SQL SERVER

Mr Dušan Marković dipl ing

Tipovi podataka u SQL serveru

- Celobrojne vrednosti
- Tekstualne vrednosti
- Decimalne vrednosti
- Novčane vrednosti
- Vrednosti sa pokretnim zarezom
- Datumske vrednosti
- Binarne vrednosti
- Ostali tipovi podataka

binary	**bigint**	**bit**	**char**	**datetime**
decimal	**Float**	**image**	**int**	**money**
nchar	**ntext**	**nvarchar**	**numeric**	**Real**
smalldatetime	**smallint**	**smallmoney**	**sql_variant**	**sysname**
text	**timestamp**	**tinyint**	**varbinary**	**varchar**
uniqueidentifier				

Binarne vrednosti

binary [(*n*)]
Fiksna dužina *n* bajta. *n* mora biti vrednost od 1 do 8,000. Smeštajna veličina je *n*+4 bajta.

varbinary [(*n*)]
Promenljiva dužina *n* bajta. *n* mora biti vrednost od 1 do 8,000.
Smeštajna veličina je aktuelna dužina + 4 a ne *n*.

Ako *n* nije definisano prilikom deklarisanja podrazumevana vrednost je 1.

image
Kolona tipa image sadrži dugačke nizove binarnih podataka, do 2^31 –1 bajtova. Za kolone tipa image ne morate da zadajete maksimalnu veličinu, jer se kolona automatski širi tako da može da uskladišti najveći podatak

Celobrojne vrednosti

bigint
Celobrojni tip podataka od –2^63 (-9223372036854775808) do 2^63 –1 (9223372036854775807) zauzeće je 8 bajta

int
Celobrojni tip podataka od –2^31 (-2147483648) do 2^31 –1 (2147483647), zauzeće je 4 bajta

smallint
Celobrojni tip podataka od –2^15 (-32768) do 2^15 –1 (32767), zauzeće je 2 bajta

tinyint
Celobrojni tip podataka od 0 do 255, zauzeća od 1 bajta

bit
Celobrojni tip podataka koji može sadržati 1, 0 ili null

Tekstualne vrednosti

char [(*n*)]
Fiksna dužina ne – unikodnog karaktera dužine *n* bajta. *n* mora biti vrednost od 1 do 8000. Smestajna veličina je *n* bajta

varchar [(*n*)]
Promenljiva dužina ne – unikodnog karaktera dužine *n* bajta. *n* mora biti vrednosti od 1 do 8000. Smeštajna veličina je aktuelna veličina a ne broj *n*. Unešeni broj podataka može biti dužine od 0 karaktera.

text
Kolona tipa *text* su automatski definisane, kao kolone promenljive širine. Namena kolone tipa *text* je skladištenje izuzetno dugačkog znakovnog niza koji nisu u unicode formatu. Maksimalna širina kolone je 2^31 –1 ili 2 147 483 647 znakova

nchar [(*n*)]
Kolona sadrži fiksni broj unicod znakova. Kolona će sadržati *n* karaktera i ako joj dodelite manji broj. Pošto kolone tipa *nchar* koriste skup znakova unicode, one mogu sadržavati znatno širi opseg znakova nego standardne kolone tipa *char*

nvarchar [(*n*)]
Kolona tipa *nvarchar* sadrži promenljiv do *n* broj unicod znakova.

ntext

Kolone tipa *ntext* automatski su definisane kao kolone promenljive širine. Namena je da uskladištite izuzetno dugačke nizove unicod znakova. Maksimalna širina kolone tipa *ntext* je 2^30 –1 ili 1 073 741 823 znaka.

Novčane vrednosti

money

Kolona tipa *money* može da sadrži vrednosti u opsegu od -922 337 203 685 477,5808 do 922 337 203 685 477,5807. Podaci tipa *money* uvek se čuvaju sa četiri decimalna mesta

smallmoney

Kolona tipa *smallmoney* može da sadrži vrednosti u opsegu od -214 748,3648 do 214 748,3647. Podaci tipa *smallmoney* uvek se čuvaju sa četiri decimalna mesta.

Datumske vrednosti

datetime

Kolona definisana sa ovim tipom podataka može da sadrži datumske vrednosti od 1. januara 1753. godine do 31. decembra 9999 uz tačnost od 3,33 milisekunde

smalldatetime

Kolona definisana kao ovaj tip promenljive sadrži datumske vrednosti od 1. januara 1900 do 6. juna 2079 godine uz tačnost od 1 minuta.

Decimalne vrednosti

decimal/numeric

Kada se definiše ovaj tip podataka mora se zadati preciznost i razmer
- Preciznost je ukupan broj cifara koji će imati vrednosti u koloni
- Razmer je broj decimalnih mesta iza iza tačke ili zareza...

prim. *decimal(5,3)*

Vrednosti sa pokretnim zarezom

float [(*n*)]

Kolona tipa float može da sadrži vrednosti u opsegu od –1,79x10^308 do 1,79x10^308. Vrednost *n* je broj bitova za skladištenje preciznosti kolone. Opseg za vrednost *n* je od 1 do 53.

real

To je float(24). Vrednosti su u opsegu od približno -3,4 x 10^38 do 3,4 x 10^38

Ostali tipovi podataka

timestamp

To je kolona binarnog tipa širine 8 bajtova, koja sadrži jedinstvenu vrednost koju generiše SQL server. Jedna tabela može da ima jednu kolonu *timestamp*.

uniquidentifier

Kolona čiji je tip podataka *uniquidentifier* može da sadrži globalni jedinstveni identifikator.

sql_variant

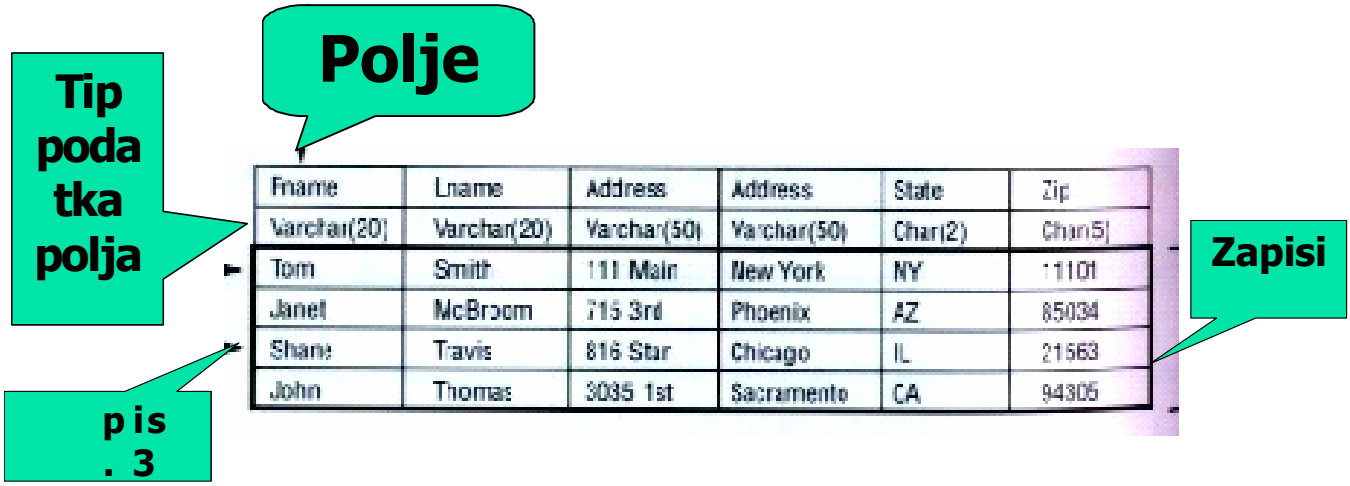
To je "džoker" tip podataka koji može da sadrži bilo koji drugi tip podataka osim *text*, *ntext*, *timestamp*

table

Ovaj tip se koristi za privremeno skladištenje skupa rezultata tokom izvršavanja određene funkcije, uskladištene procedure ili grupne operacije.

