

Administracija Baza podataka

Objekti Relacione baze podataka

Objekti Baze podataka

- ❑ Indeksi
- ❑ Pogledi
- ❑ Proceduralna proširenja SQL-a (Transact-SQL)
- ❑ Trigeri
- ❑ Korisničke funkcije (*User Defined Functions*)
- ❑ Uskladištene procedure (*Stored procedures*)

Pogledi - VIEW

- ❑ Pogled je “prozor” kroz koji se vide podaci baze podataka
 - Pojednostavljuje korišćenje baze podataka
 - Može se koristiti za zaštitu podataka od neovlašćenog pristupa
 - Sa aspekta performansi – pogledi se čuvaju u kompiliranom obliku
- ❑ Pogled se može posmatrati kao pdšema neke šema Relacione baze podataka.

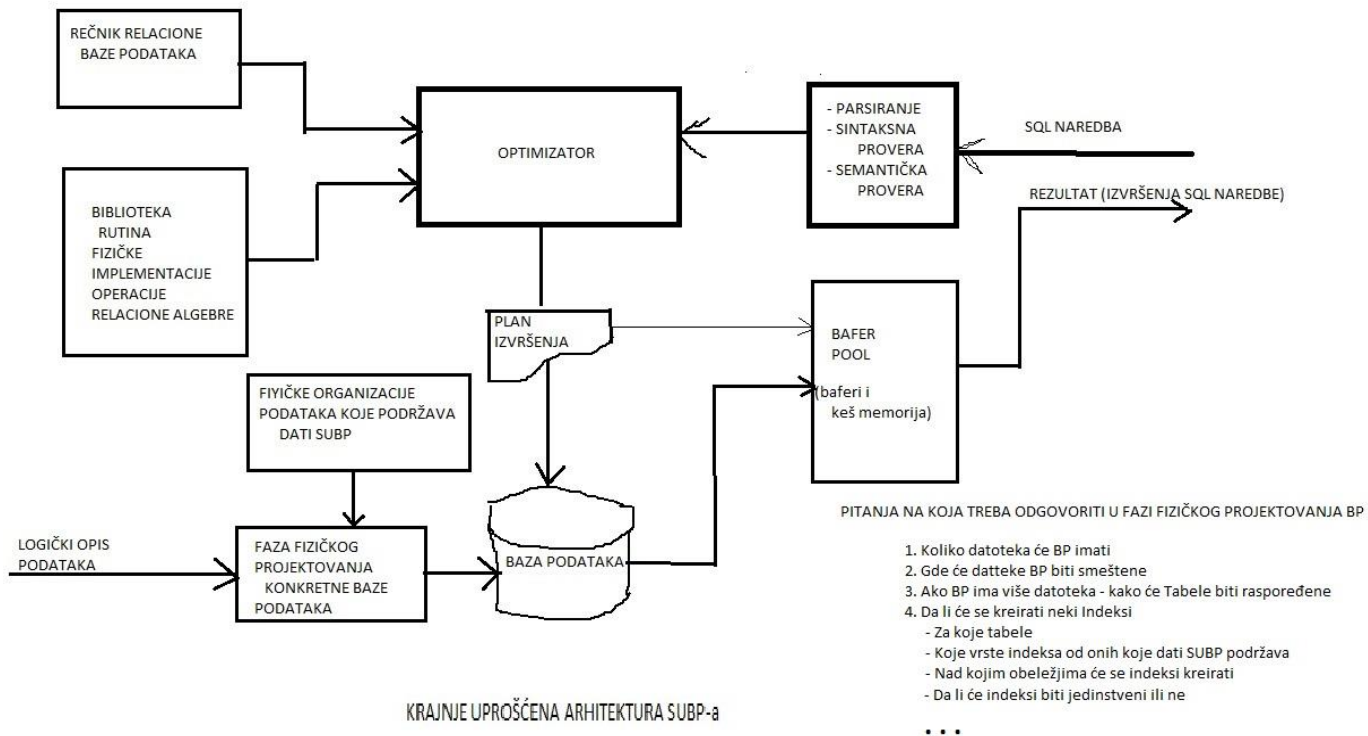
Sintaksa naredbe za kreiranje pogleda

```
CREATE VIEW <naziv_pogleda>  
    [(naziv_obl1 [, naziv_obl2] , . . .)]
```

AS Podupit

- Podupit – Standardna Select SQL naredba (koju smo obradili iz predmeta Osnove baza podataka)

