Ministerul Educației al Republicii Moldova
Universitatea Tehnică a Moldovei
Facultatea Calculatoare, Informatică și Microelectronică
Ingineria Software

REPORT

Laboratory work Nr.8

Theme: Management of Visions and Table Expressions

Realized by: Popa Eugeniu
FAF-202

Chișinău, 2021

The theoretical part

1. Types of temporary tables. The use of temporary tables.
   There are 2 types of Temporary Tables: Local Temporary Table, and Global Temporary Table. Temporary tables behave just like normal ones; you can sort, filter and join them as if they were permanent tables. Because SQL Server has less logging and locking overheads for temporary tables (after all, you're the only person who can see or use the temporary table you've created), they execute more quickly.

2. Visions and their role in the database.
   A view in SQL Server is like a virtual table that contains data from one or multiple tables. It does not hold any data and does not exist physically in the database. Similar to a SQL table, the view name should be unique in a database. It contains a set of predefined SQL queries to fetch data from the database.

3. Methods and features for creating visions in SQL Server.
   To create a vision, use the CREATE VIEW statement. The instruction assigns a name of the vision and specifies the query that defines the vision. Optionally, the CREATE VIEW statement can name columns on the newly created view. If a list of column names is present, it must have the same number of elements as the fumigation query columns.

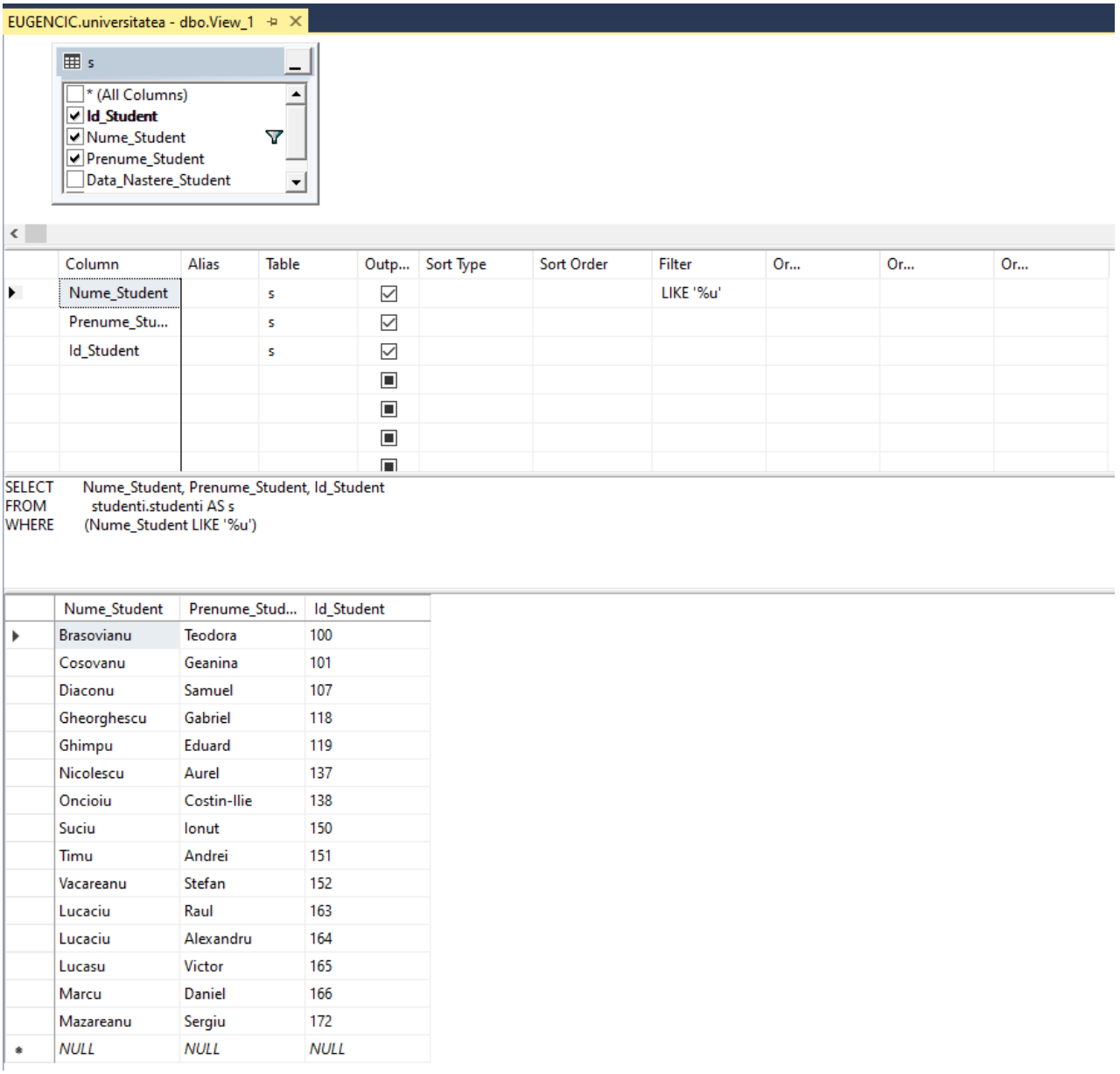
4. Situations in which the creation of the vision is possible or impossible.
   You cannot pass parameters to SQL Server views.
   You cannot use an Order By clause with views without specifying FOR XML or TOP.
   Views cannot be created on Temporary Tables.
   You cannot associate rules and defaults with views.

