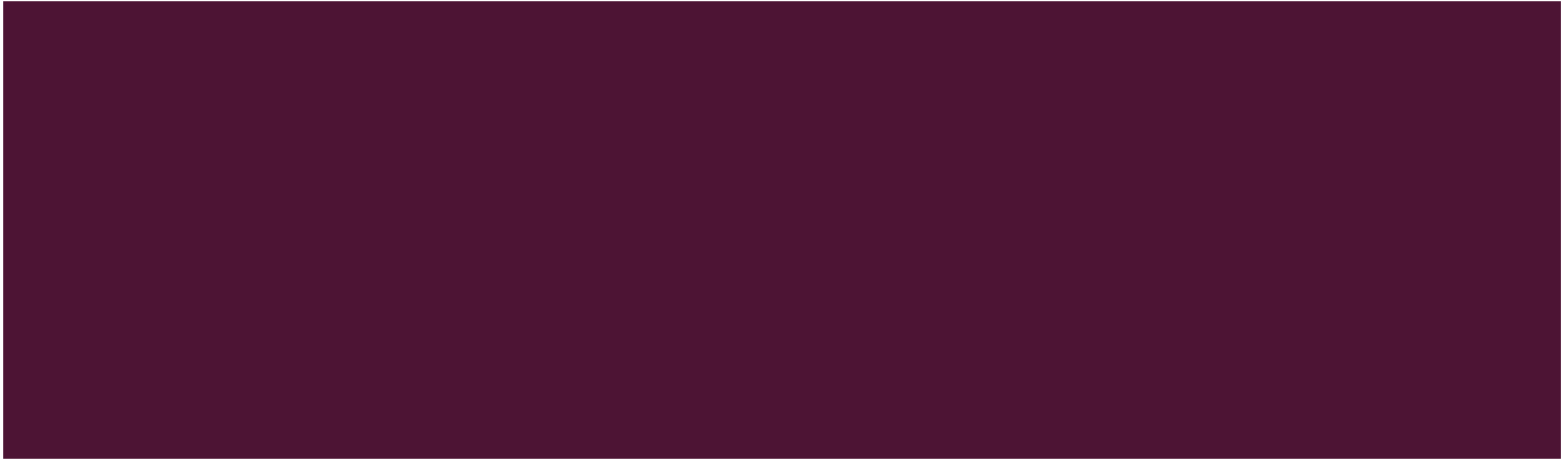




SQL



SQL

Strukturirani upitni jezik (Structured query language)

- najšire korišteni komercijalni jezik relacionih baza podataka.
- nastao kao rezultat standardizacije operacija 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 *
```

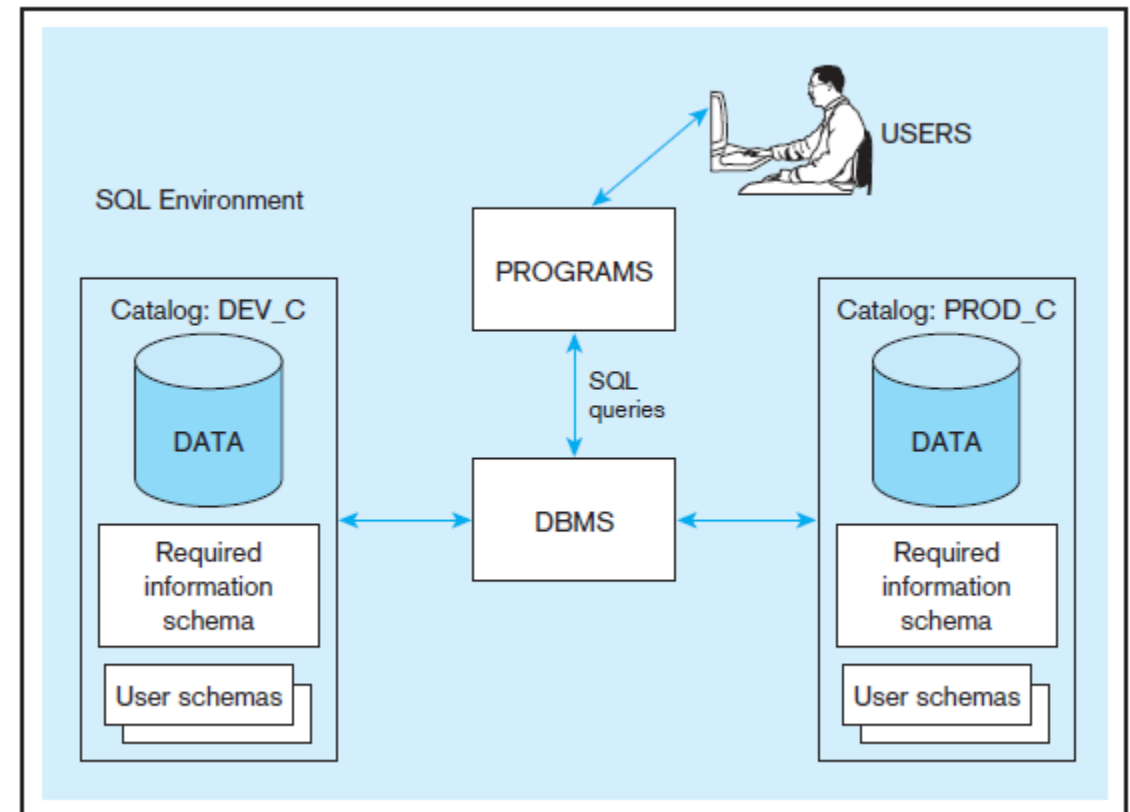
```
FROM Zaposleni
```

```
WHERE Godine > 40 OR Zarada > 60000
```

- Verzija SQL:2003 unela karakteristike proceduralnih jezika kao što su BEGIN blokovi, IF naredbe, funkcije i procedure.
- T-SQL – MS dijalekt SQL-a

SQL OKRUŽENJE

- Skup svih komponenti koje rade međusobno povezano sa ciljem čuvanja i operisanja podacima uz podršku SQL operacija čini **SQL okruženje**.
- SQL okruženje podrazumeva instancu DBMS-a zajedno sa bazama podataka (korisnički definisanim), kao i korisnike i programe koji koriste DBMS za pristup bazi.



SQL GRUPE KOMANDI

Tri osnovne funkcije koje podržava SQL su:

- **definisanje 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), pogledi
DDL-jezik baze podataka (Data Definon 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 pristupa 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)



DML



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]} [...] ]
```

SELECT

```
select A1,A2, ...,An  
from r1, r2, ..., rm  
where P
```

A_i - atributi relacija, R_i - relacije, P - predikat.

Rezultat SQL upita je **RELACIJA**.