Pogled (VIEW) - šta je suštinski



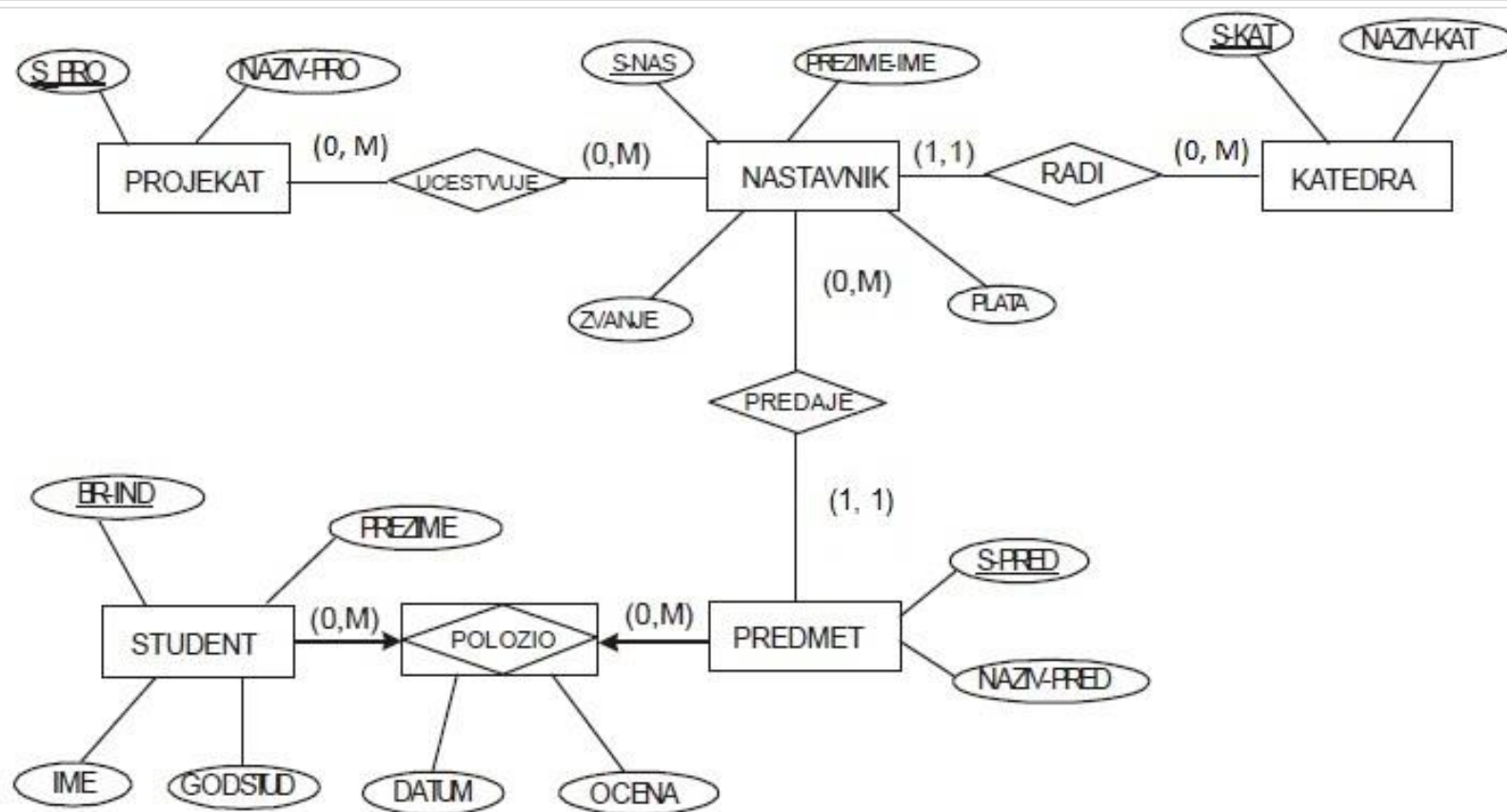
Pogledi (View) - primer

Kreirati pogled `NastavnikPredmet` kroz koji se "vide":

```
IdNas  Prezime_Ime  Naziv  FondCasova  
-----
```

1. Formiranje i testiranje upita
2. Kreiranje pogleda
3. Korišćenje pogleda

Konceptualna šema Baze podataka Fakultet



Kreiranje pogleda - NastavnikPredmet

1. Kreiranje podupita

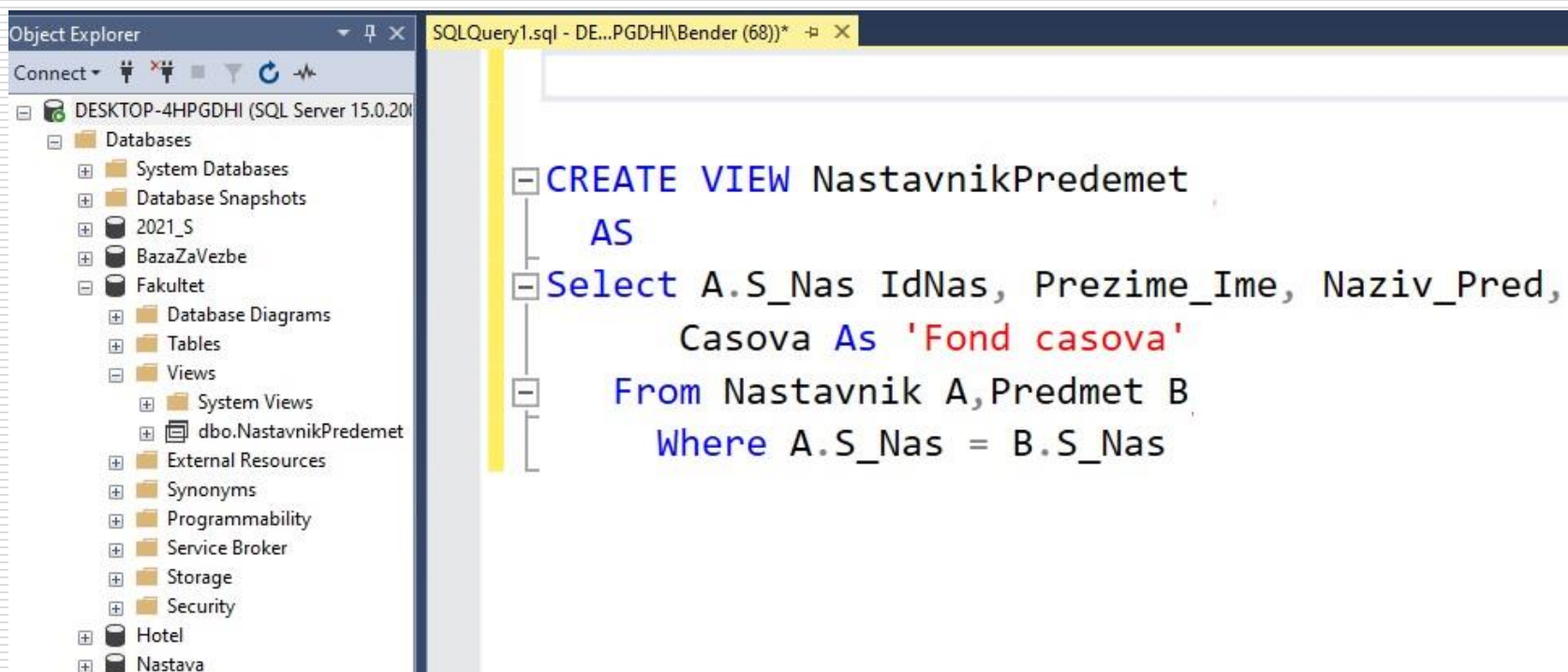
```
Select A.S_Nas IdNas, Prezime_Ime, Naziv_Pred,  
       Casova As 'Fond casova'  
From Nastavnik A,Predmet B  
Where A.S_Nas = B.S_Nas
```


Kreiranje pogleda - NastavnikPredmet

Kreiranje pogleda

```
CREATE VIEW NastavnikPredmet
AS
Select A.S_Nas IdNas, Prezime_Ime,
       Naziv_Pred,
       Casova As 'Fond casova'
From Nastavnik A, Predmet B
Where A.S_Nas = B.S_Nas
```

Kreiranje pogleda



The screenshot displays the SQL Server Enterprise Manager interface. On the left, the 'Object Explorer' pane shows the database structure for 'DESKTOP-4HPGDHI (SQL Server 15.0.2000.13)'. The 'Databases' folder is expanded, showing 'System Databases', 'Database Snapshots', '2021_S', 'BazaZaVezbe', and 'Fakultet'. Under 'Fakultet', the 'Views' folder is expanded, showing 'System Views' and 'dbo.NastavnikPredemet'. The right pane shows the SQL query editor with the following code:

```
CREATE VIEW NastavnikPredemet
AS
Select A.S_Nas IdNas, Prezime_Ime, Naziv_Pred,
      Casova As 'Fond casova'
From Nastavnik A,Predmet B
Where A.S_Nas = B.S_Nas
```

Korišćenje pogleda

- Pogledi se koriste kao i bazne tabele.
- Korisnik ne zna da li koristi baznu tabelu ili pogled (odnosno izvedenu tabelu).

Na primer:

```
Select *  
      From  
          NastavnikPredemet
```

Uklanjanje pogleda

`DROP VIEW <Naziv_Pogleda>`

Ukloniti iz baze pogled pod nazivom
NastavnikPredmet.

