresa2       VARCHAR(40),
     grad               VARCHAR(20),
     država             CHAR(2),
```

```
    poštanski broj     CHAR(5),
    phone              CHAR(12),
    prodaja_rep        INT,
CONSTRAINT pk_prod _id  PRIMARY KEY (prod_id),
CONSTRAINT fk_zap_id   FOREIGN KEY (prodaja_rep)
   REFERENCES Zaposleni(zap_id));
```

UNIQUE ograničenja (UNIQUE constraints)

Ograničenjem UNIQUE (ograničenje kandidata za ključ) se deklariše da vrednosti atributa ili kolekcije atributa moraju biti jedinstvene.
SQL2003 dozvoljava zamenu liste kolona, prikazanu u opštem dijagramu sintakse za ograničenja, sa ključnom reči (VALUE). UNIQUE (VALUE) označava da su sve kolone u tabeli deo jedinstvenog ključa.

CHECK ograničenja (CHECK Constraints)

Sintaksa opcije zadavanja ograničenja uslovom je

```
[CONSTRAINT ime] CHECK (uslov)
```

Ograničenje uslovom koje se odnosi samo na jednu kolonu tabele može se zadati i u definiciji te kolone, istom sintaksom.
CHECK ograničenje se smatra odgovarajućim kada se uslov pretraživanja sračunava u **TRUE ili UNKNOWN**.

Primer.

```
CREATE TABLE Planst (
Ssmer SMALLINT REFERENCES Smer(Ssmer),
Spred SMALLINT REFERENCES Predmeti(Spred),
Semestar TINYINT,
CHECK(Semestar IN ('1','2','3','4','5','6','7','8','9','10')));
```

1.5 Indeksi

Indeksi su posebne tabele za pretraživanje koje odgovarajući mehanizam RDBMS-a može da koristi za povećanje brzine u rukovanju podacima. Glavna razlika u odnosu na tabele je to što su indeksi skriveni od korisnika i ne pojavljuju se ni u jednoj DML naredbi.
Naredba CREATE INDEX nije deo ANSI standarda i zato značajno varira među proizvođačima baza podataka.
Uobičajena sintaksa je:

```
CREATE [UNIQUE] INDEX ime_indeksa ON ime_tabele
(ime_kolone [, ...])
```

S obzirom da se indeksi vezuju za tabelu, ime indeksa treba da bude jedinstveno u okviru tabele na koju se odnosi. Ključna reč UNIQUE definiše indeks kao jedinstveno ograničenje za tabelu i onemogućava duple vrednosti kolone ili kolona u tabeli.
Neki RDBMS-vi (MS SQL Server npr.) omogućavaju kreiranje indeksa nad pogledima.
Indeksi posebno ubrzavaju operacije manipulacije podacima nad tabelama sa WHERE i JOIN klauzulama.

Primer.

```
CREATE INDEX dipl_ind ON Diplomirani(Indeks, Upisan)
```

1.6 Pogledi

Pogledi su virtuelne tabele koje nisu predstavljene fizičkim podacima, već je njihova definicija zapamćena u katalogu.
Neke platforme baza podataka podržavaju **materijalizovani pogled**; to jest, fizički kreiranu tabelu koja je definisana upitom slično pogledu.

```
CREATE VIEW ime_pogleda [(column [,...])]
AS
select_naredba [WITH [CASCADED | LOCAL] CHECK OPTION]
```

ANSI standard definiše da mora da se koristi lista kolona u delu **[(column [,...])]** ili klauzula AS uz **select_naredba**. Neki proizvođači SUBP omogućavaju više fleksibilnosti, tako da treba slediti ova pravila kada se koristi klauzula AS:

- Kada naredba SELECT sadrži izračunate kolone, takve kao (prodaja * 1.04)
- Kada naredba SELECT sadrži puna kvalifikovana imena kolona, takva kao studije.studenti.mesto.
- Kada naredba SELECT sadrži isto ime za više od jedne kolone (iako sa posebnom šemom ili prefiksima baze podataka)

Pogled može biti objekat delovanja UPDATE i DELETE komade, tj. moguće je ažuriranje podataka preko pogleda. Prema ANSI standardu pogled može da ažurira baznu tabelu(e) na kojima se zasniva ako ispunjava sledeće uslove:

- Pogled ne sadrži operatore UNION, EXCEPT ili INTERSECT
- Definisana naredba SELECT ne sadrži klauzule GROUP BY ili HAVING
- Definisana naredba SELECT ne sadrži klauzulu DISTINCT
- Pogled nije materijalizovan

Najznačajnije pravilo za koga se treba setiti kada se ažurira bazna tabela preko pogleda je da sve kolone u tabeli koje su definisane kao NOT NULL moraju da prime ne nula vrednost kada dobijaju novu ili menjaju vrednost. Ovo može da se čini eksplicitno direktnim upisivanjem ili ažuriranjem not null vrednosti u koloni ili oslanjanjem na podrazumevanu vrednost. Osim toga, pogledi ne skidaju ograničenja na osnovnoj tabeli. Tako, vrednosti koje se upisuju ili ažuriraju u osnovnoj tabeli moraju da ispunjavaju sva ograničenja postavljena na njima sa jedinstvenim indeksima, primarnim ključevima itd.

2. DML – manipulacija podacima

2.1 Upiti nad bazom podataka

Upiti predstavljaju različite oblike naredbe SELECT i mogu da budu različite složenosti.

Skraćena verzija naredbe SELECT:

```
SELECT [{ALL | DISTINCT}] select_item [AS alias] [,...]
FROM
   {    table_name  [[AS] alias]
        | view_name [[AS] alias]} [,...]
        [ [join_type] JOIN join_condition ]
[WHERE search_condition] [ {AND | OR | NOT} search_condition [...] ]
[GROUP BY group_by_expression{group_by_columns}
[HAVING search_condition] ]
[ORDER BY {order_expression [ASC | DESC]} [,...] ]
```

2.1.1 Upiti nad bazom podataka

Poglavlje 7.pdf izuzev sekcije koja se odnosi na RollUp

2.1.2. Korišćenje predikata

Poglavlje 9.pdf

2.2 Izmena podataka u bazi podataka

SQL obezbeđuje tri naredbe za rukovanje podacima: **INSERT**, **UPDATE** i **DELETE**.

INSERT

Naredba INSERT omogućava upisivanje vrsta u tabeli korišćenjem jednog od nekoliko načina:

- Upisivanjem jedne ili više vrsta korišćenjem DEFAULT vrednosti koja je specificirana za kolonu preko naredbi CREATE TABLE ili ALTER TABLE.
- Drugi način i uobičajeniji način je navodjenje vrednosti koje će biti upisane u kao vrednosti atributa u zapisu (record-u).

