Proceduri stocate. Execuție dinamică. Limbaj de control al fluxului

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
[@param_1 parameter_datatype,
@param_2 parameter_datatype, ...
@param_n 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, localitate 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 @Nr AS INT;
SET @Nr=0;
EXEC uspReturneazaPersoane 'Pop', 'Oana',
@Numar=@Nr OUTPUT;
PRINT @Nr;
```

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;
```

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

SET NOCOUNT

- SET NOCOUNT ON – oprește returnarea mesajului cu numărul de înregistrări afectate de către ultima instrucțiune Transact-SQL sau procedură stocată
- SET NOCOUNT OFF – mesajul cu numărul de înregistrări afectate de către ultima instrucțiune Transact-SQL sau procedură stocată va fi returnat ca parte din result set
- Variabila globală @@ROWCOUNT va fi modificată întotdeauna
- Dacă NOCOUNT este setat pe ON, performanța procedurilor stocate care conțin bucle Transact-SQL sau instrucțiuni care nu returnează multe date se va îmbunătăți (traficul pe rețea este redus)

Execuție dinamică

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

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

- Exemplu:

```
DECLARE @nume_tabel AS VARCHAR(100);
SET @nume_tabel = 'Cadouri';
EXEC('SELECT * FROM '+ @nume_tabel);
```

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;
```

Limbaj de control al fluxului

- Transact-SQL oferă un set de cuvinte speciale, numit limbaj de control al fluxului care pot fi folosite pentru a controla fluxul execuției instrucțiunilor Transact-SQL, al blocurilor de instrucțiuni, al funcțiilor definite de utilizator și al procedurilor stocate
- În lipsa limbajului de control al fluxului, instrucțiunile Transact-SQL se execută în ordine secvențială
- BEGIN … END – delimitează un grup de instrucțiuni SQL care se execută împreună
- Blocurile BEGIN … END pot fi încorporate

Limbaj de control al fluxului

- Sintaxa:

```
BEGIN
        { sql_statement | sql_block}
END
```

- RETURN – iese necondiționat dintr-o interogare sau dintr-o procedură stocată
- Poate fi folosit în orice punct pentru a ieși dintr-o procedură, batch sau bloc de instrucțiuni
- Sintaxa:

```
RETURN [integer_expression]
```

- Se poate folosi pentru a returna status codes - procedurile stocate returnează zero (success) sau o valoare întreagă diferită de zero (failure)

Limbaj de control al fluxului

- Exemplu de procedură stocată care returnează status codes:

```
CREATE PROCEDURE usp_verifica_status @cod_student INT
AS
BEGIN
IF((SELECT judet FROM Studenti WHERE cod_student=@cod_student)='Cluj')
        RETURN 1;
ELSE
        RETURN 2;
END;
---------------
DECLARE @status INT;
EXEC @status=usp_verifica_status 1;
SELECT 'Status'=@status;
```

Limbaj de control al fluxului

- IF … ELSE – condiționează execuția unei instrucțiuni SQL sau a unui bloc de instrucțiuni SQL
- Sintaxa:

```
IF Boolean_expression
 { sql_statement | statement_block }
[ ELSE
 { sql_statement | statement_block } ]
```

Limbaj de control al fluxului

- WHILE – setează o condiție pentru execuția repetată a unei instrucțiuni SQL sau a unui bloc de instrucțiuni SQL
- Sintaxa:

```
WHILE Boolean_expression
      { sql_statement | statement_block | BREAK | CONTINUE }
```

Limbaj de control al fluxului

- BREAK – iese din cea mai interioară buclă WHILE sau dintr-o instrucțiune IF… ELSE din interiorul unei bucle WHILE
- CONTINUE – cauzează reînceperea buclei WHILE, ignorând toate instrucțiunile care apar după CONTINUE
- GOTO – execută un salt în execuție la o porțiune din cod marcată printr-un label

```
Label: -- some SQL statements
GOTO Label;
```

Limbaj de control al fluxului

- WAITFOR – blochează execuția unui batch, tranzacție sau procedură stocată până când un interval de timp sau timp specificat este atins sau o instrucțiune specificată modifică sau returnează cel puțin o înregistrare
- Sintaxa:

```
WAITFOR
{
DELAY 'time_to_pass' | TIME 'time_to_execute' |
[ ( receive_statement ) | (
get_conversation_group_statement ) ]
[ , TIMEOUT timeout ]}
```

Limbaj de control al fluxului

- În funcție de nivelul activității pe server, timpul de așteptare poate varia, deci poate fi mai mare decât timpul specificat în **WAITFOR**
- Exemple:

```
-- execuția continuă la 22:00
WAITFOR TIME '22:00';
-- execuția continuă peste 3 ore
WAITFOR DELAY '03:00';
```

Limbaj de control al fluxului

- THROW – aruncă o excepție și transferă execuția unui bloc CATCH dintr-o construcție TRY … CATCH
- Severitatea excepției este mereu setată pe valoarea 16
- Sintaxa:

```
THROW [ { error_number | @local_variable },
        { message | @local_variable },
        { state | @local_variable } ] [ ; ]
```

- Exemplu:

```
THROW 50002, 'Înregistrarea nu există! ', 1;
```

Limbaj de control al fluxului

- TRY … CATCH – implementează tratarea erorilor pentru Transact-SQL
- Captează toate erorile de execuție care au o severitate mai mare decât 10 și care nu închid conexiunea la baza de date
- Sintaxa:

```
BEGIN TRY
{ sql_statement | statement_block }
END TRY
BEGIN CATCH
[ { sql_statement | statement_block } ]
END CATCH
[ ; ]
```

Limbaj de control al fluxului

- În interiorul unui bloc **CATCH** pot fi folosite următoarele funcții sistem pentru a obține informații despre eroarea care a cauzat execuția blocului **CATCH**:
- **ERROR_NUMBER()** – returnează numărul erorii
- **ERROR_SEVERITY()** – returnează severitatea erorii
- **ERROR_STATE()** – returnează error state number
- **ERROR_PROCEDURE()** – returnează numele procedurii stocate sau al trigger-ului în care a avut loc eroarea
- **ERROR_LINE()** – returnează numărul liniei care a cauzat eroarea
- **ERROR_MESSAGE()** – returnează mesajul erorii

Limbaj de control al fluxului

Mesaje de eroare

- Error number (numărul erorii)
  - o valoare întreagă cuprinsă între 1 și 49999
  - Pentru erorile custom (definite de către utilizator) valoarea este cuprinsă între 50000 și 2147483647
- Error severity (severitatea erorii)
  - 26 de nivele de severitate
  - Erorile care au nivelul de severitate ≥ 16 sunt înregistrate în error log în mod automat
  - Erorile care au nivelul de severitate cuprins între 20 și 25 sunt fatale și închid conexiunea
- Error message (mesajul erorii) – NVARCHAR(2048)