`Drop View NastavnikPredmet`

Proceduralni mehanizmi za kontrolu integriteta

Proceduralni mehanizmi za kontrolu integriteta relacione baze podataka

- Proceduralni mehanizmi za proveru uslova integriteta se, u sistemima za upravljanje bazom podataka, najčešće realizuju korišćenjem
 - Okidača (*trigger*),
 - Procedura (*stored procedures*) I
 - Funkcija baze podataka (*user defined functions*)

Okidači (trigeri)

- Okidač je mehanizam SUBP koji se automatski pokreće i izvršava svaki put kada se izvrši određena operacija ažuriranja nad tabelom za koju je vezan triger.
- Okidači se najčešće koriste za realizaciju poslovnih pravila koja se tiču provere integriteta podataka i/ili obezbedjenja konzistentnog stanja baze podataka

Okidači (trigeri)

- Pri kreiranju okidača obavezno se navode predmet i vrsta ažuriranja koja bi trebala da aktivira okidač.
- Predmet ažuriranja je uvek neka tabela (ili pogled), a pod vrstom ažuriranja se podrazumevaju različite operacije ažuriranja, kao što su unos, modifikacija ili brisanje podataka.
- Uobičajeno je da se za svaku operaciju ažuriranja nad nekom tabelom definiše poseban okidač, mada je teoretski moguće koristiti isti okidač za sve tri operacije ažuriranja.
- Ono što ne može, jeste da se za istu operaciju ažuriranja nad jednom tabelom definiše više okidača.

Okidači (trigeri)

- Okidač obično služi za proveru ili izmenu podataka pomoću odgovarajućih SQL iskaza i ne bi trebao da vraćati neku vrednost(i) korisniku.
- DDL naredbe koji se koriste za rad sa okidačima su:
 - `CREATE TRIGGER,`
 - `ALTER TRIGGER i`
 - `DROP TRIGGER`

Okidači (trigeri)

```
CREATE TRIGGER trigger_name
ON { table | view }
{ {FOR | AFTER | INSTEAD OF } { [DELETE] [,]
    [INSERT] [,] [UPDATE] }
AS
    [ { IF UPDATE ( column )
        [{ AND | OR } UPDATE ( column )]
        [ ...n ] } ]
sql_statement [ ...n ]
}
```

Okidači (trigeri)

FOR | AFTER | INSTEAD OF

- Ključne reči koje odredjuju vreme aktiviranja okidača u odnosu na operaciju žuriranja. Navodi se jedna od tri opcije.

Okidači (trigeri)

FOR | AFTER

- `FOR` i `AFTER` imaju isti efekat.
- Ključna reč `FOR` se koristi zbog kompatibilnosti sa prethodnim verzijama SUBP.
- Danas se u osnovi razlikuju tri vrste okidača: `AFTER`, `BEFORE` i `INSTEAD OF` od kojih SQL - Server podržava samo 2 vrste (`AFTER` i `INSTEAD OF`).
- `AFTER` okidač se aktivira tek nakon uspešnog izvršenja operacija ažuriranja. Ukoliko se pojavi neki problem pri izvršenju operacije ažuriranja okidač se neće ni pokrenuti.

Okidači (trigleri)

INSTEAD OF

- Govori da će se okidač izvršavati umesto okidajućeg SQL iskaza.
- Koristi se kada se izvršenje operacije ažuriranja uslovljava ispunjenošću nekih preduslova, pa se u telu okidača prvo proverava ispunjenost ovih uslova pa tek onda izvršava sama operacija ažuriranja.
- Ovo ažuriranje se vrši na osnovu sadržaja logičkih tabela koje se automatski formiraju pri aktiviranju okidača.

Okidači (trigери)

[DELETE] [,] [INSERT] [,] [UPDATE]

- Ključne reči koje specificiraju koje operacije ažuriranja podataka nad tabelom ili pogledom aktiviraju okidač.
- Najmanje jedna opcija mora biti navedena.

AS

- Ključna reč koja označava početak tzv. tela okidača u kome se navode akcije koje okidač treba da preduzme pri sopstvenom aktiviranju.

Okidači (trigeri)

Telo okidača

sql_statement

- Okidači mogu sadržati proizvoljan broj i vrste Transact - SQL iskaza. Pored generičkih SQL iskaza (DDL i DML iskazi) u okidaču se mogu koristiti i naredbe kontrole toka kao i druge naredbe svojstvene struktuiranim programskim jezicima (naredbe dodeljivanja vrednosti (SET), ...).
- Rad sa okidačima je povezan sa korišćenjem specijalnih (implicitnih) tabela:
 - deleted i
 - inserted

Okidači (trigери)

- deleted i
- inserted
- su logičke (konceptualne) tabele. One su strukturalno identične tabeli nad kojom je okidač definisan i sadrže stare ili nove vrednosti n-torki koje su obuhvaćene operacijom ažuriranja koja je aktivirala okidač.
- Na primer, da bi se pretražile sve vrednosti u deleted tabeli, treba koristiti sledeću naredbu:

Primer:

```
SELECT * FROM deleted
```

Okidači (trigери)

IF UPDATE (column)

- Testira da li je izvršena INSERT ili UPDATE operacija nad specificiranim kolonom. Može se navesti više kolona. UPDATE(column) se može koristiti bilo gde unutar tela okidača.
- Ukoliko postoje ograničenja definisana nad triggerovanom tabelom, ona se proveravaju posle izvršenja INSTEAD OF okidača, odnosno pre izvršenja AFTER okidača.
- Ukoliko dodje do narušavanja ograničenja, akcije INSTEAD OF okidača se poništavaju (rollback), dok se AFTER okidač uopšte ne izvršava.

Proceduralna proširenja SQL-a

- ❑ Deklaracija promenljivih
- ❑ Naredbe za kontrolu toka programa
 - IF..... ELSE naredba
 - WHILE naredba
- ❑ Kursor i kursori tip promenljive
 - Deklarisanje Kursora (DECLARE naredba)
 - Otvaranje kursora (OPEN naredba)
 - Čitače podataka iz kursora (FETCH naredba)
 - Zatvaranje kursora (CLOSE naredba)
 - Uklanjanje ukursora (DEALLOCATE naredba)

Deklaracija promenljivih

```
DECLARE {{ @local_variable data_type }  
        | { @cursor_variable_name CURSOR }  
        } [ ,...n]
```

- Promenljivama se vrednost dodeljuje pomoću SET ili SELECT iskaza.
- Kursorske varijable se mogu koristiti samo u iskazima koji se tiču rada sa kursorima.
- Nakon deklaracije sve varijable imaju NULL vrednost.

Naredbe za kontrolu toka

IF Boolean_expression

{ sql_statement | statement_block }

[ELSE

{ sql_statement | statement_block }]

Boolean_expression

- Je izraz koji vraća TRUE ili FALSE. Ukoliko sadrži SELECT iskaz isti se mora navesti u zagradama.

