

Capitolul 7. Interogări SQL avansate

Dacă precedentul capitol a fost unul de introducere în mecanismul de interogare al bazelor de date folosind nucleul SQL al Oracle, în continuare zăbovim preț de câteva pagini pentru câteva delicatese specifice, în general, serverelor de baze de date de categoria grea. Astfel, vom parcurge: subconsultări declarate în clauza FROM, interogări simplu și dublu corelate, subconsultări scalare, expresii tabelă, funcții analitice și interogări ierarhice.

7.1. Subconsultări în clauza FROM

Sună paradoxal din partea unor autodeclarați "veterani SQL", dar multe din soluțiile altminteri interesante se pierd în SQL din pricina acestei facilități care e atât de confortabilă încât uneori îmbie la o oarecare "lene SQL-istă". Noroc că unele SGBD-uri nu o au implementată... Lăsând gluma la o parte, trebuie recunoscut că și problemele cele mai "sângeroase" își pot afla nașul în subconsultările definite în clauza FROM.

Care sunt zilele în care au lucrat simultan Angajat 1 și Primul Angajat Nou ?
Probabil că cele patru soluții din capitolul precedent nu sunt suficiente, așa că formulăm una care să valorifice subconsultări definite în clauza FROM:

```
SELECT ZILE1.data AS Ziua
FROM
  (SELECT data
   FROM pontaje po INNER JOIN personal pe
     ON po.marca = pe.marca
   WHERE numepren = 'Angajat 1') ZILE1
INNER JOIN
  (SELECT data
   FROM pontaje po INNER JOIN personal pe
     ON po.marca = pe.marca
   WHERE numepren = 'Primul Angajat Nou') ZILE2
ON ZILE1.data = ZILE2.data
```

Prima subconsultare se materializează printr-o tabelă temporară ad-hoc numită ZILE1 ce conține zilele lucrate de primul dintre angajați. A doua generează tabela temporară ZILE2, iar clauza FROM a frazei SELECT principale le jonctionează, după arată figura 7.1.

```

SQL> --ZILE1
SQL> SELECT data
  2 FROM pontaje po INNER JOIN personal pe
  3 ON po.marca = pe.marca
  4 WHERE numepren = 'Angajat 1'
  5 /

DATA
-----
01-JUL-03
02-JUL-03
03-JUL-03
04-JUL-03
07-JUL-03
08-JUL-03
09-JUL-03
10-JUL-03
11-JUL-03
01-AUG-03
04-AUG-03
07-AUG-03
08-AUG-03
01-SEP-03
02-SEP-03
03-SEP-03
04-SEP-03
08-SEP-03
18 rows selected.

SQL> --ZILE2
SQL> SELECT data
  2 FROM pontaje po INNER JOIN personal pe
  3 ON po.marca = pe.marca
  4 WHERE numepren = 'Primul Angajat Nou'
  5 /

DATA
-----
09-JUL-03
10-JUL-03
11-JUL-03

```

↓

```

SQL> SELECT ZILE1.data AS Ziua
  2 FROM
  3 (SELECT data
  4 FROM pontaje po INNER JOIN personal pe
  5 ON po.marca = pe.marca
  6 WHERE numepren = 'Angajat 1') ZILE1
  7 INNER JOIN
  8 (SELECT data
  9 FROM pontaje po INNER JOIN personal pe
 10 ON po.marca = pe.marca
 11 WHERE numepren = 'Primul Angajat Nou') ZILE2
 12 ON ZILE1.data = ZILE2.data
 13 /

ZIUA
-----
09-JUL-03
10-JUL-03
11-JUL-03

```

Figura 7.1. Joncționarea a două subconsultări definite în clauza FROM

Care sunt orele lucrate de fiecare angajat în lunile iulie și august 2003, atât pe luni, cât și însumat ?

În clauza FROM vom defini câte o subconsultare pentru fiecare lună, iar apoi le vom joncționa extern cu tabela PERSONAL:

```

SELECT p.marca, numepren, NVL(Ore_Lucr_Iul, 0)
      AS Ore_Iulie, NVL(Ore_Lucr_Aug,0) AS Ore_Aug,
      NVL(Ore_Lucr_Iul,0) + NVL(Ore_Lucr_Aug,0)
      AS "Ore_Lucr_Iul-Aug"
FROM personal p
LEFT OUTER JOIN
  (SELECT marca, SUM(orelucrate) AS Ore_Lucr_Iul
   FROM pontaje
   WHERE TO_CHAR(data, 'MM/YYYY') = '07/2003'
   GROUP BY Marca) IULIE
  ON p.marca = IULIE.marca
LEFT OUTER JOIN
  (SELECT marca, SUM(orelucrate) AS Ore_Lucr_Aug
   FROM pontaje
   WHERE TO_CHAR(data, 'MM/YYYY') = '08/2003'
   GROUP BY Marca) AUGUST
  ON p.marca = AUGUST.marca
ORDER BY marca

```

Care sunt angajații cu numărul de zile lucrate peste cel al Primului Angajat Nou ?

În clauza FROM definim două subconsultări, ZILE_TOTI ce conține mărcile, numele și numărul zilelor lucrate de fiecare angajat, și ZILE_A1 a cărei singură linie (și coloană) conține numărul zilelor de lucru pentru Primul Angajat Nou. Cele două, să le zicem, tabele ad-hoc sunt jonctionate după condiția ZILE_TOTI.Zile_Lucrate > ZILE_A1.Zile_Lucrate:

```
SELECT marca, numepren, ZILE_TOTI.Zile_Lucrate,
       ZILE_A1.Zile_Lucrate AS Zile_A1
FROM
  (SELECT po.marca, numepren, COUNT(data) AS Zile_Lucrate
   FROM pontaje po INNER JOIN personal pe
     ON po.marca=pe.marca
   WHERE orelucrate > 0
   GROUP BY po.marca, numepren) ZILE_TOTI,
  (SELECT COUNT(data) AS Zile_Lucrate
   FROM pontaje po INNER JOIN personal pe
     ON po.marca=pe.marca
   WHERE orelucrate > 0 AND numepren = 'Primul Angajat Nou'
   GROUP BY po.marca, numepren) ZILE_A1
WHERE ZILE_TOTI.Zile_Lucrate > ZILE_A1.Zile_Lucrate
```

Care este angajatul (sau angajații) cu cel mai mare venit (obținut din orele lucrate, plus eventualele concedii de odihnă) ?

Definim două subconsultări, una care grupează veniturile obținute de angajați, și o alta care determină venitul de bază maxim; în fraza SELECT cele două se compară după cum urmează:

```
SELECT *
FROM
  (SELECT po.marca, numepren, SUM (orelucrate * salorar +
    oreco * salorarco) AS Venit_Baza
   FROM pontaje po INNER JOIN personal pe
     ON po.marca=pe.marca
   GROUP BY po.marca, numepren) VENITURI1 INNER JOIN
  (SELECT MAX(SUM (orelucrate * salorar + oreco * salorarco))
   AS Venit_Baza_Max
   FROM pontaje po INNER JOIN personal pe
     ON po.marca=pe.marca
   GROUP BY po.marca) VENITURI2
ON VENITURI1.Venit_Baza = VENITURI2.Venit_Baza_MAX
```

Care sunt angajații prezenți la lucru măcar în zilele în care a fost Angajat 3 ?

Simplă la formulare, problema ridică destule probleme, deoarece nu e vorba de un număr, ci de un set de date. Pentru a îndeplini condiția, setul de zile de lucru al fiecărui angajat trebuie să conțină setul zilelor Angajatului 3. Soluția pe care v-o prezentăm ia în calcul o zi pentru un angajat oarecare numai dacă în data respectivă Angajat 3 a fost și el la lucru:

```

SELECT marca, numepren, ZILE_GEN.Zile_Lucrate,
       ZILE_A3.Zile_Lucrate AS Zile_A3
FROM
  (SELECT po.marca, numepren, COUNT(data) AS Zile_Lucrate
   FROM pontaje po INNER JOIN personal pe
     ON po.marca=pe.marca
   WHERE orelucrate > 0 AND data IN
     (SELECT data
      FROM pontaje
      WHERE orelucrate > 0 AND marca IN
        (SELECT marca
         FROM personal
         WHERE numepren = 'Angajat 3')
      )
   GROUP BY po.marca, numepren) ZILE_GEN,
  (SELECT COUNT(data) AS Zile_Lucrate
   FROM pontaje po INNER JOIN personal pe
     ON po.marca=pe.marca
   WHERE orelucrate > 0 AND numepren = 'Angajat 3'
   GROUP BY po.marca) ZILE_A3
WHERE ZILE_GEN.Zile_Lucrate = ZILE_A3.Zile_Lucrate

```

ZILE_GEN are conținutul din figura 7.2 și numără câte sunt zilele în care angajatul de pe linia respectivă a fost lucru în același timp cu Angajat 3, fapt pentru care s-a folosit o înlănțuire de subconsultări.

```

SQL> -- ZILE_GEN
SQL> SELECT po.marca, numepren, COUNT(data) AS Zile_Lucrate
2    FROM pontaje po INNER JOIN personal pe ON po.marca=pe.marca
3    WHERE orelucrate > 0 AND data IN
4    (SELECT data FROM pontaje WHERE orelucrate > 0 AND marca IN
5    (SELECT marca FROM personal WHERE numepren = 'Angajat 3') )
6    GROUP BY po.marca, numepren
7    /

```

MARCA	NUMEPREN	ZILE_LUCRATE
101	Angajat 1	13
102	Angajat 2	7
103	Angajat 3	13
104	Angajat 4	13
105	Angajat 5	13
106	Angajat 6	13
107	Angajat 7	7
108	Angajat 8	13

Figura 7.2. Numărul de zile în care fiecare angajat a lucrat simultan cu Angajat 3

ZILE_A3 conține numărul zilelor de lucru ale Angajatului 3 (13), iar fraza SELECT principală extrage prin condiția ZILE_GEN.Zile_Lucrate = ZILE_A3.Zile_Lucrate pe toți angajații pentru care numărul zilelor lucrate simultan cu Angajat 3 este 13 – vezi figura 7.3.

MARCA	NUMEPREN	ZILE_LUCRATE	ZILE_A3
101	Angajat 1	13	13
103	Angajat 3	13	13
104	Angajat 4	13	13
105	Angajat 5	13	13
106	Angajat 6	13	13
108	Angajat 8	13	13

Figura 7.3. Numărul de zile în care fiecare angajat a lucrat simultan cu Angajat 3

Singurul lucru rămas nefăcut este eliminarea din rezultat a însuși angajatului-reper, dar aceasta e deja floare la ureche.

