OBLIKOVANJE BAZA PODATAKA

Predavanje 08

Dohvaćanje IDENTITY vrijednosti (1/2)

- Ako je stupac definiran kao `IDENTITY`, baza podataka generira novu vrijednost za svaki umetnuti redak
- Ponekad nam treba ta nova generirana vrijednost jer o njoj ovisi nastavak skripte
  - Primjerice, pišemo skriptu koja umeće državu i zatim umeće dva grada u tu državu:

```
INSERT INTO Drzava (Naziv) VALUES ('Indija')
INSERT INTO Grad (Naziv, DrzavaID) VALUES ('Agra', ?)
INSERT INTO Grad (Naziv, DrzavaID) VALUES ('Delhi', ?)
```

  - Kako možemo znati koji je IDDrzava generiran tako da ga možemo upisati umjesto upitnika?

Dohvaćanje IDENTITY vrijednosti (2/2)

- Sistemska funkcija **`SCOPE_IDENTITY()`** vraća zadnje eksplicitno kreiranu `IDENTITY` vrijednost
- Dobra praksa je odmah staviti tu vrijednost u varijablu
  - Idući INSERT će generirati novu vrijednost i stara je izgubljena
- Rješenje prethodnog primjera:

```
DECLARE @ID int

INSERT INTO Drzava (Naziv) VALUES ('Indija')

SET @ID = SCOPE_IDENTITY()

INSERT INTO Grad (Naziv, DrzavaID) VALUES ('Agra', @ID)
INSERT INTO Grad (Naziv, DrzavaID) VALUES ('Delhi', @ID)
```

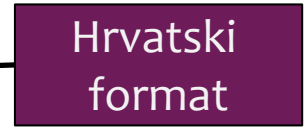
Pretvaranje tipova podataka

- Pretvaranje može biti:
  - Implicitno – pretvaranje se događa bez korisnikovog zahtjeva
  - Eksplicitno – korištenjem sistemskih funkcija `CAST` ili `CONVERT`
- Sintaksa funkcije `CAST` je:

  `CAST(`*`izraz AS tip_podataka`*`)`
- Sintaksa funkcije `CONVERT` je:

  `CONVERT(`*`tip_podataka, izraz`* `[ , stil ])`
- U većini slučajeva svejedno koju funkciju ćemo koristiti
- `CONVERT` je fleksibilnija zbog mogućnosti definiranja stila i u nekim slučajevima je jedini izbor

```
CONVERT(char(10), @Datum, 104)
```



Naredba CASE

- Služi za provjeru uvjeta i **vraćanje** jedne skalarne vrijednosti
  - Može se koristiti tamo gdje se očekuje jedna vrijednost
- **Ne može** se koristiti za upravljanje tijekom programa kao u programskim jezicima!
- Sintaksa naredbe **CASE**:

```
CASE
   WHEN uvjet1 THEN rezultat1
   WHEN uvjet2 THEN rezultat2
   ...
   ELSE rezultat
END
```

Vraća **jednu** vrijednost

Naredba `WHILE`

- Služi za implementiranje **petlji** u T-SQL-u:

```
WHILE uvjet
BEGIN
      naredbe
END
```

- Primjer:

```
DECLARE @i int
SET @i = 1
WHILE @i <= 10
BEGIN
      PRINT @i
      SET @i = @i + 1
END
```

Primjeri

1. Dohvatite sve podatke za račune iz 2003, s tim da datum izdavanja formatirate na hrvatski način.
2. Dohvatite nazive svih proizvoda i uz svaki naziv u zagradi ispišite i njegov ID.
3. Dohvatite sve proizvode i pri tome dodajte još jedan stupac u kojem ćete ispisati:
   - "Jeftino" ako je cijena manja od 100
   - "Srednje skupo" ako je cijena između 100 i 1000
   - "Skupo" ako je cijena veća od 1000
4. Dohvatite naziv i boju svih proizvoda. Za boju koja je NULL ispisati "Nije definirana".

Greške u SQL Serveru

- Svaka greška u SQL Serveru ima svoju **razinu ozbiljnosti** (engl. *severity level*):
  - Razine **1 do 10** predstavljaju **informacije**
  - Razine **11 do 16** predstavljaju **korisničke greške**
    - 11 – traženi objekt ne postoji (npr. `DROP TABLE NemaMe`)
    - 15 – upit nema ispravnu sintaksu (npr. `SELECT * FRMO Student`)
    - 16 – sve ostale korisničke greške (npr. `PRINT 17/0`)
  - Razine **17 do 19** predstavljaju **softverske greške** koje zahtijevaju intervenciju administratora
    - 17 – nema prostora na disku za traženu operaciju
  - Razine **20 do 25** predstavljaju **fatalne greške** i mogu rezultirati zatvaranjem konekcije

