Funcții definite de utilizator. View. Trigger. Cursoare

SEMINAR 4

Funcții definite de către utilizator

- Microsoft SQL Server oferă posibilitatea de a crea **funcții** care pot fi mai apoi folosite în **interogări**
- Funcțiile definite de utilizator:
  - **returnează** o **valoare simplă** (sau un **tabel**)
  - pot avea **parametri de intrare**

- În Microsoft SQL Server sunt disponibile **trei** tipuri de funcții definite de utilizator:

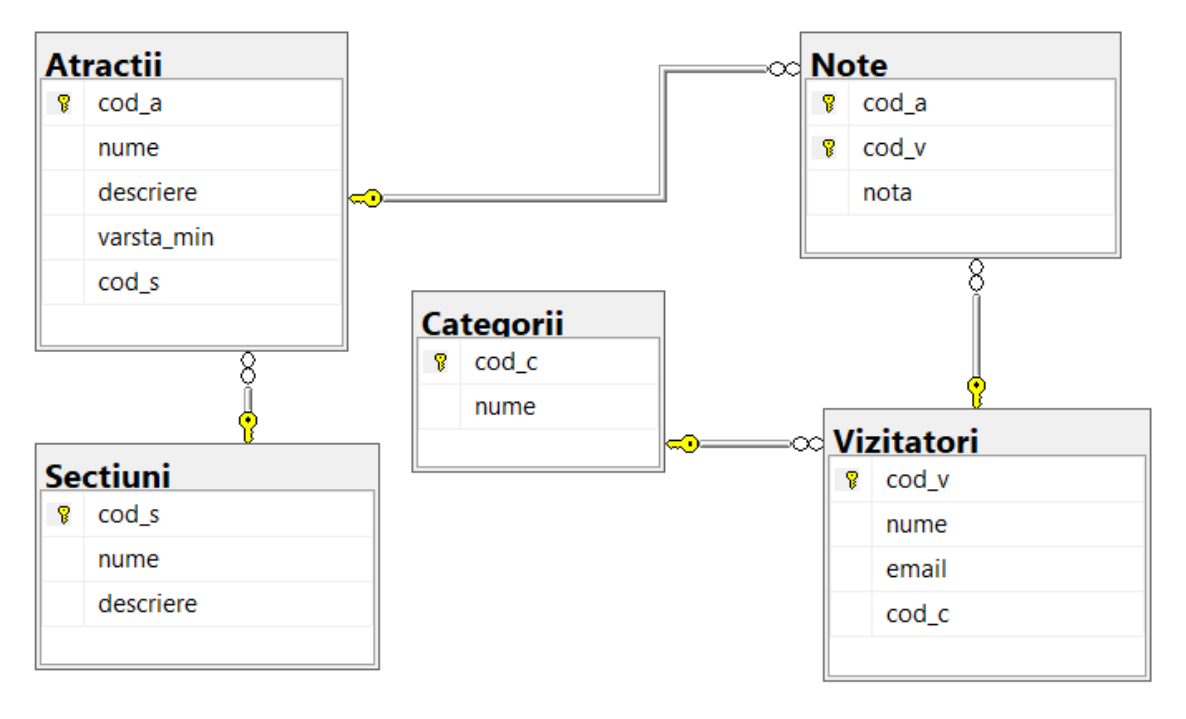
  - Funcții **scalare** (returnează o **singură valoare**)

  - Funcții **inline table-valued** (returnează **un tabel**)

  - Funcții **multi-statement table-valued** (returnează **un tabel**)

Funcții definite de către utilizator

- Se dă o bază de date având următoarea structură:



Funcții definite de către utilizator

- Următoarea funcție **scalară** verifică dacă numele unei categorii există în tabelul *Categorii*:

```
CREATE FUNCTION ExistaCategorie(@nume VARCHAR(70))
RETURNS BIT AS
BEGIN
IF(EXISTS(SELECT * FROM Categorii WHERE nume=@nume))
        RETURN 1;
RETURN 0;
END;
GO
--Apelul funcției
PRINT dbo.ExistaCategorie('elevi');
```