SELEKCIJA I PROJEKCIJA REALIZOVANE U SQL-U

$\sigma_{\text{upisan}=2003}(\text{Studenti})$

```
SELECT *  
FROM Studenti  
WHERE Upisan=2003
```

$\pi_{\text{indeks,upisan,imes,mesto}}(\sigma_{\text{upisan}=2003}(\text{Studenti}))$

```
SELECT indeks,upisan,imes,mesto  
FROM Studenti  
WHERE Upisan=2003
```

DISTINCT, ORDER BY

```
SELECT DISTINCT mesto  
FROM Studenti  
WHERE Upisan=2003
```

```
SELECT *  
FROM Studenti  
WHERE Upisan=2003  
ORDER BY mesto, imes desc
```

ALIASI I RAČUNSKA POLJA

```
SELECT imes as 'Ime i prezime'
```

```
FROM Studenti
```

```
WHERE Upisan=2003
```

```
SELECT spred, semestar/2 + 1 'Godina'
```

```
FROM Planst
```

```
ORDER BY Godina desc
```

```
SELECT spred, Godina=semestar/2 + 1
```

```
FROM Planst
```

```
ORDER BY Godina desc
```

ALIASI I RAČUNSKA POLJA

```
SELECT imes as 'Ime i prezime'  
FROM Studenti  
WHERE Upisan=2003
```

```
SELECT spred, ceiling(semestar/2.0) as  
godina  
FROM Planst  
ORDER BY Godina desc
```

```
SELECT spred, semestar/2 +1 'Godina'  
FROM Planst  
ORDER BY Godina desc
```

```
SELECT spred, Godina=semestar/2 +1  
FROM Planst  
ORDER BY Godina desc
```

AGREGATNE FUNKCIJE I GRUPISANJE

Funkcije koje se primenjuju na skup vrednosti jednog atributa u relaciji ili čitave relacije i vraća neku 'sumarnu'.

AVG, MAX, MIN, SUM, COUNT

```
SELECT COUNT(*)
```

```
FROM Prijave
```

```
SELECT COUNT(SNAST)
```

```
FROM Prijave
```

```
SELECT Index, Upisan, COUNT(*)
```

```
FROM Prijave
```

```
GROUP BY Index, Upisan
```

```
SELECT Index, Upisan, COUNT(DISTINCT SPRED)
```

```
FROM Prijave
```

```
GROUP BY Index, Upisan
```

KLAUZULA HAVING

SELECT Index, Upisan, **AVG(OCENA*1.0)**

FROM Prijave

WHERE Ocena>5

GROUP BY Index,Upisan

HAVING count(*)>3

- Sve agregatne funkcije, osim count(*), ignorišu torke sa null vrednostima atributa po kojem se radi agregacija
- Ako kolekcija sadrži samo null vrednosti atributa
 - Count(attr) vraća 0
 - Ostali vraćaju null

SKUPOVNE OPERACIJE

- UNION, INTERSECT, EXCEPT eliminišu duplikate
- Da bi se zadržali duplikati, koriste se verzije definisane multiskupove (u T-SQL-u postoje samo za presek)

UNION ALL

Ako se K javlja m u relaciji r i n puta u relaciji s , koliko puta se javlja u

$r \cup s$

SKUPOVNE OPERACIJE

- **Dekartov proizvod**

```
SELECT s.*,p.*  
FROM Studenti s, Prijave p
```

- **Spajanje**

```
SELECT p.indeks,p.upisa,imes,spred,ocena  
FROM Studenti s JOIN Prijave p ON  
    s.indeks=p.indeks AND  
    s.upisan=p.upisan  
WHERE Ocena>5
```

Zadatak

Napraviti upit kojim se dobija spisak u kome je navedeno koji studenti su koje predmete polagali. Ako je neki student predmet polagao više puta ne treba ponavljati podatke.

VIŠE TABELA U UPITU

- Spajanje

```
SELECT p.indeks,p.upisan,imes,spred,ocena  
FROM Studenti s LEFT OUTER JOIN Prijave p ON  
    s.indeks=p.indeks AND  
    s.upisan=p.upisan  
WHERE Ocena>5
```

- LEFT OUTER JOIN, RIGHT OUTER JOIN, OUTER JOIN

UGNJEŽDENI UPITI

- Skalarni podupit

```
SELECT Indeks,Upisan,Spred,Ocena  
FROM Prijave p1  
WHERE Ocena > (SELECT AVG(p2.Ocena*1.0)  
                FROM Prijave p2)
```

- Skupovni podupit IN, NOT IN

```
select *  
from Studenti  
where Ssmer in (select Ssmer  
                from Angazovanje  
                where Snast=1)
```

POREĐENJE SA SKUPOM VREDNOSTI

- SOME/ANY, ALL

- $<$, $\leq$, $>$, $=$, $\neq$

$$(5 < \text{all } \begin{array}{|c|} \hline 0 \\ \hline 5 \\ \hline 6 \\ \hline \end{array}) = \text{false}$$

$$(5 < \text{all } \begin{array}{|c|} \hline 6 \\ \hline 10 \\ \hline \end{array}) = \text{true}$$

$$(5 = \text{all } \begin{array}{|c|} \hline 4 \\ \hline 5 \\ \hline \end{array}) = \text{false}$$

$$(5 \neq \text{all } \begin{array}{|c|} \hline 4 \\ \hline 6 \\ \hline \end{array}) = \text{true}$$

$(5 \neq 4 \text{ i } 5 \neq 6)$

$$(5 < \text{some } \begin{array}{|c|} \hline 0 \\ \hline 5 \\ \hline 6 \\ \hline \end{array}) = \text{true}$$

$$(5 < \text{some } \begin{array}{|c|} \hline 0 \\ \hline 5 \\ \hline \end{array}) = \text{false}$$

$$(5 = \text{some } \begin{array}{|c|} \hline 0 \\ \hline 5 \\ \hline \end{array}) = \text{true}$$

$$(5 \neq \text{some } \begin{array}{|c|} \hline 0 \\ \hline 5 \\ \hline \end{array})$$

TESTIRANJE PRAZNIH RELACIJA

- EXISTS, NOT EXISTS
 - **exists** $r \Leftrightarrow r \neq \emptyset$
 - **not exists** $r \Leftrightarrow r = \emptyset$

```
EXISTS (SELECT *  
        FROM Prijave  
        WHERE DATEPART(YEAR,Datump) = 2013)
```

Vraća

- TRUE ako ima prijava u 2013oj godini
- FALSE ako nema prijava u 2013oj godini

KORELISANI UPITI I KORELISANE VARIJABLE

```
SELECT *  
FROM Studenti  
WHERE (Studenti.Indeks,Studenti.Upisan) IN (SELECT Prijave.Indeks, Prijave.Upisan  
FROM Prijave  
WHERE Ocena = 10);
```

```
SELECT *  
FROM Studenti  
WHERE EXISTS (SELECT Prijave.Indeks, Prijave.Upisan  
FROM Prijave  
WHERE Prijave.Indeks = Studenti.Indeks AND  
Prijave.Upisan = Studenti.Upisan AND  
Ocena = 10);
```

UGNJEŽDENI UPITI

Podupit (ugnježdeni upit) može koristiti i u

SELECT A1,...,An za generisanje vrednosti atributa

FROM R1, ..., Rm za generisanje privremenih kolekcija (torki) koje se koriste kao izvor podataka za upit

Šta će biti rezultat sledećeg upita

```
SELECT indeks, upisan, prosek, (SELECT count(*)+1
                                FROM (SELECT indeks,upisan, avg(ocena*1.0) as prosek
                                      FROM Prijave
                                      WHERE ocena>5
                                      GROUP BY indeks,upisan) as u2
                                WHERE (u2.indeks != u1.indeks OR u2.upisan != u1.upisan) AND
                                       u2.prosek > u1.prosek) RBR
FROM (SELECT indeks,upisan, avg(ocena*1.0) as prosek
      FROM Prijave
      WHERE ocena>5
      GROUP BY indeks,upisan) u1
order by rbr
```

