10. Funcții SQL specifice pentru raportare

Cuprins

10. Funcții SQL specifice pentru raportare

10.1. Ferestre pentru agregări

- Una dintre cele mai importante noțiuni introduse de standardul *OLAP* al *SQL99* este fereastra.
  - Fereastra se definește în cadrul unei partiții și se referă la intervalul liniilor luat în calcul relativ la linia curentă.
  - Mărimea ferestrei se poate specifica:
    - fizic, printr-un număr de linii;
    - logic, printr-un interval de tip dată calendaristică sau interval de valori.
  - Calculele se efectuează pentru fiecare linie din cadrul ferestrei care este determinată de o poziție de start și una de final. Linia curentă reprezintă punctul de referință pentru determinarea începutului și sfârșitului ferestrei.
- Specificațiile unei ferestre privesc trei componente:
  - partiționarea;
  - ordonarea;
  - grupurile de agregare.
- Funcțiile de tip fereastră pot fi utilizate pentru a calcula agregări cumulative, de evoluție și centrate.
  - Acestea întorc o valoare pentru fiecare linie din mulțime, care depinde de celelalte linii din fereastra corespunzătoare liniei respective.
  - Aceste funcții pot fi utilizate numai în clauzele *SELECT* și *ORDER BY* ale interogărilor.
  - Exemple de funcții fereastră: *SUM*, *AVG*, *MAX*, *MIN*, *COUNT*, *STDDEV*, *VARIANCE*, *FIRST_VALUE*, *LAST_VALUE* etc
  - Sintaxa funcțiilor de tip fereastră este următoarea:

```
{<FUNCȚIE_FEREASTRĂ>}
({<expr1> | *}) OVER
  ([PARTITION BY <expr2>[,...]]
    ORDER BY <expr3>
    [ASC| DESC] [NULLS FIRST | NULLS LAST] [,...]
    ROWS | RANGE
```

```
        {{UNBOUNDED PRECEDING | <expr4> PRECEDING}
        | BETWEEN
          {UNBOUNDED PRECEDING | <expr4> PRECEDING}
       AND{CURRENT ROW | <expr4> FOLLOWING}}
```

Exemplul 10.1

Presupunem că se dorește să se obțină pentru fiecare factură valoarea cumulată după fiecare produs (ascendent după codul produsului). În această situație:

- partiționarea se va realiza la nivel de factură;
- numărul de submulțimi al partiției va fi egal cu numărul de facturi;
- o submulțime a partiției va avea dimensiunea unei facturi; numărul de linii care o alcătuiesc va depinde de numărul de produse de pe factura respectivă;
- ordonarea în cadrul submulțimilor se va realiza după produs;
- fereastra pentru care se efectuează calculul pentru fiecare linie din submulțime are o margine fixă (începutul submulțimii) și una mobilă care este chiar linia curentă.

```
SELECT factura, produs_id,
       pret_unitar_vanzare*cantitate AS valoare,
       SUM(pret_unitar_vanzare*cantitate)
          OVER  (PARTITION BY factura
                 ORDER BY     produs_id
                 ROWS UNBOUNDED PRECEDING)
          AS Valoare_Cumulata
FROM  vanzari;

   FACTURA  PRODUS_ID    VALOARE VALOARE_CUMULATA
---------- ---------- ---------- ----------------
       162       3004         49               49
       162       3790        138              187
       162       5798         60              247
       163       3790         46               46
       166        556        103              103
       166        941        251              354
       166       5099        229              583
...
```

Observații:

- Pe factura 162 au fost vândute trei produse, ce au codurile 3004, 3790 și 5790.
- Pentru fiecare linie din această factură se calculează valoarea cumulată a produselor precedente plus cea a produsului curent.
- Totalul este asigurat de funcția agregat *SUM*(*pret_unitar_vanzare*cantitate*).

- Clauza *PARTITION BY factura* va limita calculul totalului la nivel de factură și, implicit, resetarea sumei la schimbarea valorii atributului *factura*.
- În cadrul fiecărei submulțimi (facturi), liniile sunt dispuse ascendent, în funcție de valoarea codului produsului (*ORDER BY id_produs*), iar mărimea ferestrei (din cadrul submulțimii) este nelimitată, cuprinzând linia curentă și toate liniile care o preced, lucru realizat prin clauza *ROWS UNBOUNDED PRECEDING*.

Exemplul 10.2

În exemplul următor se afișează și valoarea obținută prin însumarea valorii facturate aferente produsului curent și a celui precedent.

În acest caz, mărimea ferestrei (din cadrul submulțimii corespunzătoare unei facturi) este limitată, cuprinzând linia curentă și linia precedentă acesteia, lucru realizat prin clauza *ROWS 1 PRECEDING*.

```
SELECT  factura, produs_id,
        pret_unitar_vanzare*cantitate AS valoare,
        SUM(pret_unitar_vanzare*cantitate)
          OVER (PARTITION BY factura
                ORDER BY produs_id
                ROWS UNBOUNDED PRECEDING)
          AS Val_Cumulata,
        SUM(pret_unitar_vanzare*cantitate)
           OVER (PARTITION BY factura
                 ORDER BY     produs_id
                 ROWS 1 PRECEDING)
           AS Val_Curent_si_Precedent
FROM vanzari;
```

```
   FACTURA  PRODUS_ID    VALOARE VAL_CUMULATA VAL_CURENT_SI_PRECEDENT
----------------------------------------------------------------------
       170       2122         18          18                      18
       170       2436          9          27                      27
       170       2445     101.43      128.43                  110.43
       170       3505          9      137.43                  110.43
       170       5278      28.56      165.99                   37.56
       170       5650       2.52      168.51                   31.08
       170       5756      87.01      255.52                   89.53
       170       5904      17.92      273.44                  104.93
       170       6010     180.53      453.97                  198.45
...
```

- Atunci când funcțiile analitice trebuie să utilizeze informații obținute prin grupare, soluția cea mai simplă este definirea unei vizualizări *inline* cu o clauză de grupare.

