

**MINISTERUL EDUCAŢIEI ȘI CERCETĂRII AL REPUBLICII MOLDOVA**

**Universitatea Tehnică a Moldovei**

**Facultatea Calculatoare, Informatică şi Microelectronică**

**Departamentul Ingineria Software și Automatică**

**Programul de studii: Tehnologia informației**

**Raport**

*Lucrare de laborator nr.6*

*La Baze de Date*

|  |  |  |
| --- | --- | --- |
| **A efectuat:** |  | **Popa Cătălin, st.gr. TI-211** |
| **A verificat:** |  | **Olga Grosu** |

**Chişinău 2024**

**Tema:** SQL Analitic, funcții de clasificare și expresii tabel

**10.1 Date generalizate cu funcții de agregare**

**Funcția GROUPING**

Funcția GROUPING este o funcție utilizată în SQL împreună cu operatorii ROLLUP și CUBE pentru a determina dacă o valoare specifică dintr-o coloană de grupare reprezintă o valoare subtotală sau totală. Atunci când se utilizează operatorii ROLLUP sau CUBE, valorile din rezultat pot fi fie valori de grupare reale (cum ar fi identificatorii magazinului sau furnizorului), fie valori de subtotal sau total. Funcția GROUPING returnează 1 atunci când valoarea specificată este o valoare de subtotal sau total și 0 în caz contrar.

**Operatorul ROLLUP**

Operatorul ROLLUP este folosit în SQL pentru a genera subtotaluri și totaluri pentru grupuri de date într-o interogare de agregare. Atunci când este utilizat, ROLLUP creează rânduri suplimentare în rezultatul interogării care conțin subtotaluri și totaluri pentru grupuri de coloane specificate.

--In aceasta interogare, am folosit functia GROUPING pentru a determina daca o valoare specifica este subtotal sau total. ROLLUP este folosit pentru a specifica coloanele dupa care sa se efectueze gruparea.

--Rezultatul va contine subtotaluri pentru fiecare magazin, subtotaluri pentru fiecare furnizor si un total global pentru toate produsele.

SELECT

CASE WHEN GROUPING(MagazinID) = 1 THEN 'TotalMagazine' ELSE ISNULL(CONVERT(VARCHAR(255), MagazinID), 'TotalFurnizori') END AS Magazin,

CASE WHEN GROUPING(FurnizorID) = 1 THEN 'TotalFurnizori' ELSE ISNULL(CONVERT(VARCHAR(255), FurnizorID), 'TotalGlobal') END AS Furnizor,

SUM(PretProdus) AS TotalPretProduse

FROM

Produse

GROUP BY

ROLLUP (MagazinID, FurnizorID);

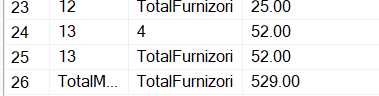


Figura 1 – ROLLUP

**Operatorul CUBE**

Operatorul CUBE este similar cu operatorul ROLLUP, dar generează toate combinațiile posibile ale grupărilor specifice, inclusiv subtotalurile și totalurile pentru fiecare combinație. Acesta oferă o analiză multidimensională mai detaliată a datelor.

--Operatorul CUBE este similar cu operatorul ROLLUP, dar genereaza toate combinatiile posibile ale gruparilor specifice,

--inclusiv subtotalurile si totalurile pentru fiecare combinatie. Acesta ofera o analiza multidimensionala mai detaliata a datelor.

SELECT

CASE

WHEN GROUPING(MagazinID) = 1 THEN 'TotalMagazine'

ELSE ISNULL(CONVERT(VARCHAR(255), MagazinID), 'TotalFurnizori')

END AS Magazin,

CASE

WHEN GROUPING(FurnizorID) = 1 THEN 'TotalFurnizori'

ELSE ISNULL(CONVERT(VARCHAR(255), FurnizorID), 'TotalGlobal')

END AS Furnizor,

SUM(PretProdus) AS TotalPretProduse

FROM

Produse

GROUP BY

CUBE (MagazinID, FurnizorID);

În acest caz, rezultatul va conține subtotaluri și totaluri pentru fiecare combinație posibilă de magazin și furnizor, precum și subtotaluri și totaluri pentru fiecare furnizor și un total global pentru toate produsele. Operatorul CUBE generează un set de rezultate mult mai mare decât ROLLUP, deoarece include toate combinațiile posibile, ceea ce poate fi util pentru analiza detaliată a datelor multidimensionale.