Care este angajatul cu numărul de zile lucrate imediat peste cel al Angajatului 7 ?

Interogarea următoare ne permite să testăm nivele de subconsultare în clauza FROM, cu atât mai interesant cu cât centrul de interes se situează în clauza HAVING a SELECT-ului principal:

```
SELECT po.marca, numepren, COUNT(data) AS Zile_Lucrate
FROM pontaje po INNER JOIN personal pe ON po.marca=pe.marca
WHERE numepren <> 'Angajat 7' AND orelucrate > 0
GROUP BY po.marca, numepren
HAVING COUNT(data) = (
    SELECT MIN(Zile_Lucrate)
    FROM
        (SELECT ZILE_L_TOTI.Zile_Lucrate
         FROM
             (SELECT po.marca, numepren, COUNT(data)
              AS Zile_Lucrate
               FROM pontaje po INNER JOIN personal pe
                ON po.marca=pe.marca
              WHERE numepren <> 'Angajat 7'
                AND orelucrate > 0
              GROUP BY po.marca, numepren
             ) ZILE_L_TOTI,
            (SELECT COUNT(data) AS Zile_Lucrate
             FROM pontaje po INNER JOIN personal pe
              ON po.marca=pe.marca
              WHERE orelucrate > 0 AND numepren = 'Angajat 7'
              GROUP BY po.marca) ZILE_A7
         WHERE ZILE_L_TOTI.Zile_Lucrate > ZILE_A7.Zile_Lucrate
        )
    )
```

MARCA	NUMEPREN	ZILE_LUCRATE
103	Angajat 3	13

Figura 7.4. Angajat 3 este cel care are numărul de zile lucrate imediat peste cel al Angajatului 7

Care sunt cele mai mari două salarii orare ?

La această interogare se poate formula una din cele mai simple soluții folosind o subconsultare și pseudo-coloana ROWNUM.

```
SELECT salorar
FROM
  (SELECT DISTINCT salorar
   FROM personal
   ORDER BY salorar DESC
  )
WHERE ROWNUM <= 2
```


<pre>SQL> SELECT DISTINCT salorar 2 FROM personal 3 ORDER BY salorar DESC 4 /</pre>		<pre>SQL> SELECT salorar 2 FROM 3 (SELECT DISTINCT salorar 4 FROM personal 5 ORDER BY salorar DESC 6) 7 WHERE ROWNUM <= 2 8 /</pre>
<pre> SALORAR ----- 75000 71500 67500 62500 61500 57500 56000 55500 54500 50500 </pre>		<pre> SALORAR ----- 75000 71500 </pre>

Figura 7.5. Primele două salare orare – soluție bazată pe ROWNUM și subconsultare

Avantajul acestei soluții ține de generalitatea ei. Astfel, dacă ne-ar interesa *Primii cinci clasați în topul angajaților cu cele mai mari venituri*, nu ar trebui să umblăm prea mult la logica interogării:

```
SELECT *
FROM
  (SELECT po.marca, numepren, SUM (orelucrate * salorar +
    oreco * salorarco) AS Venit_Baza
   FROM pontaje po INNER JOIN personal pe
    ON po.marca=pe.marca
   GROUP BY po.marca, numepren
   ORDER BY 3 DESC)
WHERE ROWNUM <= 5
```

MARCA	NUMEPREN	VENIT_BAZA
104	Angajat 4	10800000
106	Angajat 6	10296000
103	Angajat 3	9660000
105	Angajat 5	9000000
107	Angajat 7	8784000

Figura 7.6. Primele cinci poziții din topul celor mai bine plătiți angajați

7.2. Interogări corelate. Operatorul EXISTS

Interogările corelate reprezintă una dintre cele mai greu de deprins facilități ale SQL-ului. Detalii semnificative despre corelarea simplă și dublă sunt prezentate în lucrarea unuia dintre autorii cărții de față, lucrare pe care ne tot străduim să nu o eclipsăm, așa că vom discuta sumar doar câteva exemple.

Care sunt colegii de compartiment ai lui Angajat 2 ?

Tot o subconsultare constituie miezul soluției, însă una specială – corelată:

```
SELECT *
FROM personal pe1
WHERE numepren <> 'Angajat 2' AND EXISTS
  (SELECT marca
   FROM personal pe2
   WHERE numepren = 'Angajat 2'
    AND pe2.compart = pe1.compart )
```

Pentru a include o linie din PE1 în rezultat trebuie ca, pentru aceasta, să existe în PE2 cel puțin o linie în care numele să fie 'Angajat 2' iar compartimentul să fie identic cu cel al liniei curente din PE1. Ceva mai rar, corelarea se realizează și prin operatorul IN:

```
SELECT *
FROM personal pe1
WHERE numepren <> 'Angajat 2' AND compart IN
  (SELECT compart
   FROM personal pe2
   WHERE numepren = 'Angajat 2'
    AND pe2.compart = pe1.compart )
```

Care sunt zilele în care au lucrat simultan Angajat 1 și Primul Angajat Nou ?

Condiția este: zilele în care a lucrat primul angajat reper să fie aceleași ca și zilele în care a venit la lucru al doilea, așa încât corelarea se realizează după atributul Data. Pentru simplificare, subconsultarea extrage doar valoarea (constantă) 1:

```
SELECT data AS Ziua
```

```

FROM pontaje po1 INNER JOIN personal pe1
  ON po1.marca = pe1.marca
WHERE numepren = 'Angajat 1' AND EXISTS
  (SELECT 1
   FROM pontaje po2 INNER JOIN personal pe2
     ON po2.marca = pe2.marca
   WHERE numepren = 'Primul Angajat Nou'
     AND po2.data = po1.data)

```

Care sunt angajații care, în total, au exact 12 zile de lucru ?

Ar fi prea simplu să recurgem la:

```

SELECT po.marca, numepren, COUNT(data) AS Zile_Lucrate
FROM pontaje po INNER JOIN personal pe ON po.marca=pe.marca
WHERE orelucrate > 0
GROUP BY po.marca, numepren
HAVING COUNT(data) = 12

```

Așa că o să ne complicăm cu o soluție care ar merita, totuși, un premiu de frumusețe:

```

SELECT marca, numepren
FROM personal
WHERE 12 =
  (SELECT COUNT(data)
   FROM pontaje
   WHERE orelucrate > 0 AND
        pontaje.marca = personal.marca)

```

Care dintre angajați a fost la lucru în toate zilele lucrătoare ?

Subconsultarea corelată conține o clauză HAVING în care se calculează numărul zilelor lucrătoare, număr comparat cu cel al datelor lucrate de angajatul de pe linia curentă din PE:

```

SELECT marca, numepren
FROM personal pe
WHERE EXISTS
  (SELECT 1
   FROM pontaje po
   WHERE orelucrate > 0 AND po.marca = pe.marca
   GROUP BY marca
   HAVING COUNT(data) =
     (SELECT COUNT(DISTINCT data)
      FROM pontaje
      WHERE orelucrate > 0 )
  )

```

Care sunt angajații cu cele mai mari trei salarii orare ?

Tot la categoria delicatase SQL încadrăm și varianta următoare:

```
SELECT *
FROM personal pe1
WHERE 3 >
      (SELECT COUNT (DISTINCT salorar)
       FROM personal pe2
        WHERE pe2.salaror > pe1.salaror)
ORDER BY salorar DESC
```

Scenariul este următorul: se parcurge, linie cu linie, prima instanță a tabelului PERSONAL – PE1. Orice linie *i* este inclusă în rezultat numai dacă rezultatul subconsultării corelate numără în a doua instanță PERSONAL – PE2 mai puțin de 3 linii pentru care salariul orar este mai mare decât cel de pe linia curentă din PE1. Logica este un pic curioasă: dacă pentru o linie dată sunt mai puțin de trei înregistrări în care salariul orar este peste cel din linia dată, înseamnă că linia respectivă se înscrie în primele trei.

Extrageți primii cinci clasati în topul angajaților cu cele mai mari venituri

Prin comparație cu problema precedentă, elementul suplimentar de dificultate ține de corelarea la nivel de grup, și nu linie:

```
SELECT pol.marca, numepren, SUM (pol.orelucrate *
      pe1.salaror + pol.oreco * pe1.salararco) AS Venit_Baza
FROM pontaje pol INNER JOIN personal pe1
      ON pol.marca=pe1.marca
GROUP BY pol.marca, numepren
HAVING 5 >
      (SELECT COUNT(SUM(po2.orelucrate * pe2.salaror +
      po2.oreco * pe2.salararco))
      FROM pontaje po2 INNER JOIN personal pe2
       ON po2.marca=pe2.marca
      GROUP BY po2.marca
      HAVING SUM (po2.orelucrate * pe2.salaror +
      po2.oreco * pe2.salararco) >
      SUM (pol.orelucrate * pe1.salaror +
      pol.oreco * pe1.salararco)
      )
ORDER BY 3 DESC
```

Care dintre angajați a fost la lucru în toate zilele lucrătoare ?

Revenim la această problemă, încercând, de data aceasta, o soluție care să valorifice dubla corelare:

```
SELECT DISTINCT pol.marca, numepren
FROM personal pe1 INNER JOIN pontaje pol
      ON pe1.marca=pol.marca
WHERE orelucrate > 0 AND NOT EXISTS
```

```
(SELECT 1
  FROM pontaje po2
 WHERE orelucrate > 0 AND NOT EXISTS
      (SELECT 1
        FROM pontaje po3 INNER JOIN personal pe3
          ON po3.marca=pe3.marca
        WHERE orelucrate > 0 AND pe3.marca=pe1.marca
          AND po3.data=po2.data)
)
```

Care sunt angajații prezenți la lucru măcar în zilele în care a fost Angajat 3 ?
Iată un al doilea exemplu de dublu corelare:

```
SELECT DISTINCT pe1.marca, numepren
FROM personal pe1 INNER JOIN pontaje po1
  ON pe1.marca = po1.marca
WHERE orelucrate > 0 AND NOT EXISTS
      (SELECT 1
        FROM personal pe2 INNER JOIN pontaje po2
          ON pe2.marca=po2.marca
        WHERE orelucrate > 0 AND numepren = 'Angajat 3'
          AND NOT EXISTS
              (SELECT 1
                FROM personal pe3 INNER JOIN pontaje po3
                  ON pe3.marca=po3.marca
                WHERE orelucrate > 0
                  AND pe3.marca=pe1.marca
                  AND po3.data=po2.data
              )
      )
```

Credem că aceste ultime două interogări constituie un foarte nimerit promo la cartea de SQL de care pomeneam...

