Proceduri stocate. Execuție dinamică. Limbaj de control al fluxului

SEMINAR 3

Procedura stocată

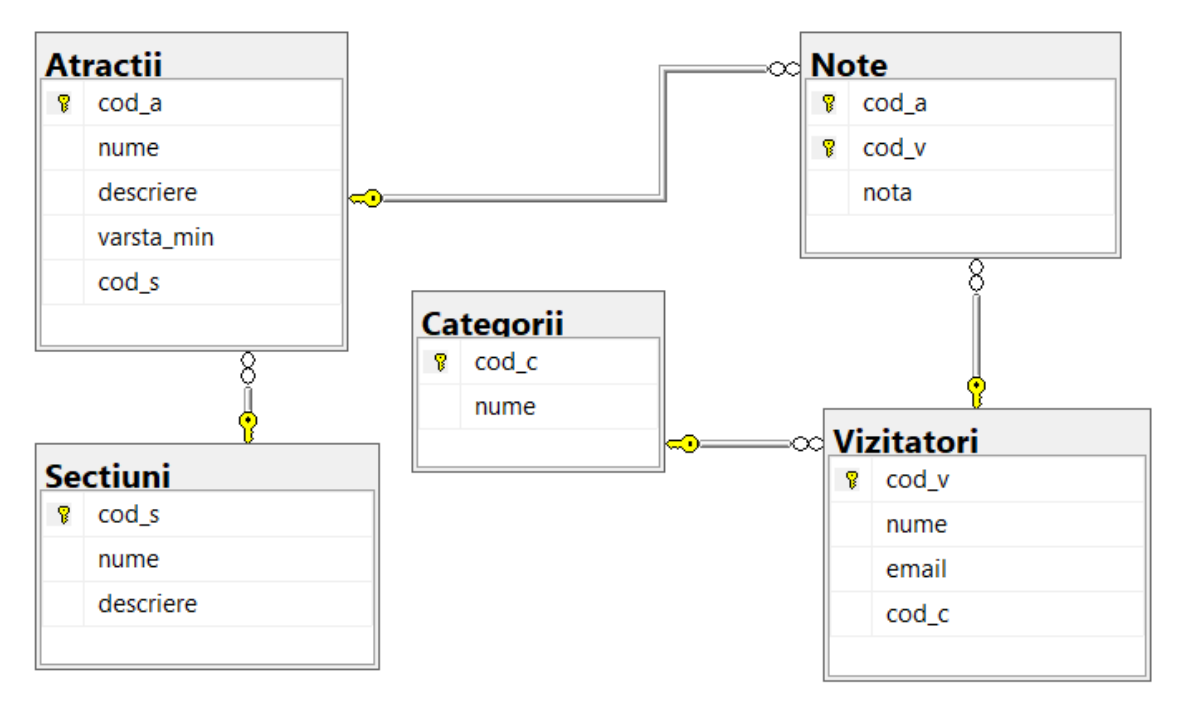
- este o un **grup de instrucțiuni SQL** compilate într-un singur plan de execuție
- **acceptă** parametri de intrare și **poate returna** multiple valori ca parametri de ieșire
- **conține** instrucțiuni de programare care efectuează operațiuni în baza de date, inclusiv **apeluri de proceduri**
- **returnează** o valoare de stare care indică succes sau eroare

Avantajele procedurilor stocate

1. **Reducerea traficului pe rețea**
2. **Control mai bun al securității**
3. **Reutilizarea codului**
4. **Întreținere simplificată**
5. **Performanță îmbunătățită**
6. **Posibilitatea de a încorpora cod pentru tratarea erorilor direct în interiorul procedurii stocate**

Proceduri stocate

- Se dă o bază de date având următoarea structură:



Proceduri stocate

- Următoarea procedură stocată va adăuga o constrângere de **valoare implicită** pentru coloana *varsta_min* din tabelul *Atracții*:

```
CREATE PROCEDURE AdaugaConstrangereDefault

AS

BEGIN

ALTER TABLE Atractii

ADD CONSTRAINT df_varsta_min DEFAULT 12 FOR varsta_min;

END;
```

