

Marin Fotache

SQL

Dialecte DB2, Oracle
și Visual FoxPro

© 2001 by Editura POLIROM

Editura POLIROM
Iași, B-dul Copou nr. 4, P.O. BOX 266, 6600
București, B-dul I.C. Brătianu nr. 6, et. 7
<http://www.polirom.ro>

Descrierea CIP a Bibliotecii Naționale :

FOTACHE, MARIN
SQL. Dialecte DB2, Oracle și Visual FoxPro / Marin Fotache
Iași: Polirom, 2001
368 p., 24 cm
ISBN : 973-683-709-2

004

Printed in ROMANIA

POLIROM
2001

Scripturi cu interogări sevențiale în Oracle și DB2

Deși oarecum stânjenitoare, sevențializarea interogărilor se poartă și la case mai mari. Una dintre ele – Oracle, iar alta IBM DB2. Fie că este vorba despre cunoștințe nu prea avansate de SQL, fie despre o anumită comoditate, cert este că de multe ori interogări insolubile se pot rezolva ceva mai primitiv.

Care este clientul cu cel mai mare rest de plată?

Încercăm să facem uitață penibila soluție formulată în paragraful precedent, ce-i drept, înlocuind-o cu o soluție tot din „lumea a treia” a SQL-ului, deoarece se bazează pe un script în care rezultatele intermediare ale consultărilor se salveză nu în cursoare, ci în tabele derivate.

Listing 5.9. Script Oracle de afilare a clientului cu cel mai mare rest de plată

```
-- Valorile facturate, pe clienti
DROP VIEW vDe_Incasat ;
DROP VIEW vIncasat ;
DROP VIEW vFacturat ;

CREATE VIEW vFacturat AS
  SELECT CodCl, SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Facturat
  FROM FACTURI F, LINIIFACT LF, PRODUSE P
  WHERE F.NrFact=LF.NrFact AND LF.CodPr=P.CodPr
  GROUP BY CodCl ;

-- Valorile incasate, pe clienti
CREATE VIEW vIncasat AS
  SELECT CodCl, SUM(Transa) AS Incasat
  FROM FACTURI F, INCASFACT I
  WHERE F.NrFact=I.NrFact
  GROUP BY CodCl ;

-- Restul de plată, pe clienti, în ordinea
-- (descrescătoare a) valorii
CREATE VIEW vDe_INCASAT AS
  SELECT DenCl, vFACTURAT.CodCl, Facturat, NVL(Incasat,0)
  AS Incasat,
  Facturat - NVL(Incasat,0) AS De_Incasat
  FROM vFACTURAT, vINCASAT, CLIENTI
  WHERE vFACTURAT.CodCl=CLIENTI.CodCl
  AND vFACTURAT.CodCl=vINCASAT.CodCl (+);

-- Finalul
SELECT DenCl, vDe_INCASAT.*
FROM vDe_INCASAT, CLIENTI
WHERE vDe_INCASAT.CodCl=CLIENTI.CodCl
AND De_Incasat >= ALL
  (SELECT De_Incasat
  FROM vDe_INCASAT) ;
```

De remarcat că și în VFP, în loc de cursoare puteam folosi tabele derivate...

Care sunt primii trei clienți în ordinea descrescătoare a datorilor (cei mai mari trei datornici)?

Iată o interogare al cărei răspuns ar interesa multă lume, mai ales dacă baza de date ar fi la nivel național... Cu toate opțiunile de care suntem în stare, la acest moment ne este cu neputință să răspundem altfel decât printr-un script – listingul 5.10. Din nefericire, soluția a fost ingreunată de faptul că nu se poate folosi clauza ORDER BY într-o comandă CREATE VIEW.

Listing 5.10. Script Oracle pentru extragerea primilor 3 datornici

```
DROP VIEW vOrdonare ;
DROP VIEW vDe_Incasat ;
DROP VIEW vIncasat ;
DROP VIEW vFacturat ;

-- Valorile facturate, pe clienti
CREATE VIEW vFacturat AS
  SELECT CodCl, SUM(Cantitate * PretUnit * (1+ProcTVA))
  AS Facturat
  FROM FACTURI F, LINIIFACT LF, PRODUSE P
  WHERE F.NrFact=LF.NrFact AND LF.CodPr=P.CodPr
  GROUP BY CodCl
  WITH READ ONLY;

-- Valorile incasate, pe clienti
CREATE VIEW vIncasat AS
  SELECT CodCl, SUM(Transa) AS Incasat
  FROM FACTURI F, INCASFACT I
  WHERE F.NrFact=I.NrFact
  GROUP BY CodCl
  WITH READ ONLY;

-- Restul de plată, pe clienti, în ordinea
-- (descrescătoare a) valorii
CREATE VIEW vDe_INCASAT AS
  SELECT DenCl, vFACTURAT.CodCl, Facturat, NVL(Incasat,0)
  AS Incasat,
  Facturat - NVL(Incasat,0) AS De_Incasat
  FROM vFACTURAT, vINCASAT, CLIENTI
  WHERE vFACTURAT.CodCl=CLIENTI.CodCl
  AND vFACTURAT.CodCl=vINCASAT.CodCl (+)
  WITH READ ONLY;

-- Prinim trei datornici
CREATE VIEW vORDONARE AS
  SELECT *
  FROM vDe_INCASAT
  WHERE De_Incasat >=
    (SELECT MAX(De_Incasat)
    FROM vDe_INCASAT
    WHERE De_Incasat <
      (SELECT MAX(De_Incasat)
      FROM vDe_INCASAT
      WHERE De_Incasat <
        (SELECT MAX(De_Incasat)
        FROM vDe_INCASAT)
      )
    )
  )
  WITH READ ONLY;

-- Ordonare finală
SELECT *
FROM vOrdonare
ORDER BY De_Incasat DESC
```

6.2. Subconsultări în clauza FROM	236
6.3. Subconsultări scalare în clauza SELECT	251
6.4. Interrogări ierarhice	256
<i>Soluția clasică – autojoinuri</i>	257
<i>Interrogări arborescente în Oracle</i>	264
6.5. Actualizarea tabelelor prin subconsultări	269

Caietul 7

SQL și OLAP

7.1. Subtotaluri. ROLLUP și GROUPING	273
<i>Combinarea opțiunilor ROLLUP și GROUPING</i>	280
<i>ROLLUP-uri parțiale</i>	281
7.2. Analize multidimensionale. Operatorii CUBE și GROUPING SETS	287
<i>Operatorul GROUPING SETS</i>	295
<i>CUBE-uri parțiale</i>	298
7.3. Clasamente – soluții clasice și OLAP	301
7.4. Ferește pentru funcțiile analitice	314
7.5. Comparări și ponderi în Oracle 8i2	318

Capitolul 8

OBIECTE DIN SCHEMA BAZEI DE DATE. EXTENSII PROCEDURALE ALE SQL

8.1. Dicționarul bazel de date. Reguli de validare avansate	324
8.2. Tabele virtuale în Oracle și VFP	329
<i>Tabele virtuale în Oracle</i>	332
<i>Tabele virtuale în Visual FoxPro</i>	333
8.3. Proceduri și funcții stocate în VFP și Oracle	334
<i>Valori implicate</i>	335
<i>Funcții stocate și fraze SELECT în Visual FoxPro</i>	338
<i>Funcții stocate și fraze SELECT în Oracle</i>	341
8.4. Declanșatoare în VFP și Oracle	343
<i>Declanșatoare în VFP6</i>	343
<i>Declanșatoare în Oracle 8i</i>	350

Bibliografie	361
--------------------	-----

La Editura POLIROM

au apărut:

Emanuela Cerchez – *Internet. Utilizarea rețelei Internet.*

Proiectarea paginilor Web (manual opțional pentru liceu)

Silvia Curteanu – *PC. Ghid de utilizare*

Doina Hrinciu Logofătu – *C++. Probleme rezolvate și algoritmi*

Emanuela Cerchez, Marinel Serban – *PC. Pas cu pas*

Marin Fotache – *SQL. Dialecte DB2, Oracle și Visual FoxPro*

în pregătire:

Cristina Perhinschi, Petronela Ilucă – *Word. Simplu și eficient*

Capitolul 2
ALGEBRA RELAȚIONALĂ

2.1. Caracterizare generală a limbajelor de interogare	49
2.2. Operatorii asamblării	51
<i>Reuniunea</i>	51
<i>Intersecția</i>	52
<i>Diferența</i>	53
<i>Produsul cartezian</i>	54
2.3. Operatorii relaționali	55
<i>Selecția</i>	55
<i>Proiecția</i>	57
<i>Înlănțuirea consultărilor</i>	58
<i>Joncțiunea</i>	60
<i>Alte două tipuri de joncțiune: joncțiunea externă și semijoncțiunea</i>	68
<i>Diviziunea</i>	70
2.4. Alte notații și reprezentarea grafică ale interogărilor	74
<i>Notația matematică</i>	75
<i>Gramatica BNF</i>	76
<i>Notații diverse</i>	77
<i>Reprezentarea grafică a interogărilor</i>	78

Capitolul 3

CREAREA BAZELOL DE DATE PRIN COMENZI SQL

3.1. Scurt istoric al SQL	82
3.2. Tipuri de date și comenzi principale SQL	88
3.3. Crearea, modificarea și stergerea tabelelor. Restricții	90
<i>Crearea tabelelor și declararea atributelor</i>	90
<i>Declararea restricțiilor</i>	92
<i>Crearea tabelelor și declararea restricțiilor în Oracle 8</i>	95
<i>Crearea tabelelor și declararea restricțiilor în DB2</i>	99
<i>Crearea tabelelor și declararea restricțiilor în Visual FoxPro</i>	102
<i>Modificarea structurii tabelelor/restricțiilor în Oracle, DB2 și VFP</i>	108
<i>Ștergerea tabelelor</i>	111
3.4. Inserarea, modificarea, ștergerea linilor	111
<i>Adăugarea unei lini</i>	111
<i>Ștergerea linilor</i>	116
<i>Modificarea valorilor unor atribute</i>	116

Capitolul 4
ELEMENTE DE BAZĂ ALE INTEROGĂRILOR SQL

4.1. Principalele clauze ale frazei SELECT	118
<i>Selecția și proiecția</i>	118
<i>Reuniune, intersecție, diferență, produs cartezian</i>	120
<i>Coloane-expresii</i>	123
<i>Opțiunea ORDER BY</i>	127
4.2. Operatorii BETWEEN, LIKE, IN	128
<i>Operatorul BETWEEN</i>	128
<i>Operatorul LIKE</i>	130
<i>Operatorul IN</i>	133
4.3. Theta- și echi-joncțiunea	134
4.4. Sinonime locale și joncțiunea unei tabele cu ea însăși	137
4.5. Funcții-agregat: COUNT, SUM, AVG, MIN, MAX	140
<i>Funcția COUNT</i>	141
<i>Funcția SUM</i>	142
<i>Funcția AVG</i>	147
<i>Funcțiile MAX și MIN</i>	148
4.6. Gruparea tuplurilor. GROUP BY și HAVING	150
<i>Clausa GROUP BY</i>	151
<i>Clausa HAVING</i>	158

Capitolul 5

SQL (CEVA MAI) AVANSAT

5.1. Prelucrarea valorilor nule	163
5.2. Joncțiunea externă	171
5.3. Structuri alternative: CASE, DECODE, IIF	177
5.4. Subconsultări. Operatorul IN	186
5.5. Operatorii ALL, SOME, ANY	200
5.6. Subconsultări în clauza HAVING	202
5.7. Secvențializarea interogărilor	213
<i>Fraze SELECT independente în VFP</i>	213
<i>Scripturi cu interogări secvențiale în Oracle și DB2</i>	214
<i>Expresii-tabele în DB2</i>	218
<i>Exprezii-tabele în DB2</i>	220

Capitolul 6

SQL ȘI MAI AVANSAT

6.1. Interogări corelate. Operatorul EXISTS	224
<i>Subconsultări dublu corelate</i>	231

- [Fotache97] Fotache, M. – *Baze de date. Organizare, interogare și normalizare*, ediția a II-a, Editura Junimea, Iași, 1997.
- [Fotache00-1] Fotache, M. – „Despre prima formă normală”, *PC Report* nr. 92, mai 2000.
- [Fotache00-2] Fotache, M. – „Despre NULL-i, dar numai în SQL”, *PC Report* nr. 93, iunie 2000.
- [Fotache00-3] Fotache, M. – „Juncțiunea externă și urmările ei”, *PC Report* nr. 94, iulie 2000.
- [Fotache00-4] Fotache, M. – „Clasamente”, *PC Report* nr. 98, noiembrie 2000.
- [Fotache00-5] Fotache, M. – „Funcții OLAP în SQL-99 și Oracle 8i2”, *PC Report* nr. 98, noiembrie 2000.
- [Fotache00-6] Fotache, M. – „Alte funcții OLAP în SQL-99 și Oracle 8i2”, *PC Report* nr. 99, decembrie 2000.
- [Fotache&Strimbei99-1] Fotache, M., Strimbei, C. – „Proceduri stocate și triggers în FoxPro”, *PC Report&Byte* nr. 80, mai 1999.
- [Fotache&Strimbei99-2] Fotache, M., Strimbei, C. – „Tabele derivate”, *PC Report&Byte* nr. 81, iunie 1999.
- [Gardarin91] Gardarin, G. – *Bases de données. Les systèmes et leurs langages*, Eyrolles, Paris, 1991.
- [Grama&Filip00] Grama, A., Filip, M. – *Medii de programare în economie*, Editura Sedcom Libris, Iași, 2000.
- [Greenblatt&Waxman78] Greenblatt, D., Waxman, J. – *A Study of Three Database Query Language: Improving Usability and Responsiveness*, Academic Press, 1978.
- [Gorman97] Gorman, M. – *The Role of NIST in SQL Standardization*, The Data Administration Newsletter, www.tdan.com.
- [Groff&Weinberg94] Groff, J., Weinberg, P. – *LAN Times Guide to SQL*, Osborne McGraw-Hill, Berkeley, 1994.
- [Hainaut94] Hainaut, J.L. – *Bases de données et modèles de calcul. Outils et méthodes pour l'utilisateur*, InterEditions, Paris, 1994.
- [Hernandez&Viescas00] Hernandez, M.J., Viescas, J.L. – *SQL Queries for Mere Mortals*, Addison Wesley, Boston, 2000.
- [Hursch&Hursch90] Hursch, C.J., Hursch, J.L. – *SQL – Le langage structuré d'interrogation*, Masson, Paris, 1990.
- [Korth&Silberschatz88] Korth, H.F., Silberschatz, A. – *Systèmes de gestion des bases de données*, McGraw-Hill, Paris, 1988.
- [Koutchouk92] Koutchouk, M. – *SQL et DB2. Le relationnel et sa pratique*, 2eme édition, Masson, Paris, 1992.
- [Kreines00] Kreines, D.C. – *Oracle SQL: The Essential Reference*, O'Reilly, Sebastopol (CA), 2000.
- [Larsen96] Larsen, S. – „Powerful SQL: Beyond the Basics”, *DB2 Magazine On Line*, Winter 1996.
- [Larsen98] Larsen, S. – „The SQL Double Double”, *DB2 Magazine On Line*, Spring 1998.
- [Lungu s.a. 95] Lungu, I., Bodea, C., Bădescu, G., Ionijă, C. – *Baze de date. Organizare, proiectare și implementare*, Editura ALL, București, 1995.
- [Melton96] Melton, J. – *SQL3 and Objects: A New Direction?*, Database Programming and Design, August 1996.
- [Miranda&Busta90] Miranda, S., Busta, J.M. – *L'art des bases de données*, vol. I-II, Eyrolles, Paris, 1990.
- Oracle Corporation – *Oracle 8 SQL Reference*, Oracle Corporation, 1997.
- Oracle Corporation – *Analytic Functions for Oracle 8i*, White Paper, Oct. 1999.
- Oracle Corporation – *Oracle 8i. SQL Reference. Release 2 (8.1.6)*, Dec. 1999.
- Oracle Corporation – *Oracle 8i. Data Warehousing Guide. Release 2 (8.1.6)*, Dec. 1999.
- [Pascu&Pascu94] Pascu, C., Pascu, A. – *Total despre... SQL. Interogarea bazelor de date*, Editura Tehnică, București, 1994.
- [Pentiuc&Duthilleul95] Pentiuc, S.G., Duthilleul, J.M. – *Elemente de teoria și proiectarea bazelor de date*, Universitatea „Ștefan cel Mare”, Suceava, 1995.
- [Popescu96] Popescu, I. – *Baze de date relaționale*, Editura Universității București, 1996.
- [Pratt&Adamski91] Pratt, P.J., Adamski, J.J. – *Database Systems. Management and Design*, Boyd & Fraser, Boston, 1991.
- [Saleh94] Saleh, I. – *Les bases de données relationnelles. Conception et réalisation*, Hermes, Paris, 1994.
- [Zemke s.a. 99-1] Zemke, F., Kulkarni, K., Witkowski, A., Lyle, B. – *Introduction to OLAP functions*, ISO/IEC JTC1/SC32 WG3: YGJ, ANSI NCITS H2-99-154, Apr. 1999.
- [Zemke s.a. 99-2] Zemke, F., Kulkarni, K., Witkowski, A., Lyle, B. – *Proposal for OLAP functions*, ISO/IEC JTC1/SC32 WG3: YGJ-nnn, ANSI NCITS H2-99-155r1, Apr. 1999.

CUPRINS

Cuvânt, oarecum, înainte	5
Conținutul lucrării	7
Capitolul 1	
NOȚIUNI ALE MODELULUI RELAȚIONAL	
1.1. Baze de date și sisteme de gestiune a bazelor de date	9
Niveluri de abstractizare a datelor	12
Sisteme de gestiune a bazelor de date	15
Limbi de definire a datelor	16
Limbi de manipulare a datelor	17
Gestionarul bazei	17
Administratorul bazei de date	18
Utilizatorii bazelor de date	19
1.2. Un pic de istorie recentă și relațională	19
1.3. Relații/tabele, domenii și atribute	24
1.4. Restricții ale bazei de date	28
Restricția de domeniu	28
Atonicitate	29
Restricția de unicitate	30
Restricția referențială	32
Restricții-utilizator	33
1.5. Schema și conținutul unei baze de date. Exemplu	34
Conținuturi suplimentare privind restricțiile referențiale	42
1.6. Alte noțiuni ale SGBD-urilor relaționale	44
Tabele virtuale (view-uri)	44
Proceduri stocate	46
1.7. Regulile modelului relațional	47

Analog pot fi create declanșatoare pentru modificări și ștergeri ale linilor din tabela virtuală.

Toate informațiile despre declanșatoarele tabelelor sau tabelelor virtuale din schema curentă, inclusiv corpul lor (Blocul PL/SQL), pot fi obținute din tabela USER_TRIGGERS a catalogului-sistem. Spre exemplu, situația trigger-elor taleei LINIIFACT este vizualizată astfel:

```
*SELECT *
FROM USER_TRIGGERS
WHERE TABLE_NAME = 'LINIIFACT'
```

BIBLIOGRAFIE

- [Adiba&Delobel82] Adiba, M., Delobel, C. – *Bases de données et systèmes relationnels*, Éditions Dunod, Paris, 1982.
- [Atkinson §.a. 89] Atkinson, M., Bancilhon, F., DeWitt, D., Dittrich, K., Maier, D., Zdonick, D. – *The Object-Oriented Database System Manifesto*, International Conference on Deductive and Object-Oriented Databases, Kyoto, 1989.
- [Beech97] Beech, D. – *Can SQL Be Simplified?*, SQL3 discussion paper, March, 3, 1997.
- [Bobrowski01] Bobrowski, S. – „Creating Updatable Views”, *Oracle Magazine*, March-April 2001.
- [Boyce §.a. 75] Boyce, R.F., Chamberlin, D.D., King, W.F., Hammer, M.M. – „Specifying Queries as Relational Expressions: The SQUARE Data Sublanguage”, *Comm. ACM* 18, No. 11, nov. 1975.
- [Celko99] Celko, J. – *Joe Celko's Data and Databases: Concepts in Practice*, Morgan Kaufmann, San Francisco, 1999.
- [Chamberlin&Boyce74] Chamberlin, D., Boyce, R.F. – *SEQUEL: A Structured English Query Language*, ACM – SIGMOD Workshop on Data Description, Access, and Control, Ann Arbor, Michigan, May 1974.
- [Chamberlin §.a. 76] Chamberlin, D. §.a. – „SEQUEL 2: Definition, Manipulation and Control”, *IBM J. Research and Development*, 20, No. 6, nov. 1976.
- [Chamberlin80] Chamberlin, D. – *A Summary of User Experience with the SQL Data Sublanguage*, Proc. of International Conference on Databases, Aberdeen, July 1980.
- [Chamberlin98] Chamberlin, D. – „Super Groups”, *DB2 Magazine On Line*, Winter 1998.
- [CODASYL69] CODASYL Systems Committee – *A Survey of Generalized Database Management Systems*, Technical Report, May 1969.
- [Codd69] Codd, E.F. – *Derivability, Redundancy and Consistency of Relations Stored in Large Data Banks*, IBM Research Report RJ 599, August 1969.
- [Codd70] Codd, E.F. – *A Relational Model of Data for Large Shared Data Banks*, ACM, vol. 13, No. 6, 1970.
- [Connolly §.a. 96] Connolly, T.M., Begg, C.E., Strachan, A.D. – *Database Systems. A Practical Approach to Design, Implementation and Management*, Addison-Wesley, Harlow, 1996.
- [Date86] Date, C.J. – *An Introduction to Database Systems*, Addison-Wesley, Reading, Massachusetts, 4th edition, 1986.
- [Date98] Date, C.J. – „Thirty Years of Relational. The Birth of Relational Model”, *Intelligent Enterprise*, Oct., Nov., Dec. 1998.
- [Date99-1] Date, C.J. – „Thirty Years of Relational. Codd's Relational Algebra”, *Intelligent Enterprise*, May 1999.
- [Date99-2] Date, C.J. – „Thirty Years of Relational: Relational Forever”, *Intelligent Enterprise*, June 1999.
- [Delobel §.a. 91] Delobel, C., Leclus, C., Richard, P. – *Bases de données. De systèmes relationnels aux systèmes à objets*, InterEditions, Paris, 1991.
- [Dodescu §.a. 87] Dodescu, G. §.a. – *Informatica*, Editura Științifică și Enciclopedică, București, 1987.
- [Dollinger98] Dollinger, R. – *Baze de date și gestiunea tranzacțiilor*, Editura Albastră, Cluj-Napoca, 1998.
- [Everest86] Everest, G.C. – *Database Management. Objectifs, System Functions, and Administration*, McGraw-Hill, New York, 1986.
- [Florescu §.a. 99] Florescu, V., Stanciu, V., Cozgarea, G., Cozgarea, A. – *Baze de date*, Editura Economică, București, 1999.
- [Fortier99] Fortier, P. – *SQL-3. Implementing the Object-Relational Database*, McGraw-Hill, New York, 1999.

În noua versiune a declanșatorului TRG_FACTURI_UPD_AFTER_ROW se memorează, în variabilele publice v_codcl_NEW și v_codcl_OLD, vechea și nouă valoarea a atributului CodCl. Aceste două valori sunt folosite în noul declanșator, cel de la nivel de comandă, în care scăpăm de problema „mutanței”.

Listing 8.21. Varianta 2 a declanșatorului TRG_FACTURI_UPD_AFTER_ROW

```
CREATE OR REPLACE TRIGGER trg_facturi_upd_after_row
AFTER UPDATE ON facturi
FOR EACH ROW
DECLARE
    v_vinzari facturi.valtotala%TYPE ;
    v_incasari facturi.valtotala%TYPE ;
BEGIN
    variabile_globale.v_trg_facturi_upd_after_row := TRUE ;
    variabile_globale.v_codcl_OLD := :OLD.codcl ;
    variabile_globale.v_codcl_NEW := :NEW.codcl ;
    IF :NEW.valtotala <> :OLD.valtotala AND NOT
        variabile_globale.v_trg_liniifact_upd_after_row THEN
        variabile_globale.v_trg_facturi_upd_after_row := FALSE ;
        RAISE_APPLICATION_ERROR(-20501,
            'Este interzisă modificarea directă a ValTotala !');
    END IF ;
    IF :NEW.nrfact <> :OLD.nrfact THEN
        UPDATE liniifact SET nrfact = :NEW.nrfact
        WHERE nrfact = :OLD.nrfact ;
        UPDATE incasfact SET nrfact = :NEW.nrfact
        WHERE nrfact = :OLD.nrfact ;
    END IF ;
    variabile_globale.v_trg_facturi_upd_after_row := FALSE ;
END ;
```

În final, listingul 8.22 conține scriptul pentru crearea declanșatorului la nivel de comandă:

Listing 8.22. Crearea declanșatorului TRG_FACTURI_UPD_AFTER_STA

```
CREATE OR REPLACE TRIGGER trg_facturi_upd_after_sta
AFTER UPDATE ON facturi
DECLARE
    v_vinzari facturi.valtotala%TYPE ;
    v_incasari facturi.valtotala%TYPE ;
BEGIN
    variabile_globale.v_trg_facturi_upd_after_sta := TRUE ;
    IF variabile_globale.v_codcl_OLD <>
        variabile_globale.v_codcl_NEW THEN
        SELECT SUM(cantitate * pretunit * (1 + proctva))
        INTO v_vinzari
        FROM facturi f, liniifact lf, produse p
```

```
WHERE f.nrfact=lf.nrfact AND lf.codpr=p.codpr AND
    codcl = variabile_globale.v_codcl_NEW ;
    SELECT SUM(transa)
    INTO v_incasari
    FROM facturi f, incasfact i
    WHERE f.nrfact=i.nrfact AND codcl =
        variabile_globale.v_codcl_NEW ;
    IF v_vinzari - v_incasari > 10000000 THEN
        variabile_globale.v_trg_facturi_upd_after_sta := FALSE ;
        RAISE_APPLICATION_ERROR(-20502,
            'Clientul e prea mare datornic !');
    END IF ;
    variabile_globale.v_trg_facturi_upd_after_sta := FALSE ;
END ;
```

Declanșatoare de tip INSTEAD OF în Oracle 8i

După cum promiteam în paragraful dedicat tabelelor derivate, în Oracle 8i problemele legate de actualizarea acestora și, implicit, propagarea modificărilor în tabelele persistente, se pot rezolva mai mult decât onorabil utilizând declanșatoare INSTEAD OF. Revenim la tabela virtuală vLOCJUDIASI creată prin joncțiunea tabelelor JUDEȚE și LOCALITĂȚI. Inserarea unei linii în view înseamnă, în primul rând, adăugarea unei noi localități. În plus, trebuie verificat dacă valoarea atributului Jud există în JUDEȚE. Dacă nu, atunci este vorba despre un județ nou-nou și se inserează o nouă linie în JUDEȚE. Iată, în listingul 8.23, blocul PL/SQL de creare a declanșatorului ce realizează aceste operațuni.

Listing 8.23. Bloc PL/SQL de creare a declanșatorului TRG_VLOCJUDIASI_INS_INSTEAD

```
CREATE OR REPLACE TRIGGER trg_vlocjudiasi_ins_instead
INSTEAD OF INSERT ON vlocjudiasi
DECLARE
    jud JUDETE.Jud%TYPE ;
BEGIN
    SELECT Jud
    INTO jud_
    FROM judete
    WHERE jud = :NEW.Jud ;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            INSERT INTO judete VALUES (:NEW.Jud, :NEW.Judet,
                :NEW.RegIune) ;
    END ;
    INSERT INTO localitati VALUES (:NEW.CodPost, :NEW.Loc,
        :NEW.Jud) ;
END ;
```

```

BEGIN
    variabile_globale.v_trg_facturi_upd_after_row := TRUE ;
    IF :NEW.valtotala <> :OLD.valtotala AND
        NOT variabile_globale.v_trg_liniifact_upd_after_row
    THEN
        variabile_globale.v_trg_facturi_upd_after_row := FALSE ;
        RAISE_APPLICATION_ERROR(-20501,
            'Este interzisa modificarea directa a ValTotala !');
    END IF ;

    IF :NEW.codcl <> :OLD.codcl THEN
        SELECT SUM(cantitate * pretunit * (1 + proctva))
        INTO v_vinzari
        FROM facturi f, liniifact lf, produse p
        WHERE f.nrfact=lf.nrfact AND lf.codpr=p.codpr
            AND codcl=:NEW.codcl ;
        SELECT SUM(transa)
        INTO v_incasari
        FROM facturi f, incasfact i
        WHERE f.nrfact=i.nrfact AND codcl = :NEW.codcl ;
        IF v_vinzari - v_incasari > 100000 THEN
            variabile_globale.v_trg_facturi_upd_after_row :=
                FALSE ;
            RAISE_APPLICATION_ERROR(-20502,
                'Clientul e prea mare datornic !');
        END IF ;
    END IF ;

    IF :NEW.nrfact <> :OLD.nrfact THEN
        UPDATE liniifact SET nrfact = :NEW.nrfact
            WHERE nrfact = :OLD.nrfact ;
        UPDATE incasfact SET nrfact = :NEW.nrfact
            WHERE nrfact = :OLD.nrfact ;
    END IF ;
    variabile_globale.v_trg_facturi_upd_after_row := FALSE ;
END ;

Dacă la comanda
    UPDATE facturi SET valtotala = 0 WHERE nrfact=1111 ;
declanșatorul se comportă onorabil, afișând mesajul de eroare:

```

ERROR at line 1:
ORA-20501: Este interzisa modificarea directa a ValTotala !
ORA-06512: at "FOTACHEM.TRG_FACTURI_UPD_AFTER_ROW", line 9
ORA-04088: error during execution of trigger
'FOTACHEM.TRG_FACTURI_UPD_AFTER_ROW'

În schimb, la modificarea lui CodCl pentru o factură:

```
UPDATE facturi SET codcl = 1002 WHERE nrfact = 1111 ;
```

facem cunoștință cu o problemă „clasică” din Oracle:

UPDATE facturi SET codcl = 1002 WHERE nrfact = 1111

*
ERROR at line 1:

ORA-04091: table FOTACHEM.FACTURI is mutating,
trigger/function may not see it

ORA-06512: at "FOTACHEM.TRG_FACTURI_UPD_AFTER_ROW", line 13

ORA-04088: error during execution of trigger
'FOTACHEM.TRG_FACTURI_UPD_AFTER_ROW'

Tabela FACTURI nu poate fi modificată deoarece, în termeni orăclisti, este mutantă.
O tabelă este mutantă (*mutating*) dacă se află în curs de modificare printr-o comandă
INSERT, UPDATE sau DELETE sau trebuie modificată ca efect al unei restricții
referențiale de tip DELETE CASCADE. O tabelă este restricționată (*constraining*) dacă
declanșatorul o accesează direct, printr-o comandă SQL, sau indirect, în virtutea unei
restricții referențiale.

Pentru toate declanșatoarele de la nivel de linie și cel de la nivel de comandă (*statement*)
lansat ca urmare a unei ștergeri în cascădă (prin opțiunea DELETE CASCADE de la
CREATE/ALTER TABLE) există câteva restricții menite să prevină situațiile de inconsis-
tență a datelor:

- intr-un declanșator nu poate fi citită sau modificată o tabelă mutantă (în curs de
modificare) datorită *trigger-ului*. Este ceea ce ni s-a întâmplat în FACTURI, când
declanșatorul citește tabela pentru calculul valorilor facturate și incasate ale noului
client.
- intr-un *trigger* nu pot fi modificate atributele ce alcătuiesc cheile primară și străine,
precum și cele pentru care a fost declarată *restricția de unicitate*, atribută unei tabele
restricționate (pentru *trigger-ul* respectiv). Excepție fac *trigger-ele* de inserare la nivel
de linie prin care se adaugă o singură înregistrare.

Pentru a rezolva problema apărută, modificăm corpul declanșatorului de mai sus și
crem un declanșator la nivel de comandă. Mai înainte, în listingul 8.20 este prezentat
pachetul variabilelor publice.

Listing 8.20. Crearea pachetului cu variabile publice

```

CREATE OR REPLACE PACKAGE variabile_globale
IS
    v_trg_facturi_upd_befo_sta BOOLEAN := FALSE ;
    v_trg_facturi_upd_befo_row BOOLEAN := FALSE ;
    v_trg_facturi_upd_after_row BOOLEAN := FALSE ;
    v_trg_facturi_upd_after_sta BOOLEAN := FALSE ;
    v_trg_liniifact_upd_befo_sta BOOLEAN := FALSE ;
    v_trg_liniifact_upd_befo_row BOOLEAN := FALSE ;
    v_trg_liniifact_upd_after_row BOOLEAN := FALSE ;
    v_trg_liniifact_upd_after_sta BOOLEAN := FALSE ;
    v_codcl_NEW clienti.codcl%TYPE ;
    v_codcl_OLD clienti.codcl%TYPE ;
END ;

```

```

INTO cc_
FROM DUAL ;
:NEW.codcl := cc_ ;
END ;

```

Declansatoare pentru restricții referențiale

În Oracle, cea mai mare parte a restricțiilor referențiale sunt rezolvate prin opțiunile comenzii CREATE TABLE. Astfel, ștergerea unei înregistrări-părinte, valoarea eronată a unei chei străine sunt săcționate prompt de SGBD. Principala problemă referențială rămasă nerezolvată în Oracle este actualizarea în cascadă a unui atribut-părinte în toate înregistrările-copil. Această opțiune (UPDATE CASCADE) poate fi implementată, cel puțin deocamdată, numai cu ajutorul declansatoarelor.

Spre exemplu, creăm un declansator pentru propagarea modificărilor atributului Jud din JUDEȚE în tabela LOCALITĂȚI – listingul 8.17.

Listing 8.17. Declansator pentru modificarea atributului jud în tabela JUDEȚE

```

CREATE OR REPLACE TRIGGER trg_judete_upd_jud_after_row
AFTER UPDATE OF jud ON judete
REFERENCING NEW AS NEW OLD AS OLD
FOR EACH ROW
BEGIN
    UPDATE localitati SET jud = :NEW.jud WHERE jud = :OLD.jud ;
END ;

```

Atribute actualizate prin declansatoare

Pentru comparație cu VFP, în listingul 8.18 este prezentat declansatorul pentru modificarea unei linii a tabelei LINIIFACT, declansator ce recalculează valoarea totală a facturii.

Listing 8.18. Crearea declansatorului TRG_LINIIFACT_UPD_AFTER_ROW

```

CREATE OR REPLACE TRIGGER trg_liniifact_upd_after_row
AFTER UPDATE OF NrFact, CodPr, Cantitate, PretUnit
ON liniifact
FOR EACH ROW
DECLARE
    v_proctva_OLD produse.proctva%TYPE ;
    v_proctva_NEW produse.proctva%TYPE ;
BEGIN
    variabile_globale.v_trg_liniifact_upd_after_row := TRUE ;
    SELECT ProctVA INTO v_proctva_OLD
    FROM produse WHERE codpr = :OLD.codpr ;
    IF :NEW.codpr <> :OLD.codpr THEN
        -- daca s-a schimbat codpr e posibil si procentul TVA

```

```

-- sa fie altul
SELECT ProctVA INTO v_proctva_NEW FROM produse
WHERE codpr = :NEW.codpr ;
ELSE
    -- procent TVA neschimbat
    v_proctva_NEW := v_proctva_OLD ;
END IF ;

IF :NEW.nrfact <> :OLD.nrfact THEN
    -- s-a modificat nrfact, deci trebuie decrementata
    -- Valtotala pt. factura veche si incrementata
    -- pentru factura nouă
    UPDATE facturi SET valtotala = valtotala -
    :OLD.cantitate * :OLD.PretUnit *
    (1 + v_proctva_OLD) WHERE nrfact = :OLD.nrfact ;
    UPDATE facturi SET valtotala = valtotala +
    :NEW.cantitate * :NEW.pretunit *
    (1 + v_proctva_NEW) WHERE nrfact = :NEW.nrfact ;
ELSE
    UPDATE facturi
    SET valtotala = valtotala - :OLD.cantitate *
    :OLD.PretUnit *(1 + v_proctva_OLD) +
    :NEW.cantitate * :NEW.PretUnit *
    (1 + v_proctva_NEW)
    WHERE nrfact = :NEW.nrfact ;
END IF ;
variabile_globale.v_trg_liniifact_upd_after_row := FALSE ;
END ;

```

Ca o nouă apare și variabila publică `v_trg_liniifact_upd_after_row`, care este TRUE numai pe parcursul declansatorului. Variabilă are un regim public în PL/SQL prin intermediu unui pachet (dacă apare în specificațiile pachetului). Listingul de creare a pachetului `variabile_globale` este prezentat peste câteva pagini.

Declansatoare pentru controlul actualizărilor

Păstrând similitudinea cu declansatoarele VFP, îi construim pe cel de modificare a tabeli FACTURI, în care trebuie să se interzică actualizarea atributului ValTotala altfel decât prin declansatorul `trg_liniifact_upd_after_row`.

Listing 8.19. Prima variantă a declansatorului TRG_FACTURI_UPD_AFTER_ROW

```

CREATE OR REPLACE TRIGGER trg_facturi_upd_after_row
AFTER UPDATE ON facturi
FOR EACH ROW
DECLARE
    v_vinzari facturi.valtotala%TYPE ;
    v_incasarri facturi.valtotala%TYPE ;

```

```

INTO cc_
FROM DUAL ;
:NEW.codcl := cc_ ;
END ;

```

Declanșatoare pentru restricții referențiale

În Oracle, cea mai mare parte a restricțiilor referențiale sunt rezolvate prin opțiunile comenzii CREATE TABLE. Astfel, ștergerea unei înregistrări-părinte, valoarea eronată a unei chei străine sunt săcționate prompt de SGBD. Principala problemă referențială rămasă nerezolvată în Oracle este actualizarea în cascădă a unui atribut-părinte în toate înregistrările-copil. Această opțiune (UPDATE CASCADE) poate fi implementată, cel puțin deocamdată, numai cu ajutorul declanșatoarelor.

Spre exemplu, creăm un declanșator pentru propagarea modificărilor atributului Jud din JUDEȚE în tabela LOCALITĂȚI – listingul 8.17.

Listing 8.17. Declanșator pentru modificarea atributului jud în tabela JUDEȚE

```

CREATE OR REPLACE TRIGGER trg_judete_upd_jud_after_row
AFTER UPDATE OF jud ON judete
REFERENCING NEW AS NEW OLD AS OLD
FOR EACH ROW
BEGIN
    UPDATE localitati SET jud = :NEW.jud WHERE jud = :OLD.jud ;
END ;

```

Atribute actualizate prin declanșatoare

Pentru comparație cu VFP, în listingul 8.18 este prezentat declanșatorul pentru modificarea unei linii a tabelei LINIIFACT, declanșator ce recalculează valoarea totală a facturii.

Listing 8.18. Crearea declanșatorului TRG_LINIIFACT_UPD_AFTER_ROW

```

CREATE OR REPLACE TRIGGER trg_liniifact_upd_after_row
AFTER UPDATE OF NrFact, CodPr, Cantitate, PretUnit
ON liniifact
FOR EACH ROW
DECLARE
    v_proctva_OLD produse.proctva%TYPE ;
    v_proctva_NEW produse.proctva%TYPE ;
BEGIN
    variabile_globale.v_trg_liniifact_upd_after_row := TRUE ;
    SELECT ProctVA INTO v_proctva_OLD
    FROM produse WHERE codpr = :OLD.codpr ;
    IF :NEW.codpr <> :OLD.codpr THEN
        -- daca s-a schimbat codpr e posibil si procentul TVA

```

```

-- sa fie altul
SELECT ProctVA INTO v_proctva_NEW FROM produse
WHERE codpr = :NEW.codpr ;
ELSE
    -- procent TVA neschimbat
    v_proctva_NEW := v_proctva_OLD ;
END IF ;

IF :NEW.nrfact <> :OLD.nrfact THEN
    -- s-a modificat nrfact, deci trebuie decrementata
    -- Valtotala pt. factura veche si incrementata
    -- pentru factura nouă
    UPDATE facturi SET valtotala = valtotala -
        :OLD.cantitate * :OLD.PretUnit *
        (1 + v_proctva_OLD) WHERE nrfact = :OLD.nrfact ;
    UPDATE facturi SET valtotala = valtotala +
        :NEW.cantitate * :NEW.pretunit *
        (1 + v_proctva_NEW) WHERE nrfact = :NEW.nrfact ;
ELSE
    UPDATE facturi
    SET valtotala = valtotala - :OLD.cantitate *
        :OLD.PretUnit *(1 + v_proctva_OLD) +
        :NEW.cantitate * :NEW.PretUnit *
        (1 + v_proctva_NEW)
        WHERE nrfact = :NEW.nrfact ;
END IF ;
variabile_globale.v_trg_liniifact_upd_after_row := FALSE ;
END ;

```

Ca o nouă apare și variabila publică `v_trg_liniifact_upd_after_row`, care este TRUE numai pe parcursul declanșatorului. O variabilă are un regim public în PL/SQL prin intermediul unui pachet (dacă apare în specificațiile pachetului). Listingul de creare a pachetului `variabile_globale` este prezentat peste câteva pagini.

Declanșatoare pentru controlul actualizărilor

Păstrând similitudinea cu declanșatoarele VFP, îi construim pe cel de modificare a tabeli FACTURI, în care trebuie să se interzică actualizarea atributului ValTotala altfel decât prin declanșatorul `trg_liniifact_upd_after_row`.

Listing 8.19. Prima variantă a declanșatorului TRG_FACTURI_UPD_AFTER_ROW

```

CREATE OR REPLACE TRIGGER trg_facturi_upd_after_row
AFTER UPDATE ON facturi
FOR EACH ROW
DECLARE
    v_vanzari facturi.valtotala%TYPE ;
    v_incasarri facturi.valtotala%TYPE ;

```

Dacă nici un client nu are codul mai mare de 3, un trigger de modificare la nivel de linie nu s-ar putea execuția deloc, în timp ce declanșatorul declarat a fi la nivel de comandă ar fi lansat (evident, o singură dată).

Declanșatoare de tip înainte (BEFORE) și după (AFTER) actualizare

O altă diferențiere (în bine) față de declanșatoarele VFP este posibilitatea de a defini momentul acțiunii trigger-ului. Un trigger de tip BEFORE intră în acțiune înainte de modificarea propriu-zisă și este ideal pentru verificarea unui set de condiții care ar putea bloca de la bun început tranzacția respectivă (autentificări, soldul 0 pentru un cont sau pentru un material într-o magazie etc.). Declanșatorul de tip AFTER se execută după momentul producerii actualizării, servind la verificarea corectitudinii operațiunii (nu a fost scoasă din magazie o cantitate de material mai mare decât stocul dinaintea operațiunii), inserarea unei linii într-o tabelă de tip jurnal (ce sănește evidența strictă a modificărilor operate în tabele cu informații sensibile – spre exemplu, calculul automat al costului mediu ponderat al unui material după fiecare intrare în magazie sau calculul soldului curent al unui cont bancar după fiecare depunere sau retragere etc.).

Spre deosebire de cele de tip BEFORE, trigger-ele de tip AFTER blochează linilele procesate.

Declanșatoarele la nivel de linie/comandă pot fi combinate cu cele de tip BEFORE/AFTER. Astfel, pentru orice tabelă și operație de inserare/modificare/ștergere pot fi definite patru tipuri de trigger care se execută în ordinea:

- înainte – la nivel de comandă (BEFORE – statement)
- înainte – la nivel de linie (BEFORE – row)
- după – la nivel de linie (AFTER – row)
- după – la nivel de comandă (AFTER – statement).

Triggeare de tip in-loc-de (INSTEAD OF)

Acest gen de declanșator constituie un excelent mijloc de propagare a modificărilor dintr-o tabelă derivată în tabelele de bază din care provine. Amânăm această discuție pentru finalul capitolului.

Un alt element foarte important este că și în trigger-ele la nivel de linie de tip BEFORE, și în cele de tip AFTER pot fi invocate și prelucrate atât valorile atributelor dinaintea operației, cât și cele de după operație. Prefixul este, după caz, :OLD sau :NEW (vă amintiți că în VFP aveam nevoie de funcția OLDVAL()).

Declanșatoare pentru generarea cheilor-surogat

După cum am mai discutat, în Oracle nu pot fi definite, nici pentru valorile implicate, nici pentru regulile de validare, funcții-utilizator. Lucrurile pot fi însă aranjate utilizând declanșatoare. Spre exemplu, pentru ca un client nou să primească codul următor, se poate declara un declanșator la nivel de linie prin care, înaintea inserării noii linii, valoarea CodCl să fie obținută de o manieră similară VFP – vezi listingul 8.15.

Listing 8.15. Declanșatorul pentru inserarea unei înregistrări în tabela CLIENTI – varianta 1

```
CREATE OR REPLACE TRIGGER trg_clienti_ins_befo_row
  BEFORE INSERT ON clienti
  FOR EACH ROW
DECLARE
  cc_clienti.codcl%TYPE ;
BEGIN
  SELECT MAX(codcl)
    INTO cc_
    FROM clienti ;
  :NEW.codcl := NVL(cc_,1000) + 1 ;
END ;
```

Chiar dacă comanda de inserare are forma:

```
INSERT INTO clienti VALUES (999, 'Client 8', NULL,
  NULL, '6600', NULL) ;
```

valoarea atributului CodCl în noua linie nu va fi 999, ci calculată prin declanșator.

O altă variantă a acestui declanșator utilizează secvențele. Secvența permite unei aplicații să preia numere consecutive unice pe un anumit interval. Gestiona secvenței se face la nivelul central; fiecare apel care „cere” o valoare din secvență poate fi absolut sigur că nici un alt apel nu va primi o valoare identică, deci nu vor apărea conflicte. O secvență se creează prin comanda CREATE SEQUENCE. Pentru ca noi clienti să primească coduri unice, în ordinea preluării în tabelă, se poate folosi secvența seq_clienti_codcl creată după cum urmează:

```
CREATE SEQUENCE seq_clienti_codcl
  INCREMENT BY 1
  MINVALUE 1010
  MAXVALUE 5555
  NOCYCLE
  ORDER
```

O dată creată, referința la o secvență se face prin NextVal, caz în care se obține valoarea următoare a secvenței (incrementarea se face automat), în timp ce valoarea curentă a secvenței se obține prin CurrVal.

Astfel încât, pentru ca un client nou să aibă codul imediat superior ultimului client introdus în tabelă, se folosește declanșatorul trg_clienti_ins_befo_row, care se execută automat înaintea adăugării unei noi înregistrări în tabela CLIENTI.

Listing 8.16. Declanșator pentru inserarea unei linii în tabela CLIENTI – varianta 2

```
CREATE OR REPLACE TRIGGER trg_clienti_ins_befo_row
  BEFORE INSERT ON clienti
  FOR EACH ROW
DECLARE
  cc_clienti.codcl%TYPE ;
BEGIN
  SELECT seq_clienti_codcl.NEXTVAL
```

```

*** se interzice modificarea interactiva a atributului ValTotala
valtotala_NEW = facturi.valtotala
valtotala_OLD = OLDVAL('valtotala', 'facturi')
IF valtotala_NEW # valtotala_OLD
  IF _TRIGGERLEVEL <= 1
    MESSAGEBOX('Nu puteti modifica interactiv valoarea totala ;
               a facturii !')
    RETURN .F.
  ENDIF
ENDIF
*** se propaga eventuala modificare a NrFact in tabelele copil -
*** LIINIIFACT si INCASFACT
nrfact_NEW = facturi.nrfact
nrfact_OLD = OLDVAL('nrfact', 'facturi')
IF nrfact_NEW # nrfact_OLD
  UPDATE liniifact SET NrFact = nrfact_NEW ;
  WHERE NrFact = nrfact_OLD
  UPDATE incasfact SET NrFact = nrfact_NEW ;
  WHERE NrFact = nrfact_OLD
ENDIF
ENPROC
*****

```

Toate procedurile stocate (codul acestora) se găsesc în atributul Code de pe linia asociată obiectului StoredProceduresSource.

Declanșatoare în Oracle 8i

Aria de utilizare a trigger-elor în Oracle este și mai extinsă: restricții complexe, asigurarea integrității BD, îmbunătățirea securității, respectarea integrității referențiale între nodurile unei BD distribuite, jurnalizarea operațiunilor (tranzacțiilor), sincronizarea replicilor unei BD etc.

Totuși, documentația Oracle 8 recomandă ca elanul trigger-esc să nu depășească anumite limite. Prin declanșatoare gestionate numai acele reguli imposibil de asigurat prin restricțiile la nivel de atribut (câmp), înregistrare, restricții de tip PRIMARY KEY, NOT NULL, SET DEFAULT etc. În plus, *Application Developer's Guide* specifică un număr maxim de linii ce ar trebui să alcătuiască un trigger – 60, la nevoie fiind indicată crearea unor proceduri stocate apelabile din trigger.

Logic, pentru cel considerat de mulți practicieni drept cel mai bun SGBD la ora actuală (dar și unul dintre cele mai scumpe), Oracle oferă o tipologie de declanșatoare mult mai bogată decât VFP. În VFP descriam trei tipuri, aferente celor trei operațiuni ce desemnează actualizarea unei tabele: inserare, modificare și ștergere. În funcție și de tipul de buffering ales, declanșatoarele intră în acțiune DUPĂ inserare, modificare sau ștergere, iar printr-un RETURN .F., plasat în corpul trigger-ului, operațiunea este anulată, revenindu-se la starea dinainte.

În Oracle 8 există 15 tipuri diferite de declanșatoare – vezi tabelul 8.1.

Tabelul 8.1. Tipologia „clasică” a declanșatoarelor Oracle

Eveniment declanșator	Declanșare pt. fiecare	Descriere
BEFORE INSERT	Comandă	Codul (programul) acestuia se lansează înaintea executării unei comenzi INSERT în tabela-junctivă
BEFORE INSERT	Linie	Se execută înaintea inserării fiecărei linii în tabelă
AFTER INSERT	Linie	Se execută după inserarea fiecărei linii în tabelă
AFTER INSERT	Comandă	Se lansează după execuția unei comenzi de inserare de linii în tabelă
INSTEAD OF INSERT	Linie	În loc să se insereze o linie în tabelă, se execută codul din acest declanșator
BEFORE UPDATE	Comandă	Codul acestuia se execută înaintea executării unei comenzi UPDATE pentru tabela-junctivă
BEFORE UPDATE	Linie	Se execută înaintea modificării fiecărei linii din tabelă
AFTER UPDATE	Linie	Se execută după modificarea fiecărei linii
AFTER UPDATE	Comandă	Se lansează după execuția comenzi UPDATE (după modificarea tuturor linioilor afectate de comandă)
INSTEAD OF UPDATE	Linie	În loc să se modifice o linie din tabela-junctivă, se execută codul din acest declanșator
BEFORE DELETE	Comandă	Se execută înaintea comenzi DELETE
BEFORE DELETE	Linie	Se execută înaintea ștergerea fiecărei linii
AFTER DELETE	Linie	Se execută după ștergerea fiecărei linii
AFTER DELETE	Comandă	Se lansează după execuția comenzi DELETE (după ștergerea tuturor linioilor afectate de comandă)
INSTEAD OF DELETE	Linie	Se execută în locul ștergerii unei linii din tabela-junctivă

O dată cu versiunea 8i, în Oracle au mai fost introduse o serie de declanșatoare lansabile în execuție la deschiderea instanței bazei, închiderea instanței bazei, conectarea și deconectarea utilizatorilor/aplicațiilor etc.

Declanșatoare la nivel de linie și la nivel de comandă (statement)

La definirea trigger-ului se poate stabili de câte ori este executat acesta: de fiecare dată când o linie este inserată/modificată/ștearsă, sau o singură dată pentru o comandă de actualizare, indiferent de câte linii sunt afectate.

Să luăm, spre exemplu, comanda:

```
UPDATE clienti SET codcl = codcl + 100 WHERE codcl > 3
```

Presupunând că 135 de clienți au codul mai mare decât 3, un trigger de modificare la nivel de linie se execută de 135 de ori, în timp ce unul la nivel de comandă o singură dată.

```

        WHERE CodPr = CodPr_OLD
        proctva_OLD = v(1,1)
        v = .00
        SELECT ProcTVA FROM produse INTO ARRAY v ;
        WHERE CodPr = CodPr_NEW
        proctva_NEW = v(1,1)
        IF proctva_NEW # proctva_OLD
            UPDATE facturi SET valtotala = valtotala + ;
            cantitate_NEW * pretunit_NEW * ;
            (1 + proctva_NEW) - ;
            cantitate_OLD * pretunit_OLD * ;
            (1 + proctva_OLD) ;
            WHERE nrfact = nrfact_OLD
        ENDIF
        ELSE && nu s-a schimbat CodPr
            IF cantitate_NEW # cantitate_OLD OR ;
                pretunit_NEW # pretunit_OLD
                v = .00
                SELECT ProcTVA FROM produse INTO ARRAY v ;
                WHERE CodPr = CodPr_OLD
                proctva_OLD = v(1,1)
                UPDATE facturi SET valtotala = valtotala + ;
                cantitate_NEW * pretunit_NEW * ;
                (1 + proctva_OLD) - ;
                cantitate_OLD * pretunit_OLD * ;
                (1 + proctva_OLD) ;
                WHERE nrfact = nrfact_OLD
            ENDIF
        ENDIF
    ENDEPROC
*****

```

Declanșatoare pentru controlul actualizărilor

Să imaginăm două situații în care s-ar cuveni să restricționăm operațiunile de actualizare a tabelei FACTURI. Prima ţine de interzicerea modificării atributului ValTotala altfel decât prin declanșatoarele tablei LINIIFACT. În lipsa unei asemenea opțiuni, orice utilizator poate să modifice direct (prin BROWSE, REPLACE sau UPDATE) valoarea totală a unei facturi, creându-se astfel o diferență față de informațiile din tabela LINIIFACT. Ideea este simplă: atunci când se modifică atributul ValTotala, se verifică dacă nivelul declanșatorului este mai mare decât 1, ceea ce ar constitui un indiciu că modificarea provine din unul dintre trigger-ele tablei LINIIFACT. În această situație, modificarea este autorizată. În caz contrar, interzisă.

A doua problemă se referă la o restricție încadrabilă în categoria *reguli ale afacerii* (Business Rules): se interzic noi vânzări pentru clienții care au un rest de plată mai mare de 100 milioane de lei. Problema trebuie rezolvată prin două declanșatoare, unul pentru inserare și altul de actualizare. În cele ce urmează îl prezentăm numai pe cel de actualizare.

```
CREATE TRIGGER ON facturi FOR UPDATE AS trg_upd_facturi()
```

Corpul declanșatorului este prezentat în listingul 8.14.