7.3. Subconsultări scalare în clauza SELECT

O subconsultare scalară este cea care furnizează o singură valoare (o singură linie/coloană). Definiția sună cât se poate de simplu, însă Oracle abia în versiunea 9 a introdus această facilități spumoasă (urmând exemplul standardelor SQL și “caprei” DB2 a vecinului IBM).

Care este totalul venitului de bază pentru fiecare angajat ?
Putem încerca o variantă în care clauza SELECT conține o subconsultare scalară ce returnează venitul pentru angajatul de pe linia curentă. Interogarea scalară este corelată prin atributul Marcă de liniile din PERSONAL (PE):

```
SELECT marca, numepren,
```

```

        (SELECT SUM (pol.orelucrate * pe1.salorar +
                    pol.oreco * pe1.salorarco)
        FROM pontaje pol INNER JOIN personal pe1
            ON pol.marca=pe1.marca
        WHERE pe.marca=pe1.marca) AS Venit_Baza
FROM personal pe
ORDER BY numepren

```

Bazându-ne pe corelare, putem renunța la joncțiunea internă din subconsultarea scalară:

```

SELECT marca, numepren,
        (SELECT SUM (po.orelucrate * pe.salorar +
                    po.oreco * pe.salorarco)
        FROM pontaje po
        WHERE po.marca=pe.marca) AS Venit_Baza
FROM personal pe
ORDER BY numepren

```

Să se calculeze, la nivelul firmei, următoarele date:

- *total ore lucrate;*
- *total ore concediu;*
- *total ore noapte;*
- *total venit de bază.*

Rezultatul va conține o singură linie – vezi figura 7.7. Fiecare dată cerută va fi furnizată de o subconsultare scalară. Deoarece nu avem o tabelă "principală", folosim "surogatul" DUAL.

```

SELECT
    (SELECT SUM (orelucrate) FROM pontaje)
        AS Total_Ore_Lucrate,
    (SELECT SUM (oreco) FROM pontaje) AS Total_Ore_Concediu,
    (SELECT SUM (orenoapte) FROM pontaje) AS Total_Ore_Noapte,
    (SELECT SUM (po.orelucrate * pe.salorar +
                po.oreco * pe.salorarco)
    FROM pontaje po INNER JOIN personal pe
        ON po.marca=pe.marca) AS Total_Venit_Baza
FROM DUAL

```

```

SQL> SELECT
  2 (SELECT SUM (orelucrate) FROM pontaje) AS Total_Ore_Lucrate,
  3 (SELECT SUM (oreco) FROM pontaje) AS Total_Ore_Concediu,
  4 (SELECT SUM (orenoapte) FROM pontaje) AS Total_Ore_Noapte,
  5 (SELECT SUM (po.orelucrate * pe.salorar + po.oreco * pe.salorarco)
  6 FROM pontaje po INNER JOIN personal pe
  7 ON po.marca=pe.marca) AS Total_Venit_Baza
  8 FROM DUAL
  9 /

```

TOTAL_ORE_LUCRATE	TOTAL_ORE_CONCEDIU	TOTAL_ORE_NOAPTE	TOTAL_VENIT_BAZA
1064	136	14	75204000

Figura 7.7. Informații sintetice obținute cu interogări scalare

Care sunt zilele în care au lucrat simultan Angajat 1 și Primul Angajat Nou ?

Declarăm două interogări scalare în clauza `SELECT`, una care referitoare la zilele lucrate de primul angajat, iar cealaltă, similară, privitoare la cel de-al doilea angajat. Corelarea se realizează cu liniile tabeli `PONTAJE` (fiecare zi de lucru):

```

SELECT
  (SELECT data
   FROM pontaje po INNER JOIN personal pe
     ON po.marca = pe.marca
   WHERE numepren = 'Angajat 1'
   AND po.data=po0.data) ZILE1,
  (SELECT data
   FROM pontaje po INNER JOIN personal pe
     ON po.marca = pe.marca
   WHERE numepren = 'Primul Angajat Nou'
   AND po.data=po0.data
  ) ZILE2
FROM pontaje po0
WHERE zile1=zile2

```

Intenția noastră este, de fapt, să testăm dacă rezultatele a două interogări corelate în clauza `SELECT` pot fi folosite și în clauza `WHERE`. Ei, bine, nu se poate – vezi figura 7.8.

```

SQL> SELECT (SELECT data
2   FROM pontaje po INNER JOIN personal pe ON po.marca = pe.marca
3   WHERE numepren = 'Angajat 1' AND po.data=po0.data) ZILE1,
4   (SELECT data
5   FROM pontaje po INNER JOIN personal pe ON po.marca = pe.marca
6   WHERE numepren = 'Primul Angajat Nou' AND po.data=po0.data) ZILE2
7   FROM pontaje po0
8   WHERE zile1=zile2
9   /
WHERE zile1=zile2
      *
ERROR at line 8:
ORA-00904: "ZILE2": invalid identifier

```

Figura 7.8. Atributele obținute prin interogări scalare nu pot fi folosite în clauza WHERE

Varianta câștigătoare se obține copiind subconsultările în clauza WHERE, unde vor fi comparate:

```

SELECT DISTINCT
  (SELECT data
   FROM pontaje po INNER JOIN personal pe
     ON po.marca = pe.marca
   WHERE numepren = 'Angajat 1'
     AND po.data=po0.data) ZILE1,
  (SELECT data
   FROM pontaje po INNER JOIN personal pe
     ON po.marca = pe.marca
   WHERE numepren = 'Primul Angajat Nou'
     AND po.data=po0.data) ZILE2
FROM pontaje po0
WHERE
  (SELECT data
   FROM pontaje po INNER JOIN personal pe
     ON po.marca = pe.marca
   WHERE numepren = 'Angajat 1' AND po.data=po0.data) =
  (SELECT data
   FROM pontaje po INNER JOIN personal pe
     ON po.marca = pe.marca
   WHERE numepren = 'Primul Angajat Nou'
     AND po.data=po0.data)

```

Soluția pare, totuși, prea brutală, așa că o mai îndulcim printr-o subconsultare în clauza FROM ce filtrează din PONTAJE numai zilele referitoare la primul angajat, iar singura subconsultare scalară este mutată în clauza WHERE:

```

SELECT data
FROM
  (SELECT data
   FROM pontaje po INNER JOIN personal pe
     ON po.marca = pe.marca
   WHERE numepren = 'Angajat 1'

```

```

) po0
WHERE data =
(SELECT data
FROM pontaje po INNER JOIN personal pe
ON po.marca = pe.marca
WHERE numepren = 'Primul Angajat Nou'
AND po.data=po0.data)

```

Care este procentul de prezență la lucru în fiecare zi din luna iulie 2003 ?

Pentru fiecare zi lucrătoare din iulie 2003, ne interesează raportul dintre numărul angajaților prezenți la lucru și numărul total de angajați ai firmei, așa încât recurgem la două subconsultări scalare, pe care le împărțim:

```

SELECT data,
(SELECT COUNT(*) FROM personal) AS Nr_Angajati_Total,
(SELECT COUNT(*) FROM pontaje po WHERE orelucrate >0
AND po.data = p_iulie.data) AS Prezenti,
((SELECT COUNT(*) FROM pontaje po WHERE orelucrate >0
AND po.data = p_iulie.data) /
(SELECT COUNT(*) FROM personal)) * 100
AS Procent_Prezenta
FROM
(SELECT *
FROM pontaje
WHERE TO_CHAR(data, 'MM/YYYY')='07/2003') P_IULIE
GROUP BY data

```

Rezultatul este cel din figura 7.9.

DATA	NR_ANGAJATI_TOTAL	PREZENTI	PROCENT_PREZENTA
01-JUL-03	10	8	80
02-JUL-03	10	6	60
03-JUL-03	10	6	60
04-JUL-03	10	6	60
07-JUL-03	10	6	60
08-JUL-03	10	7	70
09-JUL-03	10	9	90
10-JUL-03	10	9	90
11-JUL-03	10	9	90

Figura 7.9. Procentul de prezență pentru fiecare zi lucrătoare din iulie 2003

Să se calculeze partea fiecărui angajat din totalul veniturilor de bază.

După calapodul exemplului precedent, folosim două subconsultări scalare, una care calculează totalul general al veniturilor de bază, și o alta pentru determinarea venitului angajatului de pe linia curentă a tabeli PERSONAL (rezultatul se prezintă ca în figura 7.10):

```

SELECT marca, numepren,
(SELECT SUM (orelucrate * salorar + oreco * salorarco)

```

```

FROM pontaje po INNER JOIN personal pe ON
    po.marca=pe.marca) Total_Venituri,
(SELECT SUM (orelucrate * salorar + oreco * salorarco)
AS Venit_Baza
FROM pontaje po INNER JOIN personal pe
ON po.marca=pe.marca
WHERE pe.marca = personal.marca ) Venit_Angajat,
ROUND ((SELECT SUM (orelucrate * salorar + oreco
* salorarco) AS Venit_Baza
FROM pontaje po INNER JOIN personal pe
ON po.marca=pe.marca
WHERE pe.marca = personal.marca ) /
(SELECT SUM (orelucrate * salorar +
oreco * salorarco)
FROM pontaje po INNER JOIN personal
pe ON po.marca=pe.marca) * 100,
2) AS Procent
FROM personal

```

MARCA	NUMEPREN	TOTAL_VENITURI	VENIT_ANGAJAT	PROCENT
109	Primul Angajat Nou	75204000	1332000	1.77
110	Al Doilea Angajat Nou	75204000	1212000	1.61
101	Angajat 1	75204000	8064000	10.72
102	Angajat 2	75204000	8208000	10.91
103	Angajat 3	75204000	9660000	12.85
104	Angajat 4	75204000	10800000	14.36
105	Angajat 5	75204000	9000000	11.97
106	Angajat 6	75204000	10296000	13.69
107	Angajat 7	75204000	8784000	11.68
108	Angajat 8	75204000	7848000	10.44

Figura 7.10. Procentul fiecărui angajat în totalul veniturilor de bază

Care este evoluția zilnică a prezenței în iulie 2003, prin raportare la ziua calendaristică anterioară ?