Figura 2 – CUBE

**Clauza GROUPING SETS**

Clauza GROUPING SETS este utilizată în SQL pentru a specifica mai multe seturi de grupuri într-o singură interogare. Acest lucru permite să obținem totaluri pentru grupuri multiple de coloane în același rezultat al interogării. Pentru a exemplifica utilizarea clauzei GROUPING SETS pe tabela "Angajati", vom crea o interogare care să grupeze datele în funcție de mai multe coloane și să afișeze totaluri pentru fiecare set de grupuri. Să presupunem că dorim să obținem totaluri pentru următoarele seturi de grupuri:

* Numărul de angajați pentru fiecare funcție.
* Numărul de angajați pentru fiecare an de naștere.
* Numărul total de angajați.

--Rezultatul interogarii va contine totaluri separate pentru fiecare set de grupuri specificat in clauza GROUPING SETS,

--permitandu-ne sa obtinem o analiza mai detaliata a datelor din diferite perspective.

SELECT

FunctieID,

AnulNasterii,

COUNT(\*) AS NumarAngajati

FROM

Angajati

GROUP BY

GROUPING SETS ((FunctieID), (AnulNasterii), ());

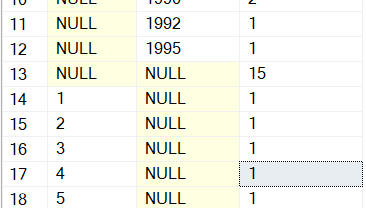


Figura 3 – Clauza GROUPING SETS

**10.2 Funcții de clasificare**

Funcțiile de clasificare (sau de rang) în SQL sunt funcții care permit atribuirea unui clasament sau unui rang valorilor într-o anumită ordine, pe baza unor criterii specifice. Aceste funcții sunt utile în situații în care dorim să identificăm cele mai mari sau cele mai mici valori dintr-un set de date sau să atribuim un clasament pentru fiecare înregistrare în funcție de anumite criterii.

**Funcția RANK**

Functia RANK() este o functie de clasificare in SQL care atribuie un clasament sau un rang fiecarei inregistrari dintr-un set de date, in functie de valoarea specificata. Aceasta este utila pentru a identifica pozitia relativa a unei inregistrari in raport cu celelalte inregistrari din setul de date, pe baza unei anumite ordini.

--Aceasta interogare va atribui un clasament fiecarui bon in functie de suma totala, ordonata descrescator. Clasamentul va

--fi asignat bonurilor in functie de cat de mare este suma totala a acestora. Rezultatul interogarii va include informatiile despre bon

--(BonID, CumparatorID, AngajatID, SumaTotala) si clasamentul fiecarui bon in raport cu suma totala.

SELECT

BonID,

CumparatorID,

AngajatID,

SumaTotala,

RANK() OVER (ORDER BY SumaTotala DESC) AS Clasament

FROM

Bon;

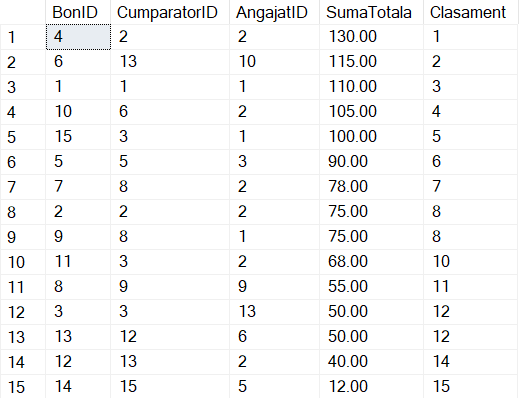


Figura 4 Funcția RANK()

**Funcția DENSE\_RANK**

Funcția DENSE\_RANK() atribuie un clasament distinct fiecărei înregistrări dintr-un set de date, fără să omită clasamente între valori egale. Este similară cu funcția RANK(), dar nu omită clasamentele între valori egale.

