



UNIVERZITET U NIŠU  
ELEKTRONSKI FAKULTET



# Uskladištene procedure i Kursori

Sistemi za upravljanje bazama podataka

Student:

Dušica Milanović, br. ind. 1059

Mentor:

doc. dr Aleksandar Stanimirović

Niš, april 2020.

## SADRŽAJ

<b>Uvod .....</b>	<b>3</b>
<b>Uskladištene procedure .....</b>	<b>4</b>
<b>Parametrizacija .....</b>	<b>6</b>
<b>Blokovi naredbi .....</b>	<b>7</b>
<b>Vrste uskladištenih procedura .....</b>	<b>8</b>
<b>Kursori .....</b>	<b>15</b>
<b>Ugneždeni kursori .....</b>	<b>19</b>
<b>Zaključak .....</b>	<b>22</b>
<b>Literatura .....</b>	<b>23</b>

# Uvod

Informacioni sistem omogućava organizaciju podataka koja obezbeđuje procese i informacije, korisne članovima informacionog sistema i njegovim klijentima. Kao takav, informacioni sistem treba da omogućiti da se svi ti procesi obavljaju što efekasnije i brže. Informacioni sistem je u stvari slika nekog realnog sistema čiji je cilj da unapredi taj realni sistem.

Postoje tri razloga za primenu informacionih tehnologija u nekoj organizaciji, koji su u direktnoj vezi sa osnovnim ulogama informacionog sistema, koje on može imati za tu organizaciju. To su: podrška poslovnim procesima i aktivnostima organizacije, podrška u donošenju odluka i podrška strategiji u realizaciji konkurentskih prednosti.

U ovom radu predstavljena su detaljna objašnjenja Uskladištenih procedura i Kursora sa primerima koji su rađeni u prethodno kreiranoj bazi podataka *Studenti*.

# Uskladištene procedure

Uskladištena procedura (*stored procedures*) je SQL kod koji se čuva na serveru i nije sastavni deo lokalne aplikacije koja se instalira na lokajnoj mašini. Koncept uskladištenih procedura (*stored procedures*) predstavlja jedan od najznačajnijih alati programera i administratora baza podataka. U suštini, uskladištene procedure predstavljaju skup prevedenih SQL funkcija (instrukcija), koje su smeštene (uskladištene) u samu bazu podataka. Pošto su smeštene u samu bazu podataka, a ne u korisničke aplikacije (na front-end kraju), ovakav skup instrukcija se brže izvršava jer se proces prevođenja instrukcija vrši na SQL serveru. Uskladištene procedure uglavnom sadrže logičke skupove instrukcija, koji se često upotrebljavaju. Njihovom upotrebom, programeri se oslobađaju mnogostrukih pozivanja istih komandi. Još jednu od prednosti predstavlja to što se sve procedure nalaze na jednom mestu, a ne na više mesta u aplikacijama (na front-end kraju), pa je njihova izmena i ažuriranje mnogo lakše.

Veoma su moćne i preko njih mogu da se izvršavaju sve operacije iz DDL<sup>1</sup>-a i DML<sup>2</sup>-a kao, na primer, kreiranje tabele, izvršavanje **UPDATE** iskaza nad više tabela, umetanje, brisanje podataka ali i postavljanje vrednosti (**SET**) kao i prihvatanje transakcije (**COMMIT**) ili vraćanje baze u predhodno stanje (**ROLLBACK**).

Generalno, uskladištene procedure rade kao i procedure u programskim jezicima. Uskladištena procedura je imenovani objekat baze podataka i čuva se na strani servera gde se i izvršava, a klijentu se prosleđuju samo rezultati. Prilikom davanja privilegija, dovoljno je dati privilegiju za pokretanje procedure; nije potrebno davati posebna ovlašćenja za pojedinačne tabele koje se koriste u okviru nje. Sama procedura može da vrati parametre, result set, kod i da kreira kursore. Takođe može da sadrži ulazne parametre, lokalne promenljive (varijable), numeričke operacije i operacije nad karakterima, operacije dodeljivanja, SQL operacije i logiku za kontrolu toka izvršavanja.

SQL procedura se kreira **CREATE PROCEDURE** iskazom, a sa komandne linije se poziva sa **EXECUTE** (ili samo **EXEC**) naziv\_procedure iskazom.

Nakon kreiranja, skladištenja procedura i njihovih kasnijih pozivanja, SQL komande se sekvencijalno izvršavaju, a rezultati obrade šalju se po završetku procedure. Na ovaj način izbegava se i velika „gužva“ na mreži. Prevođenje procedura vrši se pri njihovom prvom pozivanju. U sledećim redovima videćemo primer jedne jednostavne uskladištene procedure:

```
CREATE PROCEDURE Daj_Studente          /* Ime procedure */
@kl int                                /* Deklarisanje varijable */
as
SELECT Kadeti.Prezime, Kadeti.Ime      /* SELECT instrukcija*/
FROM Kadeti INNER JOIN Klase
ON Klase.ID_Klase=Kadeti.ID_Klase
WHERE Klase.Red_br= @kl
ORDER BY Klase.ID_Klase=Kadeti.ID_Klase
```

---

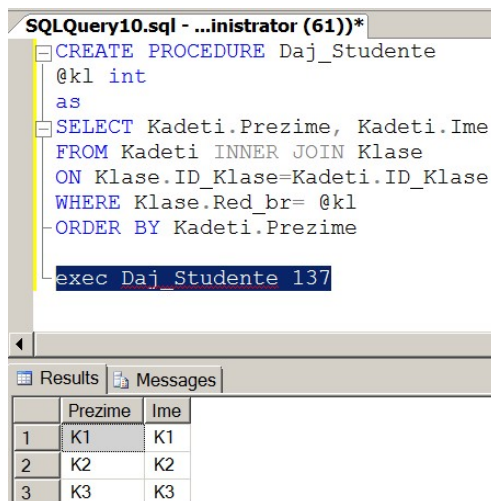
<sup>1</sup> Data Definition Language (DDL) - koristi se za kreiranje, modifikovanje ili brisanje kompletnih objekata baze podataka kao što su tabele