Cheia soluției ține de corelarea celor două două subconsultări scalare la data de pe linia curentă din P_IULIE:

```

SELECT data,
    (SELECT COUNT(*) FROM pontaje po WHERE orelucrate >0
    AND po.data = p_iulie.data)
    AS Prezenti_Zi_Crt,
    (SELECT COUNT(*) FROM pontaje po WHERE orelucrate >0
    AND po.data = p_iulie.data - 1)
    AS Prezenti_Zi_Anter,
    (SELECT COUNT(*) FROM pontaje po WHERE orelucrate >0
    AND po.data = p_iulie.data) -
    (SELECT COUNT(*) FROM pontaje po
    WHERE orelucrate >0 AND po.data = p_iulie.data -
    1)

```

```

        AS Diferenta
FROM
    (SELECT * FROM pontaje
     WHERE TO_CHAR(data,'MM/YYYY')='07/2003') P_IULIE
GROUP BY data

```

Pentru prima zi din iulie, atributul `Prezenti_Zi_Anter` are valoarea zero, deoarece nu interesează eventualele pontaje de pe iunie – vezi figura 7.11. Situație este similară zilei de 7 iulie, deoarece ziua precedentă, 6 iulie, a fost o duminică (zi mai nelucrătoare decât celelalte).

DATA	PREZENTI_ZI_CRT	PREZENTI_ZI_ANTER	DIFERENTA
01-JUL-03	8	0	8
02-JUL-03	6	8	-2
03-JUL-03	6	6	0
04-JUL-03	6	6	0
07-JUL-03	6	0	6
08-JUL-03	7	6	1
09-JUL-03	9	7	2
10-JUL-03	9	9	0
11-JUL-03	9	9	0

Figura 7.11. Comparație zi curentă – precedentă în privința prezenței

Care este evoluția zilnică a prezenței în iulie 2003, prin raportare la ziua lucrătoare anterioară ?

Problema este mult mai interesantă decât precedenta, deoarece raportarea se face la ziua calendaristică anterioară, dar numai dacă aceasta a fost lucrătoare. De aceea, a doua subconsultare scalară apelează la o subconsultare prin care se determină ziua lucrătoare precedentă:

```

SELECT data,
    (SELECT COUNT(*) FROM pontaje po WHERE orelucrate >0
     AND po.data = p_iulie.data) AS Prezenti_Zi_Crt,
    (SELECT COUNT(*) FROM pontaje po WHERE orelucrate >0
     AND po.data = (SELECT MAX(data) FROM pontaje po2
                     WHERE orelucrate > 0
                     AND po2.data < p_iulie.data)
    ) AS Prezenti_Pontaj_Anter,
    (SELECT COUNT(*) FROM pontaje po WHERE orelucrate >0
     AND po.data = p_iulie.data) -
    (SELECT COUNT(*) FROM pontaje po WHERE orelucrate >0
     AND po.data = (SELECT MAX(data) FROM pontaje po2
                     WHERE orelucrate > 0
                     AND po2.data < p_iulie.data) ) AS Diferenta
FROM
    (SELECT * FROM pontaje
     WHERE TO_CHAR(data,'MM/YYYY')='07/2003') P_IULIE
GROUP BY data

```


DATA	PREZENTI_ZI_CRT	PREZENTI_PONTAJ_ANTER	DIFERENȚA
01-JUL-03	8	0	8
02-JUL-03	6	8	-2
03-JUL-03	6	6	0
04-JUL-03	6	6	0
07-JUL-03	6	6	0
08-JUL-03	7	6	1
09-JUL-03	9	7	2
10-JUL-03	9	9	0
11-JUL-03	9	9	0

Figura 7.12. Comparație zi curentă – zi lucrătoare precedentă în privința prezenței

După aceste exemple în care subconsultările scalare și-au arătat virtuțile, nu ne mai rămân decât regretăm că nu le putem include nici în clauzele `DEFAULT` ale atributelor, nici în reguli de validare (`CHECK`) la nivel de atribut/înregistrare și nici în `GROUP BY`, `HAVING` și structuri `CASE`.

7.4. Expresii tabelă

Expresiile tabelă constituie o altă facilitate SQL introdusă în Oracle 9i prin care o tabelă poate fi definită ad-hoc înainte de clauza `SELECT`. Principalul atu este posibilitatea de a folosi attributele sale în clauza `WHERE`.

Care sunt angajații cu venitul de bază total mai mare de 8 milioane lei ?

Prin clauza `WITH` declarăm tabela `VENITURI` ce calculează venitul de bază al fiecărui angajat, iar în clauza `WHERE` apare o condiție formulată asupra unui atribut din această tabelă:

```
WITH venituri AS
  (SELECT po.marca, numepren, SUM (orelucrate * salorar +
    oreco * salorarco) AS Venit_Baza
   FROM pontaje po INNER JOIN personal pe
    ON po.marca=pe.marca
   GROUP BY po.marca, numepren)
SELECT *
FROM venituri
WHERE venit_baza >= 8000000
```

Care este angajatul (sau angajații) cu cel mai mare venit (obținut din orele lucrate, plus eventualele concedii de odihnă) ?

Ciudat, dacă dorim să jonționăm două instanțe ale tabeli ad-hoc `VENITURI`, obținem un mesaj de eroare de genul celui din figura 7.13 pe care nu ni l-am putut explica:

```
WITH venituri AS
  (SELECT numepren, SUM (orelucrate * salorar +
```

```

        oreco * salorarco) AS Venit_Baza
    FROM pontaje po INNER JOIN personal pe
        ON po.marca=pe.marca
    GROUP BY numepren)
SELECT v1.*
FROM venituri v1 INNER JOIN
    (SELECT MAX(venit_baza) AS venit_baza FROM venituri) vmax
    ON v1.venit_baza = vmax.venit_baza

SQL> WITH venituri AS
  2  (SELECT numepren, SUM (orelucrate * salorar + oreco * salorarco) AS Venit_Baza
  3  FROM pontaje po INNER JOIN personal pe ON po.marca=pe.marca
  4  GROUP BY numepren)
  5  SELECT v1.*
  6  FROM venituri v1 INNER JOIN
  7  (SELECT MAX(venit_baza) AS venit_baza FROM venituri) vmax
  8  ON v1.venit_baza = vmax.venit_baza
  9  /
(SELECT numepren, SUM (orelucrate * salorar + oreco * salorarco) AS Venit_Baza
    *)
ERROR at line 2:
ORA-00604: error occurred at recursive SQL level 1
ORA-00904: "from$_subquery$_003"."NUMEPREN_7_7": invalid identifier

```

Figura 7.13. Ciudățeni Oracle – partea I

Remediul vine de la o altă tabelă ad-hoc, VMAX, ce conține numai venitul de bază maxim la nivelul firmei – vezi figura 7.14:

```

WITH venituri AS
    (SELECT numepren, SUM (orelucrate * salorar +
        oreco * salorarco) AS Venit_Baza
    FROM pontaje po INNER JOIN personal pe
        ON po.marca=pe.marca
    GROUP BY numepren),
    vmax AS
    (SELECT MAX(SUM (orelucrate * salorar +
        oreco * salorarco)) AS Venit_Max
    FROM pontaje po INNER JOIN personal pe
        ON po.marca=pe.marca
    GROUP BY numepren)
SELECT numepren, venit_baza
FROM venituri INNER JOIN vmax ON venit_baza = venit_max

```

```

SQL> WITH
2  venituri AS
3    (SELECT numepren, SUM (orelucrate * salorar + oreco * salorarco) AS Venit_Baza
4    FROM pontaje po INNER JOIN personal pe ON po.marca=pe.marca
5    GROUP BY numepren),
6  vmax AS
7    (SELECT MAX(SUM (orelucrate * salorar + oreco * salorarco)) AS Venit_Max
8    FROM pontaje po INNER JOIN personal pe ON po.marca=pe.marca
9    GROUP BY numepren)
10 SELECT numepren, venit_baza
11 FROM venituri INNER JOIN vmax ON venit_baza = venit_max
12 /

```

NUMEPREN	VENIT_BAZA
Angajat 4	10800000

Figura 7.14. Două tabele definite în clauza WITH

Care sunt angajații cu un număr de zile lucrate peste cel al Primului Angajat Nou ?
 Ideea ar fi să definim o tabelă ad-hoc ZILE ce conține numărul zilelor în care a fost la lucru fiecare angajat:

```

WITH zile AS
  (SELECT po.marca, numepren, COUNT(data) AS Zile_Lucrate
   FROM pontaje po INNER JOIN personal pe
     ON po.marca=pe.marca
   WHERE orelucrate > 0
   GROUP BY po.marca, numepren)
SELECT *
FROM zile
WHERE zile_lucrate >
  (SELECT zile_lucrate
   FROM zile
   WHERE numepren = 'Primul Angajat Nou')

```

Din păcate, Oracle ne tratează cu același mesaj înălțător – vezi figura 7.15. Ei, bine, dacă în subconsultare se folosește o altă tabelă ad-hoc, scăpăm de mesajul de eroare:

```

WITH zile AS
  (SELECT po.marca, numepren, COUNT(data) AS Zile_Lucrate
   FROM pontaje po INNER JOIN personal pe
     ON po.marca=pe.marca
   WHERE orelucrate > 0
   GROUP BY po.marca, numepren) ,
  zile1 AS
  (SELECT COUNT(data) AS Zile_Lucrate
   FROM pontaje po INNER JOIN personal pe
     ON po.marca=pe.marca
   WHERE orelucrate > 0 AND numepren = 'Primul Angajat Nou')
SELECT * FROM zile WHERE zile_lucrate >
  (SELECT zile_lucrate FROM zile1)

```

```

SQL> WITH zile AS
  2 (SELECT po.marca, numepren, COUNT(data) AS Zile_Lucrate
  3    FROM pontaje po INNER JOIN personal pe ON po.marca=pe.marca
  4    WHERE orelucrate > 0
  5    GROUP BY po.marca, numepren)
  6 SELECT *
  7 FROM zile
  8 WHERE zile_lucrate >
  9    (SELECT zile_lucrate
 10     FROM zile
 11     WHERE numepren = 'Primul Angajat Nou')
 12 /
(SELECT po.marca, numepren, COUNT(data) AS Zile_Lucrate
                                     *
ERROR at line 2:
ORA-00604: error occurred at recursive SQL level 1
ORA-00904: "from$_subquery$_003"."NUMEPREN_7_7": invalid identifier

```

Figura 7.15. Ciudățenii Oracle – episodul 2

7.5. Funcții OLAP

De la versiunea 8i, din care au fost introduse așa numitele funcții analitice, este din ce în ce mai greu de inventariat tot ceea ce oferă Oracle în materie de SQL dedicat procesării analitice on-line (On Line Analytical Processing). Fiecare nouă variantă vine cu exotismul său în materie de funcții din această categorie. Scopul acestui paragraf este mult mai modest, anume de a prezenta câteva din cele mai des folosite opțiuni pentru analiza datelor.

