Proceduri stocate. Execuție dinamică. Cursoare

Seminar 3

Proceduri stocate

- O procedură stocată este un grup de instrucțiuni SQL compilate într-un singur plan de execuție
- Procedurile stocate pot:

  - accepta parametri de intrare și returna multiple valori ca parametri de ieșire

  - conține instrucțiuni de programare care efectuează operațiuni în baza de date, inclusiv apeluri de proceduri

  - returna o valoare de stare care indică succes sau eroare

Proceduri stocate

- Beneficii ale utilizării procedurilor stocate:

  - reducerea traficului pe rețea

  - control mai bun al securității

  - reutilizarea codului

  - întreținere simplificată

  - performanță îmbunătățită

  - posibilitatea de a încorpora cod pentru tratarea erorilor direct în interiorul procedurii stocate

Proceduri stocate

- Sintaxa:

```
CREATE PROCEDURE procedure_name
(@param1 parameter_datatype,
@param2 parameter_datatype)
AS
BEGIN
-- sequence of SQL statements
END
```

Proceduri stocate

- Execuția unei proceduri stocate:

```
EXEC procedure_name;
```

SAU

```
procedure_name;
```

SAU

```
EXEC procedure_name param_1, param_2, … , param_n;
```

SAU

```
procedure_name param_1, param_2, … , param_n;
```

Proceduri stocate

– Următoarea procedură stocată adaugă o coloană de tipul DATE cu numele *data_nașterii* în tabelul *Persoane*:

```
CREATE PROCEDURE uspAdaugaDataNasterii
AS
BEGIN
        ALTER TABLE Persoane
        ADD data_nașterii DATE;
END;
```

Proceduri stocate

- Exemplu de procedură stocată cu parametri de intrare:

```
CREATE PROCEDURE uspReturneazaPersoane
(@nume VARCHAR(30),
@prenume VARCHAR(30))
AS
BEGIN
SELECT nume, prenume, oras FROM Persoane
WHERE nume=@nume AND prenume=@prenume;
END;
```

Proceduri stocate

- Dorim să modificăm procedura stocată astfel încât să returneze numărul de persoane cu un anumit nume și prenume:

```
ALTER PROCEDURE uspReturneazaPersoane
(@nume VARCHAR(30), @prenume VARCHAR(30),
@Numar INT OUTPUT)
AS
SELECT @Numar=COUNT(*) FROM Persoane
WHERE nume=@nume AND prenume=@prenume;
GO
```

Proceduri stocate

- Procedura stocată se apelează în modul următor:

```
DECLARE @Numar AS INT;
SET @Numar=0;
EXEC uspReturneazaPersoane 'Pop', 'Oana',
@Numar=@Numar OUTPUT;
PRINT @Numar;
```

Proceduri stocate - RAISERROR

- **RAISERROR** generează un mesaj de eroare și inițiază procesarea erorilor pentru sesiune
- **RAISERROR** poate referi fie un mesaj definit de utilizator stocat în sys.messages catalog view sau poate să construiască un mesaj în mod dinamic
- Sintaxa:

```
RAISERROR ( { msg_id | msg_str | @local_variable}
{, severity, state} )
```

- Severity reprezintă nivelul de severitate definit de utilizator asociat mesajului (utilizatorii pot specifica un nivel de severitate între 0 și 18)

Proceduri stocate - RAISERROR

– O altă variantă a procedurii stocate care conține **RAISERROR**:

```
ALTER PROCEDURE uspReturneazaPersoane (@nume VARCHAR(30),
@prenume VARCHAR(30), @Numar INT OUTPUT)
AS
BEGIN
  SELECT @Numar=COUNT(*) FROM Persoane
  WHERE nume=@nume AND prenume=@prenume;
  IF @Numar=0
     RAISERROR('Nu a fost returnata nicio persoana!', 11, 1);
END;
GO
```

Proceduri stocate

- Putem șterge o procedură stocată cu ajutorul instrucțiunii **DROP PROCEDURE**
- Sintaxa:

```
DROP PROCEDURE [schema_name.]procedure_name;
```

- Exemplu:

```
DROP PROCEDURE uspReturneazaPersoane;
```

SAU

```
DROP PROCEDURE dbo.uspReturneazaPersoane;
```

Variabile globale

- Microsoft SQL Server oferă un număr mare de variabile globale, care reprezintă un tip special de variabile:

  - serverul menține în permanență valorile variabilelor globale

  - toate variabilele globale reprezintă informații specifice serverului sau sesiunii curente

  - numele variabilelor globale începe cu @@

  - variabilele globale nu trebuie declarate (practic ele sunt funcții sistem)