<sup>2</sup> Data Manipulation Language (DML) - koristi se za postavljanje upita ili modifikaciju podataka unutar baze podataka

U ovom primeru smo u proceduru dodali samo jednu **SELECT** instrukciju (iako ona može da sadrži veliki broj naredbi). Na osnovu ovoga, SQL server će kreirati jedan objekat koji će smestiti u odgovarajuću bazu podataka. Pozivanje uskladištene procedure vršimo na sledeći način:

```
exec Daj_Studente 137
```

Pozivanjem ove procedure vратиće nam se podaci koji su izdvojeni **SELECT** naredbom. Rezultat izvršenja ove procedure prikazan je na sledećoj slici:



Slika 1 – Prikaz rezultata izvršene procedure

Ovo je vrlo jednostavan primer procedure, koje inače mogu biti vrlo složene. Procedurama se mogu prosleđivati parametri i promenljive, a mogu se pozivati i iz drugih procedura.

Razlozi za korišćenjem:

1. poboljšanje performansi (često izvršavanje upita može izazvati zagušenje u mreži te se upit čuva na serveru u obliku uskladištene procedure)

```
EXEC ime_uskladistene_procedure
```

2. prevođenje (prevode se samo pri prvom izvršavanju)
3. jednostavnost upravljanja (izmene se vrše samo na jednom mestu)

Svrha uskladištenih procedura je automatizacija koda i brzina rukovanja podacima. Poenta je zapravo u brzini rukovanja podacima od strane aplikacije koja rukuje bazom. Umesto da aplikacija, koja rukuje sa bazom, šalje na server i izvršava sekvencijalno SQL naredbe, ona može samo proslediti parametar nekoj uskladištenoj proceduri na serveru, koja posle može sama uraditi kompletan proces, bez daljeg kontakta sa aplikacijom. Dakle, aplikacija je, umesto x broja SQL naredbi, prosledila serveru samo jedan parametar.

Prvo što SQL server radi je proveru sintakse. Da li je sintaksno dobro napisana naredba. Ukoliko jeste, server prevodi u niz instrukcija koje treba da urade to što treba da uradi i tek onda je izvršava i vrati rezultat. I svaki put kada pokrećemo taj upit svaki put se prolazi kroz taj algoritam. Kod uskladištenih procedura kada smestim istu naredbu **select \* from kadeti** u uskladištenu

proceduru i prvi put izvršim tu proceduru, prvo se proverava sintaksa, nakon toga prevodi u niz sql instrukcija, izvrši i vrati rezultat. Ali svaki naredni put kada pozivamo proceduru preskačemo prva tri koraka i odmah izvršava i prikazuje rezultat. Nema potrebe da jednom proverenu i parsiranu proceduru ponovo sintaksno proverava i parsira. Time značajno dobijamo na performansama.

Što se tiče naredbe **ALTER**, njome je moguće izmeniti jedino strukturu procedure, ali ne i njeno telo.

*Primer 1:* Više u telu procedure nemamo **CREATE**, već **ALTER**.

```
USE [Studenti]
/* Sada koristimo ALTER jer menjamo proceduru */
ALTER procedure [dbo].[spisak_kadeta_odredjene_klase]
@kl int
as
SELECT Klase.Red_br, Kadeti.Prezime, Kadeti.Ime
FROM Kadeti INNER JOIN Klase ON Kadeti.ID_Klase = Klase.ID_Klase
WHERE (Klase.Red_br = @kl)
```

Kada jednom kreiramo uskladištenu proceduru, ona funkcioniše samo na nivou baze u kojoj je kreirana, i to samo za korisnike koji imaju pravo na njeno korišćenje. Treba biti pažljiv u slučajevima izmene tela procedure, kada koristimo naredbu **DROP** i **CREATE**. Naredbom **DROP** gube se sva korisnička prava nad procedurom, čak i ako nova procedura ima isto ime kao i stara.

Podrazumevano, proceduru može menjati ili brisati jedino korisnik koji je i kreirao.

## Parametrizacija

Uskladištena procedura ima mogućnost prihvatanja i prosleđivanja parametara, prilikom čega je moguće koristiti tri vrste parametara: IN, OUT i INOUT. Iako su sami nazivi prilično slikoviti, ukratko ćemo ih objasniti [1]:

1. **IN** - podrazumeva ulazni parametar, koji je nakon prihvatanja vidljiv samo u telu procedure.
2. **OUT** je kreiran unutar procedure, ali može biti vidljiv izvan nje.
3. **INOUT** se kreira izvan procedure i vidljiv je sve vreme, i unutar procedure i izvan nje.

**IN** je podrazumevani pravac kretanja parametra. To znači da ne morate eksplicitno naglasiti to njegovo svojstvo. Namerno kažemo pravac kretanja, a ne tip, jer parametri takođe poseduju i tipove (koji nisu isto što i pravac kretanja). Štaviše, nemoguće je proslediti parametar proceduri, a da unutar njene definicije ne postoji definisan i njegov očekivani tip.

Procedura može da vrati vrednost, a i ne mora da vrati vrednost. U njoj može da bude samo niz naredni moje trebaju da se izvrše. Glavna moć uskladištenih procedura je prevođenje, odnosno kompajliranje. Jednom kada se procedura kreira i izvrši, nakon toga svaki naredni poziv te procedure samo se izvršava i značajno dobijamo na performansama [2].

Parametri procedure se navode nakon naziva procedure na sledeći način: **VRSTA naziv\_parametra TIP\_PODATAKA**. Ukoliko ima više parametara oni su razdvojeni zarezom.

Za dodeljivanje vrednosti parametru ili promenljivoj koristi se **SET** naredba. Kako izlaznom, tako i ulaznom parametru se može dodeliti vrednost, pri čemu dodeljivanje vrednosti izlaznom parametru nema efekta na taj podatak van procedure.

## Blokovi naredbi

Gde god je dozvoljeno da se jedan SQL iskaz upotrebi, može se koristiti i blok iskaza ograničen sa **BEGIN** i **END** ključnim rečima. U okviru njega se skup od više iskaza tretira kao jedan iskaz. Svaki iskaz u okviru bloka mora biti završen tačkom i zarezom (;) .

