

Capitolul 12. SQL, ierarhii și recursivitate

Problemele grafurilor, ierarhiilor și recursivității în SQL reprezintă un subiect generos despre care au scris, printre alții, Joe Celko¹, Stephane Faroult², Anthony Molinaro³, Fabian Pascal⁴ și Vadim Tropashko⁵. Faroult și Robson inventariază trei modele de reprezentare a ierarhiilor:

- Modelul adiacent – similar modului de reprezentare a șefilor direcți în tabela PERSONAL2 (figura 8.4);
- Modelul „căii materializate”, în care simbolul unui nod indică poziția sa în ierarhie, cum ar fi cazul planului de conturi din contabilitatea unei companii (ex. 401 – Furnizori, 401.001 – Furnizorul A, 401.002 – Furnizorul B etc.)⁶;
- Modelul setului imbricat⁷, în care fiecărui nod îi sunt asociate o pereche de numere care definesc intervalul în care sunt plasați subordonații săi.

În acest capitol ne vom ocupa fugitiv doar de primul model și câteva alte probleme circumscrise grafurilor și ierarhiilor.

12.1. Autojoncțiuni și subconsultări

Pentru aflarea majorității informațiilor privitoare la ierarhia firmei, soluția obișnuită în SQL constă în joncționarea a două instanțe ale tabelului PERSONAL2 (P1 și P2) după condiția $P1.MarcaSef = P2.Marca$.

Prin interogarea:

```
SELECT *
```

```
FROM personal2 p1 INNER JOIN personal2 p2 ON p1.MarcaSef=p2.Marca
```

se obține un rezultat de forma celui din figura 12.1. Primele șase coloane corespund primei instanțe, P1, în timp ce restul celei de-a doua instanțe, P2. P1 este legată de calitatea de subordonat, iar P2 de cea de șef. De aceea, ar fi mai nimerită redenumirea lui P1 ca *SUBORDONAȚI*, iar a lui P2 drept *ȘEFI*.

¹ [Celko2004], [Celko2005]

² [Faroult&Robson2006]

³ [Molinaro2006]

⁴ [Pascal2000]

⁵ [Tropashko 2005]

⁶ Pentru detalii, vezi și [Fotache2005], p.196

⁷ Vezi [Celko2004], [Celko2005]

marca	numepren	datanast	compart	marcasef	saltafar	marca	numepren	datanast	compart	marcasef	saltafar
2	ANGAJAT 2	1977-10-11 00:00:00	FINANCIAR	1	1450.00	1	ANGAJAT 1	1962-07-01 00:00:00	DIRECTIUNE	NULL	1600.00
3	ANGAJAT 3	1962-08-02 00:00:00	MARKETING	1	1450.00	1	ANGAJAT 1	1962-07-01 00:00:00	DIRECTIUNE	NULL	1600.00
4	ANGAJAT 4	NULL	FINANCIAR	2	1380.00	2	ANGAJAT 2	1977-10-11 00:00:00	FINANCIAR	1	1450.00
5	ANGAJAT 5	1965-04-30 00:00:00	FINANCIAR	2	1420.00	2	ANGAJAT 2	1977-10-11 00:00:00	FINANCIAR	1	1450.00
6	ANGAJAT 6	1965-11-09 00:00:00	FINANCIAR	5	1350.00	5	ANGAJAT 5	1965-04-30 00:00:00	FINANCIAR	2	1420.00
7	ANGAJAT 7	NULL	FINANCIAR	5	1280.00	5	ANGAJAT 5	1965-04-30 00:00:00	FINANCIAR	2	1420.00
8	ANGAJAT 8	1960-12-31 00:00:00	MARKETING	3	1290.00	3	ANGAJAT 3	1962-08-02 00:00:00	MARKETING	1	1450.00
9	ANGAJAT 9	1976-02-28 00:00:00	MARKETING	3	1410.00	3	ANGAJAT 3	1962-08-02 00:00:00	MARKETING	1	1450.00
10	ANGAJAT 10	1972-01-29 00:00:00	RESURSE UMANE	1	1370.00	1	ANGAJAT 1	1962-07-01 00:00:00	DIRECTIUNE	NULL	1600.00

Figura 12.1. Autojoncțiunea tabelii PERSONAL2

Să luăm în discuție câteva probleme.

Cum se numește șeful Angajatului 7 ?

Acum chiar denumim cele două instanțe ale tabelii PERSONAL2 SUBORDONATI, respectiv SEFI, pentru a limpezi atât specificarea condiției de joncțiune, cât și formularea predicatului de selecție și a coloanei de afișat (rezultatul este în figura 12.2).

```
SELECT sefi.NumePren
FROM personal2 subordonati
INNER JOIN personal2 sefi ON subordonati.MarcaSef=sefi.Marca
WHERE subordonati.NumePren = 'ANGAJAT 7'
```

NumePren
ANGAJAT 5

Figura 12.2. Șeful direct al Angajatului 7

Care sunt subordonații direcți ai Angajatului 2 ?

Păstrăm, din interogarea anterioară, titulaturile celor două instanțe, modificând predicatul și numele coloanei extrase în rezultat (rezultatul este în figura 12.3):

```
SELECT subordonati.NumePren
FROM personal2 subordonati
INNER JOIN personal2 sefi ON subordonati.MarcaSef=sefi.Marca
WHERE sefi.NumePren = 'ANGAJAT 2'
```

NumePren
ANGAJAT 4
ANGAJAT 5

Figura 12.3. Subordonații direcți ai Angajatului 2

Firește, cele două probleme pot fi rezolvate și prin subconsultări, astfel:

```
SELECT NumePren
FROM personal2
WHERE Marca IN
(SELECT MarcaSef
```

```
FROM personal2
WHERE NumePren = 'ANGAJAT 7')
```

respectiv:

```
SELECT NumePren
FROM personal2
WHERE MarcaSef IN
      (SELECT Marca
       FROM personal2
       WHERE NumePren = 'ANGAJAT 2')
```

Câți subordonați are fiecare angajat al firmei ?

Soluția:

```
SELECT sefi.NumePren, COUNT(*) AS Nr_Subordonati
FROM personal2 subordonati
INNER JOIN personal2 sefi ON subordonati.MarcaSef=sefi.Marca
GROUP BY sefi.NumePren
```

extrage numai pe cei care au măcar un subordonat. Dacă dorim ca rezultatul să conțină toți angajații în ordine alfabetică (figura 12.4), trebuie folosită joncțiunea externă la dreapta și modificat argumentul funcției COUNT:

```
SELECT sefi.NumePren,
COUNT(SUBORDONATI.MarcaSef) AS Nr_Subordonati
FROM personal2 subordonati RIGHT OUTER JOIN personal2 sefi
ON subordonati.MarcaSef=sefi.Marca
GROUP BY sefi.NumePren
ORDER BY 1
```

Figura 12.4 conține urmările acestei interogări.

NumePren	Nr_Subordonati
ANGAJAT 1	3
ANGAJAT 10	0
ANGAJAT 2	2
ANGAJAT 3	2
ANGAJAT 4	0
ANGAJAT 5	2
ANGAJAT 6	0
ANGAJAT 7	0
ANGAJAT 8	0
ANGAJAT 9	0

Figura 12.4. Numărul subordonaților pentru fiecare salariat

Care sunt subordonații subordonaților directorului general ?

Din punctul de vedere al arborelui ce oglindește ierarhia firmei (figura 12.5), ne interesează “nepoții rădăcinii”. Interogarea trebuie să parcurgă trei nivele ierarhice, și, în final, să extragă salariații cu mărcile 4, 5, 8 și 9.

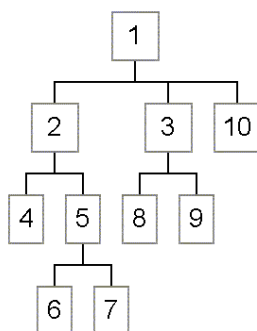


Figura 12.5. Ierarhia firmei

Varianta 1 (joncțiuni):

```

SELECT subordonati.Marca, subordonati.NumePren, subordonati.Compart
FROM personal2 subordonati
INNER JOIN personal2 sefi1 ON subordonati.MarcaSef=sefi1.Marca
INNER JOIN personal2 sefi2 ON sefi1.MarcaSef=sefi2.Marca
WHERE sefi2.MarcaSef IS NULL

```

Varianta 2 (subconsultări):

```

SELECT Marca, NumePren, Compart
FROM personal2
WHERE MarcaSef IN
  (SELECT Marca
   FROM personal2
   WHERE MarcaSef IN
    (SELECT Marca
     FROM personal2
     WHERE MarcaSef IS NULL)
  )

```

Rezultatul (figura 12.6) confirmă bănuielele noastre.

Marca	NumePren	Compart
4	ANGAJAT 4	FINANCIAR
5	ANGAJAT 5	FINANCIAR
8	ANGAJAT 8	MARKETING
9	ANGAJAT 9	MARKETING

Figura 12.6. Subordonații direcți ai subordonaților “imediați” directorului general

Care este nivelul ierarhic al fiecărui salariat ?

De la început, știm că ierarhia reflectată în tabela PERSONAL2 se derulează pe patru niveluri, așa încât o interogare pentru rezolvarea problemei poate fi:

```

SELECT NumePren, Compart, 'Nivel 1' AS Nivel
FROM personal2
WHERE MarcaSef IS NULL
UNION
SELECT NIVEL2.NumePren, NIVEL2.Compart, 'Nivel 2' AS Nivel
FROM personal2 NIVEL1 INNER JOIN personal2 NIVEL2
ON NIVEL1.Marca = NIVEL2.MarcaSef AND NIVEL1.MarcaSef IS NULL
UNION
SELECT NIVEL3.NumePren, NIVEL3.Compart, 'Nivel 3' AS Nivel
FROM personal2 NIVEL1 INNER JOIN personal2 NIVEL2
ON NIVEL1.Marca = NIVEL2.MarcaSef
AND NIVEL1.MarcaSef IS NULL
INNER JOIN personal2 NIVEL3 ON NIVEL2.Marca = NIVEL3.MarcaSef
UNION
SELECT NIVEL4.NumePren, NIVEL4.Compart, 'Nivel 4' AS Nivel
FROM personal2 NIVEL1
INNER JOIN personal2 NIVEL2 ON NIVEL1.Marca = NIVEL2.MarcaSef
AND NIVEL1.MarcaSef IS NULL
INNER JOIN personal2 NIVEL3 ON NIVEL2.Marca = NIVEL3.MarcaSef
INNER JOIN personal2 NIVEL4 ON NIVEL3.Marca = NIVEL4.MarcaSef
ORDER BY NumePren

```

Iată și rezultatul în figura 12.7. Interogarea funcționează rezonabil. Există, însă, cel puțin două umbre: trebuie să știm, aprioric, numărul nivelelor ierarhice, iar în al doilea rând, la un număr mult mai mare de nivele, întinderea consultării crește sensibil.

NumePren	Compart	Nivel
ANGAJAT 1	DIRECTIUNE	Nivel 1
ANGAJAT 10	RESURSE UMANE	Nivel 2
ANGAJAT 2	FINANCIAR	Nivel 2
ANGAJAT 3	MARKETING	Nivel 2
ANGAJAT 4	FINANCIAR	Nivel 3
ANGAJAT 5	FINANCIAR	Nivel 3
ANGAJAT 6	FINANCIAR	Nivel 4
ANGAJAT 7	FINANCIAR	Nivel 4
ANGAJAT 8	MARKETING	Nivel 3
ANGAJAT 9	MARKETING	Nivel 3

Figura 12.7. Nivelul ierarhic al fiecărui angajat

O variantă mai elegantă, ca efort de scriere este:

```

SELECT NIVEL4.NumePren, NIVEL4.Compart,

```

```

CASE
WHEN NIVEL4.MarcaSef IS NULL THEN 1
    WHEN NIVEL3.MarcaSef IS NULL THEN 2
    WHEN NIVEL2.MarcaSef IS NULL THEN 3
    WHEN NIVEL1.MarcaSef IS NULL THEN 4
END AS Nivel
FROM personal2 NIVEL1
    RIGHT OUTER JOIN personal2 NIVEL2
        ON NIVEL1.Marca = NIVEL2.MarcaSef AND NIVEL1.MarcaSef IS
        NULL
    RIGHT OUTER JOIN personal2 NIVEL3
        ON NIVEL2.Marca = NIVEL3.MarcaSef
    RIGHT OUTER JOIN personal2 NIVEL4
        ON NIVEL3.Marca = NIVEL4.MarcaSef

```

Se jonctionează extern patru instanțe ale tabelului PERSONAL2, câte una pentru fiecare nivel ierarhic, astfel încât au și fost denumite NIVEL1... NIVEL4. Fraza SELECT compune ierarhia pornind de la NIVEL1 care reprezintă o instanță a PERSONAL2 cu o singură linie, cea a directorului general, instanță jonctiionată extern la dreapta cu NIVEL2 care va furniza subordonații direcți ai înregistrării rădăcină s.a.m.d.

Să se afișeze structura ierarhică a firmei.

Practic, dorim o formă de vizualizare a angajaților care să țină cont de modul lor de subordonare – ca în figura 12.8.

R 2	NUME	R 2	COMPART	R 2	SEF1	R 2	SEF2	R 2	SEF3	R 2	SEF4
	ANGAJAT 1		DIRECTIUNE		1		1		1		1
	--ANGAJAT 2		FINANCIAR		1		2		2		2
	---ANGAJAT 4		FINANCIAR		1		2		4		4
	---ANGAJAT 5		FINANCIAR		1		2		5		5
	---ANGAJAT 6		FINANCIAR		1		2		5		6
	---ANGAJAT 7		FINANCIAR		1		2		5		7
	--ANGAJAT 3		MARKETING		1		3		3		3
	---ANGAJAT 8		MARKETING		1		3		8		8
	---ANGAJAT 9		MARKETING		1		3		9		9
	--ANGAJAT 10		RESURSE UMANE		1		10		10		10

Figura 12.8. Vizualizarea ierarhiei

Dacă până acum, interogările erau executabile în toate serverele luate în discuție în această carte, următoarea funcționează în Oracle și DB2, și cu ușoare modificări (operatorul de concatenare || se înlocuiește cu +) și în MS SQL Server.

