-Seminar 4-
Funcții definite de utilizator
View. Trigger. Cursor

- sunt funcții care pot fi folosite în interogări
- ret. o valoare simplă (sau un tabel)
- pot avea parametri de intrare

Exemple:

- funcție scalară

```
CREATE FUNCTION ExistăCategorie (@nume VARCHAR(40))
RETURNS BIT AS  -> poate fi modificată în RETURN VARCHAR(20) AS
BEGIN
  IF (EXISTS (SELECT * FROM Categorii WHERE nume=@nume))
      RETURN 1;
  RETURN 0;
END;
GO
```

```
PRINT dbo.ExistăCategorie ('elevi');
```

- pt. ștergere, scriem DROP FUNCTION dbo.ExistăCategorie

- funcție inline table-valued

```
CREATE FUNCTION RetNoteAtracții (@vârstă_min INT)
RETURNS TABLE AS
RETURN SELECT A.nume, N.nota, V.email FROM Atracții A
INNER JOIN Note N ON A.cod_a=N.cod_a
INNER JOIN Vizitatori V ON N.cod_v=V.cod_v
WHERE A.vârstă_min = @vârstă_min;
-- Apelul funcției
SELECT * FROM dbo.RetNoteAtracții (12);
```

```
CREATE FUNCTION NoteAtracții (@email VARCHAR(100))
RETURNS @NoteAtracții TABLE (atracție VARCHAR(100), email VARCHAR(100),
nota REAL, tip_evaluare VARCHAR(10)) AS
BEGIN
INSERT INTO @NoteAtracții (atracție, nota, email)
SELECT A.nume, N.nota, V.email FROM Atracții A INNER JOIN Note N ON A.cod_a=
N.cod_a
INNER JOIN Vizitatori V ON N.cod_v=V.cod_v WHERE V.email = @email;
UPDATE @NoteAtracții SET tip_evaluare = 'pozitivă' WHERE nota >=5.0;
UPDATE @NoteAtracții SET tip_evaluare = 'negativă' WHERE nota <5.0;
RETURN;
END;
```

View

- tabel virtual bazat pe result set-ul unei interogări
- nu stochează date
- cu ajutorul unui view prezentăm date din mai multe tabele ca și cum ar veni din același tabel
- prezintă mereu date actualizate
- numele coloanelor trebuie să fie unice

- CREATE VIEW view_name AS <select_statement> → sintaxa pt. crearea unui view
- ALTER VIEW view_name AS <select_statement> → sintaxa pt. modificarea unui view
- DROP VIEW view_name → pt. ștergerea unui view

```
CREATE VIEW vw_NoteAtractii
AS
SELECT A.nume AS atractie, N.nota, V.nume, V.email FROM Atractii A
INNER JOIN Note N ON A.cod_a=N.cod_a
INNER JOIN Vizitatori V ON N.cod_v = V.cod_v
Go
--Interogarea view-ului
SELECT * FROM vw_NoteAtractii
```

→ putem adăuga WHERE N.nota BETWEEN 5.0 AND 7.0; , ca să returnăm doar înregistrările a căror note se găsesc în intervalul închis [6, 7];

! Nu putem folosi ORDER BY (dar îl putem folosi când interogăm view-ul)

Trigger

- este un tip special de procedură stocată care se execută automat atunci când un anumit eveniment DML sau DDL are loc în baza de date
- nu se poate executa în mod direct
- evenimente DML:
  - INSERT
  - UPDATE
  - DELETE
- evenimente DDL:
  - CREATE
  - ALTER
  - DROP

Sintaxă: (Trigger DML)

```
CREATE TRIGGER
ON {table | view}
[ WITH <dml_trigger_option>[,...n] ]
{FOR | AFTER | INSTEAD OF}
{[INSERT], [,] [UPDATE] [,] [DELETE]}
[ WITH APPEND]
[ NOT FOR REPLICATION]
AS {sql_statement [;] [,...n] | EXTERNAL NAME <method specifier [;]>}
```