Može se primetiti da su korišćene globalne promenljive, tj. promenljive sesije koje se označavaju sa @a, @b, ... Ove promenljive se dalje mogu koristiti gde se želi unutar sesije i vrednost će im biti jednaka poslednjoj vrednosti koju su primile.

Osim toga, procedura može da sadrži lokalne promenljive. Deklaracija lokalne promenljive se vrši pomoću iskaza **DECLARE** naziv\_promenljive **TIP PODATAKA**. Ove promenljive se koriste samo unutar procedure.

Procedura može i da vrati neku vrednost, da ima i izlazni parametar. Svi ulazni parametri su input i po default-u se smatra da su input, tako da nema potrebe naglasiti da je to ulazna deklaracija. Izlazni parametri moraju da imaju argument **OUTPUT**.

Postoji pravilo da, ukoliko procedura ima više ulaznih/izlaznih parametara, parametri sa podrazumevanim vrednostima moraju biti na kraju liste. Ako procedura ima izlazne parametre, poziv za proceduru je drugačiji:

```
DECLARE @Ima int
EXEC Prebroj_Studente @Ima OUTPUT, 3
SELECT 'Broj traženih studenata je: ', @Ima
```

Na sledećem primeru prikazaćemo proceduru koja za prosleđenu klasu prebrojava koliko imamo studenata te određene klase. Rezultat smešta u neki promenljivu @broj. @broj će primiti neku vrednost. Pošto koristimo agregacionu funkciju **COUNT**, i u slučaju da korisnik unese neku vrednost za nepostojeću klasu, rezultat će biti 0, ne **NULL**, zato što je **COUNT (NULL) = 0**, a kod svih ostalih agregacionih funkcija rezultat od **NULL** je **NULL**.

Primer 2: Potrebno je kreirati uskladištenu proceduru koja za prosleđenu klasu, vraća koliko kadeta ima u toj klasi.

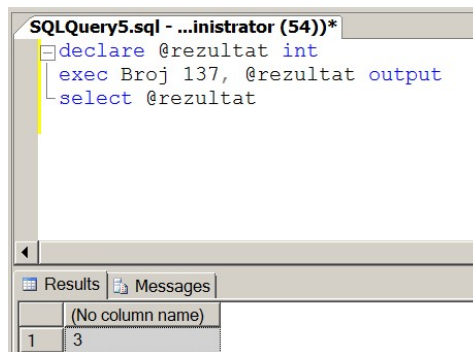
Ulazni parametar je @kl tipa podataka *intiger*, a izlazni @ima *output*, koji je takođe tipa *intiger*.

```
create procedure Broj_studenata
@kl int, @ima int output
as
SELECT @ima=COUNT(Kadeti.ID_Kadeta)
FROM Kadeti INNER JOIN Klase ON Kadeti.ID_Klase = Klase.ID_Klase
WHERE (Klase.Red_br = @kl)
```

Izvršenje:

```
declare @rezultat int
exec Broj 137, @rezultat output
select @rezultat
```

Sintaksno, pri pozivu, ukoliko procedura vraća rezultat mora da postoji **OUTPUT** kao ključna reč i u glavnom kodu i u pozivu. Ovako se koriste procedure koje vraćaju vrednost.



Slika 2 – Prikaz rezultata primera 2

Kako izlaznom, tako i ulaznom parametru se može dodeliti vrednost, pri čemu dodeljivanje vrednosti izlaznom parametru nema efekta na taj podatak van procedure.

## Vrste uskladištenih procedura

Uskladištenih procedura ima jako puno, kao i funkcije. Postoje:

1. Sistemske uskladištene procedure,
2. Proširene uskladištene procedure
3. Korisničke uskladištene procedure (UDP – *user define procedure*)

### Sistemske uskladištene procedure

Nalaze se u bazama *master* i *msdb*, a većina ih počinje znacima **sp\_**.

Opis najčešće korišćenih:

- **sp\_tables**: prikazuje imena svih objekata (sistemske tabele, tabele i pogledi) koje se mogu koristiti iza FROM upita SELECT
- **sp\_stored\_procedures**: imena svih uskladištenih procedura na raspolaganju u bazi podataka
- **sp\_databases**: daje spisak imena svih baza podataka na serveru
- **sp\_server\_info**: prikazuje skup opcija o podešavanjima SQL Server-a (skup znakova, redosled sortiranja, verzija SQL Server-a itd.)
- **sp\_addlogin**: služi za dodavanje standardnog naloga



- **sp\_grantlogin**: omogućava da se nalogu Windows-a dodeli dozvola za pristup serveru
- **sp\_configure**: menjanje globalnih konfiguracionih opcija koje određuju kako se server ponaša (da li će prihvatati direktne izmene sadržaja sistemskih tabela, količina sistemske memorije koju može da koristi itd.)
- **sp\_monitor**: daje kratak prikaz trenutnog rada servera (procenat zauzetosti procesora i sl.)
- **sp\_who**: daje spisak svih korisnika koji trenutno koriste bazu na serveru. Korisno pri obavljanju poslova administriranja
- **sp\_help**: daje podatke o svakom objektu u bazi podataka  
**sp\_help kadeti**  
Vraća mnogo informacija o ovoj bazi. Ovo je za jednog napadača ogroman izvor informacija o podacima. Daje detaljne informacije o svakom objektu koji je kreiran. Napadači baza podataka su uglavno eksperti sistemskih uskladištenih procedura.
- **sp\_helptext**: prikazuje tekst upita kojim je objekat bio napravljen u bazi podataka

Napadač prvo proveriti koje sve baze postoje na SQL serveru na sledeći način:

```
exec sp_databases
```

Pozivanjem ove sistemske procedure vraćaju se sve baze koje postoje u bazama podataka. Iako su sakrivene, SUBP<sup>3</sup> ih prikazuje. Napadaču, sama informacija o nazivu nije toliko interesantna koliko su interesantne druge informacije, kao što je informacija o veličini određene baze podataka koja je meta interesovanja napadača.