Exemplul 10.3

De exemplu, ne interesează atât valoarea cumulată a vânzărilor, cât și valoarea medie a vânzărilor realizate în ultimele 2 zile (ziua curentă și ziua precedentă).

```
SELECT data, valoare,
       SUM(valoare) OVER
         (ORDER BY data ROWS UNBOUNDED PRECEDING)
          AS valoare_cumulata,
       AVG(valoare) OVER
          (ORDER BY data ROWS 1 PRECEDING)
           AS medie_2_valori
FROM  (SELECT data,
              SUM(pret_unitar_vanzare*cantitate) valoare
       FROM   vanzari v, timp t
       WHERE  v.timp_id=t.id_timp
       AND    luna = 3
       AND    an = 2007
       GROUP BY data);
```

```
TIMP_ID     VALOARE VALOARE_CUMULATA MEDIE_2_VALORI
--------- ---------- ---------------- --------------
05-MAR-07    1442.86          1442.86        1442.86
06-MAR-07        231          1673.86         836.93
12-MAR-07    2683.01          4356.87       1457.005
13-MAR-07   20989.17         25346.04       11836.09
14-MAR-07    19232.2         44578.24      20110.685
15-MAR-07    5502.92         50081.16       12367.56
16-MAR-07    7153.97         57235.13       6328.445
19-MAR-07      484.1         57719.23       3819.035
20-MAR-07   18392.99         76112.22       9438.545
...
```

Exemplul 10.4

În următorul exemplu se obține ziua în care s-a vândut cel mai bine fiecare produs din categoria „Markere permanente“, în anul 2007.

```
SELECT produs_id, data, valoare
FROM  (SELECT v.produs_id, data,
              SUM(pret_unitar_vanzare*cantitate)
                  AS valoare,
              MAX(SUM(pret_unitar_vanzare*cantitate))
```

```
                    OVER (PARTITION BY v.produs_id)
                    AS vanz_max
          FROM   vanzari v, timp t, produse p
          WHERE  v.timp_id = t.id_timp
          AND    v.produs_id = p.id_produs
          AND    categorie_5 = 'Markere permanente'
          AND    an = 2007
          GROUP BY v.produs_id, data)
WHERE  valoare=vanz_max;

PRODUS_ID  DATA      VALOARE
---------- --------- ----------
       899 14-MAR-07       55.2
      1377 14-MAR-07       29.4
      1377 20-MAR-07       29.4
      1378 22-MAR-07          9
      1482 14-MAR-07         15
      2334 16-MAR-07       55.3
      2334 21-MAR-07       55.3
```

10.1.1. Ferestre cu limite logice

- O limită logică poate fi specificată:
  - prin constante precum *RANGE n PRECEDING*;
  - printr-o expresie care poate fi evaluată ca o constantă;
  - printr-un interval specificat prin *RANGE INTERVAL n DAY/MONTH/YEAR PRECEDING* sau o expresie ce reprezintă un *INTERVAL*.

- ❖ În exemplele anterioare mărimea ferestrei a fost definită fizic, prin numărul de linii ce preced sau succed linia curentă.
- ❖ În exemplul ce urmează apare o fereastră cu limită logică, prin folosirea unui *INTERVAL* calendaristic.

Exemplul 10.5

Dorim să se afișeze, pentru fiecare zi, următoarele valori: vânzările din ziua respectivă, vânzările cumulate (inclusiv ale zilei curente) și vânzările din ziua curentă cumulate cu cele din ziua precedentă.

```
SELECT data,
       valoare,
       SUM(valoare) OVER
         (ORDER BY data ROWS UNBOUNDED PRECEDING)
```

```
          AS valoare_cumulata,
       SUM(valoare) OVER
         (ORDER BY data RANGE INTERVAL '1' DAY PRECEDING)
          AS azi_si_ieri
FROM  (SELECT data,
              SUM(pret_unitar_vanzare*cantitate) valoare
       FROM   vanzari v, timp t
       WHERE  v.timp_id=t.id_timp
       AND    luna = 3
       AND    an = 2007
       GROUP BY data);
```

```
DATA      VALOARE VALOARE_CUMULATA AZI_SI_IERI
--------- ---------- ---------------- -----------
05-MAR-07    1442.86          1442.86     1442.86
06-MAR-07        231          1673.86     1673.86
12-MAR-07    2683.01          4356.87     2683.01
13-MAR-07   20989.17         25346.04    23672.18
14-MAR-07    19232.2         44578.24    40221.37
15-MAR-07    5502.92         50081.16    24735.12
16-MAR-07    7153.97         57235.13    12656.89
19-MAR-07      484.1         57719.23       484.1
20-MAR-07   18392.99         76112.22    18877.09
...
```

- Dacă pentru intervalul definit fizic se ia în calcul linia anterioară, indiferent cărei date calendaristice îi corespunde, în cazul intervalul calendaristic, valoarea zilei curente se cumulează numai dacă linia precedentă se referă la ziua calendaristică precedentă.
- Pentru data de *19-MAR-07* valorile coloanelor *VALOARE_CUMMULATA* și *AZI_SI_IERI* coincid, deoarece pe 18 martie nu există nicio factură emisă.

10.1.2. Agregări cumulate

Exemplul 10.6

În exemplul următor se definește pentru fiecare înregistrare, o fereastră ce pornește la începutul submulțimii (*UNBOUNDED PRECEDING*) și se termină la înregistrarea curentă (setare implicită).

```
SELECT client_id, trimestru,
       SUM(pret_unitar_vanzare*cantitate) valoare,
       SUM(SUM(pret_unitar_vanzare*cantitate))
           OVER (PARTITION BY client_id
```

```
                ORDER BY     client_id, trimestru
                ROWS UNBOUNDED PRECEDING) CLIENT_VANZARI
FROM   vanzari v, timp t
WHERE  v.timp_id=t.id_timp
AND    client_id IN (44,82,84,146)
AND    an=2007
GROUP BY client_id, trimestru;

CLIENT_ID  TRIMESTRU   VALOARE CLIENT_VANZARI
--------- ---------- ---------- --------------
       44          2     224.36         224.36
       82          1     430.56         430.56
       84          2    3408.83        3408.83
       84          3     762.38        4171.21
      146          1    2200.72        2200.72
      146          2    3821.56        6022.28
      146          3     141.67        6163.95
```

De exemplu, pentru clientul 146 avem următoarea interpretare a datelor afișate:

```
CLIENT_ID  TRIMESTRU   VALOARE      CLIENT_VANZARI
--------- ---------- ---------- ---------------------------------
      146          1    2200.72      2200.72
      146          2    3821.56      2200.72 + 3821.56 = 6022.28
      146          3     141.67      6022.28 + 141.67  = 6163.95
```

10.1.3. Agregări pentru evoluție

Exemplul 10.7

Următorul exemplu, prezintă evoluția mediei vânzărilor pe ultimele trei luni (luna curentă și cele două luni anterioare), pentru un anumit cumpărător.

```
SELECT  client_id,luna,
        SUM(pret_unitar_vanzare*cantitate) valoare,
        AVG(SUM(pret_unitar_vanzare*cantitate))
            OVER (ORDER BY client_id, luna
            ROWS 2 PRECEDING) AS MEDIA_3_LUNI
FROM   vanzari v, timp t
WHERE  v.timp_id=t.id_timp
AND    client_id =146
AND    an=2007
GROUP BY client_id, luna;
```

```
CLIENT_ID LUNA   VALOARE   MEDIA_3_LUNI
---------- ---- ---------- ---------------
       146    3    2200.72      2200.72
       146    4    3623.56      2912.14
       146    5        198      2007.42667
       146    7     141.67      1321.07667
```