Naredbe za kontrolu toka

{sql_statement | statement_block}

- Ukoliko se IF ili ELSE grana odnosi samo na jedan SQL iskaz ovaj iskaz se može navesti odmah iza IF ili ELSE.
- Ukoliko se u nekoj grani mora izvršiti više od jedne naredbe ove naredbe treba grupisati kao blok iskaza.
- Blok iskaza se definiše pomoću ključnih reči BEGIN (za označavanje početka) i END (za označavanje završetka bloka).

Kursor i kursorški tip promenljive

- Postoje situacije kada je posle izvršenog pretraživanja rezultat potrebno obradivati red po red.
- Sistemi za upravljanje bazama podataka u svojim proceduralnim proširenjima SQL-a omogućavaju deklarisanje kursora ili kursorških promenljivih koje obezbeđuju prihvatanje rezultata kada on sadrži više od jedne n-torke.
- Otvaranje kursora nad skupom rezultata omogućava obradu podataka red po red.

Deklarisanje kursora

```
DECLARE cursor_name CURSOR
  [ LOCAL | GLOBAL ]
  [ FORWARD_ONLY | SCROLL ]
  [ STATIC | KEYSET | DYNAMIC |
  FAST_FORWARD ]
  [ READ_ONLY | SCROLL_LOCKS | OPTIMISTIC
  ]
  [ TYPE_WARNING ]
  FOR select_statement
  [ FOR UPDATE [ OF column_name [ ,...n ] ] ]
```

Cursor

```
DECLARE cursor_name [ INSENSITIVE ] [  
    SCROLL ] CURSOR  
FOR select_statement
```

- Rezultat bilo koje SELECT naredbe predstavlja neki skup n-torki.
- Ukoliko rezultujući skup čini samo jedna n-torka, tada će vrednost promenljivih, za koje je u SELECT klauzuli specificirana dodela vrednosti, biti jednaka vrednosti odgovarajućih izraza u rezultujućoj n-torci.

Cursor

- Ako je rezultujući skup n-torki prazan skup, tada se promenljivima dodeljuje NULL vrednost.
- Medjutim, ako se rezultujući skup sastoji od više n-torki, tada će vrednost promenljivih po izvršenju SELECT naredbe biti jednaka vrednosti odgovarajućih izraza u poslednjoj selektovanoj n-torci.
- Na ovaj način se gubi informacija o vrednostima izraza za sve ostale n-torke rezultujućeg skupa.
- Ukoliko je neophodno znati vrednosti za sve selektirane n-torke, a postoji mogućnost da rezultujući skup čini više n-torki, koriste se kursori.

Cursor

Rad sa kursorima obično podrazumeva sledeće korake:

1. DECLARE - Deklaracija kursora pri kojoj se kursoru pridružuje neka SELECT naredba i definišu karakteristike kursora (npr. da li se n-torke u kursoru mogu modifikovati ili ne).
2. OPEN - Otvaranje kursora koje podrazumeva izvršavanje pridruženog SELECT iskaza i popunjavanje kursora n-torkama rezultujućeg skupa.
3. FETCH - Pregled n-torki u kursoru (pristup narednoj ili, eventualno, prethodnoj n-torki se naziva fetch-om).
4. Opciono, modifikaciju tekuće n-torke
5. CLOSE - Zatvaranje kursora.

Cursor

INSENSITIVE

- Definiše kursor koji pravi privremenu kopiju podataka koje koristi kursor.
- Kursor se nakon punjenja isključivo koristi podacima iz ove privremene tabele (koja se kreira u tempdb bazi), te se izmene koje su u medjuvremenu učinjene nad baznim tabelama ne reflektuju na podatke koji se dobijaju pregledom kursora.
- Ovakav kursor ne dozvoljava modifikaciju n-torki rezultujućeg skupa.
- Ukoliko se izostavi ključna reč INSENSITIVE, (commit-ovana) brisanja i modifikacije nad baznim tabelama će se reflektovati pri narednim fetch-evima.

Cursor

SCROLL

- Specificira da su sve fetch opcije (FIRST, LAST, PRIOR, NEXT, RELATIVE, ABSOLUTE) na raspolaganju. Ukoliko nije naveden SCROLL jedina podržana fetch opcija je NEXT.

select_statement

- Je standardni SELECT iskaz koji definiše rezultujući skup kursora.

READ ONLY

- Onemogućuje vršenje modifikacija na osnovu sadržaja kursora (koje je, inače, dozvoljeno po default-u).
- Kursor se ne može referencirati u WHERE CURRENT OF klauzuli UPDATE ili DELETE iskaza.

Cursor

UPDATE [OF column_name[,...n]]

- Definiše kolone koje se mogu modifikovati unutar kursora.
- Ukoliko se koristi OF column_name [,...n], modifikacije je moguće vršiti samo nad navedenim kolonama. Ukoliko se navede samo FOR UPDATE bez navodjenja liste kolona, sve kolone se mogu modifikovati

Cursor

OPEN cursor_name

- Otvaranje kursora podrazumeva izvršavanje SQL naredbe definisane prilikom deklaracije kursora i podrazumeva popunjavanje kursora n-torkama rezultujućeg skupa izvršene SQL naredbe.
- Po otvaranju kursora aktuelna n-torka je prva n-torka rezultujućeg skupa.

Cursor

FETCH

```
[ [ NEXT | PRIOR | FIRST | LAST ]  
{ { [ GLOBAL ] cursor_name } | @cursor_variable_name }  
[ INTO @variable_name [ ,...n ] ]
```

FETCH omogućava pristup n-torkama koje se nalaze u kursoru.

- Pomoću FETCH naredbe možemo biti sigurni da će svaka n-torka koja se nalazi u kursoru proći određene modifikacije zadate u telu procedure ili trigera.

NEXT predstavlja defaultnu fetch opciju.

- Podrazumeva prisutp svakoj n-torci respektivno. Prvim fetch-om pristupa se prvoj n-torci, zatim drugoj... Postupak se ponavlja sve dok se ne dodje do poslednje n-torke.

Cursor

PRIOR - podrazumeva prisutp, obradu, predhodne n-torke.

- Kursor se pozicionira tako da uvek pristupa predhodnoj n-torci i za nju daje rezultate. U slučaju da se radi o prvom fetch-u (pristup prvoj n-torci) kao rezultat promenljive će imati null vrednost i kursor će se pozicionirati na prvu n-torku. Tek pristupom drugoj n-torci, doći će do obrade prve n-torke.

FIRST – pozicioniranje na prvu n-torku u kursoru.

LAST – pozicioniranje na poslednju n-torku u kursoru

Cursor

`CLOSE {cursor_name | cursor_variable_name}`

- Zatvara otvoren kursor što podrazumeva pražnjenje kursora (oslobadja se memorijski prostor koji se koristio za čuvanje podataka kursora), ali njegova deklaracija ostaje i dalje aktuelna, te se zatvoren kursor uvek može ponovo otvoriti.