## Proširene uskladištene procedure

Proširuju funkcionalnost SQL servera. To su blokovi C++ kôda u obliku DLL-ova. Koriste operativni sistem. Vrlo su rizične i predstavljaju bezbednosnu rupu u SQL serveru zato što mogu izvršiti neki deo koda koji nema nikakve veze sa SQL serverom (npr., mogu obrisati sve fajlove na C particiji). Prefeks je **xp\_**. Bilo koja DOS komanda može se izvršiti u SQL-u.

Neke proširene procedure koje se mogu koristiti samostalno:

- **xp\_cmdshell**: za pokretanje programa projektovanih za pokretanje iz komandne linije (npr. kreiranje novog foldera za automatsko arhiviranje),
- **xp\_fileexist**: ispituje postojanje određene datoteke i prikazuju 0 ili 1 kao rezultat,

```
DECLARE @Ima int
EXEC xp_fileexist 'c:\Test_Backup.dat', @Ima OUTPUT
SELECT @Ima
```

---

<sup>3</sup> SUBP – Sistemi za upravljanje bazama podataka

- **xp\_fixeddrives**: prikazuje slovne oznake pridružene fiksnim diskovima i veličinu slobodnog prostora na njima (u megabajtima).

## Korisničke uskladištene procedure

Korisnički uskladištene procedure ili User defined stored procedures (UDP) - kreiraju programeri baza podataka ili administratori baze podataka. Ove uskladištene procedure sadrže još jednu SQL naredbu za odabir, ažuriranje ili brisanje zapisa iz tabela baza podataka. Korisnički uskladištena procedura može preuzeti ulazne parametre i vratiti izlazne parametre. Korisnički definisana uskladištena procedura je kombinacija DDL (Data Definition Language) i DML (Language Manipulation Language) naredbi.

**Primer 3:** Kreirati proceduru koja vraća spisak kadeta određene klase.

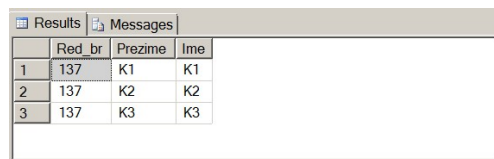
Faktički mi sada imamo dinamički upit.

```
create procedure spisak_kadeta_odredjene_klase
@kl int
as

SELECT Klase.Red_br, Kadeti.Prezime, Kadeti.Ime
FROM Kadeti INNER JOIN Klase ON Kadeti.ID_Klase = Klase.ID_Klase
WHERE (Klase.Red_br = @kl)

exec spisak_kadeta_odr_klase 137
```

Rezultat izvršenja:



	Red_br	Prezime	Ime
1	137	K1	K1
2	137	K2	K2
3	137	K3	K3

Slika 3 – Prikaz rezultata primera 3

Ako pozovemo proceduru bez parametra, SQL će izbaciti grešku, jer procedura ima ulazni parametar, a u naredbi za izvršenje nismo ga definisali. Ako procedura ima ulazni parametar, mora mu se zadati vrednost.

**Primer 4:** Kreirati uskladištenu proceduru koja za određenog kadeta (prezime i ime) vraća spisak njegovih položenih ispita sa ocenama.

```
create procedure Spisak_polozenih_ispita
@ime nvarchar(15), @prezime nvarchar(20)
as
SELECT Predmeti.Naziv_predmeta, Ocene.Vrednost
FROM Evidencija INNER JOIN Kadeti ON Evidencija.ID_Kadeta = Kadeti.ID_Kadeta
INNER JOIN Ocene ON Evidencija.ID_Ocene = Ocene.ID_Ocene INNER JOIN
```

```

Predmetni_nastavnik ON Evidencija.ID_Predmetnog_nastavnika =
Predmetni_nastavnik.ID_Predmetnog_nastavnika INNER JOIN
Predmeti ON Predmetni_nastavnik.ID_Predmeta = Predmeti.ID_Predmeta
WHERE (Kadeti.Ime = @ime) AND (Kadeti.Prezime = @prezime) and Vrednost >5
GROUP BY Predmeti.Naziv_predmeta, Ocene.Vrednost, Kadeti.ID_Kadeta
ORDER BY Predmeti.Naziv_predmeta

```

```
exec Spisak_polozenih_ispita N'Милош', N'Стошић'
```

	Naziv_predmeta	Vrednost
1	Дигитална обрада сигнала	8
2	Моделовање пословних процеса	10
3	Објектно-орјентисано програмирање	10

Slika 4 – Prikaz rezultata primera 4

**Primer 5:** Kreirati uskladištenu proceduru koja za prosleđeni mesec i godinu vraća spisak polaganih ispita kadeta u tom periodu (prezime i ime kadeta, predmet i ocena).

```

create procedure Mesec_i_godina
@mesec int, @godina int
as
SELECT Kadeti.Prezime, Kadeti.Ime, Predmeti.Naziv_predmeta, Ocene.Vrednost
FROM Evidencija INNER JOIN Kadeti ON Evidencija.ID_Kadeta = Kadeti.ID_Kadeta
INNER JOIN Ocene ON Evidencija.ID_Ocene = Ocene.ID_Ocene INNER JOIN
Predmetni_nastavnik ON Evidencija.ID_Predmetnog_nastavnika =
Predmetni_nastavnik.ID_Predmetnog_nastavnika INNER JOIN
Predmeti ON Predmetni_nastavnik.ID_Predmeta = Predmeti.ID_Predmeta
WHERE Month (Evidencija.Datum) = @mesec and YEAR (Evidencija.Datum)= @godina
GROUP BY Kadeti.Prezime, Kadeti.Ime, Predmeti.Naziv_predmeta, Ocene.Vrednost,
Kadeti.ID_Kadeta, Kadeti.Broj_indeksa
ORDER BY Kadeti.Prezime, Kadeti.Ime, Predmeti.Naziv_predmeta

```

```
exec Mesec_i_godina 2,2020
```

	Prezime	Ime	Naziv_predmeta	Vrednost
1	Мираилович	Јелена	Моделовање пословних процеса	10
2	Мирковић	Милош	Моделовање пословних процеса	7
3	Мирковић	Милош	Објектно-орјентисано програмирање	9
4	Стошић	Милош	Дигитална обрада сигнала	8
5	Стошић	Милош	Моделовање пословних процеса	10