Interpretarea rezultatelor afișate pentru coloana *MEDIA_3_LUNI*:

- Linia 1: 2200.72 (valoarea vânzărilor din prima lună)
- Linia 2: (2200.72 + 3623.56)/2 = 2912.14 (media vânzărilor pe 2 luni)
- Linia 3: (2200.72 + 3623.56 + 198)/3 = 2007.42667
- Linia 4: (3623.56 + 198 + 141.67)/3 = 1321.07667

10.1.4. Agregări centrate

Exemplul 10.8

În exemplul care urmează se calculează, pentru un anumit client, evoluția mediei tuturor vânzărilor pe trei luni consecutive, cea din mijloc fiind luna curentă.

```
SELECT client_id, data,
       SUM(pret_unitar_vanzare*cantitate) valoare,
       AVG(SUM(pret_unitar_vanzare*cantitate))
           OVER (PARTITION BY client_id ORDER BY data
           RANGE BETWEEN INTERVAL '1' DAY PRECEDING
                 AND INTERVAL '1' DAY FOLLOWING)
           AS MEDIE_CENTRATA_3_ZILE
FROM   vanzari v, timp t
WHERE  v.timp_id = t.id_timp
AND    client_id = 233
AND    an=2007
GROUP BY client_id, data;

CLIENT_ID  DATA      VALOARE  MEDIE_CENTRATA_3_ZILE
---------- --------- ---------- --------------------
       233 19-MAR-07     242.05               662.075
       233 20-MAR-07     1082.1                845.55
       233 21-MAR-07     1212.5                1147.3
       233 30-MAR-07     231.12                231.12
```

Interpretarea rezultatelor afișate pentru coloana *MEDIE_CENTRATA_ 3_ZILE*:

- Linia 1: (242.05 + 1082.1)/2 = 662.075 (zi1, zi2 consecutive)
- Linia 2: (242.05 + 1082.1 + 1212.5)/3 = 845.55 (zi1, zi2, zi3 consecutive)
- Linia 3: (2200.72 + 3623.56 + 198)/3 = 2007.42667
- Linia 4: 231.12 (zi3, zi4 nu sunt consecutive, zi5 nu există)

10.1.5. Ferestre ce includ duplicate

- Exemplul următor ilustrează modul în care funcțiile agregat aplicate pentru o fereastră calculează valori în prezența duplicatelor.

Exemplul 10.9

```
SELECT oras, produs_id, cantitate,
       SUM(cantitate) OVER
        (ORDER BY produs_id
         RANGE BETWEEN 1 PRECEDING AND CURRENT ROW)
       AS suma_grup
FROM   vanzari v, timp t, regiuni r
WHERE  v.timp_id=t.id_timp
AND    v.regiune_id=r.id_regiune
AND    oras = 'Bucuresti'
AND    produs_id IN (560,3010,4390,5650)
ORDER BY oras, produs_id;
```

```
ORAS                PRODUS_ID  CANTITATE  SUMA_GRUP    /*Calcule*/
------------------- ---------- ---------- -------------------------
Bucuresti                 560         16         16    /* 16 */
Bucuresti                3010          4         10    /* (4+4+2) */
Bucuresti                3010          4         10    /* (4+4+2) */
Bucuresti                3010          2         10    /* (4+4+2) */
Bucuresti                4390          6         11    /* (6+5) */
Bucuresti                4390          5         11    /* (6+5) */
Bucuresti                5650          2         18    /* (2+5+5+6) */
Bucuresti                5650          5         18    /* (2+5+5+6) */
Bucuresti                5650          5         18    /* (2+5+5+6) */
Bucuresti                5650          6         18    /* (2+5+5+6) */
```

- Valorile (2+5+5+6) fac parte din același grup, cel determinat de *produs_id=5650*. Acest lucru se întâmplă datorită utilizării funcției *RANGE* în loc de *ROWS*.
- Dacă s-ar fi folosit *ROWS* în loc de *RANGE*, atunci s-ar fi obținut următorul rezultat:

```
ORAS                PRODUS_ID  CANTITATE  SUMA_GRUP  /*Calcule*/
------------------- ---------- ---------- ------------------------
Bucuresti                 560         16         16  /* 16 */
Bucuresti                3010          4         20  /* 16+4 */
Bucuresti                3010          4          8  /* 4+4 */
Bucuresti                3010          2          6  /* 4+2 */
Bucuresti                4390          6          8  /* 2+6 */
```

```
Bucuresti                 4390          5          11  /* 6+5 */
Bucuresti                 5650          2           7  /* 5+2 */
Bucuresti                 5650          5           7  /* 2+5 */
Bucuresti                 5650          5          10  /* 5+5 */
Bucuresti                 5650          6          11  /* 5+6 */
```

10.1.6. Ferestre cu dimensiuni variabile

- Presupunem că dorim să calculăm variația mediei vânzărilor realizate într-un interval de trei zile lucrătoare.
  - Dacă avem un număr egal de înregistrări pentru fiecare zi lucrătoare și nu avem nicio înregistrare pentru zilele nelucrătoare, atunci putem utiliza o funcție pentru o fereastră fizică.
  - Dacă nu sunt îndeplinite condițiile menționate, vom putea calcula variația mediei vânzărilor utilizând o expresie pentru parametrii dimensionali ai ferestrei. Expresia care determină dimensiunea ferestrei poate fi o referință la o coloană din tabelă sau poate fi o funcție care întoarce limitele corespunzătoare ferestrei pe baza valorilor liniei curente.
- În exemplul următor pentru a seta dimensiunea ferestrei se utilizează în clauza *RANGE* o funcție definită de utilizator.
  - Funcția *f_rang* (*data*) utilizată întoarce valoarea:
    - 4, dacă *data* corespunde unei zile de Luni sau Marți;
    - 2 în orice alt caz.
  - Aceasta înseamnă că dacă o zi din *INTERVAL* este nelucrătoare atunci fereastră se ajustează astfel încât calculele să fie corecte.