UGNJEŽDENI UPITI

Šta će biti rezultat sledećeg upita

```
SELECT indeks, upisan, prosek, (SELECT count(*)+1
                                FROM (SELECT indeks,upisan, avg(ocena*1.0) as prosek
                                      FROM Prijave
                                      WHERE ocena>5
                                      GROUP BY indeks,upisan) as u2
                                WHERE (u2.indeks != u1.indeks OR u2.upisan != u1.upisan) AND
                                       u2.prosek > u1.prosek) RBR
FROM (SELECT indeks,upisan, avg(ocena*1.0) as prosek
      FROM Prijave
      WHERE ocena>5
      GROUP BY indeks,upisan) u1
order by rbr
```

indeks	upisan	prosek	RBR
2	2001	9.388888	1
4	2000	9.277777	2
8	2003	9.2	3
3	2001	8.823529	4
3	2002	8.722222	5
99	2000	8	6
1	2002	7.857142	7
3	2000	7.722222	8
5	2003	7.4	9
4	2001	7.333333	10
2	2002	7.285714	11
1	2000	7.266666	12
6	2001	7	13

UGNJEŽDENI UPITI

Šta će biti rezultat sledećeg upita

```
SELECT indeks, upisan, prosek, (SELECT count(*)+1
                                FROM (SELECT indeks,upisan, avg(ocena*1.0) as prosek
                                      FROM Prijave
                                      WHERE ocena>5
                                      GROUP BY indeks,upisan) as u2
                                WHERE (u2.indeks != u1.indeks OR u2.upisan != u1.upisan) AND
                                       u2.prosek > u1.prosek) RBR
FROM (SELECT indeks,upisan, avg(ocena*1.0) as prosek
      FROM Prijave
      WHERE ocena>5
      GROUP BY indeks,upisan) u1
order by rbr
```

Izmeniti upit tako da ukoliko dva studenta imaju isti prosek onda manji redni broj ima onaj student:

- koji je ranije upisao fakultet,
- čiji je broj indeksa manji, ukoliko su ista upisna godina

indeks	upisan	prosek	RBR
2	2001	9.388888	1
4	2000	9.277777	2
8	2003	9.2	3
3	2001	8.823529	4
3	2002	8.722222	5
99	2000	8	6
1	2002	7.857142	7
3	2000	7.722222	8
5	2003	7.4	9
4	2001	7.333333	10
2	2002	7.285714	11
1	2000	7.266666	12
6	2001	7	13

KLAUZULA WITH

```
with proseci as (SELECT indeks, upisan, avg(ocena*1.0) as prosek
                  FROM Prijave
                  WHERE ocena > 5
                  GROUP BY indeks, upisan)
SELECT indeks, upisan, prosek, (SELECT count(*)+1
                                FROM proseci as u2
                                WHERE (u2.indeks != u1.indeks OR u2.upisan != u1.upisan) AND
                                       u2.prosek > u1.prosek) RBR
FROM proseci u1
order by rbr
```

ZADATAK

Odrediti broj studenata koji su odustali od studija, a upisali su studije školske 2015/16, i to po:

- Vrsti srednje škole koju su završili
- Studijskim grupama

STUDIJE(brind, godup, grupalD, smerID, aktuelne, ...)

UPIS(skgodina, brind, godup, semestar, ...)

VRSTASREDNJESTRUCNESPROME(vrstassID, naziv)

SMER(smerID, naziv)

GRUPA(grupalD, naziv)

ZADATAK

with prosla as

```
(select u.brind,u.godup,vs.naziv  
from upis u join student s on  
    u.brind = s.brind and  
    u.godup = s.godup  
join vrstasrednjestrucnespreme vs on  
    s.vrstassID = vs.vrstassID  
where skgodina=2015 and semestar=1),
```

sada as

```
( select brind,godup  
  from upis where skgodina=2016  
)
```

select *

from prosla t

```
where not exists(select *  
                  from sada u  
                  where u.brind=t.brind and  
                        u.godup=t.godup )
```

RAW_NUMBER()

Rangirati godine prema broju upisanih u opadajući redosled

```
select upisan, count(*) upisano, ROW_NUMBER() over(order by count(*) desc) rbr
from studenti
group by upisan
order by rbr
```

Za svaki predmet ispisati iz koje generacije ga je najviše studenata položilo

```
with rangliste as(
select spred, count(*) broj, upisan, ROW_NUMBER() over(partition by upisan
order by count(*) desc) rbr
from prijave
where ocena > 5
group by upisan, spred
)
select nazivp, upisan, rbr, broj
from rangliste r join predmeti p on
r.spred = p.spred
where rbr=1
```

Dobiti iste podatke
bez rednog broja i
bez korišćenja
ROW_NUMBER

GRUPISANJE (2)

▪ ROLLUP

```
select upisan,spred, count(*) broj
from PRIJAVE
where ocena>5
group by rollup(upisan,spred)
```

count(*) po group by upisan,spred
+
count(*) po group by upisan
+
count(*)

▪ CUBE

```
select upisan,spred, count(*) broj
from PRIJAVE
where ocena>5
group by upisan,spred
with cube
```

count(*) po group by upisan,spred
+
count(*) po group by upisan
+
count(*) po group by spred
+
count(*)

PIVOT

```
-- Koliko je studenata koji ispit položio koje godine
select spred, year(datump) as godina, count(*) as broj
from PRIJAVE
where ocena>5
group by spred, year(datump)
order by spred
```

```
-- Koliko je studenata koji ispit položio koje godine PIVOT
with pom as
(select year(datump) as godina, spred, indeks
from PRIJAVE
where ocena>5
)
select spred,[2001],[2002],[2003],[2004],[2005],[2006]
from pom
PIVOT (COUNT(indeks) FOR godina in
      ([2001],[2002],[2003],[2004],[2005],[2006])) as p
```

spred	godina	broj
1	2001	3
1	2002	3
1	2003	3
1	2004	2
1	2006	1
2	2001	3
2	2002	3
...		

spred	2001	2002	2003	2004	2005	2006
1	3	3	3	2	0	1
2	3	3	3	2	0	0
3	2	4	2	1	2	0
4	0	5	2	4	0	0
5	0	3	3	3	0	0
6	0	3	2	3	1	0
7	0	2	1	3	2	0
8	0	0	3	2	4	0
9	0	0	3	3	3	0
10	0	0	1	3	2	1
...						