Slika 5 – Prikaz rezultata primera 5

**Primer 6:** Kreirati uskladištenu proceduru koja korišćenjem podupita vraća prezimena i imena kadeta koji do sada nisu položili nijedan ispit.

```

create procedure Kadeti_koji_nisu_položili_ispit
@predmet nvarchar(50)
as
select Ime , Prezime
from Kadeti
where ID_Kadeta not in
(
select distinct ID_Kadeta
from Evidencija

```

```

where ID_Predmetnog_nastavnika in
(
    select ID_Predmetnog_nastavnika
    from Predmetni_nastavnik
    where ID_Predmeta =
        (
            select ID_Predmeta
            from Predmeti
            where Naziv_predmeta=@predmet
        )
) and ID_Ocene >1
)
group by Prezime ,Ime ,Broj_indeksa
order by Prezime,Ime
exec Kadeti_koji_nisu_položili_ispit N'Оперативни системи'
exec Kadeti_koji_nisu_položili_ispit N'Објектно-орјентисано програмирање'

```

	Ime	Prezime
1	Zoran	Markovic
2	eee	ww

Query executed successfully.

	Ime	Prezime
1	Zoran	Markovic
2	eee	ww

Query executed successfully.

Slika 6 – Prikaz rezultata primera 6

*Primer 7:* Kreirati uskladištenu proceduru koja za prosleđenu klasu kreira i popunjava tabelu (u tekućoj bazi podataka) sa poljima: *red\_br*, *ukupno kadeta u klasi*, *max\_broj položenih ispita* i *broj kadeta sa max broj položenih ispita*.

```

create procedure Popuni_tabelu
@kl int
as
create table Rezultat
(
    red_br int,
    ukupno int,
    max_polozeni int,
    br_kadeta int
)
/*Deklarisemo promenljive*/
declare @ukupno int
declare @max_pol int
declare @broj_kadeta int
/*

```

Ovaj upit vraća koliko kadeta ima u toj klasi i prosledjuje ga u promenljivu @ukupno. Ovo je upit nad 2 tabele koje su povezane preko INNER JOIN-a. Ovo bi moglo da se reši i nad jednom tabelom, samo u slučaju ako znamo ID\_Klase, ali obzirom da se radi o korisnički uskladištenim procedurama, za korisnika ID ne predstavlja nikakvu informaciju. Zašto onda ovo I spominjemo? Zbog optimizacije. Ako su nam performanse bitne, onda se radi sa ID-jevima. Upit nad 2 tabele je značajno sporiji, nego upit nad jednom tabelom koji kao kriterijum koristi **WHERE** argument i pre će se izvršiti.

```

SELECT @ukupno=COUNT(Kadeti.ID_Kadeta)
FROM Kadeti INNER JOIN Klase ON Kadeti.ID_Klase = Klase.ID_Klase
WHERE (Klase.Red_br = @kl)

/*
Sada nam treba maksimalni broj položenih ispita. Kako ovo možemo rešiti?
Prvo za svakog kadeta tražimo koliko ima položenih ispita, pa onda trazimo
maximum od tog broja. Faktički treba nam: max(count). Raščlanjujemo zadatak na
dva manja zadatka.
Nažalost, u SQL-u, sintaksno nije dozvoljeno izvršenje poziva agregacione
funkcije nad agregacionom funkcijom. Ne postoji funkcija max(count). Možemo
druge funkcije da koristimo. Pa kako onda ovo rešiti?
Rešenja: Ubaciti promenljivu tipa table koja ima samo 1 kolonu i u kojoj se
nalazi broj koliko svaki od kadeta ima položenih ispita, a onda uradimo max nad
tim i dobijam max broj položenih ispita.
*/

declare @tabela table                                     /* Promenljiva tipa table*/
(
id int,                                                  /* Prosledjujemo ID kadeta*/
broj int
)
/*
SELECT vraća 2 vrednosti koje treba uneti u tabelu. Ovde treba obratiti pažnju
prilikom kreiranja SELECT naredbe. Treba navesti tačno onim redosledom kao kojim
smo deklarirali u tabelu @tabela.
*/
insert into @tabela                                     /* Upisujemo u tabelu rezultat
                                                         SELECT naredbe*/
SELECT Ispiti.ID_Kadeta, COUNT(Ispiti.ID_Ispita)
FROM Ispiti INNER JOIN
      Kadeti ON Ispiti.ID_Kadeta = Kadeti.ID_Kadeta INNER JOIN
      Klase ON Kadeti.ID_Klase = Klase.ID_Klase
WHERE (Klase.Red_br = @kl)
GROUP BY Ispiti.ID_Kadeta

/*Sada tražimo max od ukupnog broja kadeta koji su položili ispite*/
select @max_pol = MAX (broj)
from @tabela

/*
Ulazimo u virtuelnu tabelu koja ima 2 kolone ID kadeta i njegov broj položenih
ispita. Od svih zapisa nama su od interesa samo oni zapisi gde je taj broj
jednak maksimalnom broju položenih ispita. Uradimo COUNT od toga I smetimo u
promenljivu @broj_kadeta
*/
select @broj_kadeta= COUNT (id)
from @tabela
where broj=@max_pol

/*Upis u tabelu*/
insert into Rezultat
values (@kl, @ukupno, @max_pol, @broj_kadeta)

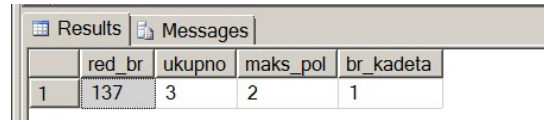
```

Izvršenje procedure:

```
exec popuni 137
```

Prikaz rezultata izvršenja:

```
select * from Rezultat
```



	red_br	ukupno	maks_pol	br_kadeta
1	137	3	2	1

Slika 7 – Prikaz rezultata primera 7

Međutim, ovde nailazimo na jedan problem. Ovako kreirana i izvršena procedura, radi samo jednom, dok će svaki naredni put prijavljivati grešku. Ova greška se odnosi na tabelu koju kreiramo u proceduri, jer ona se jednom kreira i popuni. Ovo se rešava uvođenjem nekih uslovnih naredbi.