```
INSERT INTO {table_name | view_name} [(column1 [,...] )]
{DEFAULT VALUES | VALUES (value1 [,...]) | SELECT_statement }
```

Kod oblika

```
INSERT INTO ime-tabele [( ime-kolone {,ime-kolone})]
VALUES ( konstanta {,konstanta})
```

u tabelu se upisuje pojedinačna vrsta sa navedenim konstantnim vrednostima u navedenim kolonama. Broj konstanti mora biti jednak broju kolona, a pridruživanje se vrši u navedenom redosledu. U kolone koje nisu navedene unose se NULL (odnosno podrazumevane) vrednosti ako kolona nije definisana kao NOT NULL kolona;

- Treći način, koji puni tabelu sa više zapisa odvojeno, je upisivanje rezultujućeg skupa naredbe SELECT.

  Kod oblika

  ```
  INSERT INTO ime-tabele [( ime-kolone {,ime-kolone})]
  puni upitni blok
  ```

  u tabelu se upisuje jedna ili više vrsta, koje predstavljaju rezultat izvršavanja punog upitnog bloka; pri tom i-ta kolona rezultata izvršavanja punog upitnog bloka odgovara i-toj koloni u zapisu iskaza.

Primeri.

```
INSERT INTO studenti VALUES (99,2000,'Stevan',NULL,NULL,NULL)

INSERT INTO studenti (Indeks, Upisan, Imes) VALUES (99,2000,'Stevan')

INSERT INTO studenti (Upisan,Imes,Indeks) VALUES (2000, 'Stevan', 99)

INSERT INTO studenti (indeks) VALUES (101)

INSERT INTO Studenti (Indeks, Upisan, Imes, Mesto, Datr, Ssmer)
SELECT Indeks, Upisan, Imes, Mesto, Datr, Ssmer
FROM Kandidati
WHERE Položen = 1
```

UPDATE

Naredba UPDATE menja postojeće podatke u tabeli.

```
UPDATE {table_name | view_name}
SET {column_name = { DEFAULT | NULL | scalar_expression}
      [,...] }
 [ WHERE search_condition ]
```

Primeri.

```
UPDATE Prijave SET Ocena = Ocena + 1
WHERE Indeks = 99 AND Upisan = 2000 AND Spred = 17

UPDATE Studenti SET Mesto = ( SELECT Mesto
                              FROM Studenti
                              WHERE Imes = 'Sima')
WHERE Imes = 'Stevan' AND Indeks = 99;
```

DELETE

Naredba DELETE briše slogove iz tabele ili pogleda.

```
DELETE FROM { table_name|view _name | ONLY (table_name|view_name) }
```

Primeri.

```
DELETE FROM Kandidati;
```

```
DELETE FROM STUDENTI WHERE Indeks = 40 AND Upisan = 2003;
```

4. UPRAVLJANJE TRANSAKCIJAMA

Transakcija je niz operacija nad bazom podataka koji odgovara jednoj logičkoj jedinici posla u realnom sistemu.