Listing 8.14. Declanșatorul pentru modificarea unei linii din tabela FACTURI

```

*****
PROCEDURE trg_upd_facturi
*****
LOCAL codcl_OLD, codcl_NEW, valtotala_OLD, valtotala_NEW, ;
nrfact_OLD, nrfact_NEW, vinzari_cl, incasari_cl, v
DIME v(1,1)
v = 0
*** se verifică restrictia referentială cu tabela parinte CLIENTI
codcl_NEW = facturi.codcl
codcl_OLD = OLDVAL('codcl', 'facturi')
IF codcl_NEW # codcl_OLD
    IF _TRIGGERLEVEL <= 1   && modificarea este locală
        IF !INDEXSEEK(codcl_NEW, .F., 'clienti', 'codcl')
            MESSAGEBOX('Noua valoare a atributului CodCl -';
+LTRIM(STR(codcl_NEW,6)) + ;
CHR(13)+'nu există în tabela parinte CLIENTI !')
            RETURN .F.
        ENDIF
    ENDIF
ENDIF
** dacă se modifică CodCl, trebuie vazut dacă noul client are
** datorii mai mari de 100000000 (lei)
IF codcl_NEW # codcl_OLD
    v=0
    SELECT SUM(Cantitate * PretUnit * (1 + ProcTVA)) ;
    INTO ARRAY v ;
    FROM facturi f INNER JOIN liniifact lf ;
    ON f.nrfact = lf.nrfact ;
    INNER JOIN produse p ON lf.codpr = p.codpr ;
    WHERE codcl = codcl_NEW
    vinzari_cl = v(1,1)
    v=0
    SELECT SUM(Transa) ;
    INTO ARRAY v ;
    FROM facturi f INNER JOIN incasfact i ON f.nrfact = i.nrfact ;
    WHERE codcl = codcl_NEW
    incasari_cl = v(1,1)
    IF vinzari_cl - incasari_cl > 100000000
        MESSAGEBOX('Noul client are prea multe datorii și';
        'nu-i mai vinde nimic !!!')
        RETURN .F.
    ENDIF
ENDIF

```

```
*** se propaga eventuala modificare a CodPost in tabela CLIENTI
codpost_NEW = localitati.codpost
codpost_OLD = OLDVAL('codpost', 'localitati')
IF codpost_NEW # codpost_OLD
    UPDATE clienti SET codpost = codpost_NEW ;
    WHERE codpost = codpost_OLD
ENDIF
ENDPROC
```

Variabila sistem _TRIGGERLEVEL indică tocmai dacă avem de-a face cu o modificare în cascădă sau una locală a atributului Jud. Ca un alt element de nouitate, funcția sistem TALLY întoarce numărul de linii extrase prin clauza WHERE a ultimei comenzi SQL (SELECT). Pentru a accelera căutarea în tabela-părinte, se poate folosi în locul SELECT-ului funcția INDEXSEEK.

Atribute actualizate prin declanșatoare

Să luăm în discuție două spețe. În paragraful 6.5 am adăugat câmpul ValTotală tabelei FACTURI, despre care spuneam că este un atribut calculat și actualizabil prin declanșatoare. Valoarea totală a unei facturi este dată de atribuile Cantitate și PretUnit din tabela LINIIFACT și atributul ProcTVA din PRODUSE.

Trecem peste declanșatoarele tabelei PRODUSE și il prezentăm numai pe cel corespunzător actualizării fiecărei linii a tabelei LINIIFACT.

```
CREATE TRIGGER ON liniifact FOR UPDATE AS
trg_upd_liniifact()
```

Acesta are mai multe sarcini:

- verifică dacă s-a produs o modificare a atributului NrFact și dacă eventuala valoare nouă se regăsește în tabela FACTURI;
- dacă s-a modificat CodPr, se testează existența noii valori în PRODUSE;
- dacă s-a modificat Cantitate sau PretUnit, se actualizează valoarea atributului ValTotală în tabela FACTURI.

În listingul 8.13 este prezentat corpul declanșatorului:

Listing 8.13. Declanșatorul VFP pentru modificarea unei linii din tabela LINIIFACT

```
*****
PROCEDURE trg_upd_liniifact
*****
LOCAL nrfact_OLD, nrfact_NEW, codpr_OLD, codpr_NEW, v, ,
      cantitate_OLD, cantitate_NEW, pretunit_OLD, pretunit_NEW, ,
      proctva_OLD, proctva_NEW
cantitate_OLD = OLDVAL('cantitate', 'liniifact')
cantitate_NEW = liniifact.cantitate
pretunit_OLD = OLDVAL('pretunit', 'liniifact')
pretunit_NEW = liniifact.pretunit
DIME v(1,1)
```

```
*** se verifica restrictia referentiala cu tabela parinte FACTURI
nrfact_NEW = liniifact.nrfact
nrfact_OLD = OLDVAL('nrfact', 'liniifact')
IF nrfact_NEW # nrfact_OLD
    IF _TRIGGERLEVEL <= 1   && modificarea este locala
        IF !INDEXSEEK(nrfact_NEW, .F., 'facturi', 'nrfact')
            MESSAGEBOX('Noua valoare a atributului NrFact - ';
                        +LTRIM(STR(nrfact_NEW,8)) + ;
                        CHR(13)+'nu exista in tabela parinte FACTURI !')
            RETURN .F.
        ENDIF
    ENDIF
ENDIF

*** se verifica restrictia referentiala cu tabela parinte PRODUSE
codpr_NEW = liniifact.codpr
codpr_OLD = OLDVAL('codpr', 'liniifact')
IF codpr_NEW # codpr_OLD
    IF _TRIGGERLEVEL <= 1   && modificarea este locala
        IF !INDEXSEEK(codpr_NEW, .F., 'produse', 'codpr')
            MESSAGEBOX('Noua valoare a atributului CodPr - ';
                        +LTRIM(STR(codpr_NEW,6)) + ;
                        CHR(13)+'nu exista in tabela parinte PRODUSE !')
            RETURN .F.
        ENDIF
    ENDIF
ENDIF

*** Daca s-a schimbat numarul facturii, linia trebuie scoasa de la
* fosta factura si adaugata la noua factura
IF nrfact_NEW # nrfact_OLD AND _TRIGGERLEVEL <= 1
    v = .00
    SELECT ProcTVA FROM produse INTO ARRAY v ;
    WHERE CodPr = CodPr_OLD
    proctva_OLD = v(1,1)
    v = .00
    SELECT ProcTVA FROM produse INTO ARRAY v ;
    WHERE CodPr = CodPr_NEW
    proctva_NEW = v(1,1)
    * se scade valoarea de la factura "veche"
    UPDATE facturi SET valtotala = valtotala - ;
        cantitate_OLD * pretunit_OLD * (1 + proctva_OLD) ;
    WHERE nrfactura = nrfactura_OLD
    * se aduna valoarea de la factura "noua"
    UPDATE facturi SET valtotala = valtotala + ;
        cantitate_NEW * pretunit_NEW * (1 + proctva_NEW) ;
    WHERE nrfactura = nrfactura_NEW
ELSE ** nu s-a modificat NrFact
    ** Daca s-a modificat codul produsului, deci trebuie vazut
    ** daca procentul TVA este acelasi
    IF codpr_NEW # codpr_OLD
        v = .00
        SELECT ProcTVA FROM produse INTO ARRAY v ;
```

Să începem cu tabela **JUDEȚE**. Stergerea unui județ poate aduce necazuri dacă există localități în acel județ (mă refer la baza de date...). Așa încât este necesar un declanșator de stergere care să verifice dacă eliminarea înregistrării poate avea loc sau nu. Comanda de creare a declanșatorului este:

```
CREATE TRIGGER ON judete FOR DELETE AS trg_del_judete()
```

Iată, în listingul 8.9, corpul declanșatorului.

Listing 8.9. Varianta 1 a declanșatorului TRG_DEL_JUDETE

```
*****  
PROCEDURE trg_del_judete  
*****  
LOCAL jud, v  
dime v(1,1)  
v = ''  
jud_ = judete.jud  
SELECT MIN(loc) FROM localitatii INTO ARRAY v WHERE jud = jud_  
IF v(1,1) # ''  
    MESSAGEBOX('Există cel puțin o linie copil în '  
        tabela LOCALITATI.'+',  
        CHR(13)+'Exemplu: loc. '+ALLTRIM(v(1,1)))  
    RETURN .F.  
ENDIF  
ENDPROC  
*****
```

SQL beneficiază de câteva optimizări ale motorului bazei date. Cu toate acestea, SELECT-ul din declanșator parcurge întreaga tabelă LOCALITĂȚI, așa încât pare mai rapidă varianta următoare – ce-i drept, mai puțin SQL-istă.

Listing 8.10. Varianta 2 a declanșatorului TRG_DEL_JUDETE

```
*****  
PROCEDURE trg_del_judete  
*****  
LOCAL jud  
jud_ = judete.jud  
IF INDEXSEEK(jud_, .F., 'localitatii', 'jud')  
    MESSAGEBOX('Există cel puțin o linie copil în '  
        tabela LOCALITATI !')  
    RETURN .F.  
ENDIF  
ENDPROC  
*****
```

Funcția INDEXSEEK căută în tabela LOCALITĂȚI, fără a deplasa pointerul pentru înregistrări, prima linie pentru care cheia indexului JUD (cheie care este atributul cu același nume) are aceeași valoare ca a variabilei jud_. Când căutarea este încununată de succes, valoarea returnată este TRUE (.T. în notația VFP), altminteri, FALSE.

Modificarea indicativului unui județ trebuie să se propage și în tabela LOCALITĂȚI, ocazie bună pentru încă un declanșator:

```
CREATE TRIGGER ON judete FOR UPDATE AS trg_upd_judete()
```

Corpul declanșatorului este prezentat în listingul 8.11. Valoarea dinaintea modificării se obține în VFP prin funcția OLDVAL().

Listing 8.11. Declanșator pentru modificarea unei linii din tabela JUDEȚE

```
*****  
PROCEDURE trg_upd_judete  
*****  
LOCAL jud_OLD, jud_NEW  
jud_NEW = judete.jud  
jud_OLD = OLDVAL('jud', 'judete')  
IF jud_NEW # jud_OLD  
    UPDATE localitatii SET jud = jud_NEW WHERE jud = jud_OLD  
ENDIF  
ENDPROC
```

La inserarea unei noi linii într-o tabelă-copil sau la schimbarea valorii unei chei străine, trebuie verificat dacă noile valori se regăsesc în tabela-părinte. Spre exemplu, dacă în tabela LOCALITĂȚI se modifică valoarea atributului Jud, această nouă valoare trebuie căutată în JUDEȚE. Se poate crea, în acest scop, un declanșator:

```
CREATE TRIGGER ON localitatii FOR UPDATE ,  
AS trg_upd_localitatii()
```

Trebuie, însă, să avem în vedere că modificarea atributului Jud din LOCALITĂȚI poate fi și consecința declanșatorului trg_upd_judete. Așa încât, pentru a evita circularitatea, se verifică dacă modificarea este una locală sau provine din declanșatorul cu pricina – vezi listingul 8.12.

Listing 8.12. Declanșator TRG_UPD_LOCALITATI

```
*****  
PROCEDURE trg_upd_localitatii  
*****  
LOCAL jud_OLD, jud_NEW, codpost_OLD, codpost_NEW, v  
DIME v(1,1)  
*** Mai intii se verifica restrictia referentiala cu tabela parinte  
jud_NEW = localitatii.jud  
jud_OLD = OLDVAL('jud', 'localitatii')  
IF jud_NEW # jud_OLD  
    IF _TRIGGERLEVEL < 1   && modificarea este locală  
        SELECT jud FROM judete INTO ARRAY v WHERE jud = jud_NEW  
        IF _TALLY = 0  
            MESSAGEBOX('Noua valoarea a atributului Jud ,  
            - '+jud_NEW+CHR(13)+;  
            ' nu se regăseste în tabela parinte JUDETE !')  
        RETURN .F.  
    ENDIF  
ENDIF  
ENDIF
```

```

CREATE OR REPLACE FUNCTION f_incasari_cl (
    codcl_ IN clienti.codcl%TYPE ) RETURN NUMERIC
IS
    suma_ NUMERIC (16) := 0 ;
BEGIN
    SELECT SUM(transa)
    INTO suma_
    FROM FACTURI F, INCASFACT I
    WHERE F.NrFact = I.NrFact AND CodCl = CodCl_ ;
    RETURN suma_ ;
END ;
/

```

Cât privește interogarea care le folosește, nu sunt deosebiri față de VFP:

```

SELECT dencl,
    f_vinzari_cl (codcl) AS vinzari,
    f_incasari_cl (codcl) AS incasari,
    f_vinzari_cl (codcl) - f_incasari_cl (codcl)
        AS rest_de_plata
FROM CLIENTI

```

Ar mai fi de adăugat că pentru a afla numele și alte informații despre funcțiile și procedurile stocate din schema curentă, se poate interoga catalogul sistem:

```

SELECT *
FROM USER_OBJECTS
WHERE OBJECT_TYPE = 'FUNCTION'
respectiv:
SELECT *
FROM USER_OBJECTS
WHERE OBJECT_TYPE = 'PROCEDURE'

```

În schimb, corpul funcțiilor/procedurilor este conținut în altă tabelă a dicționarului USER_SOURCE. Astfel, pentru a vizualiza conținutul funcției AF_DENCL, este necesară consultarea:

```

SELECT line, text
FROM USER_SOURCE
WHERE TYPE = 'FUNCTION' AND NAME='AF_DENCL'

```

În DB2, informații despre funcțiile stocate intr-o schemă (FOTACHEM) pot fi vizualizate prin interogarea:

```

SELECT *
FROM SYSCAT.FUNCTIONS
WHERE FUNCSCHEMA='FOTACHEM'

```

Analog stărurilor cu procedurile stocate:

```

SELECT *
FROM SYSCAT.PROCEDURES
WHERE PROCSchema='FOTACHEM'

```

8.4. Declanșatoare în VFP și Oracle

Declanșatoarele (triggere, în original) reprezintă un tip deosebit de proceduri stocate ale unei baze de date. Particularitatea lor esențială îne de faptul că, o dată create și stocate în schema bazei, acestea sunt executate automat la operațiuni de inserare, modificare sau ștergere a linilor din tabela pentru care au fost definite. La această familie a declanșatoarelor, unele SGBD-uri au mai adăugat și alte tipuri, asociate, spre exemplu, actualizărilor tabelelor derivate, conectării unei aplicații/utilizator, deschiderii și închiderii instanței baze de date etc.

Pentru lucrarea de față, problema principală a declanșatoarelor îne de faptul că acestea sunt scrise în extensiile procedurale ale SQL care prezintă diferențe sensibile de la produs la produs: PL/SQL (Oracle), DB2, Transact-SQL (SQL Server) etc. Este drept că în SQL-3 există o parte special dedicată procedurilor stocate, dar preluarea pe scară largă a specificațiilor standardului în produsele marilor actori ai pieței SGBD-urilor va lua ceva timp.

Cele mai importante avantaje ale declanșatoarelor sunt:

- permit instituirea unor reguli de validare cu mult mai complexe decât ceea ce permite clauza CHECK;
- pot ameliora sensibil mecanismul de securitate al bazei;
- permit instituirea (acolo unde nu este posibil prin clauza FOREIGN KEY) restricțiilor referențiale și modului de tratare a modificărilor ce pot cauza probleme de integritate referențială;
- constituie suportul calculării automate a valorilor unor attribute.

Într-o abordare de la simplu la complex, vom urmări câteva aspecte ale declanșatoarelor în Visual FoxPro 6 și Oracle 8i.

Declanșatoare în VFP6

Problematica declanșatoarelor în VFP6 am prezentat-o *in extenso* într-un articol publicat în PC Report împreună cu Cătălin Strimbei⁶². Fără a intra prea mult în detaliu, în VFP există trei tipuri de declanșatoare: pentru inserare, modificare și ștergerea de linii. Momentul declanșării lor depinde esențial de tipul de Buffering ales, o chestiune specifică VFP, în cele ce urmează, presupunem că opțiunea de buffering este dezactivată.

Declanșatoare pentru restricții referențiale

Mecanismul de asigurare a restricțiilor referențiale se poate implementa grafic cu ajutorul modulului Referential Integrity Builder. Paradoxal, deși există clauza FOREIGN KEY, aceasta instituie în VFP numai legături permanente între tabele. De aceea, la crearea prin program a bazei de date sunt necesare declanșatoare „manuale”, care, totuși, au avantajul că permit afișarea unor mesaje de eroare în clar sau afișarea unor restricții și explicații suplimentare.

62. [Fotache&Strimbești99-1].

Care este contribuția procentuală a fiecărui produs la totalul vânzărilor?

După cum observați, am trecut deja la OLAP. Răspunsul la această întrebare presupune folosirea interogărilor scalare în DB2 și a funcțiilor OLAP în Oracle 8i2. Varianta VFP pe care o propunem se bazează pe două funcții stocate, prezentate în listingul 8.6.

Listing 8.6. Funcții stocate pentru calculul vânzărilor pe produse și vânzărilor totale

```
*****
PROCEDURE vinz_prod
PARAMETER codpr_
LOCAL suma_
DIME suma_(1,1)
suma_ = 0
SELECT SUM(cantitate * pretunit * (1 + proctva)) ;
INTO ARRAY suma_ ;
FROM liniifact lf INNER JOIN produse p ON lf.codpr=p.codpr ;
WHERE lf.codpr = codpr_
RETURN suma_
ENDPROC
*****
PROCEDURE total_vinzari
LOCAL suma_
DIME suma_(1,1)
suma_ = 0
SELECT SUM(cantitate * pretunit * (1 + proctva)) ;
INTO ARRAY suma_ ;
FROM liniifact lf INNER JOIN produse p ON lf.codpr=p.codpr ;
RETURN suma_
ENDPROC
*****
Iată și interogarea care le folosește și furnizează răspunsul la problema formulată:
SELECT denpr AS Produs, vinz_prod (codpr) AS Vinzari, ;
total_vinzari() AS total, ;
ROUND(vinz_prod (codpr) / total_vinzari() * 100,0) ;
AS Procent ;
FROM PRODUSE
```

Care este evoluția zilnică a vânzărilor, prin raportare la ziua calendaristică anterioară?

Este nevoie de două funcții, al căror corp este prezentat în listingul 8.7.

Listing 8.7. Funcții pentru calculul vânzărilor unei zile și zilei precedente

```
*****
PROCEDURE f_vinzari_zi
PARAMETER data_
LOCAL suma_
DIME suma_(1,1)
suma_ = 0
SELECT SUM(cantitate * pretunit * (1 + proctva)) ;
```

```
INTO ARRAY suma_ ;
FROM facturi f INNER JOIN liniifact lf ;
ON f.nrfact=lf.nrfact ;
INNER JOIN produse p ON lf.codpr=p.codpr ;
WHERE datafact = data_
RETURN suma_
ENDPROC
```

```
*****
PROCEDURE f_vinzari_zi_prec
PARAMETER data_
RETURN f_vinzari_zi(data_- 1)
ENDPROC
```

În final, iată și interogarea:

```
SELECT DISTINCT datafact AS Zi, ;
f_vinzari_zi (datafact) AS Vinzari_Zi_Curenta, ;
f_vinzari_zi_prec (datafact) AS Vinzari_Zi_Precedenta, ;
f_vinzari_zi (datafact) - f_vinzari_zi_prec (datafact) ;
AS Diferenta ;
FROM FACTURI
```

Sigur, acest stil de lucru poate orișpa pe mulți ideologi și convertiți la SQL, dar până la apariția interogărilor scalare și funcțiilor OLAP în Visual FoxPro nu prea avem de ales...

Funcții stocate și fraze SELECT în Oracle

Modul de lucru prezentat mai sus funcționează și în Oracle. Este adevărat, în Oracle se poate discuta despre nivelul de puritate al funcțiilor, obligatoriu pentru cele incluse în pachete. Ca model, prezentăm în listingul 8.8 scriptul de creare a celor două funcții și echivalențele funcțiilor VFP cu aceleași nume din listingul 8.5.

Listing 8.8. Crearea în Oracle a funcțiilor stocate, utilizate în interogări

```
CREATE OR REPLACE FUNCTION f_vinzari_cl (
codcl_ IN clienti.codcl%TYPE ) RETURN NUMERIC
IS
suma_ NUMERIC (16) ;
BEGIN
SELECT SUM(cantitate * pretunit * (1 + proctva))
INTO suma_
FROM FACTURI F, LINIIFACT LF, PRODUSE P
WHERE F.NrFact = LF.NrFact AND LF.CodPr = P.CodPr
AND codcl_ = codcl ;
RETURN suma_ ;
END ;
```

```

INTO ARRAY vCodPr ;
WHERE CodPr NOT IN ;
  (SELECT CodPr ;
   FROM LINIIFACT ;
   WHERE NrFact = NrFact_) ;
GROUP BY CodPr ;
ORDER BY nr DESC
IF vCodPr_(1,1) = 0
  MESSAGEBOX('Nu se mai pot introduce produse pe aceasta ;
             factura !')
  RETURN .F.
ELSE
  RETURN vCodPr_(1,1)
ENDIF
ENDPROC

```

Cât privește regulile de validare complexe, intențiile generoase din SQL-92, de folosire a subconsultărilor în clauze CHECK, nu prea au fost preluate în SGBD-urile comerciale, însă logica acestora poate fi inclusă în totalitate în declanșatoare (triggers), pe care le vom discuta în următorul paragraf.

Poate că ar mai merita amintit că în VFP pot avea asociate reguli de validare nu numai atribute ale unor tabele, ci și ale unor tabele derivate. Astfel, dacă dorim ca în tabela derivată **vFacturi** atributul **NrFact**, Linie și **CodPr** să primească aceleași valori implicate ca și în tabela **LINIIFACT**, comenziile sunt:

```

=DBSETPROP('vFacturi.NrFact', 'FIELD', 'DefaultValue', ;
           'def_nrfact_liniifact()')
=DBSETPROP('vFACTURI.Linie', 'FIELD', 'DefaultValue', ;
           'def_linie_liniifact()')
=DBSETPROP('vFACTURI.CodPr', 'FIELD', 'DefaultValue', ;
           'def_codpr_liniifact()')

```

Functii stocate și fraze SELECT în Visual FoxPro

În afară funcțiilor și procedurilor stocate de tipul regulilor de validare, valorilor implicate sau declanșatoarelor, o facilitate importantă a VFP o constituie posibilitatea folosirii funcțiilor stocate în fraze SQL.

Să se afișeze, pentru fiecare client, valoarea vânzărilor, incasărilor și restul de plată.

La soluțiile pur neprocedurale SQL adăugăm una care presupune crearea în baza de date a două funcții, după cum urmează (listingul 8.5):

Listing 8.5. Două funcții VFP stocate, utilizate în interogări

```

*****
PROCEDURE f_vinzari_cl
PARAMETER codcl_
LOCAL suma_
DIME suma_(1,1)
suma_ = 0

```

```

SELECT SUM(cantitate * pretunit * (1 + proctva)) ;
INTO ARRAY suma_ ;
FROM facturi f INNER JOIN liniifact lf ON f.nrfact=lf.nrfact ;
  INNER JOIN produse p ON lf.codpr=p.codpr ;
WHERE codcl_ = codcl
RETURN suma_
ENDPROC
*****
PROCEDURE f_incasari_cl
PARAMETER codcl_
LOCAL suma_
DIME suma_(1,1)
suma_ = 0
SELECT SUM(transa) ;
INTO ARRAY suma_ ;
FROM facturi f INNER JOIN incasfact i ON f.nrfact=i.nrfact ;
WHERE codcl_ = codcl
RETURN suma_
ENDPROC
*****
```

Cele două funcții pot fi folosite în orice frază SELECT după cum urmează:

```

SELECT dencl, ;
  f_vinzari_cl (codcl) AS vinzari, ;
  f_incasari_cl (codcl) AS incasari, ;
  f_vinzari_cl (codcl) - f_incasari_cl (codcl) ;
  AS rest_de_plata ;
FROM CLIENTI

```

Care este cel mai mare datornic dintre clienți?

Este una dintre problemele cele mai dificile formulate în capitolul 5, la care în VFP, singurele soluții se bazau pe salvarea rezultatelor intermediare în curse. Iată o altă interogare, mult mai simplă, care se folosește de aceleași două proceduri stocate:

```

SELECT dencl, ;
  f_vinzari_cl (codcl) AS vinzari, ;
  f_incasari_cl (codcl) AS incasari, ;
  f_vinzari_cl (codcl) - f_incasari_cl (codcl) ;
  AS rest_de_plata ;
FROM CLIENTI ;
WHERE f_vinzari_cl (codcl) - f_incasari_cl (codcl) = ;
  (SELECT MAX(f_vinzari_cl (codcl) - ;
  f_incasari_cl (codcl)) ;
FROM CLIENTI)

```

Nu este neapărat ca funcțiile de tipul **f_vinzari_cl** și **f_incasari_cl** să fie stocate. Frazele SELECT pot fi incluse în programe ce includ și descrierea funcțiilor-argument. În acest mod se evită aglomerarea containerului (dicționarului) bazei.

```

IF vCodCl(1,1) = 0
  RETURN 1001
ELSE
  RETURN vCodCl(1,1) + 1
ENDIF
ENDPROC

```

Spre deosebire de alte SGBD-uri, în VFP funcțiile ce întorc valori implicate pot fi oricără de sofisticate. Lucrul acesta nu trebuie exagerat, deoarece poate afecta viteza de lucru a aplicației. Ne fixăm ca obiectiv crearea unei funcții stocate care să determine cel mai frecvent client, adică acel client care apare pe cele mai multe facturi, iar codul acestuia să constituie valoarea implicită pentru atributul CodCl din tabela FACTURI:

```

ALTER TABLE FACTURI ;
  ALTER COLUMN CodCl SET DEFAULT def_codcl_facturi()

```

În listing 8.3 se află corpul funcției (în procedurile stocate ale bazei de date):

Listing 8.3. Corpul procedurii stocate VFP6

```

*****
PROCEDURE def_codcl_facturi
*****
LOCAL vCodCl
DIME vCodCl(1,2)
vCodCl = 0
SELECT TOP 1 CodCl, COUNT(*) AS nr :
FROM FACTURI ;
INTO ARRAY vCodCl ;
GROUP BY CodCl ;
ORDER BY nr DESC
IF vCodCl(1,1) = 0
  MESSAGEBOX('Nu se pot introduce facturi, '+CHR(13)+;
             'cita vreme nu exista nici un client !')
  RETURN .F.
ELSE
  RETURN vCodCl(1,1)
ENDIF
ENDPROC

```

Intrând și mai mult în detaliu, luăm în discuție tabela LINIIFACT. Cheia primară a acesteia este combinația (NrFact, Linie). Trei attribute sunt susceptibile de a se propozi cu valori implicate calculate prin funcții stocate, NrFact, Linie și CodPr. Logica acestora este diferită. Spre exemplu, este de presupus că factura curentă, pentru care se introduc linii, este cea mai recentă (are cel mai mare număr). Numărul liniei trebuie incrementat, iar produsul propus de funcția stocată trebuie să fie cel mai frecvent facturat, însă nu trebuie să violeze unicitatea combinației (NrFact, CodPr)!

```

ALTER TABLE LINIIFACT ALTER COLUMN NrFact ,
  SET DEFAULT def_nrfact_liniifact()
ALTER TABLE LINIIFACT ALTER COLUMN Linie ;
  SET DEFAULT def_linie_liniifact()
ALTER TABLE LINIIFACT ALTER COLUMN CodPr ;
  SET DEFAULT def_codpr_liniifact()

```

Listingul 8.4. prezintă codul acestor trei funcții stocate:

Listing 8.4. Alte trei proceduri stocate VFP6

```

*****
PROCEDURE def_nrfact_liniifact
*****
LOCAL vNrFact_
DIME vNrFact_(1)
vNrFact_ =
* se selectează cea mai recentă factura
SELECT MAX(NrFact) ;
FROM FACTURI ;
INTO ARRAY vNrFact_
IF vNrFact_(1,1) = 0
  MESSAGEBOX('Nu se pot introduce linii dacă nu există nici o;
             factura !')
  RETURN .F.
ELSE
  RETURN vNrFact_
ENDIF
ENDPROC
*****
PROCEDURE def_linie_liniifact
*****
LOCAL vLinie_, NrFact_
DIME vLinie_(1,1)
vLinie_ = 0
NrFact_ = LINIIFACT.NrFact
* se extrage ultima linie introdusa in factura curentă
SELECT MAX(Linie) ;
FROM LINIIFACT ;
INTO ARRAY vLinie ;
WHERE NrFact = NrFact_
IF vLinie_(1,1) = 0      && e prima linie din factura
  RETURN 1
ELSE                      && se incrementeaza
  RETURN vLinie_(1,1) + 1
ENDIF
ENDPROC
*****
PROCEDURE def_codpr_liniifact
*****
LOCAL vCodPr_, NrFact_
DIME vCodPr_(1,2)
vCodPr_ = 0
NrFact_ = LINIIFACT.NrFact
** se selectează cel mai frecvent produs, exceptându-le pe cele care
** ar viola unicitatea combinației (NrFact, CodPr)
SELECT TOP 1 CodPr, COUNT(*) AS nr ;
FROM LINIIFACT ;

```

```

* Pe baza tăbelei derivate, se va actualiza numai LINIIFACT
DBSETPROP('vFacturi', 'View', 'Tables', 'LINIIFACT')
* Se declara atributele cheie (primara)
DBSETPROP('vFacturi.NrFact', 'Field', 'KeyField', .t.)
DBSETPROP('vFacturi.Linie', 'Field', 'KeyField', .t.)
* Se declara ca actualizabile atributul cheie primara
DBSETPROP('vFacturi.NrFact', 'Field', 'Updatable', .t.)
DBSETPROP('vFacturi.Linie', 'Field', 'Updatable', .t.)
* Tipul modificarilor (UPDATE, nu INSERT-DELETE)
DBSETPROP('vFacturi', 'View', 'UpdateType', DB_UPDATE)
* La propagarea actualizarii in tăbele de baza, se verifica
* daca au intervenit modificarri in continutul atributelor cheie
* si a celor actualizabile
DBSETPROP('vFacturi', 'View', 'WhereType', DB_KEYANDUPDATABLE)
* Semnalul final pentru propagarea modificarilor din
* tăbela derivata in cea de baza
DBSETPROP('vFacturi', 'View', 'SendUpdates', .t.)
RETURN

```

O dată creată, tăbela derivată trebuie deschisă explicit prin comanda USE, operațiune în urma căreia are loc execuția frazei SELECT și popularea cu înregistrări. De remarcat că împotrâptarea tăbelei derivate cu cele mai noi înregistrări ale tăbelelor de bază nu necesită închiderea și deschiderea sa, funcția REQUERY() asigurând acest lucru.

Afișarea tăbelelor derivate din baza de date se obține asemănător tăbelelor „normale”:

```
SELECT * FROM vinzari.dbc WHERE OBJECTTYPE='View'
```

Atributul Property conține fraza SELECT pe baza căreia a fost construită tăbela virtuală. Cât despre obținerea datelor despre atributele tăbelelor derivate, iată interogarea:

```

SELECT v2.objectname, v1.objectid, v1.objectname, ;
       left(v1.property, 200) ;
FROM vinzari.dbc vl INNER JOIN vinzari.dbc v2 ;
ON vl.parentid=v2.objectid ;
WHERE v2.OBJECTTYPE='View' AND v1.OBJECTTYPE='Field' ;
ORDER BY v1.objectid

```

8.3. Proceduri și funcții stocate în VFP și Oracle

Dacă procedura sau funcția reprezintă noțiuni vânтурate de multe decenii de informaticienii cu sau fără acte în regulă, asocierea acestora cu persistența a fost consacrată de SGBD-urile deceniului 9 al tocmai încheiatului secol. Stocarea se referă la „înăuntrul” bazei, în dicționarul de date sau catalogul sistemului, cum își spune la case (de soft) mai mari.

Deși puternic impregnate cu SQL, funcțiile și procedurile stocate sunt, după cum bine le zice numele, cam... procedurale, depinzând într-o măsură decisivă de extensiile proprietare

ale fiecărui SGBD. Dintre procedurile stocate, în acest paragraf ne ocupăm de funcțiile ce întorc valori implicate pentru atribut, proceduri/funcții de validare, iar în paragraful următor, de un tip special de proceduri stocate, și anume declanșatoarele (trigger-ele).

Valori implicate

O problemă importantă a bazelor de date o reprezintă cheile-surogat, valori ale unor atribut care nu au legătură directă cu realitatea, dar ajută la identificarea de o manieră unică a linilor unei tăbele. Spre exemplu, în tăbela CLIENTI, atributul CodCl este o cheie-surogat, fiind un număr unic atribuit fiecărui client, fără o logică anume. La fel putem spune despre CodPr din tăbela PRODUSE și CodInc din INCASARI. Din contra, CodPost din LOCALITĂȚI, Jud din JUDEȚE, CNP din PERSOANE, NrFact sunt informații cu alt grad de relevanță și standardizare.

Modul în care se pot genera valori implicate pentru atribut de tip cheie-surogat diferă de la caz la caz. În Oracle, valorile secvențiale unice pot fi generate printr-un tip special de obiecte ale bazei denumite „secvențe”, dar care nu pot fi referite în clauza DEFAULT, ci numai în declanșatoare.

DB2-ul permite, la crearea unei tăbele, declararea de câmpuri autoincrementante, astfel:

```

CREATE TABLE clienti (
  codcl DECIMAL(6) NOT NULL
    GENERATED ALWAYS AS IDENTITY
    (START WITH 1000, INCREMENT BY 1),
  dencl VARCHAR(30)
)
;
```

Orice linie inserată ulterior în tăbela CLIENTI va primi, pentru atributul CodCl, valoarea următoare, obținută prin secvența declarată prin clauzele START WITH - INCREMENT BY.

Poate paradoxal, Visual FoxPro este mult mai generos în materie de funcții stocate pentru calculul valorilor implicate ale unor atribut. Pentru atributul CodCl din tăbela CLIENTI se dorește ca valorile implicate să fie „returnate” de funcția def_codcl_clienti(). Comanda declarativă este:

```

ALTER TABLE CLIENTI ALTER COLUMN CodCl ;
  SET DEFAULT def_codcl_clienti()

```

Procedurilor stocate ale bazei de date VÂNZĂRI (al cărei program de creare a fost discutat în capitolul 3) li se adaugă și liniile din listingul 8.2.

Listing 8.2. Procedura stocată VFP6 DEF_CODCL_CLIENTI

```

*****
PROCEDURE def_codcl_clienti
*****
LOCAL vCodCl
DIME vCodCl(1,1)
vCodCl(1,1) = 0
SELECT MAX(CodCl) FROM clienti INTO ARRAY vCodCl

```

Prin clauza **CASCADE** se sterg atât tabela virtuală curentă, cât și toate cele create pe baza acesteia, în timp ce **RESTRICT** interzice operațiunea atât timp cât există măcar un **view** construit pe baza tabelii virtuale curente.

Tabele virtuale în Oracle

În Oracle, aflarea atât a frazei **SELECT** de creare, cât și a modului în care o tabelă derivată poate fi actualizată presupune consultarea dicționarului de date. Spre exemplu, interrogarea prin care a fost creată **vJudeteMoldova** se vizualizează prin:

```
SELECT TEXT
FROM USER_VIEWS
WHERE VIEW_NAME = 'VJUDETEMOLDOVA'
```

iar interrogarea:

```
SELECT column_name, updatable, insertable, deletable
FROM user_updatable_columns
WHERE table_name = 'VJUDETEMOLDOVA'
```

va determina afișarea rezultatului din figura 8.3.

COLUMN_NAME	UPDATABLE	INSERTABLE	DELETABLE
JUD	YES	YES	YES
JUDET	YES	YES	YES
REGIUNE	YES	YES	YES

Figura 8.3. Posibilități de actualizare în tabela derivată **vJudeteMoldova**

Rezultă că asupra acestei tabele derivate poate fi operată totă gama de operațiuni pentru toate atributele. În schimb, dacă pe baza acestia creăm o altă pentru vizualizarea și actualizarea localităților din județul Iași:

```
CREATE VIEW vLocJudIasi AS
  SELECT localitati.codpost, localitati.loc,
         localitati.jud, judet, regiune
    FROM localitati, vJudeteMoldova
   WHERE localitati.jud = vJudeteMoldova.jud AND
         localitati.jud='IS'
  WITH CHECK OPTION
```

situația se prezintă diferit – vezi figura 8.4.

COLUMN_NAME	UPDATABLE	INSERTABLE	DELETABLE
CODPOST	YES	YES	YES
LOC	YES	YES	YES
JUD	NO	NO	NO
JUDET	NO	NO	NO
REGIUNE	NO	NO	NO

Figura 8.4. Posibilități de actualizare în tabela derivată **vLocJudIasi**

Prin urmare, orice tentativă de a modifica ultimele trei atrbute va eşua. Ba mai mult, nici inserarea nu funcționează. Comanda:

```
INSERT INTO vLocJudIasi VALUES ('6868', 'Tg. Frumos', 'IS',
                                  'Iasi', 'Moldova')
```

va genera mesajul de eroare: ORA-01733: virtual column not allowed here. Chiar dacă schimbăm definiția în:

```
CREATE OR REPLACE VIEW vLocJudIasi AS
  SELECT localitati.codpost, localitati.loc,
         localitati.jud, judet, regiune
    FROM localitati, judete
   WHERE localitati.jud = judete.jud AND localitati.jud='IS'
  WITH CHECK OPTION
```

rezultatul va fi identic⁶⁰. Prin urmare, în **vLocIasi** este posibilă numai modificarea atrbutorilor **CodPost** și **Loc**. Aceste probleme majore de actualizare în Oracle 8i pot fi rezolvate cu ajutorul declanșatoarelor, după cum vom vedea peste cîteva pagini.

Tabele virtuale în Visual FoxPro

Visual FoxPro este un produs generos în ceea ce privește tabelele derivate. Acestea pot fi create atât grafic, cât și prin program. Important este că pot fi definite explicit atrbutele care pot fi actualizate sau nu, modul de actualizare a tabelelor, ba chiar, mai mult, se pot defini îndepărtă și reguli de validare chiar la nivel de **view**⁶¹. Programul prezentat în listingul 8.1 este grăitor în acest sens.

Listing 8.1. Program Visual FoxPro 6 de creare a unei tabele derivate

```
*** CREAREA TABELEI DERIVATE vFacturi
#INCLUDE foxpro.h
IF !DBUSED('vinzari')
  OPEN DATABASE vinzari
ENDIF
SET DATABASE TO vinzari
IF USED('vFacturi')
  SELECT vFacturi
  USE
ENDIF
* Fraza SQL de creare a tabelei virtuale
CREATE SQL VIEW vFacturi AS ;
  SELECT LF.NrFact, F.DataFact, Linie, LF.CodPr, DenPr, ,
  UM, Cantitate, PretUnit, Cantitate * PretUnit AS ;
  ValFaraTVA, Cantitate * PretUnit * ProcTVA AS TVA, ;
  Cantitate * PretUnit * (1 + ProcTVA) AS ValTotala ;
  FROM LINIIFACT LF INNER JOIN FACTURI F ON LF.NrFact=F.NrFact ;
  INNER JOIN PRODUSE P ON LF.CodPr=P.CodPr ;
  ORDER BY LF.NrFact, Linie
```

60. Pentru detalii privind actualizarea tabelelor virtuale în Oracle 8i, vezi și [Bobrowski01].

61. Vezi și [Fotache&Strimbei99-2].

atunci, cele mai dificile probleme ale view-urilor în de actualizarea lor, de fapt, de propagarea actualizărilor tabelelor virtuale în tabelele de bază din care sunt construite.

La creare, o tabelă virtuală poate fi declarată actualizabilă sau non aktualizabilă (Read-Only). Pentru a fi actualizabilă, o tabelă virtuală trebuie să aibă fiecare linie a sa asociată unei singure linii din tabelele bazelor. La modificarea unei linii din view, propagarea poate fi făcută, în aceste condiții, fără probleme de ambiguitate.

Firește, o tabelă derivată de genul:

```
CREATE VIEW vJudetel AS
    SELECT *
    FROM Judete
    WHERE regiune = 'Moldova' OR regiune = 'Dobrogea'
nu va crea probleme insolubile la actualizare. În schimb, deși cu un conținut identic vJudetel, tabela virtuală vJudete2, creată prin comanda care urmează, nu este actualizabilă nici în SQL-92, nici în majoritatea SGBD-urilor importante.

CREATE VIEW vJudete2 AS
    SELECT *
    FROM Judete
    WHERE regiune = 'Moldova'
UNION
    SELECT *
    FROM Judete
    WHERE regiune = 'Dobrogea'
```

Regulile actualizării tabelelor în SQL-92 sunt foarte stricte⁵⁹:

I. Tabela virtuală trebuie derivată dintr-un **SELECT** cu o singură tabelă de bază; mai multe tabele de bază înseamnă mai multe niveluri ale tabelelor virtuale. Spre exemplu, pentru a construi o tabelă virtuală vCLIENTI în care pot fi modificate: denumirea clientului, adresa sa, denumirea localității în care își are sediul și numele județului, sunt necesare următoarele view-uri:

```
CREATE VIEW vJudete AS
    SELECT *
    FROM Judete

CREATE VIEW vLocalitati AS
    SELECT CodPost, Loc, vJudete.Jud, Judet, Regiune
    FROM LOCALITATI INNER JOIN vJUDETE
    ON LOCALITATI.Jud = vJUDETE.Jud

CREATE VIEW vClienti AS
    SELECT CodCl, DenCl, Adresa, vLocalitati.CodPost,
    Loc, Jud, Judet, Regiune
    FROM CLIENTI INNER JOIN vLOCALITATI
    ON CLIENTI.CodPost = vLOCALITATI.CodPost
```

59. [Celko99]. p. 57.

2. Tabelele virtuale trebuie să includă toate coloanele cheilor primare/alternative ale tabelelor de bază. O linie dintr-o tabelă virtuală trebuie să corespundă unei singure linii din tabela de bază.
3. Toate coloanele neinclusă în view trebuie să permită valori NULL sau să prezinte valori implicate. Altintinderi, inserarea unei linii într-o tabelă derivată ar fi imposibilă.

Nu poate fi actualizată o tabelă virtuală creată prin fraze **SELECT** în care apar:

- clauze GROUP BY / HAVING,
- funcții-agregat,
- coloane calculate,
- UNION,
- INTERSECT,
- EXCEPT (MINUS),
- SELECT DISTINCT.

Din punctul de vedere al actualizării tabelelor virtuale, SQL-92 este mult mai limitat decât o cér regulile lui Codd, preferându-se siguranță. Multe SGBD-uri sunt însă mai îngăduitoare în această privință. Mai mult, în SQL-3 și o serie de SGBD-uri, prin mecanisme de genul declanșatoarelor INSTEAD OF, multe probleme ale tabelelor virtuale, insolubile în SQL-92, își găsesc rezolvarea.

Atunci când tabela virtuală este actualizabilă, se poate folosi clauza **WITH CHECK OPTION** pentru un mai bun control al modificărilor. vJudeteMoldova conține înregistrările tabelei JUDETE pentru care valoarea atributului Regiune este Moldova:

```
CREATE VIEW vJudeteMoldova AS
    SELECT *
    FROM judete
    WHERE Regiune='Moldova'
    WITH CHECK OPTION
```

Comanda **INSERT** în formatul:

```
INSERT INTO vJudeteMoldova VALUES ('BC', 'Bacau', 'Moldova')
se execută, fără probleme, ceea ce nu este valabil și pentru:
```

```
INSERT INTO vJudeteMoldova VALUES ('AR', 'Arad', 'Banat')
care va genera un mesaj de eroare din cauza regiunii ce nu este acceptată în tabela virtuală, grăție clauzei CHECK OPTION. În SQL-92, clauza de verificare este, implicit, declanșată în cascădă (CASCDED), adică atât pentru tabela virtuală curentă, cât și pentru cele pe baza cărora a fost construită, nivel cu nivel. Cu toate acestea, CASCDED se poate înlocui cu LOCAL, caz în care se verifică numai WHERE-ul prezentei tabele virtuale, fără a se testa modul în care modificările afectează tabelele derive de pe nivelurile inferioare.
```

Stergerea unei tabele virtuale din schema bazei de date se realizează prin comanda **DROP VIEW**.

```
DROP VIEW <nume-tabelă-virtuală> <comportament>
unde
```

```
<comportament> ::= CASCADE | RESTRICT
```

```
FROM vinzaridbc v1 INNER JOIN vinzaridbc v2 ;
  ON v1.parentid=v2.objectid ;
WHERE v2.OBJECTTYPE='Table' AND v1.OBJECTTYPE='Field' ;
ORDER BY V1.objectid
```

Din păcate, modul de prezentare a valorilor implicate, regulilor de validare și mesajelor de eroare afișate la încărcarea regulilor de validare la nivel de câmp este destul de criptic și dificil de folosit în aplicații.

Dată fiind structura mult mai analitică a dicționarului bazei (catalogul-sistem), în Oracle sunt necesare mai multe interogări:

- pentru afișarea tabelelor din schema curentă:

```
SELECT TABLE_NAME
FROM USER_TABLES
```

- pentru afișarea tuturor tabelelor, tabelelor virtuale, sinonimelor și secvențelor din schema utilizatorului curent:

```
SELECT TABLE_NAME, TABLE_TYPE
FROM USER_CATALOG
```

- pentru afișarea informațiilor despre atributele unei tabele:

```
SELECT *
FROM USER_TAB_COLUMNS
WHERE TABLE_NAME='FACTURI'
```

- pentru listarea informațiilor despre tabelele derivate:

```
SELECT *
FROM USER_VIEWS
```

- afișarea restricțiilor din schema curentă:

```
SELECT *
FROM USER_CONSTRAINTS
```

Lucrurile stau oarecum asemănător și în DB2:

- tabelele din schema FOTACHEM:

```
SELECT *
FROM SYSCAT.TABLES
WHERE TABSCHEMA='FOTACHEM'
```

- tabelele virtuale:

```
SELECT *
FROM SYSCAT.VIEWS
WHERE VIEWSCHEMA='FOTACHEM'
```

- informații despre atributele unei tabele:

```
SELECT *
FROM SYSCAT.COLUMNS
WHERE TABNAME='FACTURI'
```

- restricțiile de tip CHECK din tabela FACTURI:

```
SELECT *
FROM SYSCAT.COLUMNS
WHERE TABNAME='FACTURI'
```

- coloanele talelei FACTURI pentru care au fost declarate restricții de tip CHECK:

```
SELECT *
FROM SYSCAT.COLCHECKS
WHERE TABNAME='FACTURI'
```

8.2. Tabele virtuale în Oracle și VFP

In mod obișnuit, **tabelele derivate** sunt percepute ca tabele noi, construite prin subconsultări aplicate asupra tabelelor de bază sau altor *view-uri*. Standardul SQL-92 definește *view-urile* ca tabele virtuale ce sunt materializate atunci când este invocat numele lor. Materializarea înseamnă, de fapt, execuția frazei SELECT ce constituie definiția talelei virtuale și popularea cu înregistrări care sunt, de fapt, rezultatul interogării. Formatul general al comenzi de creare a unei asemenea tabele virtuale este:

```
CREATE VIEW <nume-tabelă-virtuală> [<listă-coloane>] AS
  <frază SELECT>
  [WITH [<clauză de nivele>] CHECK OPTION]
  unde
  <clauză de nivele> ::= CASCDED | LOCAL
```

O dată creată tabela virtuală, definiția sa este salvată în schema bazei. Ulterior, ori de câte ori este necesar, la deschiderea/reîmprospătarea tabelei derivate, aceasta este (re)populată cu înregistrări extrase din cele ale tabelelor propriu-zise ce apar în clauza FROM a interogării. O **tabelă virtuală poate fi deci privită și ca o expresie de subconsultare a unei tabele persistente, stocată în bază și invocată prin numele său**.

Potrivit SQL-92, unei tabele virtuale nu i se pot asocia indecsi și nici defini restricții, deși unele SGBD-uri optimizează lucru cu *view-urile* folosind indecsii tabelelor persistente. Numele său este unic și nu se poate autoreferi, deși un *view* poate fi creat pe baza unei combinații de tabele persistente și/sau alte *view-uri*.

SQL-92 introduce și noțiunea de **tabelă derivată**, definită asemănător celei virtuale, dar pentru care se pot defini restricții și asocia indecsi, fiind, spre deosebire de tabela virtuală, persistenta³⁸.

```
<subconsultare-tabelă> AS <nume-tabelă-virtuală>
  [<listă-coloane>]
```

În ceea ce ne privește, acest paragraf este dedicat tabelelor virtuale, problematica expusă fiind o continuare a ceea ce a fost deja prezentat în paragraful 1.5. După cum am văzut

58. Preluare din [Celko2000], p. 266.

În Oracle, lucrurile sunt cu mult mai analitice. Dicționarul este un set de tabele și tabele derivate accesibile diferențiat, în funcție de rolul utilizatorului. Dacă numele acestora este prefixat de **USER**, atunci este vorba despre obiecte ce aparțin utilizatorului curent; **ALL** desemnează, în plus, și obiectele la care utilizatorul curent are acces, dar proprietarul este un alt utilizator, iar prefixul **DBA** (*Data Base Administrator*) se referă la toate obiectele bazei, indiferent de „proprietar”.

Tabela principală a catalogului este numită **DICTIONARY** și are doar două coloane, **TABLE_NAME** și **COMMENTS**. Aflarea tuturor tabelelor virtuale ce oferă informații despre obiectele din schema este posibilă prin interogarea:

```
SELECT *
FROM DICTIONARY
```

Pentru non-administratori, inventarierea obiectelor din schema proprie (tabele, tabele virtuale, sinonime, proceduri, funcții, pachete, secvențe etc.) este posibilă folosind o altă tabelă din dicționarul datelor – **USER_OBJECTS**:

```
SELECT *
FROM USER_OBJECTS
```

DB2 (versiunea 7) administrează două seturi de tabele derivate în cadrul dicționarului de date, în schemele **SYSCAT** și **SYSSTAT**. Ambele seturi sunt accesibile utilizatorilor. Tabelele virtuale din **SYSSTAT** reprezintă un subset al celor din **SYSCAT**, singurul avantaj fiind că o parte din attribute sunt actualizabile.

Despre crearea tabelelor am discutat pe indelete în capitolul 3. Am trecut în revistă principalele clauze pentru declararea restricțiilor: NOT NULL, PRIMARY KEY, UNIQUE, CHECK, FOREIGN KEY.

Ceea ce nu am prezentat în capitolul respectiv jine de crearea unei tabele pe baza unei alte tabele, prin preluarea uneia sau mai multor attribute, eventual chiar definind noi attribute. Spre exemplu, dacă în DB2 se dorește ca din tabela **FACTURI** să se arhiveze toate facturile emise în 1999 într-o altă tabelă, denumită **FACTURI_1999**, se poate folosi următoarea secvență de comenzi SQL:

```
CREATE TABLE FACTURI_1999 AS
  (SELECT *
   FROM FACTURI)
  DEFINITION ONLY ;
INSERT INTO FACTURI_1999
  (SELECT *
   FROM FACTURI
   WHERE DataFact BETWEEN '1999-01-01' AND '1999-12-31'
  ) ;
COMMIT ;
```

Lucrurile nu diferă prea mult în Oracle, dar crearea tablei **FACTURI_1999** și popularea sa pot fi realizate dintr-o singură mișcare:

```
CREATE TABLE FACTURI_1999 AS
  (SELECT *
   FROM FACTURI
   WHERE DataFact BETWEEN
     TO_DATE('1999-01-01','YYYY-MM-DD') AND
     TO_DATE('1999-12-31','YYYY-MM-DD'))
```

*USER - obiecte E utiliz. curent
ALL - obiecte la care utilizatorul curent are acces, dar proprietarul este altul
DBA - toate obiectele bazei*

În Visual FoxPro, crearea unei tabele printr-o combinație **CREATE TABLE / SELECT** nu este posibilă. Prin clauza **FROM ARRAY**, o tabelă poate prelua structura și conținutul unui masiv (ARRAY). Revenind la problema noastră, rezolvarea este la fel de simplă, dar folosind clauza **INTO**:

```
SELECT * ;
FROM FACTURI ;
INTO TABLE FACTURI_1999 ;
WHERE DataFact BETWEEN {^1999-01-01} AND {^1999-12-31}
```

Tot în capitolul 3 prezentăm clauza **CHECK** pentru declararea unor restricții la nivel de atribut. Din păcate, nici unul dintre dialectele SQL ale celor trei produse – DB2, Oracle și VFP – nu suportă clauza SQL-92 legată de folosirea subconsultărilor. Spre exemplu, dacă dorim ca în tabela **LINIIFACT** să instituim restricția ca, în urma oricărei modificări a acestor tabele, să nu existe nici o factură fără măcar o linie, în SQL-92 se poate specifica:

```
ALTER TABLE LINIIFACT
ADD CONSTRAINT macar_o_linie
CHECK (EXISTS (SELECT 1
  FROM facturi
  WHERE facturi.NrFact=liniifact.NrFact))
```

Nici în privința domeniilor și asertiunilor implementările SQL ale produselor comerciale nu stau grozav, așa că în continuare ne ocupăm de tabele virtuale, de proceduri stocate și declanșatoare.

În VFP, dacă se dorește extragerea tuturor tabelelor din bază, se folosește o consultare relativ simplă:

```
SELECT * FROM vnzari.dbc WHERE OBJECTTYPE='Table'
```

Răspunsul este cel din figura 8.2. Atributul **Property** furnizează, prin altele (numele fișierului .DBF de pe disc, indexul primar al tablei), și numele eventualelor declanșatoare ale tablei.

Objectid	Parentid	Objecttype	Objectname	Property	Code	RInfo	User
6	1	Table	iudele	Memo	memo		
12	1	Table	localitati	Memo	memo		
19	1	Table	clienti	Memo	memo		
29	1	Table	persoane	Memo	memo		
43	1	Table	perscenstii	Memo	memo		
52	1	Table	produse	Memo	memo		
59	1	Table	facturi	Memo	memo		
63	1	Table	liniifact	Memo	memo		
74	1	Table	incasari	Memo	memo		
81	1	Table	incasfact	Memo	memo		

Figura 8.2. Extragerea din containerul BD în VFP a informațiilor despre tabele

Informații despre attributele tabelelor pot fi obținute în mod asemănător:

```
SELECT v2.objectname, v1.objectid, v1.objectname,
       left(v1.property, 250) ;
```

Capitolul 8

OBIECTE DIN SCHEMA BAZEI DE DATE. EXTENSII PROCEDURALE ALE SQL

Schema unei baze de date cuprinde, la SGBD-urile moderne, o largă varietate de obiecte, pe lângă ceea ce rămâne esențial – tabela. În SQL-89, singurele obiecte din schema la care utilizatorii aveau acces erau tabela și tabela virtuală (view). SQL-92 consacră o tipologie mult mai generoasă, din care o parte rămâne încă neimplementată în produsele comerciale. La noțiunile amintite în primul capitol – tabela, tabelă virtuală, declanșator/procedură stocată – mai putem adăuga: tabelă temporară, aserțiune, domeniu, secvență, legătură cu altă bază de date/instanță etc. Acest ultim capitol este dedicat aprofundării unor dintre noțiunile deja discutate, prezentările unor noi obiecte, precum și modului în care pot fi obținute informații despre diferite componente din schema bazei.