U uskladištenoj proceduri moguće je koristiti uslovno izvršenje naredbi IF ... ELSE slično kao što se koristi u bilo kom programskom jeziku. Moguće je ugnjezditi više IF ... ELSE naredbi.

U našem primeru, prvo podešavamo ispitujemo da li virtuelna tabela koju kreiramo u proceduri već postoji, ako postoji nećemo kreirati novu tabelu, a ako ne postoji prvo je kreiramo pa popunjavamo.

Menjamo proceduru Popuni\_tabelu.

```
ALTER procedure [dbo].[Popuni_tabelu]
@kl int
as
-- Provera da li tabela već postoji
if not exists
(
select * from sys.tables
where name=N'Rezultat'
and type='U'
)
/*
Ovaj SELECT vraća nazive tabela (ukoliko ih ima više) čiji je naziv Rezultat,
a type U je tip objekta. Type U vraca samo tabele!
*/
begin
create table Rezultat
(
red_br int,
ukupno int,
max_polozeni int,
br_kadeta int
)
end
else
begin
delete from Rezultat
end
```

... ostatak koda nije menjan ...

# Kursori

Do sada, sve naredbe koje smo izvršavali su bile skupovne naredbe. Kursor je mehanizam SQL-a koji omogućava kretanje kroz zapise. SQL naredbe rade sa skupovima slogova, redova. Kursori omogućavaju rad sa pojedinačnim redovima. Ponaša se poput *recordset*-a u Access DB.

Kursor je u suštini pokazivač, definisan nad jednim skupom slogova. Taj pokazivač redom pokazuje na svaki slog (red) i na taj način omogućava pojedinačnu obradu svakog sloga.

Kursori se mogu koristiti na klijentskoj strani, ali se to najčešće ne preporučuje: kada se obrađuje velika količina podataka (dolazi do zagušenja mreže), troše mnogo resursa, i mnogi SUBP maju problem sa zaključavanjem podataka. Mnogo značajniji i češći u upotrebi su pozadinski ili serverski kursori koji se formiraju na serveru i čuvaju se u *temp.db* bazi podataka. Po default-u je podešeno da pokazuje na prvi zapis i sa tim pokazivačem možemo da prolazimo kroz zapise kako želimo.

Serverski kursori omogućavaju da se SQL naredbe izvršavaju nad skupom podataka koji ima jedan red (granularnost). Time se dobija mogućnost da se nad podacima obavljaju različite operacije zavisno od vrednosti podataka. Dakle, uslovna logička operacija obavlja se nad jednim redom nezavisno od drugih redova u istom skupu.

Da bi obrada dobijenog skupa podataka bila efikasnija, može se koristiti više kursora. To su ugnježdeni kursori.

Da bi se koristio, svaki kursor se najpre mora deklarirati (**DECLARE**). Kada deklariramo kursor kao promenljivu moramo je popuniti nekim vrednostima. Skup podataka na koji se odnosi neki kursor dobija se **SELECT** upitom, koji se navodi pri deklaraciji kursora. Kada je deklarisan, kursor se može otvoriti (**OPEN**) i tada se izvršava upit koji je naveden u deklaraciji kursora. Rezultujući skup podataka ostaje povezan sa kursorom i oni se mogu čitati (**FETCH**). Nakon čitanja sloga, po pravilu kursor se automatski pomera susedni slog (ako on postoji). Po završetku rada kursor se zatvara (**CLOSE**), a memoriju treba osloboditi (**DEALLOCATE**).

Reodsled naredbi za rad sa kursorom

## 1. **DECLARE** ime\_kursora CURSOR

Sintaksa:

```
DECLARE ime_kursora                                /* Rezerviše memoriju koju će  
                                                    kursor koristiti I definiše  
                                                    osnovna svojstva kursora */  
  
[STATIC | SCROLL |  
  FORWARD_ONLY | DYNAMIC]                        /* Vrsta kursora (opciono) */  
CURSOR                                             /* Ime kursora */  
FOR select_iskaz                                  /* Kod kojim popunjavamo promenlj  
                                                    tipa kursor */
```

**STATIC** – izvršava **SELECT** iskaz i rezultat smešta u kursor, ali sve promene koje nastanu posle tog trenutka neće se videti u kursoru. Ne vide se promene nad tabelom.

**SCROLL** – opcija koju nam nudi mogućnost izbora kako želimo da se krećemo kroz zapise. Možemo se kretati kako nama odgovara.

**FORWARD\_ONLY** – ide samo na sledeći zapis, bez mogućnosti sopstvenog izbora

**DYNAMIC** – suprotno od **STATIC**. Svaka promena nad tabelama vidi se u kursoru. Podrazumevano stanje je **STATIC** i **FORWARD\_ONLY**.

2. **SELECT** upit - Koristi se za popunjavanje kursora zapisima koje on treba da referencira.

3. **OPEN** *ime\_kursora*

Ako je kursor deklarisan kao **STATIC** u bazi **temp.db** formira se privremena tabela sa zapisima.

4. **FETCH ... FROM** *ime\_kursora* – Omogućava prenošenje podataka iz kursora u promenljive

Sintaksa:

```
FETCH
[[ NEXT | PRIOR | FIRST | LAST | ABSOLUTE { n | @promenljiva }
| RELATIVE { n | @promenljiva } ]
FROM ]
ime_kursora
[ INTO @ime_promenljive [,...n]]    iz kog kursora u koju promenljivu
```

Dohvatanje podataka, odnosno pristup podacima može biti sekvencijalan (od početka do kraja skupa ili obrnuto, **FIRST**, **NEXT**, **LAST**, **PREVIOUS** (**PRIOR**)) ali i direktan u odnosu na početak ili kraj skupa ili u odnosu na tekući red. Kursori pamte položaj tekućeg sloga u bazi i imaju pokazivače na sledeći i prethodni red [2].