5. The basic syntax of the CREATE VIEW statement. WITH CHECK OPTION and SCHEMABINDING options.
   CREATE [OR ALTER] VIEW [schema_name .] view_name [(column [,...n])]
   [WITH <view_attribute> [,...n]]
   AS select_statement
   [WITH CHECK OPTION]
   [;]
   <view_attribute> ::=
   {
   [ENCRYPTION]
   [SCHEMABINDING]
   [VIEW_METADATA]
   }

6. What are the characteristics of the table-expression? But of recursive queries using virtual representations?
   A SQL CTE (Common Table Expression) defines a temporary result set which you can then use in a SELECT statement. It becomes a convenient way to manage complicated queries. You define Common Table Expressions using the WITH statement. You can define one or more common table expression in this fashion.

The practical part

1. Create two visions based on the questions formulated in two exercises indicated in chapter 4. The first view should be built in the Query Editor, and the second, using View Designer.




```sql
--1. Sa se creeze doua viziuni in baza interogarilor formulate in doua exercitii indicate din capitolul 4.
--Prima viziune sa fie construita in Editorul de interogari, iar a doua, utilizand View Designer.

-- Ex.13
CREATE VIEW View_2 AS
SELECT d.Id_Disciplina, d.Disciplina
FROM studenti.studenti_reusita sr
INNER JOIN studenti.studenti s
ON sr.Id_Student = s.Id_Student
INNER JOIN plan_studii.discipline d
ON sr.Id_Disciplina = d.Id_Disciplina
WHERE s.Nume_Student = 'Florea' AND s.Prenume_Student ='Ioan';

SELECT * FROM View_2
```

2. Write an example of INSERT, UPDATE, DELETE instructions on the created visions. Add the respective comments regarding the results of their execution.

```
--2. Sa se scrie cate un exemplu de instructiuni INSERT, UPDATE, DELETE asupra viziunilor
--create. Sa se adauge comentariile respective referitoare la rezultatele executarii acestor
--instructiuni.

--2.1
--INSERT
INSERT INTO View_1(Id_Student, Nume_Student, Prenume_Student)
VALUES(450, 'Popa', 'Eugeniu')

--UPDATE
UPDATE View_1
SET Nume_Student = 'Ionel'
WHERE Nume_Student = 'Ionut'

--DELETE
DELETE FROM View_1
WHERE Nume_Student = 'Ghimpu'
--The DELETE statement will not work here, because it is not possible, while the table has refference with "studenti_reusita" table.

--2.2
--INSERT
INSERT INTO View_2(Id_Disciplina, Disciplina)
VALUES(40, 'Stiinte aplicate')

--UPDATE
UPDATE View_2
SET Disciplina = 'Sisteme de calcul'
WHERE Disciplina = 'Sisteme de calculare'

--DELETE
DELETE FROM View_2
WHERE Disciplina = 'Stiinte aplicate'
```