8.1. Dicționarul bazei de date. Reguli de validare avansate

În primul capitol, preponderent terminologic, prezentam cele două modalități de abordare a unei baze de date, unul care are în vedere aspectul constant, structura (intensia), și celălalt preocupat de conținut efectiv (extensia). De asemenea, am discutat despre rolul dicționarului de date, cel de stocare a schemei bazei, dicționar numit și **container** sau **catalog-sistem**. Dincolo de nuantele specifice fiecărui SGBD, schema unei baze de date poate fi privită ca o descriere a bazei, descriere generată prin intermediul limbajului de definire a datelor (DDL) propriu SGBD-ului respectiv. Cele mai întrebuijante obiecte ale bazei sunt: tabelele, tabelele virtuale, domeniile, restricțiile, privilegiile, procedurile stocate, sinonimele, indecsii, clusterele etc.

Sintaxa SQL-92 pentru crearea schemei unei baze de date este relativ simplă:

```
CREATE SCHEMA <clauză de denumire>
[<specificarea setului de caractere>]
[<element>...]
```

unde:

```
<clauză de denumire> : : = =
<nume schema>
| AUTHORIZATION <identificator de autorizare>
| <nume schema> AUTHORIZATION
  < identificator de autorizare >
< specificarea setului de caractere > : : =
  DEFAULT CHARACTER SET
    <specificatorul setului de caractere>
< element > : : =
  < definiție domeniu >
  | < definiție tabelă >
  | < definiție tabelă virtuală >
  | < comandă pentru acordare de drepturi >
  | < definiție aserțiune >
```

Atât obiectele, cât mai ales dicționarul bazei, prezintă diferențe semnificative de la SGBD la SGBD. Dacă e să incepem cu una dintre cele mai simple forme de dicționar de date, putem face referire la containerul (database container) Visual FoxPro. Aceasta este o tabelă specială cu structura prezentată în figura 8.1.

Name	Type	Width	Decimal	Index	NULL
objectid	Integer	4	-		
parentid	Integer	4	-		
objecttype	Character	10	-		
objectname	Character	128	-		
property	Memo (binary)	4	-		
code	Memo (binary)	4	-		
nrivo	Character	6	-		
user	Memo	4	-		

Figura 8.1. Structura containerului unei BD în VFP

Fiecare obiect din bază este identificat printr-un număr întreg unic – **objectid**. Ierarhia obiectelor (de exemplu, atributele unei tabele) este reflectată prin câmpul **parentid**. Tipologia obiectelor din container (atributul **objecttype**) este mai săracă în VFP: Database, Table, View, Field, Index și Relation. Dacă primele nu necesită comentarii, despre **Relation** trebuie spus că reflectă o legătură permanentă între două tabele, legătură ce nu asigură automat și restricțiile referențiale. **Procedurile stocate** nu sunt un tip special, ci au trei obiecte corespondente, **StoredProceduresSource**, **StoredProceduresObject** și **StoredProceduresDependencies**, toate de tipul Database.

FACTURA	PRODUS	VINZARI	PONDERE_PROD_FACT
1111	Produs 1	595000	11%
1111	Produs 2	937125	17%
1111	Produs 5	3867500	72%
1112	Produs 2	980560	73%
1112	Produs 3	357000	27%
1113	Produs 2	1160250	100%
1114	Produs 2	891310	13%
1114	Produs 4	564060	8%
1114	Produs 5	5331200	79%
1115	Produs 2	1651125	100%
1116	Produs 2	1383375	100%
1117	Produs 1	1130500	49%
1117	Produs 2	1190000	51%
1118	Produs 1	1660050	81%
1118	Produs 2	392700	19%
1119	Produs 2	453985	8%
1119	Produs 3	333200	5%
1119	Produs 4	833000	12%
1119	Produs 5	5622750	76%
1120	Produs 2	1066240	100%
1121	Produs 2	1249500	23%
1121	Produs 5	4188800	77%

Figura 7.39. Pondere fiecărui produs în factură

La analiza dinamicii temporale a unei mărimi se raportează valoarea din linia curentă la cea dintr-o altă linie plasată la o anumită distanță. Spre exemplu, pentru compararea vânzărilor fiecărei luni calendaristice cu vânzările din aceeași lună din anul precedent, va trebui să împărțim valoarea de pe linia curentă la o valoare aflată cu 12 lini/luni înainte. Apare astfel noțiunea de deplasament înapoi (LAG) și înainte (LEAD). Firește, soluția funcționează numai dacă nu există luni de intrerupere, adică dacă intervalul este continuu.

În interogarea următoare, al cărui rezultat este prezentat în figura 7.40, deplasamentul este, pentru ambele funcții, 1.

```
SELECT TO_CHAR(Zi, 'YYYY-MM-DD') AS Zi, Vinzari,
       LAG(Vinzari,1) OVER (ORDER BY Zi) AS Vnz_LAG,
       LEAD(Vinzari,1) OVER (ORDER BY Zi) AS Vnz_Lead
  FROM
    (SELECT DataFact AS Zi,
           SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari
      FROM PRODUSE P, LINIIFACT LF, FACTURI F
     WHERE P.CodPr=LF.CodPr AND LF.NrFact=F.NrFact
   GROUP BY DataFact)
```

Funcțiile LAG și LEAD extrag vânzările din ziua precedentă, respectiv următoare. Locurile libere din cadrul listei reprezintă valori NULL.

ZI	VINZARI	VNZ_LAG	VNZ_LEAD
2000-08-01	14684005		3034500
2000-08-02	3034500	14684005	2320500
2000-08-03	2320500	3034500	2052750
2000-08-04	2052750	2320500	13747475
2000-08-07	13747475	2052750	

Figura 7.40. Vânzările din ziua curentă, precedentă și următoare

Care este evoluția vânzărilor de la o zi la alta?

```
SELECT TO_CHAR(Zi,'YYYY-MM-DD') AS Zi,
       TO_CHAR(Vinzari,'999999999999') AS Vinzari,
       TO_CHAR(LAG(Vinzari,1) OVER
               (ORDER BY Zi),'999999999999') AS Vnz_Preced,
       TO_CHAR(Vinzari - LAG(Vinzari,1) OVER
               (ORDER BY Zi),'999999999999') AS Diferenta
  FROM
    (SELECT DataFact AS Zi,
           SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari
      FROM PRODUSE P, LINIIFACT LF, FACTURI F
     WHERE P.CodPr=LF.CodPr AND LF.NrFact=F.NrFact
   GROUP BY DataFact )
```

obține rezultatul din figura 7.36.

DENPR	DATAFACT	VINZARI	VNZ_MAX
Produs 1	2000-08-04	1660050	1660050
Produs 1	2000-08-03	1130500	1660050
Produs 1	2000-08-01	595000	1660050
Produs 2	2000-08-01	3969245	3969245
Produs 2	2000-08-02	3034500	3969245
Produs 2	2000-08-07	2769725	3969245
Produs 2	2000-08-03	1190000	3969245
Produs 2	2000-08-04	392700	3969245
Produs 3	2000-08-01	357000	357000
Produs 3	2000-08-07	333200	357000
Produs 4	2000-08-07	833000	833000
Produs 4	2000-08-01	564060	833000
Produs 5	2000-08-07	9811550	9811550
Produs 5	2000-08-01	9198700	9811550

Figura 7.36. Determinarea, pentru fiecare produs, a vânzărilor și a maximului zilnic

În final, răspunsul problemei este cel din figura 7.37.

DENPR	DATAMAX	VNZ_MAX
Produs 1	2000-08-04	1660050
Produs 2	2000-08-01	3969245
Produs 3	2000-08-01	357000
Produs 4	2000-08-07	833000
Produs 5	2000-08-07	9811550

Figura 7.37. Zilele în care s-a vândut cel mai bine fiecare produs

Pentru respectarea adevărului istoric, se cuvine de remarcat că, în locul „cosmopolitei” funcții FIRST_VALUE, am fi putut folosi în subconsultare și „tocita” MAX, bineînțeles, într-o haină nouă:

```
SELECT DenPr, DataFact, SUM(Cantitate * PretUnit
    * (1+ProcTVA)) AS Vinzari,
    MAX(SUM(Cantitate * PretUnit * (1+ProcTVA)))
    OVER (PARTITION BY DenPr ORDER BY
        SUM(Cantitate * PretUnit * (1+ProcTVA)) DESC)
    AS Vnz_Max
FROM PRODUSE P, LINIIFACT LF, FACTURI F
WHERE P.CodPr=LF.Codpr AND LF.NrFact=F.NrFact
GROUP BY DenPr, DataFact
```

Atunci când se dorește determinarea contribuției unui produs la totalul vânzărilor, părții unui client în totalul datornicilor, procentului unui produs din valoarea unei facturi, ponderii salariului directorului general (sau al șefilor de departamente) în totalul fondului de salarii al unei întreprinderi și.m.d. se poate apela la serviciile funcției RATIO_TO_REPORT, care calculează raportul dintre o valoare (atribut/expresie) și suma totală a valorii respective pentru toate linile din sferastră.

Să se afișeze structura vânzărilor pe județe.

```
SELECT Judet, Vinzari, ROUND(RATIO_TO_REPORT(Vinzari)
    OVER (),2) AS Pondere_Judet
FROM
    (SELECT Judet, SUM(Cantitate * PretUnit * (1+ProcTVA))
        AS Vinzari
    FROM PRODUSE P, LINIIFACT LF, FACTURI F,
        CLIENTI C, LOCALITATI L, JUDETE J
    WHERE P.CodPr=LF.Codpr AND LF.NrFact=F.NrFact AND
        F.CodCl=C.CodCl
        AND C.CodPost=L.CodPost AND L.Jud=J.Jud
    GROUP BY Judet
    )
```

După cum se observă, avantajul funcției este că nu necesită precizarea explicită a numitorului. Rezultatul interogării este prezentat în figura 7.38.

JUDET	VINZARI	PONDERE_JUDET
Iasi	19088790	.53
Nearid	6786570	.19
Timis	2720935	.08
Vaslui	7242935	.2

Figura 7.38. Contribuția fiecărui județ în totalul vânzărilor

Să se calculeze contribuția produselor la valoarea fiecărei facturi.

```
SELECT Factura, Produs, TO_CHAR(Vinzari,'999999999999')
    AS Vinzari,
    ROUND(RATIO_TO_REPORT(Vinzari) OVER
        (PARTITION BY Factura),2)
        * 100 || '%' AS Pondere_Prod_Fact
FROM
    (SELECT NrFact AS Factura, DenPr AS Produs,
        SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari
    FROM PRODUSE P, LINIIFACT LF
    WHERE P.CodPr=LF.Codpr
    GROUP BY NrFact, DenPr
    )
```

```

INTERVAL '1' DAY FOLLOWING)
AS P1_Act_U1_Zile
FROM
(SELECT DenCl AS Client, DataFact,
SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari
FROM PRODUSE P, LINIIFACT LF, FACTURI F, CLIENTI C
WHERE P.CodPr=LF.Codpr AND LF.NrFact=F.NrFact AND
F.CodCl=C.CodCl
GROUP BY DenCl, DataFact
) VALFACT

```

Coloana Crt_Prec1_Linii însumează, după cum am văzut în precedenta interogare, valoarea facturii curente și cea a precedentei (ROWS 1 PRECEDING);

La calculul valorii Crt_Prec1_Linii se ia în calcul linia precedentă, numai dacă data căreia îi este asociată este imediat înaintea datei din linia curentă (RANGE INTERVAL '1' DAY PRECEDING) – vezi, pentru Client 1, situația zilei de 7 august;

CLIENT	ZI	VINZARI	CRT_PREC1_LINIE	CRT_PREC1_ZILE	P1_ACT_U1_LINIE	P1_ACT_U1_ZILE
Client 1 SRL	2000-08-01	5399625	5399625	7050750	7050750	
Client 1 SRL	2000-08-02	1651125	7050750	9371250	9371250	
Client 1 SRL	2000-08-03	2320500	3971625	6024375	6024375	
Client 1 SRL	2000-08-04	2052750	4373250	5439490	4373250	
Client 1 SRL	2000-08-07	1066240	3118990	1066240	3118990	1066240
Client 2 SA	2000-08-01	1160250	1160250	1160250	1160250	
Client 3 SRL	2000-08-07	7242935	7242935	7242935	7242935	
Client 4	2000-08-07	5438300	5438300	5438300	5438300	
Client 5 SRL	2000-08-01	1337560	1337560	1337560	1337560	
Client 6 SA	2000-08-01	6786570	6786570	6786570	6786570	
Client 7 SRL	2000-08-02	1383375	1383375	1383375	1383375	

Figura 7.34. Fereștele definite logic și fizic

Curios, deși interogarea respectă specificațiile SQL-99 și funcționează în Oracle 8i2, în DB2 v7 nu am reușit să o facem operațională.

7.5. Comparații și ponderi în Oracle 8i2

Atunci când, într-o partitură, se dorește raportarea valorilor liniei curente la cele din prima sau ultima linie, este rezonabil să se folosească funcțiile FIRST_VALUE și LAST_VALUE.

Care este raportul zilnic al vânzărilor, relativ la prima zi de vânzări?

```

SELECT
TO_CHAR(DataFact,'YYYY-MM-DD') AS Zi,
TO_CHAR(Vinzari,'999999999999') AS Vinzari,

```

```

FIRST_VALUE(Vinzari) OVER (ORDER BY DataFact) AS Vnz_Zil,
ROUND(Vinzari / FIRST_VALUE(Vinzari) OVER
(ORDER BY DataFact),2) AS Raport_Crt_Prima_Zi
FROM
(SELECT DataFact, SUM(Cantitate * PretUnit * (1+ProcTVA))
AS Vinzari
FROM PRODUSE P, LINIIFACT LF, FACTURI F
WHERE P.CodPr=LF.Codpr AND LF.NrFact=F.NrFact
GROUP BY DataFact
)
```

ZI	VINZARI	VNZ_ZIL	RAPORT_CRT_PRIMA_ZI
2000-08-01	14684005	14684005	1
2000-08-02	3034500	14684005	21
2000-08-03	2320500	14684005	16
2000-08-04	2052750	14684005	14
2000-08-07	13747475	14684005	.94

Figura 7.35. Exemplu de utilizare a funcției FIRST_VALUE

Să se determine ziua în care s-a vândut cel mai bine fiecare produs.

Una dintre soluții se bazează pe o subconsultare ce întrebuițează funcția FIRST_VALUE:

```

SELECT DenPr, TO_CHAR(DataFact, 'YYYY-MM-DD') AS DataFact,
TO_CHAR(Vnz_Max, '999999999999') AS Vnz_Max
FROM
(SELECT DenPr, DataFact,
SUM(Cantitate * PretUnit * (1+ProcTVA))
AS Vinzari,
FIRST_VALUE(SUM(Cantitate * PretUnit * (1+ProcTVA)))
OVER (PARTITION BY DenPr ORDER BY
SUM(Cantitate * PretUnit * (1+ProcTVA))) DESC)
AS Vnz_Max
FROM PRODUSE P, LINIIFACT LF, FACTURI F
WHERE P.CodPr=LF.Codpr AND LF.NrFact=F.NrFact
GROUP BY DenPr, DataFact
)
WHERE Vnzari = Vnz_Max

```

Subconsultarea:

```

SELECT DenPr, DataFact,
SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari,
FIRST_VALUE(SUM(Cantitate * PretUnit * (1+ProcTVA)))
OVER (PARTITION BY DenPr
ORDER BY SUM(Cantitate * PretUnit * (1+ProcTVA))) DESC)
AS Vnz_Max
FROM PRODUSE P, LINIIFACT LF, FACTURI F
WHERE P.CodPr=LF.Codpr AND LF.NrFact=F.NrFact
GROUP BY DenPr, DataFact

```

```

FROM
(SELECT DataFact, F.NrFact,
SUM(Cantitate * PretUnit * (1+ProcTVA)) AS ValoareFact
FROM PRODUSE P, LINIIFACT LF, FACTURI F
WHERE P.CodPr=LF.Codpr AND LF.NrFact=F.NrFact
GROUP BY DataFact, F.NrFact
) VALFACT

```

ZI	NRFACT	VALOAREFACT	VAL_CUMULATA
2000-08-01	1111	5399625.00	5399625.00
2000-08-01	1112	1337560.00	6737185.00
2000-08-01	1113	1160250.00	7897435.00
2000-08-01	1114	6786570.00	14684005.00
2000-08-02	1115	1651125.00	1651125.00
2000-08-02	1116	1383375.00	3034500.00
2000-08-03	1117	2320500.00	2320500.00
2000-08-04	1118	2052750.00	2052750.00
2000-08-07	1119	7242935.00	7242935.00
2000-08-07	1120	1068240.00	8309175.00
2000-08-07	1121	5438300.00	13747475.00

Figura 7.32. Valoarea cumulată, după fiecare factură, pentru fiecare zi

Interogarea următoare, al cărei rezultat este cel din figura 7.33, ilustrează patru moduri de construire a unei ferestre:

Coloana Cumul_Prec calculează suma valorilor facturilor din ziua respectivă, începând cu prima și terminând cu factura curentă (ROWS UNBOUNDED PRECEDING);

Coloana Crt_Prec1 insumează valoarea facturii curente și cea a precedentei (ROWS 1 PRECEDING);

La calculul Prec1_Act_Urmat1 se insumează valorile facturilor: curentă, precedentă și următoare (ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING);

În fine, Cumul_Urmat insumează valoarea facturii curente și tuturor celor ce-i succed în ziua respectivă (ROWS BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING).

```

SELECT DataFact AS Zi, NrFact, ValoareFact AS Val_Fact,
SUM(ValoareFact) OVER
(PARTITION BY DataFact
ORDER BY NrFact
ROWS UNBOUNDED PRECEDING)
AS Cumul_Prec,
SUM(ValoareFact) OVER
(PARTITION BY DataFact
ORDER BY NrFact
ROWS 1 PRECEDING)
AS Crt_Prec1,
SUM(ValoareFact) OVER
(PARTITION BY DataFact
ORDER BY NrFact
ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING)
AS Prec1_Act_Urmat1,
SUM(ValoareFact) OVER

```

```

(PARTITION BY DataFact
ORDER BY NrFact
ROWS BETWEEN CURRENT ROW AND
UNBOUNDED FOLLOWING)
AS Cumul_Urmat
FROM
(SELECT DataFact, F.NrFact,
SUM(Cantitate * PretUnit * (1+ProcTVA)) AS ValoareFact
FROM PRODUSE P, LINIIFACT LF, FACTURI F
WHERE P.CodPr=LF.Codpr AND LF.NrFact=F.NrFact
GROUP BY DataFact, F.NrFact
) VALFACT

```

ZI	NRFACT	VAL_FACT	CUMUL_PREC	CRT_PREC1	PREC1_ACT_URMAT1	CUMUL_URMAT
2000-08-01	1111	5399625.00	5399625.00	5399625.00	6737185.00	14684005.00
2000-08-01	1112	1337560.00	6737185.00	6737185.00	7897435.00	9284380.00
2000-08-01	1113	1160250.00	7897435.00	2497810.00	9284380.00	7946820.00
2000-08-01	1114	6786570.00	14684005.00	7946820.00	7946820.00	6786570.00
2000-08-02	1115	1651125.00	1651125.00	3034500.00	3034500.00	3034500.00
2000-08-02	1116	1383375.00	3034500.00	3034500.00	3034500.00	1383375.00
2000-08-03	1117	2320500.00	2320500.00	2320500.00	2320500.00	2320500.00
2000-08-04	1118	2052750.00	2052750.00	2052750.00	2052750.00	2052750.00
2000-08-07	1119	7242935.00	7242935.00	8309175.00	8309175.00	13747475.00
2000-08-07	1120	1068240.00	8309175.00	13747475.00	13747475.00	6504540.00
2000-08-07	1121	5438300.00	13747475.00	6504540.00	6504540.00	5436300.00

Figura 7.33. Patru moduri de definire a ferestrelor

Dacă în SELECT-ul precedent mărimea ferestrelor era definită fizic, prin numărul de linii ce precedă sau succed liniei curente, în următoarea interogare se pun față în față ferestre definite fizic, prin număr de linii, și logic, prin intervale (de zile).

```

SELECT Client, DataFact AS Zi, Vinzari,
SUM(Vinzari) OVER
(PARTITION BY Client
ORDER BY DataFact
ROWS 1 PRECEDING)
AS Crt_Prec1_Linii,
SUM(Vinzari) OVER
(PARTITION BY Client
ORDER BY DataFact
RANGE INTERVAL '1' DAY PRECEDING)
AS Crt_Prec1_Zile,
SUM(Vinzari) OVER
(PARTITION BY Client
ORDER BY DataFact
ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING)
AS Pl_Act_U1_Linii,
SUM(Vinzari) OVER
(PARTITION BY Client
ORDER BY DataFact
RANGE BETWEEN INTERVAL '1' DAY PRECEDING AND

```

POZITIE	ZI	NR_FACTURIOR
1	2000-08-01	4
1	2000-08-07	4
2	2000-08-02	2

Figura 7.30. DENSE_RANK pentru obținerea primelor două valori ale numărului zilnic de facturi

În afară folositorilor ROW_NUMBER, RANK și DENSE_RANK, Oracle 8i2 a fost gândit să și-apropie și mai mult o categorie profesională sedusă, de mult timp, de programe precum SPSS sau chiar Excel, și anume statisticienii. Apar astfel funcțiile: CUME_DIST, PERCENT_RANK, NTILE. Pentru a le compara, folosim interogarea următoare, al cărei rezultat este vizualizat în figura 7.31.

```
SELECT DENSE_RANK() OVER (ORDER BY COUNT(*) DESC) AS Pozitie,
       TO_CHAR(DataFact, 'YYYY-MM-DD') AS Zi,
       COUNT(*) AS Nr_Facturilor,
       PERCENT_RANK() OVER (ORDER BY COUNT(*) DESC)
          AS Poz_Proc_Zi,
       CUME_DIST() OVER (ORDER BY COUNT(*) DESC)
          AS Distrib_Cum_Zi,
       NTILE(3) OVER (ORDER BY COUNT(*) DESC) AS Tertile_Zi,
       NTILE(4) OVER (ORDER BY COUNT(*) DESC) AS Cuartile_Zi
  FROM FACTURI
 GROUP BY DataFact
```

POZITIE	ZI	NR_FACTURIOR	POZ_PROC_ZI	DISTRIB_CUM_ZI	TERTILE_ZI	CUARTILE_ZI
1	2000-08-01	4	0	4	1	1
1	2000-08-07	4	0	4	1	1
2	2000-08-02	2	5	6	2	2
3	2000-08-03	1	.75	1	2	3
3	2000-08-04	1	.75	1	3	4

Figura 7.31. Funcțiile Oracle PERCENT_RANK, CUME_DIST și NTILE

PERCENT_RANK întoarce poziția procentuală a unei linii în cadrul unei partii, calculată prin formula: $(pozitia liniei în partii - 1) / (nr. liniilor din partii - 1)$.

CUME_DIST calculează poziția unei valori specificate x relativ la o mulțime M de N valori: $CUME_DIST(x) = \text{numărul de valori (diferite de, sau egale cu } x\text{) din } M \text{ ce precedă pe } x$

NTILE(3) calculează tertilele, iar NTILE(4) calculează quartilele.

7.4. Ferestre pentru funcțiile analitice

Una dintre cele mai importante noțiuni introduse de Amendamentul OLAP al SQL-99 este **ferestra**. **Ferestra** se definește în cadrul unei partii și se referă la intervalul liniilor luate în calcul pentru **linia curentă**. Mărimea ferestrăi se poate specifica fie fizic, prin un

număr de linii, fie *logic*, printr-un interval de tip data calendaristică/timp sau interval de valori. Calculele se efectuează pentru fiecare linie din cadrul ferestrăi, fereastră mișcătoare între o poziție de start și una de final. Linia curentă servește ca punct de referință pentru determinarea incepătorului și sfârșitului ferestrăi.

Specificațiile unei ferestre privesc trei componente: partionarea, 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 8i2 mai oferă suport pentru alte câteva funcții statistice: VAR_SAMP, VAR_POP, STDDEV_SAMP și.m.d., pe care, din lipsă de cunoștințe de specialitate, nu le discutăm aici. Valorile agregate pot fi cumulative, mișcătoare sau centrate.

Să se afișeze, pentru fiecare zi, valoarea cumulată a vânzărilor după fiecare factură.

- o partie are dimensiunea unei zile; numărul de linii care o alcătuiesc depinde de numărul de facturi emise prezente în ziua respectivă;
- ordonarea în cadrul partii se face după valoarea NrFact; prin urmare, poziția de incepător a fiecărei partii este dată de prima factură din ziua respectivă, iar cea de sfârșit de factură ce are cel mare număr pentru DataFact ce definește partia curentă;
- fereastră pentru care se efectuează calculul, pentru fiecare linie, are o margine fixă – incepătorul partiei și una mobilă, care este chiar linia curentă.

În SQL-99 apare o clauză nouă denumită chiar WINDOW, în care se declară cele trei elemente enumerate anterior, partionarea, ordonarea și grupurile de agregare. Soluția problemei formulate mai sus este:

```
SELECT DataFact AS Zi, NrFact, ValoareFact,
       SUM(ValoareFact) OVER W1 AS Val_Cumulata
  FROM
    (SELECT DataFact, F.NrFact, SUM(Cantitate * PretUnit
        * (1+ProcTVVA)) AS ValoareFact
      FROM PRODUSE P, LINIIFACT LF, FACTURI F
     WHERE P.CodPr=LF.Codpr AND LF.NrFact=F.NrFact
    GROUP BY DataFact, F.NrFact
   ) VALFACT
  WINDOW W1 AS
    (PARTITION BY DataFact
     ORDER BY NrFact
     ROWS UNBOUNDED PRECEDING)
```

Fereastră definită are numele W1 și se constituie în cadrul partiiilor formate de valorile comune ale atributului DataFact. În cadrul partiiilor, liniile sunt ordonate crescător după valorile atributului NrFact, iar intervalul linioilor asupra căruia se aplică funcția de agregare SUM(...). Este alcătuit din prima linie din partia și linia curentă (ROWS UNBOUNDED PRECEDING). Altă variantă SQL-99 permite declararea „în linie” a ferestrăi, varianta utilizată de Oracle 8i2 și DB2 v7, parametrii ferestrăi fiind definiți direct în clauza SELECT, după OVER:

```
SELECT DataFact AS Zi, NrFact, ValoareFact,
       SUM(ValoareFact) OVER
         (PARTITION BY DataFact ORDER BY NrFact
          ROWS UNBOUNDED PRECEDING)
      AS Val_Cumulata
```

Pozitie	Zi	Nr_Facturilor
1	01-08-2000	4
2	07-08-2000	4
3	02-08-2000	2

Figura 7.26. Primele două valori ale numărului zilnic de facturi emise și datele respective.

Același rezultat se obține modificând, în ultima interogare ce utilizează funcția `ROW_NUMBER()`, doar 1 cu 2, adică, în loc de `X2.Poz <=1, X2.Poz <=2`. Deci putem vorbi de o cvasigenerealizare a acestei interogări.

Obținerea pozitiei corecte în rezultat se plătește scump, dar merită:

```

SELECT RANK() OVER (ORDER BY COUNT(*) DESC) AS Pozitie,
       DataFact AS Zi, COUNT(*) AS Nr_Facturilor
  FROM FACTURI
 GROUP BY DataFact
 HAVING COUNT(*) >= ANY
    (
      SELECT Nr_Facturilor
        FROM
          (SELECT RANK() OVER
            (ORDER BY Nr_Facturilor DESC)
           AS Poz, Nr_Facturilor
         FROM
           (SELECT DISTINCT COUNT(*) AS Nr_Facturilor
             FROM FACTURI
            GROUP BY DataFact) X1
          ) X2
        WHERE Poz <= 2
    )
  
```

Pozitie	Zi	Nr_Facturilor
1	2000-08-01	4
1	2000-08-07	4
3	2000-08-02	2

Figura 7.27. Indicarea corectă a pozitiei din clasament

Care sunt cele mai bine vândute două produse ale fiecărei zile?

Practic, interesează o listă precum cea din figura 7.28.

DATAFACT	DENPR	VINZARI	Pozitie_Zi
2000-08-01	Produs 5	9198700.00	1
2000-08-01	Produs 2	3969245.00	2
2000-08-02	Produs 2	3034500.00	1
2000-08-03	Produs 2	1190000.00	1
2000-08-03	Produs 1	1130500.00	2
2000-08-04	Produs 1	1660050.00	1
2000-08-04	Produs 2	392700.00	2
2000-08-07	Produs 5	9811550.00	1
2000-08-07	Produs 2	2769725.00	2

Figura 7.28. Cele mai bine vândute două produse din fiecare zi

Soluția următoare este una comună DB2 v7/Oracle 8i2:

```

SELECT *
  FROM
    (SELECT DataFact, DenPr, SUM(Cantitate * PretUnit *
                                  (1+ProcTVA)) AS Vinzari,
           RANK() OVER (PARTITION BY DataFact
                        ORDER BY SUM(Cantitate * PretUnit * (1+ProcTVA)) DESC
                        AS Pozitie_Zi
                      ) AS Pozitie_Zi
     FROM PRODUSE P, LINIIFACT LF, FACTURI F
    WHERE P.CodPr=LF.CodPr AND LF.NrFact=F.NrFact
      GROUP BY DataFact, DenPr ) X1
 WHERE Pozitie_Zi <= 2
  
```

Asemănătoare funcției `RANK` este `DENSE_RANK`, care, spre deosebire de prima, nu lasă goluri în clasament atunci când pe poziție precedență sunt două sau mai multe linii. Diferența este pusă în valoare de interogarea următoare și rezultatul acesteia din figura 7.29.

```

SELECT DataFact, COUNT(*) AS Nr_Facturilor,
       RANK() OVER (ORDER BY COUNT(*) DESC) AS Poz_RANK,
       DENSE_RANK() OVER (ORDER BY COUNT(*) DESC)
                     AS Poz_DENSE_RANK
  FROM FACTURI
 GROUP BY DataFact
  
```

DATAFACT	NR_FACTURILOR	PÓZ_RANK	PÓZ_DENSE_RANK
2000-08-01	4	1	1
2000-08-07	4	1	1
2000-08-02	2	3	2
2000-08-03	1	4	3
2000-08-04	1	4	3

Figura 7.29. Diferența dintre RANK și DENSE_RANK

Cu `DENSE_RANK` se mai simplifică și interogarea ce afișează cele mai mari două valori ale numărului zilnic de facturi emise, precum și datele (calendaristice) respective:

```

SELECT DENSE_RANK() OVER (ORDER BY COUNT(*) DESC) AS Pozitie,
       DataFact AS Zi, COUNT(*) AS Nr_Facturilor
  FROM FACTURI
 GROUP BY DataFact
 HAVING COUNT(*) >= ANY
    (
      SELECT Nr_Facturilor
        FROM
          (SELECT DENSE_RANK() OVER (ORDER BY COUNT(*) DESC)
            AS Poz, COUNT(*) AS Nr_Facturilor
           FROM FACTURI
          GROUP BY DataFact
          ) X2
        WHERE Poz <= 2
    )
  
```

Să se afișeze ziua sau zilele în care s-au emis cele mai multe facturi.

Interogarea Oracle 8i2:

```
SELECT DataFact, Nr_Facturilor, ROWNUM AS Pozitie
FROM
  (SELECT DataFact, COUNT(*) AS Nr_Facturilor
   FROM FACTURI
   GROUP BY DataFact
   ORDER BY Nr_Facturilor DESC)
WHERE ROWNUM = 1
```

este incorectă, deoarece extrage numai una dintre cele două zile (1 și 7 august) în care s-au emis câte patru facturi. Cu toate acestea, rezultatul corect poate fi obținut și îl să apela la funcția RANK, ci prin folosirea abuzivă a subconsultărilor și a posibilității versiunii 8i2 de a ordona linile oricărui subconsultare:

```
SELECT ROWNUM AS Pozitie,
       TO_CHAR(DataFact, 'DD-MM-YYYY') AS Zi, Nr_Facturilor
  FROM
    (SELECT DataFact, COUNT(*) AS Nr_Facturilor
     FROM FACTURI
     GROUP BY DataFact
     HAVING COUNT(*) >= ANY
        (SELECT Nr_Facturilor
         FROM
           (SELECT DISTINCT COUNT(*) AS Nr_Facturilor
            FROM FACTURI
            GROUP BY DataFact
            ORDER BY Nr_Facturilor DESC)
          WHERE ROWNUM <= 1)
        ORDER BY Nr_Facturilor DESC )
```

Rezultatul din figura 7.25 este satisfăcător, chiar dacă poziția a 2-a în clasament a datei de 7 august este discutabilă, deoarece, de fapt, aceasta este pe primul loc, împreună cu 1 august.

POZITIE	ZI	NR_FACTURILOR
1	01-08-2000	4
2	07-08-2000	4

Figura 7.25. Zilele pentru care se înregistrează cel mai mare număr de facturi emise

Interogarea comună bazată pe funcția analitică ROW_NUMBER care se materializează în același rezultat de mai sus este:

```
SELECT ROW_NUMBER() OVER (ORDER BY Nr_Facturilor DESC)
      AS Pozitie, DataFact AS Zi, Nr_Facturilor
  FROM
    (SELECT DataFact, COUNT(*) AS Nr_Facturilor
     FROM FACTURI
     GROUP BY DataFact)
```

```
HAVING COUNT(*) >= ANY
  (
    SELECT Nr_Facturilor
    FROM
      (
        SELECT Nr_Facturilor, ROW_NUMBER()
              OVER (ORDER BY Nr_Facturilor DESC) AS Poz
        FROM
          (
            SELECT DISTINCT COUNT(*) AS Nr_Facturilor
            FROM FACTURI
            GROUP BY DataFact
          ) X1
        ) X2
      WHERE X2.Poz <= 1
    )
  ) X3
  ORDER BY Nr_Facturilor DESC
```

Folosind funcția RANK, ambele zile au același număr în top, 1. și noua variantă este valabilă atât în Oracle 8i2, cât și în DB2 v7.

```
SELECT *
  FROM
    (SELECT RANK() OVER (ORDER BY COUNT(*) DESC) AS Pozitie,
           DataFact AS Zi, COUNT(*) AS Nr_Facturilor
     FROM FACTURI
     GROUP BY DataFact) X
  WHERE Pozitie <= 1
```

Să se afișeze cele mai mari două valori ale numărului zilnic de facturi emise, precum și datele în care s-au înregistrat respectivele valori.

Varianta non-OLAP în Oracle 8i2 este:

```
SELECT ROWNUM AS Pozitie,
       TO_CHAR(DataFact, 'DD-MM-YYYY') AS Zi, Nr_Facturilor
  FROM
    (SELECT DataFact, COUNT(*) AS Nr_Facturilor
     FROM FACTURI
     GROUP BY DataFact
     HAVING COUNT(*) >= ANY
        (SELECT Nr_Facturilor
         FROM
           (SELECT DISTINCT COUNT(*) AS Nr_Facturilor
            FROM FACTURI
            GROUP BY DataFact
            ORDER BY Nr_Facturilor DESC)
          WHERE ROWNUM <= 2)
        ORDER BY Nr_Facturilor DESC )
```

Raportul obținut este cel din figura 7.26.

```

SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari,
RANK() OVER (PARTITION BY DenCl
ORDER BY SUM(Cantitate * PretUnit * (1+ProcTVA)) DESC)
AS Poz_Client_1,
RANK() OVER (PARTITION BY DenCl, GROUPING(DenPr)
ORDER BY SUM(Cantitate * PretUnit * (1+ProcTVA)) DESC)
AS Poz_Client_2,
RANK() OVER (PARTITION BY DenPr
ORDER BY SUM(Cantitate * PretUnit * (1+ProcTVA)) DESC)
AS Poz_Produs_1,
RANK() OVER (PARTITION BY DenPr, GROUPING (DenCl)
ORDER BY SUM(Cantitate * PretUnit * (1+ProcTVA)) DESC)
AS Poz_Produs_2
FROM PRODUSE P, LINIIFACT LF, FACTURI F, CLIENTI C,
LOCALITATI L, JUDETTE J
WHERE P.CodPr=LF.CodPr AND LF.NrFact=F.NrFact AND
F.CodCl=C.CodCl AND C.CodPost=L.CodPost AND L.Jud=J.Jud
GROUP BY CUBE (DenCl, DenPr )
ORDER BY GROUPING(DenCl), DenCl, GROUPING(DenPr), DenPr

```

Coloanele Poz_Client_2 și Poz_Produs_2 conțin, grație includerii clauzelor GROUPING în funcțiile RANK, valorile care ne interesează – vezi figura 7.24.

CLIENT	PRODUS	VINZARI	POZ_CLIENT_1	POZ_CLIENT_2	POZ_PRODUS_1	POZ_PRODUS_2
Clienț 1 SRL	Produs 1	3385550,00	4	3	1	1
Clienț 1 SRL	Produs 2	5237190,00	2	1	2	4
Clienț 1 SRL	Produs 5	3867500,00	3	2	5	1
Clienț 1 SRL	Y-SUBTOTAL-	12490240,00	1	1	2	4
Clienț 2 SA	Produs 2	1160250,00	1	1	8	7
Clienț 2 SA	Y-SUBTOTAL-	1160250,00	1	1	8	7
Clienț 3 SRL	Produs 2	453985,00	4	3	3	2
Clienț 3 SRL	Produs 3	333200,00	5	4	2	1
Clienț 3 SRL	Produs 4	833000,00	3	2	2	1
Clienț 3 SRL	Produs 5	6622750,00	2	1	3	2
Clienț 3 SRL	Y-SUBTOTAL-	7242935,00	1	1	4	3
Clienț 3 SRL	Produs 2	1249500,00	3	2	4	3
Clienț 4	Produs 5	4168800,00	2	1	5	4
Clienț 4	Y-SUBTOTAL-	5439300,00	1	1	6	5
Clienț 4	Produs 2	980560,00	2	1	2	1
Clienț 5 SRL	Produs 3	357000,00	3	2	7	6
Clienț 5 SRL	Y-SUBTOTAL-	1337560,00	1	1	7	6
Clienț 6 SA	Produs 2	891310,00	3	2	3	2
Clienț 6 SA	Produs 4	564060,00	4	3	3	2
Clienț 6 SA	Produs 5	5331200,00	2	1	4	3
Clienț 6 SA	Y-SUBTOTAL-	6786570,00	1	1	3	2
Clienț 7 SRL	Produs 2	1383375,00	1	1	5	6
Clienț 7 SRL	Y-SUBTOTAL-	1383375,00	1	1	1	1
Y-SUBTOTAL-	Produs 1	3385550,00	4	3	1	1
Y-SUBTOTAL-	Produs 2	11356170,00	3	2	1	1
Y-SUBTOTAL-	Produs 3	690200,00	6	5	1	1
Y-SUBTOTAL-	Produs 4	1397080,00	5	4	1	1
Y-SUBTOTAL-	Produs 5	18010250,00	2	1	1	1
Y-SUBTOTAL-	Y-SUBTOTAL-	35839230,00	1	1	1	1

Figura 7.24. CUBE și RANK, cu și fără GROUPING

Care sunt cele mai mari cinci prețuri unitare din LINIIFACT?

În paragraful 5.4 am prezentat o soluție ingenioasă bazată pe subconsultări în cascadă. O dată cu versiunea 8i, Oracle suportă ordonări în subconsultări, astfel încât următoarea frază este operațională:

```

SELECT PretUnit
FROM
  (SELECT PretUnit
   FROM LINIIFACT
   ORDER BY PretUnit DESC) P1
WHERE ROWNUM <=5

```

Din păcate, în DB2, pseudocoloana ROWNUM nu este operațională. În schimb, versiunile 7 și 8i2 ale DB2 și Oracle implementează o nouă funcție OLAP din SQL-99, funcție cu o logică similară, ROW_NUM, astfel încât se poate redacta o soluție comună de genul:

```

SELECT PretUnit
FROM
  (SELECT PretUnit, ROW_NUMBER () OVER
   (ORDER BY PretUnit DESC) AS Poz
   FROM LINIIFACT
   ) P1
WHERE Poz <=5

```

Funcția ROW_NUMBER întoarce numărul de ordine al fiecărei linii în partitia declarată.

Care sunt cele mai mari cinci prețuri unitare de vânzare, produsele și facturile în care apar cele cinci prețuri maxime?

Bazându-ne pe soluțiile anterioare, se poate redacta fraza următoare în sintaxa Oracle 8i2:

```

SELECT DISTINCT NrFact, DenPr, LF.PretUnit
FROM LINIIFACT LF, PRODUSE P,
  (SELECT PretUnit
   FROM LINIIFACT
   ORDER BY PretUnit DESC)
 WHERE ROWNUM <=5 ) PMAX
 WHERE LF.CodPr = P.CodPr AND LF.PretUnit >= PMAX.PretUnit
 ORDER BY LF.PretUnit DESC

```

și o altă comună Oracle 8i2/DB2 v7:

```

SELECT DISTINCT NrFact, DenPr, LF.PretUnit
FROM LINIIFACT LF, PRODUSE P,
  (SELECT Poz, PretUnit
   FROM
     (SELECT PretUnit, ROW_NUMBER () OVER
      (ORDER BY PretUnit DESC) AS Poz
      FROM LINIIFACT
      ) P1
     WHERE Poz <=5 ) PMAX
 WHERE LF.CodPr = P.CodPr AND LF.PretUnit >= PMAX.PretUnit
 ORDER BY LF.PretUnit DESC

```

Cea mai importantă informație ține de faptul că, dacă luăm în calcul liniile subtotalurilor și totalului general, aflăm, spre exemplu, din penultima linie, că județul Vaslui este al treilea ca volum total al vânzărilor. Poziția exactă a Vasluiului în topul județelor este însă a doua, doar ce prima îi revine automat totalului general (ultima linie).

Lucrurile stau astăma și în cazul operatorului CUBE. Deși raportul (figura 7.23) furnizat de următoarea frază SELECT, valabilă deopotrivă în Oracle 8i2 și DB2 v7, este bogat în informații, câteva dintre acestea trebuie nuanțate.

```
SELECT
CASE WHEN GROUPING (DenCl) = 1
    THEN CONCAT (CHR(255), ' - SUBTOTAL - ')
    ELSE DenCl
END AS Client,
CASE WHEN GROUPING (DenPr) = 1
    THEN CONCAT (CHR(255), ' - SUBTOTAL - ')
    ELSE DenPr
END AS Produs,
SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari,
RANK() OVER (PARTITION BY DenCl
ORDER BY SUM(Cantitate * PretUnit * (1+ProcTVA)) DESC)
AS Poz_Client,
RANK() OVER (PARTITION BY DenPr
ORDER BY SUM(Cantitate * PretUnit * (1+ProcTVA)) DESC)
AS Poz_Produs
FROM PRODUSE P, LINIIFACT LF, FACTURI F, CLIENTI C,
LOCALITATI L, JUDETE J
WHERE P.CodPr=LF.Codpr AND LF.NrFact=F.NrFact AND
F.CodCl=C.CodCl AND C.CodPost=L.CodPost AND L.Jud=J.Jud
GROUP BY CUBE (DenCl, DenPr )
ORDER BY DenCl, DenPr
```

CLIENT	PRODUS	VINZARI	POZ_CLIENT	POZ_PRODUS
Client 1 SRL	Produs 1	3385550.00	4	1
Client 1 SRL	Produs 2	5237190.00	2	2
Client 1 SRL	Produs 5	3887500.00	3	6
Client 1 SRL	Y. SUBTOTAL -	12490240.00	1	2
Client 2 SA	Produs 2	1180260.00	1	5
Client 2 SA	Produs 5	1180260.00	1	8
Client 3 SRL	Produs 2	453985.00	4	9
Client 3 SRL	Produs 3	333200.00	5	3
Client 3 SRL	Produs 4	833000.00	3	2
Client 3 SRL	Produs 5	6622750.00	2	2
Client 3 SRL	Y. SUBTOTAL -	1242935.00	1	2
Client 4	Produs 2	1248560.00	3	4
Client 4	Produs 5	4188600.00	2	4
Client 4	Y. SUBTOTAL -	6438300.00	1	5
Client 5 SRL	Produs 2	980560.00	2	6
Client 5 SRL	Produs 3	357000.00	3	2
Client 5 SRL	Produs 5	1237580.00	1	7
Client 5 SRL	Y. SUBTOTAL -	1237580.00	1	7
Client 6 SA	Produs 2	881210.00	3	7
Client 6 SA	Produs 4	584080.00	4	3
Client 6 SA	Produs 5	5321200.00	2	3
Client 6 SA	Y. SUBTOTAL -	6786570.00	1	4
Client 7 SRL	Produs 2	1383375.00	1	3
Client 7 SRL	Produs 5	1383375.00	1	6
Client 7 SRL	Y. SUBTOTAL -	1383375.00	1	6
Y. SUBTOTAL -	Produs 1	3395550.00	4	1
Y. SUBTOTAL -	Produs 2	11351170.00	2	1
Y. SUBTOTAL -	Produs 3	892300.00	6	1
Y. SUBTOTAL -	Produs 4	1397060.00	5	1
Y. SUBTOTAL -	Produs 5	18010250.00	2	1
Y. SUBTOTAL -	Y. EUETOTAL -	25639230.00	1	1

Figura 7.23. RANK și CUBE

Astfel, ca vânzări, pentru Client 1 SRL, Produs 2 este pe locul 1, și nu pe locul 2, cum indică și două linie din figură. La modul general, Poz_Client este tot timpul „umflat” cu o pozitie datorită subtotalurilor de la nivel de client și, pentru ultimele linii, a totalului general. Analog, capul listei pentru Poz_Produs este cel al subtotalului pe produs, iar, spre exemplu, cel mai bun cumpărător al Produs 2 este Client 1 SRL. Faptul că, la Produs 1, în dreptul Clientului 1 SRL este indicată poziția corectă, acest lucru se datorează vânzării produsului unui singur client. Tinând seama de interferența subtotalurilor, am fi tentați să decrementăm rezultatele funcțiilor RANK cu o unitate:

```
SELECT
CASE WHEN GROUPING (DenCl) = 1
    THEN CONCAT (CHR(255), ' - SUBTOTAL - ')
    ELSE DenCl
END AS Client,
CASE WHEN GROUPING (DenPr) = 1
    THEN CONCAT (CHR(255), ' - SUBTOTAL - ')
    ELSE DenPr
END AS Produs,
SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari,
RANK() OVER (PARTITION BY DenCl ORDER BY
SUM(Cantitate * PretUnit * (1+ProcTVA)) DESC) - 1
AS Poz_Client,
RANK() OVER (PARTITION BY DenPr ORDER BY
SUM(Cantitate * PretUnit * (1+ProcTVA)) DESC) - 1
AS Poz_Produs
FROM PRODUSE P, LINIIFACT LF, FACTURI F, CLIENTI C,
LOCALITATI L, JUDETE J
WHERE P.CodPr=LF.Codpr AND LF.NrFact=F.NrFact AND
F.CodCl=C.CodCl AND C.CodPost=L.CodPost AND L.Jud=J.Jud
GROUP BY CUBE (DenCl, DenPr )
ORDER BY DenCl, DenPr
```

Rezultatele ar fi perfecte dacă n-ar exista produse vândute unui singur client (Produs 1) și clienți care au cumpărat un singur produs (Client 2 și Client 7). Pentru aceste două spețe, Poz_Client, respectiv Poz_Produs sunt 0, chiar dacă respectivul produs/client ar fi singur și poziția ar trebui să fie, inevitabil, 1. Am modificat, astfel, gluma aceea răsuflată: să alegări de unul singur și ajungă pe locul zero!

O soluție mult mai elegantă se bazează pe modificarea modului de declarare a partijilor. Pentru a pune în evidență cele două modalități, se afișează căte două poziții pentru clienți/produse, folosindu-se căte două funcții RANK: una identică cu exemplul anterior și o alta care beneficiază de serviciile GROUPING, după cum urmează:

```
SELECT
CASE WHEN GROUPING (DenCl) = 1
    THEN CONCAT (CHR(255), ' - SUBTOTAL - ')
    ELSE DenCl
END AS Client,
CASE WHEN GROUPING (DenPr) = 1
    THEN CONCAT (CHR(255), ' - SUBTOTAL - ')
    ELSE DenPr
END AS Produs,
```

```
FROM PRODUSE P, LINIIFACT LF, FACTURI F
WHERE P.CodPr=LF.CodPr AND LF.NrFact=F.NrFact
GROUP BY DataFact, F.NrFact
ORDER BY DataFact, F.NrFact
```

DATAFACT	NRFACT	VALFACT	POZITIE
2000-08-01	1111	5399625.00	2
2000-08-01	1112	1337560.00	3
2000-08-01	1113	1180250.00	4
2000-08-01	1114	8786570.00	1
2000-08-02	1115	1651125.00	1
2000-08-02	1116	1393375.00	2
2000-08-03	1117	2320500.00	1
2000-08-04	1118	2052750.00	1
2000-08-07	1119	7242935.00	1
2000-08-07	1120	1086240.00	3
2000-08-07	1121	5438300.00	2

Figura 7.20. Topurile zilnice ale facturilor

Factura 1111 este a doua ca valoare pe 1 august 2000, cea mai mare valoare în această zi având factura 1114.

Într-o singură frază SELECT se pot întocmi, pentru aceleasi date, clasamente diferite prin specificarea mai multor partii și criterii. Astfel, interogarea (Oracle 8i2/DB2 v7):

```
SELECT DataFact, DenPr,
       SUM(Cantitate * PretUnit * (1+ProcTVA)) AS ValFact,
       RANK() OVER (PARTITION BY DataFact ORDER BY
                     SUM(Cantitate * PretUnit * (1+ProcTVA)) DESC)
              AS Pozitie_Zi,
       RANK() OVER (PARTITION BY DenPr ORDER BY
                     SUM(Cantitate * PretUnit * (1+ProcTVA)) DESC)
              AS Pozitie_Prod
```

```
FROM PRODUSE P, LINIIFACT LF, FACTURI F
WHERE P.CodPr=LF.CodPr AND LF.NrFact=F.NrFact
GROUP BY DataFact, DenPr
ORDER BY DataFact, DenPr
```

calculează valoarea vânzărilor pe zile (DataFact) pentru fiecare produs și, în plus, determină poziția produsului (DenPr) curent, atât pentru ziua respectivă, cât și pe ansamblu, pentru toate combinațiile zi-produs (DataFact-DenPr), după cum se observă în figura 7.21.

DATAFACT	DENPR	VALFACT	POZITIE_ZI	POZITIE_PROD
2000-08-01	Produs 1	595000.00	3	3
2000-08-01	Produs 2	3969245.00	2	1
2000-08-01	Produs 3	357000.00	5	1
2000-08-01	Produs 4	564060.00	4	2
2000-08-01	Produs 5	9198700.00	1	2
2000-08-02	Produs 2	3034500.00	1	2
2000-08-03	Produs 1	1130500.00	2	2
2000-08-03	Produs 2	1190000.00	1	4
2000-08-04	Produs 1	1660050.00	1	1
2000-08-04	Produs 2	382700.00	2	5
2000-08-07	Produs 2	2769725.00	2	3
2000-08-07	Produs 3	333200.00	4	2
2000-08-07	Produs 4	633000.00	3	1
2000-08-07	Produs 5	9811550.00	1	1

Figura 7.21. Două partii de ordonare

Pe prima linie a rezultatului apar ziua de 1 august 2000 și produsul 1, pentru care valoarea facturată în această zi este de 595.000. 595.000 este, ca mărime, a treia valoare pentru ziua respectivă și a treia pentru produsul 1. Altfel spus, pentru 1 august 2000, produsul 1 este al treilea în topul pe produse al vânzărilor (cel mai bine vândut în această zi fiind produsul 5), iar dintre toate zilele în care a fost vândut „Produs 1”, 1 august este a treia în topul vânzărilor pe zile (cel mai bine s-a vândut pe 4 august).

„Produs 2” s-a vândut cel mai bine pe 1 august și cel mai puțin pe 4 august; pe 2 și pe 3 august a fost cel mai bine vândut produs al zilei.

Ziua de grăzie a produsului 5 a fost 7 august, când a fost cel mai bine vândut produs al zilei și data cu cele mai mari vânzări ale acestuia.

Se poate alcătuiri un clasament nu numai pe baza unei simple funcții-agregat, dar și prin combinarea funcției RANK cu funcțiile ROLLUP și CUBE. Rezultatele nu sunt însă întotdeauna concidente. Spre exemplu, consultarea DB2 v7 următoare produce raportul din figura 7.22.

```
SELECT Judet, DenPr,
       SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari,
       RANK() OVER (PARTITION BY Judet
                     ORDER BY SUM(Cantitate * PretUnit * (1+ProcTVA))) DESC)
              AS Poz_Judet,
       RANK() OVER (PARTITION BY DenPr
                     ORDER BY SUM(Cantitate * PretUnit * (1+ProcTVA))) DESC)
              AS Poz_Produs
FROM PRODUSE P
INNER JOIN LINIIFACT LF ON P.CodPr=LF.Codpr
INNER JOIN FACTURI F ON LF.NrFact=F.NrFact
INNER JOIN CLIENTI C ON F.CodCl=C.CodCl
INNER JOIN LOCALITATI L ON C.CodPost=L.CodPost
INNER JOIN JUDETE J ON L.Jud=J.Jud
GROUP BY ROLLUP (Judet, DenPr)
ORDER BY Judet, DenPr
```

JUDET	DENPR	VINZARI	POZ_JUDET	POZ_PRODUS
Iasi	Produs 1	3395550.00	4	1
Iasi	Produs 2	7846840.00	3	1
Iasi	Produs 5	8056300.00	2	1
Iasi		19088790.00	1	2
Neamt	Produs 2	891310.00	3	3
Neamt	Produs 4	564080.00	4	2
Neamt	Produs 5	5331200.00	2	3
Neamt		6786570.00	1	4
Timis	Produs 2	2363935.00	2	2
Timis	Produs 3	357000.00	3	1
Timis		2720935.00	1	5
Vaslui	Produs 2	453985.00	4	4
Vaslui	Produs 3	333200.00	5	2
Vaslui	Produs 4	833000.00	3	1
Vaslui	Produs 5	5622750.00	2	2
Vaslui		7242935.00	1	3
		35839230.00	1	1

Figura 7.22. RANK și ROLLUP

Este lărgit să se vadă că Poz_Judet va fi întotdeauna 1 în liniile dedicate subtotalului pentru fiecare dintre județe. „Produs 1” nu este al patrulea ca vânzări în județul Iași, ci al treilea. Valorile Poz_Produs sunt însă coerente. Județul în care „Produs 4” s-a vândut cel mai bine este Vaslui.

O variantă comună Oracle/DB2, ba chiar și VFP, este:

```
SELECT DataFact, COUNT(*) AS Nr_Facturilor
FROM FACTURI
GROUP BY DataFact
ORDER BY Nr_Facturilor DESC
```

Ba chiar în Oracle se poate folosi și o subconsultare în clauza FROM:

```
SELECT DataFact, Nr_Facturilor
FROM
  (SELECT DataFact, COUNT(*) AS Nr_Facturilor
   FROM FACTURI
   GROUP BY DataFact )
ORDER BY Nr_Facturilor DESC

iar în DB2 o expresie-tabelă:

WITH ZILE_FACT AS
  (SELECT DataFact, COUNT(*) AS Nr_Facturilor
   FROM FACTURI
   GROUP BY DataFact)
SELECT *
FROM ZILE_FACT
ORDER BY Nr_Facturilor DESC
```

Ce ne facem însă atunci când rezultatul trebuie să îmbrace forma din figura 7.18? Problema esențială este nu ordonarea, cătă indicarea poziției fiecărei zile.