Kreiranje tabela

Tabele su objekti u bazi podataka koje se koriste za skladištenje podataka i sastoje se iz polja i zapisa.



T-SQL naredbe za tabele

- **CREATE TABLE** – služi za kreiranje tabele
- **ALTER TABLE** – služi za unos, izmena kolona, ili brisanje kolona
- **DROP TABLE** – služi za brisanje određene tabele i uklanjanje svih podataka sa njom

CREATE TABLE
[*database_name*.[*owner*] . | *owner*.] *table_name*
({ < column_definition >

```
        | column_name AS computed_column_expression
        | < table_constraint > ::= [ CONSTRAINT constraint_name ] }

          | [ { PRIMARY KEY | UNIQUE } [ ,...n ]
    )

[ ON { filegroup | DEFAULT } ]
[ TEXTIMAGE_ON { filegroup | DEFAULT } ]

< column_definition > ::= { column_name data_type }
    [ COLLATE < collation_name > ]
    [ [ DEFAULT constant_expression ]
        | [ IDENTITY [ ( seed , increment ) [ NOT FOR REPLICATION ] ] ]
    ]
    [ ROWGUIDCOL]
    [ < column_constraint > ] [ ...n ]

< column_constraint > ::= [ CONSTRAINT constraint_name ]
    { [ NULL | NOT NULL ]
        | [ { PRIMARY KEY | UNIQUE }
            [ CLUSTERED | NONCLUSTERED ]
            [ WITH FILLFACTOR = fillfactor ]
            [ON {filegroup | DEFAULT} ] ]
        ]
        | [ [ FOREIGN KEY ]
            REFERENCES ref_table [ ( ref_column ) ]
            [ ON DELETE { CASCADE | NO ACTION } ]
            [ ON UPDATE { CASCADE | NO ACTION } ]
            [ NOT FOR REPLICATION ]
        ]
        | CHECK [ NOT FOR REPLICATION ]
        ( logical_expression )
    }

< table_constraint > ::= [ CONSTRAINT constraint_name ]
    { [ { PRIMARY KEY | UNIQUE }
        [ CLUSTERED | NONCLUSTERED ]
        { ( column [ ASC | DESC ] [ ,...n ] ) }
        [ WITH FILLFACTOR = fillfactor ]
        [ ON { filegroup | DEFAULT } ]
    ]
    | FOREIGN KEY
        [ ( column [ ,...n ] ) ]
        REFERENCES ref_table [ ( ref_column [ ,...n ] ) ]
        [ ON DELETE { CASCADE | NO ACTION } ]
        [ ON UPDATE { CASCADE | NO ACTION } ]
        [ NOT FOR REPLICATION ]
    | CHECK [ NOT FOR REPLICATION ]
        ( search_conditions )
    }
```

Ograničavanje prihvatljivih podataka – integritet (enforcing data integrity)

Postoje tri tipa ograničavanja prihvatanja podataka:

- Integritet domena (domain integrity)
- Integritet entiteta (entity integrity)
- Referencijalni integritet (referential integrity)

Integritet domena

Integritet domena je ograničenje podataka koje korisnici smeju da unose u polja i važe za celu kolonu podataka. Ostvaruje se preko dve naredbe:

Default - Služi za popunjavanje polja kada korisnik ga nije sam popunio (tekući sadržaj)
Check (*Check constraint*)- Definiše podatke koji su prihvatljivi u tom polju

Primer.

Vežba - Obezbeđivanje integriteta
Kreirati bazu sledećeg opisa

Baza treba da predstavlja adresar u koju će se unositi informacije sledeće sadržine: Ime i prezime, ulica i broj, mesto stanovanja, poštanski broj, broj telefona i tip telefona, kao i email. Svaki pojedinac može da ima veći broj telefona različitog tipa: fiksni, mobilni, kao i veći broj email adresa. Osoba koja ima telefon ne mora da ima email i obrnuto ali može da ima i jedno i drugo....

Adrese	Telefoni	E – mail
7. Ime	4. broj telefona	1. email
8. Prezime	5. tip telefona	
9. Ulica i broj		
10. Mesto		
11. ZIP		
12. Država		

Po formiranju tabela koristiti opcije default za mesto da je Beograd a za tip telefona da je 1. Postaviti tip provere Check constraint za proveru unosa vrednosti za poštanski broj stim da prva cifra mora biti u opsegu od 1-3, druga od 1-9, i ostale od 0-9....

1. Sintaksa pisanja kreiranje tabele za slučaj da se ne zadaju elementi integriteta, i bez polja za generisanje rednog broja:

```
CREATE TABLE adrese
(
        ime             nvarchar (15)       NULL,
        pezime          nvarchar (30)       NULL,
        ulica           nvarchar (50)       NULL,
```

```
    mesto       nvarchar (20)       NULL,
    ZIP         char (5)            NULL,
)
```

2. Sintaksa pisanja kreiranje tabele za slučaj da se ne zadaju elementi integriteta ali sa poljem za generisanje rednog broja:

```
CREATE TABLE adrese
(
    ime         nvarchar (15)       NULL,
    pezime      nvarchar (30)       NULL,
    ulica       nvarchar (50)       NULL,
    mesto       nvarchar (20)       NULL,
    ZIP         char (5)            NULL,
    ID          bigint              IDENTITY (1,1)
)
```

3. Sintaksa pisanja kreiranje tabele za slučaj da se zadaju elementi integriteta ali i sa poljem za generisanje rednog broja:

```
CREATE TABLE adrese
(
    ime         nvarchar (15)       NULL,
    prezime             nvarchar (30)       NULL,
    ulica       nvarchar (50)       NULL,
    mesto       nvarchar (20)       NULL
            DEFAULT ('Beograd'),
    ZIP         char (5)            NULL
            CONSTRAINT CK_ZIP CHECK (ZIP Like '[1-3][1-9][0-9][0-9][0-9]'),
    ID          bigint              IDENTITY (1,1)
)
```

Formiranje privremene tabele temporary

```
CREATE TABLE #Privremena (broj      int)

INSERT INTO #Privremena VALUES (1)
```

Formiranje tabele sa izračunatom kolonom

```
CREATE TABLE izracunavanje
(
    prvi        int,
    drugi       int,
    srednja_vrednost AS (prvi+drugi)/2
)
```

Formiranje tabele sa kolonom koja koristi funkciju SQL-a

```
CREATE TABLE ulaz
(
```

```
        datum           datetime,
        korisnik_id     int,
        ime_korisnika AS USER_NAME()
)
```