IZMENA PODATAKA - INSERT

```
[ WITH <common_table_expression> [ ,...n ] ]  
INSERT  
{  
    [ TOP ( expression ) [ PERCENT ] ]  
    [ INTO ] { <object> | rowset_function_limited }  
    {  
        [ ( column_list ) ]  
        [ <OUTPUT Clause> ]  
        { VALUES ( { DEFAULT | NULL | expression } [ ,...n ] ) [ ,...n ]  
          | derived_table | execute_statement | <dml_table_source> | DEFAULT VALUES  
        }  
    }  
}
```

IZMENA PODATAKA – INSERT (2)

```
INSERT INTO Studenti VALUES (99,NULL,'Stevan',NULL,NULL,NULL)
```

```
INSERT INTO studenti (Upisan,Imes,Indeks)
```

```
VALUES (2000, 'Stevan', 1000)
```

```
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
```

IZMENA PODATAKA – INSERT (3)

```
-- kreiranje tabele polozenih ispita, iz upita
```

```
select *  
into polozeni  
from prijave  
where ocena > 5
```

```
select *  
from polozeni  
drop table polozeni
```

```
-- dodavanje novih torki uz ispis dodatog
```

```
insert into polozeni  
output inserted.*  
select *  
from prijave  
where ocena > 5
```

IZMENA PODATAKA - DELETE

- **Brisanje studenata iz Kragujevca**

```
delete CopyStudent  
Output deleted.*  
where mesto = 'Kragujevac'
```

```
Select *  
Into CopyStudent  
From Studenti  
Where upisan>2001
```

- **Brisanje studenata koji su diplomirali**

```
delete from c  
output deleted.*  
from CopyStudent c join smer s on  
c.ssmer=s.ssmer  
where nazivs = 'Primenjena matematika'
```

Isti rezultat dobiti upotrebom operatora
EXISTS

IZMENA PODATAKA - UPDATE

UPDATE Prijave

SET Ocena = Ocena + 1

WHERE Indeks = 99 AND Upisan = 2000 AND Spred = 17

UPDATE Prijave SET Ocena = 8

WHERE Indeks = 99 AND Upisan = 2000 AND Spred = 17 AND Datump = '2007-03-24'

UPDATE Studenti SET Mesto = (SELECT Mesto

FROM Studenti

WHERE Imes = 'Sima')

WHERE Imes = 'Stevan' AND Indeks = 99;



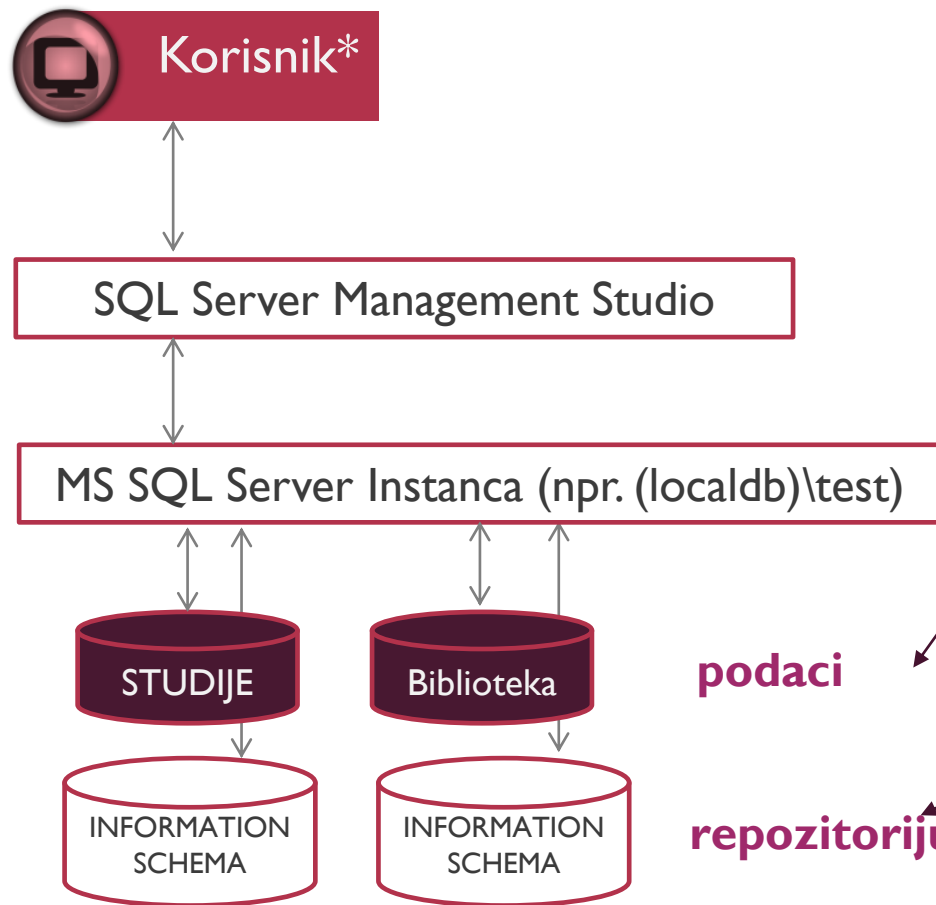
DDL



DDL

- DDL (Data Definiton Language)
- Definisanje meta podataka kojima se opisuje struktura baze podataka.
Meta podaci se nalaze u **repozitorijumu**.
SQL Server: **information_schema**.
- Kreiranje tabela, atributa, tipova, ograničenja, indeksa, pogleda, ...

SQL SERVER OKRUŽENJE



SELECT * FROM Studenti

	Indeks	Upisan	Imes	Mesto	Datr	Ssmer
1	1	2000	Ana	Kragujevac	1982-05-21 00:00:00.000	2
2	1	2002	Nenad	Kragujevac	1984-06-15 00:00:00.000	1
3	2	2001	Sanja	Jagodina	1983-04-20 00:00:00.000	3
4	2	2002	Voja	Lapovo	1984-08-15 00:00:00.000	4
5	3	2000	Sava	Kragujevac	1982-08-03 00:00:00.000	1