WITH niveluri AS

(SELECT NIVEL4.*,

```

CASE
    WHEN NIVEL4.MarcaSef IS NULL THEN 1
    WHEN NIVEL3.MarcaSef IS NULL THEN 2
    WHEN NIVEL2.MarcaSef IS NULL THEN 3
    WHEN NIVEL1.MarcaSef IS NULL THEN 4
END AS Nivel
FROM personal2 NIVEL1
    RIGHT OUTER JOIN personal2 NIVEL2
        ON NIVEL1.Marca = NIVEL2.MarcaSef
        AND Nivel1.MarcaSef IS NULL
    RIGHT OUTER JOIN personal2 NIVEL3
        ON NIVEL2.Marca = NIVEL3.MarcaSef
    RIGHT OUTER JOIN personal2 NIVEL4
        ON NIVEL3.Marca = NIVEL4.MarcaSef )
SELECT NumePren AS Nume, Compart, Marca AS Sef1,
    Marca AS Sef2, Marca AS Sef3, Marca AS Sef4
FROM niveluri
WHERE Nivel = 1
UNION
SELECT '-' | N2.NumePren AS Nume, N2.Compart, N1.Marca AS Sef1,
    N2.Marca AS Sef2, N2.Marca AS Sef3, N2.Marca AS Sef4
FROM niveluri N2 INNER JOIN niveluri N1
    ON N2.Nivel=2 AND N2.MarcaSef=N1.Marca AND N1.Nivel=1
UNION
SELECT '- -' | N3.NumePren AS Nume, N3.Compart, N1.Marca AS Sef1,
    N2.Marca AS Sef2, N3.Marca AS Sef3, N3.Marca AS Sef4
FROM niveluri N3 INNER JOIN niveluri N2
    ON N3.Nivel=3 AND N3.MarcaSef=N2.Marca AND N2.Nivel=2
    INNER JOIN niveluri N1
        ON N2.Nivel=2 AND N2.MarcaSef=N1.Marca AND N1.Nivel=1
UNION
SELECT '- - -' | N4.NumePren AS Nume, N4.Compart, N1.Marca AS Sef1,
    N2.Marca AS Sef2, N3.Marca AS Sef3, N4.Marca AS Sef4
FROM niveluri N4
    INNER JOIN niveluri N3
        ON N4.Nivel=4 AND N4.MarcaSef=N3.Marca AND N3.Nivel=3
    INNER JOIN niveluri N2
        ON N3.Nivel=3 AND N3.MarcaSef=N2.Marca AND N2.Nivel=2
    INNER JOIN niveluri N1
        ON N2.Nivel=2 AND N2.MarcaSef=N1.Marca AND N1.Nivel=1

```

ORDER BY Sef1, Sef2, Sef3, Sef4

Lucrurile pot fi simplificate sensibil utilizând o structură de tip CASE prin care determinăm nivelul de subordonare, nivel care va determina de câte ori se repetă (REPEAT în DB2, LPAD în Oracle și PostgreSQL, REPLICATE în MS SQL Server) grupul de linii, astfel încât listarea primei coloane este asemănătoare celei din figura 12.8. Iată varianta SQL Server:

```
SELECT REPLICATE('-', 1 * ( (
    CASE
        WHEN NIVEL4.MarcaSef IS NULL THEN 1
        WHEN NIVEL3.MarcaSef IS NULL THEN 2
        WHEN NIVEL2.MarcaSef IS NULL THEN 3
        WHEN NIVEL1.MarcaSef IS NULL THEN 4
    END ) - 1 ) ) + NIVEL4.NumePren AS Nume, NIVEL4.Compart
FROM personal2 NIVEL1
    RIGHT OUTER JOIN personal2 NIVEL2
        ON NIVEL1.Marca = NIVEL2.MarcaSef AND
            NIVEL1.MarcaSef IS NULL
    RIGHT OUTER JOIN personal2 NIVEL3
        ON NIVEL2.Marca = NIVEL3.MarcaSef
    RIGHT OUTER JOIN personal2 NIVEL4
        ON NIVEL3.Marca = NIVEL4.MarcaSef
```

O problemă grozav de interesantă pentru ceea ce înseamnă ierarhie și recursivitate în SQL este cea tratată în paragraful 8.3 al cărții dedicate proiectării (și implementării) bazelor de date⁸. Pornind de la o tabelă DISTANȚE {Loc1, Loc2, Distanța} care conține numărul de kilometri dintre două localități oarecum vecine între care există șosea „practicabilă” (vorba vine!), putem să determinăm traseele posibile dintre două orașe. Lucrurile nu sunt atât de simple, însă, fiind necesare joncțiuni repetate corespunzătoare „punctelor intermediare”. Pentru exemplificare (sintaxă Oracle), intimidanta interogare de mai jos extrage toate traseele posibile dintre Iași și Focșani, prin maximum cinci localități intermediare:

```
WITH distante_tot AS
    (SELECT Loc1, Loc2, Distanța FROM distante
     UNION
     SELECT Loc2, Loc1, Distanța FROM distante)
-- prima este (eventuala) rută directă
SELECT d1.Loc1 AS Plecare, d1.Loc2 AS Sosire,
```

⁸ [Fotache2005]


```

        d1.Loc1 || '***' || d1.Loc2 AS Sir,
        Distanta
FROM distante_tot d1
WHERE Loc1='Iasi' AND Loc2='Focsani'
UNION
-- rute printr-o localitate intermediara
SELECT d1.Loc1, d2.Loc2,
        d1.Loc1 || '***' || d1.Loc2 || '***' || d2.Loc2,
        d1.Distanta + d2.Distanta
FROM distante_tot d1 INNER JOIN distante_tot d2
        ON d1.Loc1='Iasi' AND d1.Loc2=d2.Loc1 AND d2.Loc2='Focsani'
UNION
-- rute prin doua localitati intermediare
SELECT d1.Loc1, d3.Loc2,
        d1.Loc1 || '***' || d1.Loc2 || '***' || d2.Loc2 || '***' || d3.Loc2,
        d1.Distanta + d2.Distanta+d3.Distanta
FROM distante_tot d1
        INNER JOIN distante_tot d2 ON d1.Loc1='Iasi' AND d1.Loc2=d2.Loc1
        AND d2.Loc2 <> d1.Loc1 AND d2.Loc2<>d1.Loc2
        INNER JOIN distante_tot d3 ON d2.Loc2=d3.Loc1 AND d3.Loc2='Focsani'
        AND d3.Loc2 <> d2.Loc1 AND d3.Loc2<>d2.Loc2
        AND d3.Loc2 <> d1.Loc1 AND d3.Loc2<>d1.Loc2
UNION
-- rute prin trei localitati intermediare
SELECT d1.loc1, d4.Loc2,
        d1.loc1 || '***' || d1.Loc2 || '***' || d2.Loc2 || '***' || d3.Loc2 || '***' || d4.Loc2,
        d1.distanta + d2.distanta+d3.distanta+d4.distanta
FROM distante_tot d1
        INNER JOIN distante_tot d2 ON d1.Loc1='Iasi' AND d1.Loc2=d2.Loc1
        AND d2.Loc2 <> d1.loc1 AND d2.Loc2<>d1.Loc2
        INNER JOIN distante_tot d3 ON d2.Loc2=d3.Loc1
        AND d3.Loc2 <> d2.loc1 AND d3.Loc2<>d2.loc2
        AND d3.Loc2 <> d1.Loc1 AND d3.Loc2<>d1.Loc2
        INNER JOIN distante_tot d4 ON d3.Loc2=d4.Loc1 AND d4.Loc2='Focsani'
        AND d4.Loc2 <> d3.loc1 AND d4.Loc2<>d3.loc2
        AND d4.Loc2 <> d2.Loc1 AND d4.Loc2<>d2.Loc2
        AND d4.Loc2 <> d1.Loc1 AND d4.Loc2<>d1.Loc2
UNION
-- rute prin patru localitati intermediare
SELECT d1.loc1, d5.Loc2,

```

```

        d1.loc1 || '***' || d1.Loc2 || '***' || d2.Loc2 || '***' || d3.Loc2 || '***' || d4.Loc2
        || '***' || d5.Loc2,
        d1.distanta + d2.distanta+d3.distanta+d4.distanta+d5.distanta
FROM distante_tot d1
    INNER JOIN distante_tot d2 ON d1.Loc1='Iasi' AND d1.Loc2=d2.Loc1
        AND d2.Loc2 <> d1.loc1 AND d2.Loc2<>d1.Loc2
    INNER JOIN distante_tot d3 ON d2.Loc2=d3.Loc1
        AND d3.Loc2 <> d2.loc1 AND d3.Loc2<>d2.loc2
        AND d3.Loc2 <> d1.Loc1 AND d3.Loc2<>d1.Loc2
    INNER JOIN distante_tot d4 ON d3.Loc2=d4.Loc1
        AND d4.Loc2 <> d3.loc1 AND d4.Loc2<>d3.loc2
        AND d4.Loc2 <> d2.Loc1 AND d4.Loc2<>d2.Loc2
        AND d4.Loc2 <> d1.Loc1 AND d4.Loc2<>d1.Loc2
    INNER JOIN distante_tot d5 ON d4.Loc2=d5.Loc1 AND d5.Loc2='Focsani'
        AND d5.Loc2 <> d4.loc1 AND d5.Loc2<>d4.loc2
        AND d5.Loc2 <> d3.loc1 AND d5.Loc2<>d3.loc2
        AND d5.Loc2 <> d2.Loc1 AND d5.Loc2<>d2.Loc2
        AND d5.Loc2 <> d1.Loc1 AND d5.Loc2<>d1.Loc2

UNION

-- rute prin cinci localitati intermediare
SELECT d1.loc1, d6.Loc2,
        d1.loc1 || '***' || d1.Loc2 || '***' || d2.Loc2 || '***' || d3.Loc2 || '***' || d4.Loc2
        || '***' || d5.Loc2 || '***' || d6.Loc2,
        d1.distanta + d2.distanta+d3.distanta+d4.distanta+d5.distanta+d6.distanta
FROM distante_tot d1
    INNER JOIN distante_tot d2 ON d1.Loc1='Iasi'
        AND d1.Loc2=d2.Loc1 AND d2.Loc2 <> d1.loc1 AND d2.Loc2<>d1.Loc2
    INNER JOIN distante_tot d3 ON d2.Loc2=d3.Loc1 AND d3.Loc2 <> d2.loc1 AND
        d3.Loc2<>d2.loc2 AND d3.Loc2 <> d1.Loc1 AND d3.Loc2<>d1.Loc2
    INNER JOIN distante_tot d4 ON d3.Loc2=d4.Loc1 AND d4.Loc2 <> d3.loc1
        AND d4.Loc2<>d3.loc2 AND d4.Loc2 <> d2.Loc1 AND d4.Loc2<>d2.Loc2
        AND d4.Loc2 <> d1.Loc1 AND d4.Loc2<>d1.Loc2
    INNER JOIN distante_tot d5 ON d4.Loc2=d5.Loc1 AND d5.Loc2 <> d4.loc1 AND
        d5.Loc2<>d4.loc2 AND d5.Loc2 <> d3.loc1 AND d5.Loc2<>d3.loc2
        AND d5.Loc2 <> d2.Loc1 AND d5.Loc2<>d2.Loc2
        AND d5.Loc2 <> d1.Loc1 AND d5.Loc2<>d1.Loc2
    INNER JOIN distante_tot d6 ON d5.Loc2=d6.Loc1 AND d6.Loc2='Focsani'
        AND d6.Loc2 <> d5.loc1 AND d6.Loc2<>d5.loc2
        AND d6.Loc2 <> d4.loc1 AND d6.Loc2<>d4.loc2
        AND d6.Loc2 <> d3.loc1 AND d6.Loc2<>d3.loc2

```

```
AND d6.Loc2 <> d2.Loc1 AND d6.Loc2<>d2.Loc2
AND d6.Loc2 <> d1.Loc1 AND d6.Loc2<>d1.Loc2
```

ORDER BY 4

Ceea ce obținem seamănă cu rezultatul din figura 12.9. Deși exagerat de lungă, soluția poate fi aplicată pentru orice pereche de localități (de pornire, respectiv, sosire). Unde mai punem la socoteală că în PostgreSQL, expresiile tabelă nu funcționează.

R 2	PLECARE	R 2	SOSIRE	R 2	SIR	R 2	DISTANTA
	Iasi		Focsani		Iasi**Vaslui**Birlad**Tecuci**Tisita**Focsani		206
	Iasi		Focsani		Iasi**Tg.Frumos**Roman**Bacau**Adjud**Tisita**Focsani		239
	Iasi		Focsani		Iasi**Vaslui**Bacau**Adjud**Tisita**Focsani		261
	Iasi		Focsani		Iasi**Vaslui**Roman**Bacau**Adjud**Tisita**Focsani		297
	Iasi		Focsani		Iasi**Vaslui**Birlad**Tecuci**Galati**Focsani		320
	Iasi		Focsani		Iasi**Vaslui**Birlad**Tecuci**Galati**Braila**Focsani		337

Figura 12.9. Traseele (prin maximum cinci localități intermediare) dintre Iași și Focșani

Vom vedea, însă, că există și variante mai elegante de rezolvare a acestei probleme.

12.2. Interogări arborescente în Oracle

Pentru problemele formulate în acest paragraf, până în acest punct au fost formulate numai soluții (mai mult sau mai puțin) generale. Dialectele SQL ale celor patru servere au câteva opțiuni speciale pentru parcurgerea elegantă a structurilor ierarhice și/sau recursive. Începem cu Oracle.

Care este nivelul ierarhic al fiecărui salariat ?

În Oracle există două clauze prin care se construiește o structură ierarhică/iterativă, START WITH și CONNECT BY. Soluția următoare conduce la rezultatul din figura 12.10.

```
SELECT PERSONAL2.*, LEVEL
FROM personal2
START WITH MarcaSef IS NULL
CONNECT BY PRIOR Marca=MarcaSef
```

Construirea structurii ierarhice începe cu înregistrarea (înregistrările) care îndeplinesc condiția din clauza START WITH. Această înregistrare-părinte (care este „rădăcina” ierarhiei) va fi legată de înregistrarea sau înregistrările copil prin condiția din CONNECT BY PRIOR Marca = MarcaSef. La rândul lor, înregistrările copil pot avea înregistrări „copil” (care sunt nepoții „rădăcinii”) care sunt extrase prin același predicat din CONNECT BY.

Așadar, prin CONNECT BY sunt selectate toate generațiile succesive de linii-copil (copii, nepoți, strănepoți etc.). Clauza PRIOR plasată în stânga condiției

semnifică: valoarea atributului Marca din părinte trebuie să fie egală cu valoarea MarcaSef din înregistrările copil, nepot, strănepot, ... stră (de n ori) nepot.

După construirea ierarhiei, se elimină tuplurile ce nu îndeplinesc condiția formulată în clauza WHERE. Este important de notat că selecția se aplică linie cu linie, iar eliminarea unei linii-părinte nu atrage automat eliminarea copiilor, nepoților s.a.m.d. Dacă există clauza ORDER BY, aceasta va determina dispunerea înregistrărilor în rezultat. În lipsa clauzei de ordonare, înregistrările sunt dispuse în funcție de ordinea parcurgerii arborelui, așa cum se observă în figura 12.10.

R	MARCA	R	NUMEPREN	R	DATANAST	R	COMPART	R	MARCASEF	R	SALTARIFAR	R	LEVEL
	1 ANGAJAT 1				01-07-1962		DIRECTIUNE		(null)		1600		1
	2 ANGAJAT 2				11-10-1977		FINANCIAR		1		1450		2
	4 ANGAJAT 4				(null)		FINANCIAR		2		1380		3
	5 ANGAJAT 5				30-04-1965		FINANCIAR		2		1420		3
	6 ANGAJAT 6				09-11-1965		FINANCIAR		5		1350		4
	7 ANGAJAT 7				(null)		FINANCIAR		5		1280		4
	3 ANGAJAT 3				02-08-1962		MARKETING		1		1450		2
	8 ANGAJAT 8				31-12-1960		MARKETING		3		1290		3
	9 ANGAJAT 9				28-02-1976		MARKETING		3		1410		3
	10 ANGAJAT 10				29-01-1972		RESURSE UMANE		1		1370		2

Figura 12.10. Interogare ierarhică în Oracle

Un avantaj major a interogărilor ierarhice ține de folosirea pseudo-coloanei LEVEL ce semnifică tocmai nivelul ierarhiei, relativ la înregistrarea/înregistrările "rădăcină" care îndeplinește/îndeplinesc condiția din START WITH. Ca principale restricții Oracle trebuie amintit că SELECT-ul care execută o interogare ierarhică nu poate efectua o joncțiune și nici extrage date dintr-o tabelă virtuală creată printr-o joncțiune.