Strukturirano hvatanje grešaka

- T-SQL nudi mogućnost strukturiranog hvatanja grešaka
  - Hvataju se **greške razine 11 do 19** (1-10 i 20-25 se ne hvataju)
- Sastoji se od dva bloka, `TRY` i `CATCH`
  - Ako se u `TRY` **desi** greška, izvođenje se prebacuje u `CATCH`
  - Ako se u `TRY` **ne desi** greška, `CATCH` blok neće biti izvršen
- Sintaksa:

```
BEGIN TRY
      niz_naredbi
END TRY
BEGIN CATCH
      niz_naredbi
END CATCH
```

Korisne funkcije pri hvatanju grešaka

- U `CATCH` bloku su nam korisne sljedeće sistemske funkcije:

`ERROR_MESSAGE()`

  - Vraća tekst greške

`ERROR_NUMBER()`

  - Vraća broj greške (svaka greška ima svoj jedinstveni broj)

`ERROR_SEVERITY()`

  - Vraća razinu greške koja se desila

`ERROR_LINE()`

  - Vraća liniju u kojoj se desila greška

`ERROR_PROCEDURE()`

  - Vraća naziv procedure u kojoj se desila greška

Primjeri

5. Deklarirajte varijable @a i @b i dajte im vrijednosti 51 i 0. Unutar TRY bloka izračunajte @a / @b. Unutar CATCH bloka ispišite informacije o greški koja se desila.

6. Napravite tablicu Vrsta koja ima stupce IDVrsta (primarni ključ, ali nije IDENTITY) i Naziv. Napravite proceduru koja prima IDVrsta i Naziv i umeće ih u tablicu. Pozovite proceduru dva puta s vrijednostima 1 i "Pingvin".
   - Implementirajte TRY/CATCH izvan procedure i pozovite je.
   - Implementirajte TRY/CATCH unutar procedure i pozovite je.

Primjeri primjene procedura i funkcija

Procedure vs funkcija - podsjetnik

- Funkcije koristimo kad:
  1. Želimo vratiti jednu vrijednost i iskoristiti je na nekom mjestu u upitu (umjesto podupita)
  2. Želimo vratiti skup redaka i iskoristiti ga kao izvor podataka za FROM

- Procedure koristimo u svim ostalim slučajevima:
  - Kada je potrebno dodavati, mijenjati ili brisati podatke
  - Kada je potrebno vratiti više skupova redaka kao rezultate
  - Kad su nam potrebni izlazni parametri
  - Kad želimo poslati e-mail iz baze podataka
  - Kad želimo koristiti strukturirano hvatanje grešaka
  - …

Upotreba procedura u CRUD operacijama

- CRUD operacije:
  - **C** = Create
  - **R** = Retrieve
  - **U** = Update
  - **D** = Delete
- Tri načina implementacija CRUD operacija:
  - Svakoj operaciji se dodjeljuje posebna procedura
  - Operacije umetanja i izmjene se obavljaju u jednoj proceduri, druge dvije operacije se obavljaju u posebnim procedurama
  - Operacije umetanja, izmjene i brisanja se obavljaju u jednoj proceduri, a dohvaćanje u drugoj

Primjeri

7. Napravite procedure koje rade CRUD operacije na tablici Student tako da svakoj operaciji dodijelite posebnu proceduru. Iskoristite procedure za umetanje, izmjenu, dohvaćanje i brisanje zapisa.

8. Napravite procedure koje rade CRUD operacije na tablici Student tako da operacije umetanja i izmjene obavite u jednoj proceduri, a druge dvije operacije obavite u posebnim procedurama. Iskoristite procedure za umetanje, izmjenu, dohvaćanje i brisanje zapisa.

9. Napravite procedure koje rade CRUD operacije na tablici Student tako da operacije umetanja, izmjene i brisanja obavite u jednoj proceduri, a dohvaćanje u drugoj. Iskoristite procedure za umetanje, izmjenu, dohvaćanje i brisanje zapisa.

Upotreba funkcija u CRUD operacijama

- U obzir dolaze samo tablične funkcije
- Pošto ne mogu mijenjati podatke, ne možemo ih koristiti za `INSERT`, `UPDATE` niti `DELETE`, već samo za `SELECT`
  - U sva tri načina implementacije CRUD operacija možemo `SELECT` proceduru zamijeniti funkcijom
- Primjer:

```
CREATE FUNCTION dbo.StudentGet ( @IDStudent int )
RETURNS TABLE
AS
   RETURN
         SELECT IDStudent, Ime, Prezime
         FROM Student WHERE IDStudent = @IDStudent
```

Primjer

10. Za tablicu Student koja se sastoji od IDStudent, Ime, Prezime i JMBAG implementirajte tabličnom funkcijom operaciju SELECT. Iskoristite funkciju za dohvaćanje zapisa.