--Aceasta interogare va atribui un clasament distinct fiecarui produs in functie de pretul sau, ordonat descrescator. Clasamentul va fi

--asignat produselor in functie de cat de mare este pretul lor, fara sa omita clasamente intre produsele care au acelasi pret. Rezultatul

--interogarii va include informatiile despre fiecare produs si clasamentul sau in functie de pret.

SELECT

ProdusID,

NumeProdus,

MagazinID,

PretProdus,

DENSE\_RANK() OVER (ORDER BY PretProdus DESC) AS ClasamentPret

FROM

Produse;

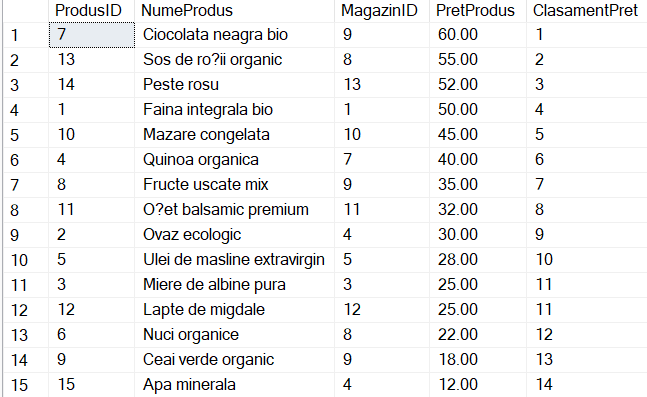


Figura 5 – Funcția DENSE\_RANK

**Funcția NTILE**

Funcția dată permite împărțirea rândurilor în subgrupuri(aproximativ egale in mărime) în funcție de un anumit număr de subgrupuri și de ordonarea setului de date.

--Aceasta interogare va atribui fiecarui produs un numar de quartil (1, 2, 3 sau 4) in functie de pretul sau. Grupurile

--sunt create astfel incat sa aiba numarul egal de produse in fiecare grup, iar produsele sunt clasificate in interiorul fiecarui quartil in

--functie de pret. De exemplu, produsele cu pretul cel mai mic vor fi in quartilul 1, iar cele cu pretul cel mai mare vor fi in quartilul 4.

SELECT

ProdusID,

NumeProdus,

MagazinID,

PretProdus,

NTILE(4) OVER (ORDER BY PretProdus) AS Quartil

FROM

Produse;

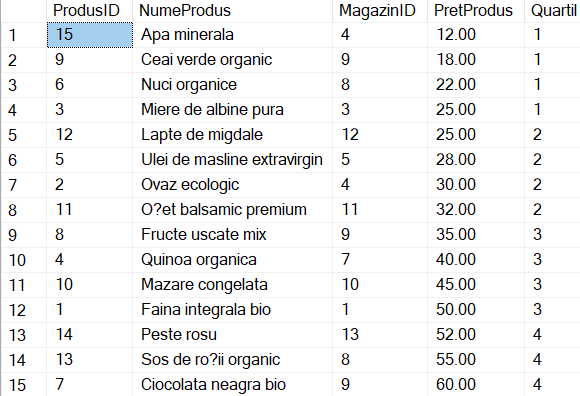


Figura 6 – Funcția NTILE

**Funcția ROW\_NUMBER**

Funcția ROW\_NUMBER() este o funcție de analiză în SQL care atribuie un număr secvențial unic fiecărei înregistrări dintr-un set de date, în funcție de ordinea specificată. Aceasta este utilă pentru a identifica fiecare înregistrare distinctă și pentru a le număra secvențial în funcție de criteriile de sortare.

--Aceasta interogare va atribui un numar secvential unic fiecarui produs in functie de pretul sau, ordonat descrescator.

--Fiecare produs va primi un numar secvential unic, incepand de la 1 pentru cel mai scump produs si continuand in ordinea descrescatoare a pretului.