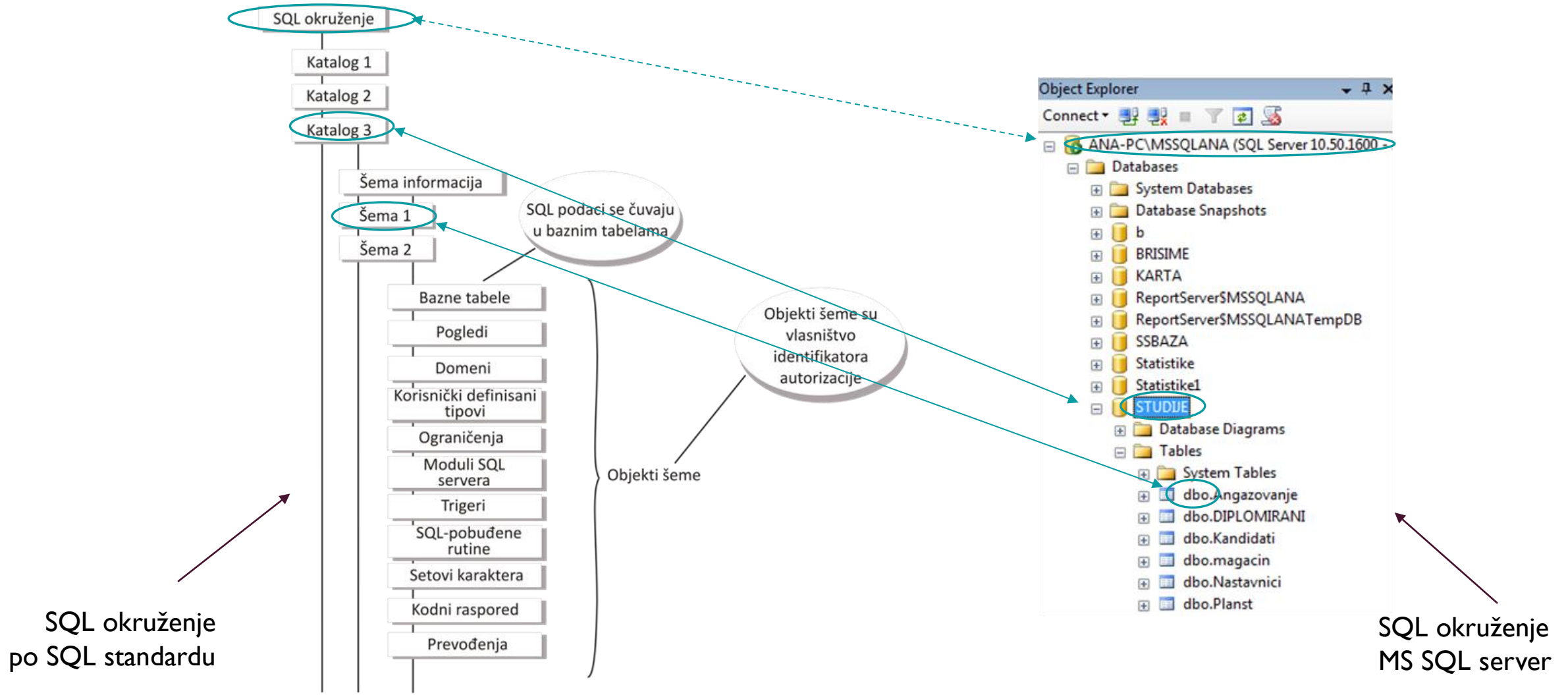
SELECT *
FROM INFORMATION_SCHEMA.COLUMNS WHERE
TABLE_NAME = 'Studenti'

	TABLE_CATALOG	TABLE_SCHEMA	TABLE_NAME	COLUMN_NAME
1	STUDIJE	dbo	Studenti	Indeks
2	STUDIJE	dbo	Studenti	Upisan
3	STUDIJE	dbo	Studenti	Imes
4	STUDIJE	dbo	Studenti	Mesto
5	STUDIJE	dbo	Studenti	Datr
6	STUDIJE	dbo	Studenti	Ssmer

podaci

repozitorijum

KATALOG, ŠEMA, INFORMATION_SCHEMA



KATALOG, ŠEMA, INFORMATION_SCHEMA

SQL Standard

- **SQL okruženje** sadrži nula ili više kataloga, a katalog sadrži jednu ili više šema.
- **Katalog** je imenovana kolekcija šema baze podataka u SQL okruženju (jedna ili više logički povezanih šema čine katalog).
Katalog obezbeđuje hijerarhijsku strukturu za organizovanje podataka u okviru šema.
- **Šema** je kontejner za objekte kao što su tabele, pogledi i domeni.
- Svaki katalog sadrži jednu specifičnu šemu pod nazivom **INFORMATION_SCHEMA** (informaciona šema), koja predstavlja rečnik podataka tog kataloga. Nju čini skup pogleda nad sistemskim tabelama, koje sadrže sve bitne informacije o SQL okruženju datog kataloga.

IMENOVANJE OBJEKATA

- Za svaki objekat se zna 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 u trenutno aktivnom katalogu.

DATA MANIPULATION LANGUAGE

- 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 T-SQL (nije predviđena SQL standardom)
 - Schema
 - Tabela
 - Ograničenje
 - Indeks
 - Pogled

KREIRANJE BAZE PODATAKA

```
CREATE DATABASE database_name
```

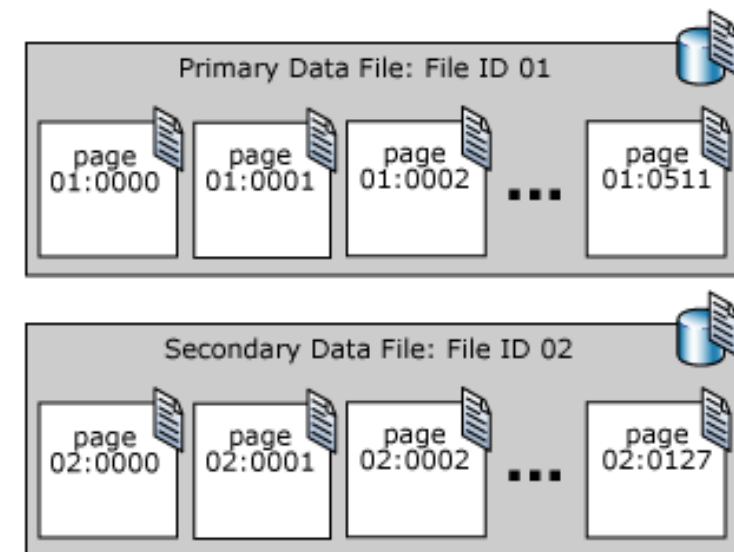
```
    [ ON          [ PRIMARY ] [ <filespec> [ ,...n ]  
        [ , <filegroup> [ ,...n ] ]  
        [ LOG ON { <filespec> [ ,...n ] } ] ]  
    [ COLLATE collation_name ]  
] [:]
```

```
CREATE DATABASE STUDIJE
```

```
ON PRIMARY  
    (NAME = PODACI1,  
     FILENAME = 'D:\ispit\primarna.mdf',  
     SIZE = 5,  
     MAXSIZE = 20,  
     FILEGROWTH = 5),  
FILEGROUP GRUPA2  
    (NAME = PODACI2,  
     FILENAME = 'D:\ispit\sekundarna1.ndf',  
     SIZE = 5,  
     MAXSIZE = 25,  
     FILEGROWTH = 5)  
LOG ON  
    (NAME = fakultet_log,  
     FILENAME = 'D:\ispit\dnevnik\fak_dnevnik.ldf',  
     SIZE = 5,  
     MAXSIZE = 20,  
     FILEGROWTH = 5)
```

KREIRANJE BAZE PODATAKA

- Ime baze podataka mora da bude jedinstveno u okviru instance SQL servera.
- 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.
- Baza podataka može da ima samo jednu primarnu datoteku. Prva datoteka koja se specificira u <filespec> u primarnoj filegroup postaje primarna datoteka.
- Mogu postojati i sekundarne grupe koje se mogu koristiti i za podatke i za dnevnik sa ekstenzijom NDF.



IZMENA I BRISANJE BAZE PODATAKA

ALTER DATABASE STUDIJE

ADD FILE

(

NAME = PODACI3,

FILENAME = 'D:\ispit\sekundarna2.ndf',

SIZE = 5MB,

MAXSIZE = 50MB,

FILEGROWTH = 5MB

)

TO FILEGROUP GRUPA2

ALTER DATABASE STUDIJE

REMOVE FILE PODACI3

DROP DATABASE STUDIJE

KREIRANJE, IZMENA I BRISANJE ŠEME

```
use studije  
go  
CREATE SCHEMA NastOdr
```

```
CREATE TABLE NastOdr.Nastavnici (  
  Snast int not null,  
  Imen char(25)  
)
```

```
use [STUDIJE]  
GO  
GRANT TAKE OWNERSHIP ON SCHEMA::[NastOdr] TO [Savic]  
GO
```

```
ALTER SCHEMA NastOdr  
TRANSFER OBJECT::dbo.Prij;
```

```
DROP SCHEMA NastOdr
```

Šema ne sme sadržati objekt, odnosno vi objekti moraju biti uklonjeni iz šeme.