DATAFACT	NR_FACTURILOR	POZITIE
2000-08-01	4	1
2000-08-07	4	1
2000-08-02	2	3
2000-08-03	1	4
2000-08-04	1	4

Figura 7.18. Un clasament veritabil

În Oracle 8i2 este posibilă ordonarea liniilor unei subconsultări din clauza FROM, așa că soluția următoare, bazată pe pseudocoloana ROWNUM, funcționează:

```
SELECT DataFact, Nr_Facturilor, ROWNUM AS Pozitie
FROM
  (SELECT DataFact, COUNT(*) AS Nr_Facturilor
   FROM FACTURI
   GROUP BY DataFact
   ORDER BY Nr_Facturilor DESC)
```

Chiar dacă ne putem declara satisfăcuți de rezultat (figura 7.19), se poate argumenta că 7 august 2000 este pe nedrept „retrogradată” pe locul 2 al clasamentului, atât timp că are același număr de zile ca și 1 august și nu există nici un criteriu suplimentar de departajare. Interrogarea de mai sus este corectă și nu este atunci când fie toate valorile ordonate sunt diferite, fie cănd există un număr suficient de criterii de balotaj care să confere unicitate fiecărei poziții la clasamentul.

DATAFACT	NR_FACTURILOR	POZITIE
2000-08-01	4	1
2000-08-07	4	2
2000-08-02	2	3
2000-08-03	1	4
2000-08-04	1	5

Figura 7.19. Pseudoclasamentul obținut printr-o subconsultare

Pentru astfel de situații, Amendamentul OLAP al SQL-99 prezintă două funcții ideale, RANK și DENSE_RANK. În prealabil însă, câteva chestiuni ce țin de logica execuției unui asemenea gen de consultări. Procesarea interogărilor ce utilizează funcții analitice se derulează în trei etape. Mai întâi, se operează toate jonctionurile, se constituie grupurile și se efectuează selecția asupra grupurilor, altfel spus, se execută clauzele JOIN (sau FROM/WHERE), WHERE, GROUP BY și HAVING. Apoi se aranjează rezultatul în vederea aplicării funcțiilor analitice și se efectuează calculele (sunt create partiiile), iar funcțiile analitice sunt aplicate linie cu linie în fiecare partitură. 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.

Partiile sunt seturi de linii create după delimitarea grupurilor prin GROUP BY, astfel încât pot constitui subiectul (sau obiectul) oricărei funcții de agregare (SUM, AVG...). Constituirea unei partii se poate face în funcție de valorile unuia sau mai multor atribute sau expresii de atribute.

Soluția comună Oracle 8i2/DB2 v7 pentru a obține o situație identică celei din figura 7.18 este:

```
SELECT DataFact, COUNT(*) AS Nr_Facturilor,
       RANK() OVER (ORDER BY COUNT(*) DESC) AS Pozitie
  FROM FACTURI
 GROUP BY DataFact
```

Funcția RANK lejerizează după valorile obținute de COUNT(*) liniile rezultatului și, peea ce este cel mai important, la valori egale, atribuie fiecărei linii aceeași poziție în clasament. Luăm un alt exemplu care să pună în valoare și partiiile.

Să se afișeze, pentru fiecare zi de vânzări, topul celor mai „valorioase” facturi.

Să analizăm problema:

- partitura are dimensiunea unei zile; numărul liniilor depinde de numărul facturilor întocmite în ziua respectivă; prin urmare, atributul de partisionare este DataFact;
- în cadrul unei partii, clasamentul se face în funcție de valorile calculate de funcția SUM;
- în rezultatul final, pentru fiecare factură, liniile sunt dispuse în funcție de valorile DataFact și NrFact, deci la prezentare se păstrează ordinea cronologică.

Iată și soluția DB2 v7 și Oracle 8i2 (rezultatul în figura 7.20):

```
SELECT DataFact, F.NrFact,
       SUM(Cantitate * PretUnit * (1+ProcTVA)) AS ValFact,
       RANK() OVER (PARTITION BY DataFact ORDER BY
                     SUM(Cantitate * PretUnit * (1+ProcTVA)) DESC)
       AS Pozitie
```

În schimb, dacă GROUP BY are forma:

GROUP BY CUBE(DenCl, DenPr), CAST(DataFact AS CHAR(14))

vor fi calculate patru subtotaluri (tabel 7.3), pentru combinațiile:

- (DenCl, DenPr, CAST(DataFact AS CHAR(14))) – GROUP BY-ul obișnuit
- (DenCl, CAST(DataFact AS CHAR(14)))
- (DenPr, CAST(DataFact AS CHAR(14)))
- (CAST(DataFact AS CHAR(14))).

Tabelul 7.3. Alt CUBE parțial

CLIENT	PRODUS	ZIUA	VINZARI
Client 1 SRL	Produs 1	01/08/2000	595000
Client 1 SRL	Produs 1	03/08/2000	1130500
Client 1 SRL	Produs 1	04/08/2000	1660050
Client 1 SRL	Produs 2	01/08/2000	937125
Client 1 SRL	Produs 2	02/08/2000	1651125
Client 1 SRL	Produs 2	03/08/2000	1190000
Client 1 SRL	Produs 2	04/08/2000	392700
Client 1 SRL	Produs 2	07/08/2000	1066240
Client 1 SRL	Produs 5	01/08/2000	3867500
Client 1 SRL	ÿsubtotal	01/08/2000	5399625
Client 1 SRL	ÿsubtotal	02/08/2000	1651125
Client 1 SRL	ÿsubtotal	03/08/2000	2320500
Client 1 SRL	ÿsubtotal	04/08/2000	2052750
Client 1 SRL	ÿsubtotal	07/08/2000	1066240
Client 2 SA	Produs 2	01/08/2000	1160250
Client 2 SA	ÿsubtotal	01/08/2000	1160250
Client 3 SRL	Produs 2	07/08/2000	453985
Client 3 SRL	Produs 3	07/08/2000	333200
Client 3 SRL	Produs 4	07/08/2000	833000
Client 3 SRL	Produs 5	07/08/2000	5622750
Client 3 SRL	ÿsubtotal	07/08/2000	7242935
Client 4	Produs 2	07/08/2000	1249500
Client 4	Produs 5	07/08/2000	4188800
Client 4	ÿsubtotal	07/08/2000	5438300
Client 5 SRL	Produs 2	01/08/2000	980560
Client 5 SRL	Produs 3	01/08/2000	357000
Client 5 SRL	ÿsubtotal	01/08/2000	1337560
Client 6 SA	Produs 2	01/08/2000	891310
Client 6 SA	Produs 4	01/08/2000	564060
Client 6 SA	Produs 5	01/08/2000	5331200
Client 6 SA	ÿsubtotal	01/08/2000	6786570

Client 7 SRL	Produs 2	02/08/2000	1383375
ÿ subtotal	Produs 1	01/08/2000	595000
ÿ subtotal	Produs 1	03/08/2000	1130500
ÿ subtotal	Produs 1	04/08/2000	1660050
ÿ subtotal	Produs 2	01/08/2000	3969245
ÿ subtotal	Produs 2	02/08/2000	3034500
ÿ subtotal	Produs 2	03/08/2000	1190000
ÿ subtotal	Produs 2	04/08/2000	392700
ÿ subtotal	Produs 2	07/08/2000	2769725
ÿ subtotal	Produs 3	01/08/2000	357000
ÿ subtotal	Produs 3	07/08/2000	333200
ÿ subtotal	Produs 4	01/08/2000	564060
ÿ subtotal	Produs 4	07/08/2000	833000
ÿ subtotal	Produs 5	01/08/2000	9198700
ÿ subtotal	Produs 5	07/08/2000	9811550
ÿ subtotal	ÿsubtotal	01/08/2000	14684005
ÿ subtotal	ÿsubtotal	02/08/2000	3034500
ÿ subtotal	ÿsubtotal	03/08/2000	2320500
ÿ subtotal	ÿsubtotal	04/08/2000	2052750
ÿ subtotal	ÿsubtotal	07/08/2000	13747475

7.3. Clasamente – soluții clasice și OLAP

Prefațăm discuția despre funcțiile *analitice* din SQL-99 cu o problemă mai veche decât SQL-ul – clasamentele.

Să se afișeze clasamentul zilelor în ordinea descrescătoare a numărului de facturi emise.

Gradul de dificultate al interogărilor care să rezolve această problemă depinde de modul în care se dorește a se obține clasamentul. Cine se mulțumește cu lista din figura 7.17 poate folosi câteva interogări relativ simple.

DATAFACT	NR_FACTURILOR
2000-08-01	4
2000-08-07	4
2000-08-02	2
2000-08-03	1
2000-08-04	1

Figura 7.17. O formă simplă de prezentare a clasamentului

CUBE-uri parțiale

Discuția se poartă în termeni similari ROLLUP-ului parțial. Subtotalurile și combinațiile posibile sunt limitate la atributele-dimensiuni incluse între paranteze. Spre exemplu, dacă GROUP BY are forma:

GROUP BY atribut1, CUBE (atribut2, atribut3)

vor fi calculate patru subtotaluri, pentru combinațiile:

{
 (atribut1, atribut2, atribut3) – GROUP BY-ul obișnuit,
 (atribut1, atribut2),
 (atribut1, atribut3),
 (atribut1).

Versiune DB2:

```
SELECT
  CASE GROUPING (DenCl)
    WHEN 1 THEN CONCAT (CHR(255), ' subtotal ')
    ELSE DenCl END AS Client,
  CASE GROUPING (DenPr)
    WHEN 1 THEN CONCAT (CHR(255), ' subtotal ')
    ELSE DenPr END AS Produs,
  CASE GROUPING (CAST( DataFact AS CHAR(14) ) )
    WHEN 1 THEN CONCAT (CHR(255), ' subtotal ')
    ELSE CAST( DataFact AS CHAR(14) ) END AS Ziua,
  SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari
FROM PRODUSE P
INNER JOIN LINIIFACT LF ON P.CodPr=LF.Codpr
INNER JOIN FACTURI F ON LF.NrFact=F.NrFact
INNER JOIN CLIENTI C ON F.CodCl=C.CodCl
GROUP BY DenCl, CUBE (DenPr, CAST( DataFact AS CHAR(14) ) )
ORDER BY DenCl, DenPr, CAST (DataFact AS CHAR(14) )
```

Tabelul 7.2. CUBE parțial

CLIENT	PRODUS	ZIUA	VINZARI
Client 1 SRL	Produs 1	01/08/2000	595000
Client 1 SRL	Produs 1	03/08/2000	1130500
Client 1 SRL	Produs 1	04/08/2000	1660050
Client 1 SRL	Produs 1	ÿ subtotal	3385550
Client 1 SRL	Produs 2	01/08/2000	937125
Client 1 SRL	Produs 2	02/08/2000	1651125
Client 1 SRL	Produs 2	03/08/2000	1190000
Client 1 SRL	Produs 2	04/08/2000	392700
Client 1 SRL	Produs 2	07/08/2000	1066240
Client 1 SRL	Produs 2	ÿ subtotal	5237190
Client 1 SRL	Produs 5	01/08/2000	3867500
Client 1 SRL	Produs 5	ÿ subtotal	3867500

Client 1 SRL	ÿsubtotal	01/08/2000	5399625
Client 1 SRL	ÿsubtotal	02/08/2000	1651125
Client 1 SRL	ÿsubtotal	03/08/2000	2320500
Client 1 SRL	ÿsubtotal	04/08/2000	2052750
Client 1 SRL	ÿsubtotal	07/08/2000	1066240
Client 1 SRL	ÿsubtotal	ÿ subtotal	12490240
Client 2 SA	Produs 2	01/08/2000	1160250
Client 2 SA	Produs 2	ÿ subtotal	1160250
Client 2 SA	ÿsubtotal	01/08/2000	1160250
Client 2 SA	ÿsubtotal	ÿ subtotal	1160250
Client 3 SRL	Produs 2	07/08/2000	453985
Client 3 SRL	Produs 2	ÿ subtotal	453985
Client 3 SRL	Produs 3	07/08/2000	333200
Client 3 SRL	Produs 3	ÿ subtotal	333200
Client 3 SRL	Produs 4	07/08/2000	833000
Client 3 SRL	Produs 4	ÿ subtotal	833000
Client 3 SRL	Produs 5	07/08/2000	5622750
Client 3 SRL	Produs 5	ÿ subtotal	5622750
Client 3 SRL	ÿsubtotal	07/08/2000	7242935
Client 3 SRL	ÿsubtotal	ÿ subtotal	7242935
Client 4	Produs 2	07/08/2000	1249500
Client 4	Produs 2	ÿ subtotal	1249500
Client 4	Produs 5	07/08/2000	4188800
Client 4	Produs 5	ÿ subtotal	4188800
Client 4	ÿsubtotal	07/08/2000	5438300
Client 4	ÿsubtotal	ÿ subtotal	5438300
Client 5 SRL	Produs 2	01/08/2000	980560
Client 5 SRL	Produs 2	ÿ subtotal	980560
Client 5 SRL	Produs 3	01/08/2000	357000
Client 5 SRL	Produs 3	ÿ subtotal	357000
Client 5 SRL	ÿsubtotal	01/08/2000	1337560
Client 5 SRL	ÿsubtotal	ÿ subtotal	1337560
Client 6 SA	Produs 2	01/08/2000	891310
Client 6 SA	Produs 2	ÿ subtotal	891310
Client 6 SA	Produs 4	01/08/2000	564060
Client 6 SA	Produs 4	ÿ subtotal	564060
Client 6 SA	Produs 5	01/08/2000	5331200
Client 6 SA	Produs 5	ÿ subtotal	5331200
Client 6 SA	ÿsubtotal	01/08/2000	6786570
Client 6 SA	ÿsubtotal	ÿ subtotal	6786570
Client 7 SRL	Produs 2	02/08/2000	1383375
Client 7 SRL	Produs 2	ÿ subtotal	1383375
Client 7 SRL	ÿsubtotal	02/08/2000	1383375
Client 7 SRL	ÿsubtotal	ÿ subtotal	1383375

În clauza GROUP BY a fost introdus operatorul GROUPING SETS, prin care se precizează două grupuri de agregare, unul alcătuit din combinația (DenCl, DenPr), celălalt din data facturii, astfel încât se obține un rezultat cum este cel din figura 7.14.

Dacă se dorește o situație sintetică cu subtotaluri ale celor trei attribute, plus un total general (obținut prin perechea de paranteze fără argumente), se schimbă clauza GROUP BY astfel:

CLIENT	PRODUS	ZIUA	VINZARI
Client 1 SRL	Produs 1	ÿ-subtotal-	3385550.00
Client 1 SRL	Produs 2	ÿ-subtotal-	5237190.00
Client 1 SRL	Produs 5	ÿ-subtotal-	3867500.00
Client 2 SA	Produs 2	ÿ-subtotal-	1160250.00
Client 3 SRL	Produs 2	ÿ-subtotal-	453895.00
Client 3 SRL	Produs 3	ÿ-subtotal-	333200.00
Client 3 SRL	Produs 4	ÿ-subtotal-	833000.00
Client 3 SRL	Produs 5	ÿ-subtotal-	5622750.00
Client 4	Produs 2	ÿ-subtotal-	1249500.00
Client 4	Produs 5	ÿ-subtotal-	4188000.00
Client 5 SRL	Produs 2	ÿ-subtotal-	980560.00
Client 5 SRL	Produs 3	ÿ-subtotal-	357000.00
Client 6 SA	Produs 2	ÿ-subtotal-	891310.00
Client 6 SA	Produs 4	ÿ-subtotal-	564060.00
Client 6 SA	Produs 5	ÿ-subtotal-	5331200.00
Client 7 SRL	Produs 2	ÿ-subtotal-	1383375.00
ÿ-subtotal-	ÿ-subtotal-	2000-08-01	14684005.00
ÿ-subtotal-	ÿ-subtotal-	2000-08-02	3034500.00
ÿ-subtotal-	ÿ-subtotal-	2000-08-03	2320500.00
ÿ-subtotal-	ÿ-subtotal-	2000-08-04	2052750.00
ÿ-subtotal-	ÿ-subtotal-	2000-08-07	13747475.00
ÿ-subtotal-	ÿ-subtotal-	2000-08-07	35839230.00

Figura 7.14. Clauza GROUPING SETS

```

SELECT
CASE GROUPING (DenCl)
    WHEN 1 THEN CONCAT (CHR(255), ' -subtotal- ')
    ELSE DenCl END AS Client,
CASE GROUPING (DenPr)
    WHEN 1 THEN CONCAT (CHR(255), ' -subtotal- ')
    ELSE DenPr END AS Produs,
CASE GROUPING (CAST( DataFact AS CHAR(14) ) )
    WHEN 1 THEN CONCAT (CHR(255), ' -subtotal- ')
    ELSE CAST( DataFact AS CHAR(14) ) END AS Ziua,
    SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari
FROM PRODUSE P
INNER JOIN LINIIFACT LF ON P.CodPr=LF.Codpr
INNER JOIN FACTURI F ON LF.NrFact=F.NrFact
INNER JOIN CLIENTI C ON F.CodCl=C.CodCl
GROUP BY GROUPING SETS (DenCl, DenPr,
    CAST ( DataFact AS CHAR(14) ), ( ) )
ORDER BY DenCl, DenPr, CAST (DataFact AS CHAR(14) )

```

CLIENT	PRODUS	ZIUA	VINZARI
Client 1 SRL	ÿ-subtotal-	ÿ-subtotal-	12490240.00
Client 2 SA	ÿ-subtotal-	ÿ-subtotal-	1160250.00
Client 3 SRL	ÿ-subtotal-	ÿ-subtotal-	7242935.00
Client 4	ÿ-subtotal-	ÿ-subtotal-	5438300.00
Client 5 SRL	ÿ-subtotal-	ÿ-subtotal-	1337560.00
Client 6 SA	ÿ-subtotal-	ÿ-subtotal-	8786570.00
Client 7 BRL	ÿ-subtotal-	ÿ-subtotal-	1383375.00
ÿ-subtotal-	Produs 1	ÿ-subtotal-	3385550.00
ÿ-subtotal-	Produs 2	ÿ-subtotal-	11356170.00
ÿ-subtotal-	Produs 3	ÿ-subtotal-	690200.00
ÿ-subtotal-	Produs 4	ÿ-subtotal-	1397060.00
ÿ-subtotal-	Produs 5	ÿ-subtotal-	19016250.00
ÿ-subtotal-	ÿ-subtotal-	2000-08-01	14684005.00
ÿ-subtotal-	ÿ-subtotal-	2000-08-02	3034500.00
ÿ-subtotal-	ÿ-subtotal-	2000-08-03	2320500.00
ÿ-subtotal-	ÿ-subtotal-	2000-08-04	2052750.00
ÿ-subtotal-	ÿ-subtotal-	2000-08-07	13747475.00
ÿ-subtotal-	ÿ-subtotal-	ÿ-subtotal-	35839230.00

Figura 7.15. GROUPING SETS – exemplul 2

Cele 18 linii din figura 7.15 sunt obținute astfel: 7 clienți pentru care sunt vânzări, plus 5 produse vândute, plus 5 zile în care s-au întocmit facturi, plus linia totalului general.

Interesant este și faptul că cei trei operatori pot fi combinații. Spre exemplu, dacă GROUP BY are forma:

```

GROUP BY
GROUPING SETS ( ROLLUP (DenCl, DenPr),
    CAST( DataFact AS CHAR(14) ) )

```

prin execuția consultării se obțin liniile din figura 7.16.

CLIENT	PRODUS	ZIUA	VINZARI
Client 1 SRL	Produs 1	ÿ-subtotal-	3385550.00
Client 1 SRL	Produs 2	ÿ-subtotal-	5237190.00
Client 1 SRL	Produs 5	ÿ-subtotal-	3867500.00
Client 1 SRL	ÿ-subtotal-	ÿ-subtotal-	12490240.00
Client 2 SA	Produs 2	ÿ-subtotal-	1160250.00
Client 2 SA	ÿ-subtotal-	ÿ-subtotal-	1160250.00
Client 3 SRL	Produs 2	ÿ-subtotal-	453895.00
Client 3 SRL	Produs 3	ÿ-subtotal-	333200.00
Client 3 SRL	Produs 4	ÿ-subtotal-	833000.00
Client 3 SRL	Produs 5	ÿ-subtotal-	5622750.00
Client 3 SRL	ÿ-subtotal-	ÿ-subtotal-	7242935.00
Client 4	Produs 2	ÿ-subtotal-	1249500.00
Client 4	Produs 5	ÿ-subtotal-	4188000.00
Client 4	ÿ-subtotal-	ÿ-subtotal-	5438300.00
Client 5 SRL	Produs 2	ÿ-subtotal-	980560.00
Client 5 SRL	Produs 3	ÿ-subtotal-	357000.00
Client 5 SRL	Produs 4	ÿ-subtotal-	1337560.00
Client 5 SRL	Produs 5	ÿ-subtotal-	8786570.00
Client 6 SA	Produs 2	ÿ-subtotal-	891310.00
Client 6 SA	Produs 4	ÿ-subtotal-	564060.00
Client 6 SA	Produs 5	ÿ-subtotal-	5331200.00
Client 6 SA	ÿ-subtotal-	ÿ-subtotal-	8786570.00
Client 7 SRL	Produs 2	ÿ-subtotal-	1383375.00
Client 7 SRL	ÿ-subtotal-	ÿ-subtotal-	1383375.00
Client 7 BRL	ÿ-subtotal-	ÿ-subtotal-	1383375.00
ÿ-subtotal-	ÿ-subtotal-	2000-08-01	14684005.00
ÿ-subtotal-	ÿ-subtotal-	2000-08-02	3034500.00
ÿ-subtotal-	ÿ-subtotal-	2000-08-03	2320500.00
ÿ-subtotal-	ÿ-subtotal-	2000-08-04	2052750.00
ÿ-subtotal-	ÿ-subtotal-	2000-08-07	13747475.00
ÿ-subtotal-	ÿ-subtotal-	ÿ-subtotal-	35839230.00

Figura 7.16. GROUPING SETS și ROLLUP

```

SELECT DenCl AS Client, DenPr AS Produs,
CHR(255) || ' subtotal ' AS Ziua,
SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari
FROM PRODUSE P
INNER JOIN LINIIFACT LF ON P.CodPr=LF.Codpr
INNER JOIN FACTURI F ON LF.NrFact=F.NrFact
INNER JOIN CLIENTI C ON F.CodCl=C.CodCl
GROUP BY DenCl, DenPr
UNION
SELECT DenCl AS Client, CHR(255) || ' subtotal ' AS Produs,
CAST( DataFact AS CHAR(14) ) AS Ziua,
SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari
FROM PRODUSE P
INNER JOIN LINIIFACT LF ON P.CodPr=LF.Codpr
INNER JOIN FACTURI F ON LF.NrFact=F.NrFact
INNER JOIN CLIENTI C ON F.CodCl=C.CodCl
GROUP BY DenCl, CAST( DataFact AS CHAR(14) )
UNION
SELECT CHR(255) || ' subtotal ' AS Client, DenPr AS Produs,
CAST( DataFact AS CHAR(14) ) AS Ziua,
SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari
FROM PRODUSE P
INNER JOIN LINIIFACT LF ON P.CodPr=LF.Codpr
INNER JOIN FACTURI F ON LF.NrFact=F.NrFact
INNER JOIN CLIENTI C ON F.CodCl=C.CodCl
GROUP BY DenPr, CAST( DataFact AS CHAR(14) )
UNION
SELECT DenCl AS Client, CHR(255) || ' subtotal ' AS Produs,
CHR(255) || ' subtotal ' AS Ziua,
SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari
FROM PRODUSE P
INNER JOIN LINIIFACT LF ON P.CodPr=LF.Codpr
INNER JOIN FACTURI F ON LF.NrFact=F.NrFact
INNER JOIN CLIENTI C ON F.CodCl=C.CodCl
GROUP BY DenCl
UNION
SELECT CHR(255) || ' subtotal ' AS Client, DenPr AS Produs,
CHR(255) || ' subtotal ' AS Ziua,
SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari
FROM PRODUSE P
INNER JOIN LINIIFACT LF ON P.CodPr=LF.Codpr
INNER JOIN FACTURI F ON LF.NrFact=F.NrFact
INNER JOIN CLIENTI C ON F.CodCl=C.CodCl
GROUP BY DenPr
UNION
SELECT CHR(255) || ' subtotal ' AS Client, DenPr AS Produs,
CHR(255) || ' subtotal ' AS Ziua,
SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari
FROM PRODUSE P
INNER JOIN LINIIFACT LF ON P.CodPr=LF.Codpr
INNER JOIN FACTURI F ON LF.NrFact=F.NrFact
INNER JOIN CLIENTI C ON F.CodCl=C.CodCl
GROUP BY DenPr
UNION
SELECT CHR(255) || ' subtotal ' AS Client, CHR(255) ||
'subtotal ' AS Produs, CAST( DataFact AS CHAR(14) )
AS Ziua,

```

```

SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari
FROM PRODUSE P
INNER JOIN LINIIFACT LF ON P.CodPr=LF.Codpr
INNER JOIN FACTURI F ON LF.NrFact=F.NrFact
INNER JOIN CLIENTI C ON F.CodCl=C.CodCl
GROUP BY CAST( DataFact AS CHAR(14) )
UNION
SELECT CHR(255) || ' subtotal ' AS Client, CHR(255) ||
'subtotal ' AS Produs, CHR(255) || ' subtotal '
AS Ziua,
SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari
FROM PRODUSE P
INNER JOIN LINIIFACT LF ON P.CodPr=LF.Codpr
INNER JOIN FACTURI F ON LF.NrFact=F.NrFact
INNER JOIN CLIENTI C ON F.CodCl=C.CodCl
ORDER BY Client, Produs, Ziua

```

Operatorul GROUPING SETS

Uneori, prea multă detaliere, precum cea din tabelul de mai sus, devine obosită. Prin GROUPING SETS se poate regla granularitatea agregărilor. Dacă, spre exemplu, în interogarea anterioară se dorește urmărirea zilnică a vânzărilor pentru fiecare client (și toate produsele) și pentru fiecare produs (și toți clienții), se poate recurge la varianta următoare:

```

SELECT
CASE GROUPING (DenCl)
WHEN 1 THEN CONCAT (CHR(255), ' -subtotal- ')
ELSE DenCl END AS Client,
CASE GROUPING (DenPr)
WHEN 1 THEN CONCAT (CHR(255), ' -subtotal- ')
ELSE DenPr END AS Produs,
CASE GROUPING (CAST( DataFact AS CHAR(14) ))
WHEN 1 THEN CONCAT (CHR(255), ' -subtotal- ')
ELSE CAST( DataFact AS CHAR(14) ) END AS Ziua,
SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari
FROM PRODUSE P
INNER JOIN LINIIFACT LF ON P.CodPr=LF.Codpr
INNER JOIN FACTURI F ON LF.NrFact=F.NrFact
INNER JOIN CLIENTI C ON F.CodCl=C.CodCl
GROUP BY GROUPING SETS ( (DenCl, DenPr),
CAST( DataFact AS CHAR(14) ) )
ORDER BY DenCl, DenPr, CAST (DataFact AS CHAR(14) )

```

Client 3 SRL	Produs 4	subtotal	833000
Client 3 SRL	Produs 5	07/08/2000	5622750
Client 3 SRL	Produs 5	subtotal	5622750
Client 3 SRL	subtotal	07/08/2000	7242935
Client 3 SRL	subtotal	subtotal	7242935
Client 4	Produs 2	07/08/2000	1249500
Client 4	Produs 2	subtotal	1249500
Client 4	Produs 5	07/08/2000	4188800
Client 4	Produs 5	subtotal	4188800
Client 4	subtotal	07/08/2000	5438300
Client 4	subtotal	subtotal	5438300
Client 5 SRL	Produs 2	01/08/2000	980560
Client 5 SRL	Produs 2	subtotal	980560
Client 5 SRL	Produs 3	01/08/2000	357000
Client 5 SRL	Produs 3	subtotal	357000
Client 5 SRL	subtotal	01/08/2000	1337560
Client 5 SRL	subtotal	subtotal	1337560
Client 6 SA	Produs 2	01/08/2000	891310
Client 6 SA	Produs 2	subtotal	891310
Client 6 SA	Produs 4	01/08/2000	564060
Client 6 SA	Produs 4	subtotal	564060
Client 6 SA	Produs 5	01/08/2000	5331200
Client 6 SA	Produs 5	subtotal	5331200
Client 6 SA	subtotal	01/08/2000	6786570
Client 6 SA	subtotal	subtotal	6786570
Client 7 SRL	Produs 2	02/08/2000	1383375
Client 7 SRL	Produs 2	subtotal	1383375
Client 7 SRL	subtotal	02/08/2000	1383375
Client 7 SRL	subtotal	subtotal	1383375
subtotal	Produs 1	01/08/2000	595000
subtotal	Produs 1	03/08/2000	1130500
subtotal	Produs 1	04/08/2000	1660050
subtotal	Produs 1	subtotal	3385550
subtotal	Produs 2	01/08/2000	3969245
subtotal	Produs 2	02/08/2000	3034500
subtotal	Produs 2	03/08/2000	1190000
subtotal	Produs 2	04/08/2000	392700
subtotal	Produs 2	07/08/2000	2769725
subtotal	Produs 2	subtotal	11356170
subtotal	Produs 3	01/08/2000	357000
subtotal	Produs 3	07/08/2000	333200
subtotal	Produs 3	subtotal	690200

subtotal	Produs 4	01/08/2000	564060
subtotal	Produs 4	07/08/2000	833000
subtotal	Produs 4	subtotal	1397060
subtotal	Produs 5	01/08/2000	9198700
subtotal	Produs 5	07/08/2000	9811550
subtotal	Produs 5	subtotal	19010250
subtotal	subtotal	01/08/2000	14684005
subtotal	subtotal	02/08/2000	3034500
subtotal	subtotal	03/08/2000	2320500
subtotal	subtotal	04/08/2000	2052750
subtotal	subtotal	07/08/2000	13747475
subtotal	subtotal	subtotal	35839230

Obtinerea aceluiși rezultat utilizând operatorul CUBE presupune, în Oracle 8i2, următoarea interogare:

```
SELECT
  CASE WHEN GROUPING (DenCl) = 1
    THEN ' subtotal '
    ELSE DenCl
  END AS Client,
  CASE WHEN GROUPING (DenPr) = 1
    THEN ' subtotal '
    ELSE DenPr
  END AS Produs,
  CASE WHEN GROUPING (TO_CHAR(DataFact, 'DD-MM-YYYY')) = 1
    THEN ' subtotal '
    ELSE TO_CHAR(DataFact, 'DD-MM-YYYY')
  END AS Ziua,
  SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari
FROM PRODUSE P, LINIIFACT LF, FACTURI F, CLIENTI C,
  LOCALITATI L, JUDETE J
WHERE P.CodPr=LF.Codpr AND LF.NrFact=F.NrFact AND
  F.CodCl=C.CodCl AND C.CodPost=L.CodPost AND L.Jud=J.Jud
GROUP BY CUBE (DenCl, DenPr, TO_CHAR(DataFact, 'DD-MM-YYYY'))
```

În absența operatorului CUBE, SQL-92 cere ceva efort de scriere, deși logica interogării este relativ simplă:

```
SELECT DenCl AS Client, DenPr AS Produs,
  CAST( DataFact AS CHAR(14) ) AS Ziua,
  SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari
FROM PRODUSE P
  INNER JOIN LINIIFACT LF ON P.CodPr=LF.Codpr
  INNER JOIN FACTURI F ON LF.NrFact=F.NrFact
  INNER JOIN CLIENTI C ON F.CodCl=C.CodCl
GROUP BY DenCl, DenPr, CAST( DataFact AS CHAR(14) )
UNION
```

```

SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari
FROM PRODUSE P, LINIIFACT LF, FACTURI F, CLIENTI C,
LOCALITATI L, JUDETE J
WHERE P.CodPr=LF.CodPr AND LF.NrFact=F.NrFact AND
F.CodCl=C.CodCl AND C.CodPost=L.CodPost AND L.Jud=J.Jud
GROUP BY CUBE (DenCl, DenPr)
ORDER BY DenCl, DenPr

```

CLIENT	PRODUS	VINZARI
Client 1 SRL	Produs 1	3385550.00
Client 1 SRL	Produs 2	6237190.00
Client 1 SRL	Produs 5	3867500.00
Client 1 SRL	ÿ Total CLIENT	12490240.00
Client 2 SA	Produs 2	1160250.00
Client 2 SA	ÿ Total CLIENT	1160250.00
Client 3 SRL	Produs 2	453985.00
Client 3 SRL	Produs 3	333200.00
Client 3 SRL	Produs 4	833000.00
Client 3 SRL	Produs 5	5622750.00
Client 3 SRL	ÿ Total CLIENT	7242935.00
Client 4	Produs 2	1249500.00
Client 4	Produs 5	4188900.00
Client 4	ÿ Total CLIENT	5438300.00
Client 5 SRL	Produs 2	990560.00
Client 5 SRL	Produs 3	357000.00
Client 5 SRL	ÿ Total CLIENT	1337560.00
Client 6 SA	Produs 2	691310.00
Client 6 SA	Produs 4	564060.00
Client 6 SA	Produs 5	5331200.00
Client 6 SA	ÿ Total CLIENT	6786570.00
Client 7 SRL	Produs 2	1383375.00
Client 7 SRL	ÿ Total CLIENT	1383375.00
ÿ Total PRODUS	Produs 1	3385550.00
ÿ Total PRODUS	Produs 2	11356170.00
ÿ Total PRODUS	Produs 3	690200.00
ÿ Total PRODUS	Produs 4	1397060.00
ÿ Total PRODUS	Produs 5	19010250.00
ÿ TOTAL	ÿ GENERAL	35839230.00

Figura 7.13. Varianta DB2 de utilizare a operatorului CUBE

Atunci cand analiza se intreprinde pe trei axe: *clienti, produse si zile*, interogarea DB2 se schimba astfel:

```

SELECT
CASE GROUPING (DenCl)
WHEN 1 THEN CONCAT (CHR(255), ' subtotal ')
ELSE DenCl
END AS Client,
CASE GROUPING (DenPr)
WHEN 1 THEN CONCAT (CHR(255), ' subtotal ')
ELSE DenPr
END AS Produs,

```

```

CASE GROUPING (CAST( DataFact AS CHAR(14) ) )
WHEN 1 THEN CONCAT (CHR(255), ' subtotal ')
ELSE CAST( DataFact AS CHAR(14) )
END AS Ziua,
SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari
FROM PRODUSE P
INNER JOIN LINIIFACT LF ON P.CodPr=LF.CodPr
INNER JOIN FACTURI F ON LF.NrFact=F.NrFact
INNER JOIN CLIENTI C ON F.CodCl=C.CodCl
GROUP BY CUBE (DenCl, DenPr, CAST( DataFact AS CHAR(14) ) )
ORDER BY DenCl, DenPr, CAST (DataFact AS CHAR(14) )

```

Prin CAST s-a realizat conversia din data calendaristica in sir de caractere. Rezultatul, fiind mai lung de o pagină, este afisat sub forma de tabel. Cu acest prilej, l-am mai cosmetizat un pic.

Tabelul 7.1. Analiză pe trei dimensiuni

CLIENT	PRODUS	ZIUA	VINZARI
Client 1 SRL	Produs 1	01/08/2000	595000
Client 1 SRL	Produs 1	03/08/2000	1130500
Client 1 SRL	Produs 1	04/08/2000	1660050
Client 1 SRL	Produs 1	subtotal	3385550
Client 1 SRL	Produs 2	01/08/2000	937125
Client 1 SRL	Produs 2	02/08/2000	1651125
Client 1 SRL	Produs 2	03/08/2000	1190000
Client 1 SRL	Produs 2	04/08/2000	392700
Client 1 SRL	Produs 2	07/08/2000	1066240
Client 1 SRL	Produs 2	subtotal	5237190
Client 1 SRL	Produs 5	01/08/2000	3867500
Client 1 SRL	Produs 5	subtotal	3867500
Client 1 SRL	subtotal	01/08/2000	5399625
Client 1 SRL	subtotal	02/08/2000	1651125
Client 1 SRL	subtotal	03/08/2000	2320500
Client 1 SRL	subtotal	04/08/2000	2052750
Client 1 SRL	subtotal	07/08/2000	1066240
Client 1 SRL	subtotal	subtotal	12490240
Client 2 SA	Produs 2	01/08/2000	1160250
Client 2 SA	Produs 2	subtotal	1160250
Client 2 SA	subtotal	01/08/2000	1160250
Client 2 SA	subtotal	subtotal	1160250
Client 3 SRL	Produs 2	07/08/2000	453985
Client 3 SRL	Produs 2	subtotal	453985
Client 3 SRL	Produs 3	07/08/2000	333200
Client 3 SRL	Produs 3	subtotal	333200
Client 3 SRL	Produs 4	07/08/2000	833000

→ subtotal pe zile
→ subtotal pe produse

```

SELECT CHR(255) AS Client, CHR(255) AS Produs,
       SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari
  FROM PRODUSE P
 INNER JOIN LINIIFACT LF ON P.CodPr=LF.Codpr
 INNER JOIN FACTURI F ON LF.NrFact=F.NrFact
 INNER JOIN CLIENTI C ON F.CodCl=C.CodCl
 ORDER BY Client, Produs
  
```

CLIENT	PRODUS	VINZARI
Client 1 SRL	Produs 1	3385550.00
Client 1 SRL	Produs 2	5237180.00
Client 1 SRL	Produs 5	3867500.00
Client 1 SRL		12490240.00
Client 2 SA	Produs 2	1160250.00
Client 2 SA		1160250.00
Client 3 SRL	Produs 2	453985.00
Client 3 SRL	Produs 3	333200.00
Client 3 SRL	Produs 4	833000.00
Client 3 SRL	Produs 5	5822750.00
Client 3 SRL		7242935.00
Client 4	Produs 2	1249500.00
Client 4	Produs 5	4188800.00
Client 4		5438300.00
Client 5 SRL	Produs 2	980560.00
Client 5 SRL	Produs 3	357000.00
Client 5 SRL		1337560.00
Client 6 SA	Produs 2	891310.00
Client 6 SA	Produs 4	564060.00
Client 6 SA	Produs 5	5331200.00
Client 6 SA		6788570.00
Client 7 SRL	Produs 2	1383375.00
Client 7 SRL		1383375.00
	Produs 1	3385550.00
	Produs 2	11356170.00
	Produs 3	890200.00
	Produs 4	1397060.00
	Produs 5	19010250.00
		35839230.00

Figura 7.12. Analiză pe două dimensiuni

Pentru astfel de situații există un operator cuceritor prin simplitate: CUBE. Cu ajutorul acestuia, se poate reda următoarea variantă DB2, valabilă și Oracle 8i2 (schimbând sintaxa juncțiunii):

```

SELECT DenCl AS Client, DenPr AS Produs,
       SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari
  FROM PRODUSE P
 INNER JOIN LINIIFACT LF ON P.CodPr=LF.Codpr
 INNER JOIN FACTURI F ON LF.NrFact=F.NrFact
 INNER JOIN CLIENTI C ON F.CodCl=C.CodCl
 GROUP BY CUBE (DenCl, DenPr)
 ORDER BY DenCl, DenPr
  
```

Un plus de atraktivitate a rezultatului presupune folosirea clauzei GROUPING astfel:

```

SELECT
CASE GROUPING (DenCl)
WHEN 1 THEN CASE GROUPING (DenPr)
WHEN 1 THEN CHR(255) || CHR(255) ||
' TOTAL '
ELSE CHR(255) || ' Total PRODUS'
END
ELSE DenCl
END AS Client,
CASE GROUPING (DenPr)
WHEN 1 THEN CASE GROUPING (DenCl)
WHEN 1 THEN CHR(255) || CHR(255) ||
' GENERAL '
ELSE CHR(255) || ' Total CLIENT '
END
ELSE DenPr
END AS Produs,
SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari
FROM PRODUSE P
INNER JOIN LINIIFACT LF ON P.CodPr=LF.Codpr
INNER JOIN FACTURI F ON LF.NrFact=F.NrFact
INNER JOIN CLIENTI C ON F.CodCl=C.CodCl
GROUP BY CUBE (DenCl, DenPr)
ORDER BY DenCl, DenPr
  
```

mfd de produs

Drept răspuns, forma raportului este cea din figura 7.13. Ultima linie este cea a totalului general. Se cuvine de adăugat că atributele care definesc dimensiunile de analiză trebuie să fie de același tip. Varianta Oracle este în același gen, doar că, în afară de juncțiunii, trebuie modificat și modul de scriere a sevenței CASE din fraza DB2:

```

SELECT
CASE WHEN GROUPING (DenCl) = 1
THEN CASE WHEN GROUPING (DenPr) = 1
THEN CHR(255) || CHR(255) || ' TOTAL '
ELSE CHR(255) || ' Total PRODUS'
END
ELSE DenCl
END AS Client,
CASE WHEN GROUPING (DenPr) = 1
THEN CASE WHEN GROUPING (DenCl) = 1
THEN CHR(255) || CHR(255) || ' GENERAL '
ELSE CHR(255) || ' Total CLIENT '
END
ELSE DenPr
END AS Produs,
  
```

JUDET	LOC	DENCL	FACTURA	VINZARI
Iasi	Iasi	Client 1 SRL	1111	5399625
Iasi	Iasi	Client 1 SRL	1115	1651125
Iasi	Iasi	Client 1 SRL	1117	2320500
Iasi	Iasi	Client 1 SRL	1118	2052750
Iasi	Iasi	Client 1 SRL	1120	1068240
Iasi	Iasi	Client 1 SRL	Subtotal CLIENT Client 1 SRL	12490240
Iasi	Iasi		Subtotal CLIENT Client 1 SRL	12490240
Iasi	Iasi	Client 2 SA	1113	1160250
Iasi	Iasi	Client 2 SA	Subtotal CLIENT Client 2 SA	1160250
Iasi	Iasi		Subtotal CLIENT Client 2 SA	1160250
Iasi	Pescari	Client 4	1121	5438300
Iasi	Pescari	Client 4	Subtotal CLIENT Client 4	5438300
Iasi	Iasi	Subtotal JUDET Iasi	Subtotal CLIENT Client 4	5438300
Neamt	Roman	Client 5 SA	1114	6788570
Neamt	Roman	Client 5 SA	Subtotal CLIENT Client 5 SA	6788570
Neamt	Neamt	Subtotal JUDET Neamt	Subtotal CLIENT Client 5 SA	6788570
Timis	Timisoara	Client 5 SRL	1112	1337560
Timis	Timisoara	Client 5 SRL	Subtotal CLIENT Client 5 SRL	1337560
Timis	Timisoara		Subtotal CLIENT Client 5 SRL	1337560
Timis	Timisoara	Client 7 SRL	1116	1383375
Timis	Timisoara	Client 7 SRL	Subtotal CLIENT Client 7 SRL	1383375
Vaslui	Vaslui	Subtotal JUDET Timis	Subtotal CLIENT Client 7 SRL	1383375
Vaslui	Vaslui	Client 3 SRL	1119	7242935
Vaslui	Vaslui	Client 3 SRL	Subtotal CLIENT Client 3 SRL	7242935
Vaslui	Vaslui	Subtotal JUDET Vaslui	Subtotal CLIENT Client 3 SRL	7242935

Figura 7.11. Un ultim exemplu de ROLLUP parțial în Oracle 8i2

```

SELECT NVL(Judet, ' TOTAL GENERAL') AS Judet,
CASE WHEN Judet IS NULL
    THEN ''
    ELSE NVL(Loc, ' Subtotal JUDET ' || Judet)
END AS Loc,
CASE WHEN Loc IS NULL
    THEN ''
    ELSE NVL(DenCl, ' Subtotal LOCALITATE ' || Loc)
END AS DenCl,
CASE WHEN DenCl IS NULL
    THEN ''
    ELSE NVL(TO_CHAR(F.NrFact,'99999999'), CHR(255) ||
        ' Subtotal CLIENT ' || RTRIM(DenCl))
END AS Factura,
SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari
FROM PRODUSE P, LINIIFACT LF ON P.CodPr=LF.Codpr
INNER JOIN FACTURI F ON LF.NrFact=F.NrFact
INNER JOIN CLIENTI C ON F.CodCl=C.CodCl
GROUP BY DenCl, DenPr
UNION
SELECT DenCl AS Client, CHR(255) AS Produs,
SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari
FROM PRODUSE P
INNER JOIN LINIIFACT LF ON P.CodPr=LF.Codpr
INNER JOIN FACTURI F ON LF.NrFact=F.NrFact
INNER JOIN CLIENTI C ON F.CodCl=C.CodCl
GROUP BY DenCl
UNION
SELECT CHR(255) AS Client, DenPr AS Produs,
SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari
FROM PRODUSE P
INNER JOIN LINIIFACT LF ON P.CodPr=LF.Codpr
INNER JOIN FACTURI F ON LF.NrFact=F.NrFact
INNER JOIN CLIENTI C ON F.CodCl=C.CodCl
GROUP BY DenPr
UNION

```

7.2. Analize multidimensionale. Operatorii CUBE și GROUPING SETS

De multe ori, pentru a produce o impresie mai puternică partenerilor de discuții, analiza datelor trebuie să fie una multidimensională. Apanaj, prin excelență, al... Excel-ului sau pretențioasele instrumente OLAP, acest paragraf este menit să demonstreze că, atât prin SQL-ul clasic (SQL-92), dar mai ales prin SQL-99, sunt create condiții ideale pentru acest gen de operațiuni.

Pentru început, interesează vizualizarea vânzărilor pe două axe, produse și clienti, ca în figura 7.12. Întrucât produse și clienti sunt două variabile independente, vor exista patru variante de grupare a datelor:

- grupare după client și produs,
- grupare numai după client,
- grupare numai după produs și
- un grup pentru total general.

O soluție în sintaxa SQL-92 este impresionantă, ca întindere, iar dacă analiza se realizează după trei, patru... variabile, impresionantul se transformă în halucinant. Iată prima interogare DB2 pentru obținerea unui raport de genul celui din figura 7.12.

```

SELECT DenCl AS Client, DenPr AS Produs,
SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari
FROM PRODUSE P
INNER JOIN LINIIFACT LF ON P.CodPr=LF.Codpr
INNER JOIN FACTURI F ON LF.NrFact=F.NrFact
INNER JOIN CLIENTI C ON F.CodCl=C.CodCl
GROUP BY DenCl, DenPr
UNION
SELECT DenCl AS Client, CHR(255) AS Produs,
SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari
FROM PRODUSE P
INNER JOIN LINIIFACT LF ON P.CodPr=LF.Codpr
INNER JOIN FACTURI F ON LF.NrFact=F.NrFact
INNER JOIN CLIENTI C ON F.CodCl=C.CodCl
GROUP BY DenCl
UNION
SELECT CHR(255) AS Client, DenPr AS Produs,
SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari
FROM PRODUSE P
INNER JOIN LINIIFACT LF ON P.CodPr=LF.Codpr
INNER JOIN FACTURI F ON LF.NrFact=F.NrFact
INNER JOIN CLIENTI C ON F.CodCl=C.CodCl
GROUP BY DenPr
UNION

```

Judet	Loc	DenCl	Factura	Vinzari
Iasi	Iasi	Client 1 SRL	1111	5399625
Iasi	Iasi	Subtotal LOCALITATE Iasi		5399625
Iasi	Subtotal JUDET Iasi			5399625
Iasi	Iasi	Client 2 SA	1113	1160250
Iasi	Iasi	Subtotal LOCALITATE Iasi		1160250
Iasi	Subtotal JUDET Iasi			1160250
Iasi	Iasi	Client 1 SRL	1115	1651125
Iasi	Iasi	Subtotal LOCALITATE Iasi		1651125
Iasi	Subtotal JUDET Iasi			1651125
Iasi	Iasi	Client 1 SRL	1117	2320500
Iasi	Iasi	Subtotal LOCALITATE Iasi		2320500
Iasi	Subtotal JUDET Iasi			2320500
Iasi	Iasi	Client 1 SRL	1118	2052750
Iasi	Iasi	Subtotal LOCALITATE Iasi		2052750
Iasi	Subtotal JUDET Iasi			2052750
Iasi	Iasi	Client 1 SRL	1120	1066240
Iasi	Iasi	Subtotal LOCALITATE Iasi		1066240
Iasi	Subtotal JUDET Iasi			1066240
Iasi	Pascani	Client 4	1121	5438300
Iasi	Pascani	Subtotal LOCALITATE Pascani		5438300
Iasi	Subtotal JUDET Iasi			5438300
Neamt	Roman	Client 6 SA	1114	6786570
Neamt	Roman	Subtotal LOCALITATE Roman		6786570
Neamt	Subtotal JUDET Neamt			6786570
Timis	Timisoara	Client 5 SRL	1112	1337560
Timis	Timisoara	Subtotal LOCALITATE Timisoara		1337560
Timis	Subtotal JUDET Timis			1337560
Timis	Timisoara	Client 7 SRL	1116	1383375
Timis	Timisoara	Subtotal LOCALITATE Timisoara		1383375
Timis	Subtotal JUDET Timis			1383375
Vaslui	Vaslui	Client 3 SRL	1119	7242935
Vaslui	Vaslui	Subtotal LOCALITATE Vaslui		7242935
Vaslui	Subtotal JUDET Vaslui			7242935

Figura 7.9. Un alt ROLLUP parțial, dar ceva mai curios

Rezultatul este destul de anapoda, dar nici interogarea nu-i mai prejos! Practic, după fiecare valoare distinctă a combinației atributelor din GROUP BY se calculează un subtotal pe județ-localitate și altul pe județ (deoarece attributele din ROLLUP sunt DenCl și Loc).

Varianta DB2, obținută prin schimbarea funcției NVL cu VALUE, conduce la aceeași linii rezultat, dar într-o cu totul altă ordine, după cum se observă în figura 7.10.

SELECT VALUE(Judet, CHR(255) || ' TOTAL GENERAL') AS Judet,
CASE WHEN Judet IS NULL

THEN ''
ELSE VALUE(Loc, CHR(255) || ' Subtotal JUDET '

|| Judet)

END AS Loc,

CASE WHEN Loc IS NULL

THEN ''
ELSE VALUE(DenCl, CHR(255) ||

' Subtotal LOCALITATE ' || Loc)

END AS DenCl,

```
CASE WHEN DenCl IS NULL THEN ''
ELSE VALUE(CAST (F.NrFact AS CHAR(8)), CHR(255) || ''
Subtotal CLIENT ' || RTRIM(DenCl))
END AS Factura,
SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari
FROM PRODUSE P, LINIIFACT LF, FACTURI F, CLIENTI C,
LOCALITATI L, JUDETE J
WHERE P.CodPr=LF.CodPr AND LF.NrFact=F.NrFact AND
F.CodCl=C.CodCl AND C.CodPost=L.CodPost AND L.Jud=J.Jud
GROUP BY Judet, ROLLUP(Loc, DenCl), F.NrFact
```

JUDET	LOC	DENCL	FACTURA	VINZARI
Iasi	Subtotal JUDET Iasi			5399625
Iasi	Subtotal JUDET Iasi			1160250
Iasi	Subtotal JUDET Iasi			1651125
Iasi	Subtotal JUDET Iasi			2320500
Iasi	Subtotal JUDET Iasi			2052750
Iasi	Subtotal JUDET Iasi			1066240
Iasi	Subtotal JUDET Iasi			5438300
Iasi	Iasi	Subtotal LOCALITATE Iasi		5399625
Iasi	Iasi	Subtotal LOCALITATE Iasi		1160250
Iasi	Iasi	Subtotal LOCALITATE Iasi		1651125
Iasi	Iasi	Subtotal LOCALITATE Iasi		2320500
Iasi	Iasi	Subtotal LOCALITATE Iasi		2052750
Iasi	Iasi	Subtotal LOCALITATE Iasi		1066240
Iasi	Pascani	Client 4	1121	5438300
Iasi	Pascani	Subtotal LOCALITATE Pascani		5438300
Iasi	Subtotal JUDET Iasi			5438300
Neamt	Roman	Client 6 SA	1114	6786570
Neamt	Roman	Subtotal LOCALITATE Roman		6786570
Neamt	Subtotal JUDET Neamt			6786570
Timis	Subtotal JUDET Timis			1337560
Timis	Subtotal JUDET Timis			1383375
Timis	Timisoara	Subtotal LOCALITATE Timisoara		1337560
Timis	Timisoara	Subtotal LOCALITATE Timisoara		1383375
Timis	Timisoara	Client 5 SRL	1112	1337560
Timis	Timisoara	Client 7 SRL	1116	1383375
Vaslui	Subtotal JUDET Vaslui			7242935
Vaslui	Vaslui	Subtotal LOCALITATE Vaslui		7242935
Vaslui	Vaslui	Client 3 SRL	1119	7242935

Figura 7.10. Dispunerea liniilor în DB2 în un ROLLUP parțial

Practic, este cu neputință să obținem aceeași dispunere a liniilor ca în Oracle.

În finalul paragrafului, pentru o deplină edificare asupra mecanismului de execuție a operațiunii de ROLLUP parțial, prezintăm o ultimă interogare Oracle 8i2 și rezultatul acestia (figura 7.11).