Dacă în clauza CONNECT BY PRIOR inversăm ordinea atributelor, rezultatul va conține o singură linie (vezi figura 12.11) întrucât linia de pornire este a Angajatului 1 pentru care nu există nici o linie „părinte” (angajat superior ierarhic) pentru care marca să fie egală cu marca Angajatului 1.

```

SELECT personal2.*, LEVEL
FROM personal2
START WITH MarcaSef IS NULL
CONNECT BY PRIOR MarcaSef = Marca

```

Results
 Script Output
 Explain
 Autotrace
 DBMS Output
 OWA Output

Results:

	MARCA	NUMEPREN	DATANAST	COMPART	MARCASEF	SALTARIFAR	LEVEL
1	1 ANGAJAT 1	01-07-1962	DIRECTIUNE	(null)	1600		1

Figura 12.11. Inversarea ordinii atributelor în predicatul clauzei CONNECT BY PRIOR

Explicațiile de mai sus sunt corecte doar pe jumătate, deoarece o ierarhie poate fi construită și de sus în jos (TOP-DOWN) și de jos în sus (BOTTOM-UP). În prima

interogare, am specificat înregistrarea „rădăcină” și apoi condiția prin care sunt extrași copii, nepoții, strănepoții etc.. Dacă am dori să construim structura de sus în jos, începând de la Angajat 2, interogarea ar fi:

```
SELECT personal2.*, LEVEL
FROM personal2
START WITH NumePren = 'ANGAJAT 2'
CONNECT BY PRIOR Marca = MarcaSef
```

iar rezultatul cel din figura 12.12. Titlaturile *părinte-copii-nepoți-strănepoți...* sunt, deci, corecte.

1	MARCA	2	NUMEPREN	3	DATANAST	4	COMPART	5	MARCASEF	6	SALTARIFAR	7	LEVEL
	2	ANGAJAT 2		11-10-1977		FINANCIAR		1		1450		1	
	4	ANGAJAT 4		(null)		FINANCIAR		2		1380		2	
	5	ANGAJAT 5		30-04-1965		FINANCIAR		2		1420		2	
	6	ANGAJAT 6		09-11-1965		FINANCIAR		5		1350		3	
	7	ANGAJAT 7		(null)		FINANCIAR		5		1280		3	

Figura 12.12. Construirea ierarhiei de sus în jos

Ierarhiile pot fi însă construite și de jos în sus (BOTTOM-UP), specificând „cea mai de jos” înregistrare și găsind apoi, rând pe rând, superiorii ierarhici de ordin 1, 2 etc. În aceste situații, titlaturile *părinte-copil-nepot...* nu mai sunt operaționale, pentru că se „urcă pe linie genealogică”. Interogarea următoare, ce obține lista din figura 12.13, extrage toți superiorii Angajatului 7:

```
SELECT personal2.*, LEVEL
FROM personal2
START WITH NumePren = 'ANGAJAT 7'
CONNECT BY PRIOR MarcaSef = Marca
```

1	MARCA	2	NUMEPREN	3	DATANAST	4	COMPART	5	MARCASEF	6	SALTARIFAR	7	LEVEL
	7	ANGAJAT 7		(null)		FINANCIAR		5			1280		1
	5	ANGAJAT 5		30-04-1965		FINANCIAR		2			1420		2
	2	ANGAJAT 2		11-10-1977		FINANCIAR		1			1450		3
	1	ANGAJAT 1		01-07-1962		DIRECTIUNE		(null)			1600		4

Figura 12.13. Construirea ierarhiei de jos în sus

Așa că este mai nimerit să folosim termenii *subordonați de ordinul (nivelul) 1, 2, ..., n*, respectiv *superiori de ordinul 1, 2, ..., n*. Acum, că tot am lămurit modalitățile de construire a ierarhiilor, putem rezolva destul de simplu problemele formulate în paragraful anterior.

Cum se numește șeful Angajatului 7 ?

Una dintre variantele generalizabile ține de includerea ultimei interogări BOTTOM-UP într-o expresie-tabelă și extragerea liniei pentru care nivelul (LEVEL) angajatului este superior cu 1:

```
WITH sefi_angajat7 AS
    (SELECT personal2.*, LEVEL AS Nivel
     FROM personal2
     START WITH NumePren = 'ANGAJAT 7'
     CONNECT BY PRIOR MarcaSef = Marca )
SELECT *
FROM sefi_angajat7
WHERE Nivel = (SELECT Nivel + 1
               FROM sefi_angajat7
               WHERE NumePren='ANGAJAT 7')
```

Ierarhia se construiește începând cu Angajat 7, care va avea implicit nivelul (LEVEL) 1. Este sigur, deci, că șeful direct al acestuia va avea nivelul 2, așa că putem simplifica interogarea în oarecare măsură:

```
WITH sefi_angajat7 AS
    (SELECT personal2.*, LEVEL AS Nivel
     FROM personal2
     START WITH NumePren = 'ANGAJAT 7'
     CONNECT BY PRIOR MarcaSef = Marca
     )
SELECT * FROM sefi_angajat7 WHERE Nivel = 2
```

Care sunt subordonații direcți ai Angajatului 2 ?

Am răspuns deja la această întrebare când am constuit ierarhia de sus în jos prin interogarea al cărei rezultat era cel din figura 12.13.

Care sunt subordonații subordonaților directorului general ?

Ierarhia se construiește de sus în jos, începând cu directorul general. Condiția poate fi exprimată astfel:

```
WITH ierarhie AS (
    SELECT PERSONAL2.*, LEVEL AS Nivel
    FROM personal2
    START WITH MarcaSef IS NULL
    CONNECT BY PRIOR Marca=MarcaSef)
SELECT * FROM ierarhie
WHERE Nivel = (SELECT Nivel + 2 FROM ierarhie WHERE MarcaSef IS NULL)
```

sau, dacă ținem seama că nivelul ierarhic al directorului general este 1, al subordonaților săi direcți este 2, iar al subordonaților subordonaților este 3:

```
SELECT personal2.*, LEVEL
FROM personal2
```

```
WHERE LEVEL = 3
START WITH MarcaSef IS NULL CONNECT BY MarcaSef = PRIOR Marca
```

Câte niveluri ierarhice are firma ?

Trebuie să aflăm LEVEL-ul maxim, ceea ce nu e chiar complicat:

```
SELECT MAX(Nivel)
FROM (SELECT personal2.*, LEVEL AS Nivel
      FROM personal2
      START WITH MarcaSef IS NULL
      CONNECT BY MarcaSef = PRIOR Marca )
```

sau, și mai simplu:

```
SELECT MAX(LEVEL)
FROM personal2
START WITH MarcaSef IS NULL CONNECT BY MarcaSef = PRIOR Marca
```

Să se afișeze structura ierarhică a firmei.

Nu putem să reprezentăm arborele ierarhic cu “verdele în jos”, ci cu rădăcina la stânga, la fel ca în soluția cu rezultatul în figura 12.8. Indentarea subordonaților se obține cu ajutorul funcției LPAD și pseudo-atributului LEVEL după cum urmează:

```
SELECT LPAD(' ', 3 * (LEVEL - 1), '– ') || NumePren AS Nume, Compart
FROM personal2
START WITH MarcaSef IS NULL
CONNECT BY PRIOR Marca = MarcaSef
```

Pentru afișarea structurii ierarhice, inclusiv a modului de subordonare a fiecărui angajat se poate folosi în Oracle o funcție specială – SYS_CONNECT_BY_PATH:

```
SELECT SYS_CONNECT_BY_PATH(NumePren,'-> ') AS "Cale de subordonare",
Compart
FROM personal2
START WITH MarcaSef IS NULL
CONNECT BY PRIOR Marca = MarcaSef
```

Funcția returnează întreaga „cale” de subordonare a fiecărui angajat – vezi figura 12.14. Între nivelurile ierarhice se poate insera un caracter sau un șir de caractere (în exemplul nostru este ->).

Cale de subordonare	COMPART
-> ANGAJAT 1	DIRECTIUNE
-> ANGAJAT 1 -> ANGAJAT 2	FINANCIAR
-> ANGAJAT 1 -> ANGAJAT 2 -> ANGAJAT 4	FINANCIAR
-> ANGAJAT 1 -> ANGAJAT 2 -> ANGAJAT 5	FINANCIAR
-> ANGAJAT 1 -> ANGAJAT 2 -> ANGAJAT 5 -> ANGAJAT 6	FINANCIAR
-> ANGAJAT 1 -> ANGAJAT 2 -> ANGAJAT 5 -> ANGAJAT 7	FINANCIAR
-> ANGAJAT 1 -> ANGAJAT 3	MARKETING
-> ANGAJAT 1 -> ANGAJAT 3 -> ANGAJAT 8	MARKETING
-> ANGAJAT 1 -> ANGAJAT 3 -> ANGAJAT 9	MARKETING
-> ANGAJAT 1 -> ANGAJAT 10	RESURSE UMANE

Figura 12.14. Funcția SYS_CONNECT_BY_PATH

În lipsa unei condiții formulate în clauza START WITH, se construiește câte o ierarhie pentru fiecare angajat, (fiecare angajat va fi, pe rând, rădăcină a unei ierarhii). Astfel, interogarea următoare va genera rezultatul din figura 12.15.

NUME	COMPART
--> ANGAJAT 2	FINANCIAR
--> ANGAJAT 2 --> ANGAJAT 4	FINANCIAR
--> ANGAJAT 2 --> ANGAJAT 5	FINANCIAR
--> ANGAJAT 2 --> ANGAJAT 5 --> ANGAJAT 6	FINANCIAR
--> ANGAJAT 2 --> ANGAJAT 5 --> ANGAJAT 7	FINANCIAR
--> ANGAJAT 3	MARKETING
--> ANGAJAT 3 --> ANGAJAT 8	MARKETING
--> ANGAJAT 3 --> ANGAJAT 9	MARKETING
--> ANGAJAT 10	RESURSE UMANE
--> ANGAJAT 4	FINANCIAR
--> ANGAJAT 5	FINANCIAR
--> ANGAJAT 5 --> ANGAJAT 6	FINANCIAR
--> ANGAJAT 5 --> ANGAJAT 7	FINANCIAR
--> ANGAJAT 8	MARKETING
--> ANGAJAT 9	MARKETING
--> ANGAJAT 6	FINANCIAR
--> ANGAJAT 7	FINANCIAR
--> ANGAJAT 1	DIRECTIUNE
--> ANGAJAT 1 --> ANGAJAT 2	FINANCIAR
--> ANGAJAT 1 --> ANGAJAT 2 --> ANGAJAT 4	FINANCIAR
--> ANGAJAT 1 --> ANGAJAT 2 --> ANGAJAT 5	FINANCIAR
--> ANGAJAT 1 --> ANGAJAT 2 --> ANGAJAT 5 --> ANGAJAT 6	FINANCIAR
--> ANGAJAT 1 --> ANGAJAT 2 --> ANGAJAT 5 --> ANGAJAT 7	FINANCIAR
--> ANGAJAT 1 --> ANGAJAT 3	MARKETING
--> ANGAJAT 1 --> ANGAJAT 3 --> ANGAJAT 8	MARKETING
--> ANGAJAT 1 --> ANGAJAT 3 --> ANGAJAT 9	MARKETING
--> ANGAJAT 1 --> ANGAJAT 10	RESURSE UMANE

Figura 12.15. O ierarhie pentru fiecare angajat


```

SELECT SYS_CONNECT_BY_PATH(numepren, ' -> ') AS Nume, Compart
FROM personal2
CONNECT BY PRIOR Marca = MarcaSef

```

A sosit momentul să formulăm o soluție mai elegantă pentru afișarea traseelor dintre două localități (Iași și Focșani, în exemplul nostru), pe baza celor aflate mai sus. Începem cu o interogare „ajutătoare”:

```

SELECT SYS_CONNECT_BY_PATH (Loc1, '**') || '**' || Loc2 AS Traseu,
       LEVEL AS Nivel, SYS_CONNECT_BY_PATH (Distanța, '+') AS Dist,
       d.*, ROWNUM AS Ord
FROM distante d
START WITH Loc1='Iasi'
CONNECT BY PRIOR Loc2 = Loc1 AND Loc1<>'Focsani' AND Level < 20
ORDER BY Ord

```

Localitățile prin care trece ruta sunt separate prin două asteriscuri, iar distanțele dintre cele două localități prin semnul plus – vezi figura 12.16. Prin clauza LEVEL < 20, sunt extrase numai eventualele trasee care trec prin maxim 20 de localități (21, cu localitatea de plecare), valorile atributului Nivel indicând destul de bine numărul de localități prin care trece traseul (cu excepția plecării).

TRASEU	NIVEL	DIST	LOC1	LOC2	DISTANTA	ORD
IasiTg.Frumos	1	+53	Iasi	Tg.Frumos	53	1
IasiTg.Frumos**Roman	2	+53+40	Tg.Frumos	Roman	40	2
IasiTg.Frumos**Roman**Bacau	3	+53+40+41	Roman	Bacau	41	3
IasiTg.Frumos**Roman**Bacau**Adjud	4	+53+40+41+60	Bacau	Adjud	60	4
IasiTg.Frumos**Roman**Bacau**Adjud**Tisita	5	+53+40+41+60+32	Adjud	Tisita	32	5
IasiTg.Frumos**Roman**Bacau**Adjud**Tisita**Focsani	6	+53+40+41+60+32+13	Tisita	Focsani	13	6
IasiTg.Frumos**Roman**Bacau**Vaslui	4	+53+40+41+85	Bacau	Vaslui	85	7
IasiTg.Frumos**Roman**Bacau**Vaslui**Birlad	5	+53+40+41+85+54	Vaslui	Birlad	54	8
IasiTg.Frumos**Roman**Bacau**Vaslui**Birlad**Tecuci	6	+53+40+41+85+54+48	Birlad	Tecuci	48	9
IasiTg.Frumos**Roman**Bacau**Vaslui**Birlad**Tecuci**Galati	7	+53+40+41+85+54+48+67	Tecuci	Galati	67	10
IasiTg.Frumos**Roman**Bacau**Vaslui**Birlad**Tecuci**Tisita	7	+53+40+41+85+54+48+20	Tecuci	Tisita	20	11
IasiTg.Frumos**Roman**Bacau**Vaslui**Birlad**Tecuci**Tisita**Focsani	8	+53+40+41+85+54+48+20+13	Tisita	Focsani	13	12
IasiTg.Frumos**Roman**Vaslui	3	+53+40+80	Roman	Vaslui	80	13
IasiTg.Frumos**Roman**Vaslui**Birlad	4	+53+40+80+54	Vaslui	Birlad	54	14

Figura 12.16. Trasee, trasee... (fragment)

Din păcate, nu avem o funcție „directă” pentru evaluarea adunării kilometrilor separați prin + (valorile atributului Dist). Am putea crea o funcție-utilizator, dar vom fi în stare de așa ceva abia în capitolul 16. Așa că, pentru a calcula lungimea fiecărui traseu și sub-traseu, vom avea nevoie de o subconsultare scalară (în clauza SELECT). Pentru a urmări mai ușor logica interogării vom recurge și la o expresie-tabelă, iar (sub)traseele se vor „opri” la Focșani:

WITH tab AS

```

(SELECT SYS_CONNECT_BY_PATH (Loc1, '**') || '**' || Loc2 AS Traseu,
       LEVEL AS Nivel,
       SYS_CONNECT_BY_PATH (Distanța, '+') AS Dist,
       d.*,

```

```

        ROWNUM AS Ord
    FROM distante d
    START WITH Loc1='Iasi'
    CONNECT BY PRIOR Loc2 = Loc1 AND Loc1<>'Focsani' AND Level < 10
    ORDER BY Ord)
SELECT t1.*, (SELECT SUM(Distanta) FROM tab t2 WHERE t1.Traseu
        LIKE Traseu || '%' ) AS Km
FROM tab t1

```

Rezultatul este cel din figura 12.17. De remarcat opțiunea LIKE din clauza WHERE a subconsultării ce calculează numărul de kilometri ai traseului.