PODRAZUMEVANA ŠEMA

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

```
SELECT *  
FROM Studenti
```

ima isto značenje kao i

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

Ukoliko je podrazumevana šema korisnika koji ga izvršava dbo.

* Ako se ne naglasi drugačije default schema koja se definiše za svakog korisnika je dbo.

- Podrazumevana šema može biti eksplicitno definisana za svakog korisnika pojedinačno, pri kreiranju ili promeni definicije korisnika (CREATE/ALTER USER).

```
USE Studije;  
CREATE USER Savic FOR LOGIN SavicMilos  
WITH DEFAULT_SCHEMA = NastOdr;
```

KREIRANJE I IZMENA TABELE

SQL održava tri tipa tabela:

- **bazne tablele**

Mogu da se podele u dve kategorije:

- stalne (persistent) i
- privremene (temporary).

- **izvedene tablele** - rezultati realizacije SQL upita nad bazom podataka,
pogledi (view) - tip izvedenih tabela čija definicija se čuva u šemi.

KREIRANJE I IZMENA TABELA (2)

SQL podržava četiri tipa baznih tabela

- **Stalne bazne table.** Ove table predstavljaju imenovane objekte šeme koji su definisani naredbom CREATE TABLE.
- **Globalne privremene table.** Ove table predstavljaju imenovane objekte šeme koji su definisani naredbom CREATE GLOBAL TEMPORARY TABLE. Postoje dok traje sesija u kojoj su kreirane. Dostupne su ostalim sesijama.
- **Lokalne privremene table.** Ove table predstavljaju imenovane objekte šeme koji su definisani naredbom CREATE LOCAL TEMPORARY TABLE. Vidljive samo u okviru konteksta sesije u kojoj su kreirani.
- **Deklarisane privremene table.** To su table deklarisanе kao deo procedure u modulu (npr. storne procedure). Definicija table nije smeštena u šemi i ne može da postoji dok se procedura ne pozove. Kao i sve privremene table može da bude pozvana samo u okviru konteksta SQL sesije.

KREIRANJE I IZMENA TABELA (3)

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-podatka [ NULL | NOT NULL ]  
[[ CONSTRAINT ime_ogranicenja] DEFAULT constant_expression]
```

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

KREIRANJE I IZMENA TABELE (4)

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

```
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' )));
```

```
CREATE TABLE Studenti  
(Indeks SMALLINT,  
Upisan SMALLINT DEFAULT(2000),  
Imes CHAR(25),  
Mesto CHAR(30),  
Datr DATETIME,  
Ssmer SMALLINT REFERENCES Smer(Ssmer),  
CONSTRAINT pk_stud PRIMARY KEY CLUSTERED(Indeks,Upisan));
```

KREIRANJE I IZMENA TABELE (4)

Modifikuje definiciju tabele menjanjem, dodavanjem ili brisanjem kolona i ograničenja, onemogućava i omogućava ograničenja i okidače.

```
ALTER TABLE Smer ALTER COLUMN Nazivs CHAR(40) NOT NULL;
```

```
ALTER TABLE Smer NOCHECK CONSTRAINT SMER;
```

```
ALTER TABLE Smer ALTER COLUMN Nazivs CHAR(40) NOT NULL;
```

```
ALTER TABLE Prijave ADD CONSTRAINT tek_dat  
DEFAULT GETDATE() FOR Datump;
```

PRIVEMENE TABELE

- Lokalna privremena tabela se definiše imenovanjem sa prefiksom # (#table_name).
- Globalna privremena tabela se definiše imenovanjem sa prefiksom ## (##table_name).

```
CREATE TABLE #MyTempTable (  
    col1 INT PRIMARY KEY  
);  
INSERT INTO #MyTempTable VALUES (1);
```

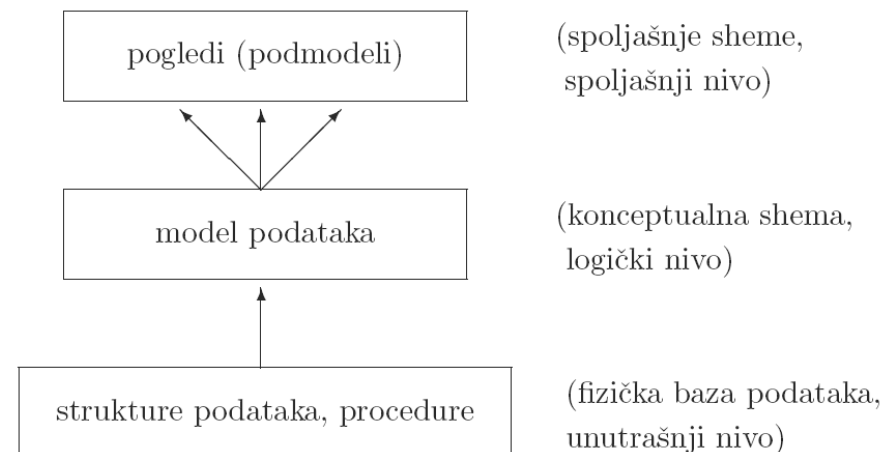
- Deklarisane privremene tabele. Deklarišu se i koriste unutar storne procedure. Njihovo postojanje traje dok se izvršava sama procedura.

```
CREATE PROCEDURE dbo.Test2 AS  
    CREATE TABLE @t (x INT PRIMARY KEY);  
    INSERT INTO @t VALUES (2);  
    SELECT Test2Col = x FROM @t;  
GO
```

POGLEDI

- Pogledi su virtuelne tabele koje nisu predstavljene fizičkim podacima, već je njihova definicija zapamćena u katalogu.
- U relacionom modelu, pogledima su analogne izvedene tabele.
- Pogledima je omogućeno:
 - skrivanje izabranih podataka
 - modularnost baze

Pogledi (Views)
sadrže podskupove konceptualnog nivoa
Konceptualni nivo (Schema)
definiše logičku strukturu; sadrži opise podaka
i njihovih odnosa
Fizička šema
opisuje kako su podaci memorisani



POGLEDI (2)

CREATE VIEW [schema_name .] view_name [(column [,...n])]

AS select_statement

- Većina DMBS-ova dozvoljava kreiranje takozvanih **materijalizovanih pogleda** - fizički kreiranu tabelu čiji je sadržaj definisan rezultatom upita.
- Materijalizovani pogledi, tj. rezultujuća tabela se periodično ažurira u skladu sa sadržajem baznih tabela koje učestvuju u pogledu.
- SQL server dozvoljava definisanje **indeksiranih pogleda**.

POGLEDI (2)

```
CREATE SCHEMA predm_sch
CREATE VIEW predm_sch.uslovni_pogl
AS
SELECT Predmeti.Spred, Predmeti.Nazivp FROM dbo.Predmeti
JOIN dbo.Uslovni ON dbo.Predmeti.Spred = dbo.Uslovni.Spred
```

POGLEDI (3)

- Ažuriranje preko pogleda je moguće uz određene uslove.
- 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
- Vrednosti koje se upisuju ili ažuriraju u osnovnoj tabeli moraju da ispunjavaju sva ograničenja postavljena na njima sa jedinstvenim indeksima, primarnim ključevima itd.

OGRANIČENJA

ANSI standard predviđa četiri tipa ograničenja:

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

Promena definicije tabele dodavanjem ograničenja

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

U ograničenje spada i NOT NULL opcija koja se koristi u definiciji kolone.

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.

OGRANIČENJA (2)

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);
```

OGRANIČENJA (3)

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.