`DEALLOCATE { cursor_name | @cursor_variable_name }`

- Poništava deklaraciju kursora. Ukoliko se ovo ne bi uradilo prilikom sledećeg aktiviranja procedure ili okidača bi nastao problem usled nemogućnosti ponovne deklaracije kursora sa istim nazivom.

Cursor

@@FETCH_STATUS

- Predefinisana (sistemska) varijabla koja vraća status poslednjeg FETCH iskaza nad aktivnim kursorom.

| Return vrednost | Opis |
|-----------------|---|
| 0 | FETCH iskaz je bio uspešan. |
| -1 | FETCH iskaz nije uspeo ili je n-torka izvan rezultujućeg skupa. |
| -2 | Fetch-ovana n-torka ne postoji. |

WHILE naredba

WHILE Boolean_expression

{ sql_statement | statement_block }

[BREAK]

{ sql_statement | statement_block }

[CONTINUE]

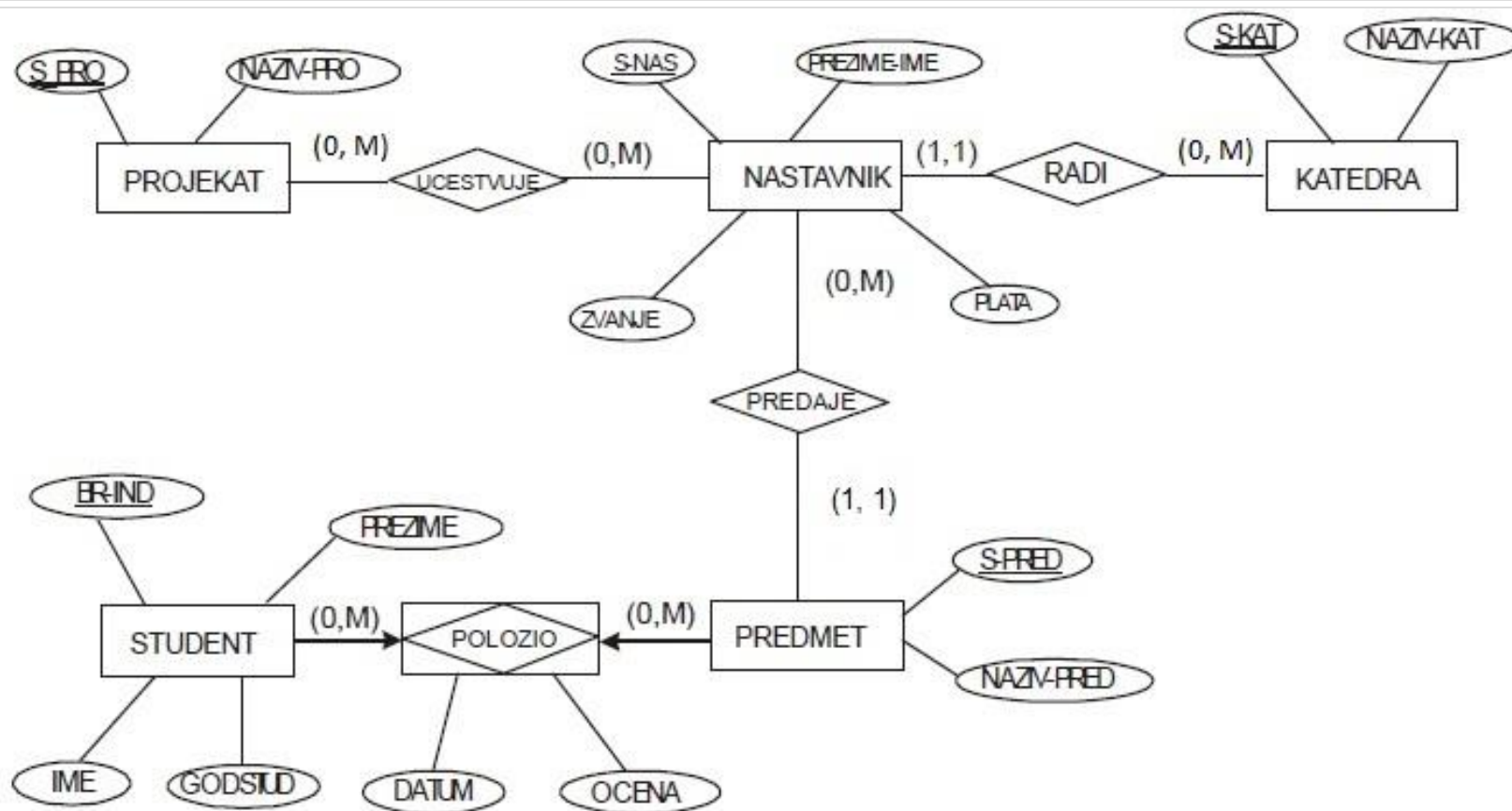
- Izvršenje bloka koda se ponavlja sve dok je uslov ispunjen.
- U nekim situacijama postoji potreba za postavljanjem dodatnih uslova unutar bloka koda koji mogu dovesti do nasilnog prekida izvršavanja WHILE ciklusa (BREAK).
- U slučaju navodjenja iskaza CONTINUE nasilno se prekida dalje izvršenje ostatka bloka koda i inicira se ponovna provera ispunjenosti opšteg uslova izvršenja ciklusa

CURSOR - Zadatak

Zadatak: Koristeci kursor napraviti proceduru kojom se iz baze podataka Fakultet prikazuje tabela sa sledecim zaglavljem:

| Prezime | Ime | Brlspita |
|---------|-----|----------|
| ----- | | |

ER Model – BP Fakultet



CURSOR – Zadatak - Resenje

```
Declare @Prezime Varchar(20), @Ime Varchar(20), @BrIspita int;
```

```
DECLARE PolozeniIspiti_Cursor SCROLL CURSOR FOR  
    SELECT Prezime, Ime, Count(*) As BrIspita  
        FROM Student A, Polozio B  
        WHERE A.Br_Ind = B.Br_Ind  
    Group By Prezime, Ime  
    Order By BrIspita Desc
```