3. Write SQL statements that would change the views created (in Exercise 1) so that it should not be possible to modify or delete the tables on which they are defined and the visions not to accept DML operations, if the conditions of the WHERE clause are not satisfied.

```
--3. Sa se scrie instructiunile SQL care ar modifica viziunile create (in exercitiul 1) in asa fel, incat sa nu fie posibila modificarea sau stergerea
--tabelelor pe care acestea sunt definite si viziunile sa nu accepte operatiuni DML, daca conditiile clauzei WHERE nu sunt satisfacute.

--3.1
ALTER VIEW View_1
WITH SCHEMABINDING AS
(SELECT s.Nume_Student, s.Prenume_Student
FROM studenti.studenti s
WHERE s.Nume_Student LIKE '%u')
WITH CHECK OPTION;

--3.2
ALTER VIEW View_2
WITH SCHEMABINDING AS
(SELECT d.Id_Disciplina, d.Disciplina
FROM studenti.studenti_reusita sr
INNER JOIN studenti.studenti s
ON sr.Id_Student = s.Id_Student
INNER JOIN plan_studii.discipline d
ON sr.Id_Disciplina = d.Id_Disciplina
WHERE s.Nume_Student = 'Florea' AND s.Prenume_Student ='Ioan')
WITH CHECK OPTION;
```

4. Write the instructions for testing the newly defined properties.

```
--4. Sa se scrie instructiunile de testare a proprietatilor noi definite.
SELECT * FROM View_1
SELECT * FROM View_2

INSERT INTO View_1(Id_Student, Nume_Student, Prenume_Student)
VALUES(452, 'Popa', 'Eugeniu')

INSERT INTO View_2(Id_Disciplina, Disciplina)
VALUES(45, 'Stiinte aplicate')
```

5. Rewrite 2 queries formulated in the exercises in Chapter 4, so that the nested queries to be rendered as CTE expressions.

```
--5. Sa se rescrie 2 interogari formulate in exercitiile din capitolul 4, in asa fel,
--incat interogarile imbricate sa fie redate sub forma expresiilor CTE.

--5.1
WITH CTE1
AS(SELECT s.Nume_Student, s.Prenume_Student FROM studenti.studenti s
WHERE s.Nume_Student LIKE '%u')

SELECT * FROM CTE1;

--5.2
WITH CTE2
AS(SELECT d.Id_Disciplina, d.Disciplina
FROM studenti.studenti_reusita sr
INNER JOIN studenti.studenti s
ON sr.Id_Student = s.Id_Student
INNER JOIN plan_studii.discipline d
ON sr.Id_Disciplina = d.Id_Disciplina
WHERE s.Nume_Student = 'Florea' AND s.Prenume_Student ='Ioan')

SELECT * FROM CTE2;
```

6. Consider an oriented graph, like the one in the figure below and either you want to follow the path from the node where id = 3 to the node where id = 0. Represent the oriented graph in the form of a recursive table expression.
Observe the post UNION ALL instruction of the recursive member, as well as the part of to UNION ALL represented by the anchor member.

```
--6. Se considera un graf orientat, precum cel din figura de mai jos si fie se doreste parcursa calea de la nodul id = 3 la nodul unde id = 0. Sa se faca reprezentarea grafului
--orientat in forma de expresie-tabel recursiv. Sa se observe instructiunea de dupa UNION ALL a membrului recursiv, precum si partea de pana la UNION ALL reprezentata de membrul-ancora.

DECLARE @Graph_tab TABLE
(
ID INT,
prev_ID INT
)

INSERT @Graph_tab
SELECT 5, NULL UNION ALL
SELECT 4, NULL UNION ALL
SELECT 3, NULL UNION ALL
SELECT 2, 4 UNION ALL
SELECT 2, 3 UNION ALL
SELECT 1, 2 UNION ALL
SELECT 0, 5 UNION ALL
SELECT 0, 1;

WITH graph_ex6
AS
(SELECT *, 0 AS generation FROM @Graph_tab WHERE ID = 3
UNION ALL
SELECT graph. *, generation + 1
FROM @Graph_tab AS graph
INNER JOIN graph_ex6
ON graph.prev_ID = graph_ex6.ID)

SELECT * FROM graph_ex6;
```