OGRANIČENJA (5)

UNIQUE ograničenja (UNIQUE constraints)

Ograničenjem UNIQUE (ograničenje kandidata za ključ) se deklarira da vrednosti atributa ili kolekcije atributa moraju biti jedinstvene.

CHECK ograničenja (CHECK Constraints)

Sintaksa opcije zadavanja ograničenja uslovom je

[CONSTRAINT ime] CHECK (uslov)

CHECK ograničenje se smatra odgovarajućim kada se uslov pretraživanja sračunava u **TRUE** ili **UNKNOWN**.

Primer.

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

OGRANIČENJA (4)

```
CREATE TABLE Prodavci
(
    prod_id          CHAR(4)      NOT NULL,
    prod_ime         VARCHAR(40),
    prod_adresa1     VARCHAR(40) UNIQUE,
    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));
```

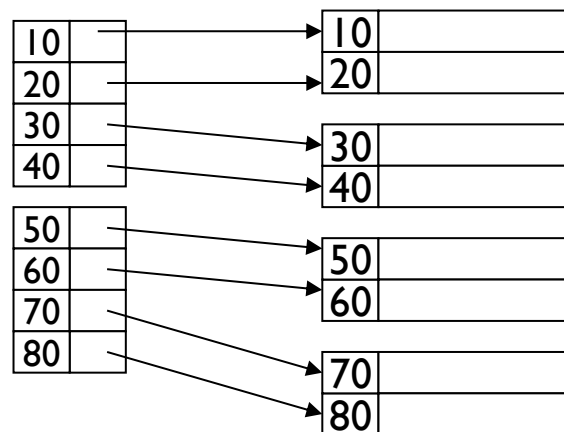
INDEKSI

- Indeksi su posebne strukture podataka RDBMS koristi za povećanje brzine u rukovanju podacima.
- Indeksi su 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.

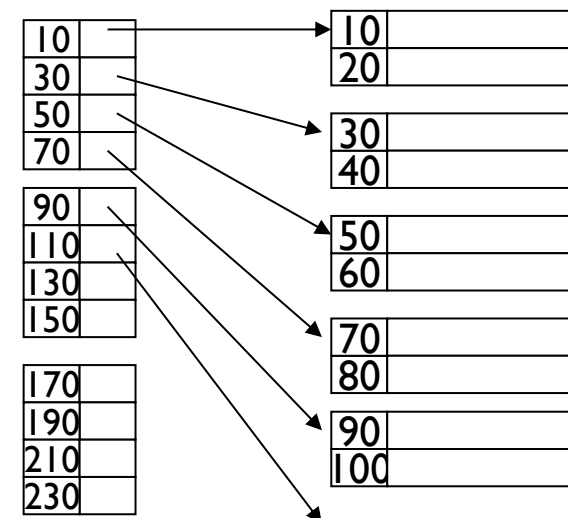
CREATE [UNIQUE] INDEX ime_indeksa ON ime_tabele

(ime_kolone [, ...])

Dense Index



Sparse Index



Sequential File



SQL RUTINE

Proceduralno programiranje

PROCEDURALNO PROGRAMIRANJE I SQL

- Verzija SQL:2003 unela karakteristike proceduralnih jezika.
- SQL rutine su objekti baza podataka koji su čuvani i kontrolisani od strane SUBP-a i:
 - sastoje se iz blokova proceduralnog koda,
 - moraju biti **eksplicitno pozvane** da bi bile izvršene.
- U SQL rutine spadaju: **funkcije** (user-defined function - UDF) i **procedure** (stored procedures).
- I funkcije i procedure su memorisani skupovi predefinisanih SQL naredbi, koje izvršavaju određeni skup akcija nad bazom podataka.

FUNKCIJE – T-SQL PRIMERI

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

```
SELECT * FROM plan_na_smeru(3)
```

FUNKCIJE – T-SQL PRIMERI

```
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)

PROCEDURE – T-SQL PRIMERI

```
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
```

KURSORI

- Mehanizam koji omogućava sekvencijalni pristup skupu n-torki koje se nalaze u rezultujućem skupu nekog upita.
- Komande:
 - **DECLARE** - deklarira kursor definisanjem njegovog imena, karakteristika i upita koji se poziva pri otvaranju kursora
 - **OPEN** - izvršava naredbu SELECT i omogućava da rezultat bude raspoloživ za naredbu FETCH, tj. za upotrebu
 - **FETCH** - podatke iz skupa koji predstavlja rezultat izvršenog upita i smešta ih u promenljive koje predaju podatke jeziku domaćinu ili drugim SQL naredbama

FETCH

[[NEXT|PRIOR|FIRST|LAST]FROM]

{cursor_name | @cursor_variable_name}

[INTO @variable_name [,...n]]

- **CLOSE** - oslobađa blokove podataka sa kojima je kursor povezan, kao i lokove
- **DEALLOCATE** - T-SQL, uklanja kursorску strukturu.

KURSORI