Funcții definite de către utilizator

- Vom **modifica** definiția funcției astfel încât valoarea returnată să fie de tipul **VARCHAR(20)**:

```
ALTER FUNCTION ExistaCategorie(@nume VARCHAR(70))
RETURNS VARCHAR(20) AS
BEGIN
IF(EXISTS(SELECT * FROM Categorii WHERE nume=@nume))
        RETURN 'Exista';
RETURN 'Nu exista';
END;
GO
--Apelul funcției
PRINT dbo.ExistaCategorie('elevi');
```

Funcții definite de către utilizator

- Dacă dorim să ștergem funcția, vom folosi instrucțiunea **DROP FUNCTION**:

```
--Ștergerea funcției

DROP FUNCTION dbo.ExistaCategorie;
```

Funcții definite de către utilizator

- Funcțiile definite de utilizator de **tip inline table-valued** returnează **un tabel** în locul unei singure valori
- Pot fi folosite oriunde poate fi folosit un tabel, de obicei în clauza ***FROM*** a unei interogări
- O funcție definită de utilizator de tip **inline table-valued** conține o singură instrucțiune SQL
- O funcție definită de utilizator de tipul **multi-statement table-valued** returnează un tabel și conține mai multe instrucțiuni SQL, spre deosebire de o funcție **inline table-valued** care conține o singură instrucțiune SQL

Funcții definite de către utilizator

- Următoarea funcție de tipul **inline table-valued** returnează numele atracțiilor, nota primită și adresa de email a vizitatorului, pentru toate atracțiile care au vârsta minimă recomandată egală cu cea dată ca parametru:

```
CREATE FUNCTION ReturneazaNoteAtractii(@varsta_min INT)

RETURNS TABLE AS

RETURN SELECT A.nume, N.nota, 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

WHERE A.varsta_min=@varsta_min;

--Apelul funcției

SELECT * FROM dbo.ReturneazaNoteAtractii(12);
```

Funcții definite de către utilizator

```
CREATE FUNCTION NoteAtractii(@email VARCHAR(100))
RETURNS @NoteAtractii TABLE (atractie VARCHAR(100), email VARCHAR(100),
nota REAL, tip_evaluare VARCHAR(10)) AS
BEGIN
INSERT INTO @NoteAtractii (atractie, nota, email)
SELECT A.nume, N.nota, 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 WHERE V.email=@email;
UPDATE @NoteAtractii SET tip_evaluare='pozitiva' WHERE nota>=5.0;
UPDATE @NoteAtractii SET tip_evaluare='negativa' WHERE nota<5.0;
RETURN;
END;
```

Funcții definite de către utilizator

- Funcția **multi-statement table-valued** definită anterior primește ca parametru o valoare ce reprezintă adresa de email a unui vizitator și returnează **un tabel** care conține atracțiile evaluate de către vizitatorul respectiv
- Tabelul returnat conține numele atracției, nota primită, adresa de email a vizitatorului și **tipul evaluării**:
  - dacă **nota este mai mică decât 5**, atunci evaluarea va fi **negativă**
  - dacă **nota este mai mare sau egală cu 5**, atunci evaluarea va fi **pozitivă**)
- Funcția va fi apelată în modul următor:

```
SELECT * FROM dbo.NoteAtractii('andrei@gmail.com');
```

View

- Un **view** este un tabel virtual bazat pe result set-ul unei interogări
- Conține înregistrări și coloane ca un tabel real
- Un **view** nu stochează date, stochează definiția unei interogări
- Cu ajutorul unui **view** putem prezenta date din mai multe tabele ca și cum ar veni din același tabel
- De fiecare dată când un **view** este interogat, motorul bazei de date va recrea datele folosind **instrucțiunea SELECT** specificată la crearea **view-ului**, astfel că un **view** va prezenta întotdeauna **date actualizate**
- **Numele coloanelor** dintr-un **view** trebuie să fie **unice** (în cazul în care avem două coloane cu același nume provenind din tabele diferite, putem folosi un **alias** pentru una dintre ele)

View

- Sintaxa pentru crearea unui view:

```
CREATE VIEW view_name AS <select_statement>;
```

- Sintaxa pentru modificarea unui view:

```
ALTER VIEW view_name AS <select_statement>;
```

- Sintaxa pentru ștergerea unui view:

```
DROP VIEW view_name;
```

View

- Următorul view returnează numele atracțiilor, nota, numele și adresa de email a vizitatorilor:

```
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;
```