Sintaksa T-SQL za izmenu tabele je sledeća

```
ALTER TABLE table
{ [ ALTER COLUMN column_name
   { new_data_type [ ( precision [ , scale ] ) ]
      [ COLLATE < collation_name > ]
      [ NULL | NOT NULL ]
      | {ADD | DROP } ROWGUIDCOL }
   ]
   | ADD
      { [ < column_definition > ]
      | column_name AS computed_column_expression
      } [ ,...n ]
   | [ WITH CHECK | WITH NOCHECK ] ADD
      { < table_constraint > } [ ,...n ]
   | DROP
      { [ CONSTRAINT ] constraint_name
         | COLUMN column } [ ,...n ]
   | { CHECK | NOCHECK } CONSTRAINT
      { ALL | constraint_name [ ,...n ] }
   | { ENABLE | DISABLE } TRIGGER
      { ALL | trigger_name [ ,...n ] }
}

< column_definition > ::=
   { column_name data_type }
   [ [ DEFAULT constant_expression ] [ WITH VALUES ]
   | [ IDENTITY [ (seed , increment ) [ NOT FOR REPLICATION ] ] ]
      ]
   [ ROWGUIDCOL ]
   [ COLLATE < collation_name > ]
   [ < column_constraint > ] [ ...n ]

< column_constraint > ::=
   [ CONSTRAINT constraint_name ]
   { [ NULL | NOT NULL ]
      | [ { PRIMARY KEY | UNIQUE }
         [ CLUSTERED | NONCLUSTERED ]
         [ WITH FILLFACTOR = fillfactor ]
         [ ON { filegroup | DEFAULT } ]
         ]
      | [ [ FOREIGN KEY ]
         REFERENCES ref_table [ ( ref_column ) ]
         [ ON DELETE { CASCADE | NO ACTION } ]
```

```
        [ ON UPDATE { CASCADE | NO ACTION } ]
        [ NOT FOR REPLICATION ]
        ]
      | CHECK [ NOT FOR REPLICATION ]
        ( logical_expression )
  }

< table_constraint > ::=
  [ CONSTRAINT constraint_name ]
  { [ { PRIMARY KEY | UNIQUE }
      [ CLUSTERED | NONCLUSTERED ]
      { ( column [ ,...n ] ) }
      [ WITH FILLFACTOR = fillfactor ]
      [ ON {filegroup | DEFAULT } ]
      ]
      |   FOREIGN KEY
        [ ( column [ ,...n ] ) ]
        REFERENCES ref_table [ ( ref_column [ ,...n ] ) ]
        [ ON DELETE { CASCADE | NO ACTION } ]
        [ ON UPDATE { CASCADE | NO ACTION } ]
        [ NOT FOR REPLICATION ]
      | DEFAULT constant_expression
        [ FOR column ] [ WITH VALUES ]
      |   CHECK [ NOT FOR REPLICATION ]
        ( search_conditions )
  }
```

Komanda ALTER TABLE sadrži dve dodatne komande

1. ADD za dodavanje kolone i
2. DROP COLUMN za brisanje kolone zapisa

Primeri:

1. Tabeli email dodati kolonu provajder sa tipom nvarchar

```
ALTER TABLE email ADD provajder NVARCHAR (20) NULL
```

2. Obrisati kolonu provajder iz tabele email

```
ALTER TABLE email DROP COLUMN provajder
```

3. Tabeli adrese dodati kolonu id sa mogućnošću generisanja nejednakog broja

```
ALTER TABLE adrese ADD ID bigint INDENTITY (1,1)
```

4. Tabeli adrese izmeniti proveru integriteta na polju ZIP

```
ALTER TABLE adrese DROP CONSTRAINT CK_ZIP
```

Sintaksa T-SQL za brisanje tabele je sledeća

DROP TABLE *table_name*

Pošto je naredba vrlo jednostavna nećemo je posebno obrađivati.

INDEX – i

Indeksi su elementi SQL baze koji omogućavaju brže pronalaženje i pristup podacima, kao i reorganizaciju podataka. Formiraju se nad kolonama određene tabele.

Kolone koje sadrže *ntext, text ili image* tip podataka ne mogu biti kolone nad kojima se prave indeksi

Postoje dve vrste indeksa
1. Grupišući
2. Ne grupišući

Grupišući indeksi

Grupišući indeksi (engl. *clustered indexes*) menjaju fiziči raspored podataka koje korisnici unose. Način na koji se podaci u grupišućim indeksima raspoređuju može se porediti sa rečima u rečniku. Kada u rečniku potražite reč *firma*, prvo potražite slovo *f* pa onda pod njim slovo *i* pa dalje. Na sličan način pretražuje podatak i SQL baza kada se primenjuje grupišući indeks. Jedna tabela može da ima samo jedan grupišući indeks.
Savršeni su za kolone iz kojih se često učitavaju opsezi podataka.

Negrupišući indeksi

Slično grupišućem indeksima negrupišući (engl. *nonclustered index*) ima strukturu u oblika B-stabla, koje ima korensku stranicu, međunivoe i nivoe listova. Međutim postoje dve značajne razlike između njih.
1. Prva je što se na nivou listova u negrupišućem indeksu ne nalaze podaci već pokazivači na podatke .
2. Druga značajna razlika je da negrupišući indeks ne menja fizički redosled podataka
Ako umesto jednog podatka, tražite opseg (ili grupu) podataka, moraćete više puta da se vraćate u indeks zato što se većina podataka nalazi na različitim stranicama (što povećava vreme pretraživanja podataka).

Grupišući	Negrupišući
Može da postoji samo jedan po tabeli	Može da bude do 249 po tabeli
Fizički raspoređuje podatke u skladu sa redosledom koji je definisan indeksom	Formira odvojenu listu ključnih vrednosti sa pokazivačima ka lokacijama podataka u stranicama za podatke

Prikladniji za kolone u kojima se češće traže opsezi podataka	Prikladniji za kolone u kojima se češće traže pojedinačne vrednosti
Prikladniji za kolone sa niskom selektivnošću	Prikladniji za kolone sa visokom selektivnošću

T-SQL naredbe za Index-e

- CREATE – Kreiranje indeksa
- DROP – Brisanje indeksa

Sintaksa T-SQL za kreiranje index-a je sledeća

```
CREATE [ UNIQUE ] [ CLUSTERED | NONCLUSTERED ] INDEX index_name
   ON { table | view } ( column [ ASC | DESC ] [ ,...n ] )
[ WITH < index_option > [ ,...n] ]
[ ON filegroup ]

< index_option > :: =
   { PAD_INDEX |
      FILLFACTOR = fillfactor |
      IGNORE_DUP_KEY |
      DROP_EXISTING |
   STATISTICS_NORECOMPUTE |
   SORT_IN_TEMPDB
}
```

Primeri:

1. Kreiranje indeksa na tabeli adrese nad kolonom prezime.

```
USE adresar
IF EXISTS (SELECT name FROM sysindexes
          WHERE name='prezime_ind')
  DROP INDEX adrese.prezime_ind
GO
USE adresar
CREATE INDEX prezime_ind
  ON adrese (prezime)
GO
```

2. Kreiranje klasterisanog indeksa na tabeli adrese nad kolonom prezime

```
USE adresar
IF EXISTS (SELECT name FROM sysindexes
          WHERE name='prezime_ind')
  DROP INDEX adrese.prezime_ind
GO
USE adresar
CREATE CLUSTERED INDEX prezime_cls ON adresa (prezime)
```

3. Kreiranje jedinstvenog klasterisanog indeksa na tabeli adrese i nad kolonom prezime

```
USE adresar
IF EXISTS (SELECT name FROM sysindexes
        WHERE name='prezime_cls')
  DROP INDEX adrese.prezime_cls
GO
USE adresar
CREATE UNIQUE CLUSTERED INDEX prezime_ucls ON adresa (prezime)
```

4. Kreiranje jednostavnog složenog indeksa na tabeli adresa sa kolonama prezime i ime

```
USE adresar
IF EXISTS (SELECT name FROM sysindexes
        WHERE name='prezime_ucls')
  DROP INDEX adrese.prezime_ucls
GO
USE sdresar
CREATE INDEX prezime_ime ON adrese (prezime, ime)
```

Sintaksa T-SQL za brisanje index-a je sledeća

DROP INDEX '*table*.*index* | *view*.*index*' [,...*n*]