TRASEU	NIVEL	DIST	LOC1	LOC2	DISTANTA	ORD	KM
IasiVaslui	1	+71	Iasi	Vaslui	71	1	71
IasiVaslui**Birlad	2	+71+54	Vaslui	Birlad	54	2	125
IasiVaslui**Birlad**Tecuci	3	+71+54++48	Birlad	Tecuci	48	3	173
IasiVaslui**Birlad**Tecuci**Tisita	4	+71+54++48+20	Tecuci	Tisita	20	4	193
IasiVaslui**Birlad**Tecuci**Tisita**Focsani	5	+71+54++48+20+13	Tisita	Focsani	13	5	206
IasiVaslui**Birlad**Tecuci**Galati	4	+71+54++48+67	Tecuci	Galati	67	6	240
IasiTg.Frumos	1	+53	Iasi	Tg.Frumos	53	7	53
IasiTg.Frumos**Roman	2	+53+40	Tg.Frumos	Roman	40	8	93
IasiTg.Frumos**Roman**Bacau	3	+53+40+41	Roman	Bacau	41	9	134
IasiTg.Frumos**Roman**Bacau**Adjud	4	+53+40+41+60	Bacau	Adjud	60	10	194
IasiTg.Frumos**Roman**Bacau**Adjud**Tisita	5	+53+40+41+60+32	Adjud	Tisita	32	11	226

Figura 12.17. Trasee și subtrasee, cu plecare din Iași, și destinație "limită" Focșani (fragment)

Dacă dorim doar afișarea „rutelor întregi” Iași-Focșani, modificăm ușor interogarea de mai sus (vezi figura 12.18):

```

WITH tab AS (
    SELECT SYS_CONNECT_BY_PATH (Loc1, '**') || '**' || Loc2 AS Traseu,
           LEVEL AS Nivel, d.*, ROWNUM AS Ord
    FROM distante d
    START WITH Loc1='Iasi'
    CONNECT BY PRIOR Loc2 = Loc1 AND Loc1<>'Focsani' AND Level < 50
    ORDER BY Ord)
SELECT Ord,Traseu, Dist, (SELECT SUM(Distanta) FROM tab t2
        WHERE t1.Traseu LIKE Traseu || '%' ) AS Km
FROM tab t1
WHERE Traseu LIKE '**Iasi%Focsani'
ORDER BY Km

```

ORD	TRASEU	DIST	KM
5	**Iasi**Vaslui**Birlad**Tecuci**Tisita**Focsani	+71+54+48+20+13	206
12	**Iasi**Tg.Frumos**Roman**Bacau**Adjud**Tisita**Focsani	+53+40+41+60+32+13	239
23	**Iasi**Tg.Frumos**Roman**Vaslui**Birlad**Tecuci**Tisita**Focsani	+53+40+80+54+48+20+13	308
17	**Iasi**Tg.Frumos**Roman**Bacau**Vaslui**Birlad**Tecuci**Tisita**Focsani	+53+40+41+85+54+48+20+13	354

Figura 12.18. Trasee întregi Iași-Focșani

În continuare, ne punem problema generării stațiilor pentru o rută, “lung prilej de bârfe și de ipoteze”. Iată o soluție care impresionează prin dimensiune:

```

SELECT rute.*,
      SUBSTR(traseu, INSTR(traseu, '**', 1, 1) + 2, INSTR(traseu, '**', 1, 2) - 3) AS L1,
      CASE
        WHEN INSTR(traseu, '**', 1, 2) = 0 THEN NULL
        ELSE SUBSTR(traseu, INSTR(traseu, '**', 1, 2) + 2,
          CASE
            WHEN INSTR(traseu, '**', 1, 3) > 0 THEN INSTR(traseu, '**', 1, 3) -
              INSTR(traseu, '**', 1, 2) - 2
            ELSE LENGTH(traseu)
          END )
      END AS L2,
      CASE
        WHEN INSTR(traseu, '**', 1, 3) = 0 THEN NULL
        ELSE SUBSTR(traseu, INSTR(traseu, '**', 1, 3) + 2,
          CASE
            WHEN INSTR(traseu, '**', 1, 4) > 0 THEN INSTR(traseu, '**', 1, 4) -
              INSTR(traseu, '**', 1, 3) - 2
            ELSE LENGTH(traseu)
          END )
      END AS L3,
      CASE
        WHEN INSTR(traseu, '**', 1, 4) = 0 THEN NULL
        ELSE SUBSTR(traseu, INSTR(traseu, '**', 1, 4) + 2,
          CASE
            WHEN INSTR(traseu, '**', 1, 5) > 0 THEN INSTR(traseu, '**', 1, 5) -
              INSTR(traseu, '**', 1, 4) - 2
            ELSE LENGTH(traseu)
          END )
      END AS L4,
      CASE
        WHEN INSTR(traseu, '**', 1, 5) = 0 THEN NULL
        ELSE SUBSTR(traseu, INSTR(traseu, '**', 1, 5) + 2,

```

```

CASE
  WHEN INSTR(traseu, '**',1,6) > 0 THEN INSTR(traseu,'**',1,6) -
    INSTR(traseu,'**',1,5) -2
  ELSE LENGTH(traseu)
END )
END AS L5,
CASE
  WHEN INSTR(traseu,'**',1,6) = 0 THEN NULL
  ELSE SUBSTR(traseu, INSTR(traseu,'**',1,6)+2,
    CASE
      WHEN INSTR(traseu, '**',1,7) > 0 THEN INSTR(traseu,'**',1,7) -
        INSTR(traseu,'**',1,6) -2
      ELSE LENGTH(traseu)
    END )
  END AS L6,
CASE
  WHEN INSTR(traseu,'**',1,7) = 0 THEN NULL
  ELSE SUBSTR(traseu, INSTR(traseu,'**',1,7)+2,
    CASE
      WHEN INSTR(traseu, '**',1,8) > 0 THEN INSTR(traseu,'**',1,8) -
        INSTR(traseu,'**',1,7) -2
      ELSE LENGTH(traseu)
    END )
  END AS L7
FROM (
  SELECT Ord, Traseu, ( SELECT SUM(Distanta)
    FROM (
      SELECT SYS_CONNECT_BY_PATH (Loc1,
        '**') || '**' || Loc2 AS Traseu,
        LEVEL AS Nivel,
        d.*,
        ROWNUM AS Ord
      FROM distante d
      START WITH Loc1='Iasi'
      CONNECT BY PRIOR Loc2 = Loc1 AND
        Loc1<>'Focsani' AND Level < 50
      ORDER BY Ord
    ) t2
    WHERE tab.Traseu LIKE Traseu || '%' ) AS Km
  FROM (

```

```

SELECT SYS_CONNECT_BY_PATH (Loc1, '**') || '**'
      || Loc2 AS Traseu, LEVEL AS Nivel,
      SYS_CONNECT_BY_PATH (Distanța, '+') AS Dist,
      d.*,
      ROWNUM AS Ord
FROM distante d
START WITH Loc1='Iasi'
CONNECT BY PRIOR Loc2 = Loc1 AND Loc1<>'Focsani' AND
      Level < 50
ORDER BY Ord
) tab
WHERE Traseu LIKE '**Iasi%Focsani'
ORDER BY Km
) rute

```

Chiar dacă rezultatul din figura 12.19 este încurajator, efortul de scriere este insuportabil, mai ales dacă ne gândim că e posibil să repetăm operațiunea pentru trasele cu 20 de localități sau chiar mai multe.

ORD	TRASEU	KM	L1	L2	L3	L4	L5	L6	L7
24	**Iasi**Vaslui**Birlad**Tecuci**Tisita**Focsani	206	Iasi	Vaslui	Birlad	Tecuci	Tisita	Focsani	(null)
6	**Iasi**Tg.Frumos**Roman**Bacau**Adjud**Tisita**Focsani	239	Iasi	Tg.Frumos	Roman	Bacau	Adjud	Tisita	Focsani
18	**Iasi**Tg.Frumos**Roman**Vaslui**Birlad**Tecuci**Tisita**Focsani	308	Iasi	Tg.Frumos	Roman	Vaslui	Birlad	Tecuci	Tisita
12	**Iasi**Tg.Frumos**Roman**Bacau**Vaslui**Birlad**Tecuci**Tisita**Focsani	354	Iasi	Tg.Frumos	Roman	Bacau	Vaslui	Birlad	Tecuci

Figura 12.19. Localitățile (stațiile) de pe traseele Iași-Focșani – varianta incompletă

Așa că avem motive suficiente să ne străduim în a găsi o soluție mai eficientă. Ce ziceți de următoarea prin care extragem, pe câte o linie, câte o stație (localitate) de pe traseul Iași-Focșani via Vaslui ?

WITH tab AS

```

(SELECT SYS_CONNECT_BY_PATH (Loc1, '**') || '**' || Loc2 AS Traseu,
      LEVEL AS Nivel, d.*, ROWNUM AS Ord
FROM distante d
START WITH Loc1='Iasi'
CONNECT BY PRIOR Loc2 = Loc1 AND Loc1<>'Focsani' AND Level < 20
ORDER BY Ord
)

```

```

SELECT DISTINCT t2.traseu, t2.Nivel AS LocNr, t2.Loc2

```

```

FROM tab t1 INNER JOIN tab t2 ON t1.Traseu LIKE '**Iasi**Vaslui%Focsani' AND
      t1.traseu LIKE t2.Traseu || '%'

```

```

ORDER BY 1,2

```

Să recunoaștem că soluția este de-a dreptul simpatică, prin comparație cu precedenta, iar rezultatul este corect – sau cel puțin așa pare în figura 12.20.

TRASEU	LOCNR	LOC2
IasiVaslui	1	Vaslui
IasiVaslui**Birlad	2	Birlad
IasiVaslui**Birlad**Tecuci	3	Tecuci
IasiVaslui**Birlad**Tecuci**Tisita	4	Tisita
IasiVaslui**Birlad**Tecuci**Tisita**Focsani	5	Focsani

Figura 12.20. Localitățile (stațiile) de pe traseul Iași-Vaslui-Focșani

12.3. Interogări arborescente în MS SQL Server

Sistemul de construire a unei ierarhii în MS SQL Server este sensibil diferit de cel din Oracle:

```

WITH ierarhie ( Marca, NumePren, Compart, MarcaSef, Nivel) AS
    (SELECT Marca, NumePren, Compart, MarcaSef, 0 AS Nivel
     FROM personal2
     WHERE MarcaSef IS NULL
     UNION ALL
     SELECT p.Marca, p.NumePren, p.Compart, p.MarcaSef, Nivel + 1
     FROM personal2 p INNER JOIN ierarhie ON p.MarcaSef=ierarhie.Marca
    )
SELECT *
FROM ierarhie

```

Mai întâi, sintaxa presupune obligatoriu folosirea unei expresii-tabelă, denumită, în cazul nostru, IERARHIE. Expresia tabelă este definită prin două interogări conectate prin operatorul de reuniune – UNION ALL. Prima este interogarea-ancoră și definește înregistrarea „rădăcină” – vezi figura 12.21. A doua folosește ancora care este jonționată cu tabela PERSONAL2. Expresia de jonțiune exprimă, de fapt, recursivitatea prin care se construiește ierarhia.

```

WITH ierarhie ( Marca, NumePren, Compart, MarcaSef, Nivel)
AS
  (SELECT Marca, NumePren, Compart, MarcaSef, 0 AS Nivel
   FROM personal2
   WHERE MarcaSef IS NULL
  UNION ALL
   SELECT p.Marca, p.NumePren, p.Compart, p.MarcaSef, Nivel + 1
   FROM personal2 p INNER JOIN ierarhie
    ON p.MarcaSef=ierarhie.Marca
  )
SELECT *
FROM ierarhie

```

← Ancoră (pornire)

← Definirea expresiei de recursivitate

← Interogare ce folosește expresia-tabelă

	Marca	NumePren	Compart	MarcaSef	Nivel
1	1	ANGAJAT 1	DIRECTIUNE	NULL	0
2	2	ANGAJAT 2	FINANCIAR	1	1
3	3	ANGAJAT 3	MARKETING	1	1
4	10	ANGAJAT 10	RESURSE UMANE	1	1
5	8	ANGAJAT 8	MARKETING	3	2
6	9	ANGAJAT 9	MARKETING	3	2
7	4	ANGAJAT 4	FINANCIAR	2	2
8	5	ANGAJAT 5	FINANCIAR	2	2
9	6	ANGAJAT 6	FINANCIAR	5	3
10	7	ANGAJAT 7	FINANCIAR	5	3

Figura 12.21. Construirea ierarhiei în MS SQL Server

Cum se numește șeful Angajatului 7 ?

Ierarhia se va construi de jos în sus, începând cu Angajat 7, care va avea implicit nivelul (LEVEL) 0; de aceea, vom modifica predicatul din expresia de recursivitate. Șeful direct al Angajatului 7 va avea nivelul 1 (vezi figura 12.22):

```

WITH ierarhie7 ( Marca, NumePren, Compart, MarcaSef, Nivel) AS
  (SELECT Marca, NumePren, Compart, MarcaSef, 0 AS Nivel
   FROM personal2
   WHERE NumePren='ANGAJAT 7'
  UNION ALL
   SELECT p.Marca, p.NumePren, p.Compart, p.MarcaSef, Nivel + 1
   FROM personal2 p INNER JOIN ierarhie7 ON p.Marca=ierarhie7.MarcaSef)
SELECT *
FROM ierarhie7
WHERE Nivel=1

```

Marca	NumePren	Compart	MarcaSef	Nivel
5	ANGAJAT 5	FINANCIAR	2	1

Figura 12.22. Șeful direct al Angajatului 7 (MS SQL Server)

Care sunt subordonații direcți ai Angajatului 2 ?