View

- Vom modifica definiția view-ului, astfel încât acesta să returneze doar înregistrările a căror note se găsesc în intervalul închis [5, 7]:

```
ALTER 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

WHERE N.nota BETWEEN 5.0 AND 7.0;

GO

--Interogarea view-ului

SELECT * FROM vw_NoteAtractii;
```

View

- Când interogăm view-ul putem specifica explicit coloanele pe care dorim să le returnăm în result-set:

```
SELECT atractie, nota, nume, email FROM vw_NoteAtractii;
```

- Sau putem folosi * dacă returnăm toate coloanele în result-set:

```
SELECT * FROM vw_NoteAtractii;
```

- Dacă view-ul nu mai este necesar, poate fi șters folosind instrucțiunea **DROP VIEW**:

```
DROP VIEW vw_NoteAtractii;
```

View

- **Nu** se poate folosi clauza **ORDER BY** în definiția unui view (decât dacă se specifică în definiția view-ului clauza **TOP**, **OFFSET** sau **FOR XML**)
- Dacă dorim să ordonăm înregistrările din result-set, putem folosi clauza **ORDER BY** atunci când interogăm view-ul
- Pentru a afișa definiția unui view, putem folosi funcția **OBJECT_DEFINITION** sau procedura stocată **sp_helptext**:

```
PRINT OBJECT_DEFINITION(OBJECT_ID('schema_name.view_name'));

EXEC sp_helptext 'schema_name.view_name';
```

View

- Se pot insera date într-un view doar dacă inserarea afectează un singur *base table* (în cazul în care view-ul conține date din mai multe tabele)
- Se pot actualiza date într-un view doar dacă actualizarea afectează un singur *base table* (în cazul în care view-ul conține date din mai multe tabele)
- Se pot șterge date dintr-un view doar dacă view-ul conține date dintr-un singur tabel
- Operațiunile de inserare într-un view sunt posibile doar dacă view-ul expune toate coloanele care nu permit valori NULL
- Numărul maxim de coloane pe care le poate avea un view este 1024

Tabele sistem

- Tabelele sistem sunt niște tabele speciale care conțin informații despre toate obiectele create într-o bază de date, cum ar fi:
  - Tabele
  - Coloane
  - Proceduri stocate
  - Trigger-e
  - View-uri
  - Funcții definite de utilizator
  - Indecși

Tabele sistem

- Tabelele sistem sunt gestionate de către **server** (nu se recomandă modificarea lor direct de către utilizator)
- Exemple:

  sys.objects – conține câte o înregistrare pentru fiecare obiect creat în baza de date, cum ar fi: procedură stocată, trigger, tabel, constrângere

  sys.columns – conține câte o înregistrare pentru fiecare coloană a unui obiect care are coloane, cum ar fi: tabel, funcție definită de utilizator care returnează un tabel, view

  sys.databases – conține câte o înregistare pentru fiecare bază de date existentă pe server

Trigger

- Trigger-ul 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
- Fiecare trigger (DML) aparține unui singur tabel

Trigger DML

- Sintaxa:

```
CREATE TRIGGER trigger_name

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 [ ; ] > }
```

Trigger

- **Momentul** execuției unui trigger
  - **FOR**, **AFTER** (se pot defini mai multe trigger-e de acest tip) – trigger-ul se execută după ce s-a executat evenimentul declanșator
  - **INSTEAD OF** – trigger-ul se execută în locul evenimentului declanșator
- Dacă se definesc mai multe trigger-e pentru aceeași acțiune (eveniment), ele se execută în **ordine aleatorie**
- Când se execută un trigger, sunt disponibile două **tabele speciale:**
  - **inserted**
  - **deleted**

Trigger pentru INSERT - Exemplu

- Următorul trigger are scopul de a impiedica adăugarea unor înregistrări noi în tabelul *Categorii*:

```
CREATE TRIGGER IntroducereCategorie

ON Categorii

INSTEAD OF INSERT

AS

BEGIN

        RAISERROR('Momentan nu se pot insera date in acest tabel',16,1);

END;
```