7.5.1. Subtotaluri

Reamintim o problemă care, la momentul formulării, nu era chiar din cale-afară de simplă:

Să se afișeze venitul de bază corespunzător orelor lucrate de fiecare angajat pe luna iulie 2003, calculându-se și subtotaluri pe compartimente, precum și un total general

Paragraful 6.7 este cel în care am lansat problema subtotalurilor pe compartimente și totalului general pentru orele lucrate. Interogarea al cărei rezultat era reprezentat în figura 6.52 avea nevoie de trei fraze SELECT conectate prin operatorul UNION. Cu funcția analitică ROLLUP lucrurile se simplifică vizibil:

```

SELECT compart, numepren, SUM(orelucrate * salorar)
  AS Venit_Baza
FROM personal pe LEFT OUTER JOIN pontaje po
  ON pe.marca=po.marca
  AND TO_CHAR(data, 'MM/YYYY')='07/2003'
GROUP BY ROLLUP(compart, numepren)
ORDER BY compart, numepren

```

Cum funcția `ROLLUP` are două argumente, vor fi calculate două (sub)totaluri. Ceea ce putem reproșa rezultatului din figura 7.16 sunt spațiile în spatele cărora ghicim subtotalurile și totalul general. De fapt, spațiile afișate în SQL*Plus sunt valori `NULL`.

COMPART	NUMEPREN	VENIT_BAZA
CONTA	Angajat 2	2300000
CONTA	Angajat 6	5148000
CONTA	Angajat 7	2460000
CONTA		9908000
IT	Angajat 1	4032000
IT	Angajat 3	2700000
IT	Angajat 5	4500000
IT	Primul Angajat Nou	1332000
IT		12564000
PROD	Al Doilea Angajat Nou	1212000
PROD	Angajat 4	5400000
PROD	Angajat 8	3924000
PROD		10536000
		33008000

Figura 7.16. `ROLLUP` – primul exemplu

Așa încât prin două secvențe `CASE` ameliorăm “afișajul”, aducându-l la standardul figurii 6.52:

```
SELECT
  CASE
    WHEN compart IS NULL THEN RPAD(CHR(123), 8, '=')
    ELSE compart
  END AS compart,
  CASE
    WHEN compart IS NULL THEN RPAD(CHR(123) ||
      ' TOTAL General ', 32, '=')
    WHEN numepren IS NULL THEN RPAD(CHR(123) ||
      ' Subtotal - compart ', 32, '- ')
    ELSE numepren
  END AS numepren,
  SUM(orelucrate * salorar) AS Venit_Baza
FROM personal pe LEFT OUTER JOIN pontaje po
  ON pe.marca=po.marca AND TO_CHAR(data, 'MM/YYYY')='07/2003'
GROUP BY ROLLUP(compart, numepren)
ORDER BY compart, numepren
```

7.5.2. Clauzele `ROLLUP` și `GROUPING`

Funcția `GROUPING` poate fi inclusă în clauza `SELECT`, argumentul acestuia fiind coloana de grupare. Rezultatul întors este 1 atunci când coloana respectivă este inclusă într-un grup de agregare superior, sau 0 pentru liniile “normale” (din afara subtotalurilor). Pentru a complica un pic lucrurile, dorim ca venitul de bază să fie calculat pe angajați, compartimente, luni calendaristice și pe total firmă.

LUNA	COMPART	NUMEPREN	VENIT_BAZA
7	CONTA	Angajat 2	2300000
7	CONTA	Angajat 6	5148000
7	CONTA	Angajat 7	2460000
7	CONTA	{ subtotal - compart	CONTA----- 9908000
7	IT	Angajat 1	4032000
7	IT	Angajat 3	2700000
7	IT	Angajat 5	4500000
7	IT	Primul Angajat Nou	1332000
7	IT	{ subtotal - compart	IT----- 12564000
7	PROD	Al Doilea Angajat Nou	1212000
7	PROD	Angajat 4	5400000
7	PROD	Angajat 8	3924000
7	PROD	{ subtotal - compart	PROD----- 10536000
7	{=====	{ SUBTOTAL - LUNA 7-----	33008000
8	CONTA	Angajat 2	920000
8	CONTA	Angajat 6	2288000
8	CONTA	Angajat 7	984000
8	CONTA	{ subtotal - compart	CONTA----- 4192000
8	IT	Angajat 1	1792000
8	IT	Angajat 3	1620000
8	IT	Angajat 5	2000000
8	IT	{ subtotal - compart	IT----- 5412000
8	PROD	Angajat 4	2400000
8	PROD	Angajat 8	1744000
8	PROD	{ subtotal - compart	PROD----- 4144000
8	{=====	{ SUBTOTAL - LUNA 8-----	13748000
9	CONTA	Angajat 2	2300000
9	CONTA	Angajat 6	2860000
9	CONTA	Angajat 7	2460000
9	CONTA	{ subtotal - compart	CONTA----- 7620000
9	IT	Angajat 1	2240000
9	IT	Angajat 3	2700000
9	IT	Angajat 5	2500000
9	IT	{ subtotal - compart	IT----- 7440000
9	PROD	Angajat 4	3000000
9	PROD	Angajat 8	2180000
9	PROD	{ subtotal - compart	PROD----- 5180000
9	{=====	{ SUBTOTAL - LUNA 9-----	20240000
{===	{=====	{ TOTAL General =====	66996000

Figura 7.17. Trei niveluri de (sub)totalizare

Testarea NULLității atributelor de grupare este înlocuită cu verificarea valorii returnate de funcția GROUPING:

```

SELECT
CASE
WHEN GROUPING (EXTRACT (MONTH FROM data)) = 1
THEN RPAD(CHR(123), 4, '=')
ELSE CAST (EXTRACT (MONTH FROM data) AS CHAR(4))
END AS Luna,
CASE
WHEN GROUPING(compart) =1
THEN RPAD(CHR(123), 8, '=')
ELSE compart
END AS compart,
CASE
WHEN GROUPING (EXTRACT (MONTH FROM data)) = 1
THEN RPAD(CHR(123) || ' TOTAL General ', 37, '=')

```

```

    WHEN GROUPING (compart) = 1
    THEN RPAD(CHR(123) || ' SUBTOTAL - LUNA ' ||
      EXTRACT (MONTH FROM data),37,'-')
    WHEN GROUPING (numepren) = 1
    THEN RPAD(CHR(123) || ' subtotal - compart ' ||
      compart,37,'-')
    ELSE numepren
  END AS numepren,
  SUM(orelucrate * salorar) AS Venit_Baza
FROM personal pe LEFT OUTER JOIN pontaje po
  ON pe.marca=po.marca
GROUP BY ROLLUP(EXTRACT (MONTH FROM data), compart, numepren)
ORDER BY EXTRACT (MONTH FROM data), compart, numepren

```

Din “scanarea” figurii 7.17 deducem că lucrurile par a fi în regulă.

7.5.3. Analize multidimensionale: clauzele CUBE și GROUPING SETS

Avansăm în zona ce face deliciul analiștilor economici, de marketing, financiari etc., aceștia fiind interesați ca datele să fie agregate simultan după doi, trei s.a.m.d. parametri, scop pentru care a fost proiectat operatorul CUBE:

```

SELECT
  EXTRACT (MONTH FROM data) AS Luna,
  CASE GROUPING (numepren)
  WHEN 1
  THEN
    CASE GROUPING (EXTRACT (MONTH FROM data))
    WHEN 1
    THEN CHR(123) ||
      'T O T A L   G E N E R A L'
    ELSE CHR(123) || '--- Subtotal luna --- '
      || EXTRACT (MONTH FROM data)
    END
  ELSE
    CASE GROUPING (EXTRACT (MONTH FROM data))
    WHEN 1
    THEN CHR(123) || ' * Subtotal * ' ||
      numepren
    ELSE numepren
    END
  END AS Numepren,
  SUM(orelucrate * salorar) AS Venit_Baza
FROM personal pe LEFT OUTER JOIN pontaje po
  ON pe.marca=po.marca
GROUP BY CUBE (EXTRACT (MONTH FROM data), numepren)
ORDER BY EXTRACT (MONTH FROM data), numepren

```

Prin folosirea operatorului CUBE (EXTRACT (MONTH FROM data), numepren) datele "curente" (veniturile brute) vor fi afișate pentru fiecare combinație angajat – lună, iar subtotalurile vor fi calculate astfel:

- pentru fiecare lună și toți angajații;
- pentru fiecare angajat și toate lunile;
- pentru toți angajații și toate lunile (totalul general) – vezi figura 7.18.

LUNA	NUMEPREN	VENIT_BAZA
7	Al Doilea Angajat Nou	1212000
7	Angajat 1	4032000
7	Angajat 2	2300000
7	Angajat 3	2700000
7	Angajat 4	5400000
7	Angajat 5	4500000
7	Angajat 6	5148000
7	Angajat 7	2460000
7	Angajat 8	3924000
7	Primul Angajat Nou	1332000
7	{--- Subtotal luna --- 7	33008000
8	Angajat 1	1792000
8	Angajat 2	920000
8	Angajat 3	1620000
8	Angajat 4	2400000
8	Angajat 5	2000000
8	Angajat 6	2288000
8	Angajat 7	984000
8	Angajat 8	1744000
8	{--- Subtotal luna --- 8	13748000
9	Angajat 1	2240000
9	Angajat 2	2300000
9	Angajat 3	2700000
9	Angajat 4	3000000
9	Angajat 5	2500000
9	Angajat 6	2860000
9	Angajat 7	2460000
9	Angajat 8	2180000
9	{--- Subtotal luna --- 9	20240000
	{ * Subtotal * Al Doilea Angajat Nou	1212000
	{ * Subtotal * Angajat 1	8064000
	{ * Subtotal * Angajat 2	5520000
	{ * Subtotal * Angajat 3	7020000
	{ * Subtotal * Angajat 4	10800000
	{ * Subtotal * Angajat 5	9000000
	{ * Subtotal * Angajat 6	10296000
	{ * Subtotal * Angajat 7	5904000
	{ * Subtotal * Angajat 8	7848000
	{ * Subtotal * Primul Angajat Nou	1332000
	{ T O T A L G E N E R A L	66996000

