Proceduri stocate

Syntax:

```
CREATE PROCEDURE <Name> [@param1 type1, ...] AS
    -- secventa de comenzi SQL
GO
```

O procedura stocata se ruleaza cu **EXEC:**

```
EXEC <Name>
```

Modificarea unei proceduri:

```
ALTER PROCEDURE <Name> [@param1 type1, ...] AS
    -- secventa de comenzi SQL
GO
```

Procedura simpla:

```
CREATE PROCEDURE afiseazaCursuri
AS
      SELECT titlu
      FROM Cursuri
GO
```

Procedura cu parametru:

```
ALTER PROCEDURE afiseazaCursuri (@credite int)
AS
      SELECT titlu
      FROM Cursuri
      WHERE ects = @credite
GO
```

Procedura cu parametru Output:

```
CREATE PROCEDURE nrCursuri(@credite int, @numar int output)
AS
      SELECT @numar = COUNT(*)
      FROM Cursuri
      WHERE credite = @credite
GO
```

Rularea procedurii cu output:

```
DECALRE @nr int
SET @nr = 0
exec nrCursuri 6, @numar=@nr output
print @nr
```

Procedura cu RAISEERROR

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

- severity :
  - untilizatorul poate folosi levels 0-18
  - sys admin poate folosi severity levels 19-25
- state :
  - un numar intre 0 si 255 care ne ajuta sa determinam unde a aparut o eroare

```
ALTER PROCEDURE nrCursuri(@credite int, @numar int output)
      AS
      BEGIN
            SELECT @numar = COUNT(*)
            FROM Cursuri
            WHERE credite = @credite

            If @Number = 0
                  RAISERROR (‘nu am gasit cursuri’, 10, 1)

      END
      GO
```

Functii definite de utilizator

Exista functii scalare si functii care returneaza un table.

Functii scalare – returneaza o valoare scalara

```
CREATE FUNCTION ufGetCursuriNr (@credite int)
RETURNS int AS
BEGIN
      DECLARE @Return int
      SET @Return = 0
      SELECT @Return = COUNT(*)
      FROM Cursuri
      WHERE ECTS = @credite

      RETURN @Return
END
```

Putem folosi procedura in diverse moduri, de exemplu sa afisam rezultatul:

```
print dbo.ufGetCursuriNr (6)
```

Se poate folosi si pentru a verifica o conditie, dar nu e recomandat:

```
      select *
      from table1
      where column1 = ufGetSomeFunctionValue([param])
```

-> **functia se va apela pentru fiecare rand din table, e ca o subinterogare si poate fi foarte ineficienta**

View

Un view este un table virtual care contine rezultatul unui select (se poate folosi apoi ca orice alt table):

```
CREATE VIEW view_name AS
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

Trigger

Triggerele sunt un fel de procedure stocate care se activeaza automat la diferite evenimente:

- INSERT, UPDATE, DELETE
- CREATE DATABASE, DROP LOGIN , etc.

```
CREATE TRIGGER <trigger-name>
ON {table | view}
[WITH <dml_trigger_option> [,…n] ]
{FOR | AFTER | INSTEAD OF}    --cand se activeaza triggerul
{ [INSERT] [,] [UPDATE] [,] [DELETE] }   -- la ce instructiune se activeaza
triggerul
[ WITH APPEND ] [NOT FOR REPLICATION ]
AS
     {sql_statement [;] [,…n] |    --actiunea triggerului
EXTERNAL NAME <method specifier [;] > }
```

Momentul cand se activeaza triggerul:

- **FOR, AFTER** – au acelasi efect si inseamna ca actiunea triggerului are loc dupa ce s-a executat instructiunea care l-a activat
- **INSTEAD OF** – instructiunea care a activat triggerul nu se mai executa, dar in locul ei se executa actiunea triggerului

Alte informatii utile:

- `inserted` – tabel de sistem care contine informatii despre randurile inserate de ultima instructiune
- `deleted` – tabel de sistem care contine informatii despre randurile sterse de ultima instructiune
- in cazul in care s-a efectuat un update, randurile modificate cu valori vechi apar in `deleted`, iar randurile modificate cu valori noi apar in `inserted`
- `@@ROWCOUNT` – variabila de sistem care contine numarul de randuri afectate de ultima isntructiune