Exemplul 10.10

```
CREATE OR REPLACE FUNCTION f_rang (P_ZI_SAPTAMANA number)
RETURN NUMBER IS
rezultat NUMBER;
BEGIN
IF P_ZI_SAPTAMANA IN (2,3) -- 2=Monday or 3=Tuesday
   THEN rezultat:=4;
   ELSE rezultat:=2;
END IF;
RETURN rezultat;
END;
/
```

```
SELECT data, pret_unitar_vanzare*cantitate valoare,
       AVG(pret_unitar_vanzare*cantitate)
       OVER (ORDER BY data RANGE f_rang(zi_saptamana)
       PRECEDING)
         AS medie_valoare
FROM   vanzari v, timp t, regiuni r
WHERE  v.timp_id=t.id_timp
AND    v.regiune_id=r.id_regiune
AND    luna = 7
AND    an = 2007
AND    oras='Bucuresti'
ORDER BY data;
```

```
DATA         VALOARE MEDIE_VALOARE
--------- ---------- -------------
13-JUL-07       8.15    10.5242857
13-JUL-07       8.76    10.5242857
13-JUL-07       4.88    10.5242857
13-JUL-07      10.02    10.5242857
13-JUL-07       5.46    10.5242857
13-JUL-07      19.26    10.5242857
13-JUL-07      17.14    10.5242857
16-JUL-07      12.16    6.99777778
16-JUL-07       7.29    6.99777778
16-JUL-07       3.68    6.99777778
16-JUL-07       2.64    6.99777778
16-JUL-07       1.65    6.99777778
16-JUL-07       1.65    6.99777778
16-JUL-07       7.44    6.99777778
16-JUL-07        1.5    6.99777778
16-JUL-07       9.27    6.99777778
16-JUL-07       2.16    6.99777778
16-JUL-07       2.85    6.99777778
17-JUL-07       1.38    6.29363636
17-JUL-07       3.88    6.29363636
17-JUL-07        3.1    6.29363636
17-JUL-07       4.14    6.29363636
18-JUL-07       61.2    10.2888235
18-JUL-07      48.92    10.2888235
...
```

Interpretarea rezultatelor obținute:

- Zi 13 Iulie (12 Iulie nu este inclusă, 14 Iulie este nelucrătoare)

  (8.15+8.76+4.88+10.02+5.46+19.26+17.14)/7 = 10.5242857

- Zi 16 Iulie (15 Iulie este nelucrătoare, deci precedenta zi lucrătoare este 13 Iulie)

((8.15+8.76+4.88+10.02+5.46+19.26+17.14) +
(12.16+7.29+3.68+2.64+1.65+1.65+7.44+1.5+9.27+2.16+2.85))/18
= 6.99777778

- Zi 17 Iulie (15 Iulie, 16 Iulie zile lucrătoare)

((8.15+8.76+4.88+10.02+5.46+19.26+17.14) +
(12.16+7.29+3.68+2.64+1.65+1.65+7.44+1.5+9.27+2.16+2.85) +
(1.38+3.88+3.1+4.14))/22 = 6.29363636

- Zi 18 Iulie (16 Iulie, 17 Iulie zile lucrătoare)

((12.16+7.29+3.68+2.64+1.65+1.65+7.44+1.5+9.27+2.16+2.85) +
(1.38+3.88+3.1+4.14)+
(61.2+48.92))/17 = 10.2888235

❖ Se remarcă faptul că atunci când o fereastră este definită utilizând un număr într-o funcție fereastră care are clauza *ORDER BY* după o coloană de tipul dată calendaristică, se va înțelege numărul zilelor.

10.1.7. Ferestre cu limite fizice

- Pentru ferestrele la care se fac referiri prin numărul de linii, expresiile de ordonare ar trebui să fie unice pentru a se obține rezultate deterministe.

Exemplul 10.11

De exemplu, în interogarea de mai jos coloana *data* nu are valori unice în mulțimea rezultat.

```
SELECT data, pret_unitar_vanzare*cantitate valoare,
       SUM(pret_unitar_vanzare*cantitate)
           OVER (PARTITION BY data ORDER BY data
           ROWS UNBOUNDED PRECEDING) AS SUMA_VALOARE
FROM   vanzari v, timp t
WHERE  v.timp_id=t.id_timp
AND    luna= 4
AND    zi_luna IN (3, 11)
AND    an = 2007
AND    client_id IN (146,165)
AND    pret_unitar_vanzare*cantitate >20;

DATA         VALOARE   SUMA_VALOARE
--------- ------------ -------------
03-APR-07     319.68     716.59
03-APR-07       64.3     780.89
```

```
03-APR-07       29.7     810.59
03-APR-07       23.8       23.8
03-APR-07        152     962.59
03-APR-07     239.62    1226.01
03-APR-07       33.4    1357.41
03-APR-07       23.8    1550.56
03-APR-07         60    1850.18
03-APR-07     239.62    1790.18
03-APR-07     169.35    1526.76
03-APR-07         98    1324.01
03-APR-07       23.8     986.39
03-APR-07     239.62     263.42
03-APR-07         60     323.42
03-APR-07      73.49     396.91
11-APR-07      538.6      785.6
11-APR-07       23.1      808.7
11-APR-07         49         49
11-APR-07         46      877.7
11-APR-07         23      831.7
11-APR-07        138        187
11-APR-07         60        247
```

În funcție de ordonarea fizică a înregistrărilor se putea obține următorul rezultat:

```
DATA        VALOARE  MEDIE_VALOARE
--------- ---------- -------------
03-APR-07       23.8       23.8
03-APR-07       23.8     986.39
03-APR-07       23.8    1550.56
03-APR-07       29.7     810.59
03-APR-07       33.4    1357.41
03-APR-07         60     323.42
03-APR-07         60    1850.18
03-APR-07       64.3     780.89
03-APR-07      73.49     396.91
03-APR-07         98    1324.01
03-APR-07        152     962.59
03-APR-07     169.35    1526.76
03-APR-07     239.62     263.42
03-APR-07     239.62    1790.18
03-APR-07     239.62    1226.01
03-APR-07     319.68     716.59
11-APR-07         23      831.7
11-APR-07       23.1      808.7
11-APR-07         46      877.7
11-APR-07         49         49
11-APR-07         60        247
11-APR-07        138        187
11-APR-07      538.6      785.6
```

Exemplul 10.12

O modalitate de a rezolva problema expusă în exemplul anterior constă în adăugarea coloanei *client_id* în lista *SELECT* și în includerea clauzei de ordonare în funcție de *data* și de *client_id*.

```
SELECT data, client_id,
       pret_unitar_vanzare*cantitate valoare,
       SUM(pret_unitar_vanzare*cantitate)
           OVER (PARTITION BY data ORDER BY data
            ROWS UNBOUNDED PRECEDING) AS SUMA_VALOARE
FROM   vanzari v, timp t
WHERE  v.timp_id=t.id_timp
AND    luna= 4
AND    zi_luna IN (3, 11)
AND    an = 2007
AND    client_id IN (146,165)
AND    pret_unitar_vanzare*cantitate  >20
ORDER BY data, client_id;
```