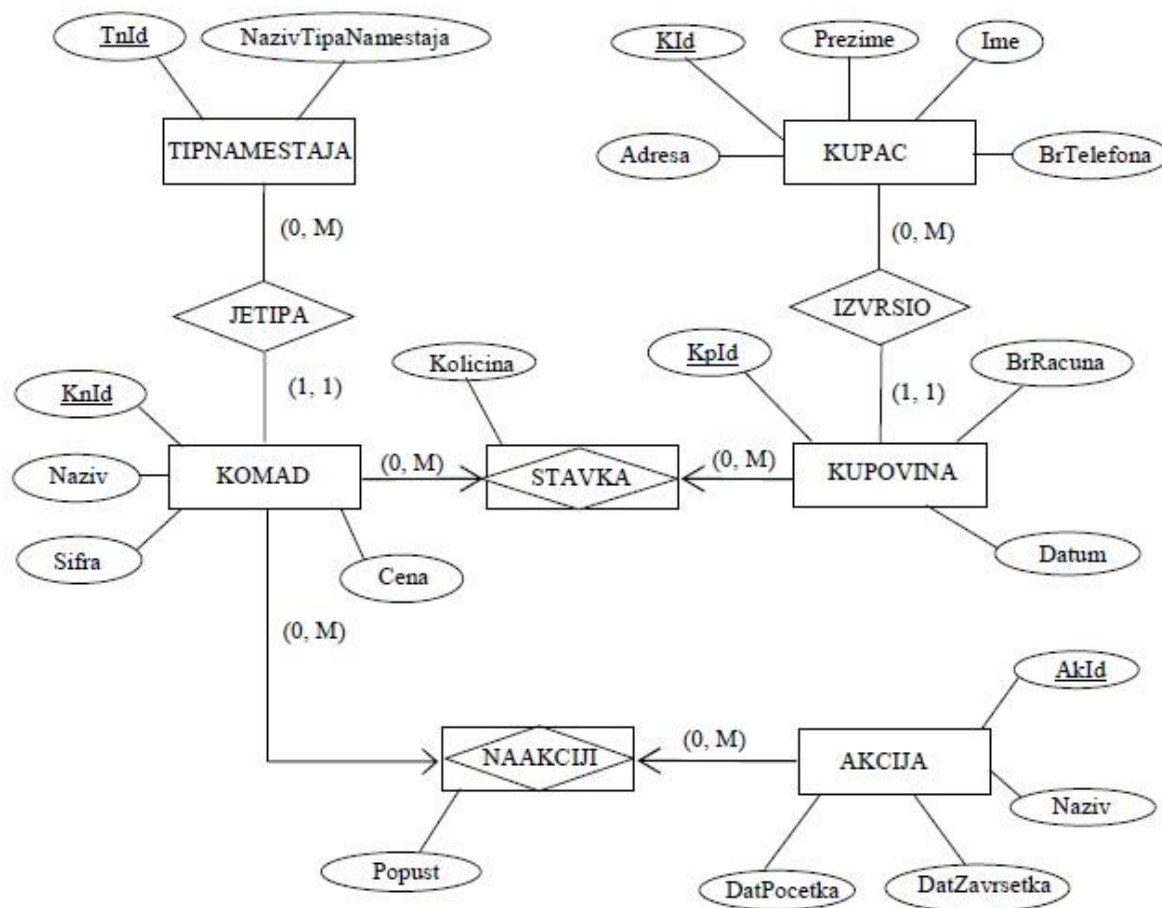
CURSOR — Zadatak - Resenje

```
OPEN PolozeniIspiti_Cursor;
Set @Prezime='Prezime'
Set @Ime='Ime'
Print @Prezime+' '+@Ime+' '+BrIspita'
FETCH PolozeniIspiti_Cursor INTO @Prezime, @Ime, @BrIspita;
Print '-----'
Print @Prezime+' '+@Ime+' '+Convert(Varchar,@BrIspita)
WHILE @@FETCH_STATUS = 0
BEGIN
    FETCH NEXT FROM PolozeniIspiti_Cursor
    INTO @Prezime, @Ime, @BrIspita
    Print @Prezime+' '+@Ime+' '+Convert(Varchar,@BrIspita)
END;
```

Trigger – primer

Zadatak 1: Kreirati trigger Komad_Ins za proveru ispunjenosti referencijalnog uslova integriteta u odnosu na relaciju TipNamestaja, pri dodavanju n-torke u relaciju Komad.

Konceptualna šema BP - SALON



Trigger – primer – (Rešenje - **Zadatak 1**)

```
Create Trigger Komad_Ins
    On Komad After Insert
As
If (Select Count(*)
    From TipNamestaja A, Inserted
    Where A.TnId = Inserted.TnId)= 0
Begin
    Raiserror ('Vrednost TnId ne postoji u tabeli
TipNamestaja.', 16, 1)
    Rollback Transaction
    Return
End
Return
```

Triger - Primer

Zadatak 2: Kreirati triger TipNamestaja_Del za proveru ispunjenosti referencijalnog uslova integriteta u odnosu na relaciju Komad, pri brisanju n-torke u relaciju TipNamestaja.

Zadatak resiti tako da se, ukoliko se brise n-torka iz tabele TipNamestaja za koju postoje n-torke u tabeli Komad, “kaskadno” brisu i sve n-torke iz tabele Komad koje se referisu na n-torku Tabele TipNamestaja koja se brise.

Triger - Primer

```
Create Trigger TipNameestaja_Del
    On TipNameestaja After Delete
As
    Delete From Komad
        Where TnId In (Select TnId From Deleted)
```

Triger – Primer 3

Zadatak 3: Kreirati triger nad tabelom Komad koji ce, pri brisanju n -torki iz tabele, prikazati sve obrisane n-torke.

Triger – Primer 3 - Resenje

```
Create Trigger PrikaziOrisano  
    On Komad After Delete  
As  
    Select *  
        From Deleted
```

TRIGERI

```
CREATE TRIGGER trigger_name
ON { table | view }
{ {FOR | AFTER | INSTEAD OF } { [DELETE] [,]
  [INSERT][,][UPDATE] }
AS
  [ { IF UPDATE ( column )
    [{ AND | OR } UPDATE ( column )]
    [ ...n ] } ]
sql_statement [...n ]
}
```

Trigger - primer

```
Create Trigger Komad_Ins
    On Komad After Insert
As
If (Select Count(*)
    From TipNamestaja A, Inserted
    Where A.TnId = Inserted.TnId)= 0
Begin
    Raiserror ('Vrednost TnId ne postoji u tabeli
TipNamestaja.', 16, 1)
    Rollback Transaction
    Return
End
Return
```

Funkcije - Korisničke

- Kao i funkcije u programskim jezicima, SQL Server korisnički definisane funkcije su rutine koje prihvataju parametre, izvršavaju različite iskaze i vraćaju rezultate tih iskaza kao vrednosti.
- Povratna vrednost može biti jedna skalarna vrednost ili skup rezultata.
- Postoje systemske i korisnički definisane funkcije baze podataka.
- Systemske funkcije su uskladištene u master bazi i ne mogu se menjati.
- Kao primeri sistemskih funkcija mogu se navesti funkcije COUNT, SUM, ROUND, GETDATE,...

Funkcije

Bilo da se radi o sistemskim ili korisnički definisanim funkcijama, funkcije se mogu podeliti na:

a)

Skalarne - korisnički definisane skalarne funkcije vraćaju jednu vrednost podataka kao rezultat funkcije. Ukoliko telo funkcije sadrži više od jednog iskaza isti se moraju navesti kao blok iskaza.