Modificăm ușor SELECT-ul principal al interogării din deschiderea acestui paragraf:

```

WITH ierarhie ( Marca, NumePren, Compart, MarcaSef, Nivel) AS
    (SELECT Marca, NumePren, Compart, MarcaSef, 0 AS Nivel
     FROM personal2
     WHERE NumePren='ANGAJAT 2'
    UNION ALL
     SELECT p.Marca, p.NumePren, p.Compart, p.MarcaSef, Nivel + 1
     FROM personal2 p INNER JOIN ierarhie
     ON p.MarcaSef=ierarhie.Marca)
SELECT *
FROM ierarhie
WHERE Nivel=1

```

Care sunt subordonații subordonaților directorului general ?

Ierarhia de construiește de sus în jos, începând cu directorul general, iar angajații care ne interesează se plasează pe nivelul 2:

```

WITH ierarhie ( Marca, NumePren, Compart, MarcaSef, Nivel) AS
    (SELECT Marca, NumePren, Compart, MarcaSef, 0 AS Nivel
     FROM personal2
     WHERE MarcaSef IS NULL
    UNION ALL
     SELECT p.Marca, p.NumePren, p.Compart, p.MarcaSef, Nivel + 1
     FROM personal2 p INNER JOIN ierarhie ON p.MarcaSef=ierarhie.Marca)
SELECT *
FROM ierarhie
WHERE Nivel=2

```

Câte niveluri ierarhice are firma ?

Ținând seama de faptul că nivelul pornește de la zero, trebuie să incrementăm cu 1 rezultatul obținut prin funcția MAX:

```

WITH ierarhie ( Marca, NumePren, Compart, MarcaSef, Nivel) AS
    (SELECT Marca, NumePren, Compart, MarcaSef, 0 AS Nivel
     FROM personal2
     WHERE MarcaSef IS NULL
    UNION ALL
     SELECT p.Marca, p.NumePren, p.Compart, p.MarcaSef, Nivel + 1
     FROM personal2 p
     INNER JOIN ierarhie ON p.MarcaSef=ierarhie.Marca)
SELECT MAX(Nivel) + 1
FROM ierarhie

```

Să se afișeze structura ierarhică a firmei.

MS SQL Server nu prezintă o funcție similară SYS_CONNECT_BY_PATH. Cu toate acestea, putem să furnizăm un rezultat de forma celui din figura 12.14, inclusiv „calea de subordonare” ierarhică a fiecărui angajat. Ideea ar fi ca, în interogarea destinată precizării expresiei de recursivitate, numele și prenumele să fie obținut prin concatenarea *ierarhie.NumePren* + ' -> ' + *p.NumePren*. Din păcate, SQL Serverul spune că tipul lui *NumePren* din „ancoră” diferă de tipul obținut prin concatenarea din propoziția anterioară, chiar dacă toate valorile sunt șiruri de caractere – vezi figura 12.23.

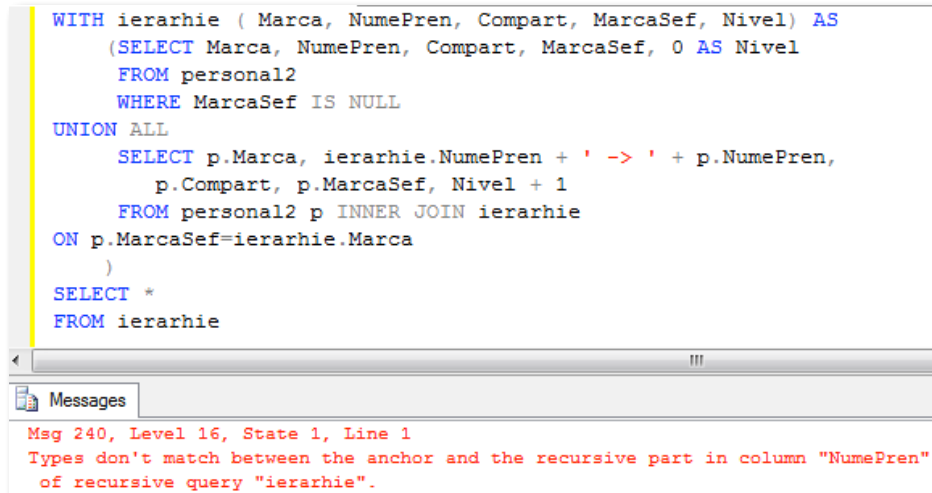


Figura 12.23. Un naz de-al SQL Serverului

Rezolvarea vine de la funcția CAST:

```
WITH ierarhie ( Marca, NumePren, Compart, MarcaSef, Nivel) AS
  (SELECT Marca, CAST(NumePren AS VARCHAR(500)),
    Compart, MarcaSef, 0 AS Nivel
  FROM personal2
  WHERE MarcaSef IS NULL
  UNION ALL
  SELECT p.Marca, CAST (ierarhie.NumePren + ' -> '
    + p.NumePren AS VARCHAR(500)), p.Compart, p.MarcaSef, Nivel + 1
  FROM personal2 p INNER JOIN ierarhie ON p.MarcaSef=ierarhie.Marca
  )
SELECT *
FROM ierarhie
ORDER BY 2
```

De data aceasta, SQL Server-ul este chiar cooperant, afișând rezultatul din figura 12.24.

Marca	NumePren	Compart	MarcaSef	Nivel
1	ANGAJAT 1	DIRECTIUNE	NULL	0
10	ANGAJAT 1 -> ANGAJAT 10	RESURSE UMANE	1	1
2	ANGAJAT 1 -> ANGAJAT 2	FINANCIAR	1	1
4	ANGAJAT 1 -> ANGAJAT 2 -> ANGAJAT 4	FINANCIAR	2	2
5	ANGAJAT 1 -> ANGAJAT 2 -> ANGAJAT 5	FINANCIAR	2	2
6	ANGAJAT 1 -> ANGAJAT 2 -> ANGAJAT 5 -> ANGAJAT 6	FINANCIAR	5	3
7	ANGAJAT 1 -> ANGAJAT 2 -> ANGAJAT 5 -> ANGAJAT 7	FINANCIAR	5	3
3	ANGAJAT 1 -> ANGAJAT 3	MARKETING	1	1
8	ANGAJAT 1 -> ANGAJAT 3 -> ANGAJAT 8	MARKETING	3	2
9	ANGAJAT 1 -> ANGAJAT 3 -> ANGAJAT 9	MARKETING	3	2

Figura 12.24. Calea ierarhică completă a fiecărui angajat (în SQL Server)

Ca și în Oracle, în lipsa unei condiții formulate în interogarea-ancoră, se construiește câte o ierarhie pentru fiecare angajat, (fiecare angajat va fi, pe rând, rădăcină a unei ierarhii), astfel că interogarea:

Marca	NumePren	Compart	MarcaSef	Nivel
1	ANGAJAT 1	DIRECTIUNE	NULL	0
10	ANGAJAT 1 -> ANGAJAT 10	RESURSE UMANE	1	1
2	ANGAJAT 1 -> ANGAJAT 2	FINANCIAR	1	1
4	ANGAJAT 1 -> ANGAJAT 2 -> ANGAJAT 4	FINANCIAR	2	2
5	ANGAJAT 1 -> ANGAJAT 2 -> ANGAJAT 5	FINANCIAR	2	2
6	ANGAJAT 1 -> ANGAJAT 2 -> ANGAJAT 5 -> ANGAJAT 6	FINANCIAR	5	3
7	ANGAJAT 1 -> ANGAJAT 2 -> ANGAJAT 5 -> ANGAJAT 7	FINANCIAR	5	3
3	ANGAJAT 1 -> ANGAJAT 3	MARKETING	1	1
8	ANGAJAT 1 -> ANGAJAT 3 -> ANGAJAT 8	MARKETING	3	2
9	ANGAJAT 1 -> ANGAJAT 3 -> ANGAJAT 9	MARKETING	3	2
10	ANGAJAT 10	RESURSE UMANE	1	0
2	ANGAJAT 2	FINANCIAR	1	0
4	ANGAJAT 2 -> ANGAJAT 4	FINANCIAR	2	1
5	ANGAJAT 2 -> ANGAJAT 5	FINANCIAR	2	1
6	ANGAJAT 2 -> ANGAJAT 5 -> ANGAJAT 6	FINANCIAR	5	2
7	ANGAJAT 2 -> ANGAJAT 5 -> ANGAJAT 7	FINANCIAR	5	2
3	ANGAJAT 3	MARKETING	1	0
8	ANGAJAT 3 -> ANGAJAT 8	MARKETING	3	1
9	ANGAJAT 3 -> ANGAJAT 9	MARKETING	3	1
4	ANGAJAT 4	FINANCIAR	2	0
5	ANGAJAT 5	FINANCIAR	2	0
6	ANGAJAT 5 -> ANGAJAT 6	FINANCIAR	5	1
7	ANGAJAT 5 -> ANGAJAT 7	FINANCIAR	5	1
6	ANGAJAT 6	FINANCIAR	5	0
7	ANGAJAT 7	FINANCIAR	5	0
8	ANGAJAT 8	MARKETING	3	0
9	ANGAJAT 9	MARKETING	3	0

Figura 12.25. Calea ierarhică completă a fiecărui angajat (în SQL Server)

WITH ierarhie (Marca, NumePren, Compart, MarcaSef, Nivel) AS
(SELECT Marca, CAST(NumePren AS VARCHAR(500)), Compart,

```

        MarcaSef, 0 AS Nivel
    FROM personal2
    UNION ALL
    SELECT p.Marca, CAST (ierarhie.NumePren + ' -> '
        + p.NumePren AS VARCHAR(500)), p.Compart, p.MarcaSef, Nivel + 1
    FROM personal2 p INNER JOIN ierarhie ON p.MarcaSef=ierarhie.Marca
)
SELECT * FROM ierarhie ORDER BY 2

```

va genera rezultatul din figura 12.25.

Atacăm problema traseelor posibile dintre două localități. Construirea grafului nu ridică probleme deosebite. Astfel, interogarea:

```

WITH ierarhie (Cale, Nivel, Distanta, Loc2) AS
    (SELECT CAST(Loc1 + ' -> ' + Loc2 AS VARCHAR(500)), 0 AS Nivel,
        CAST (Distanta AS NUMERIC), Loc2
    FROM distante WHERE Loc1='Iasi'
    UNION ALL
    SELECT CAST (i.Cale + ' -> ' + d.Loc2 AS VARCHAR(500)),
        Nivel + 1, CAST (i.Distanta+d.Distanta AS NUMERIC), d.Loc2
    FROM ierarhie i INNER JOIN distante d ON i.Loc2=d.Loc1
    WHERE Nivel < 20
)
SELECT * FROM ierarhie

```

obține un rezultat de forma celui din figura 12.26.

Cale	Nivel	Distanta	Loc2
Iasi -> Tg.Frumos	0	53	Tg.Frumos
Iasi -> Vaslui	0	71	Vaslui
Iasi -> Vaslui -> Birlad	1	125	Birlad
Iasi -> Vaslui -> Birlad -> Tecuci	2	173	Tecuci
Iasi -> Vaslui -> Birlad -> Tecuci -> Galati	3	240	Galati
Iasi -> Vaslui -> Birlad -> Tecuci -> Tisita	3	193	Tisita
Iasi -> Vaslui -> Birlad -> Tecuci -> Tisita -> Focsani	4	206	Focsani
Iasi -> Vaslui -> Birlad -> Tecuci -> Tisita -> Focsani -> Braila	5	292	Braila
Iasi -> Vaslui -> Birlad -> Tecuci -> Tisita -> Focsani -> Galati	5	286	Galati
Iasi -> Vaslui -> Birlad -> Tecuci -> Tisita -> Focsani -> Rm.Sarat	5	248	Rm.Sarat
Iasi -> Vaslui -> Birlad -> Tecuci -> Tisita -> Focsani -> Rm.Sarat -> Braila	6	332	Braila
Iasi -> Vaslui -> Birlad -> Tecuci -> Tisita -> Focsani -> Rm.Sarat -> Braila -> Galati	7	343	Galati
Iasi -> Vaslui -> Birlad -> Tecuci -> Tisita -> Focsani -> Braila -> Galati	6	303	Galati
Iasi -> Tg.Frumos -> Roman	1	93	Roman
Iasi -> Tg.Frumos -> Roman -> Bacau	2	134	Bacau
Iasi -> Tg.Frumos -> Roman -> Vaslui	2	173	Vaslui
Iasi -> Tg.Frumos -> Roman -> Vaslui -> Birlad	3	227	Birlad
Iasi -> Tg.Frumos -> Roman -> Vaslui -> Birlad -> Tecuci	4	275	Tecuci

Figura 12.26. Trasee care încep din Iași (în SQL Server) - fragment

Răspunsul „punctual” la problema afișării rutelor posibile dintre Iași și Focșani, în ordinea distanțelor (figura 12.27) se află acum destul de ușor:

```
WITH ierarhie (Cale, Nivel, Distanța, Loc2) AS
    (SELECT CAST(Loc1 + ' -> ' + Loc2 AS VARCHAR(500)), 0 AS Nivel,
        CAST (Distanța AS NUMERIC), Loc2
    FROM distante
    WHERE Loc1='Iasi'
    UNION ALL
    SELECT CAST (i.Cale + ' -> ' + d.Loc2 AS VARCHAR(500)),
        Nivel + 1, CAST (i.Distanța+d.Distanța AS NUMERIC), d.Loc2
    FROM ierarhie i INNER JOIN distante d ON i.Loc2=d.Loc1
    WHERE Nivel < 20
)
SELECT Cale, Distanța FROM ierarhie WHERE Cale LIKE 'Iasi%Focsani'
ORDER BY Distanța
```

cale	distanța
Iasi -> Vaslui -> Birlad -> Tecuci -> Tisita -> Focsani	206
Iasi -> Tg.Frumos -> Roman -> Bacau -> Adjud -> Tisita -> Focsani	239
Iasi -> Tg.Frumos -> Roman -> Vaslui -> Birlad -> Tecuci -> Tisita -> Focsani	308
Iasi -> Tg.Frumos -> Roman -> Bacau -> Vaslui -> Birlad -> Tecuci -> Tisita -> Focsani	354

Figura 12.27. Trasee Iași-Focșani (în SQL Server)

Iar pentru a obține stațiile (localitățile) de pe acest traseu – vezi figura 12.20, interogarea va fi:

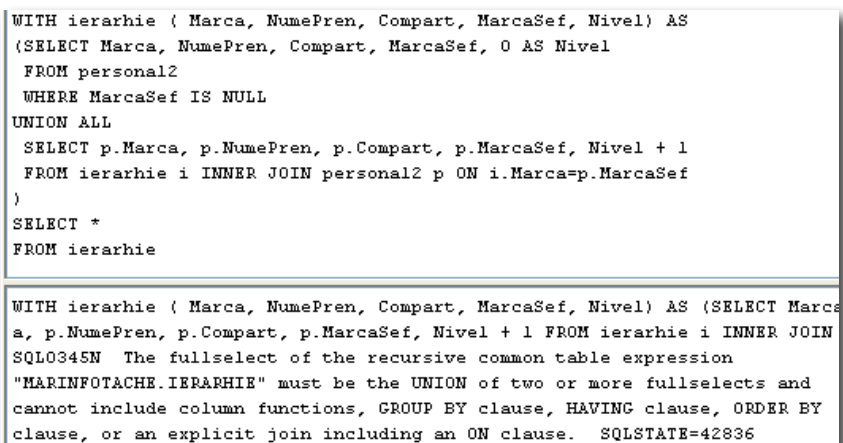
```
WITH ierarhie (Cale, Nivel, Distanța, Loc2) AS(
    SELECT CAST(Loc1 + ' -> ' + Loc2 AS VARCHAR(500)), 0 AS Nivel,
        CAST (Distanța AS NUMERIC), Loc2
    FROM distante
    WHERE Loc1='Iasi'
    UNION ALL
    SELECT CAST (i.Cale + ' -> ' + d.Loc2 AS VARCHAR(500)),
        Nivel + 1, CAST (i.Distanța+d.Distanța AS NUMERIC), d.Loc2
    FROM ierarhie i INNER JOIN distante d ON i.Loc2=d.Loc1
    WHERE Nivel < 20
)
SELECT i1.Cale, i2.Nivel + 1 AS LocNr, i2.Loc2
FROM ierarhie i1
    INNER JOIN ierarhie i2 ON i1.Cale LIKE 'Iasi -> Vaslui%Focsani'
    AND i1.Cale LIKE i2.Cale + '%'
```

ORDER BY 1,2

12.4. Interogări arborescente în DB2

Sintaxa redactării interogărilor ierarhice în DB2 seamănă binișor cu surata ei din SQL Server. La fel și logica acestor tipuri de SELECT-uri. O diferență, însă, este alergia la INNER JOIN în expresia de recursivitate – vezi figura 12.28:

```
WITH ierarhie ( Marca, NumePren, Compart, MarcaSef, Nivel) AS (
    SELECT Marca, NumePren, Compart, MarcaSef, 0 AS Nivel
    FROM personal2
    WHERE MarcaSef IS NULL
    UNION ALL
    SELECT p.Marca, p.NumePren, p.Compart, p.MarcaSef, Nivel + 1
    FROM ierarhie i
    INNER JOIN personal2 p ON i.Marca=p.MarcaSef)
SELECT * FROM ierarhie
```



```
WITH ierarhie ( Marca, NumePren, Compart, MarcaSef, Nivel) AS
(SELECT Marca, NumePren, Compart, MarcaSef, 0 AS Nivel
 FROM personal2
 WHERE MarcaSef IS NULL
 UNION ALL
 SELECT p.Marca, p.NumePren, p.Compart, p.MarcaSef, Nivel + 1
 FROM ierarhie i INNER JOIN personal2 p ON i.Marca=p.MarcaSef
 )
SELECT *
FROM ierarhie

WITH ierarhie ( Marca, NumePren, Compart, MarcaSef, Nivel) AS (SELECT Marca
a, p.NumePren, p.Compart, p.MarcaSef, Nivel + 1 FROM ierarhie i INNER JOIN
SQL0345N The fullselect of the recursive common table expression
"MARINFOTACHE.IERARHIE" must be the UNION of two or more fullselects and
cannot include column functions, GROUP BY clause, HAVING clause, ORDER BY
clause, or an explicit join including an ON clause.  SQLSTATE=42836
```

Figura 12.28. O alergie DB2 la clauza INNER JOIN a unei interogări ierarhice

Dacă însă înlocuim INNER JOIN-ul cu sintaxa joncțiunii din SQL-89 – FROM/ WHERE, lucrurile decurg fără incidente. Astfel, rezultatul din figura 12.21 se obține prin interogarea:

```
WITH ierarhie ( Marca, NumePren, Compart, MarcaSef, Nivel) AS
    (SELECT Marca, NumePren, Compart, MarcaSef, 0 AS Nivel
    FROM personal2
    WHERE MarcaSef IS NULL
    UNION ALL
    SELECT p.Marca, p.NumePren, p.Compart, p.MarcaSef, Nivel + 1
```

```

FROM ierarhie i, personal2 p
WHERE i.Marca=p.MarcaSef)
SELECT * FROM ierarhie

```

```

WITH ierarhie (Cale, Nivel, Distanta, Loc2) AS(
    SELECT CAST(Loc1 || ' -> ' || Loc2 AS VARCHAR(500)), 0 AS Nivel,
           CAST (Distanta AS NUMERIC), Loc2
    FROM distante
    WHERE Loc1='Iasi'
    UNION ALL
    SELECT CAST (i.cale || ' -> ' || d.Loc2 AS VARCHAR(500)),
           Nivel + 1, CAST (i.distanta+d.distanta AS NUMERIC), d.loc2
    FROM ierarhie i, distante d
    WHERE i.Loc2=d.Loc1 AND Nivel < 20
)
SELECT il.cale, i2.nivel + 1 AS LocNr, i2.Loc2
FROM ierarhie i1, ierarhie i2
WHERE i1.Cale LIKE 'Iasi -> Vaslui%Focsani' AND i1.Cale LIKE i2.Cale || '%'
ORDER BY 1,2

```

```

SQL0132N  A LIKE predicate or POSSTR scalar function is not valid because the
first operand is not a string expression or the second operand is not a
string.  A LOCATE or POSITION scalar function is not valid because the first
operand is not a string or the second operand is not a string expression.
SQLSTATE=42824

```

Figura 12.29. O altă alergie DB2 (la cea de-a două clauză LIKE)

Astfel, înlocuind INNER JOIN-urile cu sintaxa SQL-89 și simbolul de concatenare (din + în ||), interogările ierarhice din MS SQL Server sunt funcționale și în DB2⁹. Nu toate, însă. Cea de mai jos se împotrivește destul de ciudat (vezi figura 12.29):

```

WITH ierarhie (Cale, Nivel, Distanta, Loc2) AS (
    SELECT CAST(Loc1 || ' -> ' || Loc2 AS VARCHAR(500)), 0 AS Nivel,
           CAST (Distanta AS NUMERIC), Loc2
    FROM distante
    WHERE Loc1='Iasi'
    UNION ALL
    SELECT CAST (i.cale || ' -> ' || d.Loc2 AS VARCHAR(500)),
           Nivel + 1, CAST (i.distanta+d.distanta AS NUMERIC), d.loc2
    FROM ierarhie i, distante d

```

⁹ La drept vorbind, este de presupus MS SQL Serverul s-a inspirat din DB2, chiar dacă noi le-am prezentat în acest capitol în cronologie inversă.

```

        WHERE i.Loc2=d.Loc1 AND Nivel < 20
    )
SELECT i1.cale, i2.nivel + 1 AS LocNr, i2.Loc2
FROM ierarhie i1, ierarhie i2
WHERE i1.Cale LIKE 'Iasi -> Vaslui%Focsani' AND i1.Cale LIKE i2.Cale || '%'
ORDER BY 1,2

```

Știm măcar de unde ni se trage. Noroc că, ultimul LIKE se poate substitui cu o funcție SUBSTR, astfel că problema e rezolvată:

```

WITH ierarhie (Cale, Nivel, Distanta, Loc2) AS (
    ...
)
SELECT i1.cale, i2.nivel + 1 AS LocNr, i2.Loc2
FROM ierarhie i1, ierarhie i2
WHERE i1.Cale LIKE 'Iasi -> Vaslui%Focsani' AND
      SUBSTR(i1.Cale,1,LENGTH(i2.Cale)) = i2.Cale
ORDER BY 1,2

```

12.5. Interogări arborescente în PostgreSQL

În materie de opțiuni ierarhice și recursive, PostgreSQL-ul este mult mai modest decât greii bazelor de date¹⁰. Doar interogările clasice din primul paragraf (făcând abstracție de inexistența expresiilor-tabelă WITH SELECT...) sunt funcționale. Singura speranță este o funcție-tabelă (ce returnează un set de înregistrări) numită CONNECTBY. Sintaxa sa este destul de laborioasă¹¹. Astfel, interogarea:

```

SELECT *
FROM CONNECTBY('personal2', 'Marca', 'MarcaSef', '1', 0, '->')
AS t (Marca NUMERIC, MarcaSef NUMERIC, LEVEL INT, cale text)

```

obține liniile din figura 12.30. Ordinea argumentelor trebuie nimerită (din una sau mai multe încercări).

¹⁰ În ediția din 18 mai 2008 a săptămânalului „buletin de știri” PostgreSQL (disponibil la adresa <http://people.planetpostgresql.org/dfetter/index.php?/archives/173-PostgreSQL-Weekly-News-May-18-2008.html>) a fost anunțată comanda WITH RECURSIVE disponibilă, probabil, într-o viitoare (sub)versiune, grație unui patch al cărui autor este Tatsuoo Ishii.

¹¹ Vezi explicațiile de la adresa: <http://www.postgresql.org/docs/8.3/interactive/tablefunc.html>

marca numeric	marcasef numeric	level integer	cale text
1		0	1
2	1	1	1->2
4	2	2	1->2->4
5	2	2	1->2->5
6	5	3	1->2->5->6
7	5	3	1->2->5->7
3	1	1	1->3
8	3	2	1->3->8
9	3	2	1->3->9
10	1	1	1->10

Figura 12.30. Funcția CONNECTBY din PostgreSQL

Rezultatul unei funcții CONNECTBY este o tabelă, așa că o putem folosi după modelul subconsultărilor din clauza FROM (vezi figura 12.31):

```
SELECT *
FROM
(SELECT *
FROM CONNECTBY('personal2', 'Marca', 'MarcaSef', '1', 0, '->')
AS t (Marca NUMERIC, MarcaSef NUMERIC, LEVEL INT, cale text)
) ierarhie
INNER JOIN personal2 p ON ierarhie.Marca=p.Marca
```

marca numeric	marcasef numeric	level integer	cale text	marca numeric(5,0)	numepren character vai	datanast date	compart character vai	marcasef numeric(5,0)	saltarifar numeric(12,2)
1		0		1	ANGAJAT 1	1962-07-01	DIRECTIUNE		1600.00
2	1	1	1->2	2	ANGAJAT 2	1977-10-11	FINANCIAR	1	1450.00
4	2	2	1->2->4	4	ANGAJAT 4		FINANCIAR	2	1380.00
5	2	2	1->2->5	5	ANGAJAT 5	1965-04-30	FINANCIAR	2	1420.00
6	5	3	1->2->5->6	6	ANGAJAT 6	1965-11-09	FINANCIAR	5	1350.00
7	5	3	1->2->5->7	7	ANGAJAT 7		FINANCIAR	5	1280.00
3	1	1	1->3	3	ANGAJAT 3	1962-08-02	MARKETING	1	1450.00
8	3	2	1->3->8	8	ANGAJAT 8	1960-12-31	MARKETING	3	1290.00
9	3	2	1->3->9	9	ANGAJAT 9	1976-02-28	MARKETING	3	1410.00
10	1	1	1->10	10	ANGAJAT 10	1972-01-29	RESURSE UMAN	1	1370.00

Figura 12.31. Joncțiunea tabeli obținute prin CONNECTBY cu tabela PERSONAL2

În schimb afișarea “căii de subordonare” a fiecărui angajat, după modelul din figura 12.14, nu pare a fi la îndemână, întrucât sintaxa impune ca primul argument al funcției CONNECTBY să fie o tabelă (sau tabelă virtuală). O idee ar fi, însă, chiar dacă ne ia gura pe dinaintea capitolului 14 în care vom discuta despre tabele virtuale. Vom crea o tabelă virtuală în care structura ierarhică o vor indica nu prin perechea Marcă/MarcăŞef, ci prin perechea NumePren/NumeŞef:

```
CREATE VIEW personal2_modif01 AS (
SELECT p1.*, p2.NumePren AS NumeSef
FROM personal2 p1 LEFT OUTER JOIN personal2 p2 ON p1.MarcaSef=p2.Marca)
```

Această tabelă virtuală devine obiectul muncii unei funcții CONNECTBY:

```
SELECT *
```



```
FROM CONNECTBY('personal2_modif01', 'NumePren', 'NumeSef', 'ANGAJAT 1', 0, ' -> ')
      AS t2 (NumePren TEXT, NumeSef TEXT, LEVEL INT, cale text)
```

Rezultatul este pe măsura așteptărilor – vezi figura 12.32.

numepren text	numesef text	level integer	cale text
ANGAJAT 1		0	ANGAJAT 1
ANGAJAT 2	ANGAJAT 1	1	ANGAJAT 1 -> ANGAJAT 2
ANGAJAT 4	ANGAJAT 2	2	ANGAJAT 1 -> ANGAJAT 2 -> ANGAJAT 4
ANGAJAT 5	ANGAJAT 2	2	ANGAJAT 1 -> ANGAJAT 2 -> ANGAJAT 5
ANGAJAT 6	ANGAJAT 5	3	ANGAJAT 1 -> ANGAJAT 2 -> ANGAJAT 5 -> ANGAJAT 6
ANGAJAT 7	ANGAJAT 5	3	ANGAJAT 1 -> ANGAJAT 2 -> ANGAJAT 5 -> ANGAJAT 7
ANGAJAT 3	ANGAJAT 1	1	ANGAJAT 1 -> ANGAJAT 3
ANGAJAT 8	ANGAJAT 3	2	ANGAJAT 1 -> ANGAJAT 3 -> ANGAJAT 8
ANGAJAT 9	ANGAJAT 3	2	ANGAJAT 1 -> ANGAJAT 3 -> ANGAJAT 9
ANGAJAT 10	ANGAJAT 1	1	ANGAJAT 1 -> ANGAJAT 10

Figura 12.32. “Calea” de subordonare a fiecărui angajat

Întâmplarea face ca, în PostgreSQL, problema cu distanțele să ne fie mai la îndemână decât cea cu structura ierarhică a firmei, întrucât graful se construiește direct din Loc1 și Loc2. Fără explicații preliminare, iată interogarea PostgreSQL care ne afișează rutele între Iași și Focșani și distanțele corespunzătoare fiecărei rute – vezi figura 12.33:

```
SELECT x1.cale, SUM(x2.distanta) AS Km
FROM
  (SELECT ierarhie.Loc2 AS Loc1, ierarhie.Loc1 AS Loc2, distanta, ierarhie.Cale
   FROM (SELECT *
         FROM CONNECTBY('distanțe', 'Loc2', 'Loc1', 'Iasi', 0, '**')
         AS t2 (Loc1 TEXT, Loc2 TEXT, LEVEL INT, cale text)
        ) ierarhie, distanțe d
   WHERE ierarhie.Loc2 =d.Loc1 AND ierarhie.Loc1=d.Loc2
  ) x1,
  (SELECT ierarhie.Loc2 AS Loc1, ierarhie.Loc1 AS Loc2, distanta, ierarhie.Cale
   FROM (SELECT *
         FROM CONNECTBY('distanțe', 'Loc2', 'Loc1', 'Iasi', 0, '**')
         AS t2 (Loc1 TEXT, Loc2 TEXT, LEVEL INT, cale text)
        ) ierarhie, distanțe d
   WHERE ierarhie.Loc2 =d.Loc1 AND ierarhie.Loc1=d.Loc2
  ) x2
WHERE x1.cale LIKE 'Iasi%Focsani' AND x1.cale LIKE x2.cale || '%'
GROUP BY x1.cale
ORDER BY Km
```

cale text	km numeric
Iasi**Vaslui**Birlad**Tecuci**Tisita**Focsani	206
Iasi**Tg.Frumos**Roman**Bacau**Adjud**Tisita**Focsani	239
Iasi**Tg.Frumos**Roman**Vaslui**Birlad**Tecuci**Tisita**Focsani	308
Iasi**Tg.Frumos**Roman**Bacau**Vaslui**Birlad**Tecuci**Tisita**Focsani	354