DATA	CLIENT_ID	VALOARE	SUMA_VALOARE
03-APR-07	146	319.68	716.59
03-APR-07	146	64.3	780.89
03-APR-07	146	29.7	810.59
03-APR-07	165	23.8	23.8
03-APR-07	165	152	962.59
03-APR-07	165	239.62	1226.01
03-APR-07	165	33.4	1357.41
03-APR-07	165	23.8	1550.56
03-APR-07	165	60	1850.18
03-APR-07	165	239.62	1790.18
03-APR-07	165	169.35	1526.76
03-APR-07	165	98	1324.01
03-APR-07	165	23.8	986.39
03-APR-07	165	239.62	263.42
03-APR-07	165	60	323.42
03-APR-07	165	73.49	396.91
11-APR-07	146	538.6	785.6
11-APR-07	146	23.1	808.7
11-APR-07	165	49	49
11-APR-07	165	46	877.7
11-APR-07	165	23	831.7
11-APR-07	165	138	187
11-APR-07	165	60	247

10.2. Funcțiile FIRST_VALUE și LAST_VALUE

- Funcțiile *FIRST_VALUE* și *LAST_VALUE* obțin prima, respectiv ultima înregistrare din cadrul unei ferestre.
- Aceste înregistrări sunt importante, deoarece constituie baza multor calcule.
  - De exemplu, am putea obține răspunsul la următoarele întrebări:
    - „Care este volumul vânzărilor zilnice, în comparație cu valoarea vânzărilor realizate în prima zi (*FIRST_VALUE*) a perioadei?“
    - „Care este procentul valorii vânzărilor unei regiuni în comparație cu cea mai mare valoare (*LAST_VALUE*) înregistrată în regiunea respectivă?“
- Funcțiile *FIRST_VALUE* și *LAST_VALUE* oferă acces la mai mult de o înregistrare din tabelă fără folosirea unui *self-join*.

Funcția FIRST_VALUE

- Este o funcție analitică care întoarce prima valoare dintr-o mulțime ordonată. Dacă prima valoare a mulțimii ordonate este *null*, atunci funcția întoarce *null* (dacă nu s-a specificat opțiunea *IGNORE NULL*S).

```
FIRST_VALUE (expresie [ IGNORE NULLS ])
OVER (clauza)
```

Exemplul 10.13

În exemplul următor se selectează pentru fiecare produs din categoria „Plicuri securizate“, numele celui mai scump produs.

```
SELECT categorie_5 AS categorie, denumire,
       pret_unitar_curent AS pret,
       FIRST_VALUE(denumire)
          OVER (ORDER BY pret_unitar_curent DESC
                ROWS BETWEEN UNBOUNDED PRECEDING
                AND UNBOUNDED FOLLOWING)
          AS den_prod_cu_pret_maxim
FROM  (SELECT *
       FROM   produse
       WHERE  categorie_5 = 'Plicuri securizate'
       ORDER BY id_produs);
```

```
CATEGORIE           DENUMIRE                       PRET  DEN_PROD_CU_PRET_MAXIM
------------------------------------------------------- ----------------------------
Plicuri securizate  Plicuri securizate 229x324 mm  2.37  Plicuri securizate 229x324 mm
Plicuri securizate  Plicuri securizate 162x229 mm  1.03  Plicuri securizate 229x324 mm
Plicuri securizate  Plicuri securizate 114x229 mm  0.77  Plicuri securizate 229x324 mm
```

- Această funcție are o natură nedeterministă.
- Dacă ar există două produse cu același preț maxim atunci va fi ales ca produs cu preț maxim, acel produs care este returnat primul de subcerere.

Exemplul 10.14

În următorul exemplu, se obține pentru fiecare produs procentul zilnic al vânzărilor, calculat prin raportare la prima zi de vânzare.

```
SELECT produs_id, data, valoare,
       FIRST_VALUE(valoare)
          OVER (PARTITION BY produs_id ORDER BY data)
          AS valoare_zi1,
       ROUND(valoare/FIRST_VALUE(valoare)
                    OVER (PARTITION BY produs_id
                    ORDER BY data)
             * 100,2)  AS procent_din_zi1
FROM  (SELECT v.produs_id, data,
             SUM(pret_unitar_vanzare*cantitate) valoare
      FROM   vanzari v, timp t, produse p
      WHERE  v.timp_id=t.id_timp
      AND    v.produs_id=p.id_produs
      AND    v.produs_id IN (899, 1377, 1378, 1482, 2334 )
      AND    an = 2007
      GROUP BY produs_id, data);
```

```
PRODUS_ID  DATA      VALOARE    VALOARE_ZI1 PROCENT_DIN_ZI1
---------- --------- ---------- ----------- ---------------
       899 14-MAR-07       5.52        5.52             100
       899 21-MAR-07        .92        5.52           16.67
      1377 14-MAR-07       2.94        2.94             100
      1377 20-MAR-07       2.94        2.94             100
      1377 22-MAR-07       1.47        2.94              50
      1378 14-MAR-07        4.5         4.5             100
      1378 22-MAR-07          9         4.5             200
      1378 26-JUN-07       4.26         4.5           94.67
      1378 27-JUN-07        7.1         4.5          157.78
      1482 14-MAR-07         15          15             100
      2334 16-MAR-07       5.53        5.53             100
      2334 21-MAR-07       5.53        5.53             100
      2334 22-MAR-07       2.37        5.53           42.86
      2334 12-JUN-07        4.4        5.53           79.57
      2334 02-JUL-07       3.48        5.53           62.93
```

❖ În exemplul anterior, partiționarea se realizează după codul produsului, ordonarea liniilor în ferestre realizându-se după data calendaristică, astfel încât funcția *FIRST_VALUE*(valoare) determină valoarea vânzărilor din prima zi a submulțimii.

Funcția LAST_VALUE

- Este o funcție analitică care întoarce ultima valoare dintr-o mulțime ordonată de valori.
- Dacă ultima valoare a mulțimii ordonate este *null*, atunci funcția întoarce *null* (dacă nu s-a specificat opțiunea *IGNORE NULLS*).

```
LAST_VALUE (expresie [ IGNORE NULLS ])
OVER (clauza)
```

Exemplul 10.15

În exemplul următor se selectează pentru fiecare produs din categoria „Plicuri securizate", numele celui mai ieftin produs.

```
SELECT  categorie_5 AS categorie, denumire,
        pret_unitar_curent AS pret,
        LAST_VALUE(denumire)
          OVER (ORDER BY pret_unitar_curent DESC
                ROWS BETWEEN UNBOUNDED PRECEDING
                AND UNBOUNDED FOLLOWING)
          AS den_prod_cu_pret_minim
FROM   (SELECT *
        FROM   produse
        WHERE  categorie_5 = 'Plicuri securizate'
        ORDER BY id_produs);
```

```
CATEGORIE          DENUMIRE                      PRET   DEN_PROD_CU_PRET_MINIM
------------------ ----------------------------- ------ -----------------------------
Plicuri securizate  Plicuri securizate 114x229 mm  0.77  Plicuri securizate 114x229 mm
Plicuri securizate  Plicuri securizate 162x229 mm  1.03  Plicuri securizate 114x229 mm
Plicuri securizate  Plicuri securizate 229x324 mm  2.37  Plicuri securizate 114x229 mm
```