- Procedura stocată creată anterior se va apela în modul următor:

```
EXEC AdaugaConstrangereDefault;
```

Proceduri stocate

- Următoarea procedură stocată va returna numele, descrierea și vârsta minimă recomandată pentru toate înregistrările din tabelul *Atracții* care au vârsta minimă recomandată egală cu cea furnizată prin intermediul **parametrului de intrare**:

```
CREATE PROCEDURE ReturneazaAtractiiCuVarstaMin @varsta_min INT
AS
BEGIN
SELECT nume, descriere, varsta_min FROM Atractii
WHERE varsta_min=@varsta_min;
END;
```

Proceduri stocate

- Procedura stocată creată anterior se va apela în modul următor:

```
EXEC ReturneazaAtractiiCuVarstaMin 12;
```

SAU

```
EXEC ReturneazaAtractiiCuVarstaMin @varsta_min=12;
```

Proceduri stocate

- Putem **modifica** definiția procedurii stocate astfel încât să returneze prin intermediul unui **parametru de ieșire** numărul de atracții care au vârsta minimă recomandată egală cu cea furnizată prin intermediul **parametrului de intrare**:

```
ALTER PROCEDURE ReturneazaAtractiiCuVarstaMin @varsta_min INT,
@nr_atractii INT OUTPUT
AS
BEGIN
SELECT @nr_atractii=COUNT(*) FROM Atractii WHERE varsta_min=@varsta_min;
END;
```

Proceduri stocate

- Noua variantă a procedurii stocate (care conține un parametru de ieșire) va fi apelată în modul următor:

```
--Se declară o variabilă locală

DECLARE @nratractii AS INT;

--Se inițializează variabila locală

SET @nratractii=0;

--Se apelează procedura stocată

EXEC ReturneazaAtractiiCuVarstaMin 12,@nr_atractii=@nratractii OUTPUT;

--Se afișează pe ecran valoarea parametrului de ieșire, stocată în variabila locală

PRINT @nratractii;
```

Generarea mesajelor de eroare cu ajutorul RAISERROR

- Putem folosi **RAISERROR** pentru a genera mesaje de eroare și pentru a iniția procesarea erorilor pentru sesiune
- **Sintaxa**:

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

- **RAISERROR** poate referi un mesaj definit de utilizator stocat în **sys.messages catalog view** sau poate **construi un mesaj în mod dinamic**
- ***Severity*** reprezintă **nivelul de severitate** definit de utilizator asociat mesajului (utilizatorii pot specifica un nivel de severitate între 0 și 18)

Generarea mesajelor de eroare cu ajutorul RAISERROR

- Putem modifica definiția procedurii stocate astfel încât să genereze un mesaj de eroare dacă **nu** se găsește nicio atracție pentru vârsta minimă recomandată furnizată prin intermediul parametrului de intrare:

```
ALTER PROCEDURE ReturneazaAtractiiCuVarstaMin @varsta_min INT,
@nr_atractii INT OUTPUT
AS
BEGIN
SELECT @nr_atractii=COUNT(*) FROM Atractii WHERE varsta_min=@varsta_min;
IF(@nr_atractii=0)
        RAISERROR('Nu a fost returnata nicio atractie!',16,1);
END;
```

Ștergerea procedurilor stocate

- Procedurile stocate se pot **șterge** cu ajutorul instrucțiunii **DROP PROCEDURE**:

```
--Exemplu de ștergere a unei proceduri stocate

DROP PROCEDURE ReturneazaAtractiiCuVarstaMin;

--Exemplu de ștergere a unei proceduri stocate în care numele procedurii este prefixat cu numele schemei (în acest caz este vorba de schema default, dbo)

DROP PROCEDURE dbo.ReturneazaAtractiiCuVarstaMin;
```

Variabile globale

- În Microsoft SQL Server se pot utiliza **variabile globale**, care **nu** trebuie declarate (ele fiind **funcții sistem**)
- Numele variabilelor globale din Microsoft SQL Server începe cu **@@**
- Server-ul menține în permanență valorile variabilelor globale, care reprezintă informații specifice server-ului sau sesiunii curente