Povratna vrednost može biti bilo koji tip podataka, osim teksta, nteksta, slika, kursora i vremenskih tipova podataka.

b)

Table valued funkcije – predstavljaju korisnički definisane funkcije koje vraćaju tabele kao rezultat rada funkcije.

DDL naredbe za definisanje skalarne funkcije

```
CREATE FUNCTION NazivFunkcije
    ( [@parametar1 datatype1 [ = default ]
      [, @parametar2 datatype2 [ = default ]] [,...n] ]
      -- Ovde se dodaju ostali parametri funkcije
    )
RETURNS Data_Tipe - Tip povratne vrednosti
AS
BEGIN
-- Telo funkcije
-- Ovde se dodaje kod tela funkcije
RETURN PovratnaVrednostFunkcije;
END;
```

Funkcije

Primer funkcije koja vraca skalarnu vrednost:

```
CREATE FUNCTION nVrednost (@Puti TinyInt)
    RETURNS Int
AS
BEGIN
    DECLARE @Umnozak Int
    SET @Umnozak = @Puti * 5
    RETURN (@Umnozak)
END
```

Funkcije

Primer poziva (koriscenja) funkcije koja vraca skalarnu vrednost:

```
Select dbo.nVrednost (2)
```

Funkcija vraca vrednost $5 * 2 = 10$

DDL naredbe za definisanje tabelearne funkcije

```
CREATE FUNCTION NazivFunkcije
    (@Parametar1 TipPodatka, @Parametar2 tipPodatka)
RETURNS TABLE
AS
RETURN
(
    SELECT Elm-Selekcije-1, Elm-Selekcije-2....
        FROM NazivTabele
        WHERE Uslov
)
```

Funkcije – Primer Table valued funkcije

Primer funkcije koja kao povratnu vrednost vraca tabelu:

```
CREATE FUNCTION funIspitiStudenta (@Indeks char(6))  
    RETURNS TABLE  
AS  
    RETURN (Select NazivPredmeta,  
        Convert(varchar, Datumispita, 104) As DatumIspita, Ocena  
        From StudentPolozio  
        Where BrInd = @indeks)
```

Funkcije –Primer poziva/koriscenja **Table valued funkcije**

Koristeći funkciju funIspitiStudenta prikazati položene ispite studenta s brojem indeksa 'E 7398'.

```
Select *  
      From dbo.FunIspitiStudenta('E 7398')
```

Funkcije - Zadaci

Zadatak 1:

Kreirati funkciju koja vraća broj položenih ispita za studenta sa zadatim brojem indeksa. Funkciju nazvati:

PolozenoIspita.

Funkcije - Zadaci

```
Create Function PolozenoIspita (@BrInd Char(6))  
    Returns Integer
```

```
AS
```

```
    Begin
```

```
        Declare @BrIspita Int
```

```
        Select @BrIspita = COUNT(*)
```

```
            From Polozio
```

```
                Where Br_Ind = @BrInd
```

```
        Return @BrIspita
```

```
    End
```

Procedure - Funkcije - Zadaci

Zadatak 2:

Kreirati funkciju ProsecnaOcena koja vraća prosečnu ocenu tokom studija za studenta sa zadatim brojem indeksa.

Funkcije - Zadaci

```
Create FUNCTION ProsecnaOcena (@BrInd Char(6))
    RETURNS Decimal(4,2)
AS
BEGIN
    Declare @Prosecna Decimal(4,2)
    Select @Prosecna = AVG(Cast(Ocena As Decimal(4,2)))
        From Polozio
        Where Br_Ind = @BrInd
    Return @Prosecna
END
```

Funkcije - Zadaci

Zadatak 3:

Koristeći funkcije `PolozenoIspita` i `ProsecnaOcena` kreirati uskladistenu proceduru `SpisakStudenata` koja vraca: Prezime, Ime studenta, Broj položenih ispita i prosečnu ocenu studija za svakog studenta koji je položio najmanje jedan ispit.

Funkcije - Zadaci

```
Select Prezime, Ime, dbo.PolozenoIspita(Br_Ind)
      As Polozeno, dbo.ProsecnaOcena(Br_Ind)
      As ProsecnaOcena
From Student
      Where dbo.PolozenoIspita(Br_Ind) > 0
Order By Polozeno Desc
```

Procedure - Funkcije - Zadaci

Zadatak: Koristeci kursor napraviti Korisnicku uskladistenu proceduru pod nazivom Spisak kojom se iz baze podataka Student prikazuje tabela sa sledecim zaglavljem:

| Prezime | Ime | Brlspita |
|---------|-----|----------|
| ----- | | |

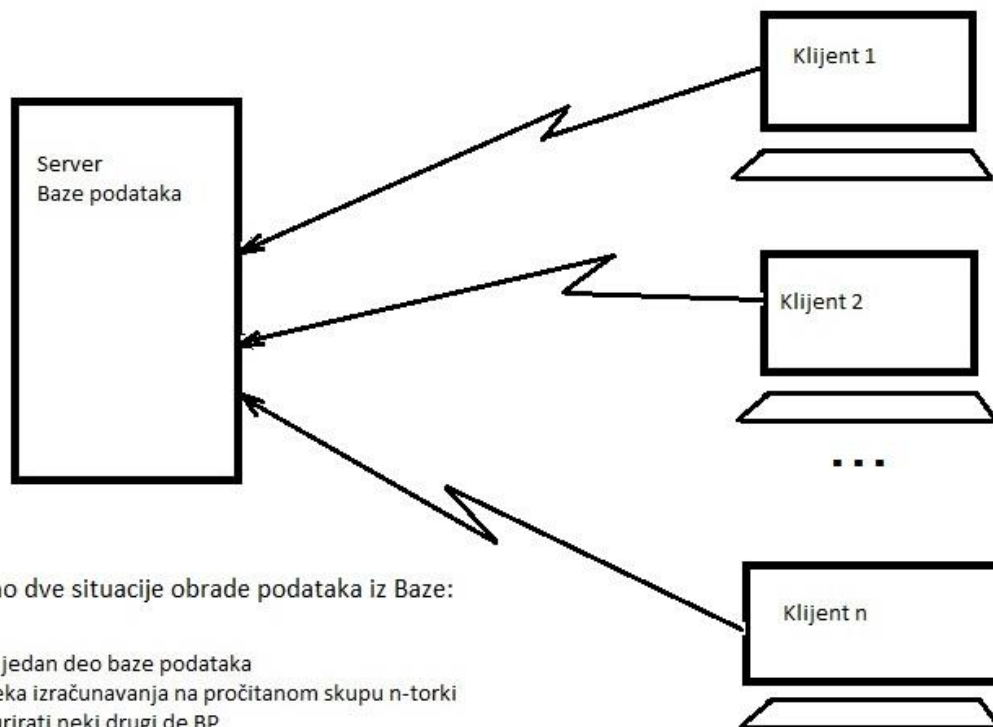
Uskladištene procedure

- Korisničke
- Systemske

Uskladištene procedure

- Uskladištene procedure se mogu definisati kao kolekcija Transact - SQL iskaza koje mogu da vrate određene rezultate na osnovu prosledjenih parametara od strane korisnika.
- Uskladištene procedure se prevashodno koriste za neke obrade koje najčešće podrazumevaju čitnje podataka iz jednog dela BP, izračunavanja i ažuriranje sadržaja nekog drugog dela BP (obračun kamata na bankarske račune, obračun zarada,...).
- Ovi postupci obrade se mogu vršiti i pomoću programa pisanim u nekom od programskih jezika koji se izvršavaju na klijentima ali je prednost korišćenja uskladištenih procedura u tome što se one u nekim situacijama obrade (ali ne svim) mnogo brže izvršavaju.