JUDET	LOC	DENCL	FACTURA	VINZARI
Iasi	Iasi	Client 1 SRL	1111	5399825
Iasi	Iasi	Client 1 SRL	1115	1651125
Iasi	Iasi	Client 1 SRL	1117	2320500
Iasi	Iasi	Client 1 SRL	1118	2052750
Iasi	Iasi	Client 1 SRL	1120	1066240
Iasi	Iasi	Client 1 SRL	Subtotal CLIENT Client 1 SRL	12490240
Iasi	Iasi	Client 2 SA	1113	1160250
Iasi	Iasi	Client 2 SA	Subtotal CLIENT Client 2 SA	1160250
Iasi	Iasi	Subtotal LOCALITATE Iasi		13650490
Iasi	Pescari	Client 4	1121	5438300
Iasi	Pescari	Client 4	Subtotal CLIENT Client 4	5438300
Iasi	Pescari	Subtotal LOCALITATE Pescari		5438300
Neamt	Roman	Client 6 SA	1114	6786570
Neamt	Roman	Client 6 SA	Subtotal CLIENT Client 6 SA	6786570
Neamt	Roman	Subtotal LOCALITATE Roman		6786570
Timis	Timisoara	Client 5 SRL	1112	1337560
Timis	Timisoara	Client 5 SRL	Subtotal CLIENT Client 5 SRL	1337560
Timis	Timisoara	Client 7 SRL	1116	1383375
Timis	Timisoara	Client 7 SRL	Subtotal CLIENT Client 7 SRL	1383375
Timis	Timisoara	Subtotal LOCALITATE Timisoara		2720935
Vaslui	Vaslui	Client 3 SRL	1119	7242935
Vaslui	Vaslui	Client 3 SRL	Subtotal CLIENT Client 3 SRL	7242935
Vaslui	Vaslui	Subtotal LOCALITATE Vaslui		7242935

Figura 7.8. ROLLUP parțial

```

SELECT NVL(Judet, CHR(255) || ' TOTAL GENERAL') AS Judet,
CASE WHEN Judet IS NULL
    THEN ''
    ELSE NVL(Loc, CHR(255) || ' Subtotal JUDET ' || -
        Judet)
    END AS Loc,
CASE WHEN Loc IS NULL
    THEN ''
    ELSE NVL(DenCl, CHR(255) || ' Subtotal LOCALITATE ' -
        || Loc)
    END AS DenCl,
CASE WHEN DenCl IS NULL
    THEN ''
    ELSE NVL(TO_CHAR(F.NrFact,'99999999'), CHR(255) || -
        ' Subtotal CLIENT ' || RTRIM(DenCl))
    END AS Factura,
SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari
FROM PRODUSE P, LINIIFACT LF, FACTURI F, CLIENTI C,
LOCALITATI L, JUDETE J
WHERE P.CodPr=LF.Codpr AND LF.NrFact=F.NrFact AND
F.CodCl=C.CodCl AND C.CodPost=L.CodPost AND L.Jud=J.Jud
GROUP BY Judet, ROLLUP(Loc, DenCl), F.NrFact

```

```

FROM PRODUSE P, LINIIFACT LF, FACTURI F, CLIENTI C,
LOCALITATI L, JUDETE J
WHERE P.CodPr=LF.Codpr AND LF.NrFact=F.NrFact AND
F.CodCl=C.CodCl AND C.CodPost=L.CodPost AND L.Jud=J.Jud
GROUP BY Judet, Loc, ROLLUP (DenCl, F.NrFact)
ORDER BY Judet, Loc, DenCl, F.NrFact

```

Prin clauza GROUP BY Judet, Loc, ROLLUP (DenCl, F.NrFact) se vor calcula două tipuri de subtotaluri:

- unul pentru valori distincte ale combinației (Judet, Loc și DenCl) și
- altul pentru valori distincte ale combinației (Judet, Loc).

Reținem deci: o coloană declarată în ROLLUP determină calcularea subtotalurilor pentru valori distincte ale celorlalte coloane. Astfel, interogarea următoare produce în Oracle 8i2 lista din figura 7.9.

```

SELECT NVL(Judet, CHR(255) || ' TOTAL GENERAL') AS Judet,
CASE WHEN Judet IS NULL
    THEN ''
    ELSE NVL(Loc, CHR(255) || ' Subtotal JUDET ' || -
        Judet)
    END AS Loc,
CASE WHEN Loc IS NULL
    THEN ''
    ELSE NVL(DenCl, CHR(255) || ' Subtotal LOCALITATE ' -
        || Loc)
    END AS DenCl,
CASE WHEN DenCl IS NULL
    THEN ''
    ELSE NVL(TO_CHAR(F.NrFact,'99999999'), CHR(255) || -
        ' Subtotal CLIENT ' || RTRIM(DenCl))
    END AS Factura,
SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari
FROM PRODUSE P, LINIIFACT LF, FACTURI F, CLIENTI C,
LOCALITATI L, JUDETE J
WHERE P.CodPr=LF.Codpr AND LF.NrFact=F.NrFact AND
F.CodCl=C.CodCl AND C.CodPost=L.CodPost AND L.Jud=J.Jud
GROUP BY Judet, ROLLUP(Loc, DenCl), F.NrFact

```

Combinarea opțiunilor ROLLUP și GROUPING

Structura alternativă de tip CASE poate utiliza și o nouă funcție introdusă în SQL-99 – GROUPING. Argumentul acestela este coloana de grupare, rezultatul fiind 1 atunci când coloana respectivă este inclusă într-un grup de agregare superior, sau 0 pentru linile „normale” (din afara subtotalurilor). Pentru edificare, vezi SELECT-ul următor, ce produce rezultatul din figura 7.7.

```
SELECT Judet, Loc, DenCl,
       SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari,
       GROUPING (Judet) AS Grup_Judet,
       GROUPING (Loc) AS Grup_Loc,
       GROUPING (DenCl) AS Grup_DenCl
FROM PRODUSE P
INNER JOIN LINIIFACT LF ON P.CodPr=LF.Codpr
INNER JOIN FACTURI F ON LF.NrFact=F.NrFact
INNER JOIN CLIENTI C ON F.CodCl=C.CodCl
INNER JOIN LOCALITATI L ON C.CodPost=L.CodPost
INNER JOIN JUDETE J ON L.Jud=J.Jud
GROUP BY ROLLUP (Judet, Loc, DenCl)
ORDER BY Judet, Loc, DenCl
```

JUDET	LOC	DENCL	VINZARI	GRUP_JUDET	GRUP_LOC	GRUP_DENCL
Iasi	Iasi	Client 1 SRL	12490240.00	0	0	0
Iasi	Iasi	Client 2 SA	1160250.00	0	0	0
Iasi	Iasi		13650490.00	0	0	1
Iasi	Pascari	Client 4	5438300.00	0	0	0
Iasi	Pascari		5438300.00	0	0	1
Iasi			19088790.00	0	1	1
Neamt	Roman	Client 6 SA	6788570.00	0	0	0
Neamt	Roman		6788570.00	0	0	1
Neamt			6788570.00	0	1	1
Timis	Timisoara	Client 5 SRL	1337560.00	0	0	0
Timis	Timisoara	Client 7 SRL	1383375.00	0	0	0
Timis	Timisoara		2720935.00	0	0	1
Timis			2720935.00	0	1	1
Vaslui	Vaslui	Client 3 SRL	7242935.00	0	0	0
Vaslui	Vaslui		7242935.00	0	0	1
Vaslui			7242935.00	0	1	1
Vaslui			35839230.00	1	1	1

Figura 7.7. Valori întoarse de funcția GROUPING

Valoarea 1 întoarsă de funcția GROUPING pentru un atribut indică un subtotal pentru acel atribut, relativ la atributele din stânga (în ordinea declarată în clauza ROLLUP). Pe baza funcției GROUPING, rezultatul din figura 7.5 poate fi obținut în DB2 și astfel:

```
SELECT
CASE
WHEN GROUPING (Judet) = 1 THEN CHR(255) || ' TOTAL GENERAL'
ELSE Judet
END AS Judet,
CASE
WHEN GROUPING (Loc) = 1 THEN CHR(255) || ' Subtotal JUDET ' || Judet
ELSE Loc
END AS Loc,
CASE
WHEN GROUPING (DenCl) = 1 THEN CHR(255) || ' Subtotal LOCALITATE ' || DenCl
ELSE DenCl
END AS DenCl,
SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari
FROM PRODUSE P
INNER JOIN LINIIFACT LF ON P.CodPr=LF.Codpr
INNER JOIN FACTURI F ON LF.NrFact=F.NrFact
INNER JOIN CLIENTI C ON F.CodCl=C.CodCl
INNER JOIN LOCALITATI L ON C.CodPost=L.CodPost
INNER JOIN JUDETE J ON L.Jud=J.Jud
GROUP BY ROLLUP (Judet, Loc, DenCl)
ORDER BY Judet, Loc, DenCl
```

ROLLUP-uri parțiale

Este de dorit, uneori, ca subtotalizarea atributelor de grupare să fie parțială: fie nu interesează totalul general, fie subtotalizarea este necesară numai pentru anumite attribute sau grupuri de attribute. Complicăm un pic problema, în sensul că acum vânzările interesează pe facturi, clienti, localități și județe. Subtotalurile însă, trebuie calculate numai la nivel de client și localitate. Interrogarea ce furnizează răspunsul (figura 7.8) este în Oracle 8i2:

```

SELECT COALESCE(Judet, CHR(255) || ' TOTAL GENERAL')
      AS Judet,
      COALESCE(Loc, CHR(255) || ' Subtotal JUDET ' ||
      Judet) AS Loc,
      COALESCE(DenCl, CHR(255) || ' Subtotal LOCALITATE ' ||
      Loc) AS DenCl,
      SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari
FROM PRODUSE P
INNER JOIN LINIIFACT LF ON P.CodPr=LF.Codpr
INNER JOIN FACTURI F ON LF.NrFact=F.NrFact
INNER JOIN CLIENTI C ON F.CodCl=C.CodCl
INNER JOIN LOCALITATI L ON C.CodPost=L.CodPost
INNER JOIN JUDETE J ON L.Jud=J.Jud
GROUP BY ROLLUP (Judet, Loc, DenCl)
ORDER BY Judet, Loc, DenCl

```

și rezultatul în figura 7.5.

Interesant este că aceeași soluție, transcrisă în sintaxa Oracle 8i2, conduce la un rezultat diferit în ceea ce privește liniile subtotalurilor, după cum se observă în figura 7.6.

```

SELECT NVL(Judet, CHR(255) || ' TOTAL GENERAL') AS Judet,
      NVL(Loc, CHR(255) || ' Subtotal JUDET ' || Judet)
      AS Loc,
      NVL(DenCl, CHR(255) || ' Subtotal LOCALITATE ' ||
      Loc) AS DenCl,
      SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari
FROM PRODUSE P, LINIIFACT LF, FACTURI F, CLIENTI C,
      LOCALITATI L, JUDETE J
WHERE P.CodPr=LF.Codpr AND LF.NrFact=F.NrFact AND
      F.CodCl=C.CodCl AND C.CodPost=L.CodPost AND L.Jud=J.Jud
GROUP BY ROLLUP (Judet, Loc, DenCl)
ORDER BY Judet, Loc, DenCl

```

JUDET	LOC	DENCL	VINZARI
Iasi	Iasi	Client 1 SRL	12490240
Iasi	Iasi	Client 2 SA	1160250
Iasi	Iasi	Subtotal LOCALITATE Iasi	13650490
Iasi	Pascari	Client 4	5438300
Iasi	Pascari	Subtotal LOCALITATE Pascari	5438300
Iasi	Subtotal JUDET Iasi	Subtotal LOCALITATE	19088790
Neamt	Roman	Client 6 SA	6786570
Neamt	Roman	Subtotal LOCALITATE Roman	6786570
Neamt	Subtotal JUDET Neamt	Subtotal LOCALITATE	6786570
Timis	Timisoara	Client 5 SRL	1337560
Timis	Timisoara	Client 7 SRL	1383375
Timis	Timisoara	Subtotal LOCALITATE Timisoara	2720935
Timis	Subtotal JUDET Timis	Subtotal LOCALITATE	2720935
Vaslui	Vaslui	Client 3 SRL	7242935
Vaslui	Vaslui	Subtotal LOCALITATE Vaslui	7242935
Vaslui	Subtotal JUDET Vaslui	Subtotal LOCALITATE	7242935
TOTAL GENERAL	Subtotal JUDET	Subtotal LOCALITATE	35839230

Figura 7.5. ROLLUP cu marcarea explicită a subtotalurilor

JUDET	LOC	DENCL	VINZARI
Iasi	Iasi	Client 1 SRL	12490240
Iasi	Iasi	Client 2 SA	1160250
Iasi	Iasi	Subtotal LOCALITATE Iasi	13650490
Iasi	Pascari	Client 4	5438300
Iasi	Pascari	Subtotal LOCALITATE Pascari	5438300
Iasi	Subtotal JUDET Iasi	Subtotal LOCALITATE	19088790
Neamt	Roman	Client 6 SA	6786570
Neamt	Roman	Subtotal LOCALITATE Roman	6786570
Neamt	Subtotal JUDET Neamt	Subtotal LOCALITATE	6786570
Timis	Timisoara	Client 5 SRL	1337560
Timis	Timisoara	Client 7 SRL	1383375
Timis	Timisoara	Subtotal LOCALITATE Timisoara	2720935
Timis	Subtotal JUDET Timis	Subtotal LOCALITATE	2720935
Vaslui	Vaslui	Client 3 SRL	7242935
Vaslui	Vaslui	Subtotal LOCALITATE Vaslui	7242935
Vaslui	Subtotal JUDET Vaslui	Subtotal LOCALITATE	7242935
TOTAL GENERAL	Subtotal JUDET	Subtotal LOCALITATE	35839230

Figura 7.6. Rezultat diferit al aplicării ROLLUP în Oracle 8i2, față de DB2 v7

Raportul din figura 7.6 obținut în Oracle are un inconvenient: la subtotaluri pe județe apare, oarecum anapoda, și un *Subtotal LOCALITATE*. Lucrul acesta poate fi corectat printr-o structură de tip CASE, deși nici DECODE-ul nu-i de lepădat.

```

SELECT NVL(Judet, CHR(255) || ' TOTAL GENERAL') AS Judet,
CASE WHEN Judet IS NULL
      THEN ''
      ELSE NVL(Loc, CHR(255) || ' Subtotal JUDET ' ||
      Judet)
      END AS Loc,
CASE WHEN Loc IS NULL
      THEN ''
      ELSE NVL(DenCl, CHR(255) || ' Subtotal LOCALITATE ' ||
      Loc)
      END AS DenCl,
      SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari
FROM PRODUSE P, LINIIFACT LF, FACTURI F, CLIENTI C,
      LOCALITATI L, JUDETE J
WHERE P.CodPr=LF.Codpr AND LF.NrFact=F.NrFact AND
      F.CodCl=C.CodCl AND C.CodPost=L.CodPost AND L.Jud=J.Jud
GROUP BY ROLLUP (Judet, Loc, DenCl)
ORDER BY Judet, Loc, DenCl

```

De această dată, lista are chiar forma (și conținutul) ca în figura 7.5.

```

UNION
SELECT CONCAT (CHR(255), ' TOTAL GENERAL') AS Judet,
       AS Loc, '' AS DenCl,
       SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari
  FROM PRODUSE P, LINIIFACT LF, FACTURI F,
  CLIENTI C, LOCALITATI L, JUDETE J
 WHERE P.CodPr=LF.CodPr AND LF.NrFact=F.NrFact AND
       F.CodCl=C.CodCl AND C.CodPost=L.CodPost AND L.Jud=J.Jud
 ORDER BY Judet, Loc, DenCl

```

JUDET	LOC	DENCL	VINZARI
Iasi	Iasi	Client 1 SRL	12490240.00
Iasi	Iasi	Client 2 SA	1160250.00
Iasi	Iasi	Subtotal LOCALITATE	13650490.00
Iasi	Pascani	Client 4	5438300.00
Iasi	Pascani	Subtotal LOCALITATE	5438300.00
Iasi		Subtotal JUDET	19088790.00
Neamt	Roman	Client 6 SA	6786570.00
Neamt	Roman	Subtotal LOCALITATE	6786570.00
Neamt		Subtotal JUDET	6786570.00
Timis	Timisoara	Client 5 SRL	1337560.00
Timis	Timisoara	Client 7 SRL	1383375.00
Timis	Timisoara	Subtotal LOCALITATE	2720935.00
Timis		Subtotal JUDET	2720935.00
Vascul	Vascul	Client 3 SRL	7242935.00
Vascul	Vascul	Subtotal LOCALITATE	7242935.00
Vascul		Subtotal JUDET	7242935.00
TOTAL GENERAL			35839230.00

Figura 7.2. Plasarea corectă a subtotalurilor

Pentru acest gen de probleme, SQL-99, precum și versiunile recente ale DB2 și Oracle, pun la dispoziție operatorul ROLLUP, care simplifică mult lucrurile. Să începem cu o interogare lejeră, ce folosește un singur atribut ca argument. Fraza SELECT de mai jos se materializează, în DB2, într-un raport de genul celui din figura 7.3.

```

SELECT DenCl,
       SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari
  FROM PRODUSE P
 INNER JOIN LINIIFACT LF ON P.CodPr=LF.CodPr
 INNER JOIN FACTURI F ON LF.NrFact=F.NrFact
 INNER JOIN CLIENTI C ON F.CodCl=C.CodCl
 INNER JOIN LOCALITATI L ON C.CodPost=L.CodPost
 INNER JOIN JUDETE J ON L.Jud=J.Jud
 GROUP BY ROLLUP (DenCl)
 ORDER BY DenCl

```

DENCL	VINZARI
Client 1 SRL	12490240.00
Client 2 SA	1160250.00
Client 3 SRL	7242935.00
Client 4	5438300.00
Client 5 SRL	1337560.00
Client 6 SA	6786570.00
Client 7 SRL	1383375.00
	35839230.00

Figura 7.3. Primul ROLLUP (mai simplu nu se poate)

Singurul argument va determina calculul unui total general. Pentru a răspunde punctual la problema formulată la începutul paragrafului, se includ, ca argumente ale ROLLUP, cele trei atribute, Judet, Loc și DenCl.

```

SELECT Judet, Loc, DenCl,
       SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari
  FROM PRODUSE P
 INNER JOIN LINIIFACT LF ON P.CodPr=LF.CodPr
 INNER JOIN FACTURI F ON LF.NrFact=F.NrFact
 INNER JOIN CLIENTI C ON F.CodCl=C.CodCl
 INNER JOIN LOCALITATI L ON C.CodPost=L.CodPost
 INNER JOIN JUDETE J ON L.Jud=J.Jud
 GROUP BY ROLLUP (Judet, Loc, DenCl)
 ORDER BY Judet, Loc, DenCl

```

În DB2 v7 se obține situația din figura 7.4. Pentru binele soluției, este necesară și clauza ORDER BY. Primul atribut-argument, Judet, determină calculul totalului general (grand total, cum i se spune în popor). Al doilea argument, Loc, atrage după sine insumarea vânzărilor pentru fiecare valoare distinctă a Judet, iar al treilea, DenCl, are ca efect calcularea a către unui subtotal pentru fiecare combinație (Judet, Loc). Pentru ca subtotalurile și totalul general să fie marcate explicit, se poate folosi una dintre funcțiile: VALUE, COALESCE, NVL, aceasta deoarece în rândurile respective, valorile atributelor de grupare sunt NULL. Iată soluția DB2 care condiționează obținerea rezultatului corect de utilizarea clauzei ORDER BY după GROUP BY:

JUDET	LOC	DENCL	VINZARI
Iasi	Iasi	Client 1 SRL	12490240.00
Iasi	Iasi	Client 2 SA	1160250.00
Iasi	Iasi		13650490.00
Iasi	Pascani	Client 4	5438300.00
Iasi	Pascani		5438300.00
Iasi			19088790.00
Neamt	Roman	Client 6 SA	6786570.00
Neamt	Roman		6786570.00
Neamt			6786570.00
Timis	Timisoara	Client 5 SRL	1337560.00
Timis	Timisoara	Client 7 SRL	1383375.00
Timis	Timisoara		2720935.00
Timis			2720935.00
Vascul	Vascul	Client 3 SRL	7242935.00
Vascul	Vascul		7242935.00
Vascul			7242935.00
			35839230.00

Figura 7.4. Subtotaluri multiple prin ROLLUP

```

SELECT Judet, Loc, DenCl,
       SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari
  FROM PRODUSE P
 INNER JOIN LINIIFACT LF ON P.CodPr=LF.Codpr
 INNER JOIN FACTURI F ON LF.NrFact=F.NrFact
 INNER JOIN CLIENTI C ON F.CodCl=C.CodCl
 INNER JOIN LOCALITATI L ON C.CodPost=L.CodPost
 INNER JOIN JUDETE J ON L.Jud=J.Jud
 GROUP BY Judet, Loc, DenCl
 UNION
 SELECT Judet, Loc, ' Subtotal LOCALITATE' AS DenCl,
       SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari
  FROM PRODUSE P
 INNER JOIN LINIIFACT LF ON P.CodPr=LF.Codpr
 INNER JOIN FACTURI F ON LF.NrFact=F.NrFact
 INNER JOIN CLIENTI C ON F.CodCl=C.CodCl
 INNER JOIN LOCALITATI L ON C.CodPost=L.CodPost
 INNER JOIN JUDETE J ON L.Jud=J.Jud
 GROUP BY Judet, Loc
 UNION
 SELECT Judet, ' Subtotal JUDET' AS Loc, '' AS DenCl,
       SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari
  FROM PRODUSE P
 INNER JOIN LINIIFACT LF ON P.CodPr=LF.Codpr
 INNER JOIN FACTURI F ON LF.NrFact=F.NrFact
 INNER JOIN CLIENTI C ON F.CodCl=C.CodCl
 INNER JOIN LOCALITATI L ON C.CodPost=L.CodPost
 INNER JOIN JUDETE J ON L.Jud=J.Jud
 GROUP BY Judet
 UNION
 SELECT ' TOTAL GENERAL' AS Judet, '' AS Loc, '' AS DenCl,
       SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari
  FROM PRODUSE P
 INNER JOIN LINIIFACT LF ON P.CodPr=LF.Codpr
 INNER JOIN FACTURI F ON LF.NrFact=F.NrFact
 INNER JOIN CLIENTI C ON F.CodCl=C.CodCl
 INNER JOIN LOCALITATI L ON C.CodPost=L.CodPost
 INNER JOIN JUDETE J ON L.Jud=J.Jud
 ORDER BY Judet, Loc, DenCl

```

Rezultatul este cel din figura 7.1. Varianta funcționeză aproape identic în VFP, nefiind necesară prezența clauzei ORDER BY. Atât interogarea, cât și ecranul de răspuns sunt specifice DB2, iar pentru trecerea în Oracle trebuie modificată sintaxa pentru joncțiune.

De obicei, în rapoarte, subtotalurile și totalul general sunt afișate după fiecare grup, respectiv pe ultima linie a rezultatului, în timp ce interogarea precedentă plasează subtotalul la început. Plasarea normală (la sfârșitul grupului) necesă utilizarea unui caracter cu un cod ASCII suficient de mare pentru ca linia respectivă să fie ultima din grup la ordonare, după cum am văzut și în paragraful 4.6.

JUDET	LOC	DENCL	VINZARI
TOTAL GENERAL			35839230.00
Iasi	Subtotal JUDET		19088790.00
Iasi	Iasi	Subtotal LOCALITATE	13650490.00
Iasi	Iasi	Client 1 SRL	12490240.00
Iasi	Iasi	Client 2 SA	1160250.00
Iasi	Pascani	Subtotal LOCALITATE	5438300.00
Iasi	Pascani	Client 4	5438300.00
Neamt	Subtotal JUDET		6786570.00
Neamt	Roman	Subtotal LOCALITATE	6786570.00
Neamt	Roman	Client 6 SA	6786570.00
Timis	Subtotal JUDET		2720935.00
Timis	Timisoara	Subtotal LOCALITATE	2720935.00
Timis	Timisoara	Client 5 SRL	1337560.00
Timis	Timisoara	Client 7 SRL	1383375.00
Vaslui	Subtotal JUDET		7242835.00
Vaslui	Vaslui	Subtotal LOCALITATE	7242835.00
Vaslui	Vaslui	Client 3 SRL	7242835.00

Figura 7.1. Subtotaluri la începutul grupurilor

Pentru a mai schimba un pic acela, soluția următoare este scrisă în Oracle 8i2, raportul obținut fiind cel din figura 7.2.

```

SELECT Judet, Loc, DenCl,
       SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari
  FROM PRODUSE P, LINIIFACT LF, FACTURI F,
        CLIENTI C, LOCALITATI L, JUDETE J
 WHERE P.CodPr=LF.Codpr AND LF.NrFact=F.NrFact AND
       F.CodCl=C.CodCl AND C.CodPost=L.CodPost AND L.Jud=J.Jud
 GROUP BY Judet, Loc, DenCl
 UNION
 SELECT Judet, Loc, CONCAT (CHR(255), ' Subtotal
LOCALITATE') AS DenCl,
       SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari
  FROM PRODUSE P, LINIIFACT LF, FACTURI F,
        CLIENTI C, LOCALITATI L, JUDETE J
 WHERE P.CodPr=LF.Codpr AND LF.NrFact=F.NrFact AND
       F.CodCl=C.CodCl AND C.CodPost=L.CodPost AND L.Jud=J.Jud
 GROUP BY Judet, Loc
 UNION
 SELECT Judet, CONCAT (CHR(255), ' Subtotal JUDET') AS Loc,
       '' AS DenCl,
       SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari
  FROM PRODUSE P, LINIIFACT LF, FACTURI F,
        CLIENTI C, LOCALITATI L, JUDETE J
 WHERE P.CodPr=LF.Codpr AND LF.NrFact=F.NrFact AND
       F.CodCl=C.CodCl AND C.CodPost=L.CodPost AND L.Jud=J.Jud
 GROUP BY Judet

```

De această dată, și VFP este mai conciliant, lucrurile petrecându-se identic ca în DB2 și Oracle. O soluție mai puțin impresionantă utilizează tot o subconsultare, dar ceva mai puțin corelată:

```
DELETE FROM FACTURI
WHERE NrFact NOT IN
  (SELECT DISTINCT NrFact
   FROM LINIIFACT)
```

Și această soluție este una generală (DB2/Oracle/VFP).

În Oracle, singura deosebire ține de exprimarea joncțiunii interne:

```
UPDATE FACTURI
SET ValTotala =
  SELECT SUM(Cantitate * PretUnit * (1+ProcTVA))
  FROM LINIIFACT LF, PRODUSE P
  WHERE LF.CodPr=P.CodPr AND NrFact = FACTURI.NrFact
)
```

Cu regrete de rigoare, trebuie să constatăm că și această interogare este prea savanță pentru dialectul SQL din VFP. Astfel, toate comenziile de actualizare următoare vor fi tratate cu ostilitate de acest SGBD.

Cât privește atributul Reduceri, instituim următoarea regulă: se acordă o reducere de 5% pentru toate tranșele unei facturi incasate în termen de 15 zile de la data vânzării.

```
UPDATE FACTURI
SET Reduceri =
  SELECT SUM ( CASE
    WHEN DataInc <= DataFact + 15 DAY
      THEN Transa * 0.05 ELSE 0 END)
  FROM INCASFACT INCF
  INNER JOIN INCASARI I
    ON INCF.CodInc=I.CodInc
  INNER JOIN FACTURI F2
    ON INCF.NrFact=F2.NrFact
  WHERE F2.NrFact = FACTURI.NrFact
)
```

Noul conținut al tabeliei FACTURI este cel din figura 6.25.

NRFACT	DATAFACT	CODCL	OBS	VALTOTALA	REDUCERI	PENALIZARI
1111	2000-08-01	1001		5399625	269981	
1112	2000-08-01	1005 Probleme ...		1337560	24385	
1113	2000-08-01	1002		1160250	0	
1114	2000-08-01	1006		6786570		
1115	2000-08-02	1001		1651125		
1116	2000-08-02	1007 Pretul prop...		1383375		
1117	2000-08-03	1001		2320500	116025	
1118	2000-08-04	1001		2052750	102837	
1119	2000-08-07	1003		7242935		
1120	2000-08-07	1001		1086240	36577	
1121	2000-08-07	1004		5438300		
1122	2000-08-07	1005		0		

Figura 6.25. Reduceri de 5% pentru incasări în mai puțin de 15 zile de la facturare

Interogarea Oracle 8i are două diferențe: una legată de joncțiune, iar cealaltă, de modul de redare a expresiei de tip dată calendaristică.

```
UPDATE FACTURI
SET Reduceri =
  SELECT SUM ( CASE
    WHEN DataInc <= DataFact + 15
      THEN Transa * 0.05 ELSE 0 END)
  FROM INCASFACT INCF, INCASARI I, FACTURI F2
```

```
WHERE INCF.CodInc=I.CodInc AND
INCF.NrFact=F2.NrFact AND
F2.NrFact = FACTURI.NrFact
)
```

Complicăm un pic cazul. Acordăm 10% pentru tranșele incasate în mai puțin de 15 zile de la data vânzării, 9% pentru 16 zile și 8% pentru 17 zile. Soluția DB2 este:

```
UPDATE FACTURI
SET Reduceri =
  SELECT SUM ( CASE
    WHEN DataInc <= DataFact + 15 DAY
      THEN Transa * 0.1
    WHEN DataInc <= DataFact + 16 DAY
      THEN Transa * 0.09
    WHEN DataInc <= DataFact + 17 DAY
      THEN Transa * 0.08
    ELSE 0 END)
  FROM INCASFACT INCF
  INNER JOIN INCASARI I
    ON INCF.CodInc=I.CodInc
  INNER JOIN FACTURI F2 ON INCF.NrFact=F2.NrFact
  WHERE F2.NrFact = FACTURI.NrFact
)
```

NRFACT	DATAFACT	CODCL	OBS	VALTOTALA	REDUCERI	PENALIZARI
1111	2000-08-01	1001		5399625	539962	
1112	2000-08-01	1005 Probleme ...		1337560	48770	
1113	2000-08-01	1002		1160250	104422	
1114	2000-08-01	1006		6786570		
1115	2000-08-02	1001		1651125		
1116	2000-08-02	1007 Pretul prop...		1383375		
1117	2000-08-03	1001		2320500	232050	
1118	2000-08-04	1001		2052750	205275	
1119	2000-08-07	1003		7242935		
1120	2000-08-07	1001		1086240	73155	
1121	2000-08-07	1004		5438300		
1122	2000-08-07	1005		0		

Figura 6.26. Reduceri pe tranșe

Nu numai modificările pot fi operate utilizând subconsultări, corelate sau nu, ci și stergerile. Spre exemplu, vrem să stergem din tabela FACTURI liniile care nu au nici un copil în LINIIFACT:

```
DELETE FROM FACTURI
WHERE NOT EXISTS
  (SELECT 1
  FROM LINIIFACT
  WHERE LINIIFACT.NrFact = FACTURI.NrFact)
```

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

Figura 6.24. Ierarhii pentru fiecare angajat

```

SELECT LPAD(' ', 5*(LEVEL - 1), '-') || numepren AS Nume,
       Compart
  FROM PERSONAL2
 CONNECT BY PRIOR Marca = MarcaSef
    ORDER BY MarcaSef;
  
```

Primele zece înregistrări reprezintă structura ierarhică pentru care rădăcina este directorul general; liniile 11-15 ale rezultatului constituie ierarhia a cărei bază este Angajat 2 și.m.d.

6.5. Actualizarea tabelelor prin subconsultări

Pentru o mai bună priză la public, în cele ce urmează apelăm la ceea ce se numește de-normalizarea bazei de date. În practică, redundanța datelor este uneori condamnată cu jumătate de gură sau chiar apreciată de mulți specialiști. Aceasta deoarece un atribut redundant, adăugat unei tabele, poate duce la evitarea joncțiunilor frecvente, mari consumatoare de resurse. Aplecându-ne asupra bazei noastre de date, dacă tot ne apucăm de treabă, în tabela FACTURI adăugăm nă mai puțin de trei atribute:

- ValTotală reprezintă valoarea totală (inclusiv TVA) a facturii;
- Reducerii: într-o țară civilizată, ca și noastră, dacă un client plătește o factură înainte de termenul obișnuit (să zicem zece zile), îl putem acorda o reducere de 5% pentru că ne-am procopisit rapid cu lichidități.
- Penalități: este opusul atributului anterior. Prin contract sau prin lege, dacă un client este mai rezinut în a plăti facturile la timp, i se pot aplica penalități, fără a avea însă certitudinea că le-am putea încasa vreodată.

Practic, prin aceste noi atribute, urmărirea încasării facturilor se schimbă sensibil. Valoarea de încasat dintr-o factură este valoarea totală plus penalități minus reduceri. Asta e veste bună. Veste proastă ține de faptul că atributul ValTotală n-ar trebui să se modifice decât la actualizarea tablei LINIIFACT, nu? Altăminteri, dacă utilizatorul ar modifica, prin UPDATE sau alt mijloc, acest atribut, ar apărea un decalaj supărător între liniile din facturi și valorile acestora. Actualizarea automată a unor atribute calculate este unul din scopurile declarate ale declanșatoarelor (trigger-elor), după cum vom vedea în capitolul 8.

Deocamdată adăugăm cele trei atribute tablei FACTURI:

```

ALTER TABLE FACTURI ADD ValTotala DECIMAL(16) ;
ALTER TABLE FACTURI ADD Reducerii DECIMAL(15) ;
ALTER TABLE FACTURI ADD Penalizari DECIMAL(15) ;
  
```

Acum, dacă tot le-am creat, să le „umplem”. Astfel, pentru calculul valorii totale a unei facturi avem nevoie de o interogare corelată după cum urmează:

```

UPDATE FACTURI
  SET ValTotala =
    (SELECT SUM(Cantitate * PretUnit * (1+ProcTVA))
     FROM LINIIFACT LF INNER JOIN PRODUSE P
       ON LF.CodPr=P.CodPr
      WHERE NrFact = FACTURI.NrFact
    )
  
```

Subconsultarea ia în calcul cantitatea, prețul unitar și procentul TVA pentru fiecare produs vândut. Prin corelare se vor lua în considerare numai liniile din LINIIFACT (și PRODUSE) corespunzătoare facturii curente (linia curentă din FACTURI).

Răspunsul exact la întrebare (figura 6.21) presupune următoarea soluție:

```
SELECT PERSONAL2.* , LEVEL
FROM PERSONAL2
WHERE LEVEL - 1 =
    ( SELECT LEVEL
      FROM PERSONAL2
      WHERE NumePren = 'ANGAJAT 7'
        START WITH NumePren = 'ANGAJAT 7'
        CONNECT BY PRIOR MarcaSef = Marca )
    START WITH NumePren = 'ANGAJAT 7'
    CONNECT BY PRIOR MarcaSef = Marca
```

MARCA	NUMEPREN	DATANAST	COMPART	MARCASEF	SALTARFAR	LEVEL
5	ANGAJAT 5	30-APR-65	FINANCIAR	2	4200000	2

Figura 6.21. Șeful direct al Angajatului 7

Care sunt subordonații direcți ai Angajatului 2?

Subordonații de ordin 1 (direcți), 2, 3..., adică cei din figura 6.22, pot fi extrași prin:

```
SELECT PERSONAL2.* , LEVEL
FROM PERSONAL2
START WITH NumePren = 'ANGAJAT 2'
CONNECT BY MarcaSef = PRIOR Marca
```

MARCA	NUMEPREN	DATANAST	COMPART	MARCASEF	SALTARFAR	LEVEL
2	ANGAJAT 2	11-OCT-77	FINANCIAR	1	4500000	1
4	ANGAJAT 4		FINANCIAR	2	3800000	2
5	ANGAJAT 5	30-APR-65	FINANCIAR	2	4200000	2
6	ANGAJAT 6	09-NOV-65	FINANCIAR	5	3500000	3
7	ANGAJAT 7		FINANCIAR	5	2800000	3

Figura 6.22. Toți subordonații Angajatului 2

Ca să obținem răspunsul punctual la întrebare, apelăm la o subconsultare:

```
SELECT PERSONAL2.* , LEVEL
FROM PERSONAL2
WHERE LEVEL - 1 =
    ( SELECT LEVEL
      FROM PERSONAL2
      WHERE NumePren = 'ANGAJAT 2'
        START WITH NumePren = 'ANGAJAT 2'
        CONNECT BY PRIOR MarcaSef = Marca )
    START WITH NumePren = 'ANGAJAT 2'
    CONNECT BY MarcaSef = PRIOR Marca
```

Care sunt subordonații subordonaților directorului general?

Acestea sunt înregistrările „nepot” ale înregistrării-rădăcină (MarcaSef IS NULL). O soluție bazată tot pe pseudocoloana LEVEL și o subconsultare este:

```
SELECT PERSONAL2.* , LEVEL
FROM PERSONAL2
WHERE LEVEL - 2 =
    ( SELECT LEVEL
      FROM PERSONAL2
      WHERE MarcaSef IS NULL
        START WITH MarcaSef IS NULL
        CONNECT BY PRIOR MarcaSef = Marca )
    START WITH MarcaSef IS NULL
    CONNECT BY MarcaSef = PRIOR Marca
```

Însă dacă ținem seama că nivelul ierarhic al directorului general este 1, al subordonaților săi direcți este 2, iar al subordonaților subordonaților este 3, soluția se simplifică apreciabil:

```
SELECT PERSONAL2.* , LEVEL
FROM PERSONAL2
WHERE LEVEL = 3
START WITH MarcaSef IS NULL
CONNECT BY MarcaSef = PRIOR Marca
```

Să se afișeze structura ierarhică a firmei.

Nu putem să reprezentăm arborele ierarhic cu „verdele în jos”, ci cu rădăcina la stânga, la fel ca în soluția DB2 de acum câteva figuri. Indentarea subordonaților se obține cu ajutorul funcției LPAD și pseudoutilului LEVEL după cum urmează:

```
SELECT LPAD(' ', 5*(LEVEL - 1), '-') || numepren AS Nume,
       Compart
  FROM PERSONAL2
 START WITH MarcaSef IS NULL
 CONNECT BY PRIOR Marca = MarcaSef
```

În lipsa unei condiții formulate în clauza START WITH, se construiește căte o ierarhie pentru fiecare angajat (fiecare angajat va fi, pe rând, rădăcină a unei ierarhii). Astfel, interogarea următoare va genera rezultatul din figura 6.24.

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

Figura 6.23. Structura ierarhică a firmei

Lucrurile pot fi simplificate sensibil utilizând o structură de tip CASE, prin care determinăm nivelul de subordonare, nivel care va determina de câte ori se repetă (funcția DB2 REPEAT) grupul de șase liniuți, astfel încât listarea primei coloane este asemănătoare celei din figura 6.18.

```
SELECT RTRIM ( CHAR ( REPEAT ('-', 6 * ( (
CASE
    WHEN NIVEL4.MarcaSef IS NULL THEN 1
    WHEN NIVEL3.MarcaSef IS NULL THEN 2
    WHEN NIVEL2.MarcaSef IS NULL THEN 3
    WHEN NIVEL1.MarcaSef IS NULL THEN 4
END ) - 1 ) ) ) ||

NIVEL4.NumePren AS Nume, NIVEL4.Compart
FROM PERSONAL2 NIVEL1
RIGHT OUTER JOIN PERSONAL2 NIVEL2
    ON NIVEL1.Marca = NIVEL2.MarcaSef
    AND NIVEL1.MarcaSef IS NULL
RIGHT OUTER JOIN PERSONAL2 NIVEL3
    ON NIVEL2.Marca = NIVEL3.MarcaSef
RIGHT OUTER JOIN PERSONAL2 NIVEL4
    ON NIVEL3.Marca = NIVEL4.MarcaSef
```

În VFP, același rezultat se obține printr-un lanț de IF-uri imediate (IIF-uri):

```
SELECT REPLICATE('-', (IIF(ISNULL(NIVEL4.MarcaSef), 1, ;
    IIF(ISNULL(NIVEL3.MarcaSef), 2, ;
        IIF(ISNULL(NIVEL2.MarcaSef), 3, 4) ) ) - 1) ;
    * 7)+NIVEL4.NumePren AS Nume, NIVEL4.Compart ;
FROM PERSONAL2 NIVEL1 ;
RIGHT OUTER JOIN PERSONAL2 NIVEL2 ;
    ON NIVEL1.Marca = NIVEL2.MarcaSef AND ;
    NIVEL1.MarcaSef IS NULL ;
RIGHT OUTER JOIN PERSONAL2 NIVEL3 ;
    ON NIVEL2.Marca = NIVEL3.MarcaSef ;
RIGHT OUTER JOIN PERSONAL2 NIVEL4 ;
    ON NIVEL3.Marca = NIVEL4.MarcaSef
```

Interogări arborescente în Oracle

Pentru problemele formulate în acest paragraf, până acum au fost formulate numai soluții DB2 și VFP, lăsând să se înțeleagă că formularea variantelor echivalente în Oracle și fi o temă pentru acasă/la birou. De fapt ne-am rezervat pentru paginile care urmează. Dialectul SQL al Oracle are câteva opțiuni speciale pentru parcurgerea structurilor ierarhice, cele mai importante fiind START WITH și CONNECT BY.

Care este nivelul ierarhic al fiecărui salariat?

Soluția următoare conduce la rezultatul din figura 6.19.

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

Construirea structurii ierarhice începe cu înregistrarea (înregistrările) care îndeplinește (îndeplinește) condiția din clauza START WITH. Această înregistrare-părinte va fi legată de înregistrarea sau înregistrările-copil prin condiția Marca = MarcaSef. Clauza PRIOR plasată în stânga condiției semnifică: valoarea atributului Marca din părinte trebuie să fie egală cu valoarea MarcaSef din înregistrările-copil.

MARCA	NUMEPREN	DATANAST	COMPART	MARCASEF	SALTARIAR	LEVEL
1	ANGAJAT 1	01-JUL-62	DIRECTIUNE		600000	1
2	ANGAJAT 2	11-OCT-77	FINANCIAR	1	450000	2
4	ANGAJAT 4		FINANCIAR	2	380000	3
5	ANGAJAT 5	30-APR-85	FINANCIAR	2	420000	3
6	ANGAJAT 6	09-NOV-65	FINANCIAR	5	350000	4
7	ANGAJAT 7		FINANCIAR	5	280000	4
3	ANGAJAT 3	22-AUG-62	MARKETING	1	450000	2
8	ANGAJAT 8	31-DEC-60	MARKETING	3	290000	3
9	ANGAJAT 9	28-FEB-78	MARKETING	3	410000	3
10	ANGAJAT 10	29-JAN-72	RESURSE UMANE	1	550000	2

Figura 6.19. Interrogare ierarhică

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 și.a.m.d. Dacă există clauza ORDER BY, aceasta va determina dispunerea înregistrărilor în rezultat. În lipsa clauzei de ordonare, înregistrările sunt dispuse în funcție de ordinea parcurgerii arborelui, așa cum se observă din figura 6.15.

Un avantaj major al interogărilor ierarhice ține de folosirea pseudocoloanei LEVEL, ce semnifică tocmai nivelul ierarhiei, relativ la înregistrarea/inregistrările „rădăcină” care îndeplinește/îndeplinește 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.

Cum se numește șeful Angajatului 7?

Dață folosim interogarea:

```
SELECT PERSONAL2.* , LEVEL
FROM PERSONAL2
START WITH NumePren = 'ANGAJAT 7'
CONNECT BY PRIOR Marca=MarcaSef = Marca
```

se obține:

MARCA	NUMEPREN	DATANAST	COMPART	MARCASEF	SALTARIAR	LEVEL
7	ANGAJAT 7		FINANCIAR	5	280000	1
5	ANGAJAT 5	30-APR-85	FINANCIAR	2	420000	2
2	ANGAJAT 2	11-OCT-77	FINANCIAR	1	450000	3
1	ANGAJAT 1	01-JUL-62	DIRECTIUNE		600000	4

Figura 6.20. Toți șefii Angajatului 7

MARCA	NUMEPREN	DATANAST	COMPART	MARCASEF	SALTARIAR	NIVEL
1 ANGAJAT 1	1962-07-01	DIRECTIUNE			6000000	1
10 ANGAJAT 10	1972-01-29	RESURSE UMANE		1	3700000	2
2 ANGAJAT 2	1977-10-11	FINANCIAR		1	4500000	2
3 ANGAJAT 3	1962-08-22	MARKETING		1	4500000	2
4 ANGAJAT 4		FINANCIAR		2	3800000	3
5 ANGAJAT 5	1965-04-30	FINANCIAR		2	4200000	3
6 ANGAJAT 6	1965-11-09	FINANCIAR		5	3500000	4
7 ANGAJAT 7		FINANCIAR		5	2800000	4
8 ANGAJAT 8	1960-12-31	MARKETING		3	2900000	3
9 ANGAJAT 9	1976-02-28	MARKETING		3	4100000	3

Figura 6.17. Soluție DB2 pentru afișarea nivelului ierarhic al fiecărui angajat

Fraza SELECT compune ierarhia pornind de la NIVEL1, care reprezintă o instanță a PERSONAL2 cu o singură linie, cea a directorului general, instanță jonctionată extern la dreapta cu NIVEL2, care va furniza subordonării direcții ai înregistrării-rădăcină și.a.m.d.

Transcrisă în dialectul VFP, interogarea se prezintă astfel:

```
SELECT NIVEL4.* ,  
      IIF(ISNULL(NIVEL4.Marcasef), 1, ;  
           IIF(ISNULL(NIVEL3.Marcasef), 2, ;  
                IIF(ISNULL(NIVEL2.Marcasef), 3, 4) ) ) AS Nivel ;  
FROM PERSONAL2 NIVEL1 ;  
      RIGHT OUTER JOIN PERSONAL2 NIVEL2 ;  
          ON NIVEL1.Marcasef = NIVEL2.Marcasef AND ;  
              NIVEL1.Marcasef IS NULL ;  
      RIGHT OUTER JOIN PERSONAL2 NIVEL3 ;  
          ON NIVEL2.Marcasef = NIVEL3.Marcasef ;  
      RIGHT OUTER JOIN PERSONAL2 NIVEL4 ;  
          ON NIVEL3.Marcasef = NIVEL4.Marcasef ;
```

Să se afișeze structura ierarhică a firmei.

Practic, dorim o formă de vizualizare a angajaților care să țină cont de modul lor de subordonare - ca în figura 6.18.

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

Figura 6.18. Vizualizarea ierarhiei

Dintre variantele DB2 operaționale, începem cu una destul de stufoasă, bazată pe expresia tabelă NIVELE.

```
WITH NIVELE AS  
      (SELECT NIVEL4.* ,  
      CASE  
          WHEN NIVEL4.Marcasef IS NULL THEN 1  
          WHEN NIVEL3.Marcasef IS NULL THEN 2  
          WHEN NIVEL2.Marcasef IS NULL THEN 3  
          WHEN NIVEL1.Marcasef IS NULL THEN 4  
      END AS Nivel  
      FROM PERSONAL2 NIVEL1  
      RIGHT OUTER JOIN PERSONAL2 NIVEL2  
          ON NIVEL1.Marcasef = NIVEL2.Marcasef  
          AND NIVEL1.Marcasef IS NULL  
      RIGHT OUTER JOIN PERSONAL2 NIVEL3  
          ON NIVEL2.Marcasef = NIVEL3.Marcasef  
      RIGHT OUTER JOIN PERSONAL2 NIVEL4  
          ON NIVEL3.Marcasef = NIVEL4.Marcasef )  
SELECT NIVELE.NumePren AS Nume, NIVELE.Compart,  
      NIVELE.Marcasef AS Sef1, NIVELE.Marcasef AS Sef2,  
      NIVELE.Marcasef AS Sef3, NIVELE.Marcasef AS Sef4  
      FROM NIVELE  
      WHERE Nivel = 1  
      UNION  
      SELECT '-----'||N2.NumePren AS Nume, N2.Compart,  
          N1.Marcasef AS Sef1, N2.Marcasef AS Sef2, N2.Marcasef AS Sef3,  
          N2.Marcasef AS Sef4  
      FROM NIVELE N2 INNER JOIN NIVELE N1 ON  
          N2.Nivel=2 AND N2.Marcasef=N1.Marcasef AND N1.Nivel=1  
      UNION  
      SELECT '-----'||N3.NumePren AS Nume, N3.Compart,  
          N1.Marcasef AS Sef1, N2.Marcasef AS Sef2, N3.Marcasef AS Sef3,  
          N3.Marcasef AS Sef4  
      FROM NIVELE N3  
      INNER JOIN NIVELE N2  
          ON N3.Nivel=3 AND N3.Marcasef=N2.Marcasef AND N2.Nivel=2  
      INNER JOIN NIVELE N1  
          ON N2.Nivel=2 AND N2.Marcasef=N1.Marcasef AND N1.Nivel=1  
      UNION  
      SELECT '-----'||N4.NumePren AS Nume, N4.Compart,  
          N1.Marcasef AS Sef1,  
          N2.Marcasef AS Sef2, N3.Marcasef AS Sef3, N4.Marcasef AS Sef4  
      FROM NIVELE N4 INNER JOIN NIVELE N3  
          ON N4.Nivel=4 AND N4.Marcasef=N3.Marcasef AND N3.Nivel=3  
      INNER JOIN NIVELE N2  
          ON N3.Nivel=3 AND N3.Marcasef=N2.Marcasef AND N2.Nivel=2  
      INNER JOIN NIVELE N1  
          ON N2.Nivel=2 AND N2.Marcasef=N1.Marcasef AND N1.Nivel=1  
      ORDER BY Sef1, Sef2, Sef3, Sef4
```

```

FROM PERSONAL2 NIVEL1
    INNER JOIN PERSONAL2 NIVEL2
        ON NIVEL1.Marcă = NIVEL2.MarcăSef AND
        NIVEL1.MarcăSef IS NULL
    UNION
    SELECT 'Nivel 3' AS Nivel, NIVEL3.NumePren, NIVEL3.Compart
FROM PERSONAL2 NIVEL1
    INNER JOIN PERSONAL2 NIVEL2
        ON NIVEL1.Marcă = NIVEL2.MarcăSef AND
        NIVEL1.MarcăSef IS NULL
    INNER JOIN PERSONAL2 NIVEL3
        ON NIVEL2.Marcă = NIVEL3.MarcăSef
    UNION
    SELECT 'Nivel 4' AS Nivel, NIVEL4.NumePren, NIVEL4.Compart
FROM PERSONAL2 NIVEL1
    INNER JOIN PERSONAL2 NIVEL2
        ON NIVEL1.Marcă = NIVEL2.MarcăSef AND
        NIVEL1.MarcăSef IS NULL
    INNER JOIN PERSONAL2 NIVEL3
        ON NIVEL2.Marcă = NIVEL3.MarcăSef
    INNER JOIN PERSONAL2 NIVEL4
        ON NIVEL3.Marcă = NIVEL4.MarcăSef
ORDER BY NumePren
Iată și rezultatul.

```

NUMEPREN	COMPART	NIVEL
ANGAJAT 1	DIRECTIUNE	Nivel 1
ANGAJAT 10	RESURSE UMANE	Nivel 2
ANGAJAT 2	FINANCIAR	Nivel 2
ANGAJAT 3	MARKETING	Nivel 2
ANGAJAT 4	FINANCIAR	Nivel 3
ANGAJAT 5	FINANCIAR	Nivel 3
ANGAJAT 6	FINANCIAR	Nivel 4
ANGAJAT 7	FINANCIAR	Nivel 4
ANGAJAT 8	MARKETING	Nivel 3
ANGAJAT 9	MARKETING	Nivel 3

Figura 6.16. Nivelul ierarhic al fiecărui angajat

Pentru Visual FoxPro, diferența principală ține de regimul clauzei ORDER BY. Aceasta trebuie să fie atașată primei fraze SELECT din reuniune. În plus, paradoxal, coloana de ordonare poate fi indicată numai prin număr (în această situație). SELECT-ul care obține rezultatul din figura 6.16 este:

```

SELECT 'Nivel 1' AS Nivel, NumePren, Compart ;
FROM PERSONAL2 ;
WHERE MarcăSef IS NULL ;
ORDER BY 2 ;
UNION ;
SELECT 'Nivel 2' AS Nivel, NIVEL2.NumePren, NIVEL2.Compart ;

```

```

FROM PERSONAL2 NIVEL1 ;
    INNER JOIN PERSONAL2 NIVEL2 ;
        ON NIVEL1.Marcă = NIVEL2.MarcăSef AND
        NIVEL1.MarcăSef IS NULL ;
    UNION ;
    SELECT 'Nivel 3' AS Nivel, NIVEL3.NumePren, NIVEL3.Compart ;
FROM PERSONAL2 NIVEL1 ;
    INNER JOIN PERSONAL2 NIVEL2 ;
        ON NIVEL1.Marcă = NIVEL2.MarcăSef AND
        NIVEL1.MarcăSef IS NULL ;
    INNER JOIN PERSONAL2 NIVEL3 ;
        ON NIVEL2.Marcă = NIVEL3.MarcăSef ;
    UNION ;
    SELECT 'Nivel 4' AS Nivel, NIVEL4.NumePren, NIVEL4.Compart ;
FROM PERSONAL2 NIVEL1 ;
    INNER JOIN PERSONAL2 NIVEL2 ;
        ON NIVEL1.Marcă = NIVEL2.MarcăSef AND
        NIVEL1.MarcăSef IS NULL ;
    INNER JOIN PERSONAL2 NIVEL3 ;
        ON NIVEL2.Marcă = NIVEL3.MarcăSef ;
    INNER JOIN PERSONAL2 NIVEL4 ;
        ON NIVEL3.Marcă = NIVEL4.MarcăSef

```

Interogarea funcționează rezonabil. Există însă cel puțin două umbre: trebuie să știm, aprioric, numărul nivelurilor ierarhice, iar în al doilea rând, la un număr mult mai mare de niveluri, intinderea consultării crește sensibil.

O variantă mai elegantă, ca volum de scris, nu și ca resurse consumate, este:

```

SELECT NIVEL4.*,
CASE
    WHEN NIVEL4.MarcăSef IS NULL THEN 1
    WHEN NIVEL3.MarcăSef IS NULL THEN 2
    WHEN NIVEL2.MarcăSef IS NULL THEN 3
    WHEN NIVEL1.MarcăSef IS NULL THEN 4
END AS Nivel
FROM PERSONAL2 NIVEL1
    RIGHT OUTER JOIN PERSONAL2 NIVEL2
        ON NIVEL1.Marcă = NIVEL2.MarcăSef AND
        NIVEL1.MarcăSef IS NULL
    RIGHT OUTER JOIN PERSONAL2 NIVEL3
        ON NIVEL2.Marcă = NIVEL3.MarcăSef
    RIGHT OUTER JOIN PERSONAL2 NIVEL4
        ON NIVEL3.Marcă = NIVEL4.MarcăSef

```

Se jonctionează extern patru instanțe ale tablei PERSONAL2, cîte una pentru fiecare nivel ierarhic, astfel încât au și fost denumite NIVEL1... NIVEL4.

respectiv:

```
SELECT NumePren
FROM PERSONAL2
WHERE MarcaSef IN
    (SELECT Marca
     FROM PERSONAL2
     WHERE NumePren = 'ANGAJAT 2')
```

Ultimele două soluții sunt valabile în aproape orice SGBD.

Câți subordonati are fiecare angajat al firmei?

Soluția:

```
SELECT SEFI.NumePren, COUNT(*) AS Nr_Subordonati
FROM PERSONAL2 SUBORDONATI INNER JOIN PERSONAL2 SEFI
    ON SUBORDONATI.MarcaSef = SEFI.Marcă
GROUP BY SEFI.NumePren
```

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

```
SELECT SEFI.NumePren,
       COUNT(SUBORDONATI.MarcaSef) AS Nr_Subordonati
  FROM PERSONAL2 SUBORDONATI RIGHT OUTER JOIN PERSONAL2 SEFI
    ON SUBORDONATI.MarcaSef = SEFI.Marcă
 GROUP BY SEFI.NumePren
```

Figura 6.14 conține urmările acestei interogări.

NUMEPREN	NR_SUBORDONATI
ANGAJAT 1	3
ANGAJAT 10	0
ANGAJAT 2	2
ANGAJAT 3	2
ANGAJAT 4	0
ANGAJAT 5	2
ANGAJAT 6	0
ANGAJAT 7	0
ANGAJAT 8	0
ANGAJAT 9	0

Figura 6.14. Numărul subordonaților pentru fiecare salariat

Care sunt subordonații subordonaților directorului general?