Variabile globale - Exemple

- **@@ERROR** – conține numărul celei mai recente erori T-SQL (0 indică faptul că nu s-a produs nicio eroare)
- **@@IDENTITY** – conține valoarea câmpului IDENTITY al ultimei înregistrări inserate
- **@@ROWCOUNT** – conține numărul de înregistrări afectate de cea mai recentă instrucțiune
- **@@SERVERNAME** – conține numele instanței
- **@@SPID** – conține ID-ul de sesiune al procesului de utilizator curent
- **@@VERSION** – conține informații în legătură cu sistemul și compilarea curentă a serverului instalat

Execuție dinamică

- **EXEC** poate fi folosit pentru a executa SQL în mod dinamic
- **EXEC** acceptă ca parametru un șir de caractere și execută codul SQL din interiorul acestuia
- Sintaxa:

```
EXEC(<command>);
```

- Exemplu:

```
EXEC('SELECT cod_p, nume, prenume FROM Persoane WHERE
cod_p=1;');
```

Execuție dinamică

– În exemplul de mai jos se declară o variabilă de tipul VARCHAR(MAX) în care se stochează o interogare care va fi transmisă mai apoi ca parametru instrucțiunii **EXEC**:

```
DECLARE @var VARCHAR(MAX);
SET @var='SELECT cod_p, nume, prenume FROM Persoane
WHERE  cod_p=1;';
EXEC(@var);
```

Execuție dinamică

- Dezavantajele principale ale execuției dinamice sunt problemele de performanță și posibilele probleme de securitate
- În locul instrucțiunii **EXEC** putem folosi procedura stocată **sp_executesql**
- Procedura stocată **sp_executesql** evită o mare parte din problemele generate de **SQL injection** și este uneori mult mai rapidă decât **EXEC**
- Spre deosebire de **EXEC**, **sp_executesql** suportă doar șiruri de caractere Unicode și permite parametri de intrare și de ieșire

Execuție dinamică

– Dorim să returnăm toate înregistrările din tabelul *Orders* care au *id_customer* egal cu 1 și *id_shipment* egal cu 1:

```
DECLARE @sql NVARCHAR(100);

SET @sql=N'SELECT id_customer, id_order, id_shipment
FROM Orders WHERE id_shipment=@id_shipment AND
id_customer=@id_customer;';

EXEC sp_executesql @sql, N'@id_shipment AS
INT,@id_customer AS INT', @id_shipment=1
,@id_customer=1;
```

Clauza OUTPUT

- Cu ajutorul clauzei **OUTPUT** avem acces la înregistrările modificate, șterse sau adăugate
- În exemplul de mai jos se actualizează numele persoanei care are *cod_p*=5 din tabelul *Persoane* și se stochează în tabelul *ModificăriNumePersoane* valoarea din coloana *cod_p*, valoarea veche a numelui (*deleted.nume*), valoarea nouă a numelui (*inserted.nume*), data curentă (GETDATE()) și numele login-ului care a realizat modificarea (SUSER_SNAME()):

```
UPDATE Persoane SET nume='Pop' OUTPUT inserted.cod_p,
deleted.nume, inserted.nume, GETDATE(), SUSER_SNAME()
INTO ModificăriNumePersoane (cod_p, nume_vechi,
nume_nou, data_modificării, nume_login) WHERE cod_p=5;
```

Cursoare

- Sunt anumite situații în care procesarea unui result-set este mai eficientă dacă se procesează pe rând fiecare înregistrare din result-set
- Deschiderea unui cursor pe un result-set permite procesarea result-set-ului înregistrare cu înregistrare (se procesează o singură înregistrare la un moment dat)
- Cursoarele extind procesarea rezultatelor prin faptul că:

  - permit poziționarea la înregistrări specifice dintr-un result-set

  - returnează o înregistrare sau un grup de înregistrări aflate la poziția curentă din result-set

Cursoare

- suportă modificarea înregistrărilor aflate în poziția curentă în result-set

- suportă diferite niveluri de vizibilitate a modificărilor făcute de către alți utilizatori asupra datelor din baza de date care fac parte din result-set

- permit instrucțiunilor Transact-SQL din script-uri, proceduri stocate și trigger-e accesul la datele dintr-un result-set

Cursoare

- Cursoarele Transact-SQL necesită anumite instrucțiuni pentru declarare, populare și extragere de date:

  - se folosește o instrucțiune **DECLARE CURSOR** pentru a declara cursorul și se specifică o instrucțiune **SELECT** care va produce result-set-ul cursorului