Exemple de variabile globale

- **@@ERROR** – conține numărul celei mai recente erori Transact-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 executată
- **@@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 server-ului 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

```
--Exemplu de procedură stocată care execută cod SQL în mod dinamic

CREATE PROCEDURE ReturneazaDateDinTabel @nume_tabel VARCHAR(100)

AS

BEGIN

EXEC('SELECT * FROM '+@nume_tabel);

END;
```

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

- Exemplu de utilizare a procedurii stocate **sp_executesql**:

```
DECLARE @sql NVARCHAR(100);

SET @sql=N'SELECT nume, descriere FROM Sectiuni WHERE nume<>@nume;';

EXEC sp_executesql @sql, N'@nume AS VARCHAR(100)', @nume='sectiunea 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
  - blocurilor de instrucțiuni
  - funcțiilor definite de utilizator
  - 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 VerificaVarstaMin @cod_a INT
AS
BEGIN
IF((SELECT varsta_min FROM Atractii WHERE cod_a=@cod_a)=12)
        RETURN 1;
ELSE
        RETURN 2;
END;
```

Limbaj de control al fluxului

- Procedura stocată creată anterior se va apela în modul următor:

```
DECLARE @status INT;

EXEC @status=VerificaVarstaMin 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 '03:00:00';
```