5. **INTO** opcija

Opcija **INTO** definiše promenljive u koje će se podaci učitati (broj promenljivih mora odgovarati broju kolona kursora). Promenljive se popunjavaju vrednostima po redosledu kolona, a tip podataka svake pojedine promenljive mora da bude isti kao tip podataka odgovarajuće kolone kursora.

Globalna promenljiva **@@FETCH\_STATUS** sadrži podatak o izvršavanju poslednje operacije **FETCH** (ako njena vrednost nije nula, to znači da izvršavanje iskaza **FETCH** iz nekog razloga nije uspelo).

Pristup svim članovima kursora:

```
FETCH NEXT FROM Tcursor INTO @prom
WHILE @@FETCH_STATUS = 0
BEGIN
PRINT @prom
FETCH NEXT FROM Tcursor INTO @prom
END
```



6. **CLOSE** ime\_kursora

7. **DEALLOCATE** ime\_kursora

Primer 8: Kreirati uskladištenu proceduru koja korišćenjem kursora kreira i popunjava novu tabelu koja će sadržati polja: prezime kadeta, ime kadeta, broj šestica, broj sedmica, broj osmica, broj devetki i broj desetki (za sve kadete).

```
create procedure Tabela_sa_kursorom
as
if not exists (select * from sys.tables
               where name='Pregled' and type='U')
create table Pregled
(
    prezime nvarchar(30),
    ime nvarchar(30),
    br_sestica varchar(2),
    br_sedmica varchar(2),
    br_osmica varchar(2),
    br_devetki varchar(2),
    br_desetki varchar(2)
)
else delete from Pregled

declare @prezime nvarchar(30)
declare @ime nvarchar(30)
declare @sestice int
declare @sedmice int
declare @osmice int
declare @devetke int
declare @desetke int

declare kadetii cursor for
select prezime, ime
from kadeti inner join klasa
on kadeti.id_klase=klasa.id_klase
where ID_Statusa=1

open kadetii
fetch next from kadetii into @prezime,@ime
while @@FETCH_STATUS=0
begin
select @sestice= COUNT(Evidencija.ID_Evidencije )
FROM          Evidencija INNER JOIN Kadeti ON Evidencija.ID_Kadeta =
              Kadeti.ID_Kadeta INNER JOIN Ocene ON Evidencija.ID_ocene =
              Ocene.ID_Ocene
WHERE          (Ocene.Vrednost = 6) AND (Kadeti.Prezime = @prezime) AND
              (Kadeti.Ime = @ime)

select @sedmice= COUNT(Evidencija.ID_Evidencije )
FROM          Evidencija INNER JOIN Kadeti ON Evidencija.ID_kadeta =
              Kadeti.ID_kadeta INNER JOIN Ocene ON Evidencija.ID_ocene =
              Ocene.ID_Ocene
WHERE          (Ocene.Vrednost = 7) AND (Kadeti.Prezime = @prezime) AND
              (Kadeti.Ime = @ime)
```

```

select @osmice= COUNT(Evidencija.ID_Evidencije )
FROM      Evidencija INNER JOIN Kadeti ON Evidencija.ID_kadeta =
          Kadeti.ID_kadeta INNER JOIN Ocene ON Evidencija.ID_ocene =
          Ocene.ID_Ocene
WHERE      (Ocene.Vrednost = 8) AND (Kadeti.Prezime = @prezime) AND
          (Kadeti.Ime = @ime)

select @devetke= COUNT(Evidencija.ID_Evidencije )
FROM      Evidencija INNER JOIN Kadeti ON Evidencija.ID_kadeta =
          Kadeti.ID_kadeta INNER JOIN Ocene ON Evidencija.ID_ocene =
          Ocene.ID_Ocene
WHERE      (Ocene.Vrednost = 9) AND (Kadeti.Prezime = @prezime) AND
          (Kadeti.Ime = @ime)

select @desetke= COUNT(Evidencija.ID_evidencije)
FROM      Evidencija INNER JOIN Kadeti ON Evidencija.ID_kadeta =
          Kadeti.ID_kadeta INNER JOIN Ocene ON Evidencija.ID_ocene =
          Ocene.ID_Ocene
WHERE      (Ocene.Vrednost = 10) AND (Kadeti.Prezime = @prezime) AND
          (Kadeti.Ime = @ime)

if (@sestice is null) select @sestice=0
if (@sedmice is null) select @sedmice=0
if (@osmice is null) select @osmice=0
if (@devetke is null) select @devetke=0
if (@desetke is null) select @desetke=0

insert into Pregled values
(@prezime,@ime,@sestice,@sedmice,@osmice,@devetke,@desetke)
fetch next from kadetii into @prezime,@ime
end
close kadetii
deallocate kadetii

exec Tabela_sa_kursorom
select * from Pregled

```

Results		Messages					
	prezime	ime	br_sestica	br_sedmica	br_osmica	br_devetki	br_desetki
1	Мијаиловић	Јелена	0	0	2	0	1
2	Milanovic	Dusica	0	0	0	1	0
3	Стошић	Милош	0	0	1	0	2
4	Мирковић	Милош	0	1	0	1	0
5	Milanovic	Aleksandar	0	0	0	0	0
6	tot	ivanaaa	0	1	0	2	0
7	Markovic	Zoran	0	0	0	0	0

Query executed successfully. (local) (10.0 RTM)

Slika 8 – Prikaz rezultata primera 8

## Ugneždeni kursori

Do sada smo videli kako možemo koristiti kursora za upravljanje potencijalno višim brojem redova u rezultatu upita. Obrada ovih redova je do sada obuhvatala jednostavno procesiranje podataka, poput ispisivanja na standardni izlaz, uz eventualne transformacije ili ažuriranje ili brisanje podataka na koje pokazuje kursor [3].