Figura 12.33. Trasee și kilmometri între Iași și Focșani – varianta PostgreSQL

În fine, localitățile (stațiile) de pe traseul Iași-Vaslui-...Focșani (figura 12.20) se obțin prin:

```
SELECT x1.cale, x2.Nivel AS LocNr, x2.Loc2
FROM
  (SELECT ierarhie.Loc2 AS Loc1, ierarhie.Loc1 AS Loc2, distanta,
    ierarhie.Cale, LEVEL AS Nivel
  FROM (SELECT *
    FROM CONNECTBY('distanta', 'Loc2', 'Loc1', 'Iasi', 0, '**')
    AS t2 (Loc1 TEXT, Loc2 TEXT, LEVEL INT, cale text)
  ) ierarhie, distante d
  WHERE ierarhie.Loc2=d.Loc1 AND ierarhie.Loc1=d.Loc2
  ) x1,
  (SELECT ierarhie.Loc2 AS Loc1, ierarhie.Loc1 AS Loc2, distanta,
    ierarhie.Cale, LEVEL AS Nivel
  FROM (SELECT *
    FROM CONNECTBY('distanta', 'Loc2', 'Loc1', 'Iasi', 0, '**')
    AS t2 (Loc1 TEXT, Loc2 TEXT, LEVEL INT, cale text)
  ) ierarhie, distante d
  WHERE ierarhie.Loc2 =d.Loc1 AND ierarhie.Loc1=d.Loc2
  ) x2
WHERE x1.cale LIKE 'Iasi**Vaslui%Focsani' AND x1.cale LIKE x2.cale || '%'
```

12.6. Interogări mai puțin arborescente, dar tot recursive

Pentru fiecare factură, cu titlu informativ, vrem să afișăm sub forma unui șir de caractere destul de lung, lista produselor din acea factură, inclusiv cantitatea vândută și prețul unitar (consemnate în factura respectivă). Să începem cu o variantă ajutătoare. Interogarea Oracle următoare produce rezultatul din figura 12.34:

```
SELECT lf.*, SYS_CONNECT_BY_PATH(lf.NrFact || '-' || lf.Linie || '-' || DenPr, '\')
  AS Liniarizare, LEVEL
FROM liniifact lf INNER JOIN produse p ON lf.CodPr=p.CodPr
WHERE NrFact=1111
```

START WITH Linie=1

CONNECT BY PRIOR NrFact=NrFact AND PRIOR Linie=Linie-1

ORDER BY 1, 2

R	NRFAC	R	LINE	R	CODPR	R	CANTITATE	R	PRETUNIT	R	LINIARIZARE	R	LEVEL
	1111		1		1		50		1000		1111-1-Produs 1		1
	1111		2		2		75		1050		1111-1-Produs 1 \ 1111-2-Produs 2		2
	1111		3		5		500		7060		1111-1-Produs 1 \ 1111-2-Produs 2 \ 1111-3-Produs 5		3

Figura 12.34. Lista produselor din factura 1111

La fiecare linie se adaugă în listă (atributul Liniarizare): numărul facturii, numărul liniei și denumirea produsului, astfel că în rândul corespunzător liniei finale din factură avem șirul de caractere care ne interesează. Varianta SQL Server a acestei interogări Oracle este:

```
WITH ierarhie ( NrFact, Linie, CodPr, Cantitate, PretUnit, Nivel, DenPr) AS
  (SELECT NrFact, Linie, lf.CodPr, Cantitate, PretUnit, 0 AS Nivel,
    CAST ('\'+CAST (NrFact AS VARCHAR)+ ' ' +
    CAST (Linie AS VARCHAR)+'-'+DenPr AS VARCHAR(500))
  FROM liniifact lf
    INNER JOIN produse p ON lf.CodPr=p.CodPr
  WHERE NrFact=1111 AND Linie=1
    UNION ALL
  SELECT lf.NrFact, lf.Linie, lf.CodPr, lf.Cantitate, lf.PretUnit, Nivel + 1,
    CAST (ierarhie.DenPr + ' \ ' + +CAST (lf.NrFact AS VARCHAR)+
    ' ' + CAST (lf.Linie AS VARCHAR)+'-'+p.DenPr AS VARCHAR(500))
  FROM liniifact lf
    INNER JOIN ierarhie ON lf.NrFact = ierarhie.NrFact AND
    lf.Linie=ierarhie.Linie+1
    INNER JOIN produse p ON lf.CodPr=p.CodPr
  WHERE lf.NrFact=1111
  )
SELECT * FROM ierarhie
```

iar cea DB2 este:

```
WITH ierarhie ( NrFact, Linie, CodPr, Cantitate, PretUnit, Nivel, Sir) AS
  (SELECT NrFact, Linie, lf.CodPr, Cantitate, PretUnit, 0 AS Nivel,
    CAST ('\ ' || CAST (NrFact AS CHAR(8)) || ' ' ||
    CAST (Linie AS CHAR(2)) || ' ' || TRIM(DenPr) AS CHAR(250)) AS Sir
  FROM liniifact lf, produse p
  WHERE lf.CodPr=p.CodPr AND NrFact=1111 AND Linie=1
    UNION ALL
  SELECT lf.NrFact, lf.Linie, lf.CodPr, lf.Cantitate, lf.PretUnit, Nivel + 1,
```

```

        CAST ((TRIM(ierarhie.Sir) || ' \ ' || CAST (lf.NrFact AS CHAR(8)) ||
        ' ' || CAST (lf.Linie AS CHAR(2)) || ' ' || TRIM(p.DenPr))
        AS CHAR(250))
    FROM ierarhie, liniifact lf, produse p
    WHERE ierarhie.NrFact = lf.NrFact AND lf.Linie=ierarhie.Linie+1
    AND lf.CodPr=p.CodPr
)
SELECT * FROM ierarhie

```

Pe baza acestor interogări, ne putem gândi la o soluție care să ne afișeze toate produsele vândute în fiecare factură emisă în luna septembrie 2007 – vezi figura 12.35:

```

SELECT lf.*, SYS_CONNECT_BY_PATH(lf.NrFact || ' ' || lf.Linie || ' ' || DenPr, '\')
    AS Liniarizare, LEVEL
FROM liniifact lf
    INNER JOIN produse p ON lf.CodPr=p.CodPr
    INNER JOIN facturi f ON lf.NrFact=f.NrFact
WHERE EXTRACT (YEAR FROM DataFact)=2007 AND
    EXTRACT (MONTH FROM DataFact)=9 AND
    Linie = (SELECT MAX(Linie) FROM liniifact WHERE NrFact=lf.NrFact)
START WITH Linie=1
CONNECT BY PRIOR NrFact=NrFact AND PRIOR Linie=Linie-1
ORDER BY 1, 2

```

NRFAC	LINE	CODPR	CANTITATE	PREUNIT	LINIARIZARE	LEVEL
3111	3	5	510	7060	3111-1-Produs 1 \ 3111-2-Produs 2 \ 3111-3-Produs 5	3
3112	2	3	65	750	3112-1-Produs 2 \ 3112-2-Produs 3	2
3113	1	2	120	975	3113-1-Produs 2	1
3115	1	2	110	925	3115-1-Produs 2	1
3116	1	2	135	930	3116-1-Produs 2	1
3117	2	1	110	950	3117-1-Produs 2 \ 3117-2-Produs 1	2
3118	2	1	120	930	3118-1-Produs 2 \ 3118-2-Produs 1	2
3119	4	5	755	6300	3119-1-Produs 2 \ 3119-2-Produs 3 \ 3119-3-Produs 4 \ 3119-4-Produs 5	4

Figura 12.35. Lista produselor vândute în fiecare factură

Iată și versiunea SQL Server:

```

WITH ierarhie ( NrFact, Linie, CodPr, Cantitate, PretUnit, Nivel, DenPr) AS
    (SELECT NrFact, Linie, lf.CodPr, Cantitate, PretUnit, 0 AS Nivel,
        CAST ('\' + CAST (NrFact AS VARCHAR) + ' ' +
        CAST (Linie AS VARCHAR) + ' ' + DenPr AS VARCHAR(500))
    FROM liniifact lf INNER JOIN produse p ON lf.CodPr=p.CodPr
    WHERE Linie=1 AND NrFact IN (SELECT NrFact FROM facturi
        WHERE YEAR(DataFact)=2007 AND MONTH(DataFact)=9)
    UNION ALL

```

```

SELECT lf.NrFact, lf.Linie, lf.CodPr, lf.Cantitate, lf.PretUnit, Nivel + 1,
       CAST (ierarhie.DenPr + ' \ ' + +CAST (lf.NrFact AS VARCHAR)+
       ' ' + CAST (lf.Linie AS VARCHAR)+'-' +p.DenPr AS VARCHAR(500))
FROM liniifact lf INNER JOIN ierarhie ON lf.NrFact = ierarhie.NrFact AND
       lf.Linie=ierarhie.Linie+1 INNER JOIN produse p ON lf.CodPr=p.CodPr
WHERE lf.NrFact IN (SELECT NrFact FROM facturi
                    WHERE YEAR(DataFact)=2007 AND MONTH(DataFact)=9)
)
SELECT * FROM ierarhie i1
WHERE Linie = (SELECT MAX(Linie) FROM ierarhie WHERE NrFact=i1.NrFact)
ORDER BY 1

```

Ajunghând la enunțul inițial al problemei, includem în șirul de caractere al fiecărei facturi și cantitățile și prețurile unitare de vânzare din fiecare produs. Iată sintaxa Oracle:

```

SELECT f.NrFact, DataFact,
       SYS_CONNECT_BY_PATH(DenPr || ':' || Cantitate || ' ' || UM
                           || '*' || PretUnit || 'RONi ', '-\\-') AS Liniarizare
FROM liniifact lf
       INNER JOIN produse p ON lf.CodPr=p.CodPr
       INNER JOIN facturi f ON lf.NrFact=f.NrFact
WHERE EXTRACT (YEAR FROM DataFact)=2007 AND
       EXTRACT (MONTH FROM DataFact)=9 AND
       Linie = (SELECT MAX(Linie) FROM liniifact WHERE NrFact=lf.NrFact)
START WITH Linie=1
CONNECT BY PRIOR NrFact=NrFact AND PRIOR Linie=Linie-1
ORDER BY 1, 2

```

NRFACT	DATAFACT	LINIARIZARE
3111	01-09-2007	-\\-Produs 1: 57 buc*1000RONi -\\-Produs 2: 79 kg*1050RONi -\\-Produs 5: 510 buc*7060RONi
3112	01-09-2007	-\\-Produs 2: 85 kg*1030RONi -\\-Produs 3: 65 kg*750RONi
3113	02-09-2007	-\\-Produs 2: 120 kg*975RONi
3115	02-09-2007	-\\-Produs 2: 110 kg*925RONi
3116	10-09-2007	-\\-Produs 2: 135 kg*930RONi
3117	10-09-2007	-\\-Produs 2: 150 kg*1000RONi -\\-Produs 1: 110 buc*950RONi
3118	17-09-2007	-\\-Produs 2: 39 kg*1100RONi -\\-Produs 1: 120 buc*930RONi
3119	07-09-2007	-\\-Produs 2: 35 kg*1090RONi -\\-Produs 3: 40 kg*700RONi -\\-Produs 4: 55 l*1410RONi -\\-Produs 5: 755 buc*6300RONi

Figura 12.36. Lista produselor vândute (plus cantitățile și prețurile unitare) în fiecare factură cu echivalentele sale în SQL Server:

```

WITH ierarhie ( NrFact, Linie, CodPr, Cantitate, PretUnit, Nivel, DenPr) AS
(SELECT NrFact, Linie, lf.CodPr, Cantitate, PretUnit, 0 AS Nivel,
       CAST ('-\\-' + DenPr + ':' + CAST(Cantitate AS VARCHAR) + ' ' + UM
       + '*' + CAST (PretUnit AS VARCHAR) +'RONi ' AS VARCHAR(500))

```

```

        FROM liniifact lf INNER JOIN produse p ON lf.CodPr=p.CodPr
        WHERE Linie=1 AND NrFact IN (SELECT NrFact FROM facturi
            WHERE YEAR(DataFact)=2007 AND MONTH(DataFact)=9)
        UNION ALL
        SELECT lf.NrFact, lf.Linie, lf.CodPr, lf.Cantitate, lf.PretUnit, Nivel + 1,
            CAST (ierarhie.DenPr + '-\\-' + p.DenPr + ':' +
            CAST(lf.Cantitate AS VARCHAR) + '' + UM
            + '*' + CAST (lf.PretUnit AS VARCHAR) + 'RONi ' AS VARCHAR(500))
        FROM liniifact lf INNER JOIN ierarhie ON lf.NrFact = ierarhie.NrFact AND
            lf.Linie=ierarhie.Linie+1 INNER JOIN produse p ON lf.CodPr=p.CodPr
        WHERE lf.NrFact IN (SELECT NrFact FROM facturi
            WHERE YEAR(DataFact)=2007 AND MONTH(DataFact)=9)
    )
SELECT * FROM ierarhie i1
WHERE Linie = (SELECT MAX(Linie) FROM ierarhie WHERE NrFact=i1.NrFact)
ORDER BY 1

și DB2:
WITH ierarhie ( NrFact, Linie, CodPr, Cantitate, PretUnit, Nivel, DenPr) AS
    (SELECT NrFact, Linie, lf.CodPr, Cantitate, PretUnit, 0 AS Nivel,
        CAST ('-\\-' || TRIM(DenPr) || ':' || CHAR(Cantitate,',') || '' || UM
        || '*' || CHAR(PretUnit,',') || 'RONi ' AS CHAR(240)
    )
    FROM liniifact lf, produse p
    WHERE lf.CodPr=p.CodPr AND Linie=1 AND NrFact IN
        (SELECT NrFact FROM facturi WHERE YEAR(DataFact)=2007
            AND MONTH(DataFact)=9)
    UNION ALL
    SELECT lf.NrFact, lf.Linie, lf.CodPr, lf.Cantitate, lf.PretUnit, Nivel + 1,
        CAST (TRIM(ierarhie.DenPr) || '-\\-' || p.DenPr || ':'
        || CHAR(lf.Cantitate,',') || '' || UM
        || '*' || CHAR (lf.PretUnit,',') || 'RONi ' AS CHAR(240))
    FROM liniifact lf, ierarhie, produse p
    WHERE lf.NrFact = ierarhie.NrFact AND lf.CodPr=p.CodPr
        AND lf.Linie=ierarhie.Linie+1 AND lf.NrFact IN
            (SELECT NrFact FROM facturi
                WHERE YEAR(DataFact)=2007 AND MONTH(DataFact)=9)
    )
SELECT * FROM ierarhie i1
WHERE Linie = (SELECT MAX(linie) FROM ierarhie WHERE NrFact=i1.NrFact)
ORDER BY 1

```

Răspunsul este relativ mulțumitor (chiar dacă un pic cam stufos). În „construirea” recursivității, ne-am folosit (fără rușine) de atributul Linie. Nu avem, însă, bafta aceasta de fiecare dată. Spre exemplu, pentru fiecare client dorim să afișăm, sub formă de șir, toate facturile (numărul și data emiterii) din luna septembrie 2007. Cei pentru care nu avem nicio factură, nu vor apărea în rezultat – vezi figura 12.37.