Primer:

```
USE adresar
IF EXISTS (SELECT name FROM sysindexes
        WHERE name='prezime_ucls')
    DROP INDEX adrese.prezime_ucls
```

Obezbeđivanje integriteta entiteta

To je postupak kojim se obezbeđuje da svaki zapis u tabeli ima jedinstven sadržaj i da nijedan od njih ne može da bude dupliran. Ostvarivanje provere entiteta je omogućeno na dva načina

1. Korišćenjem ***primarnih ključeva***
2. Korišćenjem ograničenja ***UNIQUE***

Primarni ključ

Primarni ključ se koristi da bi se obezbedilo da sadržaj svakog zapisa u tabeli na određeni način bude jedinstven (da se ne ponavlja). To se postiže pomoću specijalne vrste indeksa koji se zovu indetifikujući indeksi (engl. ***unique index***). Indeksi se obično koriste za ubrzavanje pristupa podacima tako što se učitavaju sve vrednosti u koloni i održavaju uređenu listu sa podacima. Identifikujući indeks ne samo što generiše takvu listu već i ne dozvoljava da se u indeks unesu duplirane vrednosti. Trebalo bi da se primarni ključ sastoji od jedne ili više kolona koje sadrže jedinstvene vrednosti.

Da bi se formirao primarni ključ kolona ne sme imati vrednost *NULL*. Jedna tabela može imati samo jedan primarni ključ i može se povezivati sa spoljnim ključevima – drugim tabelama.

Ograničenje tipa UNIQUE

UNIQUE – jedinstvena vrednost. Koja se ne koristi za vezu sa spoljnim ključevima kao **primary key** i može sadržati vrednost NULL koju primary key ne može. Ako se ova vrednost unese, može postojati samo jedna na nivou cele kolone. Ograničenja ovog tipa treba da koristite kada morate da sprečite da se duplirane vrednosti unose u polje koje nije deo primarnog ključa i možete ih imati veći broj u jednoj tabeli.

Primeri:

Formiranje primarnog ključa prilikom formiranja tabele.

Ukoliko je tabela adrese postojala, obrisati je.

1. Formirati primarni ključ prilikom formiranja tabele.

```
CREATE TABLE adrese
(
        ime             nvarchar (15)           NULL,
        prezime                 nvarchar (30)           NULL,
        ulica           nvarchar (50)           NULL,
        mesto           nvarchar (20)           NULL
                DEFAULT ('Beograd'),
        ZIP             char (5)                NULL
                CONSTRAINT CK_ZIP CHECK (ZIP Like '[1-3][1-9][0-9][0-9][0-9]'),
        ID              bigint                  INDENTITY (1,1) NOT NULL PRIMARY
KEY
)
```

2. Ukoliko je postojala tabela onda se primarni ključ formira na sledeći način

```
ALTER TABLE adrese ADD CONSTRAINT PK_ID PRIMARY KEY (ID)
```

3. Formirati UNIQUE vrednost nad poljem prezime, prilikom formiranja tabele

```
CREATE TABLE adrese
(
        ime             nvarchar (15)           NULL,
        prezime                 nvarchar (30)   NOT NULL UNIQUE,
        ulica           nvarchar (50)           NULL,
        mesto           nvarchar (20)           NULL
                DEFAULT ('Beograd'),
        ZIP             char (5)                NULL
                CONSTRAINT CK_ZIP CHECK (ZIP Like '[1-3][1-9][0-9][0-9][0-9]'),
```

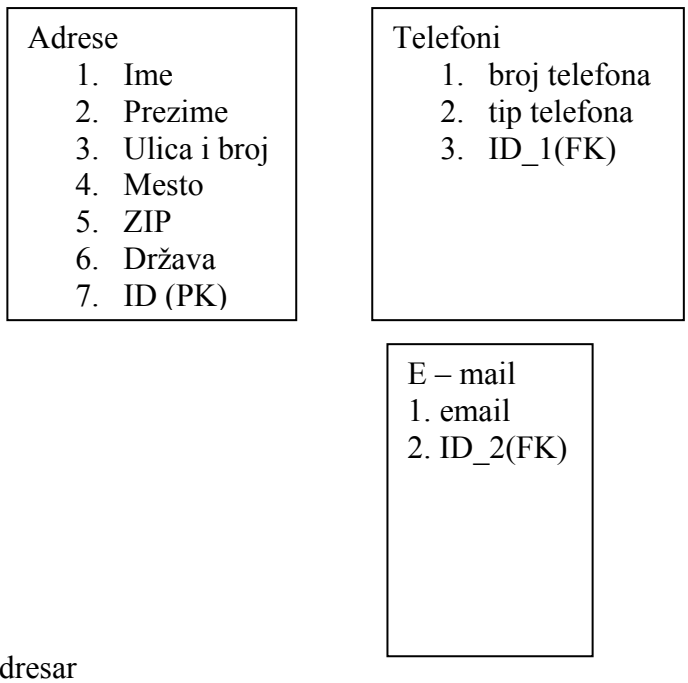
ID bigint INDENTITY (1,1) NOT NULL PRIMARY KEY
)

4. Ukoliko je postojala tabela onda se nejednaka vrednost formira na sledeći način

ALTER TABLE adrese ADD CONSTRAINT UN_prezime UNIQUE (prezime)

Referencijalni integritet

To je način zaštite povezanih podataka koji se nalaze u odvojenim tabelama. Spoljni ključ jedne tabele se povezuje sa primarnim ključem druge tabele. Preko uključivanja lančanog referencijalnog integriteta omogućeno vam je da brišete i ažurirate zapise iz tabele na strani primarnog ključa, a da se pri tome izmene koje unesete automatski prenose u tabelu na strani spoljnog ključa.
Da biste dodali spoljni ključ u tabelama telefoni i e-mail napravite negrupišuće indekse nad poljima ID_1 i ID_2.



USE adresar
IF EXISTS (SELECT name FROM sysindexes
WHERE name='IX_ID_1')
DROP INDEX telefoni.IX_ID_1
CREATE INDEX IX_ID_1 ON telefoni (ID_1)

USE adresar
IF EXISTS (SELECT name FROM sysindexes
WHERE name='IX_ID_2')
DROP INDEX email.IX_ID_2
CREATE INDEX IX_ID_2 ON email (ID_2)

USE adresar

```
IF EXISTS (SELECT name FROM sysindexes
                WHERE name='PK_ID')
        DROP INDEX adrese.PK_ID
ALTER TABLE adrese ADD CONSTRAINT PK_ID PRIMARY KEY (ID)
```

T-SQL naredba za dodavanje spoljnog ključa je:

1. Kada tabele koje treba da se spoje već postoje

```
ALTER TABLE telefoni ADD CONSTRAINT FK_adrese_telefoni FOREIGN KEY(id_1)
REFERENCES adrese (ID)
```

POGLEDI

Pogledi služe za izbor grupe podataka iz jedne ili više povezanih tabela. Takođe ta grupa može obuhvatati sve kolone i sve zapise ili samo određene kolone i određene zapise. Osnovna sintaksa pisanja je:

SELECT [**TOP** *n* [**PERCENT**]] *column list*
FROM *source_list*
| **WHERE** *search_condition*]
[**ORDER BY** *expression*]

1. Biranje svih kolona svih kolona i svih zapisa iz jedne tabele

SELECT * FROM *table_name*

Primer :
USE Northwind
SELECT * FROM *Employees*

2. Biranje svih zapisa određenih kolona.
SELECT *columns name* **FROM** *table_name*

Primer :
USE *Northwind*
SELECT *lastname, firstname, title* **FROM** *Employees*