```
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
```

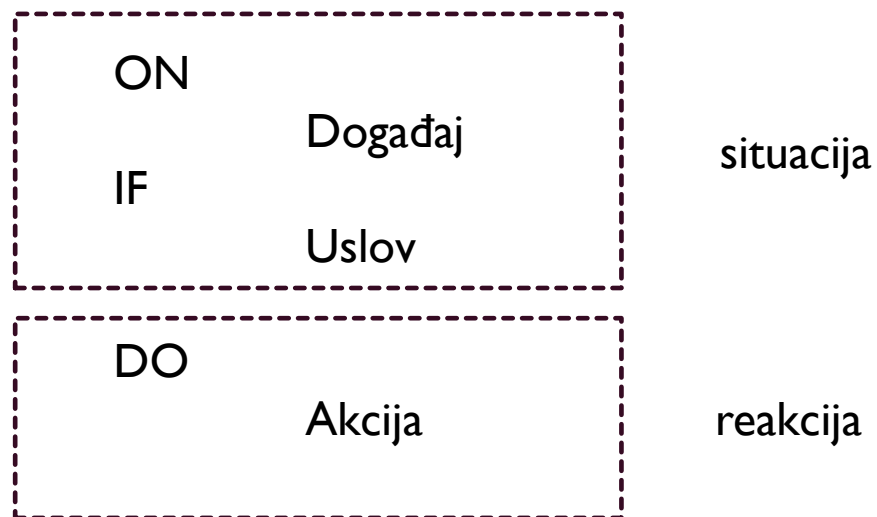
TRIGERI

Aktivne baze podataka

G. Pavlović-Lažetić, Osnove relacionih baza podataka, drugo izdanje, Matematički fakultet, 1999.
B. Lazarević, Z. Marjanović, N. Aničić, S. Babrogić, Baze podataka, FON, Beograd, 2003.

AKTIVNE BAZE PODATAKA

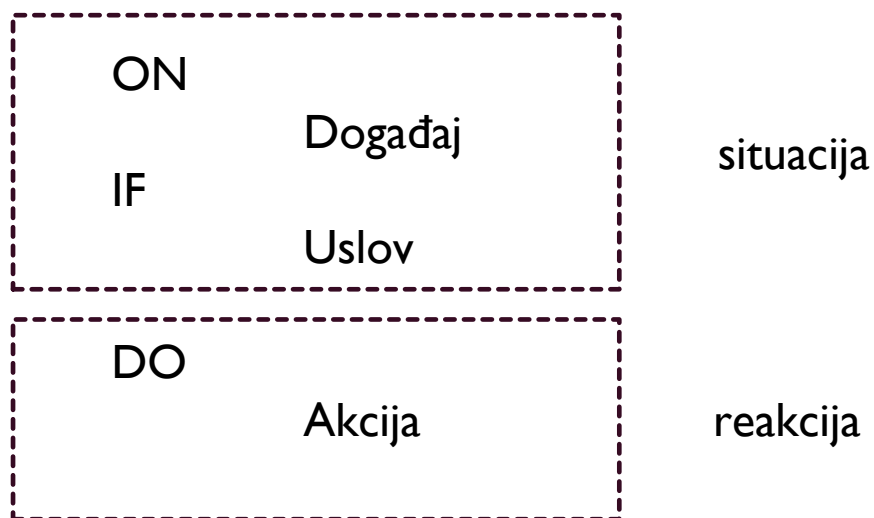
- Pasivni sistemi baza podataka beleže logičku šemu i podatke i podržavaju samo proveru osnovnih integritetnih pravila, PK, FK. Sva aktivnost se inicira od strane korisnika.
- **Aktivni sistemi baza podataka** imaju mogućnost definisanja i prepoznavanja situacija u kojima se vrši automatsko pokretanje procedura kojima je definisana reakcija na prepoznatu situaciju.



- Koriste unificirani mehanizam za:
 - Kontrolu integriteta,
 - Kontrolu prava pristupa,
 - Prikupljanje statističkih podataka,
 - Praćenje funkcionisanja baze ...

SPECIFIKACIJA PRAVILA

- Omogućuju implementaciju dodatne dinamike sistema iskazane složenim pravilima integriteta, **ECA (Event-Condition-Action) pravila**.



- Formiranje pravila zahteva definisanje događaja.
- Događaji:
 - **Primitivni**
 - Ažuriranje podataka
 - Prikaz podataka
 - Vreme
 - Aplikativno definisan događaj
 - **Složeni** – kombinacija više primitivnih i složenih
- Akcije se pokreću neposredno nakon (**AFTER**) ili neposredno pre (**BEFORE**) detektovanja događaja.

TRIGERI

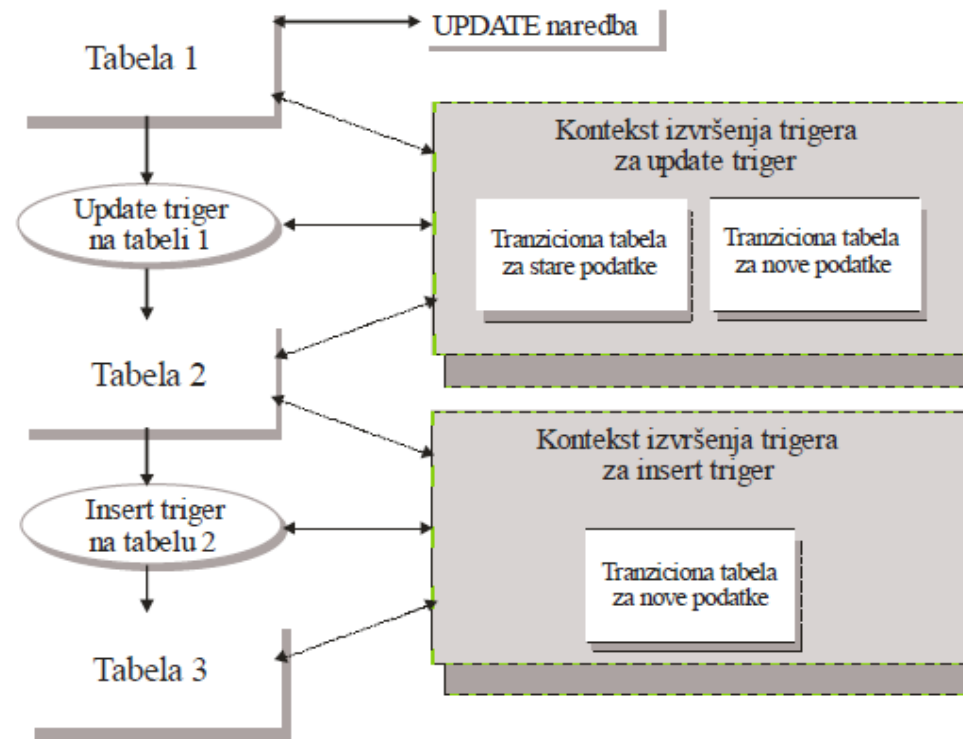
- **Trigeri (okidači)** su specifična vrsta ECA (Event-Condition-Action) pravila, uvedena u SQL standard od SQL:1999.
- Triger je **proceduralni mehanizam** koji specifičnoj operaciji modifikovanja podataka – **triger operaciji**, nad specifičnom baznom relacijom, pridružuje niz SQL iskaza, **triger proceduru**, koja se aktivira/**okida**/izvrši kadgod se izvrši triger operacija.
- Trigeri predstavljaju deo baze i njima je obezbeđena opštija forma definisanja uslova integriteta:
 - Validacija ulaznih podataka
 - Automatsko generisanje vrednosti atributa nove vrste
 - Za čitanje iz ili upis u druge tabele

TRIGERI

- Tri vrste trigera:
 - INSERT
 - UPDATE
 - DELETE
- Tri komponente:
 - **događaj**, koji predstavlja određenu modifikaciju,
 - **uslov** koji se proverava i na osnovu njegove ispunjenosti pokreće trigger. Rezultat provere uslova je TRUE ili FALSE.
 - ako je uslov ispunjen automatski se pokreće **akcija** opisana u definiciji trigera.

TRIGERI

- Trigger se izvršava u svom **izvođačkom kontekstu**.
- Izvođački kontekst uključuje 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.
- Po SQL sandardu:
- 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.
- Ako se **brišu** podaci kreira se jedna tabela izmena za stare podatke.



Slika 14-1. Konteksti izvršavanja trigeru za dva trigera

TRIGERI

```
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:
 - **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.

- Trigeri **FOR/AFTER** se 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 narušeno.
- **INSTEAD OF** trigeri se izvršavaju 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.

TRIGERI

```
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
```