10.3. Funcții SQL de agregare pentru raportare

- După ce s-a procesat interogarea, valorile agregate, numărul înregistrărilor din rezultat sau valoarea mediei dintr-o coloană pot fi ușor calculate la nivel de partiție și puse la dispoziția altor funcții de raportare.
- Funcțiile agregat pentru rapoarte (de exemplu, *SUM*, *AVG*, *MAX*, *MIN*, *COUNT* etc) întorc aceeași valoare agregat pentru fiecare linie a submulțimii unei partiții. Comportamentul acestor funcții față de valoarea *null* este același ca și al funcțiilor de agregare standard.
- Un avantaj al funcțiilor de raportare este multiprocesarea datelor într-o singură interogare și o mai bună performanță a interogării. Cereri precum: „Numărul vânzătorilor care au vânzări mai mari de 10% din vânzările înregistrate la nivel de oraș“, nu necesită *join*-uri între blocuri de interogări separate.

Exemplul 10.16

De exemplu, considerăm următoarea cerere: „Pentru fiecare subgrupă de produse al raionului IT, să se determine orașul care a înregistrat vânzări maxime în luna mai a anului 2007“.

```
SELECT categorie_3 AS subgrupa, oras, valoare
FROM(SELECT categorie_3, oras,
            SUM(pret_unitar_vanzare*cantitate) AS valoare,
            MAX(SUM(pret_unitar_vanzare*cantitate ))
              OVER (PARTITION BY categorie_3) AS max_val
     FROM   vanzari v, regiuni r,
            produse p, timp t
     WHERE  v.regiune_id=r.id_regiune
     AND    v.produs_id=p.id_produs
     AND    v.timp_id=t.id_timp
     AND    categorie_1 = 'IT'
     AND    luna = 5
     AND    an = 2007
     GROUP BY categorie_3, oras)
WHERE valoare= max_val;
```

Subcererea obține următoarele date:

```
CATEGORIE_3                 ORAS                VALOARE    MAX_VAL
--------------------------- ------------------- ---------- ----------
Carduri de memorie          Bucuresti                   64      148.8
Carduri de memorie          Iasi                     148.8      148.8
```

```
Carduri de memorie            OTOPENI                      64      148.8
Proiectoare                   Bucuresti                 1553.78    2952.18
Proiectoare                   Iasi                      2952.18    2952.18
```

Rezultatul cererii este următorul:

```
CATEGORIE_3                   ORAS                    VALOARE
--------------------------------------------------- ---------
Carduri de memorie            Iasi                      148.8
Proiectoare                   Iasi                     2952.18
```

Agregatele de raportare în combinație cu subinterogările, ne permit să răspundem eficient la interogări complexe.

Exemplul 10.17

Următoarea interogare obține pentru anul 2007 cele mai bine vândute 5 produse din fiecare grupă, care contribuie cu mai mult de 20% la vânzările raionului din care face parte.

```
SELECT categorie_1 AS raion, categorie_2 AS grupa,
       produs, rang , vanzari
FROM (SELECT categorie_1, categorie_2,
             v.produs_id produs, denumire,
             SUM(pret_unitar_vanzare*cantitate ) VANZARI,
             SUM(SUM(pret_unitar_vanzare*cantitate ))
               OVER (PARTITION BY categorie_1) AS CAT_1,
             SUM(SUM(pret_unitar_vanzare*cantitate))
               OVER (PARTITION BY categorie_2) AS CAT_2,
             DENSE_RANK()
               OVER  (PARTITION BY categorie_2
               ORDER BY SUM(pret_unitar_vanzare*cantitate))
               AS rang
       FROM  vanzari v, regiuni r,
             produse p, timp t
       WHERE v.regiune_id=r.id_regiune
       AND   v.produs_id=p.id_produs
       AND   v.timp_id=t.id_timp
       AND   an = 2007
       GROUP BY categorie_1, categorie_2,
                v.produs_id, denumire)
WHERE CAT_2>0.2*CAT_1 AND rang<=5;
```

```
RAION           GRUPA                                  PRODUS       RANG    VANZARI
--------------- ------------------------------------- ---------- ---------- ----------
IT              Audio-Video                                2095          1    9105.15
Papetarie       Hartie de copiator                          103          1      18.42
Papetarie       Hartie de copiator                         4749          1      18.42
Papetarie       Hartie de copiator                         3531          1      18.42
```

```
Papetarie       Hartie de copiator                    2337          4        50.4
Papetarie       Hartie de copiator                    3712          5       88.54
Papetarie       Organizare și arhivare                 411          1         1.4
Papetarie       Organizare și arhivare                4850          1         1.4
Papetarie       Organizare și arhivare                4734          3         1.5
Papetarie       Organizare și arhivare                3358          4        1.75
Papetarie       Organizare și arhivare                4513          5         2.2
```

10.3.1. Funcția RATIO_TO_REPORT

- Această funcție calculează proporția unei valori raportată la suma tuturor valorilor. Dacă *expr* este *null*, atunci funcția *RATIO_TO_REPORT* va întoarce *null*, dar valoarea *null* va fi tratată ca zero în calculul sumei de la numitor.

  Sintaxa este următoarea:

```
RATIO_TO_REPORT
(<expr1>) OVER
        ([PARTITION BY <expr>[,...]])
```

- Atunci când se dorește determinarea contribuției unui produs la totalul vânzărilor, partea unui client din totalul datornicilor, procentul valorii unui produs din valoarea unei facturi, ponderea salariului directorului general (sau al șefilor de departamente) în totalul fondului de salarii al unei companii etc se poate utiliza funcția *RATIO_TO_REPORT*.