Trigger pentru DELETE - Exemplu

- Următorul trigger inserează fiecare înregistrare ștearsă din tabelul *Categorii* într-un tabel numit *CategoriiEliminate*:

```
CREATE TRIGGER EliminareCategorie

ON Categorii

AFTER DELETE

AS

BEGIN

INSERT INTO CategoriiEliminate (cod_c, nume, data_si_ora_eliminarii)

SELECT cod_c, nume, GETDATE() FROM deleted;

END;
```

Trigger pentru DELETE - Exemplu

- Instrucțiunea folosită la crearea tabelului *CategoriiEliminate* este următoarea:

```
CREATE TABLE CategoriiEliminate
(cod_e INT PRIMARY KEY IDENTITY,
 cod_c INT,
 nume VARCHAR(70),
 data_si_ora_eliminarii DATETIME
);
```

Trigger pentru UPDATE - Exemplu

- Următorul trigger înregistrează într-un tabel numit *ModificariNote* toate modificările de note care au loc în tabelul *Note*:

```
CREATE TRIGGER ActualizareNota
ON Note
FOR UPDATE
AS
BEGIN
INSERT INTO ModificariNote (cod_a, cod_v, nota_initiala, nota_actualizata,
data_si_ora_actualizarii) 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;
```

Trigger pentru UPDATE - Exemplu

- Instrucțiunea folosită la crearea tabelului *ModificariNote* este următoarea:

```
CREATE TABLE ModificariNote
(cod_m INT PRIMARY KEY IDENTITY,
 cod_a INT,
 cod_v INT,
 nota_initiala REAL,
 nota_actualizata REAL,
 data_si_ora_actualizarii DATETIME
);
```

Clauza OUTPUT

- Cu ajutorul clauzei **OUTPUT** avem acces la înregistrările modificate, șterse sau adăugate
- În exemplul de mai jos se actualizează numele categoriei cu valoarea ‘seniori’ din tabelul *Categorii* și se afișează într-un result-set valoarea din coloana *cod_c*, 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 Categorii SET nume='pensionari'
OUTPUT inserted.cod_c, deleted.nume nume_initial, inserted.nume nume_actual,
GETDATE() AS data_si_ora, SUSER_SNAME() AS server_user
WHERE nume='seniori';
```

Cursoare

- Sunt anumite situații în care este necesară procesarea pe rând a fiecărei înregistrări dintr-un 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)

Cursoare

- 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

  - 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

Cursoare

- Instrucțiunea **FETCH** suportă un număr de opțiuni care permit returnarea unor înregistrări specifice:

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

  - **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

Cursoare

- **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)

- **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)

Cursoare

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)
- prin specificarea comportamentului unui cursor cu ajutorul tipurilor de cursoare
  - de obicei API-urile 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**

Cursoare

- 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:

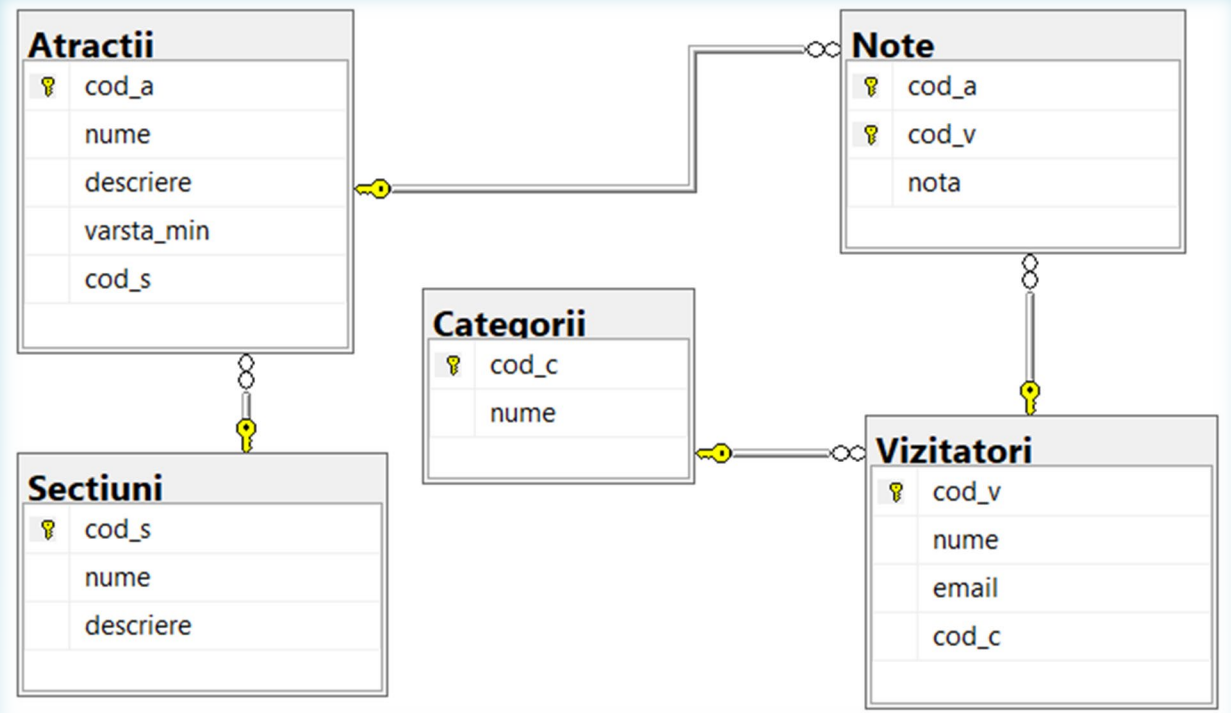
```
DECLARE cursor_name CURSOR [ LOCAL | GLOBAL ]
[ FORWARD_ONLY | SCROLL ]
[ STATIC | KEYSET | DYNAMIC | FAST_FORWARD ]
[ READ_ONLY | SCROLL_LOCKS | OPTIMISTIC ]
[ TYPE_WARNING ]
FOR select_statement
[ FOR UPDATE [ OF column_name [ ,...n ] ] ]
```

Cursoare - Exemplu