```
Učitaj iznosp za prenos;
Nađi račun R1 sa koga se iznosp skida;
Upiši iznosR1 - iznosp na račun R1;
Nađi račun R2 na koga se iznosp stavlja;
Upiši iznosR2 + iznosp na račun R2.
```

Transakcija u izvršenju mora da ima tzv. ACID osobine (po početnim slovima sledećih engleskih reči):

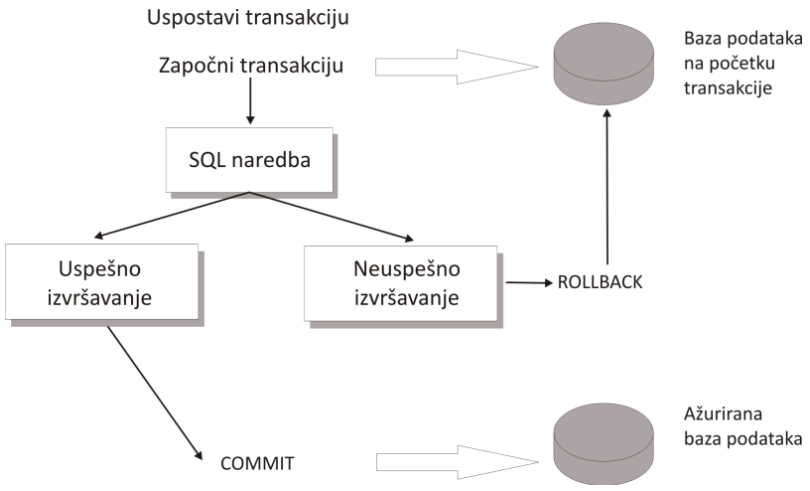
- **Atomičnost (Atomicity)**. Zahteva se da se sve operacije nad bazom podataka predviđene transakcijom uspešno obave ili da se ne prihvati nijedna, tj. ako se bar jedna operacija ne obavi kako je očekivano, onda se poništavaju efekti i svih ostalih.
  Da bi se ostvarila atomičnost transakcije definišu se dve specifične operacije nad bazom podataka:
  - COMMIT koja označava uspešan kraj transakcije i koja "potvrđuje" sve promene u bazi koje je posmatrana transakcija proizvela;
  - ROLLBACK kojom se poništavaju efekti svih prethodnih operacija nad bazom podataka u jednoj transakciji, ako ona, zbog predviđene ili nepredviđene greške (otkaza sistema) može da dovede bazu podataka u nekonzistentno stanje.
- **Konzistentnost (Consistency)**. Očigledno je da se transakcija može definisati i kao "jedinica konzistentnosti" baze podataka: pre početka i posle okončanja transakcije stanje baze podataka mora da zadovolji uslove konzistentnosti. Za vreme obavljanja transakcije konzistentnost baze podataka može da bude narušena.
- **Izolacija (Isolation)**. Kada se dve ili više transakcija izvršavaju istovremeno, njihovi efekti moraju biti međusobno izolovani. Drugim rečima efekti koje izazovu transakcije koje se obavljaju istovremeno moraju biti jednaki efektima nekog njihovog serijskog (jedna posle druge) izvršenja.
- **Trajnost (Durability)**. Kada se transakcija završi njeni efekti ne mogu biti izgubljeni, čak i ako se neposredno po njenom okončanju desi neki ozbiljan otkaz sistema.

KOMANDE

SQL obezbeđuje sledeće naredbe za upravljanje transakcijama:

- `SET TRANSACTION`,
- `START TRANSACTION (BEGIN TRANSACTION)`,
- `SAVEPOINT`,
- `ROLLBACK` i
- `COMMIT`.

Većina platformi baza podataka sprovodi **implicitnu kontrolu transakcija**, korišćenjem takozvanog **autocommit** mehanizma. U autocommit modu, baza podataka tretira svaku naredbu kao transakciju kako unutar tako i izvan nje, upotpunjenu implicitnom naredbom BEGIN TRAN i COMMIT TRAN. Pod **eksplicitnom kontrolom transakcije**, deklariše se svaka nova transakcija naredbom `START TRANSACTION`. Transakcija nije potvrđena ili poništena sve dok se eksplicitno izdaju naredbe bilo `COMMIT` ili `ROLLBACK`.



SET TRANSACTION

Naredba SET TRANSACTION kontroliše karakteristike promene podataka, najpre read/write karakteristike i nivo izolacije (izdvajanja) transakcija. Karakteristike transakcije koje nisu navedene će preuzeti podrazumevane vrednosti.

```
SET TRANSACTION  [READ ONLY | READ WRITE]
   [ISOLATION LEVEL {READ COMMITTED | READ UNCOMMITTED |
    REPEATABLE READ | SERIALIZABLE}]
```

READ ONLY

Postavlja sledeću dolazeću transakciju kao read-only transakciju. Ako se koristi ova opcija, u transakciju ne mogu da se uključe naredbe UPDATE ili CREATE TABLE, koje menjaju podatke, odnosno strukturu baze podataka. Pošto je transakcija završena, ponašanje transakcije se vraća na podrazumevana postavljenja.

READ WRITE

Postavlja sledeću dolazeću transakciju tako da ona može da izvršava operacije koje čitaju i menjaju podatke.

ISOLATION LEVEL

Postavlja izolacioni nivo za sledeću transakciju u sesiji. Izolacioni nivo definiše kako će se transakcija izolovati od akcija drugih transakcija.

Izolacioni nivo transakcija specificira koji stepen izolacije od drugih sesija koje rade istovremeno transakcija ima. Nivo izolacije kontroliše:

- **Da li su vrste koje su pročitane i ažurirane vašom sesijom baze podataka raspoložive drugim konkurentnim sesijama baze podataka koje se izvode**
- **Da li aktivnosti ažuriranja, čitanja i upisivanja drugih sesija baze podataka mogu da utiču na vašu sesiju baze podataka.**

Naredba SET TRANSACTION podržava četiri izolaciona nivoa:

- **READ COMMITTED**
  Omogućava transakciji da čita vrste upisane od drugih transakcija samo kada su one bile potvrđene.
- **READ UNCOMMITTED**
  Omogućava transakciji da čita vrste koje su bile upisane, ali nisu bile potvrđene od drugih transakcija.
- **SERIALIZABLE**
  Sve sesije mogu da vide vrste koje su potvrđene pre nego što su njihove transakcije počele. Druge otvorene sesije mogu da vide, ali ne mogu da upisuju ili menjaju vrste unutar drugih korisničkih sesija dok se ove transakcije ne završe.
  Ovo je najveći ograničavajući nivo (i podrazumevan) za SQL2003.

Kada se izda, SET TRANSACTION postavlja svojstva sledeće dolazeće transakcije. Dakle, **SET TRANSACTION je privremena naredba, koju bi trebalo izdavati posle jedne završene transakcije, a pre nego što sledeća transakcija počne**.

START TRANSACTION (nema je u T-SQL)

Naredba START TRANSACTION započinje izvršavanje transakcije, a pri tome može da uradi isto što i SET TRANSACTION.