3. Biranje svih kolona ograničenog zapisa.
SELECT * FROM *table_name* **WHERE** *conditions*

Primer:
USE *Northwind*
SELECT * FROM *Employees* **Where** *lastname* **LIKE** 'D%'

4. Biranje određeni broj zapisa i određene kolone:
SELECT *columns name* **FROM** *table_name* **WHERE** *conditions*

Primer:
USE *Northwind*
SELECT *lastname, firstname, title* **FROM** *Employees* **Where** *title* **LIKE** '%sales%'

5. Korišćenje nadimka za izabrane kolone:
SELECT *column name* **AS** *[alias]* **FROM** *table name* **WHERE** *conditions*

Primer:
USE *Northwind*
SELECT *lastname* **AS** *[prezime], firstname* **AS** *[ime], title* **FROM** *employees* **WHERE** *title* **LIKE** '%sales'

6. Korišćenje sortiranja u izabranim podacima :
SELECT *column names* **FROM** *table name* **WHERE** *conditions* **ORDER BY** *column* **ASC| DSC**

Primer :
SELECT *lastname* **AS** *prezime, firstname* **AS** *ime, title* **FROM** *employees* **WHERE** *title* **LIKE** '%sales%' **ORDER BY** *lastname* **ASC**

Prikaz skupa rezultata

SELECT TOP *n* **[PERCENT]** *columns name* **FROM** *table name* **WHERE** *conditions* **ORDER BY** *column name* **ASC| DSC**

Primer za prikaz prvih 5 zapisa

SELECT TOP *5 lastname* **AS** *prezime, firstname* **AS** *ime, title* **FROM** *employees* **WHERE** *title* **LIKE** '%sales%' **ORDER BY** *lastname* **ASC**

Primer za prikaz prvih 5% zapisa

SELECT TOP *5* **PERCENT** *lastname* **AS** *prezime, firstname* **AS** *ime, title* **FROM** *employees* **WHERE** *title* **LIKE** '%sales%' **ORDER BY** *lastname* **ASC**

Spajanje tabela.

Pošto se svi podaci ne nalaze u jednoj tabeli, kada bi hteli da vidimo podatke iz obe tabele istovremeno, morali bismo da spajamo takve tabele. Tabele se spajaju preko službene reči **JOIN**. Postoji više vrsta spojeva od kojih je najjednostavniji unutrašnji spoj.

a) *Unutrašnji spoj* – (**INNER JOIN** koji se skraćeno piše samo **JOIN**) deo je iskaza **SELECT** koji učitava zapise iz više tabela da bi dao jedan skup zapisa. Odredba **JOIN** povezuje tabele na osnovu zajedničke kolone i daje zapise čije su vrednosti poklapaju u spojenim tabelama.

Sintaksa pisanja je:
SELECT *columns name* **FROM** *table name_1* **JOIN** *table name_2* **ON** *table name_1.column name=table name_2.column name*

Primer:

USE *pubs*
SELECT *sales.qty, sales.title_id, stores.stor_name* **FROM** *sales* **JOIN** *stores* **ON** *sales.stor_id=stores.stor_id*

b) *Spoljni spoj* – (**OUTER JOIN**), postoje tri vrste spoljnih spojeva. Desni spoljni spoj (**RIGHT OUTER JOIN** koji se skraćeno piše **RIGHT JOIN**) se koristi ukoliko želite da vidite sve zapise iz tabele koja se nalazi na desnoj strani odrednice **JOIN**, bez obzira na to da li zapisi postoje ili nepostoje odgovarajući zapisi u levoj tabeli. Obrnuto se koristi **LEFT OUTER JOIN** ili **LEFT JOIN**. Ako želite da vidite sve zapise iz obe tabele koristi se potpuni spoljni spoj **FULL OUTER JOIN** ili **OUTER JOIN**.

Primer :
USE pubs
SELECT *sales.qty, sales.title_id, stores.stor_name* **FROM** *sales* **RIGHT JOIN** *stores* **ON** *sales.stor_id=stores.stor_id*

Spajanje više tabela.

Primer:
USE *pubs*
SELECT *sales.qty, titles.title, stores.stor_name* **FROM** *sales* **JOIN** *stores* **ON** *sales.stor_id=stores.stor_id* **JOIN** *titles* **ON** *sales.title_id=titles.title_idI*

Korišćenje odredbe **GROUP BY i HAVING**

Dosta čest slučaj je da osim slaganja po redu podataka, neophodno je da se prikažu i neki zbirni rezultati. Na primer da bi saznali zbirnu prodaju knjiga neke knjižare koristimo odredbu **GROUP BY**. Ova odrednica će vam dati zbirni podatak u koliko se primenjuje sa nekom agregatnom funkcijom. Agregatne funkcije su funkcije koje vam daju određene vrste zbirnih podataka, kao što su prosek ili ukupan zbir vrednosti u određenoj koloni.

Primer:

USE *pubs*
SELECT *stores.stor_name,* **SUM**(*sales.qty*) **as** *sumqty* **FROM** *sales* **JOIN** *stores* **ON** *sales.stor_id=stores.stor_id* **GROUP BY** *stores.stor_name*

Ako želite na ovako izračunatim poljima da izvršite selekciju podataka, onda se koristi odrednica **HAVING**.

Primer:

USE *pubs*
SELECT *stores.stor_name,* **SUM**(*sales.qty*) **as** *sumqty* **FROM** *sales* **JOIN** *stores* **ON** *sales.stor_id=stores.stor_id* **GROUP BY** *stores.stor_name* **HAVING SUM***(sales.qty)>=90*

Odrednica **HAVING** deluje slično kao i odrednica **WHERE** sa tom razlikom što sa odredbom **HAVING** možete da koristite i agregatne funkcije dok kod **WHERE** ne možete.

Operator **ROLLUP**

Ukoliko osim zbirnih podataka želite da vidite i podzbirove onda treba koristiti operator **ROLLUP.**

Primer:

USE *pubs*
SELECT *stores.stor_name,* sales.title_id, **SUM**(*sales.qty*) **as** *sumqty* **FROM** *sales* **JOIN** *stores* **ON** *sales.stor_id=stores.stor_id* **GROUP BY** *stores.stor_name, sales.title_id* **WITH ROLLUP ORDER BY** *stores.stor_name, sales.title_id*

Operatori **CUBE** i **GROUPING**

Ako želite da utvrdite koliko knjiga jednog naslova je ukupno prodato u svim knjižarama i koliko po pojedinim knjižarama onda u takvim slučajevima se koristi operator **CUBE**. Ovaj operator vam obezbeđuje zbirne podatke za sve moguće kombinacije kolona skupa rezultata.

Primer:

USE *pubs*
SELECT *stores.stor_name,* sales.title_id, **SUM**(*sales.qty*) **as** *sumqty* **FROM** *sales* **JOIN** *stores* **ON** *sales.stor_id=stores.stor_id* **GROUP BY** *stores.stor_name, sales.title_id* **WITH CUBE ORDER BY** *stores.stor_name, sales.title_id*

Rezultat ovakvog pretraživanja je vrlo koristan mada i donekle nerazumljiv. Da bi se razlikovali zbirni od pojedinih podataka, koristi se operator **GROUPING**. Ovaj operator se koristi i sa operatorom **CUBE** i sa operatorom **ROLLUP**, omogućava umetanje dodatnih kolona koje pokazuju da li prethodna kolona sadrži detaljan podatak (vrednost nula) ili zbirni (vrednost jedan).