Din punctul de vedere al arborului ce oglindește ierarhia firmei (figura 6.15), ne interesează „nepoții rădăcinii”. Interrogarea trebuie să parcurgă trei niveluri ierarhice și, în final, să extragă salariații cu mările 4, 5, 8 și 9.

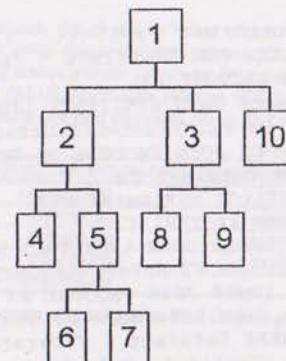


Figura 6.15. Ierarhia firmei

- Varianta I (generală):

```
SELECT SUBORDONATI.Marcă,
       SUBORDONATI.NumePren, SUBORDONATI.Compart
  FROM PERSONAL2 SUBORDONATI INNER JOIN PERSONAL2 SEFI1
    ON SUBORDONATI.MarcaSef = SEFI1.Marcă
    INNER JOIN PERSONAL2 SEFI2
      ON SEFI1.MarcăSef = SEFI2.Marcă
 WHERE SEFI2.MarcăSef IS NULL
```

- Varianta non-VFP (două niveluri de subconsultare):

```
SELECT Marca, NumePren, Compart
  FROM PERSONAL2
 WHERE MarcaSef IN
    (SELECT Marca
     FROM PERSONAL2
     WHERE MarcaSef IN
        (SELECT Marca
         FROM PERSONAL2
         WHERE MarcaSef IS NULL)
    )
```

Care este nivelul ierarhic al fiecărui salariat?

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

```
SELECT 'Nivel 1' AS Nivel, NumePren, Compart
  FROM PERSONAL2
 WHERE MarcaSef IS NULL
 UNION
SELECT 'Nivel 2' AS Nivel, NIVEL2.NumePren, NIVEL2.Compart
```

```

SUM(Cantitate * PretUnit * (1+ProcTVA)) - VALUE(
    (SELECT SUM(Cantitate * PretUnit * (1+ProcTVA))
     FROM FACTURI F2
      INNER JOIN LINIIFACT LF2
        ON F2.NrFact=LF2.NrFact
      INNER JOIN PRODUSE P2 ON LF2.CodPr=P2.CodPr
     WHERE DataFact IN
          (SELECT MAX(DataFact)
           FROM FACTURI F3
            INNER JOIN LINIIFACT LF3
              ON F3.NrFact=LF3.NrFact
            INNER JOIN PRODUSE P3
              ON LF3.CodPr=P3.CodPr
             WHERE DataFact < F.DataFact AND
                  (Cantitate * PretUnit * (1+ProcTVA)) > 0)
        ),0) AS Diferenta

FROM FACTURI F
  INNER JOIN LINIIFACT LF ON F.NrFact=LF.NrFact
  INNER JOIN PRODUSE P ON LF.CodPr=P.CodPr
 GROUP BY DataFact

```

Artificiul necesar extragerii precedentei zi de vânzări se găsește în subconsultările din cele două interogări scalare. Prin joncțiunea instanțelor F3 și LF3 ale tabelelor FACTURI și LINIIFACT și condiția DataFact < F.DataFact, vor fi extrase toate liniile din faturi întocmite înaintea datei curente (F este instanța principală a tabeliei FACTURI – cea care indică ziua curentă). Ca măsură suplimentară de precuție se verifică dacă Cantitate * PretUnit * (1+ProcTVA)) > 0 (ca nu cumva să fie vreo zi în care apare o factură în care liniile să prezinte Cantitate = 0, deși situația aceasta este greu de imaginat). Dintre zilele de vânzare ce precedă ziua curentă, cea mai apropiată (calendaristic) se extrage prin funcția MAX.

Nu știu ce ziceți dumneavoastră, dar mie-mi place...

6.4. Interrogări ierarhice

Un titlu mai adekvat al acestui paragraf ar fi fost *Ierarhii în SQL* sau, și mai general, *Structuri arborescente în SQL*. Cred că, dintre cei care au scris despre SQL, lider al subiectului este de departe Joe Celko, ce publică periodic un material pe această temă, fie în cărțile sale, fie în reviste precum *DBMS Magazine*, actualmente *Intelligent Enterprise*.

Preambulul discuției este plasat în paragraful 5.1, unde au fost prezentate două tabele, PERSONAL2 și SPORURI. Prima dintre acestea reflectă structura ierarhică a firmei, deoarece, pentru fiecare angajat, este indicată și marca șefului său direct (figura 5.2).

Celălalte chestiuni care sunt descrise în continuare privesc modul în care pot fi făcute comparații, analize între angajați aflați pe anumite niveluri ierarhice, între angajați și șefii lor direcți (sau indirecți).

În tabela PERSONAL2 angajatul cu marca 1 este chiar directorul general, pentru acesta valoarea atributului MarcaSef fiind NULL.

Soluția clasică – autojoncțiunea

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

Prin interogarea DB2 (valabilă și în VFP):

```

SELECT *
FROM PERSONAL2 P1 INNER JOIN PERSONAL2 P2
ON P1.Marcasef=P2.Marcă

```

se obține un rezultat de forma celui din figura 6.13.

MARCA	NUMEPREN	DATANASTI	COMPARTIT	MARCASEF	BALTARIAR	MARCA1	NUMEPREN1	DATANASTI1	COMPARTIT1	MARCASEF1	BALTARIAR1
1 ANGAJAT 2	1977-10-11	FINANCIAR	1	4500000	1 ANGAJAT 1	1982-07-01	1 ANGAJAT 1	1982-07-01	1 ANGAJAT 1	1	8000000
1 ANGAJAT 3	1985-08-22	MARKETING	1	4500000	1 ANGAJAT 1	1982-07-01	1 ANGAJAT 1	1982-07-01	1 ANGAJAT 1	1	8000000
10 ANGAJAT 10	1972-01-28	RESURSEUMANE	1	3700000	1 ANGAJAT 1	1982-07-01	1 ANGAJAT 1	1982-07-01	1 ANGAJAT 1	1	8000000
4 ANGAJAT 4		FINANCIAR	2	3900000	2 ANGAJAT 2	1977-10-11	2 ANGAJAT 2	1977-10-11	2 ANGAJAT 2	1	4500000
5 ANGAJAT 5	1985-04-30	FINANCIAR	2	4200000	2 ANGAJAT 2	1977-10-11	2 ANGAJAT 2	1977-10-11	2 ANGAJAT 2	1	4500000
8 ANGAJAT 8	1980-12-31	MARKETING	2	2900000	3 ANGAJAT 3	1982-08-22	3 ANGAJAT 3	1982-08-22	3 ANGAJAT 3	1	4500000
8 ANGAJAT 9	1978-02-28	MARKETING	3	4100000	3 ANGAJAT 3	1982-08-22	3 ANGAJAT 3	1982-08-22	3 ANGAJAT 3	1	4500000
6 ANGAJAT 6	1985-11-09	FINANCIAR	5	3500000	5 ANGAJAT 5	1985-04-30	5 ANGAJAT 5	1985-04-30	5 ANGAJAT 5	2	4200000
7 ANGAJAT 7		FINANCIAR	9	2600000	5 ANGAJAT 5	1985-04-30	5 ANGAJAT 5	1985-04-30	5 ANGAJAT 5	2	4200000

Figura 6.13. Autojoncțiunea tablei PERSONAL2

Primele șase coloane corespund primei instanțe, P1, în timp ce restul – celei de-a două instanțe, P2. P1 este legată de calitatea de subordonat, iar P2 de cea de șef. De aceea, este mai nimerită denumirea lui P1 ca SUBORDONATI, iar a lui P2, ȘEFII.

Să luăm în discuție câteva probleme.

Cum se numește șeful Angajatului 2?

- Soluție SQL-92/DB2/VFP:

```

SELECT SEFI.NumePren
FROM PERSONAL2 SUBORDONATI INNER JOIN PERSONAL2 SEFI
  ON SUBORDONATI.Marcasef = SEFI.Marcă
WHERE SUBORDONATI.NumePren = 'ANGAJAT 2'

```

Care sunt subordonații direcți ai Angajatului 2?

- Soluție SQL-92/DB2/VFP:

```

SELECT SUBORDONATI.NumePren
FROM PERSONAL2 SUBORDONATI INNER JOIN PERSONAL2 SEFI
  ON SUBORDONATI.Marcasef = SEFI.Marcă
WHERE SEFI.NumePren = 'ANGAJAT 2'

```

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

```

SELECT NumePren
FROM PERSONAL2
WHERE Marca IN
  (SELECT MarcaSef
   FROM PERSONAL2
   WHERE NumePren = 'ANGAJAT 2')

```

```

SELECT
    DenPr AS Produs,
    SUM(Cantitate * PretUnit * (1+ProcTVA))
        AS Vinzari_Produs,
    (SELECT SUM(Cantitate * PretUnit * (1+ProcTVA))
    FROM LINIIFACT LF, PRODUSE P
    WHERE LF.CodPr=P.CodPr) AS Total_Vinzari,
    (SUM(Cantitate * PretUnit * (1+ProcTVA)) * 100) /
    (SELECT SUM(Cantitate * PretUnit * (1+ProcTVA))
    FROM LINIIFACT LF, PRODUSE P
    WHERE LF.CodPr=P.CodPr)
        AS Procent
FROM LINIIFACT LF, PRODUSE P
WHERE LF.CodPr=P.CodPr
GROUP BY DenPr

```

PRODUS	VINZARI_PRODUS	TOTAL_VINZARI	PROCENT
Produs 1	3385550.00	35839230.00	9
Produs 2	11356170.00	35839230.00	31
Produs 3	690200.00	35839230.00	1
Produs 4	1397060.00	35839230.00	3
Produs 5	18010250.00	35839230.00	53

Figura 6.10. Contribuția fiecărui produs la cifra de afaceri

Care este evoluția zilnică a vânzărilor, raportat la ziua calendaristică anterioară?

```

SELECT DataFact AS Zi,
    SUM(Cantitate * PretUnit * (1+ProcTVA)) AS
        Vinzari_Zi_Curenta,
    (SELECT SUM(Cantitate * PretUnit * (1+ProcTVA))
    FROM FACTURI F2
    INNER JOIN LINIIFACT LF2 ON F2.NrFact=LF2.NrFact
    INNER JOIN PRODUSE P2 ON LF2.CodPr=P2.CodPr
    WHERE DataFact = F.DataFact - 1 DAY )
        AS Vinzari_Zi_Precedenta,
    SUM(Cantitate * PretUnit * (1+ProcTVA)) -
    VALUE((SELECT SUM(Cantitate * PretUnit * (1+ProcTVA))
    FROM FACTURI F2
    INNER JOIN LINIIFACT LF2
    ON F2.NrFact=LF2.NrFact
    INNER JOIN PRODUSE P2
    ON LF2.CodPr=P2.CodPr
    WHERE DataFact = F.DataFact - 1 DAY ),0)
        AS Diferenta
FROM FACTURI F INNER JOIN LINIIFACT LF ON F.NrFact=LF.NrFact
INNER JOIN PRODUSE P ON LF.CodPr=P.CodPr
GROUP BY DataFact

```

Soluția este una mai laborioasă, dar și mai de efect (figura 6.11). Prima interogare scalară furnizează totalul vânzărilor pentru ziua precedentă, ziua curentă fiind cea a grupului de referință. A doua instanță a interogării scalare permite calcularea diferenței dintre vânzările zilnic curente și a celei precedente. Funcția VALUE convertește valoarea NULL, datorată inexistenței vânzărilor în data calendaristică precedentă (așa cum este cazul zilelor de 1 și 7 august 2000), în zero.

ZI	VINZARI_ZI_CURENTA	VINZARI_ZI_PRECEDENTA	DIFERENTA
2000-08-01	14684005.00		14684005.00
2000-08-02	3034500.00	14684005.00	-11649505.00
2000-08-03	2320500.00	3034500.00	-714000.00
2000-08-04	2052750.00	2320500.00	-267750.00
2000-08-07	13747475.00		13747475.00

Figura 6.11. Diferența vânzărilor între ziua curentă și cea precedentă

Care este evoluția zilnică a vânzărilor, raportat la ziua de vânzări anterioară?

În problema de mai sus, diferența era calculată între ziua curentă și ziua precedentă. Astfel încât toate zilele de luni erau raportate la zero, deoarece, pentru cea mai mare parte a firmelor, duminica nu se lucrează. Acum dorim ca diferența să fie calculată între vânzările din ziua curentă și cele din *precedenta zi de vânzări*, adică între luni și vineri și-a.m.d., după cum se observă în figura 6.12.

ZI	VINZARI_ZI_CURENTA	VINZARI_ZI_PRECEDENTA	DIFERENTA
2000-08-01	14684005.00		14684005.00
2000-08-02	3034500.00	14684005.00	-11649505.00
2000-08-03	2320500.00	3034500.00	-714000.00
2000-08-04	2052750.00	2320500.00	-267750.00
2000-08-07	13747475.00	2052750.00	11694725.00

Figura 6.12. Diferențele dintre două zile consecutive de vânzări

```

SELECT DataFact AS Zi,SUM(Cantitate * PretUnit * (1+ProcTVA))
    AS Vinzari_Zi_Curenta,
    (SELECT SUM(Cantitate * PretUnit * (1+ProcTVA))
    FROM FACTURI F2
    INNER JOIN LINIIFACT LF2
    ON F2.NrFact=LF2.NrFact
    INNER JOIN PRODUSE P2
    ON LF2.CodPr=P2.CodPr
    WHERE DataFact IN
        (SELECT MAX(DataFact)
        FROM FACTURI F3
        INNER JOIN LINIIFACT LF3
        ON F3.NrFact=LF3.NrFact
        INNER JOIN PRODUSE P3
        ON LF3.CodPr=P3.CodPr
        WHERE DataFact < F.DataFact AND
            (Cantitate * PretUnit * (1+ProcTVA)) > 0)
        ) AS Vinzari_Zi_Precedenta,
    FROM FACTURI F INNER JOIN LINIIFACT LF ON F.NrFact=LF.NrFact
    INNER JOIN PRODUSE P ON LF.CodPr=P.CodPr
    GROUP BY DataFact

```

A doua coloană a rezultatului este obținută printr-o interogare scalară care operează oarecum independent de fraza SELECT principală, furnizându-i însă o valoare.

Care sunt totalurile vânzărilor și incasărilor?

Formulăm o variantă curioasă:

```
SELECT
    (SELECT SUM(Cantitate * PretUnit * (1+ProcTVA))
     AS Vinzari
      FROM LINIIFACT LF, PRODUSE P
     WHERE LF.CodPr=P.CodPr) AS Facturat,
    (SELECT SUM(Transa)
      FROM INCASFACT) AS Incasat
   FROM SYSIBM.SYSDUMMY1
```

Fraza SELECT conține două subconsultări scalare: una care extrage totalul vânzărilor, celalată, totalul incasărilor. Deoarece în clauza FROM era nevoie de o tabelă cu o singură linie, s-a folosit pseudotabela SYSIBM.SYSDUMMY1 (echivalentă [pseudo]tabelei DUAL din Oracle) care are un rând și un atribut.

Care este ziua în care s-au emis cele mai multe facturi?

```
SELECT DataFact, COUNT(*) AS Nr_Facturi,
       (SELECT MAX(NrF)
        FROM
           (SELECT COUNT(*) AS NrF
            FROM FACTURI
            GROUP BY DataFact) T1 ) AS NrMax
     FROM FACTURI
    GROUP BY DataFact
   HAVING COUNT(*) >=
      (SELECT MAX(NrF)
       FROM
          (SELECT COUNT(*) AS NrF
           FROM FACTURI
           GROUP BY DataFact) T2 )
```

Valorile coloanelor Nr_Facturi și NrMax ale rezultatului din figura 6.7 sunt furnizate de două subconsultări scalare, una care calculează numărul zilnic al facturilor, iar a doua, numărul maxim de facturi emise într-o zi.

DATAFACT	NR_FACTURI	NRMAX
2000-08-01	4	4
2000-08-07	4	4

Figura 6.7. Zilele în care s-au întocmit cele mai multe facturi

Care sunt localitățile în care se află sediul fiecărui client?

Revenim la un exemplu banal, rezolvat atât de simplu prin juncțiune sau subconsultare, pentru a demonstra că de mult ne putem complica viața în SQL (desi au fost exemple mai convingătoare). Ei bine, această problemă supărător de simplă poate fi rezolvată și prin subconsultări scalare:

```
SELECT DenCl,
       (SELECT Loc
        FROM LOCALITATI
       WHERE CodPost=CLIENTI.CodPost)
     FROM CLIENTI
    ORDER BY Loc
```

Figura 6.8 este edificatoare în ceea ce privește corectitudinea rezultatului interogării. Unicul merit al exemplului este, probabil, acela de a demonstra că o subconsultare scalară poate fi corelată cu tabela din clauza FROM a frazei principale.

DENCL	LOC
Client1 SRL	Iasi
Client 2 SA	Iasi
Client 4	Păscani
Client 6 SA	Roman
Client 5 SRL	Timisoara
Client 7 SRL	Timisoara
Client 3 SRL	Vaslui

Figura 6.8. Subconsultare scalară corelată cu tabela din fraza principală

Continuăm cu alte exemple în care subconsultările scalare își arată pe deplin eficiența. Abia o dată cu apariția funcțiilor OLAP în SQL-99 și versiunile recente ale Oracle (8i2), DB2 (6.1 și 7) pot fi formulate soluții bazate pe funcții și clauze preferabile subconsultărilor scalare.

Pentru că la săptămâna clienti s-au întocmit, zilnic, facturi?

```
SELECT DataFact, COUNT(DISTINCT CodCl) AS Nr_Clienti,
       (SELECT COUNT(*) FROM CLIENTI) AS Nr_Total_Clienti,
       (COUNT(DISTINCT CodCl) * 100) /
      (SELECT COUNT(*) FROM CLIENTI) AS Procent
     FROM FACTURI
    GROUP BY DataFact
```

DATAFACT	NR_CLIENTI	NR_TOTAL_CLIENTI	PROCENT
2000-08-01	4	7	57
2000-08-02	2	7	28
2000-08-03	1	7	14
2000-08-04	1	7	14
2000-08-07	4	7	57

Figura 6.9. Procentajul zilnic al clientilor pentru care există facturi

Pentru a obține procentul care interesează, se imparte rezultatul calculat de funcția COUNT pentru fiecare grup (zi calendaristică) la valoarea extrasă de interogarea scalară.

Care este contribuția (procentuală) a fiecărui produs la totalul vânzărilor?

Informația este una esențială pentru orice firmă. Ne vom servi de o subconsultare scalară pentru a determina, pe fiecare linie (corespunzătoare unui produs), totalul vânzărilor și a face astfel raportul care ne interesează.

```

(SELECT CodCl, SUM(Transa) AS Incasari
FROM FACTURI F, INCASFACT I
WHERE I.NrFact=F.NrFact
GROUP BY CodCl) INCASAT
ON FACTURAT.CodCl=INCASAT.CodCl
) TEMP1
INNER JOIN
(SELECT MAX (Vinzari - VALUE(Incasari, 0)) AS DifMax
FROM
(SELECT CodCl, SUM(Cantitate * PretUnit *
(1+ProcTVA)) AS Vinzari
FROM FACTURI F, LINIIFACT LF, PRODUSE P
WHERE F.NrFact=LF.NrFact AND LF.CodPr=P.CodPr
GROUP BY CodCl) FACTURAT
LEFT OUTER JOIN
(SELECT CodCl, SUM(Transa) AS Incasari
FROM FACTURI F, INCASFACT I
WHERE I.NrFact=F.NrFact
GROUP BY CodCl) INCASAT
ON FACTURAT.CodCl=INCASAT.CodCl) TEMP2
ON TEMP1.DeIncasat=TEMP2.DifMax
INNER JOIN CLIENTI ON TEMP1.CodCl=CLIENTI.CodCl
• și Oracle:
SELECT DenCl, Vinzari, Incasari, DeIncasat
FROM
CLIENTI,
(SELECT FACTURAT.CodCl, Vinzari,
NVL(Incasari, 0) AS Incasari,
Vinzari - NVL(Incasari, 0) AS DeIncasat
FROM
(SELECT CodCl, SUM(Cantitate * PretUnit *
(1+ProcTVA)) AS Vinzari
FROM FACTURI F, LINIIFACT LF, PRODUSE P
WHERE F.NrFact=LF.NrFact AND LF.CodPr=P.CodPr
GROUP BY CodCl) FACTURAT,
(SELECT CodCl, SUM(Transa) AS Incasari
FROM FACTURI F, INCASFACT I
WHERE I.NrFact=F.NrFact
GROUP BY CodCl) INCASAT
WHERE FACTURAT.CodCl=INCASAT.CodCl (+) ) TEMP1,
(SELECT MAX (Vinzari - NVL(Incasari, 0)) AS DifMax
FROM
(SELECT CodCl, SUM(Cantitate * PretUnit *
(1+ProcTVA)) AS Vinzari

```

```

FROM FACTURI F, LINIIFACT LF, PRODUSE P
WHERE F.NrFact=LF.NrFact AND LF.CodPr=P.CodPr
GROUP BY CodCl) FACTURAT,
(SELECT CodCl, SUM(Transa) AS Incasari
FROM FACTURI F, INCASFACT I
WHERE I.NrFact=F.NrFact
GROUP BY CodCl) INCASAT
WHERE FACTURAT.CodCl=INCASAT.CodCl (+) ) TEMP2
WHERE TEMP1.DeIncasat=TEMP2.DifMax AND
TEMP1.CodCl=CLIENTI.CodCl

```

6.3. Subconsultări scalare în clauza SELECT

Standardul SQL-92 face trimitere și la interogările scalare ce pot fi definite ca fraze SELECT ce obțin un rezultat alcătuit dintr-o singură linie și o singură coloană. Utilitatea acestora este vizibilă în expresii complexe.

Deși este prima oară când vomenim de interogări scalare, le-am folosit de multe ori până acum în clauzele WHERE și HAVING. Ceea ce vom parcurge în continuare se referă la includerea unei interogări scalare în clauza SELECT a unei interogări.

Din păcate, nici Oracle 8, nici VFP nu au implementată această facilitate, aşa încât toate exemplele care urmează sunt specifice DB2.

Care sunt totalurile salariilor tarifare și ale sporurilor pe luna iulie 2000 pentru întreaga firmă?

Soluția clasică este:

```

SELECT SUM(SalTarifar) AS Total_Sal_Tarifar,
SUM (
    VALUE(SporVechime,0) + VALUE(SporNoapte,0) +
    VALUE(SporCD,0)+VALUE(AlteSpor,0)
) AS Total_Sporuri_Iulie
FROM PERSONAL2
INNER JOIN SPORURI ON PERSONAL2.Marcă=SPORURI.Marcă
WHERE An=2000 AND Luna=7

```

Intrucât toate persoanele din tabela PERSONAL au lucrat în luna iulie 2000 (nu a plecat nici un angajat din organizație), se poate formula și interogarea:

```

SELECT SUM(SalTarifar) Total_Sal_Tarifar,
(SELECT SUM ( VALUE(SporVechime,0) +
    VALUE(SporNoapte,0)+ VALUE(SporCD,0) +
    VALUE(AlteSpor,0))
) AS Total_Sporuri_Iulie
FROM SPORURI
WHERE An=2000 AND Luna=7
FROM PERSONAL2

```

Care este județul cu vânzări imediat superioare județului Neamț?

Soluția precedentă este rezonabilă, dar nu răspunde punctual la întrebare. Deși care, la prezentul exemplu, ne punem problema afișării numai a județului și valorii vânzărilor. Vom apela la un truc ieftin, pe care l-am mai utilizat. Iată interogarea DB2 și rezultatul său în figura 6.6.

```

SELECT MIN (CAST (CAST (VINZ_JUD1.Vinzari
    AS DECIMAL (15,0) AS CHAR(16)) CONCAT
    ' - ' CONCAT VINZ_JUD1.Judet)
AS Rezultat
FROM
    (SELECT Judet, SUM(Cantitate * PretUnit * (1+ProcTVA))
        AS Vinzari
    FROM JUDETE J, LOCALITATI L, CLIENTI C, FACTURI F,
        LINIIFACT LF, PRODUSE P
    WHERE J.Jud=L.Jud AND L.CodPost=C.CodPost AND
        C.CodCl=F.CodCl AND F.NrFact=LF.NrFact
        AND LF.CodPr=P.CodPr
        GROUP BY Judet) VINZ_JUD1,
    (SELECT Judet, SUM(Cantitate * PretUnit * (1+ProcTVA))
        AS Vinzari
    FROM JUDETE J, LOCALITATI L, CLIENTI C, FACTURI F,
        LINIIFACT LF, PRODUSE P
    WHERE J.Jud=L.Jud AND L.CodPost=C.CodPost AND
        C.CodCl=F.CodCl AND F.NrFact=LF.NrFact
        AND LF.CodPr=P.CodPr
        GROUP BY Judet) VINZ_JUD2
    WHERE VINZ_JUD1.Vinzari > VINZ_JUD2.Vinzari
    AND VINZ_JUD2.Judet='Neamt'

```

REZULTAT
00000007242935. - Vaslui

Figura 6.6. Județul cu vânzări imediat superioare județului Neamț

Tabelele ad-hoc VINZ_JUD1 și VINZ_JUD2 sunt identice și conțin valoarea vânzărilor pentru fiecare județ. Fraza SELECT principală le theta-joncționează, după expresia VINZ_JUD1.Vinzari > VINZ_JUD2.Vinzari, în condițiile în care această comparație se face pentru linia în care VINZ_JUD2.Judet='Neamt'. Rezultă că vor fi extrase acele județe (și vânzările corespunzătoare) care au valoarea vânzărilor peste cea a județului Neamț. Trucul ieftin de care vorbeam este plasat în clauza SELECT a interogării principale. Ideea este de a concatena valoarea vânzărilor cu denumirea județului. Cum minimul este decis de primul argument, căreala funcționează!

O soluție fără improvizații este următoarea:

```

SELECT Judet, SUM(Cantitate * PretUnit * (1+ProcTVA))
    AS Vinzari
FROM JUDETE J, LOCALITATI L, CLIENTI C, FACTURI F,
    LINIIFACT LF, PRODUSE P

```

```

WHERE J.Jud=L.Jud AND L.CodPost=C.CodPost
    AND C.CodCl=F.CodCl
    AND F.NrFact=LF.NrFact AND LF.CodPr=P.CodPr
GROUP BY Judet
HAVING SUM(Cantitate * PretUnit * (1+ProcTVA)) =
    (SELECT MIN (VINZ_JUD1.Vinzari)
    FROM
        (SELECT Judet, SUM(Cantitate * PretUnit * (1+ProcTVA))
            AS Vinzari
        FROM JUDETE J, LOCALITATI L, CLIENTI C,
            FACTURI F, LINIIFACT LF, PRODUSE P
        WHERE J.Jud=L.Jud AND L.CodPost=C.CodPost AND
            C.CodCl=F.CodCl AND F.NrFact=LF.NrFact
            AND LF.CodPr=P.CodPr
            GROUP BY Judet) VINZ_JUD1,
        (SELECT Judet, SUM(Cantitate * PretUnit * (1+ProcTVA))
            AS Vinzari
        FROM JUDETE J, LOCALITATI L, CLIENTI C, FACTURI F,
            LINIIFACT LF, PRODUSE P
        WHERE J.Jud=L.Jud AND L.CodPost=C.CodPost AND
            C.CodCl=F.CodCl AND F.NrFact=LF.NrFact
            AND LF.CodPr=P.CodPr
            GROUP BY Judet) VINZ_JUD2
        WHERE VINZ_JUD1.Vinzari > VINZ_JUD2.Vinzari
        AND VINZ_JUD2.Judet='Neamt')

```

Prima variantă furnizează un răspuns incomplet dacă sunt la egalitate două sau mai multe județe care îndeplinesc condiția.

Care este clientul cel mai datornic (care are cel mai mare rest de plată)?

Fondul problemei este asemănător celei precedente. În interogare obținem o tabelă ad-hoc cu diferența de incasat pe clienți (TEMP1) și o altă ce conține cea mai mare diferență de incasat pentru un client (TEMP2). Cele două tabele sunt jonctionate după diferență și, în final, pentru a afla denumirea clientului, adăugăm în joncționare tabela CLIENTI.

- Soluția DB2:

```

SELECT Denc1, Vinzari, Incasari, DeIncasat
FROM
    (SELECT FACTURAT.CodCl, Vinzari,
        VALUE(Incasari, 0) AS Incasari,
        Vinzari - VALUE(Incasari, 0) AS DeIncasat
    FROM
        (SELECT CodCl, SUM(Cantitate * PretUnit *
            (1+ProcTVA)) AS Vinzari
        FROM FACTURI F, LINIIFACT LF, PRODUSE P
        WHERE F.NrFact=LF.NrFact AND LF.CodPr=P.CodPr
        GROUP BY CodCl) FACTURAT
        LEFT OUTER JOIN

```

Care sunt clienții cu valoarea vânzărilor peste medie?

• Soluția 1:

```
SELECT DenCl, SUM(Cantitate * PretUnit * (1+ProcTVA))
      AS Vinzari
  FROM CLIENTI C, FACTURI F, LINIIFACT LF, PRODUSE P
 WHERE C.CodCl=F.CodCl AND F.NrFact=LF.NrFact
   AND LF.CodPr=P.CodPr
 GROUP BY DenCl
 HAVING SUM(Cantitate * PretUnit * (1+ProcTVA)) >=
 (SELECT Vinzari / NrClienti
  FROM
    (SELECT SUM(Cantitate * PretUnit * (1+ProcTVA))
         AS Vinzari
      FROM LINIIFACT LF, PRODUSE P
     WHERE LF.CodPr=P.CodPr
    ) TEMP1,
    (SELECT COUNT(DISTINCT CodCl) AS NrClienti
      FROM FACTURI)
  TEMP2
 )
```

• Soluția 2:

```
SELECT DenCl, VINZ_CL.Vinzari
FROM
  (SELECT DenCl, SUM(Cantitate * PretUnit * (1+ProcTVA))
       AS Vinzari
    FROM CLIENTI C, FACTURI F, LINIIFACT LF, PRODUSE P
   WHERE C.CodCl=F.CodCl AND F.NrFact=LF.NrFact
     AND LF.CodPr=P.CodPr
 GROUP BY DenCl
  ) VINZ_CL,
  (SELECT DISTINCT Vinzari / NrClienti AS Medie_Vinz
   FROM
    (SELECT SUM(Cantitate * PretUnit * (1+ProcTVA))
         AS Vinzari
      FROM LINIIFACT LF, PRODUSE P
     WHERE LF.CodPr=P.CodPr,
      (SELECT COUNT(DISTINCT CodCl) AS NrClienti
        FROM FACTURI)
     ) MEDIE_VINZ
   WHERE VINZ_CL.Vinzari >= MEDIE_VINZ.Medie_Vinz
```

Care este factura cu cea mai mică valoare peste cea medie?

```
SELECT NrFact, SUM(Cantitate * PretUnit * (1+ProcTVA))
      AS ValFact
  FROM LINIIFACT LF, PRODUSE P
 WHERE LF.CodPr=P.CodPr
 GROUP BY NrFact
```

```
HAVING SUM(Cantitate * PretUnit * (1+ProcTVA)) =
 (SELECT MIN(ValFact)
  FROM
    (SELECT SUM(Cantitate * PretUnit * (1+ProcTVA))
         AS ValFact
      FROM LINIIFACT LF, PRODUSE P
     WHERE LF.CodPr=P.CodPr
 GROUP BY NrFact) TEMP1
 WHERE ValFact >
 (SELECT Vinzari / NrFacturi AS ValMedie
  FROM
    (SELECT SUM(Cantitate * PretUnit *
 (1+ProcTVA)) AS Vinzari,
     COUNT(DISTINCT NrFact) AS NrFacturi
      FROM LINIIFACT LF, PRODUSE P
     WHERE LF.CodPr=P.CodPr ) TEMP1
  )
 )
```

Apelăm și la o soluție care să afișeze toate facturile cu valoarea peste medie, în ordinea crescătoare a acestei valori. Prima factură din rezultat – figura 6.5 – va fi răspunsul la întrebarea formulată.

```
SELECT NrFact, ValFact, ValMedie
FROM
  (SELECT NrFact, SUM(Cantitate * PretUnit * (1+ProcTVA))
       AS ValFact
    FROM LINIIFACT LF, PRODUSE P
   WHERE LF.CodPr=P.CodPr
 GROUP BY NrFact) TEMP1,
  (SELECT ROUND(Vinzari / NrFacturi,0) AS ValMedie
   FROM
    (SELECT SUM(Cantitate * PretUnit * (1+ProcTVA))
         AS Vinzari,
         COUNT(DISTINCT NrFact) AS NrFacturi
        FROM LINIIFACT LF, PRODUSE P
       WHERE LF.CodPr=P.CodPr ) MEDIE1
    ) MEDII
 WHERE ValFact > ValMedie
 ORDER BY ValFact
```

NRFAC	VALFACT	VALMEDIE
1111	5399625	3258112
1121	5438300	3258112
1114	6786570	3258112
1119	7242935	3258112

Figura 6.5. Facturile cu valori peste medie

Care este ziua în care s-au emis cele mai multe facturi?

Din păcate (sau din fericire!), interogarea:

```
SELECT TEMP1.DataFact, TEMP1.Nr
FROM
  (SELECT DataFact, COUNT(NrFact) AS Nr
   FROM FACTURI
   GROUP BY DataFact) TEMP1,
  (SELECT DataFact, COUNT(NrFact) AS Nr
   FROM FACTURI
   GROUP BY DataFact) TEMP2
WHERE TEMP1.Nr >= ALL
  (SELECT Nr FROM TEMP2)
```

nu este operațională. Aceasta înseamnă că o tabelă definită ad-hoc într-o frază SELECT nu este recunoscută în subconsultări. Se poate, totuși, utiliza varianta:

```
SELECT DataFact, COUNT(*) AS Nr_Facturi
FROM FACTURI
GROUP BY DataFact
HAVING COUNT(*) =
  (SELECT MAX(Nr)
   FROM
     (SELECT DataFact, COUNT(NrFact) AS Nr
      FROM FACTURI GROUP BY DataFact) TEMP1)
```

In subconsultare s-a definit tabela intermediară TEMP1, al cărei atribut Nr este folosit în funcția MAX din clauza SELECT. O altă variantă bazată pe subconsultare în FROM este și:

```
SELECT DataFact, NrFacturi
FROM
  (SELECT DataFact, COUNT(*) AS NrFacturi
   FROM FACTURI
   GROUP BY DataFact) FACT_ZILE,
  (SELECT MAX(COUNT(*)) AS NrMax
   FROM FACTURI
   GROUP BY DataFact) MAX
WHERE NrFacturi=NrMax
```

Care este clientul care a cumpărat cele mai multe produse?

După modelul interogării anterioare, formulăm două soluții generale (SQL/DB2/Oracle) bazate pe subconsultări în clauza FROM.

- Soluția 1:

```
SELECT DenCl, COUNT(DISTINCT CodPr) AS Nr_Produse
FROM CLIENTI, FACTURI, LINIIFACT
WHERE CLIENTI.CodCl=FACTURI.CodCl AND
FACTURI.NrFact=LINIIFACT.NrFact
GROUP BY DenCl
```

```
HAVING COUNT(DISTINCT CodPr) =
  (SELECT MAX(Nr)
   FROM
     (SELECT CodCl, COUNT(DISTINCT CodPr) AS Nr
      FROM FACTURI, LINIIFACT
      WHERE FACTURI.NrFact=LINIIFACT.NrFact
      GROUP BY CodCl
    ) TEMP1
  )
```

- Soluția 2:

```
SELECT DenCl, CL_PROD.NrProd
FROM
  (SELECT DenCl, COUNT(DISTINCT CodPr) AS NrProd
   FROM CLIENTI, FACTURI, LINIIFACT
   WHERE CLIENTI.CodCl=FACTURI.CodCl AND
   FACTURI.NrFact=LINIIFACT.NrFact
   GROUP BY DenCl ) CL_PROD,
  (SELECT MAX(COUNT(DISTINCT CodPr)) AS NrProd
   FROM FACTURI, LINIIFACT
   WHERE FACTURI.NrFact=LINIIFACT.NrFact
   GROUP BY CodCl) CL_MAX
WHERE CL_PROD.NrProd=CL_MAX.NrProd
```

Care este județul în care berea s-a vândut cel mai bine?

```
SELECT Judet, SUM(Cantitate * PretUnit * (1+ProcTVA))
  AS Vinzari_Bere
FROM JUDETE J, LOCALITATI L, CLIENTI C, FACTURI F,
LINIIFACT LF, PRODUSE P
WHERE J.Jud=L.Jud AND L.CodPost=C.CodPost AND C.CodCl=F.CodCl
  AND F.NrFact=LF.NrFact AND LF.CodPr=P.CodPr
  AND Grupa='Bere'
GROUP BY Judet
HAVING SUM(Cantitate * PretUnit * (1+ProcTVA)) =
  (SELECT MAX(Vinzari)
   FROM
     (SELECT SUM(Cantitate * PretUnit * (1+ProcTVA))
      AS Vinzari
      FROM LOCALITATI L, CLIENTI C, FACTURI F,
      LINIIFACT LF, PRODUSE P
      WHERE L.CodPost=C.CodPost AND C.CodCl=F.CodCl
        AND F.NrFact=LF.NrFact AND LF.CodPr=P.CodPr
        AND Grupa='Bere'
      GROUP BY Jud
    ) TEMP1
  )
```

Ce facturi conțin măcar produsele din factura 1112?

```
SELECT DISTINCT NrFact
FROM
  (SELECT NrFact, COUNT(*) AS NrProd
  FROM LINIIFACT
  WHERE CodPr IN
    (SELECT CodPr
    FROM LINIIFACT
    WHERE NrFact = 1112)
  GROUP BY NrFact
) T1,
  (SELECT COUNT(CodPr) AS NrP1112
  FROM LINIIFACT
  WHERE NrFact = 1112) T2
WHERE T1.NrProd = T2.NrP1112
```

T2 conține numărul produselor din factura 1112. T1 conține, pentru fiecare factură, câte produse sunt comune acesteia și facturii 1112. Prin joișunarea T1 cu T2 prin condiția T1.NrProd = T2.NrP1112, se extrag acele linii din T1 care au același număr de produse prezente în factura 1112 ca și aceasta.

Care sunt clienții cărora li s-au vândut cel puțin produsele vândute clientului CLIENT 4?

```
SELECT DISTINCT DenCl
FROM
  (
    SELECT DenCl, COUNT(*) AS NrProd
    FROM CLIENTI C, FACTURI F, LINIIFACT LF
    WHERE C.CodCl=F.CodCl AND F.NrFact=LF.NrFact AND
      CodPr IN
        (SELECT CodPr
        FROM CLIENTI C, FACTURI F, LINIIFACT LF
        WHERE C.CodCl=F.CodCl AND F.NrFact=LF.NrFact AND
          DenCl='Client 4')
    GROUP BY DenCl
  ) T1,
  (
    SELECT COUNT(CodPr) AS NrProd
    FROM CLIENTI C, FACTURI F, LINIIFACT LF
    WHERE C.CodCl=F.CodCl AND F.NrFact=LF.NrFact AND
      DenCl='Client 4'
  ) T2
WHERE T1.NrProd = T2.NrProd
```

Logica soluției este că se poate se asemănătoare precedentei, doar că T2 conține numărul produselor vândute clientului 4, iar în T1, pe fiecare linie se găsește un client și numărul produselor vândute clientului 4 care i-au fost vândute și lui.

Să se afișeze căte facturi sunt: neîncasate deloc, încasate parțial și încasate total?

Prezentăm numai soluția DB2.

```
SELECT
  CASE
    WHEN VALUE(Incasa,0) = 0
      THEN 'Fara nici o incasare'
    WHEN Facturat > VALUE(Incasa,0)
      THEN 'Incasata parțial'
    ELSE 'INCASATA TOTAL'
  END AS Situație, COUNT(*) AS Nr
FROM
  (
    SELECT NrFact, SUM(Cantitate * PretUnit * (1+ProcTVA))
    AS Facturat
    FROM LINIIFACT LF
    INNER JOIN PRODUSE P ON LF.CodPr = P.CodPr
    GROUP BY NrFact
  ) VINZARI
LEFT OUTER JOIN
  (
    SELECT NrFact, SUM(Transa) AS Incasat
    FROM INCASFAC
    GROUP BY NrFact
  ) INCASARI
  ON VINZARI.NrFact = INCASARI.NrFact
GROUP BY CASE
  WHEN VALUE(Incasa,0) = 0
    THEN 'Fara nici o incasare'
  WHEN Facturat > VALUE(Incasa,0)
    THEN 'Incasata parțial'
  ELSE 'INCASATA TOTAL'
```

Joișunarea externă la stânga dintre VÂNZĂRI și INCASARI este completată de o structură alternativă multiplă – CASE sau DECODE (in Oracle).

Care angajați au salariul tarifar egal cu cel al ANGAJAT2?

```
SELECT NumePren
FROM
  (SELECT NumePren, SalTarifar
  FROM PERSONAL2) TEMP1,
  (SELECT SalTarifar
  FROM PERSONAL2
  WHERE NumePren='ANGAJAT 2') TEMP2
WHERE TEMP1.SalTarifar = TEMP2.SalTarifar
```

și SQL-92/DB2 plus Oracle:

```
SELECT DISTINCT X
FROM R1
WHERE X NOT IN
  (SELECT DISTINCT PROD_CART.X
   FROM (SELECT DISTINCT R1.X, R2.Y FROM R1, R2) PROD_CART,
        R1
   WHERE PROD_CART.X=R1.X (+) AND PROD_CART.Y=R1.Y (+)
        AND R1.X IS NULL)
```

Putem încerca însă și ceva mai elegant. Ce ziceți de soluția:

```
SELECT DISTINCT X
FROM
  (SELECT X, COUNT(Y) AS Nr FROM R1 GROUP BY X) TEMP1,
  (SELECT COUNT(Y) AS Nr FROM R2) TEMP2
WHERE TEMP1.Nr=TEMP2.Nr
```

Prima tabelă, TEMP1, conține numărul valorilor lui Y pentru fiecare X din R1, iar a doua, TEMP2, numai numărul total al valorilor lui Y din R2 – figura 6.4.

TEMP1		TEMP2	
X	Nr		Nr
x1	5		
x2	2		
x3	5		
x4	2		
x5	2		

Figura 6.4. Tabelele intermediare (ad-hoc) TEMP1 și TEMP2

Care sunt clienții pentru care există cel puțin câte o factură emisă în fiecare zi?

Este exemplul 22 din algebra relațională (figura 2.27), formulat pentru ilustrarea operatorului diviziune.

• Soluția 1 – DB2 (și Oracle, operând modificările în sintaxa juncțiunii externe):

```
SELECT DISTINCT DenCl
FROM CLIENTI
WHERE CodCl NOT IN
  (SELECT DISTINCT PROD_CART.CodCl
   FROM
     (SELECT DISTINCT CLIENTI.CodCl, FACTURI.DataFact
      FROM CLIENTI, FACTURI) PROD_CART
    LEFT OUTER JOIN FACTURI
      ON PROD_CART.CodCl=FACTURI.CodCl
      AND PROD_CART.DataFact=FACTURI.DataFact
   WHERE FACTURI.CodCl IS NULL)
```

• Soluția 2 (cea preferată) – DB2 și Oracle:

```
SELECT DISTINCT DenCl
FROM
  (SELECT CodCl, COUNT(DISTINCT DataFact) AS Nr
   FROM FACTURI
   GROUP BY CodCl) TEMP1,
  (SELECT COUNT(DISTINCT DataFact) AS Nr
   FROM FACTURI) TEMP2,
  CLIENTI
WHERE TEMP1.Nr=TEMP2.Nr AND TEMP1.CodCl=CLIENTI.CodCl
```

In ce zile s-au vândut și produsul cu denumirea „Produs 1”, și cel cu denumirea „Produs 2”?

Este (tot) exemplul 23 din capitolul 2.

• Soluția 1 – DB2 (și Oracle, înlocuind LEFT OUTER JOIN):

```
SELECT DISTINCT DataFact
FROM FACTURI F, LINIIFACT LF, PRODUSE P
WHERE F.NrFact=LF.NrFact AND LF.CodPr=P.CodPr AND
  DenPr IN ('Produs 1', 'Produs 2')
  AND DataFact NOT IN
    (SELECT DISTINCT PROD_CART.DataFact
     FROM
       (SELECT DISTINCT DataFact, CodPr
        FROM FACTURI, PRODUSE
        WHERE DenPr IN ('Produs 1', 'Produs 2'))
      ) PROD_CART
LEFT OUTER JOIN
  (SELECT F.DataFact, LF.CodPr
   FROM FACTURI F, LINIIFACT LF, PRODUSE P
   WHERE F.NrFact=LF.NrFact AND LF.CodPr=P.CodPr
   ) TEMP1
  ON PROD_CART.DataFact=TEMP1.DataFact AND
    PROD_CART.CodPr=TEMP1.CodPr
  WHERE TEMP1.DataFact IS NULL)
```

• Soluția 2 – DB2/Oracle:

```
SELECT DISTINCT DataFact
FROM
  (SELECT F.DataFact, COUNT(DISTINCT LF.CodPr) AS Nr
   FROM FACTURI F, LINIIFACT LF, PRODUSE P
   WHERE F.NrFact=LF.NrFact AND LF.CodPr=P.CodPr AND
     DenPr IN ('Produs 1', 'Produs 2')
     GROUP BY F.DataFact) TEMP1
  WHERE Nr = 2
```

Din nou, soluția a doua contrastează evident, prin simplitate, cu prima. Tabela TEMP1 conține, pentru fiecare dată calendaristică, numărul de produse, dintre Produs 1 și Produs 2 (1 sau 2) care au fost vândute în ziua respectivă.

VINZARI

NRFACT	FACTURAT
1111	5399625.00
1112	1337560.00
1113	1160250.00
1114	6786570.00
1115	1651125.00
1116	1383375.00
1117	2320500.00
1118	2052750.00
1119	7242935.00
1120	1066240.00
1121	5438300.00

INCASARI

NRFACT	INCASAT
1111	5399625
1112	487705
1113	1160250
1117	2320500
1118	2052750
1120	731557

Figura 6.2. Rezultatele celor două subconsultări din clauza FROM

Cele două variante nu sunt echivalente. Soluția Oracle 8 afișează, în plus, și situația, oarecum anormală, când valoarea încasată a unei unei facturi o depășește pe cea facturată.

Să se obțină sporurile de noapte pentru al doilea trimestru al anului 2000, atât lunar, cât și cumulat.

• Soluția SQL-92/DB2:

```
SELECT SL1.Marcă, NumePren,
       VALUE(SL1.SporNoapte,0) AS Spor_Noapte_Aprilie,
       VALUE(SL2.SporNoapte,0) AS Spor_Noapte_Mai,
       VALUE(SL3.SporNoapte,0) AS Spor_Noapte_Iunie,
       VALUE(SL1.SporNoapte,0) + VALUE(SL2.SporNoapte,0) +
       VALUE(SL3.SporNoapte,0) AS Spor_Noapte_Trim_II
  FROM
    (SELECT PERSONAL2.Marcă, NumePren, SporNoapte
      FROM PERSONAL2
     LEFT OUTER JOIN SPORURI
       ON PERSONAL2.Marcă=SPORURI.Marcă
      AND An=2000 AND Luna =4) SL1
    INNER JOIN
    (SELECT PERSONAL2.Marcă, SporNoapte
      FROM PERSONAL2 LEFT OUTER JOIN SPORURI
       ON PERSONAL2.Marcă=SPORURI.Marcă
      AND An=2000 AND Luna =5) SL2
    ON SL1.Marcă=SL2.Marcă
    INNER JOIN
    (SELECT PERSONAL2.Marcă, SporNoapte
      FROM PERSONAL2 LEFT OUTER JOIN SPORURI
       ON PERSONAL2.Marcă=SPORURI.Marcă
      AND An=2000 AND Luna =6) SL3
    ON SL1.Marcă=SL3.Marcă
  ORDER BY NumePren
```

Soluția Oracle 8:

```
SELECT SL1.Marcă, NumePren,
       NVL(SL1.SporNoapte,0) AS Spor_Noapte_Aprilie,
       NVL(SL2.SporNoapte,0) AS Spor_Noapte_Mai,
       NVL(SL3.SporNoapte,0) AS Spor_Noapte_Iunie,
       NVL(SL1.SporNoapte,0) + NVL(SL2.SporNoapte,0) +
       NVL(SL3.SporNoapte,0) AS Spor_Noapte_Trim_II
  FROM
    (SELECT PERSONAL2.Marcă, NumePren, SporNoapte
      FROM PERSONAL2, SPORURI
     WHERE PERSONAL2.Marcă=SPORURI.Marcă (+)
       AND An (+) = 2000 AND Luna (+)=4) SL1,
    (SELECT PERSONAL2.Marcă, SporNoapte
      FROM PERSONAL2, SPORURI
     WHERE PERSONAL2.Marcă=SPORURI.Marcă (+)
       AND An (+) = 2000 AND Luna (+)=5) SL2,
    (SELECT PERSONAL2.Marcă, SporNoapte
      FROM PERSONAL2, SPORURI
     WHERE PERSONAL2.Marcă=SPORURI.Marcă (+)
       AND An (+) = 2000 AND Luna (+) =6) SL3
   WHERE SL1.Marcă=SL2.Marcă AND SL1.Marcă=SL3.Marcă
  ORDER BY NumePren
```

Clauza FROM principală „calculează” trei tabele ce conțin sporurile de noapte ale lunilor aprilie (SL1), mai (SL2) și iunie (SL3) 2000 ale fiecărui angajat (indiferent de data angajării acestuia) – figura 6.3. Cele trei tabele sunt jonctionate după atributul Marca.

Revînem la diviziunea relațională. Succesiunea de pași prezentată în figura 2.28 se poate realiza și prin soluția SQL-92/DB2:

```
SELECT DISTINCT X
  FROM R1
 WHERE X NOT IN
    (SELECT DISTINCT PROD_CART.X
      FROM (SELECT DISTINCT R1.X, R2.Y FROM R1,R2) PROD_CART
     LEFT OUTER JOIN R1 ON PROD_CART.X=R1.X
     AND PROD_CART.Y=R1.Y
    WHERE R1.X IS NULL)
```

SL1

MARCA	NUMEPREN	SPORNOAPTE
1ANGAJAT 1		0
2ANGAJAT 2		450000
3ANGAJAT 3		560000
4ANGAJAT 4		0
5ANGAJAT 5		0
6ANGAJAT 6		0
7ANGAJAT 7		0
8ANGAJAT 8		0
9ANGAJAT 9		0
10ANGAJAT 10		0

SL2

MARCA	SPORNOAPTE
1	0
2	450000
3	0
4	0
5	0
6	0
7	0
8	0
9	0
10	0

SL3

MARCA	SPORNOAPTE
1	0
2	0
3	0
4	150000
5	150000
6	0
7	0
8	0
9	0
10	0

Figura 6.3. Rezultatele celor trei subconsultări

```

SELECT DISTINCT NrFact
FROM LINIIFACT
WHERE CodPr IN
  (SELECT CodPr
   FROM LINIIFACT
   WHERE NrFact = 1112)
GROUP BY NrFact
HAVING COUNT(*) =
  (SELECT COUNT(CodPr)
   FROM LINIIFACT
   WHERE NrFact = 1112)

```

Care sunt clienții cărora li s-au vândut cel puțin produsele vândute clientului CLIENT 4?

```

SELECT DenCl
FROM CLIENTI C1
WHERE NOT EXISTS
  (SELECT 1
   FROM CLIENTI C2, FACTURI F2, LINIIFACT LF2
   WHERE F2.NrFact=LF2.NrFact AND C2.CodCl=F2.CodCl
     AND DenCl='Client 4' AND NOT EXISTS
       (SELECT 1
        FROM FACTURI F3, LINIIFACT LF3
        WHERE F3.NrFact=LF3.NrFact AND
          C1.CodCl=F3.CodCl AND LF3.CodPr=LF2.CodPr))

```

Analizând enunțul problemei „Care sunt clienții...?”, rezultă că atributul pentru corelarea consultării principale (top) și celei de jos (bottom) este CodCl, iar din partea condițională a enunțului „... cel puțin produsele...” pentru corelarea mijloc (middle) – jos (bottom) se utilizează atributul CodPr.

6.2. Subconsultări în clauza FROM

Poibilitatea de a defini tabele ad-hoc, în clauza FROM, este un privilegiu pe care și-l pot permite numai SGBD-urile profesionale, serverele de baze de date. Ne desprîjm, aşadar, din nou de VFP pentru o vreme.

Care sunt valorile facturate și incasate, precum și situația („fără nici o incasare”, „incasată parțial” sau „incasată total”) pentru fiecare factură?