Exemplul 10.18

De exemplu, pentru a calcula contribuția vânzărilor realizate pe tipuri de plată la totalul vânzărilor se poate utiliza următoarea cerere:

```
SELECT cod,
      SUM(pret_unitar_vanzare*cantitate) VANZARI,
      SUM(SUM(pret_unitar_vanzare*cantitate)) OVER ()
          AS TOTAL_VANZ,
      RATIO_TO_REPORT(
          SUM(pret_unitar_vanzare*cantitate))
          OVER ()
          AS RATIO_REP
FROM   vanzari v, plati p
WHERE  v.plata_id=p.id_plata
GROUP BY cod;

COD  VANZARI TOTAL_VANZ RATIO_ REP
---- ------- ----------- --------
DP    117656  121762     0.966
Num     4106  121762     0.034
```

10.3.2. Funcțiile LAG/LEAD

- Funcțiile *LAG* și *LEAD* sunt utile pentru a compara valori atunci când pozițiile relative ale liniilor sunt știute. Acestea funcționează pe baza numărului de linii care separă linia țintă de linia curentă. Pentru că aceste funcții asigură acces la mai mult de o înregistrare a tabelei, fără să se utilizeze *self-join*, ele măresc viteza de procesare.
- Funcția *LAG* asigură acces la una sau mai multe linii anterioare poziției curente, iar funcția *LEAD* asigură acces la una sau mai multe linii ulterioare poziției curente.
- Aceste funcțiile au următoarea sintaxa:

```
{LAG | LEAD}
   (<expr1>, [<nr_randuri_anterioare> [, <default>]])
    OVER
      ([PARTITION BY <expr2>[,...]]
       ORDER BY <expr3>
      [ASC | DESC] [NULLS FIRST | NULLS LAST] [,...])
```

  - *Nr_randuri_anterioare* este un parametru opțional care are valoarea implicita 1.
  - *Default* este un parametru opțional și reprezintă valoarea întoarsă atunci când *nr_randuri_anterioare* depășește limitele tabelei sau ale submulțimiii**.**
- La analiza dinamicii în timp a unei mărimi se compară valoarea din linia curentă cu cea dintr-o altă linie, plasată la o anumită distanță.
  - De exemplu, pentru compararea vânzărilor fiecărei luni calendaristice cu aceiași lună din anul precedent, va trebui să se împartă valoarea de pe linia curentă la o valoare aflată cu 12 linii (adică luni) înainte. Apare astfel noțiunea de *deplasare înapoi* (*LAG*) sau *înainte* (*LEAD*). Soluția funcționează numai dacă nu există luni de întrerupere, adică intervalul este continuu.

Exemplul 10.19

În exemplul de mai jos se compară vânzările zilei curente cu cele ale zilei anterioare, respectiv ale zilei următoare (deplasarea este 1).

```
SELECT data,
       SUM(pret_unitar_vanzare*cantitate) AS VANZARI,
       LAG(SUM(pret_unitar_vanzare*cantitate),1)
             OVER (ORDER BY data) AS LAG1,
       LEAD(SUM(pret_unitar_vanzare*cantitate),1)
             OVER (ORDER BY data) AS LEAD1
FROM   vanzari v, timp t
WHERE  v.timp_id = t.id_timp
```

```
AND    luna = 7
AND    an = 2007
GROUP BY data;
```

```
DATA         VANZARI       LAG1      LEAD1
--------- ---------- ---------- ----------
16-JUL-07        128                   13
17-JUL-07         13        128        113
18-JUL-07        113         13        164
19-JUL-07        164        113        413
23-JUL-07        413        164         21
24-JUL-07         21        413        452
26-JUL-07        452         21         45
27-JUL-07         45        452
...
```

Exemplul 10.20

În exemplul următor se determină diferența dintre valoarea vânzărilor de la o zi la alta pentru fiecare produs (diferențele zilnice dintre vânzările fiecărui produs).

```
SELECT produs_id, data, vanzari,
       vanzari-LAG (vanzari, 1)
            OVER (PARTITION BY produs_id ORDER BY data)
         AS Diferenta
FROM (SELECT produs_id, data,
             SUM(pret_unitar_vanzare*cantitate) AS vanzari
      FROM   vanzari v, timp t
      WHERE  v.timp_id=t.id_timp
      AND    produs_id IN (899, 1377, 1378, 1482, 2334 )
      AND    luna = 3
      AND    an = 2007
      GROUP BY produs_id, data);
```

```
PRODUS_ID   DATA        VANZARI  DIFERENȚA
---------- --------- ---------- ----------
       899 14-MAR-07       5.52
       899 21-MAR-07        .92       -4.6
      1377 14-MAR-07       2.94
      1377 20-MAR-07       2.94          0
      1377 22-MAR-07       1.47      -1.47
      1378 14-MAR-07        4.5
      1378 22-MAR-07          9        4.5
      1482 14-MAR-07         15
      2334 16-MAR-07       5.53
      2334 21-MAR-07       5.53          0
      2334 22-MAR-07       2.37      -3.16
```

Exemplul 10.21

Pentru listarea evoluției globale a vânzărilor, de pe o zi pe alta, soluția este dată mai jos (evoluția zilnică a vânzărilor).

```
SELECT data, vanzari,
       vanzari-LAG(vanzari, 1) OVER (ORDER BY data)
         AS Diferenta
FROM (SELECT data,
             SUM(pret_unitar_vanzare*cantitate) AS vanzari
      FROM   vanzari v, timp t
      WHERE  v.timp_id=t.id_timp
      AND    luna = 3
      AND    an = 2007
      GROUP BY data);
```

```
DATA         VANZARI  DIFERENTA
--------- ---------- ----------
05-MAR-07    1442.86
06-MAR-07        231   -1211.86
12-MAR-07    2683.01    2452.01
13-MAR-07   20989.17   18306.16
14-MAR-07    19232.2   -1756.97
15-MAR-07    5502.92  -13729.28
16-MAR-07    7153.97    1651.05
19-MAR-07      484.1   -6669.87
20-MAR-07   18392.99   17908.89
21-MAR-07    7666.15  -10726.84
22-MAR-07    6408.46   -1257.69
23-MAR-07    2813.51   -3594.95
28-MAR-07     475.89   -2337.62
29-MAR-07      594.9     119.01
30-MAR-07    1836.94    1242.04
```

10.3.3. Funcțiile FIRST/LAST

- Funcțiile agregat *FIRST* și *LAST* ne permit să obținem rezultatul unui agregat aplicat unei mulțimi de înregistrări ce sunt considerate ca fiind primele sau ultimele după o ordine specificată.
- Atunci când este necesară o valoare din prima sau din ultima linie a unui grup sortat, dar valoarea necesară nu este cea după care s-a realizat sortarea grupului, funcțiile *FIRST* și *LAST* elimină necesitatea utilizării *selfjoin*-ului sau a vizualizărilor, mărind astfel performanțele.
- Funcțiile *FIRST*/*LAST* pot fi folosite și ca funcții de agregare obișnuite.

- Sintaxa funcțiilor *FIRST*/*LAST* este următoarea:

```
[MIN | MAX | COUNT | SUM | AVG ...] (<expr>)
KEEP (DENSE_RANK [FIRST | LAST]
      ORDER BY <order by expr> [, ...]
[ASC|DESC] [NULLS FIRST| NULLS LAST] )
```

  - Funcțiile agregat sunt aplicate pe o mulțime de înregistrări clasificate ca fiind primele/ultimele în funcție de ordinea specificată.
  - Prin clauza *KEEP* se specifică ordonarea utilizată în fiecare grup.
  - Clauza *DENSE_RANK FIRST* sau *DENSE_RANK LAST* indică faptul că se vor agrega doar acele înregistrări cu valoarea *DENSE RANK* minimă (*FIRST*) sau maximă (*LAST*).