SELECT

ROW\_NUMBER() OVER (ORDER BY PretProdus DESC) AS NumarSecvential,

ProdusID,

NumeProdus,

MagazinID,

PretProdus,

FurnizorID

FROM

Produse;

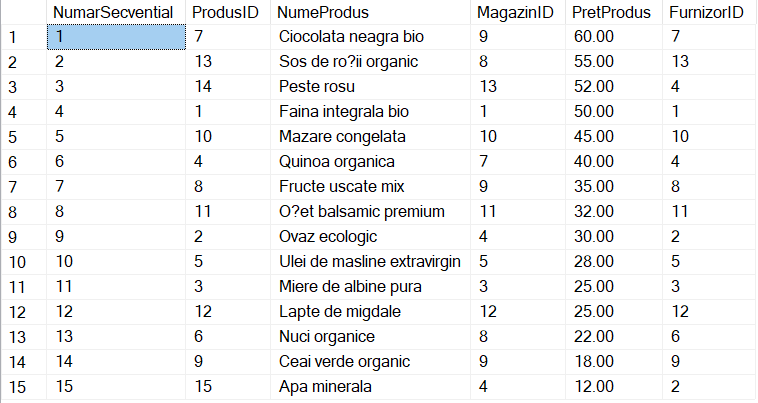


Figura 7 – Funcția ROW\_NUMBER

**10.3 Interogări încruceșate**

Crearea unei interogări înruceșate constă în rotirea rezultatelor unei interogări astfel, încât rândurile să fie afișate vertical, iar coloanele orizontal.

**Operatorul PIVOT**

Operatorul PIVOT este folosit în SQL pentru a transforma datele dintr-o coloană într-o structură de coloane pivot, adică pentru a roti datele din rânduri în coloane, bazându-se pe o valoare specificată.

Funcția asociată cu operatorul PIVOT este funcția de agregare, care este folosită pentru a sintetiza valorile într-o coloană pivotată. Această funcție poate fi, de exemplu, SUM(), AVG(), COUNT(), MAX(), MIN().

--In aceasta interogare, functia AVG() este folosita pentru a calcula media pretului fiecarui produs in fiecare magazin. Apoi, rezultatul este pivotat

--pentru a afisa pretul mediu pentru fiecare magazin sub forma de coloane separate.

SELECT \*

FROM (

SELECT NumeProdus, PretProdus, MagazinID

FROM Produse

) AS SourceTable

PIVOT (

AVG(PretProdus)

FOR MagazinID IN ([1], [2], [3], [4])

) AS PivotTable;

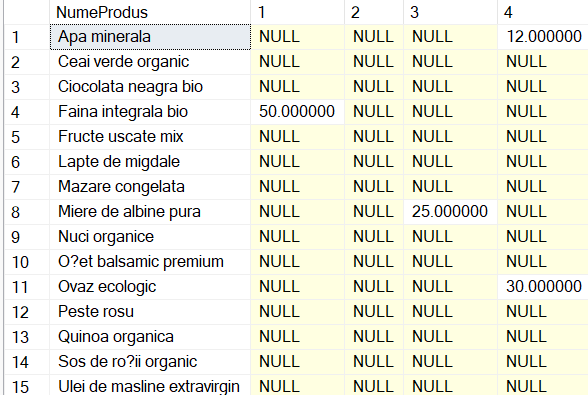


Figura 8 – Operatorul PIVOT

**Operatorul UNPIVOT**

Operatorul UNPIVOT este opusul operatorului PIVOT. În timp ce PIVOT rotește datele din rânduri în coloane, UNPIVOT transformă datele din coloane în rânduri, permițând astfel o analiză mai detaliată a datelor. UNPIVOT este util atunci când avem date pivotate (coloane separate pentru diferite categorii de informații) și dorim să le transformăm înapoi în formă tabulară (date în rânduri și coloane) pentru a le putea analiza sau prelucra mai ușor.

--Astfel, operatorul UNPIVOT transforma datele din coloanele "DataProducere" si "DataExpirare"

--in randuri separate pentru fiecare produs si tip de data

SELECT ProdusID, TipData, Data

FROM (

SELECT ProdusID, DataProducere, DataExpirare

FROM Valabilitate

) AS SourceTable

UNPIVOT (

Data FOR TipData IN (DataProducere, DataExpirare)

) AS UnpivotTable;

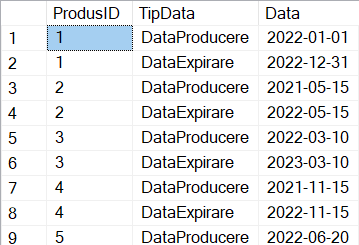


Figura 9 – Operatorul UNPIVOT

**10.4 Tabele derivate**

Tabelele derivate sunt tabele temporare sau virtuale care sunt create ca rezultat al unei interogări sau operații asupra unei sau mai multor tabele existente. Aceste tabele derivate nu sunt stocate fizic în baza de date, ci sunt generate dinamic în timpul execuției interogării și sunt folosite pentru a obține rezultate intermediare sau transformate într-o anumită formă, tabelă.