```
CREATE TRIGGER [dbo].[On_Product_Insert]
ON [dbo].[Products]
AFTER INSERT
AS
BEGIN
     SET NOCOUNT ON;
     insert into LogBuys (Name, Date, Quantity)
     select Name, GETDATE(), Quantity
     from inserted
END
```

Cursor

Cursorul poate procesa rezultatul unui SELECT rand cu rand.
Pasii pentru a lucre cu un cursor:

- DECLARE CURSOR – se defineste cursorul si se specifica SELECT-ul al carui rezultat vrem sa il procesam
- OPEN – in momentul in care deschidem cursorul se executa SELECT-ul
- FETCH – obtinem cate un rand din rezultat si il procesam
- CLOSE – inchidem cursorul (cursorul nu mai are nevoie de resultset si nu mai pastreaza eventuale lock-uri)
- DEALLOCATE – sterge cursorul cu totul, nu mai putem sa il refolosim

Sintaxa pentru DECLARE:

```
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] ] ]
```

Categorii de cursori:

- FORWARD-ONLY – in cadrul cursorului putem citi randurile doar unul dupa altul pana la ultimul rand (FAS-FORWARD inseamna FORWARD-ONLY si READ ONLY)
- STATIC – cursorul folsoeste o copie temporara a datelor
- DYNAMIC – in cursor se vad toate modificarile aparute pe randurile din resultset

Operatia FETCH are mai multe optiuni:

- **FETCH FIRST** – returneaza primul rand din cursor si acesta devine randul actual
- **FETCH NEXT** – returneaza urmatorul rand din cursor si acesta devine randul actual
- **FETCH PRIOR** – returneaza randul anterior din cursor si acesta devine randul actual
- **FETCH LAST** – returneaza ultimul rand din cursor si acesta devine randul actual
- **FETCH ABSOLUTE { n | @nvar}**
    - daca contanta *n* sau variabila *@nvar* este pozitiva, atunci returneaza al *n*-lea rand de la inceputul cursorului si acesta devine randul actual
    - daca contanta *n* sau variabila *@nvar* este negativa, atunci returneaza al *n*-lea rand de la sfarsitul cursorului si acesta devine randul actual
    - daca contanta *n* sau variabila *@nvar* este 0 nu returneaza nimic
- **FETCH RELATIVE { n | @nvar}**
    - daca contanta *n* sau variabila *@nvar* este pozitiva, atunci returneaza al *n*-lea rand dupa randul actual din cursor si acesta devine randul actual
    - daca contanta *n* sau variabila *@nvar* este negativa, atunci returneaza al *n*-lea rand anterior randului actual din cursor si acesta devine randul actual
    - daca contanta *n* sau variabila *@nvar* este 0, atunci returneaza randul actual

Variabila de sistem **@@FETCH_STATUS** ne ajuta sa stim daca mai sunt randuri in rezultat:

- Daca are valoarea 0, atunci FETCH s-a executat cu succes
- Daca are valoarea -1 sau -2, atunci FETCH nu s-a executat cu success sau randul cerut nu exista

Exemplu de cursor:

```
DECLARE @ProductID    INT,                    --am definit niste variabile
  @ProductName    VARCHAR(50),
  @ListPrice    MONEY
DECLARE cursorproducts CURSOR FOR       -- am definit cursorul
 SELECT ProductID, ProductName, ListPrice FROM Products.Product
FOR READ ONLY
OPEN cursorproducts                     --am deschis cursorul
FETCH cursorproducts INTO @ProductID, @ProductName, @ListPrice
WHILE @@FETCH_STATUS = 0                -- cat timp inca mai sunt randuri
BEGIN
…… -- code for processing @ProductID,@ProductName, @ListPrice
FETCH cursorproducts INTO @ProductID, @ProductName, @ListPrice  --salveaza valorile randului current in variabile
END
CLOSE cursorproducts
DEALLOCATE cursorproducts
```