Exemplul 10.22

În următorul exemplu se obține pentru fiecare raion, prețul unitar minim al produselor care au cel mai mic stoc curent și prețul unitar maxim al produselor cu cel mai mare stoc curent.

```
SELECT categorie_1 AS raion,
     MIN(pret_unitar_curent)
       KEEP (DENSE_RANK FIRST ORDER BY stoc_curent) minim,
     MAX(pret_unitar_curent)
       KEEP (DENSE_RANK LAST ORDER BY stoc_curent) maxim
FROM  produse
GROUP BY categorie_1;
```

```
RAION                MINIM      MAXIM
--------------- ---------- ----------
IT                    1.48      15.19
Industriale           2.99         41
Mobilier             46.35       51.5
Papetarie              .16         .6
```

Exemplul 10.23

Următorul exemplu realizează aceleași calcule, numai că de această dată pentru fiecare produs din raion care îndeplinește condiția de filtrare.

```
SELECT categorie_1 AS raion, denumire, pret_unitar_curent,
       MIN(pret_unitar_curent)
         KEEP (DENSE_RANK FIRST ORDER BY stoc_curent)
         OVER (PARTITION BY categorie_1) minim,
       MAX(pret_unitar_curent)
         KEEP (DENSE_RANK LAST ORDER BY stoc_curent)
         OVER (PARTITION BY categorie_1) maxim
FROM  produse
WHERE id_produs IN (394,5884,1394,757,1464,1631,380,
```

```
                    3238,135,125,123,117,119)
ORDER BY categorie_1, pret_unitar_curent;
```

RAION	DENUMIRE	PRET_UNITAR	MINIM	MAXIM
IT	Canon LS39EBL	11.32	209	11.32
IT	Panasonic KX-TS620	209	209	11.32
IT	Toner LEXMARK-12S0400	243.51	209	11.32
IT	HP LJ 5200DTN	8347	209	11.32
IT	Lexmark W840N	10556.83	209	11.32
Industriale	AMBIPUR CAR	23	23	23
Industriale	MOMENT PARCHET	23	23	23
Mobilier	Fotoliu NOVA, piele neagra	1095.97	1095.97	1440.6
Mobilier	Dulap metalic 2 usi	1440.6	1095.97	1440.6
Papetarie	Cutter mic Eagle	2.06	11.85	11.86
Papetarie	Dosare de incopciat 1/2 Lux	2.89	11.85	11.86
Papetarie	Caiet Tranz matematica	11.85	11.85	11.86
Papetarie	Hartie Office Express A4	11.86	11.85	11.86

Exemplul 10.24

În exemplul următor se obține în plus și prețul unitar al produsului ca procent din prețul unitar maxim.

```
SELECT categorie_1 raion, denumire, pret_unitar_curent,
 MAX(pret_unitar_curent)
   KEEP (DENSE_RANK LAST ORDER BY stoc_curent)
     OVER (PARTITION BY categorie_1)  maxim,
   100*pret_unitar_curent /(MAX(pret_unitar_curent)
     OVER(PARTITION BY (categorie_1))) AS procent
FROM   produse
WHERE  id_produs IN (394,5884,1394,757,1464,1631,
                     380,3238,135,125,123,117,119)
ORDER BY categorie_1, pret_unitar_curent;
```

RAION	DENUMIRE	PRET_UNITAR	MAXIM	PROCENT
IT	Canon LS39EBL	11.32	11.32	0.11
IT	Panasonic KX-TS620	209	11.32	1.98
IT	Toner LEXMARK-12S0400	243.51	11.32	2.31
IT	HP LJ 5200DTN	8347	11.32	79.07
IT	Lexmark W840N	10556.83	11.32	100.00
Industriale	AMBIPUR CAR	23	23	100.00
Industriale	MOMENT PARCHET	23	23	100.00
Mobilier	Fotoliu NOVA, piele neagra	1095.97	1440.6	76.08
Mobilier	Dulap metalic 2 usi	1440.6	1440.6	100.00
Papetarie	Cutter mic Eagle	2.06	11.86	17.37
Papetarie	Dosare de incopciat 1/2 Lux	2.89	11.86	24.37
Papetarie	Caiet Tranz matematica	11.85	11.86	99.92
Papetarie	Hartie Office Express A4	11.86	11.86	100.00

Bibliografie

1. Connolly T.M., Begg C.E., *Database Systems*: *A Practical Approach to Design, Implementation and Management*, 5th edition, Pearson Education, 2005
2. Dollinger R., Andron L., *Baze de date și gestiunea tranzacțiilor*, Editura Albastră, Cluj-Napoca, 2004
3. Inmon W.H., *Building the Data Warehouse*, 4th Edition, Wiley, 2005
4. Kimball R., *The Data Warehouse Toolkit*, 3rd Edition, Wiley, 2013
5. Kimball R., Ross M., Thornthwaite W., Mundy J., Becker B., *The Data Warehouse Lifecycle Toolkit*, 2nd Edition Wiley, 2008
6. Oracle and/or its affiliates, *Oracle Database Concepts*, 1993, 2024
7. Oracle and/or its affiliates, *Oracle Database Administrator's Guide*, 2001, 2025
8. Oracle and/or its affiliates, *Oracle Database Performance Tuning Guide*, 2013, 2025
9. Oracle and/or its affiliates, *Oracle Database SQL Language Reference*, 1996, 2025
10. Oracle and/or its affiliates, *Oracle Database PL/SQL Language Reference*, 1996, 2025
11. Oracle and/or its affiliates, *Oracle Database: SQL Tuning Workshop*, 2010, 2025
12. Oracle and/or its affiliates, *Oracle OLAP Customizing Analytic Workspace Manager*, 2006, 2019
13. Oracle and/or its affiliates, *Oracle OLAP DML Reference*, 1994, 2019
14. Oracle and/or its affiliates, *Oracle OLAP User's Guide*, 2003, 2019
15. Oracle and/or its affiliates, *Oracle Warehouse Builder Concepts*, 2000, 2021
16. Oracle and/or its affiliates, *Oracle Warehouse Builder Data Modeling, ETL, and Data Quality Guide*, 2000, 2021
17. Oracle and/or its affiliates, *Oracle Database Data Warehousing Guide*, 2001, 2021
18. Oracle University, *Oracle Database*: *PL/SQL Fundamentals*, *Student Guide*, 2009, 2025
19. Poe V., Klauer P., Brobst S., *Building A Data Warehouse for Decision Support*, 2nd Edition, Prentice Hall; 1997
20. Popescu I., Alecu A., Velcescu L., Florea (Mihai) G., *Programare avansată în Oracle9i*, Ed. Tehnică, 2004