Figura 7.18. Un CUBE

Clauza GROUPING se poate folosi după aceeași logică expusă la exemplul dedicat operatorului ROLLUP. Pentru a păstra o dimensiune rezonabilă a rezultatului (40 de linii) interogarea folosește doar două atribute în clauza GROUP BY CUBE. Dacă însă am fi dorit o analiză mai fină, după trei sau mai multe criterii, complexitatea rezultatului ar fi crescut simțitor. Atunci când se dorește excluderea

rândurilor “de detaliu” din rezultat, sau unora dintre subtotaluri, se poate folosi clauza GROUPING SETS:

```
SELECT
  EXTRACT (MONTH FROM data) AS Luna,
  CASE GROUPING (numepren)
  WHEN 1
    THEN
      CASE GROUPING (EXTRACT (MONTH FROM data))
      WHEN 1
        THEN CHR(123) ||
              'T O T A L   G E N E R A L'
        ELSE CHR(123) || '--- Subtotal luna --- '
              || EXTRACT (MONTH FROM data)
      END
    ELSE
      CASE GROUPING (EXTRACT (MONTH FROM data))
      WHEN 1
        THEN CHR(123) || ' * Subtotal * ' ||
              numepren
        ELSE numepren
      END
    END AS Numepren,
  SUM(orelucrate * salorar) AS Venit_Baza
FROM personal pe LEFT OUTER JOIN pontaje po
  ON pe.marca=po.marca
GROUP BY GROUPING SETS (EXTRACT (MONTH FROM data), numepren )
ORDER BY EXTRACT (MONTH FROM data), numepren
```

După cum se observă în figura 7.19, rezultatul conține numai 13 linii, trei pentru că pontajele au fost introduse pentru iulie, august și septembrie 2003, plus 10, câte una pentru fiecare angajat. Dacă am fi dorit să obținem și o a 14-a linie dedicată totalului general, ar fi trebuit să modificăm argumentele clauzei GROUPING SETS astfel:

```
SELECT
  ....
GROUP BY GROUPING SETS (EXTRACT (MONTH FROM data),
  numepren, ( ) )
ORDER BY...
```

LUNA	NUMEPREN	VENIT_BAZA
7	{--- Subtotal luna --- 7	33000000
8	{--- Subtotal luna --- 8	13748000
9	{--- Subtotal luna --- 9	20240000
	{ * Subtotal * Al Doilea Angajat Nou	1212000
	{ * Subtotal * Angajat 1	8064000
	{ * Subtotal * Angajat 2	5520000
	{ * Subtotal * Angajat 3	7020000
	{ * Subtotal * Angajat 4	10800000
	{ * Subtotal * Angajat 5	9000000
	{ * Subtotal * Angajat 6	10296000
	{ * Subtotal * Angajat 7	5904000
	{ * Subtotal * Angajat 8	7848000
	{ * Subtotal * Primul Angajat Nou	1332000

Figura 7.19. Clauza GROUPING SETS

7.5.4. Clasamente: RANK și DENSE_RANK

Numai în acest capitol au fost formulate cel puțin trei soluții pentru extragerea primelor două, trei s.a.m.d. valori dintr-un clasament. Să ne oprim momentan asupra topului salariilor orare. Dar, înainte de toate, să presupunem că Angajat 5 primește o mărire de salariu orar cifrată la 5000 de lei, astfel încât ajunge la nivelul Angajatului 3:

```
UPDATE personal
SET salorar = 67500
WHERE numepren = 'Angajat 5' ;
COMMIT ;
```

La modul cel mai simplu, clasamentul salariilor orare se obține apelând la o subconsultare în clauza FROM și pseudo-coloana ROWNUM:

```
SELECT ROWNUM AS Pozitie, numepren, salorar
FROM
  (SELECT *
   FROM personal
   ORDER BY salorar DESC
  )
```

Locurile 3-4 sunt pe poziție de egalitate, însă varianta SQL prezentată nu "sesizează" acest lucru. Mult mai indicate pentru acest gen de probleme sunt două funcții OLAP, RANK și DENSE_RANK. Ca pentru orice funcții analitice, procesarea interogărilor cu RANK și DENSE_RANK se derulează în trei etape.

Mai întâi, se operează joncțiunile, se constituie grupurile și se efectuează selecția asupra grupurilor. Apoi se aranjează rezultatul în vederea aplicării funcțiilor analitice și se efectuează calculele (sunt create partițiile), iar funcțiile analitice sunt aplicate linie cu linie în fiecare partiție. Pasul 3 este operațional

numai dacă interogarea prezintă la sfârșit o clauză `ORDER BY`, ceea ce atrage ordonarea finală a rezultatului în conformitate cu criteriile specificate. *Partițiile* reprezintă seturi de linii create după delimitarea grupurilor prin `GROUP BY`, astfel încât pot constitui subiectul oricărei funcții de agregare (`SUM`, `AVG`,...). Constituirea unei partiții se realizează în funcție de valorile unuia sau mai multor atribute sau expresii de atribute.

POZITIE	NUMEPREN	SALORAR
1	Angajat 4	75000
2	Angajat 6	71500
3	Angajat 3	67500
4	Angajat 5	67500
5	Angajat 7	61500
6	Angajat 2	57500
7	Angajat 1	56000
8	Primul Angajat Nou	55500
9	Angajat 8	54500
10	Al Doilea Angajat Nou	50500

Figura 7.20. Clasamentul salariilor orare

Interogarea următoare pune în evidență modul de lucru al celor două funcții – vezi figura 7.21. Valorile egale au același ordin în ambele variante, însă, în timp ce `RANK` atribuie în continuare pozițiile luând în calcul numărul valorilor egale, `DENSE_RANK` nu.

```
SELECT numepren, salorar,
       RANK() OVER (ORDER BY salorar DESC) AS Pozitie_RANK,
       DENSE_RANK() OVER (ORDER BY salorar DESC)
       AS Pozitie_DENSE_RANK
FROM personal
```

NUMEPREN	SALORAR	POZITIE_RANK	POZITIE_DENSE_RANK
Angajat 4	75000	1	1
Angajat 6	71500	2	2
Angajat 3	67500	3	3
Angajat 5	67500	3	3
Angajat 7	61500	5	4
Angajat 2	57500	6	5
Angajat 1	56000	7	6
Primul Angajat Nou	55500	8	7
Angajat 8	54500	9	8
Al Doilea Angajat Nou	50500	10	9

Figura 7.21. Funcțiile `RANK` și `DENSE_RANK`

Care sunt cele mai mari trei salarii orare și care sunt fericiții angajați ?
 Funcțiile analitice nu pot fi incluse în clauze `WHERE` – vezi figura 7.22.

```

SQL> SELECT numepren, salorar,
2  RANK() OVER (ORDER BY salorar DESC) AS Pozitie
3  FROM personal
4  WHERE RANK() OVER (ORDER BY salorar DESC) <= 3
5  /
WHERE RANK() OVER (ORDER BY salorar DESC) <= 3
*
ERROR at line 4:
ORA-30483: window functions are not allowed here

```

Figura 7.22. Clauza WHERE nu poate conține funcție analitică

Lucrurile se rezolvă prin includerea SELECT-ului ce conține funcția analitică într-o subconsultare și aplicarea predicatului de selecție în fraza principală.

```

SELECT *
FROM
  (SELECT numepren, salorar,
    RANK() OVER (ORDER BY salorar DESC) AS Pozitie
  FROM personal)
WHERE pozitie <= 3

```

Răspunsul ar fi fost identic dacă am fi folosit DENSE_RANK.

Care sunt primii cinci clasați în topul angajaților cu cele mai mari venituri ?

De data aceasta, funcția RANK se folosește în condițiile joncțiunii PERSONAL-PONTAJE și grupării după nume, criteriul de ordonare fiind specificat prin funcția SUM:

```

SELECT *
FROM
  (SELECT numepren, SUM (orelucrate * salorar +
    oreco * salorarco) AS Venit_Baza,
    RANK() OVER (ORDER BY SUM (orelucrate * salorar +
    oreco * salorarco) DESC) AS Pozitie
  FROM personal INNER JOIN pontaje
    ON personal.marca = pontaje.marca
  GROUP BY numepren)
WHERE pozitie <= 5

```

NUMEPREN	VENIT_BAZA	POZITIE
Angajat 4	10800000	1
Angajat 6	10296000	2
Angajat 5	9720000	3
Angajat 3	9660000	4
Angajat 7	8784000	5

Figura 7.23. Funcția RANK pentru aflare celor mai bine plătiți cinci angajați

Să se afișeze, pentru fiecare compartiment, primii doi plasați în topul veniturilor.

De data aceasta, topul trebuie întocmit diferențiat pe fiecare compartiment, ceea ce înseamnă că este necesară partiționarea după atributul `Compart` (vezi figura 7.24):

```
SELECT *
FROM
  (SELECT compart, numepren, SUM (orelucrate * salorar +
    oreco * salorarco) AS Venit_Baza,
    RANK() OVER (PARTITION BY compart
      ORDER BY SUM (orelucrate * salorar +
        oreco * salorarco) DESC) AS Pozitie
  FROM personal INNER JOIN pontaje
    ON personal.marca = pontaje.marca
  GROUP BY compart, numepren)
WHERE pozitie <= 2
```

COMPA	NUMEPREN	VENIT_BAZA	POZITIE
CONTA	Angajat 6	10296000	1
CONTA	Angajat 7	8784000	2
IT	Angajat 5	9720000	1
IT	Angajat 3	9660000	2
PROD	Angajat 4	10800000	1
PROD	Angajat 8	7848000	2

Figura 7.24. Clasament pe partiții

7.5.5. Ferestre pentru funcții analitice

Fereastra se definește în cadrul unei partiții și se referă la intervalul liniilor luat în calcule pentru *linia curentă*. Mărimea ferestrei se poate specifica fie *fizic*, printr-un număr de înregistrări, fie *logic*, printr-un interval de tip dată calendaristică/timp sau interval de valori. Calculele se efectuează pentru fiecare linie din cadrul ferestrei, fereastră mișcătoare între o poziție de start și una de final. Rândul curent servește ca punct de referință pentru determinarea începutului și sfârșitului ferestrei.

Specificațiile unei ferestre privesc trei componente: partiționarea, ordonarea și grupurile de agregare. Orice funcție de agregare poate fi utilizată în cadrul unei ferestre: SUM, AVG, MIN, MAX, STDDEV, VARIANCE, COUNT. Pe lângă acestea, Oracle, de la versiunea 8i2, mai oferă suport pentru alte câteva funcții statistice: VAR_SAMP, VAR_POP, STDDEV_SAMP s.a.m.d.. Valorile agregate pot fi cumulative, mișcătoare sau centrate.