Prednosti korišćenja Uskladištenih. Procedure



Analizirajmo dve situacije obrade podataka iz Baze:

1. Treba

- Pročitati jedan deo baze podataka
- Izvršiti neka izračunavanja na pročitanoj skupu n-torki
- Zatim ažurirati neki drugi deo BP

2. Štampa nekog izveštaja na osnovu sadržaja BP

Primer 2: Korišćenje Uskladištene procedure ne donosi prednosti

1. - Prenos podataka (rezultujućih n-torki) kroz mrežu na klijenta
- Obrada/izračunavanje se vrši na strani klijenta
- Vršiti se prenos kroz mrežu prenos podataka za ažuriranje

Primer 2: Opravdano korišćenje Uskladištene procedure jer se smanjuje mrežni saobraćaj u oba smera i od servera BP i ka serveru BP. Osim toga obrada se vrši u adresnom prostoru servera BP koji je po pravilu mnogo jača mašina.

Uskladištene procedure

- Kod procedura baza podataka postoje dve vrste parametara:
 - Ulazni (input) i
 - Izlazni (output) parametri.
- Ulazni parametri su korisnički prosledjeni parametri koji služe za zadavanje uslova u samom telu procedure. Pomoću njih zadaju se određeni uslovi koji će izdvojiti n-torke za obradu.
- Izlazni parametri su parametri koji se vraćaju izvršavanjem tela procedure. Oni predstavljaju prikaz rezultata obrade podataka pomoću uskladištenih procedura.

Uskladištene procedure

Postoje dve osnovne vrste procedura

- Systemske procedure i
- Korisnički definisane procedure baze podataka.

Uskladištene procedure

Sistemske procedure:

- U SQL Serveru, mnoge administrativne aktivnosti mogu se obavljati pomoću sistemskih procedura.
- Svaki put kada se doda ili izmeni tabela, napravi rezervna kopija baze podataka (backup), kreira login ili korisnik, dodeljuju korisnička prava ili obavljaju druge administratorske aktivnosti , zapravo se pozivaju sistemske procedure specijalno napisane da izvrše željenu aktivnost.
- Sistemske procedure imaju prefiks **sp_** i čine deo SQL Server instalacije. Čuvaju se u **master** bazi podataka i dostupne su za korišćenje u bilo kojoj bazi podataka konkretne instance SQL Server-a.

Uskladištene procedure

Korisnički definisane procedure:

- Korisnički definisana procedura je svaki program napisan od strane Administratora baze podataka u okruženju SUBP-a uz korišćenje SQL naredbi i njegovih proceduralnih proširenja.
- Najčešći zadatak korisničkih Uskladištenih procedura je ažuriranje sadržaja jednih tabela na osnovu obradjenog sadržaja drugih tabela baze podataka.
- Za razliku od sistemskih procedura, korisnički definisane procedure se čuvaju u lokalnoj bazi i samo su u njoj dostupne za korišćenje.

Uskladištene procedure – Osnovna sintaksa

```
CREATE PROC [ EDURE ] procedure_name [;number]
    [ { @parameter data_type } [ OUTPUT ] ]
    [ ,...n ]
AS sql_statement [ ...n ]
```

procedure_name – definisanje naziva procedure koja se kreira

@parameter data_type – navodjenje parametara koji će se koristiti u telu procedure

OUTPUT – naglašavanje output parametra

AS - ključna reč koja služi za označavanje početka tela procedure

Uskladištene procedure

- Svi Transact - SQL iskazi koji se mogu navesti u telu okidača i funkcija stoje na raspolaganju i pri pisanju tela uskladištene procedure.
- Za razliku od okidača (**Triger**-a) kod procedura ne postoje inserted i deleted logičke tabele, jer se procedure kao takve ne vezuju za konkretnu tabelu..

Primer - Uskladištene procedure

ZADATAK: Napisati uskladištenu proceduru ProcPolozeniIspiti koja za studenta sa zadatim brojem indeksa prikazuje polozene ispite studenta, i to: NazivPredmeta, DatumPolaganja i dobijenu Ocenu.

```
Create Procedure ProcPolozeniIspiti @BrojIndeksa Char(10)
As
    Select NazivPredmeta, FORMAT(Datum, 'dd.MM.yyyy') As Datum, Ocena
    From PolozeniIspiti
    Where BrInd = @BrojIndeksa
```


Primer poziva uskladištene procedure

-- Primer> Poziv procedure koja ima ulazne parametre:

1. Nacin:

```
Exec ProcPolozeniIspiti @BrojIndeksa = 'E 7398'
```

2. Nacin:

```
Exec ProcPolozeniIspiti 'E 7398'
```

Ako ima vise ulaznih parametara pri pozivu procedure parametri se razdvajaju zarezom.

Uskladištene procedure - Primer -

ZADATAK: Koristeci kursor napraviti proceduru

Procedura_SpisakStudentata kojom se
iz baze podataka Fakultet prikazuje tabela sa sledećim
Zaglavljem.

Prezime

Ime

BrIspita

Uskladištene procedure - Primer -

```
Create Procedure Procedura SpisakStudenata
AS
  Declare @Prezime Varchar(20), @Ime Varchar(20), @BrIspita int;
  DECLARE PolozeniIspiti_Cursor SCROLL CURSOR FOR
    SELECT Prezime, Ime, Count(*) As BrIspita
    FROM Student A, Polozio B
    WHERE A.Br_Ind = B.Br_Ind
    Group By Prezime, Ime
    Order By BrIspita Desc

  OPEN PolozeniIspiti_Cursor;
  Set @Prezime='Prezime'
  Set @Ime='Ime'
  Print @Prezime+' '+@Ime+' '+BrIspita

  FETCH PolozeniIspiti_Cursor INTO @Prezime, @Ime, @BrIspita;
  Print '-----'

  Print @Prezime+' '+@Ime+' '+Convert(Varchar,@BrIspita)

  WHILE @@FETCH_STATUS = 0
  BEGIN
    FETCH NEXT FROM PolozeniIspiti_Cursor INTO @Prezime, @Ime, @BrIspita
    Print @Prezime+' '+@Ime+' '+Convert(Varchar,@BrIspita)
  END;
  Close PolozeniIspiti_Cursor;
  Deallocate PolozeniIspiti_Cursor
```