Međutim, šta raditi ukoliko je potrebno da za svaki red rezultata jednog upita izvršimo akciju nad rezultatom nekog drugog upita? Da li nam kursori u ovakvim situacijama mogu pomoći? Odgovor je potvrđan zbog činjenice da je kursora moguće ugnežđavati. Tipičan tok rada podrazumeva naredne korake u slučaju dva kursora od kojih je jedan (unutrašnji) ugnežđen u drugi (spoljašnji):

1. Deklaracija spoljašnjeg kursora.
2. Deklaracija unutrašnjeg kursora.
3. Otvaranje spoljašnjeg kursora.
4. Dohvatanje jednog po jednog reda spoljašnjeg kursora. Za svaki dohvaćeni red u spoljašnjem kursoru:
  1. Obrada dohvaćenih podataka spoljašnjeg kursora.
  2. Otvaranje unutrašnjeg kursora.
  3. Dohvatanje jednog po jednog reda unutrašnjeg kursora. Za svaki dohvaćeni red u unutrašnjem kursoru:
    1. Obrada dohvaćenih podataka unutrašnjeg kursora.
  4. Zatvaranje unutrašnjeg kursora.
5. Zatvaranje spoljašnjeg kursora.

Naredna primer ilustruju rad sa ugnežđenim kursorima.

Primer 9: Napisati uskladištenu proceduru koja prikazuje tabelu u sledećem formatu:

Klasa	Predmet	Broj položenih ispita
137		
	N1	5
	N2	3
139		
	P1	4
	P2	1
	P3	0

```

create procedure daj_spisak_studenata
as
declare @t table
(klasa varchar(10), predmet nvarchar(30), broj_pol varchar(10))

--kursor kl sadrži sve klase
declare kl cursor for
select ID_Klase, Redni_broj from Klasa order by Redni_broj

--kursor pr sadrži sve predmete
declare pr cursor for
select ID_Predmeta, Naziv_predmeta from Predmeti

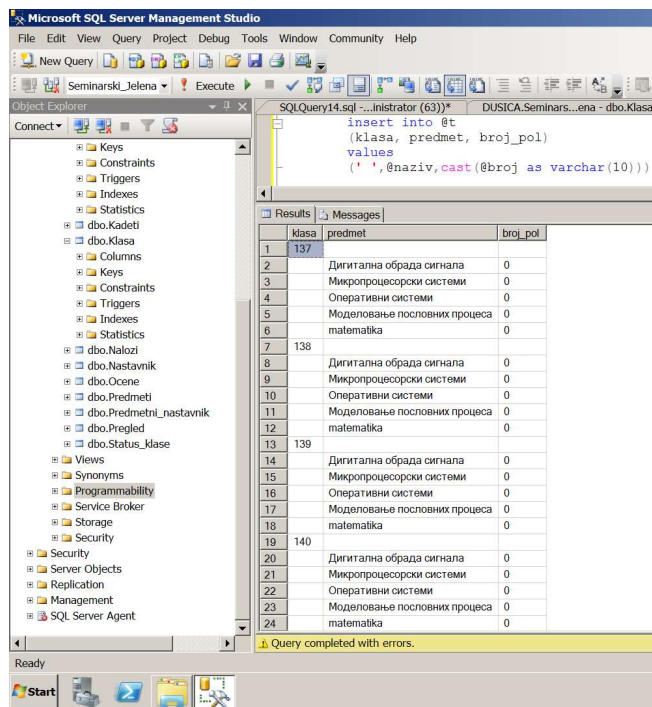
declare @red_br int
declare @id_kl int
declare @id_pr int
declare @naziv nvarchar (50)
declare @broj int

open kl
fetch next from kl into @id_kl, @red_br
--while petlja koja radi sve dok imamo klase
while @@FETCH_STATUS=0
begin
insert into @t
(klasa, predmet, broj_pol)
values
(cast(@red_br as varchar(10)), ' ', ' ')
open pr
fetch next from pr into @id_pr, @naziv
--while petlja koja radi sve dok imamo predmete
while @@FETCH_STATUS=0
begin
--koliko studenata je položilo tekući predmet
SELECT @broj= COUNT(Evidencija.ID_Evidencije)
FROM Evidencija INNER JOIN Kadeti ON Evidencija.ID_Kadeta =
Kadeti.ID_Kadeta INNER JOIN Predmetni_nastavnik ON
Evidencija.ID_Predmetnog_nastavnika =
Predmetni_nastavnik.ID_Predmetnog_nastavnika
WHERE (Kadeti.ID_Klase = @id_kl) AND (Predmetni_nastavnik.ID_Predmeta =
@id_pr)
--upiši u tabelu taj predmet
insert into @t
(klasa, predmet, broj_pol)
values
(' ', @naziv, cast(@broj as varchar(10)))
fetch next from pr into @id_pr, @naziv
end
close pr
fetch next from kl into @id_kl, @red_br --pređi na sledeću klasu
end
close kl
deallocate pr
deallocate kl
select * from @t

exec daj_spisak_studenata

```

Prikaz rezultata:



Slika 9 – Prikaz rezultata primera 9

## Zaključak

Svi pokazani primeri izvršeni su u izabranom SUBP – MS SQL Management Studio 2008. Primeri koriste blokove sastavljene od SQL naredbi koji omogućavaju izvršenje raznih kontrola i obrada podataka.

U radu smo prvo obradili Uskladište procedure koje smo kasnije integrisali sa Kursorima. Uskladištene procedure omogućavaju portabilnost poslovne logike aplikacije i smanjenje protoka podataka (odnosno, veću brzinu) između aplikacije i baze podataka

Primeri pokazani u ovom radu knjizi su zasnovani su na prethodno kreiranoj bazi podataka *Studenti* koja sa sledećim entitetima: *Kadet, Klasa, Ispiti, Predmet, Nastavnik* I *Ocena*.

# Literatura

- [1] Slobodan Obradović, Siniša Ilić, *SQL Strukturirani upitni jezik u sistemima za upravljanje relacionim bazama podataka*, Beograd 2016
- [2] Ivan Tot, *Informacioni sistemi za podršku odlučivanju*, Beograd 2012
- [3] Ivan Tot, *Razvoj On-Line Analytical Processing (OLAP) sistema za podršku odlučivanju*, Beograd 2010
- [4] <http://www.link-university.com/lekcija/Uskladi%C5%A1tene-procedure/5007>
- [5] <https://theikeofficial.github.io/PDb2BP/poglavlja/3/>