Să se afișeze, diferențiat pe luni calendaristice, valoarea cumulată, după fiecare angajat, a veniturilor de bază câștigate de angajații firmei.

Partiționarea se realizează după luni, numărul de linii al fiecărei partiții fiind egal cu numărul angajaților cu minim un pontaj pe luna respectivă. Ordonarea liniilor în partiție se realizează după nume și prenume, iar fereastra pentru care se face

calculul cuprinde toate liniile de la primul angajat până la angajatul curent din luna-partiție:

```
SELECT luna, numepren, Venit_Baza_Angajat_Luna
  AS Venit_Angaj_Crt,
  SUM(Venit_Baza_Angajat_Luna) OVER (PARTITION BY luna
    ORDER BY numepren ROWS UNBOUNDED PRECEDING)
  AS Venit_Luna_Cumulat
FROM
  (SELECT EXTRACT (MONTH FROM data) AS luna, numepren,
    SUM (orelucrate * salorar + oreco * salorarco)
    AS Venit_Baza_Angajat_Luna
  FROM personal INNER JOIN pontaje
    ON personal.marca = pontaje.marca
  GROUP BY EXTRACT (MONTH FROM data), numepren
  )
```

Pentru comparație, în rezultat (figura 7.25) au fost incluse coloanele veniturilor lunare ale fiecărui angajat (Venit_Angaj_Crt), precum și cea calculată prin fereastră/partiționare (Venit_Luna_Cumulat) care se reșetează după ultima linie (angajat) corespunzătoare unei luni calendaristice.

LUNA	NUMEPREN	VENIT_ANGAJ_CRT	VENIT_LUNA_CUMULAT
7	Al Doilea Angajat Nou	1212000	1212000
7	Angajat 1	4032000	5244000
7	Angajat 2	4092000	9336000
7	Angajat 3	4812000	14148000
7	Angajat 4	5400000	19548000
7	Angajat 5	4860000	24408000
7	Angajat 6	5148000	29556000
7	Angajat 7	4380000	33936000
7	Angajat 8	3924000	37860000
7	Primul Angajat Nou	1332000	39192000
8	Angajat 1	1792000	1792000
8	Angajat 2	1816000	3608000
8	Angajat 3	2148000	5756000
8	Angajat 4	2400000	8156000
8	Angajat 5	2160000	10316000
8	Angajat 6	2288000	12604000
8	Angajat 7	1944000	14548000
8	Angajat 8	1744000	16292000
9	Angajat 1	2240000	2240000
9	Angajat 2	2300000	4540000
9	Angajat 3	2700000	7240000
9	Angajat 4	3000000	10240000
9	Angajat 5	2700000	12940000
9	Angajat 6	2860000	15800000
9	Angajat 7	2460000	18260000
9	Angajat 8	2180000	20440000

Figura 7.25. Venituri lunare ale angajaților, curente și cumulate

7.5.6. Comparații și ponderi

Și în această categorie sunt înscrise funcții cu un larg evantai de facilități, în timp ce noi vom rezuma discuția la doar câteva exemple.

Care este evoluția zilnică a prezenței în iulie 2003, prin raportare la ziua lucrătoare anterioară ?

Problema a mai fost rezolvată în acest capitol, însă acum vom apela la funcțiile LAG prin care extragerea valorii unei coloane de pe linia precedentă devine o formalitate:

```
SELECT data,
       Prezent AS Prezent_Zi_Curenta,
       NVL(LAG (Prezent, 1) OVER (ORDER BY data),0)
         AS Prezent_Zi_Anterioara,
       Prezent - NVL(LAG (Prezent, 1) OVER (ORDER BY data),0)
         AS Diferenta
FROM
  (SELECT data, COUNT(*) AS Prezent
   FROM pontaje WHERE orelucrate > 0
   AND TO_CHAR(data, 'MM/YYYY')='07/2003'
   GROUP BY data)
```

Liniile rezultatului seamănă izbitor cu cele din figura 7.12.

Atunci când, într-o partiție, se dorește raportarea valorilor liniei curente la cele din prima sau ultima linie, se pot folosi funcțiile FIRST_VALUE și LAST_VALUE.

Să se calculeze, pentru fiecare angajat, raportul dintre venitul său de bază și venitul celui mai mic venit de bază din firmă, precum și raportul față de cel mai mare venit de bază din firmă.

Am formulat astfel problema pentru a putea folosi ambele funcții iar, pe de altă parte, am definit diferit ferestrele necesare determinării venitorilor minime și maxime:

```
SELECT numepren, venit_baza,
       FIRST_VALUE (venit_baza) OVER (ORDER BY venit_baza ASC)
         AS Venit_Minim,
       LAST_VALUE (venit_baza) OVER
         (ORDER BY venit_baza ROWS BETWEEN UNBOUNDED
          PRECEDING AND UNBOUNDED FOLLOWING)
         AS Venit_Maxim,
       ROUND( venit_baza / FIRST_VALUE (venit_baza) OVER
         (ORDER BY venit_baza ASC),2)
         AS Raport_V_Min,
       ROUND(venit_baza / LAST_VALUE (venit_baza) OVER
         (ORDER BY venit_baza ROWS BETWEEN UNBOUNDED
          PRECEDING AND UNBOUNDED FOLLOWING),2)
```

```

AS Raport_V_Max
FROM
  (SELECT numepren, SUM (orelucrate * salorar +
    oreco * salorarco) AS Venit_Baza
  FROM pontaje po INNER JOIN personal pe
    ON po.marca=pe.marca
  GROUP BY numepren)
ORDER BY numepren

```

NUMEPREN	VENIT_BAZA	VENIT_MINIM	VENIT_MAXIM	RAPORT_U_MIN	RAPORT_U_MAX
Al Doilea Angajat Nou	1212000	1212000	10000000	1	.11
Angajat 1	8064000	1212000	10000000	6.65	.75
Angajat 2	8208000	1212000	10000000	6.77	.76
Angajat 3	9660000	1212000	10000000	7.97	.89
Angajat 4	10000000	1212000	10000000	8.91	1
Angajat 5	9720000	1212000	10000000	8.02	.9
Angajat 6	10296000	1212000	10000000	8.5	.95
Angajat 7	8784000	1212000	10000000	7.25	.81
Angajat 8	7848000	1212000	10000000	6.48	.73
Primul Angajat Nou	1332000	1212000	10000000	1.1	.12

Figura 7.26. Folosirea funcțiilor FIRST_VALUE și LAST_VALUE

Care este cel mai bine plătit angajat din fiecare compartiment ?

Funcția FIRST_VALUE se plasează într-o subconsultare ce extrage, pentru fiecare angajat, venitul său de bază și cel mai mare venit de bază din compartimentul din care face parte. Fraza SELECT principală extrage numai liniile în care cele două venituri sunt egale:

```

SELECT compart, numepren, venit_baza
FROM
  (SELECT compart, numepren, venit_baza,
    FIRST_VALUE (venit_baza) OVER
      (PARTITION BY compart ORDER BY venit_baza DESC)
    AS Venit_Maxim
  FROM
    (SELECT compart, numepren,
      SUM (orelucrate * salorar +
        oreco * salorarco) AS Venit_Baza
    FROM pontaje po INNER JOIN personal pe
      ON po.marca=pe.marca
    GROUP BY compart, numepren)
  )
WHERE venit_baza = venit_maxim

```

Care este angajatul cu numărul de zile lucrate imediat peste cel al Angajatului 7 ?

Ideea problemei anterioare stă la baza unei noi soluții pentru acest enunț ceva mai vechi:

```

SELECT numepren, Zile_L
FROM

```



```

(SELECT numepren, Zile_L,
  FIRST_VALUE (Zile_L) OVER (ORDER BY Zile_L ASC)
  AS Zile_OK
FROM
  (SELECT numepren, COUNT(*) AS Zile_L
   FROM pontaje po INNER JOIN personal pe
     ON po.marca=pe.marca
   WHERE orelucrate > 0 AND numepren <> 'Angajat 7'
   GROUP BY numepren
   HAVING COUNT(*) >
     (SELECT COUNT(data) AS Zile_A7
      FROM pontaje po INNER JOIN personal pe
        ON po.marca=pe.marca
      WHERE orelucrate > 0 AND numepren = 'Angajat 7')
   )
)
WHERE Zile_L = Zile_OK

```

Să se calculeze partea fiecărui angajat din totalul veniturilor de bază.

Nu că am fi nemulțumiți de aportul subconsultărilor scalare la rezolvarea situațiilor de acest gen, dar merită un pic de atenție și funcția `RATIO_TO_REPORT` ce calculează raportul dintre o valoare (atribut/expresie) și suma totală a valorii respective pentru toate liniile din fereastră:

```

SELECT numepren, venit, ROUND (RATIO_TO_REPORT (venit)
  OVER () * 100,2) AS Pondere
FROM
  (SELECT numepren, SUM (orelucrate * salorar +
    oreco * salorarco) AS Venit
   FROM pontaje po INNER JOIN personal pe
     ON po.marca=pe.marca
   WHERE pe.marca = po.marca
   GROUP BY numepren
  )
ORDER BY numepren

```

7.6. Interogări ierarhice

Problema structura ierarhice, sau a arborilor, este una care a suscitat discuții cel puțin interesante în comunitatea SQL. Din acest punct de vedere, Oracle s-a plasat lejer, de ani buni, în fața concurenței. Cum aproape întotdeauna când spunem ierarhie ne referim la organigramă, să analizăm structura personalului firmei așa cum se prezintă în figura 7.27.

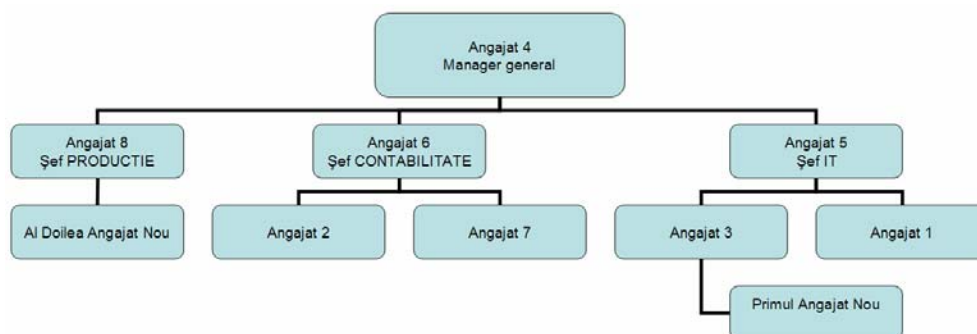


Figura 7.27. Organigrama firmei

Nu recomandăm nici unei firme să-și organizeze de această manieră personalul, însă pentru binele exemplurilor am diminuat din pretenții. Cât privește structura bazei de date, pentru reflectarea subordonării angajaților se poate adăuga în tabela PERSONAL atributul *Marca_Șef* pentru care s-ar declara o restricție referențială, atributul părinte fiind *Marcă* din aceeași tabelă.