```
DECLARE @nume VARCHAR(100), @email VARCHAR(100), @categorie VARCHAR(70);
DECLARE cursorvizitatori CURSOR FAST_FORWARD FOR
SELECT V.nume, V.email, C.nume  FROM Vizitatori V INNER JOIN Categorii C ON
V.cod_c=C.cod_c;
OPEN cursorvizitatori;
FETCH NEXT FROM cursorvizitatori INTO @nume, @email, @categorie;
WHILE @@FETCH_STATUS=0
BEGIN
PRINT 'Vizitatorul '+@nume+ ', '+@email+ ' face parte din categoria '+@categorie;
FETCH NEXT FROM cursorvizitatori INTO @nume, @email, @categorie;
END
CLOSE cursorvizitatori;
DEALLOCATE cursorvizitatori;
```

Problemă propusă

- Se dă o bază de date având următoarea structură:



Problemă propusă

Cerințe:

1) Să se creeze o funcție scalară care primește ca parametru numele unei categorii și returnează codul acesteia.

2) Creați un trigger care împiedică execuția operațiilor de ștergere din tabelul *Categorii* și afișează un mesaj corespunzător.

3) Creați un view care afișează toate înregistrările din tabelul *Categorii* al căror nume este egal cu ‘pensionari’ sau ‘copii’.

4) Creați un view care afișează toate înregistrările din tabelul *Sectiuni* al căror nume începe cu litera C.

5) Creați o funcție de tip inline table valued care returnează toate înregistrările din tabelul *Sectiuni* al căror nume se termină cu o literă dată ca parametru de intrare și au cel puțin două caractere.

6) Creați un view care care afișează numele vizitatorilor, nota și numele atracției.

Bibliografie

- https://learn.microsoft.com/en-us/sql/t-sql/statements/create-function-transact-sql?view=sql-server-ver16
- https://learn.microsoft.com/en-us/sql/t-sql/statements/create-view-transact-sql?view=sql-server-ver16
- https://learn.microsoft.com/en-us/sql/t-sql/queries/output-clause-transact-sql?view=sql-server-ver16
- https://learn.microsoft.com/en-us/sql/t-sql/statements/create-trigger-transact-sql?view=sql-server-ver16
- https://learn.microsoft.com/en-us/sql/relational-databases/cursors?view=sql-server-ver16
- https://learn.microsoft.com/en-us/sql/relational-databases/system-catalog-views/catalog-views-transact-sql?view=sql-server-ver16