Primer:

USE *pubs*
SELECT *stores.stor_name,* **GROUPING**(stores.stor_name), sales.title_id, **GROUPING**(sales.title_id), **SUM**(*sales.qty*) **as** *sumqty* **FROM** *sales* **JOIN** *stores* **ON** *sales.stor_id=stores.stor_id* **GROUP BY** *stores.stor_name, sales.title_id* **WITH CUBE ORDER BY** *stores.stor_name, sales.title_id*

Ažuriranje podataka pomoću pogleda.

Pomoću pogleda podatke možete ne samo čitati već i ažurirati. Ukoliko se opredelite za ažuriranje podataka pomoću pogleda, treba se imati na umu sledeće...

a) Ako podatke ažurirate u istom pogledu možete da menjate sadržaj samo jedne tabele koje pogled obuhvata.
b) Ne možete da ažurirate podatke pomoću pogleda koji sadrže agregatne funkcije.
c) Može nastati problem ako želite da dodate nov zapis u pogled koji ne prikazuje sve podatke....

1. Kreirajmo jednostavan pogled.

USE Northwind
Go
CREATE VIEW *CustomerView*
AS SELECT *CustomerID, CompanyName*
FROM *Customers*

2. Dodavanje podataka kroz pogled.

USE *Northwind*
INSERT *CustomerView* **VALUES** *('TEST1','Test Company')*

3. Menjanje podataka kroz pogled

Use *Northwind*
UPDATE *CustomerView*
SET *CustomerID='TEST2'* **WHERE** *CustomerID='TEST1'*

4. Brisanje podataka kroz pogled....

USE *Northwind*
DELETE *CustomerView*
WHERE *CustomerID='TEST2'*

USKLADIŠTENE PROCEDURE (stored procedure)

Uskladištena procedura je upit koji se čuva u SQL – serverovoj bazi podataka i nije ugrađena u kod neke čeone aplikacije. Zašto biste čuvali u bazi podataka na serveru? Za to postoje dva veoma dobra razloga, a prvi od njih je poboljšanje performansi.

Na koji način uskladištene procedure poboljšavaju performanse? Uzmimo na primer upit za prikaz svih autora iz baze podataka koji su iz Ouklenda. Kod za izvršavanje ovakvog upita je:

USE *pubs*
SELECT *au_fname, au_lname, address, city, state, zip*
FROM *authors*
WHERE *city='Oakland'*
ORDER BY *au_lname* **DESC**

Zamislite da 5000 korisnika šalje jedan ovakav upit, po ceo dan. To izaziva popriličan saobraćaj kroz mrežu što može izazvati zagušenje. Zagušenje nastaje kada je obim saobraćaja u mreži prevelik u odnosu na mogućnosti komponenata mreže, i podaci se u tom trenutku gube. Da bi ste dobili željene podatke, morali biste da ponovite upit, to znači da se neki podaci šalju dvaput, što značajno usporava mrežu i njene korisnike.

Da biste uklonili uska grla i obezbedili da mreža radi punom brzinom, treba da smanjite količinu koda koji se mrežom šalje od klijentskih mašina ka serveru, čime ćete smanjiti i obim saobraćaja u mreži. Da bi se to postiglo dovoljno je da kod upita smestite na server, a ne na klijentske računare, a to ćete postići ako upit pretvorite u uskladištenu proceduru. Kada se ona napravi jedini kod koji korisnici treba da pošalju kroz mrežu da bi dobili željene informacije je otprilike ovako:

EXEC *ime_uskladištene_procedure*

Druga prednost uskladištenih procedura je u poređenju sa ad hoc upitima jeste prevođenje. Kada SQL server prevodi upit, on ispisuje da li u upitu ima spajanja tabela i uslova zadatih odredbom WHERE, a zatim poredi upit sa svim raspoloživim indeksima kako bi utvrdio onaj (ako ga pronađe) koji bi korisniku najbrže obezbedio rezultate. I tako za svaki poslati upit.

Osim manjeg obima saobraćaja u mreži, korišćenje uskladištenih procedura pruža još jednu prednost, a to je jednostavnije upravljanje bazom podataka. Na primer ako treba da izmenite postojeći upit, a on se nalazi na više klijentskih mašina, istu izmenu biste morali da radite na svakoj od njih.

Uskladištene procedure koje sami pišete

Osnovne uskladištene procedure

CREATE PROC [EDURE] *procedure_name* [**;** *number*]
  [{ *@parameter data_type* }
    [VARYING] [= *default*] [OUTPUT]
  ] [**,***...n*]

[WITH
  { RECOMPILE | ENCRYPTION | RECOMPILE , ENCRYPTION }]

[FOR REPLICATION]

AS *sql_statement* [*...n*]

Najlakše se pišu uskladištene procedure koje daju jednostavan skup rezultata i nema nijedan parametar. Prethodni upit cemo pretvoriti u proceduru.

USE *pubs*
CREATE PROCEDURE *DBO.Show_Authors* **AS**
SELECT *au_fname, au_lname, address, city, state, zip* **FROM** *authors*
WHERE *city='Oakland'*
ORDER BY *au_lname* **DESC**

Da bi se koristila ova procedura u Query Analayzeru upišite sledeći kod:

USE *pubs*
EXEC *Show_authors*

Korišćenje ulaznih parametara

Ulazni parametri (***input parameters***) u uskladištenim procedurama jesu reči koje zamenjuju podatke koje korisnik treba da zada. To su memorijske promenljive jer se nalaze u memoriji i njihov sadržaj je promenljiv. U prethodnom primeru ako iyvršimo izmenu grada Oaklenda u promenljivu, moći ćete da menjate uslove poređenja i samim tim procedura postaje fleksibilnija.

USE *pubs*
CREATE PROCEDURE *DBO.Show_authors*
    @city_1 varchar (50)
AS
SELECT *au_fname, au_lname, address, city, state, zip*
FROM *authors*
WHERE *city=@city_1*
ORDER BY *au_lname* **DESC**

Pozivanjue ovakve uskladištene procedure se vrši na sledeći način:
USE *pubs*
EXEC *Show_authors 'San Jose'*

Ili

USE *pubs*
EXEC *Show_authors 'Menlo Park'*

Ukoliko biste prilikom pozivanja procedure zaboravili unos parametra, SQL server bi vam signalizirao grešku da niste uneli potreban parametar.

Prilikom definisanja ulaznog parametra možete da stavite i tekuću vrednost tih ulaznih parametara. Na primer:

USE *pubs*
go
CREATE PROCEDURE *DBO.Show_authors*
    @city_1 varchar (50) = 'Oakland'
AS
SELECT *au_fname, au_lname, address, city, state, zip*
FROM *authors*
WHERE *city=@citty_1*
ORDER BY *au_lname* **DESC**

Ako u ovom primeru prilikom poziva ove uskladištene procedure ne unesete ulazni parametar procedura će uzeti tekuću vrednost ulaznog parametra, u suprotnom procedura uzima vrednost ulaznog parametra.

Kreiranje privremenih uskladištenih procedura

Za kreiranje lokalne privremene uskladištene procedure imenu procedure u deklaraciji se dodaje znak #. Ovakva procedura se smešta u TempDB bazu. Za kreiranje globalne privremene uskladištene procedure koristimo dva znaka ## ispred imena procedure. Privremena uskladištena procedura postoji dok postoji konekcija na server.