DENCL	LISTA_FACTURI
Client 1 SRL	\ 3111 (01-09-2007) \ 3115 (02-09-2007) \ 3117 (10-09-2007) \ 3118 (17-09-2007)
Client 2 SA	\ 3113 (02-09-2007)
Client 3 SRL	\ 3119 (07-09-2007)
Client 5 SRL	\ 3112 (01-09-2007)
Client 7 SRL	\ 3116 (10-09-2007)

Figura 12.37. Lista facturilor emise în septembrie 2007 pentru fiecare client

Soluția Oracle cea mai la îndemână vine de la funcția OLAP ROW_NUMBER:

```
WITH num_fact AS      (SELECT f.*, ROW_NUMBER() OVER (
                        PARTITION BY CodCl ORDER BY NrFact) AS NrCrt
                        FROM facturi f
                        WHERE EXTRACT (YEAR FROM DataFact)=2007 AND
                        EXTRACT (MONTH FROM DataFact)=9
                        )
SELECT DenCl, MAX(SYS_CONNECT_BY_PATH(NrFact || ' (' || DataFact || ')', '\ '))
      AS Lista_facturi
FROM num_fact
      INNER JOIN clienti ON num_fact.CodCl=clienti.CodCl
START WITH NrCrt=1
CONNECT BY PRIOR num_fact.CodCl=num_fact.CodCl AND PRIOR NrCrt=NrCrt-1
GROUP BY DenCl ORDER BY 1
```

Varianta sa din MS SQL Server este¹²:

```
WITH ierarhie (DenCl, Sir_Facturi, NrCrt, Nivel) AS
      (SELECT DenCl, CAST ('\' + CAST (NrFact AS VARCHAR) +
      ' (' + CAST (DataFact AS VARCHAR) + ') ' AS VARCHAR(1000))
      AS Sir_Facturi, NrCrt, 0 AS Nivel
      FROM
      (SELECT DenCl, NrFact, DataFact,
      ROW_NUMBER() OVER (PARTITION BY DenCl ORDER BY
```

¹² Varianta DB2 rămâne ca temă pentru acasă.

```

        NrFact) AS NrCrt
    FROM clienti c INNER JOIN facturi f
        ON c.CodCl=f.CodCl AND YEAR(DataFact)=2007
        AND MONTH(DataFact)=9
    ) x
WHERE NrCrt=1
    UNION ALL
SELECT ierarhie.DenCl, CAST (ierarhie.Sir_Facturi + '\ ' +
    CAST (NrFact AS VARCHAR) + ' ('+ CAST (DataFact AS VARCHAR) + ')'
    AS VARCHAR(1000)), x.NrCrt, Nivel + 1
FROM ierarhie INNER JOIN
    (SELECT DenCl, NrFact, DataFact,
        ROW_NUMBER() OVER (PARTITION BY DenCl
            ORDER BY NrFact) AS NrCrt
    FROM clienti c INNER JOIN facturi f ON c.CodCl=f.CodCl
        AND YEAR(DataFact)=2007 AND MONTH(DataFact)=9
    ) x ON ierarhie.DenCl=x.DenCl AND x.NrCrt = ierarhie.NrCrt+1
)
SELECT DenCl, Sir_Facturi
FROM ierarhie
WHERE DenCl + CAST(NrCrt AS CHAR(3)) IN
    (SELECT DenCl + CAST (MAX(NrCrt) AS CHAR(3))
    FROM ierarhie GROUP BY DenCl)
ORDER BY 1

```

Una dintre cele mai insolite aplicări ale interogărilor ierarhice este generarea de valori consecutive pe un interval¹³, după modelul exemplului din paragraful 9.4.5 ! În Oracle, dacă dorim să creăm o tabelă „instant” în care o linie ar corespunde unei zile din luna septembrie (lucru realizat în PostgreSQL prin funcția GENERATE_SERIES încă din finalul paragrafului 9.4), o interogare grozav de interesantă este:

```

SELECT DATE'2007-09-01' + Nr AS Zi
FROM (SELECT LEVEL AS Nr FROM dual CONNECT BY LEVEL < 30 ) Serie

```

Echivalenta interogării de mai sus în MS SQL Server este:

```

WITH serie ( Nr) AS (SELECT 0 UNION ALL
    SELECT Nr + 1 FROM serie WHERE Nr+1 <= 29 )

```

¹³ Ideea am găsit-o în [Molinaro 2007], p.322


```
SELECT CAST ('2007-09-01' AS SMALLDATETIME) + Nr FROM serie
```

iar varianta DB2 se prezintă astfel:

```
WITH serie ( Nr) AS (SELECT 0 AS Nr FROM SYSIBM.SYSDUMMY1 UNION ALL
                     SELECT Nr + 1 FROM serie WHERE Nr+1 <= 29 )
SELECT CAST ('2007-09-01' AS DATE) + serie.Nr DAYS FROM serie
```

Pe acest calapod putem genera toate literele mari cuprinse între A și Z, pentru o agendă sau orice alt gen de problemă. Iată sintaxa Oracle¹⁴:

```
SELECT CHR(Nr)
FROM   (SELECT Nr
        FROM (SELECT LEVEL AS Nr FROM dual CONNECT BY LEVEL < 100
              ) Serie
        WHERE Nr BETWEEN 65 AND 90
       ) Litere
```

și pe cea SQL Server¹⁵:

```
WITH serie ( Nr) AS
      (SELECT 0 UNION ALL SELECT Nr + 1 FROM serie WHERE Nr+1 <= 100 )
SELECT CHAR(Nr) FROM serie WHERE Nr BETWEEN 65 AND 90
```

Prin urmare, putem rezolva mult mai ușor probleme precum cea din paragraful 9.4.5 (vezi figura 9.40): *Care sunt vânzările pentru toate cele zece zile cuprinse în intervalul 1-10 septembrie 2007 ?*

Cu sprijinul opțiunilor ierarhice/recursive, adăugăm și următoarele soluții:

- DB2:

```
WITH zile ( Zi) AS ( SELECT CAST ('2007-09-01' AS DATE) AS Zi FROM sysibm.dual
                   UNION ALL
                   SELECT Zi + 1 DAY FROM zile WHERE Zi + 1 DAY <= '2007-09-10')
SELECT Zi, TRUNCATE(COALESCE(SUM(facturi.ValFact),0),0) AS Nr_Facturi
FROM   zile      LEFT OUTER JOIN
      (SELECT f.NrFact, DataFact,
              SUM(Cantitate * PretUnit * (1+ProcTVA)) AS ValFact
      FROM   facturi f INNER JOIN liniifact lf ON lf.NrFact=f.NrFact
```

¹⁴ Anthony Molinaro, de la care am preluat ideea folosirii ierarhiilor pentru generea intervalelor, prezintă și o noutate post-10g: *SELECT ARRAY Nr FROM dual MODEL DIMENSION BY (0 idx) MEASURES (1 ARRAY) RULES ITERATE (10) (ARRAY [iteration_number] = iteration_number + 1)*

¹⁵ Varianta DB2 necesită adăugarea, în varianta SQL Server, după *SELECT 0* a clauzei *FROM sysibm.sysdummy1*

```

        INNER JOIN produse p ON lf.CodPr=p.CodPr
    GROUP BY f.NrFact, DataFact ) facturi
        ON Zi=DataFact
GROUP BY Zi ORDER BY Zi

```

- Oracle:

```

SELECT Zi, TRUNC(COALESCE(SUM(facturi.ValFact),0)) AS Nr_Facturi
FROM (SELECT DATE'2007-08-31' + Nr AS Zi
      FROM (SELECT LEVEL AS Nr FROM dual CONNECT BY LEVEL < 11
            ) Serie
      ) zile
LEFT OUTER JOIN
(SELECT f.NrFact, DataFact,
      SUM(Cantitate * PretUnit * (1+ProcTVA)) AS ValFact
FROM facturi f INNER JOIN liniifact lf ON lf.NrFact=f.NrFact
      INNER JOIN produse p ON lf.CodPr=p.CodPr
      GROUP BY f.NrFact, DataFact ) facturi
      ON Zi=TRUNC(DataFact)
GROUP BY Zi ORDER BY Zi

```

- SQL Server:

```

WITH zile ( Zi) AS (
    SELECT CAST ('2007-09-01' AS SMALLDATETIME) AS Zi
    UNION ALL
    SELECT DATEADD(DAY, 1, Zi) FROM zile
    WHERE DATEADD(DAY, 1, Zi) <= '2007-09-10' )
SELECT Zi, ROUND(COALESCE(SUM(facturi.ValFact),0),0) AS Nr_Facturi
FROM zile
LEFT OUTER JOIN
(SELECT f.NrFact, DataFact,
      SUM(Cantitate * PretUnit * (1+ProcTVA)) AS ValFact
FROM facturi f INNER JOIN liniifact lf ON lf.NrFact=f.NrFact
      INNER JOIN produse p ON lf.CodPr=p.CodPr
      GROUP BY f.NrFact, DataFact ) facturi
      ON Zi=DataFact
GROUP BY Zi ORDER BY Zi

```

Pentru PostgreSQL, cea mai la îndemână opțiune rămâne, pentru acest gen de probleme, tot funcția `GENERATE_SERIES`.

Ne interesează și o problemă evocată relativ frecvent în multe cărți/articole dedicate SQL (spre exemplu, lucrările lui Joe Celko): găsirea/umplerea golurilor pe un anumit interval, așa cum este cea din finalul capitolului 9: *Care sunt numerele de facturi nefolosite (în tabela FACTURI)?*

În capitolul 9 generam o tabelă ad-hoc în care pe fiecare linie era un număr întreg pozitiv cuprins între 1 și 9999 folosind produsul cartezian. Acum tabela ad-hoc care va fi jonționată extern cu FACTURI se obține printr-o subconsultare ierarhică/recursivă. Iată soluția Oracle și o parte din rezultatul acesteia (figura 12.38):

```
SELECT Nr, NrFact, DataFact, CodCl, Obs,
       CASE WHEN f.NrFact IS NULL THEN 'Numar nefolosit !'
       ELSE NULL
       END AS Situatie
FROM ( SELECT Nr FROM
       (SELECT LEVEL AS Nr FROM dual
        CONNECT BY LEVEL < 999999 ) Serie
       WHERE Nr BETWEEN (SELECT MIN(NrFact) FROM facturi) AND
       (SELECT MAX(NrFact) FROM facturi)
       ) bun LEFT OUTER JOIN facturi f ON Nr=NrFact
```

R	Nr	R	NrFact	R	DataFact	R	CodCl	R	Obs	SITUATIE
7	1117		1117 03-08-2007				1001		(null)	(null)
8	1118		1118 04-08-2007				1001		(null)	(null)
9	1119		1119 07-08-2007				1003		(null)	(null)
10	1120		1120 07-08-2007				1001		(null)	(null)
11	1121		1121 07-08-2007				1004		(null)	(null)
12	1122		1122 07-08-2007				1005		(null)	(null)
13	1123		(null) (null)				(null) (null)			Numar nefolosit !
14	1124		(null) (null)				(null) (null)			Numar nefolosit !
15	1125		(null) (null)				(null) (null)			Numar nefolosit !
16	1126		(null) (null)				(null) (null)			Numar nefolosit !
17	1127		(null) (null)				(null) (null)			Numar nefolosit !

Figura 12.38. Semnalizarea numerelor nefolosite pentru facturi

Și soluția DB2 este una „docilă”:

```
WITH serie ( Nr ) AS (
    SELECT MIN(NrFact) AS Nr FROM facturi UNION ALL
    SELECT Nr + 1 FROM serie
    WHERE Nr <= (SELECT MAX(NrFact) FROM facturi)
)
SELECT Nr, NrFact, DataFact, CodCl, Obs,
       CASE WHEN f.NrFact IS NULL THEN 'Numar nefolosit !' ELSE NULL
       END AS Situatie
FROM serie LEFT OUTER JOIN facturi f ON Nr=NrFact
```

În schimb, SQL Server este cu totul necooperant. Mai întâi, refuză execuția pe motiv că nu poate include în SELECT-ul de definire a recursivității a unei funcții agregat (SELECT MAX(NrFact)...) – vezi figura 12.39. Înlocuind „manual” valoarea

finală a numărului de factură, serverul o întoarce ca la Ploiești, afirmând că interogarea ierarhică se limitează la 100 de „recursivități” (fig. 12.40).

```

WITH serie (Nr) AS
(
    SELECT CAST (MIN(NrFact) AS NUMERIC(6)) AS Nr FROM facturi
    UNION ALL
    SELECT CAST (Nr+1 AS NUMERIC(6)) FROM serie WHERE Nr <=
        (SELECT MAX(NrFact) FROM facturi)
)
SELECT Nr, NrFact, DataFact, CodCl, Obs,
CASE WHEN f.NrFact IS NULL THEN 'Numar nefolosit !'
ELSE NULL
END AS Situatie
FROM serie LEFT OUTER JOIN facturi f ON Nr=NrFact

```

Messages

Msg 467, Level 16, State 1, Line 1
GROUP BY, HAVING, or aggregate functions are not allowed in the recursive part
of a recursive common table expression 'serie'.

Figura 12.39. Prima împotrîvire SQL Server (pe motiv de funcție MAX în interogarea recursivă)

Așa că, fie lucrăm cu intervale mici, fie construim o funcție stocată (am putea încerca treaba asta prin capitolul 16, dacă nu uităm).

```

WITH serie (Nr) AS
(
    SELECT CAST (MIN(NrFact) AS NUMERIC(6)) AS Nr FROM facturi
    UNION ALL
    SELECT CAST (Nr+1 AS NUMERIC(6)) FROM serie WHERE Nr <= 3119)
SELECT Nr, NrFact, DataFact, CodCl, Obs,
CASE WHEN f.NrFact IS NULL THEN 'Numar nefolosit !'
ELSE NULL
END AS Situatie
FROM serie LEFT OUTER JOIN facturi f ON Nr=NrFact

```

Results Messages

Msg 530, Level 16, State 1, Line 1
The statement terminated. The maximum recursion 100 has been exhausted before
statement completion.

Figura 12.40. Refuzul (de teama buclei infinite) SQL Server de a genera serii mai mari de 100 de valori

Nici funcția CONNECTBY din PostgreSQL nu poate să ne ajute prea mult, așa că ajungem tot la mâna funcției-tabelă GENERATE_SERIES:

```

SELECT Nr, NrFact, DataFact, CodCl, Obs, CASE WHEN f.NrFact IS NULL
      THEN 'Numar nefolosit !' ELSE NULL END AS Situatie
FROM (SELECT Nr
      FROM GENERATE_SERIES(0, 99999) Nr
      WHERE Nr BETWEEN (SELECT MIN(NrFact) FROM facturi) AND
        (SELECT MAX(NrFact) FROM facturi)

```

) bun

LEFT OUTER JOIN facturi f ON Nr=NrFact