  - se folosește o instrucțiune **OPEN** pentru a popula cursorul, care execută instrucțiunea **SELECT** încorporată în instrucțiunea **DECLARE CURSOR**

  - se folosește o instrucțiune **FETCH** pentru a extrage înregistrări individual din result-set (de obicei **FETCH** se execută de multe ori, cel puțin o dată pentru fiecare înregistrare din result-set)

Cursoare

- dacă este cazul, se folosește o instrucțiune **UPDATE** sau **DELETE** pentru a modifica înregistrarea (acest pas este opțional)

- se folosește o instrucțiune **CLOSE** pentru a închide cursorul și a elibera unele resurse (cum ar fi result-set-ul cursorului și lock-urile de pe înregistrarea curentă)

- cursorul este încă declarat, deci poate fi deschis din nou folosind o instrucțiune **OPEN**

- se folosește o instrucțiune **DEALLOCATE** pentru a elimina referința cursorului din sesiunea curentă iar acest proces eliberează toate resursele alocate cursorului, inclusiv numele său (după acest pas, pentru a reconstrui cursorul este nevoie ca acesta să fie declarat din nou)

- cursoarele aflate în interiorul procedurilor stocate nu necesită închidere și eliminare, aceste instrucțiuni se execută automat când procedura stocată își încheie execuția

Cursoare

- Cursoarele Transact-SQL sunt extrem de eficiente atunci când sunt încorporate în proceduri stocate și trigger-e deoarece totul este compilat într-un singur plan de execuție pe server, deci nu există trafic pe rețea asociat cu returnarea înregistrărilor
- Operațiunea de a returna o înregistrare dintr-un cursor se numește **fetch**, iar în cazul cursoarelor Transact-SQL se folosește instrucțiunea **FETCH** pentru a returna înregistrări din result-set-ul unui cursor
- Instrucțiunea **FETCH** suportă un număr de opțiuni care permit returnarea unor înregistrări specifice:

  - **FETCH FIRST** – returnează prima înregistrare din cursor

Cursoare

- **FETCH NEXT** – returnează înregistrarea care urmează după ultima înregistrare returnată

- **FETCH PRIOR** – returnează înregistrarea care se află înaintea ultimei înregistrări returnate

- **FETCH LAST** – returnează ultima înregistrare din cursor

- **FETCH ABSOLUTE n** – returnează a n-a înregistrare de la începutul cursorului dacă n este un număr pozitiv, iar dacă n este un număr negativ returnează înregistrarea care se află cu n înregistrări înaintea sfârșitului cursorului (dacă n este 0, nicio înregistrare nu este returnată)

Cursoare

- **FETCH RELATIVE n** – returnează a n-a înregistrare după ultima înregistrare returnată dacă n este pozitiv, iar dacă n este negativ returnează înregistrarea care se află înainte cu n înregistrări față de ultima înregistrare returnată (dacă n este 0, ultima înregistrare returnată va fi returnată din nou)

– Comportamentul unui cursor poate fi specificat în două moduri:

- prin specificarea comportamentului cursoarelor folosind cuvintele cheie **SCROLL** și **INSENSITIVE** în instrucțiunea **DECLARE CURSOR** (SQL-92 standard)

Cursoare

- prin specificarea comportamentului unui cursor cu ajutorul tipurilor de cursoare (de obicei API-uri pentru baze de date definesc comportamentul cursoarelor împărțindu-le în patru tipuri de cursoare: forward-only, static (uneori denumit snapshot sau insensitive), keyset-driven și dynamic

– Declararea unui cursor – sintaxa ISO:

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

FOR select_statement

[ FOR { READ ONLY | UPDATE [ OF column_name [ ,...n ]
] } ]
```

Cursoare

- Declararea unui cursor – sintaxa Transact-SQL:

```
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 ] ] ]
```

Cursoare - Exemplu

```
DECLARE @nume VARCHAR(50), @prenume VARCHAR(50), @oraș VARCHAR(50);
DECLARE cursorpersoane CURSOR FAST_FORWARD FOR
SELECT prenume, nume, oraș FROM Persoane;
OPEN cursorpersoane;
FETCH NEXT FROM cursorpersoane INTO @prenume, @nume, @oraș;
WHILE @@FETCH_STATUS=0
        BEGIN
        PRINT @prenume+' '+@nume+ N' s-a nascut in orașul '+@oraș;
        FETCH NEXT FROM cursorpersoane INTO @prenume, @nume, @oraș;
        END
CLOSE cursorpersoane;
DEALLOCATE cursorpersoane;
```