Primer:

a) Kreiranje lokalne privremene uskladištene procedure

```
CREATE PROCEDURE #localtemp
AS
SELECT * FROM [pubs].[dbo].[authors]
GO
```

b) Kreiranje globalne privremene uskladištene procedure

```
CREATE PROCEDURE ##globtemp
AS
SELECT * FROM [pubs].[dbo].[authors]
GO
```

Brisanje uskladištene procedure

```
USE pubs
GO
DROP PROCEDURE procedure01, procedure02
```

Korišćenje izlaznih parametara

Izlazni parametar (output parameter) je obrnuti ulazni parametar. Za ulazni parametar vi zadajete vrednost koja će se koristiti u proceduri, dok za izlazni parametar sama procedura obezbeđuje njegovu vrednost, koju možete da iskoristite u drugim upitima. Izlazni parametar se definiše na istom mestu gde i ulazni, jedina razlika je u tome što izlazni parametar odmah iza svog imena postoji i odrednica **OUTPUT**.

Na primer ako napišemo jednostavnu proceduru za izračunavanje zbira dva broja.

```
CREATE PROCEDURE  DBO.calc
        @prvi int,
        @drugi int,
        @zbir int OUTPUT
AS
SET @zbir=@prvi+@drugi
```

Izvršavanje ove uskladištene procedure se vrši na sledeći način

```
USE pubs
DECLARE @answer int
EXEC calc 1, 2, @answer output
Print @answer
```

Ili **SELECT** *'Odgovor je ', @answer*

Primeri:

a) Prvi primer

```
USE Northwind
GO
CREATE PROCEDURE FiveMostExpensiveProducts
AS
SELECT TOP 5 ProductName, UnitPrice FROM Products ORDER BY UnitPrice DESC
GO
```

Aktiviranje procedure je sa…
EXEC *FiveMostExpensiveProducts*

b) Drugi primer

```
USE Northwind
GO
CREATE PROCEDURE spInsertShipper
        @CompanyName nvarchar (40),
        @Phone nvarchar (24)=null
AS
        INSERT INTO Shippers
        VALUES (@CompanyName, @Phone)
```

Aktiviranje procedure je
EXEC *spInsertShipper 'dostavljac','1235-323'*

c) Treći primer

```
USE Northwind
GO
CREATE PROCEDURE spInsertOrderDATE
        @CustomerID nvarchar (5),
        @EmployeeID int,
        @OrderDate datetime=null,
        @requiredDate datetime=null,
        @shippedDate datetime=null,
        @shipVia int,
        @Freight money,
        @ShipName nvarchar (40)=null,
        @ShipAddres nvarchar (60)=null,
        @ShipCity nvarchar (15)=null,
        @ShipRegion nvarchar (15)=null,
        @ShipPostalCode nvarchar (10)=null,
        @ShipCountry nvarchar (15)=null,
        @OrderID int OUTPUT
```

AS
/* hoću porudžbine ne starije od 7 dana*/
IF DATEDIFF*(dd, @Orderdate, getdate())>7*
SELECT *@OrderDate=null*
INSERT INTO *orders*
VALUES (@customerID, @EmployeeID, @OrderDate, @RequiredDate, @ShippedDate, @ShipVia, @Freight, @ShipName, @ShipAddress, @ShipCity, @ShipRegion, @ShipPostalCode, @ShipCountry)

/* Vrednost identiteta se ubacuje u izlaznu promenljivu*/
SELECT *@OrderID=@@IDENTITY*

Korisničke definisane funkcije

Korisnički definisane funkcije, slično uskladištenim procedurama predstavljaju uređeni niz komandi T-SQL –a. Dok sa uskladištenim procedurama možemo da damo parametre i da vratimo vrednosti u parametrima, u korisničkim funkcijama možemo samo da damo vrednosti parametara ali i ne da ih vratimo. Dva osnovna tipa korisničkih funkcija su:

1. One koje vraćaju skalarne vrednosti
2. One koje vraćaju tabelu

One koje vraćaju skalarne vrednosti

CREATE FUNCTION *[owner_name.] function_name*
(*[{ @parameter_name [AS] scalar_parameter_data_type [= default] } [,...n]]* ***)***

RETURNS *scalar_return_data_type*

[WITH < function_option> [[**,**] ...*n*]]

[AS]

BEGIN
function_body
RETURN *scalar_expression*
END

1. Primer

USE *Northwind*
GO
CREATE FUNCTION *fn_FormatDatuma*
(@indate datetime, @separator char (1))
RETURNS *nchar (20)*
AS
BEGIN
RETURN

CONVERT *(nvarchar(20), datepart (dd, @indate))+ @separator*
+CONVERT *(nvarchar (20), datepart (mm, @indate)) + @separator*
+CONVERT *(nvarchar (20), datepart (yy, @indate))*
END

Korišćenje funkcije dato je u sledećem kodu.

USE *Northwind*
DECLARE *@datum nvarchar (80)*
SET *@datum=dbo.fn_FormatDatuma (getdate(),'.')*
PRINT *@datum*
Ili
SELECT *dbo.fn_FormatDatuma(**GETDATE()**,'.')*

2. Primer

USE *Northwind*
GO
CREATE FUNCTION *fn_NewRegion*
(@myinput nvarchar (30))
RETURNS *nvarchar (30)*
BEGIN
IF *@myinput is NULL*
SET *@myinput=' Region nije raspoloziv'*
RETURN *@myinput*
END

Aktiviranje primera br.2

PRINT *dbo.fn_NewRegion (NULL)*
PRINT *dbo.fn_NewRegion ('Beograd')*

SELECT *lastname, City, dbo.fn_NewRegion(Region)* **AS** *Region, Country* **FROM** *Employees*

One koje vraćaju tabelu

CREATE FUNCTION *[owner_name.] function_name*
([{ @parameter_name [AS] scalar_parameter_data_type [= default] } [,...n]])

RETURNS TABLE

[**WITH** < *function_option* > *[[,] ...n]]*

[AS]

RETURN *[( ] select-stmt [ )]*

3. Primer- fumkcija koja vraća tabelu.

CREATE FUNCTION *fn_CustomerNamesInRegion*
(@RegionParameter nvarchar (30))
RETURNS *table*
AS
RETURN (
SELECT *CustomerID, CompanyName* **FROM** *Northwind.dbo.customers* **WHERE** *Region=@RegionParameter*
)

SELECT * FROM *fn_CustomerNamesInRegion ('WA')*

OKIDAČI
(trigger)

Okidači su grupa **SQL** iskaza koja izgleda i oponaša se veoma slično uskladištenim procedurama. Jedina bitna razlika između njih je što olidač ne možete da pozivate pomoću komande **EXEC**, već se okidač automatski aktivira kad god korisnik pokuša da unese nov podatak ili da ažurira ili izbriše postojeći.

Postoje pravila koja su veoma česta u poslovnom svetu, ali čije se poštovanje ne može obezbediti povezivanjem tabela pomoću spoljnih ključeva niti pomoću dozvola za tabele. Jedino okidači mogu da obezbede pravilnu primenu složenije poslovne logike.

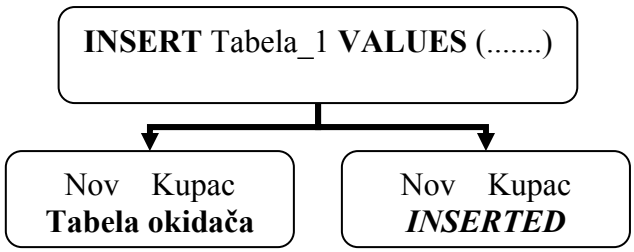
SQL server pruža dve različite vrste okidača: okidače **AFTER** i **INSTEAD OF**. Okidači **AFTER** se izvršavaju posle izvršavanja komande za koju je deklarisan, a **INSTEAD OF** se pozivaju umesto komande.

Okidači **AFTER** možemo napraviti za komande **INSERT, UPDATE i DELETE**. Okidači **AFTER** mogu da se naprave za tabele, ne i za poglede.