```
START TRANSACTION [READ ONLY | READ WRITE]
[ISOLATION LEVEL {READ COMMITTED | READ UNCOMMITTED |
REPEATABLE READ | SERIALIZABLE}
```

BEGIN TRANSACTION (T-SQL)

BEGIN TRANSACTION deklariše eksplicitnu transakciju, ali ne postavlja izolacione nivoe.

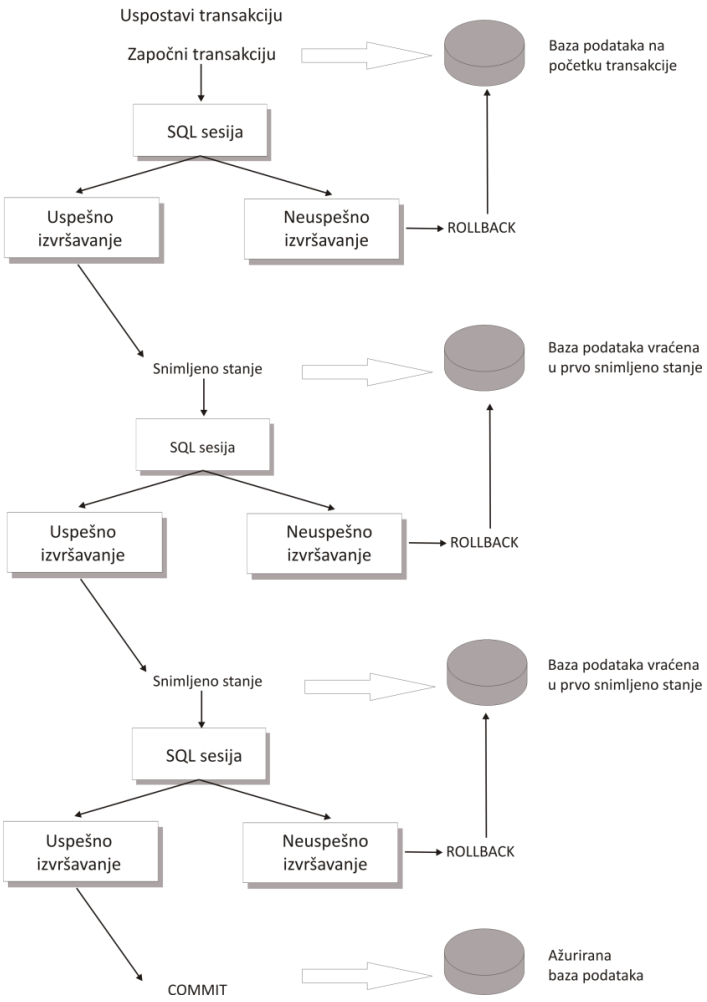
```
BEGIN { TRAN | TRANSACTION }
    [ { transaction_name | @tran_name_variable }
      [ WITH MARK [ 'description' ] ] ]
[ ; ]
```

COMMIT

Naredba COMMIT eksplicitno završava otvorenu transakciju i čini promene stalnim u bazi podataka. Transakcije mogu da budu otvorene implicitno kao deo naredbe INSERT, UPDATE ili DELETE, ili otvorene eksplicitno naredbom START. U oba slučaja, eksplicitno izdavanje naredbe COMMIT će da završi otvorenu transakciju.

SQL Server sintaksa:

```
COMMIT
{TRAN|TRANSACTION}
[transaction_name|@tran_
name_variable][;]
```



Svaka transakcija koja je eksplicitno deklarisana može da bude učinjena stalnom samo izvršavanjem naredbe COMMIT. Slično, bilo koja transakcija koja je neuspešna ili je potrebno da bude odbačena mora da bude eksplicitno poništena naredbom ROLLBACK.

Napomena: proverite da je START u paru sa bilo COMMIT ili ROLLBACK. Inače, DBMS neće moći da završi transakciju(e) dok ne naiđe na COMMIT ili ROLLBACK.

SAVEPOINT

Ova naredba prekida transakciju u logičkim tačkama prekida. U okviru jedne transakcije može biti specificirano više tačaka čuvanja. Glavna korist od naredbe SAVEPOINT jeste ta da efekti komandi

unutar transakcije mogu da budu parcijalno povraćeni do označene tačke čuvanja korišćenjem naredbe ROLLBACK.

T-SQL sintaksa:

```
SAVE  { TRAN | TRANSACTION } { savepoint_name | @savepoint_variable }
[ ; ]
```

ROLLBACK

Naredba ROLLBACK poništava transakcije do njihovog početka ili do prethodno definisane SAVEPOINT tačke. ROLLBACK takođe zatvara sve otvorene kursore.

```
ROLLBACK { TRAN | TRANSACTION }
[ transaction_name | @tran_name_variable
 | savepoint_name | @savepoint_variable ]
[ ; ]
```

Primer 1.

```
BEGIN TRANSACTION prodaja
INSERT INTO Racun VALUES (1,' ',111)
IF @@ERROR != 0
BEGIN
   ROLLBACK TRANSACTION prodaja
   PRINT 'Nije dozvoljeno upisivanje u RACUN! Transakcija je poništena'
   RETURN
END
UPDATE MAGACIN SET Stanje = Stanje - 10 WHERE Proizvod = 100
IF @@ERROR != 0
BEGIN
   ROLLBACK TRANSACTION prodaja
   PRINT 'Nije dozvoljeno upisivanje u MAGACIN! Transakcija je poništena'
   RETURN
END
INSERT INTO STAVKA_RACUNA VALUES (1, 100, 10)
IF @@ERROR != 0
BEGIN
   ROLLBACK TRANSACTION prodaja
   PRINT 'Nije dozvoljeno upisivanje u STAVKA_RACUNA! Transakcija je
poništena'
   RETURN
END
COMMIT TRANSACTION prodaja
```

Primer2.

```
BEGIN TRANSACTION t1
INSERT INTO magacin VALUES(100, 'proizvod1', 100.50, 40)
IF @@ERROR <> 0 ROLLBACK
save transaction tr1
INSERT INTO magacin VALUES(101, 'proizvod2', 110.50, 50);
IF @@ERROR <> 0 ROLLBACK TRANSACTION tr1
save transaction tr2
INSERT INTO magacin VALUES(102, 'proizvod3', 120.50, 50)
IF @@ERROR <> 0 ROLLBACK TRANSACTION tr2
COMMIT TRANSACTION t1
```

5. SQL RUTINE

SQL rutine su objekti baza podataka koji su čuvani i kontrolisani od strane SUBPa i:

- sastoje se iz blokova proceduralnog koda,
- moraju biti eksplicitno pozvane da bi bile izvršene.

U SQL rutine spadaju: **funkcije** i **procedure**.

Funkcije se često nazvaju **korisnički definisane funkcije** (user-defined function - **UDF**), a procedure se nazivaju **uskladištenim procedurama** (**stored procedures**). I funkcije i procedure su memorisani skupovi predefinisanih SQL naredbi, koje izvršavaju određeni skup akcija nad bazom podataka. Kada su ove rutine memorisane, mogu da se pozovu kada je to potrebno odgovarajućom naredbom i uz navođenje parametara.

Opšti oblik procedure izgleda ovako:

```
CREATE PROCEDURE/FUNCTION ime procedure/funkcije [parametri]
BEGIN   -- početak bloka
DECLARE promenljive
SET   -- postavljanje inicijalnih vrednosti
-- naredbe
-- DML naredbe
-- Rad u petlji
-- Uslovne naredbe
.........
END  -- kraj bloka ;
```

Jedna procedura može da sadrži više blokova. Blokovi mogu da budu ugnježđeni.

Kreiranje procedura/funkcija (T-SQL sintaksa)

Kreiranje procedura

```
CREATE { PROC|PROCEDURE } [schema_name.]procedure_name
    [ { @parameter [ type_schema_name. ] data_type }
        [ = default ] [ OUT | OUTPUT ] [READONLY]
    ] [ ,...n ]
[ WITH <procedure_option> [ ,...n ] ]
AS { [ BEGIN ] sql_statement [;] [ ...n ] [ END ] } [;]
```

T-SQL nudi nekoliko donekle različitih šema za f- je:

- skalarne funkcije

```
CREATE FUNCTION [ schema_name. ] function_name
( [ { @parameter_name [ AS ][ type_schema_name. ] data_type
    [ = default ] [ READONLY ] }  [ ,...n ] ])
RETURNS return_data_type
    [ WITH <function_option> [ ,...n ] ]
    [ AS ]
    BEGIN
        function_body
        RETURN scalar_expression
    END [ ; ]
```

- inline table-valued

```
CREATE FUNCTION [ schema_name. ] function_name
( [ { @parameter_name [ AS ] [ type_schema_name. ]
parameter_data_type
    [ = default ] [ READONLY ] } [ ,...n ] ]
)
RETURNS TABLE
    [ WITH <function_option> [ ,...n ] ]
    [ AS ]
    RETURN [ ( ] select_stmt [ ) ]
[ ; ]
```

- multi-statement table-valued

```
CREATE FUNCTION [ schema_name. ] function_name
( [ { @parameter_name [ AS ] [ type_schema_name. ]
parameter_data_type
    [ = default ] [READONLY] }  [ ,...n ] ]
)
RETURNS @return_variable TABLE <table_type_definition>
    [ WITH <function_option> [ ,...n ] ]
    [ AS ]
    BEGIN
        function_body
        RETURN
    END
[ ; ]
```

Pozivanje/izvršavanje funkcija/procedura

Procedure se mogu izvršavati samo eksplicitnim pozivanjem.

```
    CREATE PROCEDURE novo_angazovanje (@nastavnik SMALLINT, @predmet
    SMALLINT, @smer SMALLINT)
    AS
    DECLARE @NAST VARCHAR(7)
    SELECT @NAST = Snast FROM Angazovanje WHERE Spred = @predmet AND
    ssmer = @smer
    IF @NAST IS NULL
    INSERT INTO Angazovanje VALUES(@nastavnik, @predmet, @smer)
    ELSE
    SELECT 'Angazovanje vec postoji';
EXECUTE dbo.novo_angazovanje @nastavnik =1, @predmet =1, @smer = 1
EXECUTE dbo.novo_angazovanje 1, 19, 12

    CREATE FUNCTION [dbo].plan_na_smeru(@smer SMALLINT)
    RETURNS TABLE
    AS
    RETURN (SELECT * FROM Planst WHERE Ssmer = @smer)
SELECT * FROM plan_na_smeru(3)

    CREATE FUNCTION [dbo].[provera_usl_predmet] (@INDEKS INT, @UPISAN INT, @Spred INT)
    RETURNS VARCHAR(3)
```

```
AS
BEGIN
DECLARE @Uslovni_predmet INT, @P VARCHAR(3), @Polozen INT
SELECT @Uslovni_predmet = UslPredmet FROM Uslovni WHERE USLOVNI.Spred = @Spred
SELECT @Polozen = Ocena FROM PRIJAVE WHERE PRIJAVE.Spred = @Uslovni_predmet AND
Indeks = @INDEKS AND Upisan = @UPISAN AND Ocena > 5
IF @Uslovni_predmet IS NULL  SET @P = 'OK'
  ELSE
            IF (@Uslovni_predmet > 0 AND @Polozen > 5)  SET @P = 'OK'
                  ELSE   SET @P = 'NOT';
RETURN @P
END
SELECT dbo.provera_usl_predmet(2,2002,7)
```

Promenljive

Promenljive sadrže vrednosti u koje se prenose učitane vrednosti iz baze podataka, ili su u njima vrednosti koje se predaju bazi podataka.
Promenljiva može biti deklarisana kao:

- parameter procedure/funkcije

  Kao parametar procedure deklariše se ovako

  ```
  { @parameter [ type_schema_name. ] data_type }
          [ = default ] [ OUT | OUTPUT ] [READONLY]
  ```

  a kao parameter funkcije ovako

  ```
  { @parameter_name [ AS ] [ type_schema_name. ] parameter_data_type
     [ = default ] [READONLY] }
  ```

  Parametar označen sa OUT ili OUTPUT je izlazni parametar, pa se njemu dodeljena vrednost unutar procedure ili procedure može videti i nakon poziva same procedure, a unutar koda u kom je procedura pozvana.
  [= default] je opcija kojom se navodi podrazumevana vrednost parametra ukoliko mu ona nije dodeljena pri pozivu
  [READONLY] je opcija kojom se navodi da parameter ne može menjati vrednost unutar funkcije/procedure.

- lokalne promenljive rutine
  Tada se deklarišu sa

  DECLARE <ime promenljive > <tip podataka > ;

  a vrednost im se dodeljuje ovako:

  SET < ime promenljive> = <value expression> ;

  value expression može da bude neki tip podataka kao NUMBER ili CHAR ili **podupit**.

Naredbe za kontrolu toka

Naredba IF...THEN...ELSE...END IF

Primer:

```
IF ime = 'Nenad'  THEN
            UPDATE Studenti
            SET Ime = 'Nenad'
```

```
                WHERE Indeks =  20 AND Upisan = 2000;
                      ELSE
                DELETE FROM Studenti
                WHERE = Indeks =  20 AND Upisan = 2000;
      END IF
```

Naredba CASE...END CASE

Naredba CASE se pojavljuje u dva oblika:

- jednostavna naredba CASE

```
CASE smerovi
WHEN 'Teorijska matematika'
      THEN INSERT INTO VALUES (1, 'Teorijska matematika')
ELSE Poruka
END CASE
```

- CASE naredba uz upotrebu složenijih uslova

```
CASE
WHEN šifra IN (1,2,3)
      THEN INSERT INTO Profili VALUES (1, 'Matematičar')
WHEN naziv IN (4,5)
      THEN INSERT INTO Profili VALUES (1, 'Informatičar')
ELSE Poruka
END CASE
```

Naredbe WHILE...DO...END WHILE i REPEAT...UNTIL...END REPEAT

Naredba WHILE...DO...END WHILE kao i REPEAT...UNTILL...END REPEAT imaju uobičajenu upotrebu. Unutar bloka je moguće koristi BREAK i CONTINUE (nisu SQL standard) sa uobičajenim značenjem.

```
SET brojac = 0;
WHILE brojac < 1000 DO
SET brojac = brojac + 1;
INSERT INTO Izvestaj (Redni_broj) VALUES (brojac);
END WHILE

SET brojac = 0;
REPEAT
SET brojac = brojac + 1;
INSERT INTO Izveštaj (Redni_broj) VALUES (brojac);
UNTIL brojac = 1000 END REPEAT
```

Naredba FOR...DO...END FOR

Naredba FOR za rad u petlji, deklariše i otvara **kursor**, izvršava telo naredbe FOR po jednom za svaku vrstu i onda zatvara kursor. Na primer:

```
FOR dopuna_mesta AS kurs_izveštaj
      CURSOR FOR  SELECT Redni_broj FROM Izveštaj
DO
UPDATE Izveštaj SET Opis = 'Komentar'
WHERE CURRENT OF kurs_izveštaj;
END FOR
```

6. KURSORI

"Izvršavanje SQL upita (interaktivnog ili aplikativnog) proizvodi uvek tabelu, koja u opštem slučaju sadrži veći broj redova. U slučaju aplikativnog SQL-a, u kome se rezultat izvršavanja upita predaje programu na dalju obradu, ta skupovna priroda rezultata SQL upita je u suprotnosti sa "slogovnom" prirodom programskih jezika (oni mogu da obrađuju samo po jedan slog, red ili podatak u jednom trenutku). Mehanizam kojim se premošćuje ova razlika između programske i upitne (slogovne i skupovne) komponente aplikativnog SQL-a zove se **kursor**. Kursor je oblik pokazivača na vrstu u rezultujućoj tabeli, koji može da prelazi preko svake vrste, ostvarujući njihovu programsku obradu jednu po jednu. Ukoliko rezultat izvršavanja upita jeste tabela sa jednom vrstom, kursor nije potreban; tada se u SELECT iskaz uvodi još jedan red (INTO linija) kojim se navodi u koje se programske promenljive (ili u koju se programsku strukturu) smeštaju komponente jedine vrste tabele – rezultata" [3].

Kursori se mogu koristiti za čitanje, ali i za ažuriranje podataka kojima se preko njih pristupa. Iz tog razloga, na aktivnosti sa podacima sa kojima je kursor vezan utiče mehanizam zaključavanja podataka, tj. nalaze se pod nadzorom komponente (RDBMS-a) za kontrolu konkurentnosti.
Osnovne komande za rad sa kursorima su:

- **DECLARE CURSOR** deklariše kursor definisanjem njegovog imena, karakteristika i upita koji se poziva pri otvaranju kursora.
  Kursor u suštini predstavlja posebnu strukturu (smeštenu u procesnom prostoru samog RDBMS-a) koja, pored karakteristika i upita, sadrži pointer na jednu (tekuću) vrstu skupa sa kojim je kursor povezan. Pozivom DECLARE CURSOR komande se obezbeđuje prostor za kursorsku strukturu.
  Deo T-SQL sintakse
  ```
  DECLARE cursor_name CURSOR
        FOR select_statement
    [;]
  ```
  Ime kursora mora biti jedinstveno u kontekstu u kome je definisan kursor – na primer, u bazi podataka ili šemi u kojoj je kreiran.
- Naredba **OPEN** otvara kursor, tj. izvršava naredbu SELECT i omogućava da rezultat bude raspoloživ za naredbu FETCH, tj. za upotrebu.
- Naredba **FETCH** uzima podatke iz skupa koji predstavlja rezultat izvršenog upita i smešta ih u promenljive koje predaju podatke jeziku domaćinu ili drugim SQL naredbama.
  Deo T-SQL sintakse
  ```
  FETCH
        [[NEXT|PRIOR|FIRST|LAST]FROM]
                        {cursor_name | @cursor_variable_name}
  [ INTO @variable_name [ ,...n ] ]
  ```

- Naredba CLOSE zatvara kursor, tj. oslobađa blokove podataka sa kojima je kursor povezan, kao i lokove. Na zatvoreni kursor se može ponovo primeniti komanda OPEN.

Sve četiri naredbe se pozivaju iz jezika domaćina.
T-SQL predviđa još jednu komandu i to DEALLOCATE.

Primer. T-SQL