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, N'Înregistrarea nu există! ', 1;
```

Limbaj de control al fluxului

- TRY ... CATCH implementează tratarea erorilor pentru Transact-SQL și 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

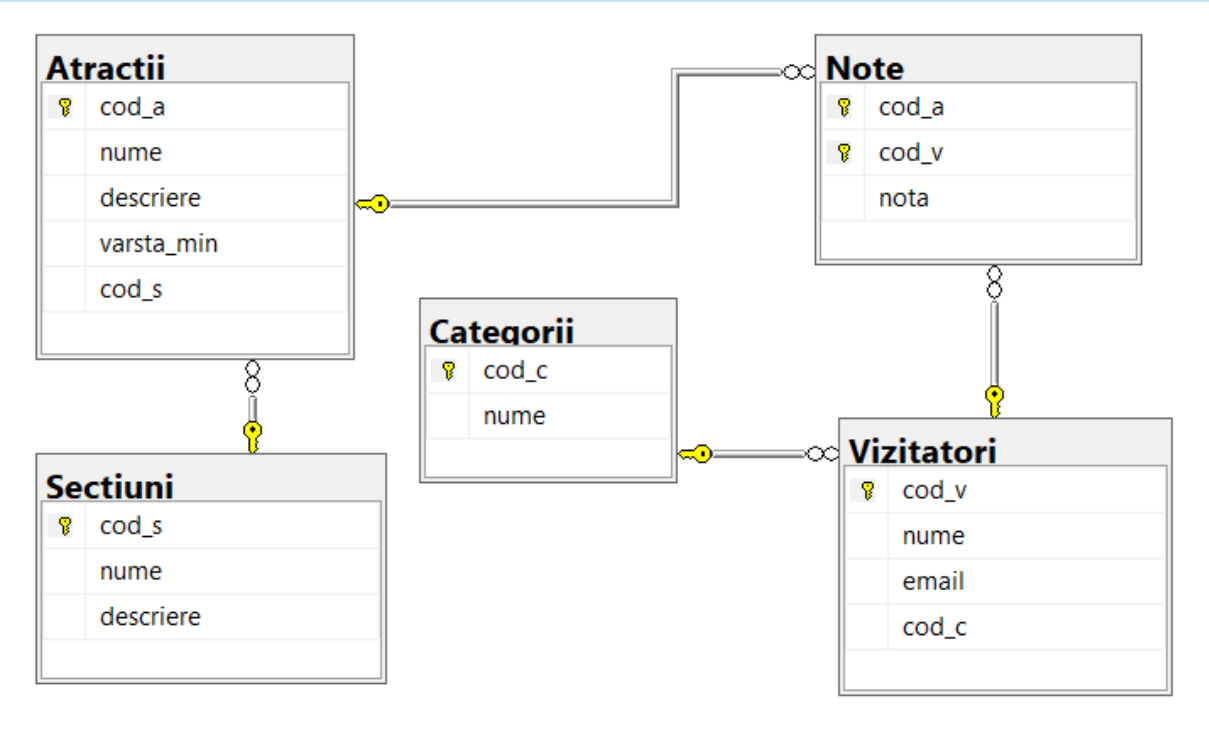
Limbaj de control al fluxului

Mesaje de eroare

- **Error number** (numărul erorii) este 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)

Problemă propusă

- Se dă baza de date cu următoarea structură:



Problemă propusă

Cerințe

1) Să se creeze o procedură stocată care inserează o secțiune nouă în tabelul *Sectiuni*. Procedura va avea doi parametri de intrare: numele și descrierea secțiunii.

2) Să se creeze o procedură stocată care inserează o categorie nouă în tabelul *Categorii*. Procedura va avea un parametru de intrare: numele categoriei. Dacă există deja categoria dată ca parametru, se va afișa un mesaj pe ecran și categoria nu va fi adăugată încă o dată.

3) Să se creeze o procedură stocată care inserează o atracție nouă în tabelul *Atractii*. Procedura va avea 4 parametri de intrare: numele, descrierea, vârsta minimă recomandată și numele secțiunii în care se găsește atracția. Dacă nu există secțiunea dată ca parametru, aceasta va fi adăugată în tabelul *Sectiuni*.

Problemă propusă

4) Să se creeze o procedură stocată care verifică dacă există un vizitator căruia îi corespunde adresa de email dată ca parametru de intrare. Dacă vizitatorul există, se va returna codul acestuia, dacă nu există se va genera un mesaj de eroare.

5) Să se creeze o procedură stocată care inserează o notă în tabelul *Note*. Procedura stocată va avea 3 parametri de intrare: codul atracției, codul vizitatorului și nota. Înainte de inserare, se va verifica dacă există codul atracției și codul vizitatorului în tabelele *Atractii* și *Vizitatori*. Dacă nu există, se va genera un mesaj de eroare. Dacă există, se verifică dacă nota este cuprinsă între 1 și 10. Se va returna un mesaj de eroare dacă nota nu are o valoare validă.

Problemă propusă

6) Să se creeze o procedură stocată care actualizează adresa de email a unui vizitator din tabelul *Vizitatori*. Procedura stocată primește 2 parametri de intrare: codul vizitatorului și noua adresă de email.

7) Să se creeze o procedură stocată care returnează numele vizitatorului, adresa de email și numărul total de note pentru toți vizitatorii care au dat cel puțin o notă unei atracții.

8) Să se creeze o procedură stocată care șterge o atracție din tabelul *Atractii*. Procedura stocată va avea un singur parametru de intrare: numele atracției. Înainte de ștergerea atracției, se va verifica dacă există note pentru acea atracție. În cazul în care există note date acelei atracții, se va afișa pe ecran un mesaj corespunzător și nu se va șterge atracția respectivă.

Bibliografie

- https://learn.microsoft.com/en-us/sql/relational-databases/stored-procedures/stored-procedures-database-engine?view=sql-server-ver16
- https://learn.microsoft.com/en-us/sql/t-sql/language-elements/execute-transact-sql?view=sql-server-ver16
- https://learn.microsoft.com/en-us/sql/relational-databases/system-stored-procedures/sp-executesql-transact-sql?view=sql-server-ver16
- https://learn.microsoft.com/en-us/sql/t-sql/statements/set-nocount-transact-sql?view=sql-server-ver16
- https://learn.microsoft.com/en-us/sql/t-sql/language-elements/variables-transact-sql?view=sql-server-ver16

Bibliografie

- https://learn.microsoft.com/en-us/sql/t-sql/language-elements/control-of-flow?view=sql-server-ver16
- https://learn.microsoft.com/en-us/sql/t-sql/functions/system-functions-transact-sql?view=sql-server-ver16