- Momentul execuției unui trigger
  - FOR, AFTER – triggerul se execută după ce s-a executat evenimentul declanșator
  - INSTEAD OF – triggerul se execută în locul evenimentului declanșator
- când se execută un trigger, sunt disponibile două tabele speciale:
  - inserted
  - deleted

Exemple: ▪ pt. INSERT → are scopul de a împiedica adăugarea unei înregistrări noi în tabelul Categorii

```
CREATE TRIGGER IntroducereCategorie
ON Categorii
INSTEAD OF INSERT
AS
BEGIN
  RAISERROR('Momentan nu se pot insera date în tabel', 16, 1);
END
```

▪ pt. DELETE → inserează fiecare înregistrare ștearsă din tabelul Categorii într-un tabel CategoriiEliminate:

```
CREATE TRIGGER EliminareCategorie
ON Categorii
AFTER DELETE
AS
BEGIN
  INSERT INTO CategoriiEliminate (cod_c, nume, data_si_ora_eliminării)
  SELECT cod_c, nume, GETDATE() FROM deleted;
END;
```

▪ pt. UPDATE → înregistrează într-un tabel numit ModificăriNote toate modificările de note care au loc în tabelul Note:

```
CREATE TRIGGER ActualizareNota
ON Note
FOR UPDATE
AS
BEGIN
  INSERT INTO ModificăriNote (cod_a, cod_v, nota_inițială, nota_actualizată, data_si_ora_actualizării)
  SELECT i.cod_a, i.cod_v, d.nota, i.nota, GETDATE()
  FROM inserted i INNER join deleted d ON i.cod_a = d.cod_a AND i.cod_v = d.cod_v
END;
```

Cursoare

- deschiderea unui cursor pe un result-set permite procesarea result-set-ului înregistrare cu înregistrare.
- extind procesarea rezultatelor prin faptul că:
  - permit poziționarea la înreg. specifice dintr-un result set
  - ret. o înregistrare sau un grup de înregistrări aflate la poziția curentă din result-set
  - suportă modificarea înregistrărilor aflate în poziția curentă în result-set
  - suportă diferite niveluri de vizibilitate a modificărilor făcute de către alți utilizatori
  - permit instrucțiunilor Tsql-SQL din script-uri, proceduri stocate și trigger-e accesul la datele dintr-un result-set
- necesită anumite instrucțiuni pt. declarare, populare, extragere de date:
  - DECLARE CURSOR și se specifică SELECT care produce result-set-ul cursorului
  - OPEN -> populează cursorul, care execută SELECT încorporată în DECLARE CURSOR
  - FETCH -> extrage înregistrări individual
  - UPDATE / DELETE (dacă este cazul) -> pt. a modifica înreg.
  - CLOSE -> pt. a închide cursorul și elibera unele resurse
  - OPEN (cursorul este declarat, deci poate fi deschis din nou)
  - DEALLOCATE -> pt. a elimina referința cursorului din sesiunea curentă => eliberează toate resursele alocate cursorului, inclusiv numele său
- instrucțiunea FETCH suportă un nr. de opțiuni care suportă returnarea unor înreg. specifice:
  - FETCH FIRST -> ret. prima înregistrare
  - FETCH NEXT -> ret. înregistrarea care urmează după ultima înregistrare ret.
  - FETCH PRIOR -> ret. înreg. care se află înaintea ultimei înregistrări ret.
  - FETCH LAST -> ret. ultima înreg. din cursor
- comportamentul poate fi specificat astfel:
  - SCROLL și INSENSITIVE în instrucțiunea DECLARE CURSOR
  - prin API-urile pt. baze de date

Sintaxă: (iso)

```
DECLARE cursor_name [INSENSITIVE] [SCROLL] CURSOR
FOR select_statement
[FOR { READ ONLY | UPDATE [OF column_name [,...n]] }]
```

(Tsql-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]]]
```