- Soluția SQL-92/DB2:

```

SELECT VINZARI.NrFact, Facturat, VALUE(Incasa,0) AS Incasa,
       Facturat - VALUE(Incasa,0) AS Diferenta,
CASE
  WHEN VALUE(Incasa,0) = 0
    THEN 'Fara nici o incasare'
  WHEN Facturat > VALUE(Incasa,0)
    THEN 'Incasata parțial'
  ELSE 'INCASATA TOTAL'
END AS Situație

```

```

FROM
  (SELECT NrFact,
         SUM(Cantitate * PretUnit * (1+ProcTVA))
           AS Facturat
   FROM LINIIFACT LF INNER JOIN PRODUSE P
   ON LF.CodPr = P.CodPr
  GROUP BY NrFact) VINZARI
LEFT OUTER JOIN
  (SELECT NrFact, SUM(Transa) AS Incasa
   FROM INCASFACT
   GROUP BY NrFact) INCASARI
ON VINZARI.NrFact = INCASARI.NrFact

```

• Soluția Oracle 8:

```

SELECT VINZARI.NrFact, Facturat, NVL(Incasa,0) AS Incasa,
       Facturat - NVL(Incasa,0) AS Diferenta,
DECODE (SIGN(Facturat-NVL(Incasa,0)),
  0, 'INCASATA TOTAL',
  -1, 'Incasa mai mult decât facturat !!!',
  DECODE (NVL(Incasa,0),
  0, 'Fara nici o incasare',
  'Incasata parțial')
)
AS Situație

```

```

FROM
  (SELECT NrFact,
         SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Facturat
   FROM LINIIFACT LF, PRODUSE P
   WHERE LF.CodPr = P.CodPr
  GROUP BY NrFact) VINZARI,
  (SELECT NrFact, SUM(Transa) AS Incasa
   FROM INCASFACT
   GROUP BY NrFact) INCASARI
WHERE VINZARI.NrFact = INCASARI.NrFact (+)

```

In clauza FROM a frazei SELECT principale au fost definite două tabele în totă regula, VINZARI și INCASARI – vezi figura 6.2. Prima conține valoarea totală a fiecărei facturi, în timp ce a doua conține valoarea incasată. Aceste două tabele sunt jonctionate extern după atributul NrFact. În Oracle 8, în locul structurii CASE s-a folosit funcția DECODE.

```

FROM FACTURI F2
WHERE NOT EXISTS
  (SELECT 1
   FROM CLIENTI C3, FACTURI F3
   WHERE C3.CodCl=F3.CodCl AND
         C3.CodCl=C1.CodCl AND
         F3.DataFact = F2.DataFact
  )
)

```

Toate soluțiile ce întrebuițează dubla corelare de până acum au o concurență care le pune în umbra. Variantele ne-bazate-pe-dubla-corelare sunt mai simple, atât ca dimensiune, cât și ca mod de înțelegere. Este timpul să scoatem din mânecă o problemă de diviziune relațională care să demonstreze adeverata forță a dublei corelări și să justifice generosul efort intelectual pentru a o înțelege:

Ce facturi conțin mărcaj produsele din factura 1112?

```

SELECT DISTINCT NrFact
FROM LINIIFACT LF1
WHERE NOT EXISTS
  (SELECT 1
   FROM LINIIFACT LF2
   WHERE NrFact = 1112
   AND NOT EXISTS
     (SELECT 1
      FROM LINIIFACT LF3
      WHERE LF3.CodPr = LF2.CodPr AND
            LF3.NrFact=LF1.NrFact )
  )

```

LINIIFACT are 22 de linii, deci execuția se desfășoară în 22 de faze, dintre care prezentăm două:

Faza 1:

- *Pas 1.1:* se „încarcă” primul tuplu din LF1: (1111, 1, 1, 50, 10000) și primul tuplu din LF2 care îndeplinește condiția NrFact = 1112: (1112, 1, 2, 80, 10300).
- *Pas 1.2.1:* se testează dacă există în LF3 mărcaj o linie în care NrFact=1111 (LF1.NrFact) și CodPr=2 (LF2.CodPr). Există (a doua), astfel încât se trece la următoarea linie din LF2 care îndeplinește condiția NrFact = 1112: (1112, 1, 3, 40, 7500).
- *Pas 1.2.2:* se testează dacă există în LF3 mărcaj o linie în care NrFact=1111 (LF1.NrFact) și CodPr=3 (LF2.CodPr). Nu există, iar
- *Pas 1.3:* subconsultarea de jos (bottom) întoarce FALSE către subconsultarea din mijloc (middle). SELECT-ul mijlociu recepționează valoarea logică FALSE și, datorită operatorului NOT EXISTS, „pasează” SELECT-ului principal TRUE.
- *Pas 1.4:* fraza SELECT principală recepționează valoarea logică TRUE de la subconsultarea mijlocie, pe care, la rândul său, o transformă în FALSE, iar valoarea 1111 nu va fi inclusă în rezultat!

Faza 16:

- *Pas 16.1:* se „încarcă” tuplul 16 din LF1: (1119, 1, 2, 35, 10900) și primul tuplu din LF2 care îndeplinește condiția NrFact = 1112: (1112, 1, 2, 80, 10300).
- *Pas 16.2.1:* se testează dacă există în LF3 mărcaj o linie în care NrFact=1119 (LF1.NrFact) și CodPr=2 (LF2.CodPr). Există (tuplul 16), astfel încât se trece la următoarea linie din LF2 care îndeplinește condiția NrFact = 1112: (1112, 1, 3, 40, 7500).
- *Pas 16.2.2:* se testează dacă există în LF3 mărcaj o linie în care NrFact=1119 (LF1.NrFact) și CodPr=3 (LF2.CodPr). Există (tuplul 17), astfel încât se încearcă trecerea la următoarea linie din LF2 care îndeplinește condiția NrFact = 1112. Cum nu mai există nici o asemenea linie,
- *Pas 16.3:* subconsultarea de jos (bottom) întoarce TRUE către subconsultarea din mijloc (middle), care, la rândul său, întoarce FALSE SELECT-ului principal.
- *Pas 16.4:* fraza SELECT principală recepționează valoarea logică FALSE de la subconsultarea mijlocie, prin negarea căreia obține TRUE, iar valoarea 1119 va fi inclusă în rezultat!

Fraza SELECT analizată răspunde, de fapt, la întrebarea: Pentru care facturi nu există nici un produs ce apare în factura 1112, dar nu este prezent în factura respectivă?

Fără dubla corelare, pot fi formulate și soluții bazate pe MINUS (EXCEPT) și IN, precum:

```

SELECT NrFact
FROM LINIIFACT
MINUS
  SELECT DISTINCT LF1.NrFact
  FROM LINIIFACT LF1, LINIIFACT LF2
  WHERE LF2.NrFact = 1112 AND
        (LF1.NrFact, LF2.CodPr) NOT IN
          (SELECT NrFact, CodPr
           FROM LINIIFACT)

```

(soluție Oracle; în DB2 se înlocuiește MINUS cu EXCEPT) sau cu aceeași logică, dar ce utilizează tandemul MINUS (EXCEPT)-NOT EXISTS:

```

SELECT NrFact
FROM LINIIFACT
MINUS
  SELECT DISTINCT LF1.NrFact
  FROM LINIIFACT LF1, LINIIFACT LF2
  WHERE LF2.NrFact = 1112 AND NOT EXISTS
    (SELECT 1
     FROM LINIIFACT LF3
     WHERE LF3.NrFact=LF1.NrFact AND
           LF3.CodPr=LF2.CodPr)

```

Deși ultimele două soluții sunt comparabile, ca dimensiune, cu varianta bazată pe dubla corelare, faptul că logica lor presupune produsul cartezian a două instanțe ale tabelei LINIIFACT, tabela care, în timp, acumulează un uriaș număr de înregistrări, le pune în umbrelă.

Pentru a respecta adeverul istoric, trebuie spus că se poate recurge la artificii non-diviziune relațională, precum cel următor:

În cazul corelării simple, scenariul de execuție este unul de tip *top-bottom-top*; la corelarea dublă lucrurile stau cu mult mai interesant: *top-bottom-middle-bottom-middle-top*. Curat ca lacrima și explicit ca reforma la români!

Dacă n-ar fi fost în joc diviziunea relațională, n-am fi ajuns aici cu „hermeneutica” SQL. Revenim la cele două tabele, R1 și R2, pe care le-am folosit în capitolul 2 la prezentarea diviziunii relaționale. Valorile lui X care se găsesc în R1 în combinație cu toate valorile lui Y din R2 pot fi aflate și prin interogarea următoare:

```
SELECT DISTINCT X
FROM R1 T1_1
WHERE NOT EXISTS
  (SELECT 1
   FROM R2
   WHERE NOT EXISTS
     (SELECT 1
      FROM R1 T1_2
      WHERE T1_2.X=T1_1.X AND T1_2.Y=T2.Y
     )
  )
```

Execuția frazei SQL se derulează astfel:

Faza 1:

- *Pas 1.1:* se „încarcă” primul tuplu din T1_1: (x1, y1) și primul tuplu din R2: y1.
 - *Pas 1.2.1:* se testează dacă există în T1_2 o linie în care X=x1 (T1_1.X) și Y=y1 (R2.Y). Există!, astfel încât se trece la următoarea linie din R2.
 - *Pas 1.2.2:* se testează dacă există în T1_2 o linie în care X=x1 (T1_1.X) și Y=y2 (R2.Y). Există!, deci se „avanseză” încă o linie în R2.
 - *Pas 1.2.3:* se testează dacă există în T1_2 o linie în care X=x1 (T1_1.X) și Y=y3 (R2.Y). Există!, deci se „avanseză” încă o linie în R2.
 - *Pas 1.2.4:* se testează dacă există în T1_2 o linie în care X=x1 (T1_1.X) și Y=y5 (R2.Y). Există!, deci se „avanseză” încă o linie în R2.
 - *Pas 1.2.5:* se testează dacă există în T1_2 o linie în care X=x1 (T1_1.X) și Y=y5 (R2.Y). Există!, deci se încearcă încărcarea următoarei linii din R2.
 - *Pas 1.2.6:* intrucât am ajuns la sfârșitul tablei R2 și toate valorile din R2 s-au regăsit în T1_2 în combinație cu x1,
- *Pas 1.3.:* subconsultarea de jos (*bottom*) întoarce TRUE către subconsultarea din mijloc (*middle*). Subconsultarea din mijloc recepționează rezultatul SELECT-ului de jos și îl pasează cu aceeași valoare logică sau cu valoare schimbă (dacă apare NOT) interogării principale. În această situație, SELECT-ul mijlociu primește de la cel de jos valoarea logică TRUE, însă, deoarece operatorul de conexiune mijloc-jos este NOT EXISTS, va întoarce SELECT-ului principal FALSE.
- *Pas 1.4.:* fraza SELECT principală recepționează valoarea logică FALSE de la subconsultarea mijlocie. Dar operatorul de conexiune sus-mijloc este NOT EXISTS, iar FALSE-ul este transformat în TRUE, iar valoarea x1 va fi inclusă în rezultat!

Faza 2:

- *Pas 2.1:* se „încarcă” al doilea tuplu din T1_1: (x2, y1) și primul tuplu din R2: y1.
 - *Pas 2.2.1:* se testează dacă există în T1_2 o linie în care X=x2 (T1_1.X) și Y=y1 (R2.Y). Există!, astfel încât se trece la următoarea linie din R2.
 - *Pas 2.2.2:* se testează dacă există în T1_2 o linie în care X=x2 (T1_1.X) și Y=y2 (R2.Y). Nu există!, iar încărcarea celorlalte linii din R2 este abandonată și
- *Pas 2.3:* subconsultarea de jos (*bottom*) întoarce FALSE către subconsultarea din mijloc (*middle*). SELECT-ul mijlociu recepționează valoarea logică FALSE și, datorită operatorului NOT EXISTS, „pasează” SELECT-ului principal TRUE.
- *Pas 2.4:* fraza SELECT principală recepționează valoarea logică TRUE de la subconsultarea mijlocie, pe care, la rândul său, o transformă în FALSE, iar valoarea x2 nu va fi inclusă în rezultat!

Lucrurile se continuă cu încă 14 faze, una pentru fiecare linie din R1. Clauza DISTINCT asigură prelucrarea în rezultat a fiecărei valori o singură dată.

Dacă privim mai îndeaproape fraza SELECT, observăm că, de fapt, aceasta răspunde la întrebarea: pentru care dintre valorile lui X nu există nici un Y care să nu apară cu X-ul respectiv în combinație, pe (măcar) o linie în R1.

În ce zile s-au vândut și produsul cu denumirea „Produs 1”, și cel cu denumirea „Produs 2”?

Soluția SQL-99/DB2/Oracle bazată pe corelare dublă este probabil cea mai complicată, mult prea complicată pentru așa o bagătelă de problemă:

```
SELECT DISTINCT DataFact
FROM FACTURI F1, LINIIFACT LF1, PRODUSE P1
WHERE F1.NrFact = LF1.NrFact AND LF1.CodPr=P1.CodPr AND
      NOT EXISTS
        (SELECT 1
         FROM PRODUSE P1
         WHERE DenPr IN ('Produs 1', 'Produs 2') AND
              NOT EXISTS
                (SELECT 1
                 FROM FACTURI F2, LINIIFACT LF2, PRODUSE P2
                 WHERE F2.NrFact = LF2.NrFact AND
                       LF2.CodPr=P2.CodPr AND
                         F2.DataFact=F1.DataFact AND
                           P2.CodPr=P1.CodPr
                )
        )
```

Extragăți clienții pentru care există cel puțin câte o factură emisă în fiecare zi.

Reluăm această problemă de la diviziunea relațională ce poate fi rezolvată și prin SELECT-ul următor:

```
SELECT DISTINCT DenCl
FROM CLIENTI C1, FACTURI F1
WHERE C1.CodCl=F1.CodCl AND
      NOT EXISTS
        (SELECT 1
```

Faza 2:

- Se „încarcă” a doua linie din CLIENTI: (1002, ‘Client 2 SA’, ‘R1002’, NULL, ‘6600’, ‘032-212121’).
 - Se numără, prin funcția COUNT, câte linii din FACTURI au CodCl = 1002. Rezultatul este 1.
 - $1 \neq 2$, deci nici ‘CLIENT2’ nu se include în rezultat.
- ...

Faza 5:

- Se „încarcă” a cincea linie din CLIENTI: (1005, ‘Client 5 SRL’, ‘R1005’, NULL, ‘1900’, ‘056-111111’).
 - Se numără, prin funcția COUNT, câte linii din FACTURI au CodCl = 1005. Rezultatul este 2.
 - $2 = 2$, iar ‘CLIENT5’ se include în rezultat.
- ...

Din interogările menite să ilustreze folosirea operatorului **EXISTS** nu puteau lipsi cele legate de diviziunea relațională. Reluăm exemplul simplu ce utilizează relațiile R1 și R2. Pentru a lăsa valorile lui X afilate în R1 în combinație cu toate valorile lui Y din R2, se poate recurge și la soluția SQL-92/DB2/Oracle:

```
SELECT DISTINCT X
FROM R1 T1_1
WHERE EXISTS
  (SELECT 1
   FROM R1 T1_2
   WHERE T1_2.X = T1_1.X
   GROUP BY T1_2.X
   HAVING COUNT(DISTINCT T1_2.Y) =
      (SELECT COUNT(Y)
       FROM R2)
  )
```

Se parcurge, pe rând, fiecare linie din R1. De fiecare dată se verifică dacă, pentru X-ul curent, în R1 apar toți igrecii din R2, verificare realizată prin GROUP BY și HAVING. Pe baza acestei soluții, se poate formula o altă soluție SQL-92/DB2/Oracle și la problema:

Extragăți clienții pentru care există cel puțin câte o factură emisă în fiecare zi.

```
SELECT DISTINCT DenCl
FROM CLIENTI C1, FACTURI F1
WHERE C1.CodCl=F1.CodCl AND EXISTS
  (SELECT 1
   FROM FACTURI F2
   WHERE F2.CodCl = F1.CodCl
   GROUP BY F2.CodCl
   HAVING COUNT(DISTINCT F2.DataFact) =
      (SELECT COUNT(DISTINCT DataFact)
       FROM FACTURI)
```

)

Ultima consultare (de jos), cea din clauza HAVING, calculează numărul total de zile în care sunt vânzări, număr care este 5. Subconsultarea „mijlocie” determină dacă numărul de zile de vânzări pentru clientul curent (C1.CodCl) este egal cu 5 (rezultatul subconsultării de jos).

Care sunt cele mai mari cinci prejuri unitare la care s-au efectuat vânzări?

Această problemă a mai fost formulată în paragraful 5.4. Revenim cu o soluție nou-nouă, bazată pe subinterrogări corelate.

```
SELECT PretUnit
FROM LINIIFACT LF1
WHERE 5 >
  (SELECT COUNT(DISTINCT LF2.PretUnit)
   FROM LINIIFACT LF2
   WHERE LF2.PretUnit > LF1.PretUnit)
ORDER BY PretUnit DESC
```

Idea este grozav de ingenioasă (păcat că nu este a mea): se parcurge linie cu linie prima instanță a tabeliei LINIIFACT – LF1. Pentru fiecare linie din LF1 se numără în a doua instanță a LINIIFACT – LF2 câte linii prezintă prejul unitar mai mare decât cel de pe linia curentă din LF1. Dacă numărul calculat prin COUNT(*) este mai mic decât 5, atunci în rezultat se extrage valoarea LF1.PretUnit.

Cum nici eu n-am înțeles prea bine explicațiile de mai sus, să analizăm interogarea mai pe indelete.

- *Pas 1:* se ia în discuție primul tuplu din LINIIFACT (LF1). PretUnit are valoarea 10000. Se calculează numărul valorilor distincte ale PretUnit din LF2 pentru care LF2.PretUnit > LF1.PretUnit. Acest număr este 8. Prin urmare, în LINIIFACT există opt prejuri unitari peste 10000. Întrucât $8 > 5$, primul tuplu din LF1 nu este inclus în rezultat.

- *Pas 2:* se „încarcă” al doilea tuplu din LINIIFACT (LF1). PretUnit are valoarea 10500. Funcția COUNT din subconsultare întoarce 6 – în LINIIFACT există șase linii cu prejuri unitare peste 10600. Cum $6 > 5$, nici al doilea tuplu din LF1 nu este inclus în rezultat.

- *Pas 3:* se „încarcă” al treilea tuplu din LINIIFACT (LF1), în care PretUnit este 6500. COUNT întoarce 15, deci nici al treilea tuplu din LF1 nu este inclus în rezultat.

...

- *Pas 8:* pe această linie PretUnit este 15800. COUNT întoarce 0; prin urmare, valoarea 15800 va face parte din rezultat.

În total sunt 22 de pași, corespunzători tuplurilor din LINIIFACT. Păcat că VFP nu suportă asemenea găselinișă, sau cel puțin aşa spune mesajul care ne întâmpină la execuția acestei interogări.

Subconsultări dublu corelate

Dubla corelare constituie o facilitate bine ocolită de către dezvoltatorii de aplicații (eu, unul, am reușit să evit cu brio până în acest paragraf). Sheryl Larsen estima în 1998 că mai puțin de 10% dintre dezvoltatorii de aplicații în DB2 aveau cunoștință de interogările dublu corelate. Mărturisesc că în articolul lui Sheryl⁵⁶ am găsit cea mai bună explicație a acestui mecanism.

56. [Larsen98].

```
        DenPr = 'Produs 2  
    }  
}  
}
```

Această soluție este una dintre cele mai bune, ca volum de resurse consumat, înlocuind atât costisitoarea jocurile din interogările de tip INNER JOIN, cât și numeroasele tabele intermediare presupuse de operatorul IN. Nu ne încercă decât un regret: că nu funcționează în VFP (din cauza multiplelor niveluri de consultare).

Prin EXISTS, firește, se poate realiza și intersecția a două relații. Dacă luăm în discuție relațiile R1 și R2 (capitolul 2 – figura 2.4), intersecția acestora se realizează și astfel:

```
SELECT *
FROM R1
WHERE EXISTS
  (SELECT 1
   FROM R2
   WHERE R1.A=R2.C AND R1.B=R2.D AND R1.C=R2.E)
```

La prima vedere, nimic spectaculos. La a doua, descoperim că, spre deosebire de soluția bazată pe IN, această frază funcționează identic și în VFP.

In ce zile s-au vândut și produsul cu denumirea „Produs 1”, și cel cu denumirea „Produs 2”?

```
SELECT DISTINCT DataFact
FROM PRODUSE P1, LINIIFACT LF1, FACTURI F1
WHERE P1.CodPr = LF1.CodPr AND
      LF1.Nrfact = F1.NrFact AND DenPr = 'Produs 1'
AND EXISTS
    (SELECT 1
     FROM PRODUSE P2, LINIIFACT LF2, FACTURI F2
     WHERE P2.CodPr = LF2.CodPr AND LF2.Nrfact = F2.NrFact
          AND P2.DenPr = 'Produs 2' AND
              F2.DataFact=F1.DataFact)
```

Sunt corelate două instanțe ale juncțiunii PRODUSE-LINIIFACT-FACTURI; prima conține linile legate de *Produs 1*, iar a doua de *Produs 2*. Întrucât interesează zilele în care s-au vândut simultan cele două produse, atributul de corelare este **DataFact**.

Ce clienti au cumpărat și „Produs 2”, și „Produs 3”, dar nu au cumpărat „Produs 5”?

```
SELECT DISTINCT DenCl
FROM PRODUSE, LINIIFACT, FACTURI, CLIENTI
WHERE PRODUSE.CodPr = LINIIFACT.CodPr AND
      LINIIFACT.Nrfact = FACTURI.NrFact AND
      FACTURI.CodCl = CLIENTI.CodCl AND DenPr = 'Produs 2'
AND EXISTS
    (SELECT 1
     FROM PRODUSE, LINIIFACT, FACTURI
     WHERE PRODUSE.CodPr = LINIIFACT.CodPr AND
```

```

LINIIFACT.NrFact = FACTURI.NrFact AND
    DenPr = 'Produs 3' AND CodCl=CLIENTI.CodCl
AND NOT EXISTS
    (SELECT 1
    FROM PRODUSE, LINIIFACT, FACTURI
    WHERE PRODUSE.CodPr = LINIIFACT.CodPr AND
        LINIIFACT.NrFact = FACTURI.NrFact AND
        DenPr = 'Produs 5' AND CodCl=CLIENTI.CodCl)

```

Fraza SELECT principală este corelată la două subconsultaři dispuse „în linie”, secvențial. S-au folosit și EXISTS și NOT EXISTS. Nici aceasta nu este o dublă corelare, deoarece avem de-a face cu un singur nivel de subconsultare.

Care sunt clienții cărora li s-au întocmit numai două facturi?

Este o problemă banală, pentru care se pot formula soluții interesante. Firește, cea mai lăudată variantă este:

```
SELECT DenCl  
FROM CLIENTI  
WHERE CodCl IN  
(SELECT CodCl  
FROM FACTURI  
GROUP BY CodCl  
HAVING COUNT(NrFact) = 2)
```

Si mai simplă decât aceasta este următoarea

```
SELECT DenCl
  FROM CLIENTI
    INNER JOIN FACTURI ON CLIENTI.CodCl=FACTURI.CodC
   GROUP BY DenCl
  HAVING COUNT(NrFact) = 2
```

Varianta pe care o consider cea mai interesantă nu folosește, pentru corelarea subconsultării cu fraza **SELECT** principală, nici **EXISTS**, nici **IN**:

```
SELECT DenCl  
FROM CLIENTI  
WHERE 2 =  
    (SELECT COUNT(*)  
     FROM FACTURI  
     WHERE FACTURI.CodCl=CLIENTI.CodCl)
```

Logica interogării

Faza I

- Se „încarcă” prima linie din CLIENTI: (1001, ‘Client 1 SRL’, ‘R1001’, ‘Tranzitiei, 13 bis’, ‘6600’, NULL).
 - Se numără, prin funcția COUNT, câte linii din FACTURI au CodC1 = 1001. Rezultatul este 5.
 - Se compară 2 din SELECT-ul principal cu rezultatul subconsultării corelate; 5 ≠ 2, deci ‘CLIENTI’ nu se include în rezultat.

Dacă se dorește eliminarea din rezultat și a fakturii de referință, 1120, se modifică ușor interogarea:

```
SELECT NrFact
FROM FACTURI F1
WHERE NrFact <> 1120 AND
EXISTS
  (SELECT DataFact
  FROM FACTURI F2
  WHERE F2.NrFact=1120 AND F1.DataFact=F2.DataFact)
```

Interesant este că interogările corelate pot fi legate nu numai prin operatorul EXISTS, ci și de IN. Altfel spus, interogarea următoare este perfect similară anterioarei:

```
SELECT NrFact
FROM FACTURI F1
WHERE NrFact <> 1120 AND
DataFact IN
  (SELECT DataFact
  FROM FACTURI F2
  WHERE NrFact=1120 AND F1.DataFact=F2.DataFact)
```

Ce facturi au fost emise în alte zile decât factura 1120?

Următoarele două soluții conduc la același rezultat:

```
SELECT NrFact
FROM FACTURI F1
WHERE NOT EXISTS
  (SELECT *
  FROM FACTURI F2
  WHERE F2.NrFact=1120 AND F1.DataFact=F2.DataFact)
```

sau

```
SELECT NrFact
FROM FACTURI F1
WHERE EXISTS
  (SELECT *
  FROM FACTURI F2
  WHERE F2.NrFact=1120 AND F1.DataFact<>F2.DataFact)
```

În subconsultarea corelată (cea care succedă operatorului EXISTS), clauza SELECT poate conține ca argument * sau un atribut care, de preferință, nu are valori nule. Cea mai simplă, sigură și rapidă (pentru SBGD) varianță este însă folosirea unei constante. Spre exemplu, ultima variantă se poate schimba, aproape înseuzabil, în:

```
SELECT NrFact
FROM FACTURI F1
WHERE EXISTS
  (SELECT 1
  FROM FACTURI F2
  WHERE F2.NrFact=1120 AND F1.DataFact<>F2.DataFact)
```

Înlocuirea asteriscului sau a oricărui alt atribut cu 1 nu modifică în nici un fel corectitudinea interogării și este de bun augur pentru simplitate și viteza de execuție.

Care sunt clienții cărora li s-au trimis facturi în aceeași zi în care a fost întocmită factura 1120?

- Soluție SQL-92/DB2/Oracle:

```
SELECT DenCl
FROM CLIENTI C1
WHERE EXISTS
  (SELECT 1
  FROM FACTURI F1
  WHERE F1.CodCl=C1.CodCl AND
EXISTS
  (SELECT 1
  FROM FACTURI F2
  WHERE F2.NrFact=1120 AND
F1.DataFact=F2.DataFact)
)
```

- Soluție VFP:

```
SELECT DenCl ;
FROM CLIENTI C INNER JOIN FACTURI F1 ;
ON C.CodCl=F1.CodCl ;
WHERE EXISTS ;
  (SELECT 1 ;
  FROM FACTURI F2 ;
  WHERE F2.NrFact=1120 AND F1.DataFact=F2.DataFact)
```

În varianta SQL-92/DB2/Oracle există două niveluri de subconsultare, însă corelarea este simplă, fiecare „etaj” fiind corelat numai la nivelul imediat superior. Peste numai câteva pagini vom discuta câteva aspecte privind dubla corelare.

În ce județ s-a vândut produsul „Produs 2”?

Formulăm numai soluția SQL-92/DB2/Oracle:

```
SELECT Judet
FROM JUDETE J
WHERE EXISTS
  (SELECT 1
  FROM LOCALITATI L
  WHERE L.Jud=J.Jud AND EXISTS
    (SELECT 1
    FROM CLIENTI C
    WHERE C.CodPost=L.CodPost AND EXISTS
      (SELECT 1
      FROM FACTURI F
      WHERE F.CodCl=C.CodCl AND EXISTS
        (SELECT 1
        FROM LINIIFACT LF
        WHERE LF.NrFact=F.NrFact AND EXISTS
          (SELECT 1
          FROM PRODUSE P
          WHERE P.CodPr=LF.CodPr AND
```

2 niveluri de subconsultare

Capitolul 6

SQL ȘI MAI AVANSAT

Primul dintre capitolele dedicate frazelor SELECT, 4, se numește Elemente de bază ale interogărilor SQL, iar capitolul 5 – SQL (ceva) mai avansat. Cum prezentul capitol a fost intitulat SQL și mai avansat, în mod normal, dacă și următorul continuu problematica interogărilor, ar fi trebuit să se numească SQL al naibii de avansat. Din fericire, acesta este ultimul în care se discută despre opțiuni de consultare, câteva opțiuni care se încadrează în elementele de finețe ale limbajului. Concret, vor fi abordate: corelarea simplă și dublă cu și fără ajutorul operatorului EXISTS, subinterrogări în clauza FROM, interogările scalare și cele ierarhice.

6.1. Interrogări corelate. Operatorul EXISTS

Ca și IN, operatorul EXISTS permite legarea frazei SELECT principale de una sau mai multe subconsultări și, astfel, formularea unor interogări complexe. Prin EXISTS se definiște un predicat care are valoarea logică „adevărat” dacă subconsultarea s-a concretizat într-o tabelă ce are cel puțin o linie.

În general, EXISTS poate înlocui cu succes operatorul IN. Reciproca nefiind valabilă⁵⁴, unii autori, precum C.J. Date, îl preferă și recomandă la formularea subconsultărilor. Pe de altă parte însă, EXISTS este unul dintre cei mai dificili operatori. Logica sa se deosebește radical de cea a lui IN.

Utilizând operatorul IN, în tabela-resultat se extrag acele linii din tabela (tabelele) din fraza SELECT principală care satisfac o condiție aplicată liniilor din subconsultare. La utilizarea lui EXISTS se extrag linii tot ale tabelei/tabelelor frazei SELECT principală, dar pe baza existenței a cel puțin o linie în tabela/tabelele subconsultării, care satisfac (satisfac), pe lângă restricții proprii, și condiții formulate luându-se în considerare atribut/e din fraza principală (cum întotdeauna, nu-i aşa?).

Predicatale din clauza WHERE operează la nivel de linie, și nu la nivel de ansamblu (seturi de înregistrări), ceea ce, după Joe Celko, este o deviere de la spiritul relațional. Pe de altă parte, Sheryl Larsen⁵⁵ pledează pentru folosirea lui EXISTS în detrimentul IN-ului pe considerentul optimizării. IN presupune crearea unor tabele temporare ce reclamă uneori timp de execuție sensibil mai mare prin comparație cu logica tuplu-cu-tuplu EXISTS-enjală.

54. [Date86], p. 144.

55. [Larsen96].

Pentru comparație, începem cu exemple formilate în capitolul precedent la prezentarea operatorului IN.

Ce facturi au fost emise în aceeași zi cu factura 1120?

Varianta de interogare bazată pe operatorul EXISTS se prezintă astfel:

```
SELECT NrFact
FROM FACTURI F1
WHERE EXISTS
  (SELECT *
   FROM FACTURI F2
   WHERE F2.NrFact=1120 AND F1.DataFact=F2.DataFact)
```

Logica execuției se derulează după un scenariu top-bottom-top (sus-jos-sus) prezentat în figura 6.1.

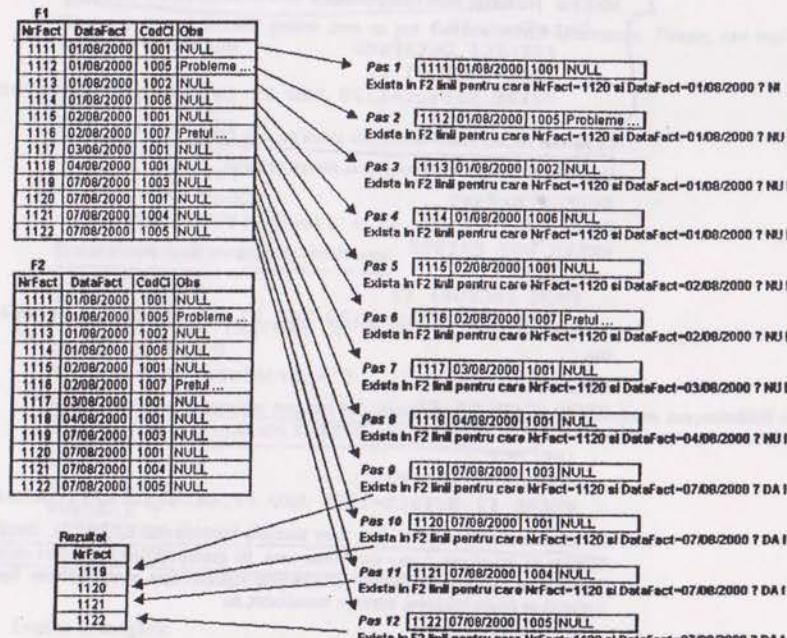


Figura 6.1. Logica operatorului EXISTS

Elementul de dificultate ţine de modalitatea de exprimare a condiției, care este una indirectă. Se parcurge fiecare linie din SELECT-ul principal, examinându-se, pe rând, dacă există măcar o linie în subconsultare care să satisfacă predicatul subconsultării. Dacă da, atunci nu aceasta se inserează în rezultat, ci linia din SELECT-ul principal! Nu știu cum e mai nimerit să-i zicem, lucru în echipă sau însușirea de către SELECT-ul principal a meritelor SELECT-ului „subaltern”.

```

WHERE PRODUSE.CodPr = LINIIFACT.CodPr AND
LINIIFACT.NrFact = FACTURI.NrFact AND
DenPr = 'Produs 2')

SELECT *
FROM ZILE_PRODUS1
INTERSECT
SELECT *
FROM ZILE_PRODUS2

```

De această dată au fost definite două expresii-tabele, ZILE_PRODUS1 și ZILE_PRODUS2. Prima conține zilele în care s-a vândut produsul 1, iar a doua – zilele în care s-a vândut cel de-al doilea produs. Fraza SELECT principală îndeplinește o simplă formalitate – intersecția celor două tabele ad-hoc.

Care sunt clientii pentru care există cel puțin câte o factură emisă în fiecare zi? (exemplul 22 din algebra relațională – figura 2.27)

Și diviziunea relațională se simplifică serios în noile condiții.

```

WITH
CLIENTI_NRZILE AS
  (SELECT DenCl, COUNT(DISTINCT DataFact) AS NrZile
   FROM CLIENTI C INNER JOIN FACTURI F ON C.CodCl=F.CodCl
   GROUP BY DenCl),
NRZILE AS
  (SELECT COUNT(DISTINCT DataFact) AS NrZile
   FROM FACTURI)
SELECT DenCl, NrZile
FROM CLIENTI_NRZILE
WHERE NrZile IN
  (SELECT NrZile
   FROM NRZILE)

```

După cum se observă, expresiile-tabele pot fi obținute și prin consultări ce conțin GROUP BY, iar pe de altă parte, în fraza SELECT principală pot fi formulate subconsultări în care apar, pe orice nivel, atribute din expresiile-tabele.

Care este judeful în care berea s-a vândut cel mai bine?

```

WITH
JUDETE_BERE1 AS
  (SELECT Judet, SUM(Cantitate * PretUnit * (1+ProctVA))
   AS Vinzari_Bere
   FROM JUDETE J, LOCALITATI L, CLIENTI C, FACTURI F,
   LINIIFACT LF, PRODUSE P
   WHERE J.Jud=L.Jud AND L.CodPost=C.CodPost AND
   C.CodCl=F.CodCl AND F.NrFact=LF.NrFact AND
   LF.CodPr=P.CodPr AND Grupa='Bere'
   GROUP BY Judet),
BERE_JUDETE AS
  (SELECT SUM(Cantitate * PretUnit * (1+ProctVA))
   AS Vinzari_Bere
   FROM JUDETE_BERE1
   GROUP BY Judet)

```

```

FROM JUDETE J, LOCALITATI L, CLIENTI C, FACTURI F,
LINIIFACT LF, PRODUSE P
WHERE J.Jud=L.Jud AND L.CodPost=C.CodPost AND
C.CodCl=F.CodCl AND F.NrFact=LF.NrFact AND
LF.CodPr=P.CodPr AND Grupa='Bere'
GROUP BY Judet)
SELECT Judet, Vinzari_Bere
FROM JUDETE_BERE1
WHERE Vinzari_Bere >= ALL
  (SELECT Vinzari_Bere
   FROM BERE_JUDETE)

```

Care este clientul cu cel mai mare rest de plată?

Revenim la una dintre cele mai dificile probleme de până acum, de data aceasta cu o soluție „aperișantă”.

```

WITH
FACTURAT AS
  (SELECT CodCl, SUM(Cantitate * PretUnit * (1+ProctVA))
   AS Facturat
   FROM FACTURI F, LINIIFACT LF, PRODUSE P
   WHERE F.NrFact=LF.NrFact AND LF.CodPr=P.CodPr
   GROUP BY CodCl),
INCASAT AS
  (SELECT CodCl, SUM(Transa) AS Incasat
   FROM FACTURI F, INCASFACT I
   WHERE F.NrFact=I.NrFact
   GROUP BY CodCl)
SELECT DenCl, Facturat, VALUE(Incasat,0) AS Incasat,
      Facturat - VALUE(Incasat,0) AS De_Incasat
   FROM CLIENTI INNER JOIN FACTURAT ON CLIENTI.CodCl=FACTURAT.CodCl
   LEFT OUTER JOIN INCASAT ON FACTURAT.CodCl=INCASAT.CodCl
  WHERE Facturat - VALUE(Incasat,0) >= ALL
    (SELECT Facturat - VALUE(Incasat,0)
     FROM FACTURAT LEFT OUTER JOIN INCASAT
     ON FACTURAT.CodCl=INCASAT.CodCl)

```

Rezultatul este prezentat în figura 5.44.

DENCL	CODCL	FACTURAT	INCASAT	DE_INCASAT
Client 3 SRL	1003	7242935	0	7242935
Client 5 SA	1006	6786570	0	6786570
Client 4	1004	5438300	0	5438300

Figura 5.44. Primii trei datornici

Expresii-tabele în DB2

Varianta de rezolvare a problemei formulate mai sus funcționează aproape identic și în DB2, sevența de comenzi din listingul 5.11 producând un rezultat identic celui din figura 5.43.

Listing 5.11. Script DB2 pentru extragerea primilor 3 datornici

```

DROP VIEW vOrdonare ;
DROP VIEW vDe_Incasat ;
DROP VIEW vIncasat ;
DROP VIEW vFacturat ;

-- Valorile facturate, pe clienti
CREATE VIEW vFacturat AS
    SELECT CodCl, SUM(Cantitate * PretUnit * (1+ProcTVA))
        AS Facturat
    FROM FACTURI F, LINIIFACT LF, PRODUSE P
    WHERE F.NrFact=LF.NrFact AND LF.CodPr=P.CodPr
    GROUP BY CodCl ;

-- Valorile incasate, pe clienti
CREATE VIEW vIncasat AS
    SELECT CodCl, SUM(Transa) AS Incasat
    FROM FACTURI F, INCASFACT I
    WHERE F.NrFact=I.NrFact
    GROUP BY CodCl ;

-- Restul de plata, pe clienti, in ordinea (descrescatoare) a
valorii
CREATE VIEW vDe_INCASAT AS
    SELECT DenCl, vFACTURAT.CodCl, Facturat, VALUE(Incasat,0)
        AS Incasat,
        Facturat - VALUE(Incasat,0) AS De_Incasat
    FROM vFACTURAT INNER JOIN CLIENTI
        ON vFacturat.CodCl=CLIENTI.CodCl
    LEFT OUTER JOIN vINCASAT ON vFACTURAT.CodCl=vINCASAT.CodCl ;

-- Primii trei datornici
CREATE VIEW vORDONARE AS
    SELECT *
    FROM vDe_INCASAT
    WHERE De_Incasat >=
        (SELECT MAX(De_Incasat)
        FROM vDe_INCASAT
        WHERE De_Incasat <

```

```

        (SELECT MAX(De_Incasat)
        FROM vDe_INCASAT
        WHERE De_Incasat <
              (SELECT MAX(De_Incasat)
        FROM vDe_INCASAT)
        )
    ;
-- Ordonare finala
SELECT *
FROM vOrdonare
ORDER BY De_Incasat DESC ;

```

DB2 prezintă însă și un ingredient interesant de rezolvare a interogărilor mai complexe, expresiile-tabele, care sunt apropiate de „filosofia” cursoarelor din VFP, deși, spre deosebire de acestea, nu pot fi închise și deschise explicit, iar „viața” lor se reduce numai la interogarea care le utilizează.

Spre deosebire de scripturile prezентate în acest paragraf, interogările bazate pe expresii-tabele sunt privite cu mult mai multă indulgență în comunitatea SQL, aceasta deoarece soluția se reduce la o singură frază SELECT principală, precedată de consultările pentru definirea expresiilor-tabele, subconsultări care se consideră a face parte integrantă din interogare.

Revenim (pentru a treia oară) la exemplul 19 din algebra relațională: *Ce facturi au fost emise în aceeași zi cu factura 1120?*

Jată și soluția DB2 bazată pe expresii-tabele:

```

WITH
Zi_1120AS
  (SELECT DataFact
   FROM FACTURI
   WHERE NrFact=1120)

SELECT NrFact
  FROM FACTURI, Zi_1120
 WHERE FACTURI.DataFact=Zi_1120.DataFact

```

ZI_1120 este o expresie-tabelă cu o singură coloană – DataFact – și o singură linie care conține date întocmirei facturii 1120, fiind folosită în fraza SELECT principală ca o tabelă obișnuită.

În ce zile s-au vândut și produsul cu denumirea „Produs 1”, și cel cu denumirea „Produs 2”?
(exemplul 17 – algebra relațională)

```

WITH
ZILE_PRODUS1 AS
  (SELECT DISTINCT DataFact
  FROM PRODUSE, LINIIFACT, FACTURI
  WHERE PRODUSE.CodPr = LINIIFACT.CodPr AND
        LINIIFACT.Nrfact = FACTURI.NrFact AND
        DenPr = 'Produs 1'),
ZILE_PRODUS2 AS
  (SELECT DISTINCT DataFact
  FROM PRODUSE, LINIIFACT, FACTURI

```

Care este județul în care berea s-a vândut cel mai bine?

După cum se observă, am selectat numai exemplele de importanță majoră pentru țară. Răspunsul la această tulburătoare problemă este oferit de programul din listing 5.5.

Listing 5.5. Județul în care berea s-a vândut cel mai bine

```
SELECT Judet, SUM(Cantitate * PretUnit * (1+ProcTVA)) ;
      AS Vinzari_Bere ;
FROM JUDETE J, LOCALITATI L, CLIENTI C, FACTURI F, ;
      LINIIFACT LF, PRODUSE P ;
INTO CURSOR cBere ;
WHERE J.Jud=L.Jud AND L.CodPost=C.CodPost AND C.CodCl=F.CodCl ;
      AND F.NrFact=LF.NrFact AND LF.CodPr=P.CodPr AND Grupa='Bere' ;
GROUP BY Judet
SELECT Judet, Vinzari_Bere ;
FROM cBere ;
WHERE Vinzari_Bere >= ALL ;
      (SELECT DISTINCT Vinzari_Bere ;
      FROM cBere)
```

Care este județul cu vânzări imediat superioare județului Neamț?

Soluția din listingul 5.6 are un dram suplimentar de interes, deoarece la ultima sa interogare, in afara clauzei WHERE care conține două subconsultări, se efectuează theta-joncțiunea dintre cele două cursoare, cVinzari_Neamt și cVinzari_Judete.

Listing 5.6. Județul cu vânzări imediat peste cele ale județului Neamț

```
* Vinzarile jud. Neamț
SELECT SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari ;
FROM JUDETE J ;
      INNER JOIN LOCALITATI L ON J.Jud=L.Jud ;
      INNER JOIN CLIENTI C ON L.CodPost=C.CodPost ;
      INNER JOIN FACTURI F ON C.CodCl=F.CodCl ;
      INNER JOIN LINIIFACT LF ON F.NrFact=LF.NrFact ;
      INNER JOIN PRODUSE P ON LF.CodPr=P.CodPr ;
INTO CURSOR cVinzari_Neamt ;
WHERE Judet='Neamt'
* Vinzarile fiecarui județ
SELECT Judet, SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari ;
FROM JUDETE J ;
      INNER JOIN LOCALITATI L ON J.Jud=L.Jud ;
      INNER JOIN CLIENTI C ON L.CodPost=C.CodPost ;
      INNER JOIN FACTURI F ON C.CodCl=F.CodCl ;
      INNER JOIN LINIIFACT LF ON F.NrFact=LF.NrFact ;
      INNER JOIN PRODUSE P ON LF.CodPr=P.CodPr ;
INTO CURSOR cVinzari_Judete ;
GROUP BY Judet
SELECT Judet, Vinzari ;
FROM cVinzari_Judete ;
WHERE Vinzari <= ALL ;
      (SELECT cVinzari_Judete.Vinzari ;
      FROM cVinzari_Judete, cVinzari_Neamt ;
```

```
WHERE cVinzari_Judete.Vinzari > cVinzari_Neamt.Vinzari) ;
AND Vinzari > ALL ;
      (SELECT Vinzari ;
      FROM cVinzari_Neamt)
```

Care este clientul cel mai mare datornic?

Este de-a dreptul reconfortant că nu trebuie să mai apelăm la artificiile pentru calculul valorilor facturate și incasate – vezi listingul 5.7.

Listing 5.7. Clientul cu cel mai mare rest de plată

```
* Valorile facturate, pe clienti
SELECT CodCl, SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Facturat ;
FROM FACTURI F ;
      INNER JOIN LINIIFACT LF ON F.NrFact=LF.NrFact ;
      INNER JOIN PRODUSE P ON LF.CodPr=P.CodPr ;
INTO CURSOR cFACTURAT ;
GROUP BY CodCl
* Valorile incasate, pe clienti
SELECT CodCl, SUM(Transa) AS Incasat ;
FROM FACTURI F ;
      INNER JOIN INCASFACT I ON F.NrFact=I.NrFact ;
INTO CURSOR cINCASAT ;
GROUP BY CodCl
* Restul de plată, pe clienti
SELECT cFACTURAT.CodCl, Facturat, NVL(Incasat,0) AS Incasat, ;
      Facturat - NVL(Incasat,0) AS De_Incasat ;
FROM cFACTURAT LEFT OUTER JOIN cINCASAT ;
      ON cFACTURAT.CodCl=cINCASAT.CodCl ;
INTO CURSOR cDe_INCASAT ;
* Finalul apoteotic
SELECT DenCl, cDe_INCASAT.* ;
FROM cDe_INCASAT INNER JOIN CLIENTI ;
      ON cDe_INCASAT.CodCl=CLIENTI.CodCl ;
WHERE De_Incasat >= ALL ;
      (SELECT De_Incasat ;
      FROM cDe_INCASAT)
```

Căror clienti li s-a trimis măcar o factură în toate zilele de vânzare?

Nici problemele ce impun soluții de tip diviziune relațională nu mai constituie o dificultate urmând logica interogărilor independente – listingul 5.8.

Listing 5.8. Clientul căruia i s-a trimis măcar o factură în toate zilele de vânzare

```
SELECT CodCl, COUNT(DISTINCT DataFact) AS Cite_Zile ;
FROM FACTURI ;
INTO CURSOR cl ;
GROUP BY CodCl
SELECT DenCl ;
FROM CLIENTI INNER JOIN cl ON CLIENTI.CodCl=cl.CodCl ;
WHERE Cite_Zile = ANY ;
      (SELECT COUNT(DISTINCT DataFact) ;
      FROM FACTURI)
```

au leac în VFP salvând rezultatul unei interogări în tabele temporare (cursoare), tabele derivate sau chiar tabele obișnuite care devin „materie primă” pentru alte fraze SELECT și.a.m.d. Prin dispunerea succesivă a interogărilor se redactează programe (.PRG) lansate în execuție atunci când se dorește informația respectivă sau când se obține un raport.

Uneori însă, și în SGBD-urile din lumea foarte bună este necesară salvarea rezultatelor intermediare în cadrul aceleiași interogări sau pentru prelucrări ulterioare. De obicei, tabelele derivate constituie un suport universal pentru asemenea gen de operațiuni. Sunt însă și soluții mai la indemână. Ceva mai înainte evocăm cursoarele VFP. În DB2, tabelele temporare îmbracă forma expresiilor-tabele care sunt tabele temporare a căror definiție este păstrată numai pe parcursul „vieții” interogării în care este inclusă. Prin urmare, nu pot fi folosite pentru „pasarea” rezultatelor intermediare între SELECT-uri independente.

Fraze SELECT independente în VFP

Ne vom focaliza atenția mai întâi pe VFP, pentru că aici avem de recuperat câteva handicapuri: un singur nivel de subconsultare, subconsultări în clauza HAVING și subconsultări în FROM.

Să se afișeze, pentru fiecare factură, dacă este fără nici o incasare, incasată parțial sau incasată total (și cele trei valori: facturată, incasată și rămasă de incasat)?

Am ales pentru început una dintre primele probleme care crează necazuri în VFP în varianta SQL curată, adică neprocedurală. Listingul 5.2 conține o soluție mult mai simplă alcătuită din trei fraze SQL independente. Primele două obțin cursoarele utilizate în a treia. Cursorul este, cel puțin pentru exemplele noastre în VFP, o tabelă temporară: poate fi închisă explicit sau la ieșirea din VFP; la închidere se șterge, deci după utilizare nu mai ocupă spațiu pe disc.

Listing 5.2. Situația incasării fiecărei facturi

```
* cursorul cVINZARI contine valoarea fiecărei facturi
SELECT NrFact, SUM(Cantitate * PretUnit * (1+ProctVA)) ,
      AS Facturat ;
FROM LINIIFACT LF INNER JOIN PRODUSE P ON LF.CodPr = P.CodPr ;
INTO CURSOR cVINZARI ;
GROUP BY NrFact

* CINCASARI contine valoarea
* incasata pentru fiecare factura
SELECT NrFact, SUM(Transa) AS Incasat ,
      FROM INCASFACT ;
INTO CURSOR CINCASARI ;
GROUP BY NrFact

* se jonctioneaza (extern) cele două cursoare
SELECT V.NrFact, Facturat, NVL(Incasat,0) AS Incasat, ,
      Facturat - NVL(Incasat,0) AS Diferenta, ,
      IIF(Facturat=NVL(Icasat,0), 'INCASATA TOTAL ', ,
           IIF(Facturat > NVL(Icasat,0) AND NVL(Icasat,0) > 0, ,
                'Incasata parțial ', 'Fara nici o incasare')) ;
      AS Situație ;
FROM cVINZARI V LEFT OUTER JOIN CINCASARI I ON V.NrFact=I.NrFact
```

Reproșul principal adus acestei soluții VFP este... proceduralitatea, în vădită contradicție cu filosofia SQL – curat neprocedurală. Trecând peste supărare, ajungem și la principalul atu: se oferă, totuși, răspunsul corect la problema formulată.

Care sunt clienții pentru care există cel puțin căte o factură emisă în fiecare zi?

Varianta procedurală este prezentată în listingul 5.3.

Listing 5.3. Clienții pentru care există cel puțin căte o factură emisă în fiecare zi

```
* Produsul cartezian al valorilor DenCl și DataFact
SELECT DISTINCT DenCl, DataFact ;
FROM CLIENTI, FACTURI ;
INTO CURSOR cPRODUS_CARTEZIAN

* Clientii și zilele în care s-au emis facturi pentru ei
SELECT DISTINCT DenCl, DataFact ;
FROM CLIENTI INNER JOIN FACTURI ON CLIENTI.CodCl=FACTURI.CodCl ;
INTO CURSOR cR1

* Valorile DenCl care nu se regăsesc combinate cu toate valorile DataFact
SELECT DISTINCT DenCl ;
FROM cProdus_Cartezian ;
INTO CURSOR cNu_I ;
WHERE DenCl+DTOC(DataFact) NOT IN (
      SELECT DenCl+DTOC(DataFact) ,
             FROM cR1).

* în fine, rezultatul
SELECT DISTINCT DenCl ;
FROM cR1 ;
WHERE DenCl NOT IN (
      SELECT DenCl ,
             FROM cNu_I)
Continuăm cu alte exemple care au fost, până acum, frunzante pentru fox-iști, cele în care în clauza HAVING apare o subconsultare.
```

Care este ziua în care s-au emis cele mai multe facturi?

Practic, în prezentul caz, ca și în celelalte de acest tip, prin cursoare, grupurile sunt transformate în tupluri și, în loc de HAVING, se va folosi WHERE.

Listing 5.4. Ziua (zilele) în care s-au emis cele mai multe facturi?

```
SELECT DataFact, COUNT(*) AS Nr ;
FROM FACTURI ;
INTO CURSOR cNrFact_Zilnic ;
GROUP BY DataFact

SELECT DataFact, Nr ;
FROM cNrFact_Zilnic ;
WHERE Nr IN (
      SELECT MAX(Nr) ,
             FROM cNrFact_Zilnic)
```

DENCL	VALOARE_FACTURATA	VALOARE_INCASATA	REST_DE_INCASAT
Client 3 SRL	7242935.00	0	7242935.00

Figura 5.42. Clientul ce are cel mai mare rest de plată (DB2)

Varianta Oracle pentru obținerea aceluiași rezultat este:

```

SELECT DenCl,
       TO_NUMBER( SUBSTR ( TO_CHAR (
           SUM(DISTINCT 10000000000000000 *
              LF.NrFact+10000000000000000 *
              LF.Linie+Cantitate * PretUnit * (1+ProcTVA)),
           '99999999999999999999999999999999'), 14,15) )
    AS Valoare_Facturata,
       SUM (DECODE (LF.Linie, 1, NVL(Transa,0), NULL))
    AS Incasari,
       TO_NUMBER( SUBSTR ( TO_CHAR (
           SUM(DISTINCT 10000000000000000 *
              LF.NrFact+10000000000000000 *
              LF.Linie+Cantitate * PretUnit * (1+ProcTVA)),
           '99999999999999999999999999999999'), 14,15) ) -
       SUM (DECODE (LF.Linie, 1, NVL(Transa,0), NULL))
    AS De_Incasat
  FROM PRODUSE P, LINIIFACT LF, FACTURI F,
        CLIENTI C, INCASFACT I
 WHERE P.CodPr = LF.CodPr AND LF.NrFact = F.NrFact AND
       F.CodCl = C.CodCl AND F.Nrfact=I.NrFact (+)
 GROUP BY DenCl
 HAVING
       TO_NUMBER( SUBSTR ( TO_CHAR (
           SUM(DISTINCT 10000000000000000 *
              LF.NrFact+10000000000000000 *
              LF.Linie+Cantitate * PretUnit * (1+ProcTVA)),
           '99999999999999999999999999999999'), 14,15) ) -
       SUM (DECODE (LF.Linie, 1, NVL(Transa,0), NULL)) >=
 ALL
       (SELECT TO_NUMBER( SUBSTR ( TO_CHAR (
           SUM(DISTINCT 10000000000000000 *
              LF.NrFact+10000000000000000 *
              LF.Linie+Cantitate * PretUnit * (1+ProcTVA)),
           '99999999999999999999999999999999'), 14,15) ) -
       SUM (DECODE (LF.Linie, 1, NVL(Transa,0), NULL))
  FROM PRODUSE P, LINIIFACT LF, FACTURI F,
        CLIENTI C, INCASFACT I
 WHERE P.CodPr = LF.CodPr AND LF.NrFact = F.NrFact AND
       F.CodCl = C.CodCl AND F.Nrfact=I.NrFact (+)
 GROUP BY DenCl)

```

Din nou despre diviziunea relațională

Revenind la cazul teoretic din capitolul 2 (figura 2.56), cătul diviziunii relaționale a R1: R2 (altfel spus, găsirea tuturor „icșilor” care sunt în R1 în combinație cu toți „igrecii” din R2) se calculează în SQL și astfel:

```

SELECT X
  FROM R1
 GROUP BY X
 HAVING COUNT(Y) = (SELECT COUNT(Y)
                      FROM R2)

```

Cărora clienti li s-a trimis măcar o factură în toate zilele cu vânzări?

```

SELECT DenCl
  FROM FACTURI F, CLIENTI C
 WHERE F.CodCl=C.CodCl
 GROUP BY DenCl
 HAVING COUNT(DISTINCT DataFact) =
      (SELECT COUNT (DISTINCT DataFact)
        FROM FACTURI)

```

Modest ca volum (nu și ca importanță), răspunsul se găsește în figura 5.43.

DENCL	VALOARE
Client 1 SRL	0

Figura 5.43. Clientul pentru care există cel puțin o factură în fiecare zi

Ce produse au fost vândute tuturor clienților?

```

SELECT DenPr
  FROM PRODUSE P, LINIIFACT LF, FACTURI F
 WHERE P.CodPr=LF.CodPr AND LF.NrFact=F.NrFact
 GROUP BY DenPr
 HAVING COUNT(DISTINCT CodCl) =
      (SELECT COUNT (CodCl)
        FROM CLIENTI)

```

Răspunsul este *Produs 2*.

5.7. Secvențializarea interogărilor

Cu tot atașamentul pentru Visual FoxPro, în materie de SQL (și nu numai) există un decalaj serios față de standardul SQL-92 și față de SGBD-urile profesionale: (IBM) DB2, Oracle, Informix, Sybase, (Microsoft) SQL Server etc.