--am crat o interogare in care am folosit o tabela derivata SumaTotala pentru a calcula suma medie totala

WITH SumaTotala AS (

SELECT SumaTotala

FROM Bon

)

SELECT AVG(SumaTotala) AS SumaMedie

FROM SumaTotala;



Figura 10 – Tabele derivate

**10.5 Expresii-tabel generalizate**

Expresia-tabel generalizat (Common Table Expression - CTE) este o funcționalitate specifică a limbajului SQL, care permite definirea și utilizarea unei tabele temporare în cadrul unei interogări SQL. CTE-urile sunt utile pentru a organiza și a structura interogările complexe, înlocuind adesea subinterogările.

**Interogări recursive**

Este evaluate repetat până când condiția de oprire specificată este îndeplinită. În fiecare iterație, se utilizează rezultatul iterației anterioare pentru a produce un nou rezultat.

--Aceasta interogare utilizeaza o CTE recursiva pentru a genera o secventa de numere intregi de la 1

--la 10, adaugand 1 la fiecare iteratie pana cand se atinge limita de 10.

WITH NumereIntregi AS (

SELECT 1 AS Numar

UNION ALL

SELECT Numar + 1

FROM NumereIntregi

WHERE Numar <= 10

)

SELECT \* FROM NumereIntregi;

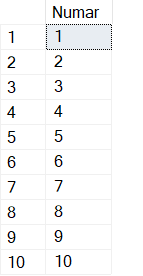


Figura 12 – Intergarea recursive

**Interogări nerecursive**

Este evaluată o singură dată în cadrul interogării. Rezultatul este returnat și utilizat în mod direct în interogarea principală.

--Aceasta interogare calculeaza pretul mediu al fiecarui produs in fiecare magazin folosind o CTE nerecursiva

--si apoi se alatura tabelului "Produse" pentru a adauga pretul mediu calculat pentru fiecare inregistrare.

WITH PretMediuProduse AS (

SELECT MagazinID, AVG(PretProdus) AS PretMediu

FROM SistemGestionalBon.dbo.Produse

GROUP BY MagazinID

)

SELECT p.ProdusID, p.NumeProdus, p.MagazinID, p.PretProdus, p.FurnizorID, pm.PretMediu

FROM SistemGestionalBon.dbo.Produse p

JOIN PretMediuProduse pm ON p.MagazinID = pm.MagazinID;

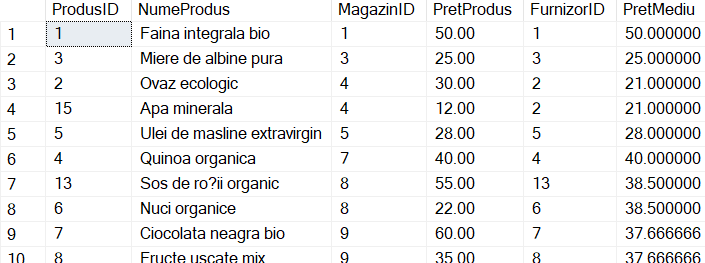


Figura 11 – Interogări nerecursive

**Expresii-tabel generalizare cu funcții analitice**

--expresie-tabel generalizata cu functie analitica

--CTE-ul "SumaCumulativa" calculeaza suma cumulativa a preturilor produselor in ordinea datelor de vanzare

--folosind functia analitica SUM() cu clauza OVER. Apoi, interogam acea CTE pentru a selecta toate coloanele

--din CTE, inclusiv suma cumulativa calculata.