Okidači **INSTEAD** možemo da definišemo za komande **INSERT, UPDATE i DELETE**. Mogu da se deklarišu za poglede.

Način rada okidača

Čim korisnik pokuša da unese neki podatak **SQL** server kopira novi zapis u tabelu baze podataka, koja se zove tabela okidača, kao i u specijalnu tabelu koja se drži u memoriji i zove se *iserted*.



To znači da novi zapis postoji u dve tabele u tabeli okidača i tabeli inserted. Trebalo bi da je zapis u tabeli inserted potpuno jednak sa onim u tabeli okidača. Tabela *inserted* je izuzetno značajna kada želite da izmene lančano prosledite u druge tabele baze podataka.

Na primer, pretpostavimo da u bazi podataka postoji tabela kupaca (customers), tabela porudžbina (orders) i tabela proizvoda (products). Kad god nešto prodate kupcu, količinu prodatog artikla treba da oduzmete od količine u tabeli proizvoda da bi zalihe bile ažurirane.

Primer u SQL bi bio:

UPDATE *p*
SET *p.instock=(p.instock-i.qty)*
FROM *Products p* **JOIN** *inserted i* **ON** *p.prodid=i.prodid*

Sintaksa kreiranja okidača.

CREATE TRIGGER *trigger_name*
ON *tble_or_view*
trigger_type command_list
AS
SQL_statments

Primer pisanja okidača tipa **AFTER**

CREATE TRIGGER *dbo.alerter*
ON *dbo.employees*
FOR INSERT, UPDATE, DELETE

Primer pisanja okidača tipa **INSTEAD OF**

CREATE TRIGGER *dbo.alerter*
ON *dbo.employees*
INSTEAD OF INSERT, UPDATE, DELETE

Primer :

Formirati bazu podataka, sa imenom zaposleni. U toj bazi kreirati dve tabele
Prva tabela sa imenom ljudi sledeće arhitekture.

```
CREATE TABLE [Ljudi] (
        [Ime] [nvarchar] (15)  NOT NULL ,
        [Prezime] [nvarchar] (30) NOT NULL ,
        [Red_br] [smallint] NOT NULL ,
        [prisutan] [bit] NOT NULL CONSTRAINT [DF_Ljudi_prisutan] DEFAULT (1),
        [ulaz] [nvarchar] (10) NULL ,
        [sifra] [nvarchar] (10) NULL ,
         )
```

Druga tabela sledeće arhitekture

```
CREATE TABLE [sar] (
        [Imena] [nvarchar] (100) NULL ,
```

```
	[Red_broj] [smallint] NOT NULL ,
	[prisutan] [bit] NOT NULL CONSTRAINT [DF_sar_prisutan] DEFAULT (1),
	CONSTRAINT [PK_sar] PRIMARY KEY  CLUSTERED
	(
		[Red_broj]
	)  ON [PRIMARY]
) ON [PRIMARY]
```

Kreiranje okidača koji će biti nad tabelom ljudi da kad god se unese novi saradnik, njegovo ime i prezime da se unese to isto u tabelu sar a nad poljem imena da se dobije kombinacija imena i prezimena.

Kreiranje okidača za ubacivanje saradnika u tabelu ljudi je sledeće:

```
CREATE TRIGGER dodaj ON dbo.Ljudi
FOR insert
AS
INSERT INTO sar (Imena,red_broj, prisutan)
SELECT i.prezime+' ' + i.ime,i.red_broj,i.prisutan FROM inserted i
```

Uneti u tabelu ljudi tri do četiri zapisa informacija o ljudima.

Okidač tipa DELETE

Ova vrsta okidača služi za ograničavanje podataka koje korisnici mogu da brišu iz baze podataka kao i za formiranje pravila brisanja. Kao i kod umetanja zapisa i ovom prilikom SQL server pravi u memoriji tabelu koja se zove deleted (d), što znači da zapis još nije sasvim izbrisan i da ga još možete referencirati u svom kodu. To je veoma korisno za razne slučajeve složene poslovne logike.

Kreiranje okidača za brisanje saradnika iz tabele ljudi je sledeće:

```
CREATE TRIGGER obrisi ON dbo.Ljudi
FOR delete
AS
delete p
FROM sar p JOIN deleted d ON p.Red_broj=d.red_br WHERE p.red_broj=d.red_br
```

Druga mogućnost je ograničavanje brisanja podataka nad tabelom.

```
CREATE TRIGGER PR_brisi ON ljudi
FOR DELETE
AS
IF (SELECT prisutan FROM deleted)=1
   BEGIN
   PRINT ' Ne može se izbrisati zaposleni koji je prisutan'
   PRINT ' Brisanje je otkazano'
   ROLLBACK
END
```

Okidači tipa UPDATE

Okidač tipa **UPDATE** ograničava delovanje iskaza **UPDATE** koje korisnici šalju. Ova vrsta okidača je projektovana tako da ograničavaju podatke koje korisnici smeju da ažuriraju. Metod koji primenjuje okidač tipa **UPDATE**, kombinacija je metoda koji se koriste u okidačima **INSERT** i **DELETE**. Okidači tipa **INSERT** koriste tabelu inserted a okidači tipa **DELETE** koriste tabelu deleted, dok okidač tipa **UPDATE** koristi obe tabele. To je zato što se operacija ažuriranja zapravo sastoji od brisanja postojećeg i dodavanje novog zapisa.

Primer okidača update nad tabelom ljudi:

CREATE TRIGGER *izmeni* **ON** *dbo.Ljudi*
FOR *update*
AS
UPDATE *p*
SET *p.imena=i.prezime + ' ' + i.ime, p.prisutan=i.prisutan, p.red_broj=i.red_br* **FROM** *sar p* **JOIN** *ljudi i* **ON** *p.Red_broj=i.red_br*

Primer ograničenja izmena nad tabelom

CREATE TRIGGER *ogranicenje* **ON** *ljudi*
FOR *UPDATE*
AS
IF (**SELECT** *Red_br* **FROM** *inserted)=133*
    BEGIN
    PRINT *'Ne može se izmeniti postojeći zapis'*
    PRINT *'Izmena je otkazana'*
    ROLLBACK
    END

Brisanje okidača

Brisanje okidača se izvodi preko naredbe **DROP TRIGGER** *ime01, ime02*

Okidači tipa INSTEAD OF

U koliko želite da vršite ažuriranje tabele preko pogleda a u pogledu nisu data sva polja bez okidača **INSTEAD OF** ta ažuriranja mogu da dožive neuspeh. Na primer, formirali ste pgled nad tabelom ljudi ali samo sa dve kolone i to ime i prezime.

CREATE VIEW *dbo.pogled*
AS
SELECT *Ime, Prezime*
FROM *dbo.Ljudi*

Pokušajte da preko ovog pogleda dodate novog saradnika. I to preko sledećog koda.

INSERT *pogled*
VALUES *('mirjana','radović')*

U Query Analayzeru napravite novi okidač

```
CREATE TRIGGER add_novog ON pogled
INSTEAD OF INSERT
AS
DECLARE
        @ime nvarchar (15),
        @prezime nvarchar (30),
        @red_br smallint

        SET @red_br=150
        SET @ime=(SELECT ime FROM INSERTED)
        SET @prezime=(SELECT prezime FROM INSERTED)
INSERT ljudi (ime, prezime, red_br) VALUES (@ime, @prezime, @red_br)
```

Da bi ste testirali rad unesite sledeći kod.

```
INSERT pogled
VALUES ('mirjana','radović')
```

Pregled da li je unešena nova vrednost se aktivira iz Query Analayzera preko upita
SELECT * FROM *ljudi*