Noi însă vom crea și popula o tabelă specială pe care o vom denumi IERARHIE – vezi listing 7.1.

Listing 7.1. Crearea și popularea tabelii IERARHIE

```

DROP TABLE ierarhie ;

CREATE TABLE ierarhie (
    marca INTEGER
    CONSTRAINT nn_ierarhie_marca NOT NULL
    CONSTRAINT pk_ierarhie PRIMARY KEY
    CONSTRAINT fk_ierarhie_personal REFERENCES personal (marca)
    , marca_sef INTEGER
    CONSTRAINT fk_ierarhie_personal2 REFERENCES personal (marca)
);

INSERT INTO ierarhie VALUES (104, NULL) ;
INSERT INTO ierarhie VALUES (108, 104) ;
INSERT INTO ierarhie VALUES (106, 104) ;
INSERT INTO ierarhie VALUES (105, 104) ;
INSERT INTO ierarhie VALUES (110, 108) ;
INSERT INTO ierarhie VALUES (102, 106) ;
INSERT INTO ierarhie VALUES (107, 106) ;
INSERT INTO ierarhie VALUES (103, 105) ;
INSERT INTO ierarhie VALUES (101, 105) ;
INSERT INTO ierarhie VALUES (109, 103) ;
COMMIT ;

```

Să luăm în discuție câteva probleme.

Care este nivelul ierarhic al fiecărui salariat ?

Din parcurgerea figurii 7.27 deducem că sunt patru nivele ierarhice: primul este cel al managerului general (Angajat 4), al doilea al șefilor de compartimente (Angajat 8 – Producție, Angajat 6 – Contabilitate, Angajat 5 – IT), al treilea pentru pălmași

(Al Doilea Angajat Nou, Angajații 2, 7 și 1) și un șef mai mititel (Angajat 3), iar pe ultimul nivel, al patrulea, apare un singur om al muncii (probabil pentru a echilibra singurătatea managerului general) – Primul Angajat Nou. Altfel spus, ne interesează o situație asemănătoare celei din figura 7.28.

MARCA	NUMEPREN	COMPA	NIVEL
104	Angajat 4	PROD	1
106	Angajat 6	CONTA	2
105	Angajat 5	IT	2
108	Angajat 8	PROD	2
102	Angajat 2	CONTA	3
107	Angajat 7	CONTA	3
101	Angajat 1	IT	3
103	Angajat 3	IT	3
110	Al Doilea Angajat Nou	PROD	3
109	Primul Angajat Nou	IT	4

Figura 7.28. Nivelul ierarhic al fiecărui angajat

Persoanele de pe fiecare nivel vor fi furnizate de un `SELECT` distinct, așa că, în final, vom reuni consultările celor patru niveluri:

```

SELECT p.marca, numepren, compart, '1' AS Nivel
FROM personal p INNER JOIN ierarhie i ON p.marca = i.marca
WHERE marca_sef IS NULL
UNION
SELECT marca, numepren, compart, '2' AS Nivel
FROM personal
WHERE marca IN
    (SELECT marca
     FROM ierarhie
     WHERE marca_sef IN
        (SELECT marca
         FROM ierarhie
         WHERE marca_sef IS NULL
        )
    )
UNION
SELECT marca, numepren, compart, '3' AS Nivel
FROM personal
WHERE marca IN
    (SELECT marca
     FROM ierarhie
     WHERE marca_sef IN
        (SELECT marca
         FROM ierarhie
         WHERE marca_sef IN
            (SELECT marca
             FROM ierarhie
             WHERE marca_sef IS NULL
            )
        )
    )

```

```

        )
    )
UNION
    SELECT marca, numepren, compart, '4' AS Nivel
    FROM personal
    WHERE marca IN
        (SELECT marca
         FROM ierarhie
         WHERE marca_sef IN
            (SELECT marca
             FROM ierarhie
             WHERE marca_sef IN
                (SELECT marca
                 FROM ierarhie
                 WHERE marca_sef IS NULL
                )
            )
        )
    )
ORDER BY 4, 3, 2

```

Este greu de reprimat senzația de discomfort provocată de întinderea interogării. Ca să nu ne mai stricăm ziua gândindu-ne ce interesant ar fi fost ca firma să aibă vreo șapte-opt niveluri ierarhice.

Prin comparație, soluția următoare este mai mult decât binevenită, având același rezultat:

```

SELECT marca, numepren, compart, LEVEL AS Nivel
FROM
    (SELECT p.marca, numepren, compart, marca_sef
     FROM personal p INNER JOIN ierarhie i
     ON p.marca = i.marca)
START WITH marca_sef IS NULL
CONNECT BY PRIOR marca = marca_sef
ORDER BY 4,3,2

```

Construirea structurii ierarhice începe cu înregistrarea (înregistrările) care îndeplinesc condiția din clauza START WITH. Această înregistrare (înregistări) părinte va fi legată de înregistrarea sau înregistrările copil prin condiția Marca = Marca_Şef. Clauza PRIOR plasată în stânga condiției semnifică: valoarea atributului Marca din părinte trebuie să fie egală cu valoarea Marca_Şef din înregistrările copil. Prin CONNECT BY sunt selectate toate generațiile succesive de linii-copil (copii, nepoți, strănepoți etc.). 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. În lipsa clauzei de ordonare, înregistrările sunt dispuse în funcție de ordinea parcurgerii arborelui.

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 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. Alt atu al interogărilor ierarhice ține de faptul că nu există o limită în privința nivelelor de subordonare.

Cum se numește șeful Angajatului 7 ?

Începem cu o variantă clasică:

```
SELECT numepren
FROM   personal
WHERE  marca IN
      (SELECT marca_sef
       FROM   personal p INNER JOIN ierarhie i
            ON p.marca = i.marca
       WHERE  numepren = 'Angajat 7')
```

Pentru varianta bazată pe interogări ierarhice, schimbăm ordinea atributelor din `START WITH` astfel încât piramida se construiește “înspre” sus, începând cu Angajat 7. Interogarea:

```
SELECT marca, numepren, compart, LEVEL AS Nivel
FROM
      (SELECT p.marca, numepren, compart, marca_sef
       FROM   personal p INNER JOIN ierarhie i
            ON p.marca = i.marca)
START WITH numepren = 'Angajat 7'
CONNECT BY PRIOR marca_sef = marca
```

obține rezultatul din figura 7.29, deci toți șefii angajatului reper (plus el-însuși).

MARCA	NUMEPREN	COMPA	NIVEL
107	Angajat 7	CONTA	1
106	Angajat 6	CONTA	2
104	Angajat 4	PROD	3

Figura 7.29. Șefii Angajatului 7

Șeful imediat al Angajatului 7 este cel de nivel 2 din figura de mai sus:

```
SELECT *
FROM
      (SELECT marca, numepren, compart, LEVEL AS Nivel
       FROM
```

```

        (SELECT p.marca, numepren, compart, marca_sef
        FROM personal p INNER JOIN ierarhie i
        ON p.marca = i.marca)
    START WITH numepren = 'Angajat 7'
    CONNECT BY PRIOR marca_sef = marca
    )
WHERE nivel = 2

```

Care sunt subordonații direcți ai Angajatului 5 ?

Piramida se construiește de la Angajat 2 în jos, plasând în stânga semnului egal al
CONNECT BY atributul Marca:

```

SELECT *
FROM
    (SELECT marca, numepren, compart, LEVEL AS Nivel
    FROM
        (SELECT p.marca, numepren, compart, marca_sef
        FROM personal p INNER JOIN ierarhie i
        ON p.marca = i.marca)
    START WITH numepren = 'Angajat 5'
    CONNECT BY PRIOR marca = marca_sef
    )

```

Rezultatul din figura 7.30 arată atât numele și nivelul ierarhic inferior al
fiecăruia dintre subordonați.

MARCA	NUMEPREN	COMPA	NIVEL
105	Angajat 5	IT	1
103	Angajat 3	IT	2
109	Primul Angajat Nou	IT	3
101	Angajat 1	IT	2

Figura 7.30. Subordonații Angajatului 5

Care sunt subordonații subordonaților directorului general ?

Practic, liniile care interesează sunt cele “nepot” ale înregistrării-rădăcină, cea
pentru care Marca_Sef IS NULL. Dacă ținem cont că directorul general este pe
primul nivel ierarhic, atunci nepoții săi sunt plasați pe nivelul 3:

```

SELECT *
FROM
    (SELECT marca, numepren, compart, LEVEL AS Nivel
    FROM
        (SELECT p.marca, numepren, compart, marca_sef
        FROM personal p INNER JOIN ierarhie i
        ON p.marca = i.marca)
    START WITH marca_sef IS NULL
    CONNECT BY PRIOR marca = marca_sef
    )

```

```
)
WHERE nivel = 3
```

Să se afișeze structura ierarhică a firmei.

Pentru un plus de vizibilitate vom afișa numele angajaților cu indentare la stânga; după fiecare angajat urmează mediat subordonații săi și subordonații subordonaților – vezi figura 7.31. Fiecărui nivel ierarhic îi va corespunde un nivel de indentare.

NUMEPREN	COMPA	NIVEL
Angajat 4	PROD	1
---- Angajat 8	PROD	2
----- Al Doilea Angajat Nou	PROD	3
---- Angajat 6	CONTA	2
----- Angajat 2	CONTA	3
----- Angajat 7	CONTA	3
---- Angajat 5	IT	2
----- Angajat 3	IT	3
----- Primul Angajat Nou	IT	4
----- Angajat 1	IT	3

Figura 7.31. Model de afișare a structurii ierarhice

Funcția folosită pentru indentare este LPAD. În rest, lucruri cunoscute:

```
SELECT LPAD(' ', 5 * (LEVEL - 1), '-') || numepren
       AS numepren, compart, LEVEL AS Nivel
FROM
  (SELECT p.marca, numepren, compart, marca_sef
   FROM personal p INNER JOIN ierarhie i
    ON p.marca = i.marca)
START WITH marca_sef IS NULL
CONNECT BY PRIOR marca = marca_sef
```