WITH SumaCumulativa AS (

SELECT

ProdusID,

NumeProdus,

PretProdus,

SUM(PretProdus) OVER (ORDER BY ProdusID) AS SumaCumulativa

FROM

SistemGestionalBon.dbo.Produse

)

SELECT

ProdusID,

NumeProdus,

PretProdus,

SumaCumulativa

FROM

SumaCumulativa;

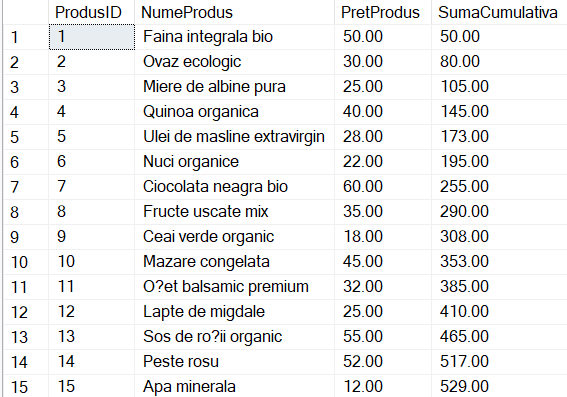


Figura 15 – Expresie-tabel generalizată cu funcție analitică

**Elemente ale CTE recursive**

Elementele unei CTE recursive sunt cele care definesc si controleaza comportamentul recursiv al expresiei-tabel generalizate. Intr-o CTE recursiva, exista doua componente principale:

* Partea de ancorare (non-recursive term): Aceasta este prima parte a expresiei-tabel generalizate si reprezinta baza sau conditia initiala pentru recursivitate. Partea de ancorare este evaluata o singura data la inceputul executiei si este utilizata pentru a initializa recursivitatea.
* Partea recursiva (recursive term): Aceasta este a doua parte a expresiei-tabel generalizate si defineste modul in care expresia se repeta sau se "recurseaza". Partea recursiva utilizeaza rezultatul partii de ancorare sau al partii recursive anterioare pentru a calcula un nou set de date care va fi utilizat in continuare in cadrul CTE.

--In acest exemplu, CTE-ul recursiv, "RecursivCTE", incepe cu un produs specificat prin ProdusID si continua

--apoi sa selecteze produsul urmator in secventa, adaugand unu la ProdusID. Acest proces se repeta pana cand

--se atinge o anumita conditie de oprire (in acest caz, limitam numarul de produse afisate la 100 pentru a

--evita buclele infinite).

--Elemente ale CTE recursive

WITH RecursivCTE AS (

-- Membru de ancorare

SELECT BonID, CumparatorID, AngajatID, SumaTotala

FROM [SistemGestionalBon].[dbo].[Bon]

WHERE BonID = 1

UNION ALL

-- Membru recursiv

SELECT p. BonID, p. CumparatorID, p. AngajatID, p. SumaTotala

FROM [SistemGestionalBon].[dbo].[Bon] p

INNER JOIN RecursivCTE c ON p.BonID = c.BonID + 1 -- Selectam produsul urmator in secventa

WHERE p.BonID <= 100 -- Limitam recursivitatea pentru a evita bucle infinite

)

SELECT \* FROM RecursivCTE;

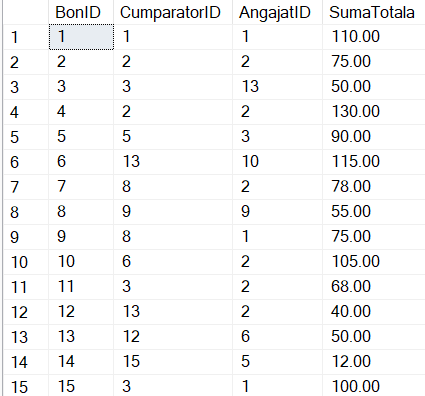


Figura 16 – Elemente ale CTE recursive

**Concluzie**

În această lucrare, am explorat SQL-ul analitic, funcțiile de clasificare și expresiile tabel, oferind exemple și explicații pentru funcții precum ROLLUP, CUBE, și NTILE. Am demonstrat utilizarea CTE-urilor nerecursive și recursive pentru a organiza și structura interogările complexe, și am prezentat diferite tipuri de interogări încrucișate, cum ar fi cele cu operatorii PIVOT și UNPIVOT. Am evidențiat, de asemenea, importanța tabelelor derivate și expresiilor-tabel generalizate, oferind exemple și explicații pentru fiecare. În final, am analizat elementele esențiale ale CTE-urilor recursive și am oferit un exemplu practic pentru a ilustra utilizarea lor într-o interogare SQL.