```
DECLARE generacija_2000 CURSOR FOR
  SELECT Indeks, Upisan
  FROM dbo.Studenti
  WHERE Upisan = 2000
OPEN generacija_2000
FETCH NEXT FROM generacija_2000
WHILE @@FETCH_STATUS = 0
BEGIN
  FETCH NEXT FROM generacija_2000
END
CLOSE generacija_2000
DEALLOCATE generacija_2000
```

Sistemska promenljiva **@@FETCH_STATUS**, vraća status poslednje tekuće **FETCH** naredbe (0 označava da je naredba uspešno izvršena, -1 da je naredba neuspešno završena ili da je vrsta izvan rezultujućeg skupa, -2 da je očitana vrsta izgubljena)

DEALLOCATE uklanja kursorsku strukturu.

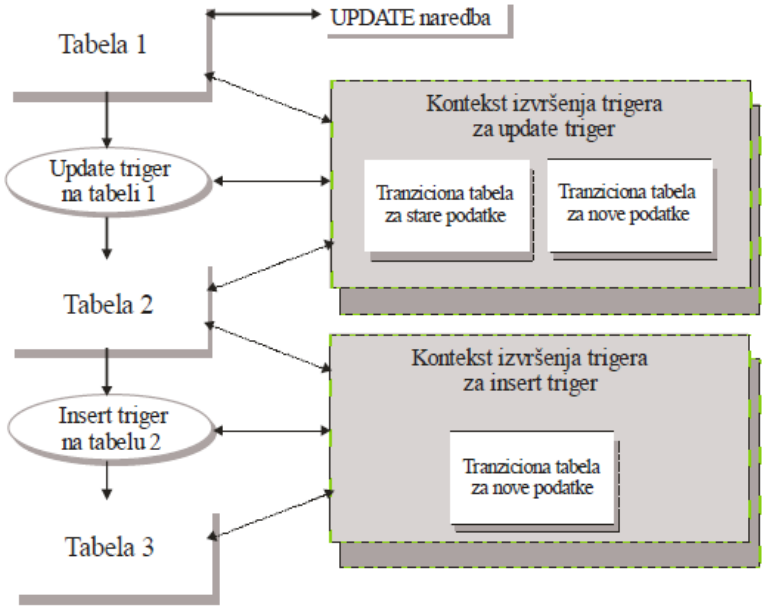
7. TRIGERI

Triger je objekat baze podataka koji predstavlja specijalnu vrstu uskladištene procedure koja se automatski pokreće (okida) pri modifikaciji podataka u okviru tabele sa kojom je povezan. U vezi sa tim postoje tri vrste trigera vezanih za DML komande (ostalim, vezanim za DDL i slično se ovde nećemo baviti, jer su specifični za SUBP): INSERT, UPDATE i DELETE. Kao objekat baze podataka jedan triger je povezan tabelom ili pogledom (što ima smisla ukoliko se preko pogleda obavljaju komande kojima se menjaju podaci u tabelama koje učestvuju u pogledu).

Kod trigera razlikujemo tri komponente:

- **događaj**, koji predstavlja upravo određenu modifikaciju,
- **uslov** koji predstavlja proveru određenog uslova da bi triger bio pokrenut. Rezultat provere uslova je TRUE ili FALSE.
- Ako je uslov ispunjen pokreće se automatski **akcija**.

Baza podataka koja ima obezbeđen mehanizam definisanja trigera se naziva **aktivna baza podataka**.



Slika 14-1. Konteksti izvršavanja trigera za dva trigera

Triger se izvršava u svom **izvođačkom kontekstu**. Ovaj kontekst čini memorijski prostor koji sadrži procese naredbi koje triger pokreće. Izvođački kontekst trigera se kreira uvek kada se triger pozove. Ako je pozvano više trigera izvođački kontekst se kreira za svakog od njih. U jednom trenutku je aktivan samo jedan izvođački kontekst. Ovo je važno kada triger u jednoj tabeli uzrokuje pokretanje trigera u drugoj tabeli.

U primeru na slici postoje tri tabele. U tabeli 1 je definisan UPDATE trigger, a u tabeli 2 INSERT triger. Kada se izvrši naredba UPDATE nad tabelom 1, pokreće se UPDATE triger, koji kreira svoj izvođački kontekst koji je aktivan. UPDATE triger koji je definisan da ažurira podatke u tabeli 2, poziva INSERT triger na tabeli 2, kada prvi triger pokuša da upiše podatke u ovoj tabeli. Zato se sada kreira drugi izvođački kontekst koji je aktivan. Kada drugi triger završi svoje izvođenje, drugi izvođački kontekst se briše i postaje aktivan prvi izvođački kontekst. Kada se prvi triger završi, briše se njegov izvođački kontekst.

Izvođački kontekst trigera sadrži informacije koje su potrebne za korektno izvršavanje trigera. Ove informacije uključuju elemente o samom trigeru, kao i o tabeli nad kojom je definisan. Izvođački kontekst uključuje i jednu ili dve tabele izmena (**transition table**) koje se često nazivaju i **pseudo tabele**. Tabela izmena je virtuelna tabela koja sadrži podatke koji su ažurirani, upisani ili obrisani iz tabele. Ako se vrši ažuriranje, kreiraju se dve tabele izmena, jedna za stare podatke, druga za nove podatke. Ako se vrši upisivanje, kreira se jedna tabela izmena za nove podatke, a ako se brišu podaci

kreira se jedna tabela izmena za stare podatke. Triger koristi informacije iz izvođačkog konteksta za svoje akcije, a u njemu se nalaze i tabele izmena.

SQL standard sintaksa komande kreiranja trigera:

```
CREATE TRIGGER trigger_name
{BEFORE | AFTER} {DELETE | INSERT | UPDATE [OF column [,...]}
ON table_name
[FOR EACH { ROW | STATEMENT }]
[WHEN (conditions)]
[BEGIN]
code_block
[END]
```

Ključne reči

CREATE TRIGGER trigger_name
Kreira triger sa imenom `trigger_name` i povezuje ga sa specificiranom tabelom.

BEFORE | AFTER
Deklariše da će se akcija trigera pokrenuti pre (BEFORE) ili posle (AFTER) operacije za rukovanje podacima (INSERT, UPDATE ili DELETE).

Trigeri BEFORE izvršavaju svoje operacije pre operacija za rukovanje podacima. To omogućava da se izvrše različite provere podataka i uslova pre izvršavanja ovih operacija, pa i njihovo zaobilaženje. Dobar primer primene ovakvih trigera je za očuvanje referencijalnog integriteta baze podataka.

Trigeri AFTER čije se akcije izvršavaju posle operacija za rukovanje podacima, omogućavaju, takođe, različite akcije, na primer, provera izračunatih zbirova itd., ali AFTER trigeri se ne izvršavaju ako je neko ograničenje već , jer se te pro triggers are never executed if a constraint violation occurs; therefore, these triggers cannot be used for any processing that might prevent constraint violations.

DELETE | INSERT | UPDATE [OF column [,...]]
Definiše operaciju za rukovanje podacima, koja uzrokuje pokretanje trigera. Pokretanje trigera je moguće vezati za određenu kolonu, na primer za ažuriranje UPDATE OF column [,...]. Tada promene na nespecificiranim kolonama ne izazvaju pokretanje trigera.

ON table_name
Deklariše postojeću tabelu na koju se triger odnosi.

FOR EACH { ROW | STATEMENT }
Govori bazi podataka da primeni triger za svaku vrstu u tabeli koja je promenjena (ROW) ili za svaku SQL naredbu koja je izdata nad tabelom (STATEMENT).
Na primer, naredba koja upisuje ocene studenata, za 100 studenata triger će se pokrenuti 100 puta ako je specificirano FOR EACH ROW. Ako se specificira FOR EACH STATEMENT, biće pokrenuta samo jednom.

WHEN (conditions)
Omogućava definisanje dodatnog uslova za triger.
Na primer, triger DELETE studenti, će se pokrenuti kad god se obrišu podaci o nekom studentu. Kada je uslov definisan se akcija trigera se pokreće samo ako je uslov koji se nalazi u klauzuli WHEN sračunat kao TRUE, inače, triger neće pokrenuti nikakvu akciju.

BEGIN | code_block | END

ANSI standard propisuje da `code_block` treba da sadrži samo jednu SQL naredbu. U sličaju da postoji veći broj komandi, tada one moraju da se smeste u okviru bloka (između BEGIN i END komandi).

Trigeri koriste dve pseudotabele koje se ne kreiraju naredbom CREATE TABLE, ali koje logički postoje u bazi podataka. Na različitim platformama one imaju različita imena, ali se mogu nazvati Pre (Before) i Posle (After) i njihova je struktura ista kao i tabele sa kojima je trigger povezan i sadrže snimak podataka u tabeli. Tabela Pre sadrži snimak svih vrsta pre nego što je trigger pokrenut, a tabela Posle snimak svih vrsta posle pokretanja trigera.
Dodavanje trigera tabeli koja već ima podatke neće uzrokovati pokretanje trigera. Triger se pokreće samo ako se izvršavaju naredbe za promenu podataka, koje su deklarisane u trigeru posle kreiranja trigera.
U odnosu na promenu strukture tabele, postojeći triger neće da reaguje na dodavanje nove kolone (izuzev UPDATE trigera), ali će javiti grešku, ako je kolona na koju se poziva izbrisana, svaki put kada se izvede.

T-SQL sintaksa komande `CREATE TIGGER` za `UPDATE`, `INSERT` i `DELETE`:

```
CREATE TRIGGER [ schema_name.]trigger_name
ON { table|view }
{ FOR | AFTER | INSTEAD OF }
{ [ INSERT ] [ , ] [ UPDATE ] [ , ] [ DELETE ] }
AS { sql_statement  [ ; ] [ ... ] }
```

SQL Server predviđa dve tranzicione tabele, koje se malo razlikuju od koncepta istih po SQL standardu i to:

- **`inserted`** – koja sadrži one zapise vrste koji treba da budu dodati u tabelu
- **`deleted`** – koji sadrži vrste koje se brišu

Za UPDATE komandu se formiraju obe tabele, inserted sadrži vrste za izmenjenim sadržajem, a deleted vrste sa starim vrednostima.

FOR je isto što i AFTER i ima ponašanje predviđeno SQL standardom.
INSTEAD OF trigeri se izvšavaju tako što se pokreću odmah nakon popunjavanja privremenih (tranzicionih tabela) tabela, a pre bilo kojih provera ograničenja, tako da mogu biti zaduženi za neku vrstu dopune ograničenjima.

SQL Server, dozvoljava definisanje ugnježdenih i rekurzivnih trigera. Pored DML trigera o kojima je dosad bilo reči, u okviru SQL Servera 2005 uvedeni su DDL trigeri koji su povezani sa događajima definisanja podataka u bazi podataka.

U sledećem primeru konstruiše se triger koji se pokreće prilikom upisivanja u tabelu Studenti. Svi kandidati sa indikatorom položio će biti upisani u tabelu Studenti, a sa indikatorom nije položio u tabelu Kandidati:

```
CREATE TRIGGER novi_studenti
INSTEAD OF INSERT ON Kandidati
BEGIN
INSERT INTO Studenti
SELECT * FROM inserted WHERE Status = 'položio'
INSERT INTO Kandidati
SELECT * FROM inserted WHERE status = 'nije položio'
END
```

LITERATURA

[1] Slajdovi sa predavanja - imi.pmf.kg.ac.rs

[2] D. Stefanović, *SQL i programiranje u relacionim bazama podataka*, PMF, Kragujevac, 2009. (postoji u bibiloteci)

[3] G. Pavlović-Lažetić, *Osnove relacionih baza podataka*, drugo izdanje, Matematički fakultet, 1999. (postoji u bibiloteci)

[4] Hoffer, Jeffrey A., *Modern database management*, Prentice Hall, 2011

[5] B. Lazarević, Z. Marjanović, N. Aničić, S. Babrogić, *Baze podataka*, FON, Beograd, 2003. (postoji u bibiloteci)

[6] J. Ullman, J .Widom, *A First Course in Database Systems*, Prentice Hall, 2008

[7] H. Garcia-Molina, J.D.Ulman, J. Widom, *Database Systems: The Complete Book*, Prentice Hall, 2002.