Aceasta este veste rea. Vesteau bună și că mai toate interogările dificile, ce nu pot fi formulate printr-o singură frază SELECT principală (plus subconsultări pe un singur nivel),

DENCL	VALOARE_FACTURATA
Client 1 SRL	10052000000012490240.00
Client 2 SA	1114000000001180250.00
Client 3 SRL	448600000007242935.00
Client 4	224500000005438300.00
Client 5 SRL	222700000001337560.00
Client 6 SA	334800000006786570.00
Client 7 SRL	111700000001383375.00

Figura 5.40. Împrovizare pentru calculul valorii vânzărilor pe clienți (DB2)

Expresia de calcul a valorii facturate introduce, pe lângă Cantitate, PretUnit și ProcTVA, și NrFact și Linie. Ultimile două atribute sunt înmulțite cu constante foarte mari (1 biliard, respectiv 10 bilioane), suficient de mari pentru ca valoarea propriu-zisă a facturilor să nu fie „afectată”. Este sigur că valorile obținute pentru fiecare client au fost calculate luându-se în calcul o singură dată fiecare linie dintr-o factură. Iar dacă privim în rezultat partea din dreapta fiecărui număr (după cele sase-șapte zerouri), observăm că aceleia sunt chiar valorile facturate care ne interesează. Prin urmare, nu ne mai rămâne decât să extragem acea parte:

```
SELECT DenCl,
       CAST (
          SUBSTR (
             CAST (
                SUM (
                   DISTINCT 1000000000000000 * LF.NrFact +
                   1000000000000000 * LF.Linie +
                   Cantitate * PretUnit * (1+ProcTVA)
                )
               AS CHAR(35)
            ), 17, 19
         )
      AS DECIMAL(19,2)
   )
  AS Valoare_Facturata
FROM PRODUSE P
INNER JOIN LINIIFACT LF ON P.CodPr = LF.CodPr
INNER JOIN FACTURI F ON LF.NrFact = F.NrFact
INNER JOIN CLIENTI C ON F.CodCl = C.CodCl
LEFT OUTER JOIN INCASFACT I ON F.NrFact=I.NrFact
GROUP BY DenCl
```

Obținem ceea ce dorim, adică valoarea vânzărilor pe clienți, după cum o arată figura 5.41.

DENCL	VALOARE_FACTURATA
Client 1 SRL	12490240.00
Client 2 SA	1180250.00
Client 3 SRL	7242935.00
Client 4	5438300.00
Client 5 SRL	1337560.00
Client 6 SA	6786570.00
Client 7 SRL	1383375.00

Figura 5.41. Valoarea vânzărilor, pe clienți (DB2)

În final, soluția SQL-92/DB2 la problema pusă (și răspunsul în figura 5.42):

```
SELECT DenCl,
       CAST (SUBSTR (CAST (SUM (
          DISTINCT 1000000000000000 * LF.NrFact +
          1000000000000000 * LF.Linie +
          Cantitate * PretUnit * (1+ProcTVA) )
       AS CHAR(35) ), 17, 19 ) AS DECIMAL(19,2) )
  AS Valoare_Facturata,
       SUM (CASE WHEN LF.Linie > 1 THEN NULL ELSE
              VALUE(Transa,0) END) AS Valoare_Incasata,
       CAST (SUBSTR (CAST (SUM (
          DISTINCT 1000000000000000 * LF.NrFact +
          1000000000000000 * LF.Linie +
          Cantitate * PretUnit * (1+ProcTVA) )
       AS CHAR(35) ), 17, 19 ) AS DECIMAL(19,2) ) -
       SUM (CASE WHEN LF.Linie > 1 THEN NULL ELSE
              VALUE(Transa,0) END)
  AS Rest_De_Incasat
FROM PRODUSE P
INNER JOIN LINIIFACT LF ON P.CodPr = LF.CodPr
INNER JOIN FACTURI F ON LF.NrFact = F.NrFact
INNER JOIN CLIENTI C ON F.CodCl = C.CodCl
LEFT OUTER JOIN INCASFACT I ON F.NrFact=I.NrFact
GROUP BY DenCl
HAVING
       CAST (SUBSTR (CAST (SUM (
          DISTINCT 1000000000000000 * LF.NrFact +
          1000000000000000 * LF.Linie +
          Cantitate * PretUnit * (1+ProcTVA) )
       AS CHAR(35) ), 17, 19 ) AS DECIMAL(19,2) ) -
       SUM (CASE WHEN LF.Linie > 1 THEN NULL ELSE
              VALUE(Transa,0) END)
=> ALL
(SELECT CAST (SUBSTR (CAST (SUM (
          DISTINCT 1000000000000000 * LF.NrFact +
          1000000000000000 * LF.Linie +
          Cantitate * PretUnit * (1+ProcTVA) )
       AS CHAR(35) ), 17, 19 ) AS DECIMAL(19,2) ) -
       SUM (CASE WHEN LF.Linie > 1 THEN NULL ELSE
              VALUE(Transa,0) END)
FROM PRODUSE P
INNER JOIN LINIIFACT LF ON P.CodPr = LF.CodPr
INNER JOIN FACTURI F ON LF.NrFact = F.NrFact
INNER JOIN CLIENTI C ON F.CodCl = C.CodCl
LEFT OUTER JOIN INCASFACT I ON F.NrFact=I.NrFact
GROUP BY DenCl)
```

Care este județul cu vânzări imediat superioare județului Neamț?

```

SELECT Judet, SUM(Cantitate * PretUnit * (1+ProcTVA))
      AS Vanzari
  FROM JUDETE J, LOCALITATI L, CLIENTI C, FACTURI F,
  LINIIFACT LF, PRODUSE P
 WHERE J.Jud=L.Jud AND L.CodPost=C.CodPost AND
   C.CodCl=F.CodCl AND F.NrFact=LF.NrFact
   AND LF.CodPr=P.CodPr
 GROUP BY Judet
 HAVING SUM(Cantitate * PretUnit * (1+ProcTVA)) <= ALL
    (SELECT SUM(Cantitate * PretUnit * (1+ProcTVA))
     FROM JUDETE J, LOCALITATI L, CLIENTI C, FACTURI F,
     LINIIFACT LF, PRODUSE P
    WHERE J.Jud=L.Jud AND L.CodPost=C.CodPost AND
      C.CodCl=F.CodCl AND F.NrFact=LF.NrFact AND
      LF.CodPr=P.CodPr
 GROUP BY Judet
 HAVING SUM(Cantitate * PretUnit * (1+ProcTVA)) >
    (SELECT SUM(Cantitate * PretUnit * (1+ProcTVA))
     FROM JUDETE J, LOCALITATI L, CLIENTI C,
     FACTURI F, LINIIFACT LF, PRODUSE P
    WHERE J.Jud=L.Jud AND L.CodPost=C.CodPost AND
      C.CodCl=F.CodCl AND F.NrFact=LF.NrFact
      AND LF.CodPr=P.CodPr
      AND Judet='Neamt'))
 AND
  SUM(Cantitate * PretUnit * (1+ProcTVA)) >
  (SELECT SUM(Cantitate * PretUnit * (1+ProcTVA))
   FROM JUDETE J, LOCALITATI L, CLIENTI C, FACTURI F,
   LINIIFACT LF, PRODUSE P
  WHERE J.Jud=L.Jud AND L.CodPost=C.CodPost AND
    C.CodCl=F.CodCl AND F.NrFact=LF.NrFact AND
    LF.CodPr=P.CodPr AND Judet='Neamt')

```

JUDET	VINZARI
Vestul 7242935	

Figura 5.38. Județul cu vânzări imediat superioare județului Neamț

Care este clientul cel mai mare datornic?

Este, cred, cea mai grea interogare de până acum. După cum am văzut atunci când am calculat pentru fiecare factură valorile facturate și incasate, la joncțiunea externă a „părții de facturare” cu „partea de incasare”, fiecare tranșă de incasare se repetă în funcție de numărul de linii ce compun factura respectivă, iar fiecare linie de factură se repetă în funcție de numărul de tranșe de incasare. Trucul utilizat pentru a scoate la capăt constă în împărțirea totalului tranșelor de incasare la numărul liniilor din factură, obținând astfel valoarea incasată a facturii și, pe de altă parte, împărțirea totalului valorilor liniilor din facturi la numărul tranșelor de incasare. Pentru problema de față trebuie găsită o altă soluție, deoarece gruparea nu o facem la nivel de factură, ci la nivel de client. Numărul de linii din facturi și numărul de tranșe nu mai au nici o relevanță la gruparea după client.

Pentru incasări, ar fi o idee. Deoarece o tranșă se repetă pentru fiecare linie a facturii, putem elmina din SUM toate tranșele pentru care Linie > 1. Încercăm ceva în DB2:

```

SELECT DenCl AS Client,
      SUM (
        CASE WHEN LF.Linie > 1
              THEN NULL ELSE VALUE(Transa,0)
          END) AS Valoare_Incasata
  FROM PRODUSE P
 INNER JOIN LINIIFACT LF ON P.CodPr = LF.CodPr
 INNER JOIN FACTURI F ON LF.NrFact = F.NrFact
 INNER JOIN CLIENTI C ON F.CodCl = C.CodCl
 LEFT OUTER JOIN INCASFACT I ON F.NrFact=I.NrFact
 GROUP BY DenCl

```

Rezultatul din figura 5.39 ne încreștează că ideea n-a fost chiar rea.

CLIENT	VALOARE_INCASATA
Client 1 SRL	10504432
Client 2 SA	1160250
Client 3 SRL	0
Client 4	0
Client 5 SRL	487705
Client 6 SA	0
Client 7 SRL	0

Figura 5.39. Incasările pe clienți (DB2)

Cu valoarea facturată însă, lucrurile sunt mai complicate. Fiecare linie din factură se repetă de atâtea ori, în funcție de căte tranșe de incasare există pentru factură respectivă. Cum gruparea se face pe clienți, numărul tranșelor este irrelevant. Trebuie însumate valorile *Distincte* ale facturii și liniei. Este adevărat, clauza DISTINCT poate fi folosită și cu SUM. Dar este aproape sigur că vom avea la un moment dat, două linii, în aceeași factură, cu aceeași valoare. E musai de însumat valorile expresiei pentru combinații distincte (NrFact, Linie). Confirmând vocația de popor de improvizatori (n-am zis cărpați, să nu supăr adevărății patrioți), recurgem la un truc ieftin, dar, vorba dñui Graur (comentatorul), născut!

```

SELECT DenCl,
      SUM( DISTINCT
            1000000000000000 * LF.NrFact +
            1000000000000000 * LF.Linie +
            Cantitate * PretUnit * (1+ProcTVA))
           AS Valoare_Facturata
  FROM PRODUSE P
 INNER JOIN LINIIFACT LF ON P.CodPr = LF.CodPr
 INNER JOIN FACTURI F ON LF.NrFact = F.NrFact
 INNER JOIN CLIENTI C ON F.CodCl = C.CodCl
 LEFT OUTER JOIN INCASFACT I ON F.NrFact=I.NrFact
 GROUP BY DenCl

```

Nu putem discuta detalii înainte de a vedea ce a putut fi obținut din interogare – figura 5.40.

Care este județul în care berea s-a vândut cel mai bine?

În tabela PRODUSE există un atribut care reprezintă grupa în care se încadrează produsul respectiv. Berea este una dintre grupele îndrăgite.

```
SELECT Judet, SUM(Cantitate * PretUnit * (1+ProcTVA)) AS
      Vinzari_Bere
  FROM JUDETE J, LOCALITATI L, CLIENTI C, FACTURI F,
  LINIIFACT LF, PRODUSE P
 WHERE J.Jud=L.Jud AND L.CodPost=C.CodPost AND C.CodCl=F.CodCl
   AND F.NrFact=LF.NrFact AND LF.CodPr=P.CodPr
   AND Grupa='Bere'
 GROUP BY Judet
 HAVING SUM(Cantitate * PretUnit * (1+ProcTVA))
    >= ALL
  (SELECT SUM(Cantitate * PretUnit * (1+ProcTVA))
  FROM JUDETE J, LOCALITATI L, CLIENTI C, FACTURI F,
  LINIIFACT LF, PRODUSE P
 WHERE J.Jud=L.Jud AND L.CodPost=C.CodPost AND
   C.CodCl=F.CodCl AND F.NrFact=LF.NrFact AND
   LF.CodPr=P.CodPr AND Grupa='Bere'
 GROUP BY Judet)
```

JUDET	VINZARI_BERE
Iasi	7546940

Figura 6.35. Județul cu cea mai bună vânzare a produselor din grupa Bere

Să nu uităm soluția funcțională în VFP, cuceritoare prin simplitate:

```
SELECT TOP 1 Judet, SUM(Cantitate * PretUnit * (1+ProcTVA)) \
  AS Vinzari_Bere ;
  FROM JUDETE J INNER JOIN LOCALITATI L ON J.Jud=L.Jud ;
  INNER JOIN CLIENTI C ON L.CodPost=C.CodPost ;
  INNER JOIN FACTURI F ON C.CodCl=F.CodCl ;
  INNER JOIN LINIIFACT LF ON F.NrFact=LF.NrFact ;
  INNER JOIN PRODUSE P ON LF.CodPr=P.CodPr ;
 WHERE Grupa='Bere' ;
 GROUP BY Judet ;
 ORDER BY Vinzari_Bere DESC
```

Care sunt clienții cu valoarea vânzărilor peste medie?

```
SELECT DenCl AS Client,
       SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari
  FROM LINIIFACT LF, PRODUSE P, FACTURI F, CLIENTI C
 WHERE P.CodPr = LF.CodPr AND LF.NrFact = F.NrFact
   AND F.CodCl=C.CodCl
 GROUP BY DenCl
 HAVING SUM(Cantitate * PretUnit * (1+ProcTVA)) >=
  (SELECT SUM(Cantitate * PretUnit * (1+ProcTVA)) /
```

```
COUNT(DISTINCT F.CodCl)
  FROM LINIIFACT LF, PRODUSE P, FACTURI F
 WHERE P.CodPr = LF.CodPr AND LF.NrFact = F.NrFact)
```

CLIENT	VINZARI
Client 1 SRL	12490240
Client 3 SRL	7242935
Client 4	5438300
Client 6 SA	6786570

Figura 5.36. Clienți cu vânzări peste medie

Extragerea factură cu valoarea imediat peste cea medie.

```
SELECT F.NrFact, SUM(Cantitate * PretUnit * (1+ProcTVA)) \
  AS Valoare
  FROM LINIIFACT LF, PRODUSE P, FACTURI F
 WHERE P.CodPr = LF.CodPr AND LF.NrFact = F.NrFact
 GROUP BY F.NrFact
 HAVING
  SUM(Cantitate * PretUnit * (1+ProcTVA)) <= ALL
  (SELECT SUM(Cantitate * PretUnit * (1+ProcTVA)) /
  COUNT(DISTINCT F.NrFact)
  FROM LINIIFACT LF, PRODUSE P, FACTURI F
  WHERE P.CodPr = LF.CodPr AND LF.NrFact = F.NrFact)
 AND
  SUM(Cantitate * PretUnit * (1+ProcTVA)) >
  (SELECT SUM(Cantitate * PretUnit * (1+ProcTVA)) /
  COUNT(DISTINCT F.NrFact)
  FROM LINIIFACT LF, PRODUSE P, FACTURI F
  WHERE P.CodPr = LF.CodPr AND LF.NrFact = F.NrFact)
```

NRFAC	VALOARE
1111	5399625

Figura 5.37. Factură cu cea mai mică valoare peste medie

Elementul de noutate al acestei interogări este includerea, într-o consultare ce prezintă clauza HAVING, a unei alte subconsultări în care apare, de asemenea, HAVING.

Pentru VFP nu am nici o variantă la îndemâna.

```

GROUP BY DenCl
HAVING COUNT(DISTINCT CodPr) >= ALL
    (SELECT COUNT(DISTINCT CodPr)
     FROM FACTURI F, LINIIFACT LF
     WHERE F.NrFact= LF.NrFact
     GROUP BY CodCl)

```

DENCL	CITEPRODUSE
Client 3 SRL	4

Figura 5.33. Clientul care a cumpărat cele mai multe produse

- alta „Oracle only” (COUNT in MAX):

```

SELECT DenCl, COUNT(DISTINCT CodPr) AS CiteProduse
FROM CLIENTI C, FACTURI F, LINIIFACT LF
WHERE C.CodCl=F.CodCl AND F.NrFact= LF.NrFact
GROUP BY DenCl
HAVING COUNT(DISTINCT CodPr) =
    (SELECT MAX(COUNT(DISTINCT CodPr))
     FROM FACTURI F, LINIIFACT LF
     WHERE F.NrFact= LF.NrFact
     GROUP BY CodCl)

```

- și una VFP (clauza TOP):

```

SELECT TOP 1 DenCl, COUNT(DISTINCT CodPr) AS CiteProduse ;
FROM CLIENTI C INNER JOIN FACTURI F ON C.CodCl=F.CodCl ;
    INNER JOIN LINIIFACT LF ON F.NrFact= LF.NrFact ;
GROUP BY DenCl ;
ORDER BY CiteProduse DESC

```

Care este comportamentul cu cea mai bună medie a salariilor tarifare?

Se exclude din discuție comportamentul DIRECȚIUNE în care apare, singur și ferice, directorul general.

```

SELECT Compart, AVG(SalTarifar) AS Medie_Sal
FROM PERSONAL2
WHERE Compart <> 'DIRECȚIUNE'
GROUP BY Compart
HAVING AVG(SalTarifar) >= ALL
    (SELECT AVG(SalTarifar)
     FROM PERSONAL2
     WHERE Compart <> 'DIRECȚIUNE'
     GROUP BY Compart)

```

COMPART	MEDIE_SAL
RESURSE UMANE	5500000

Figura 5.34. Compartimentul cu cea mai bună medie a salariilor tarifare

Pentru aducerea aminte a structurilor alternative, putem folosi și variantele:

- SQL-92/DB2:

```

SELECT Compart, AVG (CASE
    WHEN Compart <> 'DIRECȚIUNE'
        THEN SalTarifar
    ELSE 0
END) AS Medie_Sal
FROM PERSONAL2
GROUP BY Compart
HAVING AVG (CASE
    WHEN Compart <> 'DIRECȚIUNE'
        THEN SalTarifar
    ELSE 0
END) >= ALL
(SELECT AVG (CASE
    WHEN Compart <> 'DIRECȚIUNE'
        THEN SalTarifar
    ELSE 0
END)
FROM PERSONAL2
GROUP BY Compart)

```

- și Oracle:

```

SELECT Compart, ROUND(AVG (
    DECODE (Compart, 'DIRECȚIUNE', 0, SalTarifar)
),0) AS Medie_Sal
FROM PERSONAL2
GROUP BY Compart
HAVING
    AVG (DECODE (Compart, 'DIRECȚIUNE', 0, SalTarifar))
>= ALL
    (SELECT AVG (DECODE (Compart, 'DIRECȚIUNE', 0,
        SalTarifar))
     FROM PERSONAL2
     GROUP BY Compart)

```

Functia ROUND a fost necesară pentru afișarea numai a părții întregi din medie. De data aceasta (și următoarele) trecem peste varianta Oracle cu funcție inclusă în altă funcție.

- În fine, varianta VFP:

```

SELECT TOP 1 Compart, AVG(SalTarifar) AS Medie_Sal ;
FROM PERSONAL2 ;
WHERE Compart <> 'DIRECȚIUNE' ;
GROUP BY Compart ;
ORDER BY Medie_Sal DESC

```

```

SELECT DataFact, NrFact AS UltimaFactura
FROM FACTURI
WHERE NrFact IN
    (SELECT MAX(NrFact)
     FROM FACTURI)

sau

SELECT DataFact, NrFact AS UltimaFactura
FROM FACTURI
WHERE NrFact =
    (SELECT MAX(NrFact)
     FROM FACTURI)

sau

SELECT DataFact, NrFact AS UltimaFactura
FROM FACTURI
WHERE NrFact = ANY
    (SELECT MAX(NrFact)
     FROM FACTURI)

sau

SELECT DataFact, NrFact AS UltimaFactura
FROM FACTURI
WHERE NrFact = ALL
    (SELECT MAX(NrFact)
     FROM FACTURI)

Fără funcția MAX, soluția este:

SELECT DataFact, NrFact AS UltimaFactura
FROM FACTURI
WHERE NrFact >= ALL
    (SELECT NrFact
     FROM FACTURI)

```

5.6. Subconsultări în clauza HAVING

Predicalele incluse în clauza HAVING ale interogărilor de până acum compară o expresie cu o constantă. În cele ce urmează vom discuta despre includerea în clauza HAVING a subconsultărilor. Din păcate, deoarece această facilitate nu a fost încă implementată (până la versiunea VFP6) în nucleul SQL al acestui produs, trimiterile la Visual FoxPro vor fi mai lapidare în acest paragraf.

Revenim asupra unei probleme formulate în ultima parte a paragrafului 4.6:

Care sunt zilele în care s-au emis mai multe facturi decât pe 2 august 2000?

Se poate formula o soluție bazată pe subconsultări în clauza HAVING (prezenta este redactată în DB2 – în Oracle este necesară funcția de conversie TO_DATE):

```

SELECT DataFact AS Zi, COUNT(NrFact) AS Nr_Facturilor
FROM FACTURI
GROUP BY DataFact
HAVING COUNT(NrFact) >
    (SELECT COUNT(NrFact)
     FROM FACTURI
     WHERE DataFact = '08/02/2000')

```

Care este ziua în care s-au emis cele mai multe facturi?

```

SELECT DataFact, COUNT(*) AS Nr_Facturilor
FROM FACTURI
GROUP BY DataFact
HAVING COUNT(*) >= ALL
    (SELECT COUNT(*)
     FROM FACTURI
     GROUP BY DataFact)

```

Subconsultarea calculează numărul de facturi corespunzător fiecărei zile. Predicatul clauzei HAVING compară numărul de facturi al fiecărei zile cu toate valorile extrase de subconsultare. Se obține tabela din figura 5.32.

DATAFACT	NR_FACTURILOR
01-AUG-00	4
07-AUG-00	4

Figura 5.32. Zilele în care s-au emis cele mai multe facturi

Se cunosc de adăugat că Oracle mai permite și o soluție bazată pe inclusiunea unei funcții în alta:

```

SELECT DataFact, COUNT(*) AS Nr_Facturilor
FROM FACTURI
GROUP BY DataFact
HAVING COUNT(*) =
    (SELECT MAX(COUNT(*))
     FROM FACTURI
     GROUP BY DataFact)

```

Am fi nedrept cu VFP dacă nu am prezentat o soluție care mie, unuia, îmi place pentru simplitate. Iată-o:

```

SELECT TOP 1 DataFact, COUNT(*) AS Nr ;
FROM FACTURI ;
GROUP BY DataFact ;
ORDER BY Nr DESC

```

Surprinzător pentru unii, soluția funcționează. ORDER BY se aplică grupurilor, iar ordonarea împreună cu opțiunea TOP extrag rezultatul corect.

Care este clientul care a cumpărat cele mai multe produse?

- soluție SQL-92/DB2/Oracle:

```

SELECT DenCl, COUNT(DISTINCT CodPr) AS CiteProduse
FROM CLIENTI C, FACTURI F, LINIIIFACT LF
WHERE C.CodCl=F.CodCl AND F.NrFact= LF.NrFact

```

Liniile sunt ordonate crescător după prețul unitar, iar în rezultate sunt extrase primele cinci (prin TOP 5).

5.5. Operatorii ALL, SOME, ANY

Cei trei operatori prezentați în acest paragraf sunt grozav de utili în interogările cu iz „cantitativist” mai pronunțat, deoarece permit utilizarea unui predicat de comparație care este aplicat rezultatului unei subconsultări, predicat bazat pe unul dintre operatorii: =, <, >, <=, >=, < > sau #.

Dacă, în cea mai mare parte a cazurilor de până acum, se compara un atribut (sau rezultatul unei expresii/funcții) cu o constantă, ALL, SOME și ANY permit compararea valorii atributului/funcției/expresiei cu un set de tupluri (ansamblu de linii) extras printr-o subconsultare.

Care sunt produsele vândute la prețuri unitare superioare oricărui preț unitar la care a fost vândut „Produs 1”?

Soluția acestei probleme este valabilă în SQL-92 și în toate cele trei SGBD-uri:

```
SELECT DISTINCT DenPr
FROM PRODUSE P, LINIIFACT LF
WHERE P.CodPr=LF.CodPr AND PretUnit > ALL
  (SELECT DISTINCT PretUnit
   FROM PRODUSE P, LINIIFACT LF
   WHERE P.CodPr=LF.CodPr AND DenPr ='Produs 1')
```

Ca orice interogare pe două niveluri, operațiile se derulează în doi pași. Mai întâi se execută subconsultarea și se obține o tabelă intermediară în care se găsesc toate prețurile unitare la care a fost vândut, în decursul istoriei, *Produs 1* – vezi figura 5.28.

PRETUNIT
9300
8500
10000

Figura 5.28. Rezultatul subconsultării – prețurile unitare pentru „Produs 1”

Cum operatorul de conexiune a frazei SELECT principale cu subconsultarea este > ALL, din joncțiunea tabelelor PRODUSE și LINIIFACT vor fi extrase numai liniile care au valoarea atributului PretUnit mai mare decât toate valorile din figura 5.28. Așa încât rezultatul final se prezintă ca în figura 5.29.

DENPR
Produs 2
Produs 4

Figura 5.29. Produse cu cel puțin un preț unitar superior oricărui preț unitar al „Produsului 1”

Care sunt produsele vândute la prețuri unitare superioare măcar unui preț unitar al „Produsului 1”?

Este genul de situații în care se folosește SOME sau ANY (au aceeași funcție).

```
SELECT DISTINCT DenPr
FROM PRODUSE P, LINIIFACT LF
WHERE P.CodPr=LF.CodPr AND PretUnit > ANY
  (SELECT DISTINCT PretUnit
   FROM PRODUSE P, LINIIFACT LF
   WHERE P.CodPr=LF.CodPr AND DenPr = 'Produs 1')
```

Rezultatul este cel din figura 5.30, deoarece, spre deosebire de ALL, și ANY (și SOME) selectează liniile pentru care prețul unitar este măcar mai mare decât una dintre valorile obținute prin subconsultare.

DENPR
Produs 1
Produs 2
Produs 4

Figura 5.30. Produse cu cel puțin un preț unitar superior măcar unui preț unitar al „Produsului 1”

Operatorul =ANY este echivalent cu IN.

Câți alți angajați au salariul tarifar egal cu al ANGAJAT 2?

Soluția:

```
SELECT COUNT(*) - 1 AS Nr
FROM PERSONAL2
WHERE Saltarifar IN
  (SELECT Saltarifar
   FROM PERSONAL2
   WHERE NumePren='ANGAJAT 2')
```

este echivalentă cu:

```
SELECT COUNT(*) - 1 AS Nr
FROM PERSONAL2
WHERE Saltarifar =ANY
  (SELECT Saltarifar
   FROM PERSONAL2
   WHERE NumePren='ANGAJAT 2')
```

Folosirea unuia din cei trei operatori nu este obligatorie atunci când subconsultarea conține o funcție-agregat (ce întoarce o valoare dintr-un ansamblu de tupluri) – MIN, MAX, COUNT, SUM, AVG.

Care este ultima factură întocmită (factura cea mai recentă) și data la care a fost emisă?

Oricare dintre variantele următoare funcționează și obține valorile din figura 5.31.

DATAFACT	ULTIMAFACTURA
07-08-2000	1122

Figura 5.31. Ultima factură și ziua în care a fost întocmită

SITUATIUNE	NR.
Incasate TOTAL	4
Incasale partial	2
Nelincasate deloc	5

Figura 5.26. Numărul facturilor fără nici o tranșă de incasare, - Incasate parțial și incasate total

Până acum subconsultările au fost conectate la fraza SELECT superioară exclusiv prin operatorul IN. În paragraful următor vom vedea că pentru (sub)interrogările comparative pot fi întrebuițiați ALL, SOME, ANY. Atunci când rezultatul unei subconsultări se concretizează într-o tabelă cu o singură coloană și o singură linie, corelarea poate fi făcută cu operatorii de comparație obișnuiți: =, >, >=, <, <=. Vom ilustra această facilitate prin câteva exemple.

Care este cel mai mare preț unitar la care s-a vândut un produs?

```
SELECT MAX(PretUnit)  
FROM LINIIFACT
```

Care este cel mai mare preț unitar și care este produsul, precum și factura unde se înregistrează respectivul preț maxim?

```
SELECT NrFact, DenPr, FretUnit  
FROM LINIIFACT LF, PRODUSE P  
WHERE LF.CodPr=P.CodPr AND FretUnit =  
(SELECT MAX(FretUnit)  
FROM LINIIFACT)
```

Care sunt cele mai mari două prețuri unitare de vânzare, care sunt produsele și facturile pentru care se înregistrează respectivile prețuri maxime?

```

SELECT NrFact, DenPr, PretUnit
FROM LINIIFACT LF, PRODUSE P
WHERE LF.CodPr=P.CodPr AND PretUnit >=
      (SELECT MAX(PretUnit)
       FROM LINIIFACT
       WHERE PretUnit <
             (SELECT MAX(PretUnit)
              FROM LINIIFACT))

```

Pentru a înțelege mecanismul interogării de mai sus, pornim de la SELECT-ul „cel mai de jos”. SELECT MAX(PretUnit) FROM LINIIFACT extrage prețul unitar maxim din tabela LINIIFACT. Subconsultarea superioară, (SELECT MAX(PretUnit) FROM LINIIFACT WHERE PretUnit < (...ultima subconsultare...)), determină al doilea preț unitar din LINIIFACT. SELECT-ul principal afișează toate prețurile unitare mai mari sau egale cu penultimul.

Care sunt cele mai mari cinci prețuri unitare de vânzare, produsele și facturile în care apar cele cinci prețuri maxime?

Aici voi am, de fapt, să ajunge

```

SELECT NrFact, DenPr, PretUnit
FROM LINIIFACT INNER JOIN PRODUSE
    ON LINIIFACT.CodPr=PRODUSE.CodPr
WHERE PretUnit >
    (SELECT MAX(PretUnit)
     FROM LINIIFACT
     WHERE PretUnit <
         (SELECT MAX(PretUnit)
          FROM LINIIFACT
          WHERE PretUnit <
              (SELECT MAX(PretUnit)
               FROM LINIIFACT
               WHERE PretUnit <
                   (SELECT MAX(PretUnit)
                    FROM LINIIFACT
                    WHERE PretUnit <
                        (SELECT MAX(PretUnit)
                         FROM LINIIFACT
                         WHERE PretUnit <
                             (SELECT MAX(PretUnit)
                              FROM LINIIFACT)
                          )
                      )
                  )
              )
          ORDER BY PretUnit DESC

```

Celor care nu au reușit să fie impresionați de această ultimă variantă, le sugerez să încearcă cu primele 10, 20 și.m.d. prețuri unitare. Revenim însă la soluția prezentată; iată rezultatul acesteia (figura 5.27).

NRFAC	DENPR	PRETUNG
1114	Produs 4	15800
1119	Produs 4	14000
1120	Produs 2	11200
1118	Produs 2	11000
1119	Produs 2	10900

Figura 5.27. Cele mai mari cinci preturi unitare

Firește, la atâtă amar de niveluri de interogare, VFP capotează. Avem însă la indemână o soluție neverosimil de simplă, bazată de clauza TOP:

```
SELECT TOP 5 NrFact, DenPr, PretUnit ;
FROM LINIIFACT INNER JOIN PRODUSE
    ON LINIIFACT.CodPr=PRODUSE.CodPr ;
ORDER BY PretUnit DESC
```

```
WHERE C.CodCl=F1.CodCl AND
(C.CodCl, F2.DataFact) NOT IN
(SELECT C.CodCl, DataFact
FROM CLIENTI C, FACTURI F
WHERE C.CodCl=F.CodCl))
```

• Soluția VFP:

```
SELECT DISTINCT DenCl ;
FROM CLIENTI ;
INNER JOIN FACTURI ON CLIENTI.CodCl=FACTURI.CodCl ;
WHERE STR(CLIENTI.CodCl,6)+STR(0,6) NOT IN ;
(SELECT STR(F1.CodCl,6)+STR(NVL(F3.CodCl,0),6) ;
FROM FACTURI F1 INNER JOIN FACTURI F2 ON l=1 ;
LEFT OUTER JOIN FACTURI F3 ON ;
F1.CodCl=F3.CodCl AND F2.DataFact=F3.DataFact)
```

In ce zile s-au vândut și produsul cu denumirea „Produs 1”, și cel cu denumirea „Produs 2”?

Este exemplul 23 din același capitol 2.

• Soluția DB2 (și Oracle, înlocuind EXCEPT cu MINUS):

```
SELECT DataFact
FROM FACTURI F, LINIIFACT LF, PRODUSE P
WHERE F.NrFact=LF.NrFact AND LF.CodPr=P.CodPr
AND DenPr IN ('Produs 1', 'Produs 2')
EXCEPT
SELECT DISTINCT F.DataFact
FROM FACTURI F, LINIIFACT LF, PRODUSE P1, PRODUSE P2
WHERE F.NrFact=LF.NrFact AND LF.CodPr=P1.CodPr
AND P1.DenPr IN ('Produs 1', 'Produs 2')
AND P2.DenPr IN ('Produs 1', 'Produs 2')
AND (F.DataFact, P2.CodPr) NOT IN
(SELECT DISTINCT DataFact, LF.CodPr
FROM FACTURI F, LINIIFACT LF, PRODUSE P
WHERE F.NrFact=LF.NrFact AND LF.CodPr=P.CodPr
AND DenPr IN ('Produs 1', 'Produs 2'))
```

• Soluția VFP:

```
SELECT DISTINCT DataFact ;
FROM FACTURI ;
INNER JOIN LINIIFACT ON FACTURI.NrFact=LINIIFACT.NrFact;
INNER JOIN PRODUSE ON LINIIFACT.CodPr=PRODUSE.CodPr AND;
DenPr IN ('Produs 1','Produs 2') ;
WHERE DTOC(DataFact)+DTOC(//) NOT IN ;
(SELECT DISTINCT DTOC(F1.DataFact)+;
DTOC(NVL(F2.DataFact,(//)));
FROM FACTURI F1 INNER JOIN PRODUSE P1 ;
ON P1.DenPr IN ('Produs 1','Produs 2') ;
LEFT OUTER JOIN (LINIIFACT LF2 INNER JOIN FACTURI F2 :
```

```
ON LF2.NrFact=F2.NrFact) ;
ON F1.DataFact=F2.DataFact AND
P1.CodPr=LF2.CodPr)
```

Prezenta soluție mi-a preluat descoperirea unei facilități pe care altii, probabil, o știau de mult. Și în VFP este posibilă jonctionarea unei tabele cu rezultatul unei alte joncții, precedența joncționărilor fiind indicată cu ajutorul parantezelor. Fără această opțiune, lucrurile ar fi avut un aer mult mai grav.

Încheiem paragraful dedicat operatorului IN cu un exemplu mai puțin „colțuros”, dar suficient de interesant.

Să se afișeze căte facturi sunt: neincasate deloc, incasate parțial și incasate total?

Soluția reuneste facturile incasate total cu facturile incasate parțial și cu facturile neincasate deloc.

```
SELECT 'Incasate TOTAL' AS Situație, COUNT(*) AS Nr
FROM FACTURI
WHERE NrFact IN
(SELECT F.NrFact
FROM LINIIFACT LF, PRODUSE P, FACTURI F, INCASFACT I
WHERE LF.CodPr = P.CodPr AND LF.NrFact = F.NrFact
AND F.NrFact=I.NrFact
GROUP BY F.NrFact
HAVING SUM(Cantitate * PretUnit * (1+ProcTVA)) /
COUNT(DISTINCT I.CodInc) =
SUM(Transa) / MAX(LF.Linie))
UNION
SELECT 'Incasate parțial', COUNT(*)
FROM FACTURI
WHERE NrFact IN
(SELECT F.NrFact
FROM LINIIFACT LF, PRODUSE P, FACTURI F, INCASFACT I
WHERE LF.CodPr = P.CodPr AND LF.NrFact = F.NrFact
AND F.NrFact=I.NrFact
GROUP BY F.NrFact
HAVING SUM(Cantitate * PretUnit * (1+ProcTVA)) /
COUNT(DISTINCT I.CodInc) >
SUM(Transa) / MAX(LF.Linie))
UNION
SELECT 'Neincasate deloc', COUNT(*)
FROM FACTURI
WHERE
NrFact IN
(SELECT NrFact
FROM LINIIFACT)
AND
NrFact NOT IN
(SELECT NrFact
FROM INCASFACT)
```

```

WHERE PERSONAL2.Marca=S2.Marca AND 2000=S2.An AND 5= S2.Luna
AND PERSONAL2.Marca NOT IN
  (SELECT Marca
   FROM SPORURI
   WHERE An=2000 AND Luna=4)
AND PERSONAL2.Marca NOT IN
  (SELECT Marca
   FROM SPORURI
   WHERE An=2000 AND Luna=6)
UNION
  SELECT PERSONAL2.Marca, NumePren,
        0,
        0,
        S3.SporNoapte,
        S3.SporNoapte
   FROM PERSONAL2, SPORURI S3
  WHERE PERSONAL2.Marca=S3.Marca
    AND 2000=S3.An AND 6= S3.Luna
    AND PERSONAL2.Marca NOT IN
      (SELECT Marca
       FROM SPORURI
       WHERE An=2000 AND Luna=4)
    AND PERSONAL2.Marca NOT IN
      (SELECT Marca
       FROM SPORURI
       WHERE An=2000 AND Luna=5)
UNION
  SELECT PERSONAL2.Marca, NumePren,
        0,
        0,
        0,
        0
   FROM PERSONAL2
  WHERE Marca NOT IN
    (SELECT Marca
     FROM SPORURI
     WHERE An=2000 AND Luna=4)
    AND Marca NOT IN
      (SELECT Marca
       FROM SPORURI
       WHERE An=2000 AND Luna=5)
    AND Marca NOT IN
      (SELECT Marca
       FROM SPORURI
       WHERE An=2000 AND Luna=6)
ORDER BY NumePren

```

Cred că singura măngâiere este că am scris, de voie, de nevoie, cea mai lungă frază SELECT de până acum...

Tot cu ajutorul operatorului IN (și NOT IN) se poate aborda și „problema” *diviziunii* relaționale în SQL (parcă vă aud spunând: „Asta ne mai lipsea acum!”). Succesiunea de pași prezentată în figura 2.28 se realizează relativ simplu prin soluția SQL-92/DB2:

```

SELECT X
  FROM R1
EXCEPT
  SELECT DISTINCT R1.X
  FROM R1, R2
  WHERE (R1.X, R2.Y) NOT IN
    (SELECT X, Y
     FROM R1)

```

În Oracle este necesară înlocuirea operatorului EXCEPT cu MINUS. Cu VFP e o poveste mai lungă. Cum aici nu există nici EXCEPT, nici MINUS, sau ceva similar, am încercat varianta:

```

SELECT * ;
  FROM R1, R2
  WHERE R1.X+R2.Y NOT IN ;
    (SELECT X+Y ;
     FROM R1)

```

Mesajul primit a fost unul curat, împede, tonic și explicit ca al unui funcționar de la ghicsele Oficiilor Forțelor de Muncă: SQL: Queries of this type are not supported (Error 1814). Ce-i de făcut? Salvarea vine tot de la joncțiunea externă:

```

SELECT DISTINCT X ;
  FROM R1 ;
  WHERE X+' ' NOT IN ;
    (SELECT R1_1.X + NVL(R1_2.X, ' ') ;
     FROM R1 R1_1 ;
     INNER JOIN R2 R2_1 ON 1=1 ;
     LEFT OUTER JOIN R1 R1_2 ;
     ON R1_1.X=R1_2.X AND R2_1.Y=R1_2.Y)

```

Pseudojoncțiunea internă este de fapt un produs cartezian, deoarece condiția de joncțiune este 1=1. Subconsultarea extrage „îcșii” care au „goluri”, iar fraza SELECT principală efectuează scăderea lor din „îcșii” tabelui R1.

Aflați clienții pentru care există cel puțin câte o factură emisă în fiecare zi.

Acesta este exemplul 22 din algebra relațională (figura 2.27) pentru ilustrarea operatorului diviziune. Urmărm logica pe care tocmai am prezentat-o.

- Soluția DB2 (și Oracle, cu MINUS-ul de rigoare):

```

SELECT DenC1
  FROM CLIENTI
 WHERE CodC1 IN
  (SELECT CodC1
   FROM CLIENTI C
EXCEPT
  SELECT DISTINCT C.CodC1
   FROM CLIENTI C, FACTURI F1, FACTURI F2

```

```

    UNION
    SELECT F. NrFact,
           SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Facturat,
           0 AS Incasat
      FROM LINIIFACT LF, PRODUSE P, FACTURI F
     WHERE LF.CodPr = P.CodPr AND LF.NrFact = F.NrFact
       AND F.NrFact NOT IN
         (SELECT NrFact
            FROM INCASFACT)
    GROUP BY F.NrFact
  
```

Să se obțină sporurile de noapte pentru al doilea trimestru al anului 2000, atât lunar, cât și cumulat.
Trebuie reunite persoanele care au sporul de noapte pe toate cele trei luni cu persoanele care prezintă sporul numai pe căte două luni și cu persoanele cu sporul pe o singură lună.
Dacă la momentul formulării soluției anterioare afirmam că fraza SELECT este supraponderală, prezenta soluție este pur și simplu pantagruelică. Cred că fraza următoare (care funcționează în DB2 și Oracle) demonstrează fără dubii că jocurile externe este o găseală grozav de inteligență și, pe de altă parte, că scrierea frazelor SQL poate deveni, pe alocuri, o treabă înfricoșătoare.

```

    SELECT PERSONAL2.Marcă, NumePren,
           S1.SporNoapte AS Spor_Noapte_Aprilie,
           S2.SporNoapte AS Spor_Noapte_Mai,
           S3.SporNoapte AS Spor_Noapte_Iunie,
           S1.SporNoapte + S2.SporNoapte + S3.SporNoapte
              AS Spor_Noapte_Trim_II
      FROM PERSONAL2, SPORURI S1, SPORURI S2, SPORURI S3
     WHERE PERSONAL2.Marcă=S1.Marcă AND S1.An=2000 AND 4=S1.Luna
       AND PERSONAL2.Marcă=S2.Marcă AND S2.An=2000 AND
       5=S2.Luna AND PERSONAL2.Marcă=S3.Marcă AND
       S3.An=2000 AND 6 = S3.Luna
  
```

```

    UNION
    SELECT PERSONAL2.Marcă, NumePren,
           S1.SporNoapte,
           S2.SporNoapte,
           0 ,
           S1.SporNoapte + S2.SporNoapte
      FROM PERSONAL2, SPORURI S1, SPORURI S2
     WHERE PERSONAL2.Marcă=S1.Marcă AND
           S1.An=2000 AND 4 = S1.Luna AND
           PERSONAL2.Marcă=S2.Marcă AND
           S2.An=2000 AND 5 = S2.Luna AND
           PERSONAL2.Marcă NOT IN
             (SELECT Marca
                FROM SPORURI
               WHERE An=2000 AND Luna=6)
  
```

```

    UNION
    SELECT PERSONAL2.Marcă, NumePren,
           S1.SporNoapte,
           0 ,
  
```

```

           S3.SporNoapte,
           S1.SporNoapte + S3.SporNoapte
      FROM PERSONAL2, SPORURI S1, SPORURI S3
     WHERE PERSONAL2.Marcă=S1.Marcă AND S1.An=2000 AND
       4 = S1.Luna AND PERSONAL2.Marcă=S3.Marcă AND S3.An=2000
         AND 6 = S3.Luna
           AND PERSONAL2.Marcă NOT IN
             (SELECT Marca
                FROM SPORURI
               WHERE An=2000 AND Luna=5)
  
```

UNION

```

    SELECT PERSONAL2.Marcă, NumePren,
           0,
           S2.SporNoapte,
           S3.SporNoapte,
           S2.SporNoapte + S3.SporNoapte
      FROM PERSONAL2, SPORURI S2, SPORURI S3
     WHERE PERSONAL2.Marcă=S2.Marcă AND S2.An=2000 AND
       5 = S2.Luna AND PERSONAL2.Marcă=S3.Marcă AND
       S3.An=2000 AND 6 = S3.Luna
  
```

UNION

```

    SELECT PERSONAL2.Marcă, NumePren,
           S1.SporNoapte,
           0,
           0,
           S1.SporNoapte
      FROM PERSONAL2, SPORURI S1
     WHERE PERSONAL2.Marcă=S1.Marcă
       AND 2000=S1.An AND 4 = S1.Luna
       AND PERSONAL2.Marcă NOT IN
         (SELECT Marca
            FROM SPORURI
           WHERE An=2000 AND Luna=5)
           AND PERSONAL2.Marcă NOT IN
             (SELECT Marca
                FROM SPORURI
               WHERE An=2000 AND Luna=6)
  
```

UNION

```

    SELECT PERSONAL2.Marcă, NumePren,
           0,
           S2.SporNoapte,
           0,
           S2.SporNoapte
      FROM PERSONAL2, SPORURI S2
  
```

```
SELECT * ;
FROM R1 ;
WHERE STR(A, 4)+B+STR(C, 4) IN ;
  (SELECT STR(C, 4)+D+STR(E, 4) ;
   FROM R2)
```

Un alt exemplu de intersecție (exemplul 17 – algebra relațională):

In ce zile s-au vândut și produsul cu denumirea „Produs 1”, și cel cu denumirea „Produs 2”?

```
SELECT DISTINCT DataFact
FROM PRODUSE, LINIIFACT, FACTURI
WHERE PRODUSE.CodPr = LINIIFACT.CodPr AND
LINIIFACT.NrFact = FACTURI.NrFact AND
DenPr = 'Produs 1'
AND DataFact IN
  (SELECT DISTINCT DataFact
   FROM PRODUSE, LINIIFACT, FACTURI
   WHERE PRODUSE.CodPr = LINIIFACT.CodPr AND
LINIIFACT.NrFact = FACTURI.NrFact AND
DenPr = 'Produs 2')
```

Cât privește diferența relațională, cheia rezolvării este NOT IN. Diferența relațiilor R1 și R2, prezentată în figura 2.5, poate fi calculată în SQL astfel:

```
SELECT *
FROM R1
WHERE (A,B,C) NOT IN
  (SELECT C,D,E
   FROM R2)
```

Fără, în VFP vom aplica „truful” concatenării, ca și la intersecție. Apelăm iarăși la un exemplu din algebra relațională (exemplul 18):

Ce clienți au cumpărat și „Produs 2”, și „Produs 3”, dar nu au cumpărat „Produs 5”?

Soluția comună SQL-92/DB2/Oracle/Visual FoxPro este:

```
SELECT DISTINCT DenCl
FROM PRODUSE, LINIIFACT, FACTURI, CLIENTI
WHERE
  PRODUSE.CodPr = LINIIFACT.CodPr AND
  LINIIFACT.NrFact = FACTURI.NrFact AND
  FACTURI.CodCl = CLIENTI.CodCl AND
  DenPr = 'Produs 2'
  AND FACTURI.CodCl IN
    (SELECT CodCl
     FROM PRODUSE, LINIIFACT, FACTURI
     WHERE PRODUSE.CodPr = LINIIFACT.CodPr AND
LINIIFACT.NrFact = FACTURI.NrFact AND
DenPr = 'Produs 3')
  AND FACTURI.CodCl NOT IN
    (SELECT CodCl
```

```
FROM PRODUSE, LINIIFACT, FACTURI
WHERE PRODUSE.CodPr = LINIIFACT.CodPr AND
LINIIFACT.NrFact = FACTURI.NrFact AND
DenPr = 'Produs 5')
```

Până la apariția operatorilor LEFT OUTER JOIN, RIGHT OUTER JOIN și FULL OUTER JOIN, în multe SGBD-uri joncțiunea externă era realizată prin reunirea linilor obținute din echil-joncțiune cu linile unei tabele (completate cu zerouri/spății pentru atributele celeilalte tabele) ce nu au corespondent în cealaltă.

Cum joncțiunea externă a fost definită în capitolul 2, revenim la operațiunea ilustrată în figura 2.24. Joncțiunea externă la stânga a relațiilor R1 și R2 prin atributul C ar putea fi realizată și astfel:

```
SELECT *
FROM R1, R2
WHERE R1.C = R2.C
UNION
SELECT A,B,C, 0, ' ', 0
FROM R1
WHERE C NOT IN
  (SELECT C
   FROM R2)
```

Varianta de mai sus funcționează în toate cele trei SGBD-uri. La drept vorbind, soluția pe care am încercat-o prima dată și care ar fi corespuns pe deplin joncțiunii externe era:

```
SELECT *
FROM R1, R2
WHERE R1.C = R2.C
UNION
SELECT A,B,C, NULL, NULL, NULL
FROM R1
WHERE C NOT IN
  (SELECT C
   FROM R2)
```

însă toate cele trei SGBD-uri se încăpățânează să nu o execute.

Revenim la câteva exemple prezentate la joncțiunea externă pe care le rezolvăm prin noua „rețetă” – reunire/valori vide.

Care sunt valorile facturate și încasate ale fiecărei facturi?

Soluția de mai jos (valabilă deopotrivă în DB2, Oracle și VFP) reunește facturile care au măcar o tranșă de încasare cu cele neîncasate deloc.

```
SELECT F. NrFact, SUM(Cantitate * PretUnit * (1+ProcTVA)) /
COUNT(DISTINCT I.CodInc) AS Facturat,
SUM(Transa) / MAX(LF.Linie) AS Incasat
FROM LINIIFACT LF, PRODUSE P, FACTURI F, INCASFACT I
WHERE LF.CodPr = P.CodPr AND LF.NrFact = F.NrFact
  AND F.NrFact=I.NrFact
GROUP BY F.NrFact
```

Astfel, pentru a răspunde la întrebare, soluția de interogare trebuie reformulată:

```
SELECT DenCl ;
FROM CLIENTI INNER JOIN FACTURI ;
ON CLIENTI.CodCl=FACTURI.CodCl ;
WHERE DataFact IN ;
(SELECT DataFact ;
FROM FACTURI ;
WHERE NrFact=1120)
```

In ce județe s-a vândut produsul „Produs 2”?

Am ales acest exemplu pentru a „vântura”, prin subconsultări, cât mai multe tabele ale bazei (rezultatul – vezi figura 5.24):

```
SELECT Judet
FROM JUDETE
WHERE Jud IN
(SELECT Jud
FROM LOCALITATI
WHERE CodPost IN
(SELECT CodPost
FROM CLIENTI
WHERE CodCl IN
(SELECT CodCl
FROM FACTURI
WHERE NrFact IN
(SELECT NrFact
FROM LINIIFACT
WHERE CodPr IN
(SELECT CodPr
FROM PRODUSE
WHERE DenPr = 'Produs 2')))))
```

JUDET
Iasi
Neamț
Timiș
Vaslui

Figura 5.24. Județele în care s-a vândut „Produs 2”

Revenim la tabela PERSONAL2 din figura 5.2: *Căți subordonați direcți are ANGAJAT 2?*

La această problemă (la care răspunsul este 2) formulăm, pentru comparație, două soluții. Soluția bazată pe juncție este:

```
SELECT COUNT(*) AS NrSubordonati
FROM PERSONAL2 SUBORDONATI, PERSONAL2 SEFI
WHERE SUBORDONATI.MarcaSef=SEFI.Marca AND
SEFI.NumePren='ANGAJAT 2'
```

A două soluție utilizează o subconsultare:

```
SELECT COUNT(Marca) AS NrSubordonati
FROM PERSONAL2
WHERE MarcaSef IN
(SELECT Marca
FROM PERSONAL2
WHERE NumePren='ANGAJAT 2')
```

În capitolul 2 am cheltuit destul de puțin timp și spațiu pentru un tip de juncție nu prea folosit, semijonctiunea, prin care se extrag numai liniile dintr-o tabelă ce au corespondent (ca valoare a atributului de legătură) în a două tabele. Cu operatorul IN se pot formula lejer soluții ce corespund semijonctiunii. Astfel, o tabelă precum cea din figura 2.25 se obține prin:

```
SELECT *
FROM R1
WHERE C IN
(SELECT C
FROM R2)
```

Completăm discuția și cu exemplul „practic” luat pentru ilustrarea semijonctiunii (exemplul 21 din capitolul 2): *Care sunt localitățile (codul poștal, denumirea și indicativul județului) în care există măcar un client?*

```
SELECT *
FROM LOCALITATI
WHERE CodPost IN
(SELECT CodPost
FROM CLIENTI)
```

CODPOST	LOC	JUD
1900	Timisoara	TM
5550	Roman	NT
5725	Pecsenyi	IS
6500	Vaslui	VS
6600	Iasi	IS

Figura 5.25. Localități în care există măcar un client

Tot prin subconsultări putem realiza intersecția și diferența relațională. Raportându-ne la exemplul teoretic din capitolul 2 (figura 2.4), *intersecția celor două relații, R1 și R2, se poate obține și astfel:*

```
SELECT *
FROM R1
WHERE (A,B,C) IN
(SELECT C,D,E
FROM R2)
```

Varianta funcționează în DB2 și Oracle, nu însă și în VFP, deoarece într-o subconsultare nu pot fi testate simultan trei valori (ale atributelor A, B și C), ci numai una. Atfel incă, pentru a-i înșela vigilența, vom concatena cele trei atrbute, dând senzația că avem de-a face cu unul singur:

5.4. Subconsultări. Operatorul IN

Una din cele mai importante facilități ale SQL constă în includerea unei consultări în alta, pe două sau mai multe niveluri – altfel spus, utilizarea *subconsultărilor*. Prin subconsultări se obțin tabele temporare intermediare folosite ca „argumente” ale frazelor SELECT superioare.

Operatorul cel mai utilizat în materie de subconsultări este IN, pe care l-am întâlnit deja în capitolul precedent într-o cu totul altă ipostază – testarea încadrării valorii unui atribut într-o listă de constante. Pentru cele ce urmează, domeniul de testare este alcătuit din linile unei tabele obținute printr-o (sub)consultare.

Revenim (pentru a doua oară) la exemplul 19 din algebra relațională: *Ce facturi au fost emise în aceeași zi ca factura 1120?*

În capitolul anterior a fost formulată o soluție bazată pe joncțiunea a două instanțe ale tabelei FACTURI. Iată și o soluție mai simplă, bazată pe subconsultări:

```
SELECT NrFact
FROM FACTURI
WHERE DataFact IN
    (SELECT DataFact
     FROM FACTURI
     WHERE NrFact=1120)
```

se obține data eliberării facturii 1120

Execuția acestei interogări se derulează în doi timpi. Mai întâi se execută subconsultarea SELECT DataFact FROM FACTURI WHERE NrFact=1120, obținându-se o tabelă intermediară cu o singură linie (pe care apare 17-08-2000) și o singură coloană (DataFact), ca în figura 5.22. În al doilea pas sunt selectate linile talei FACTURI care îndeplinesc condiția DataFact = '17-08-2000'.

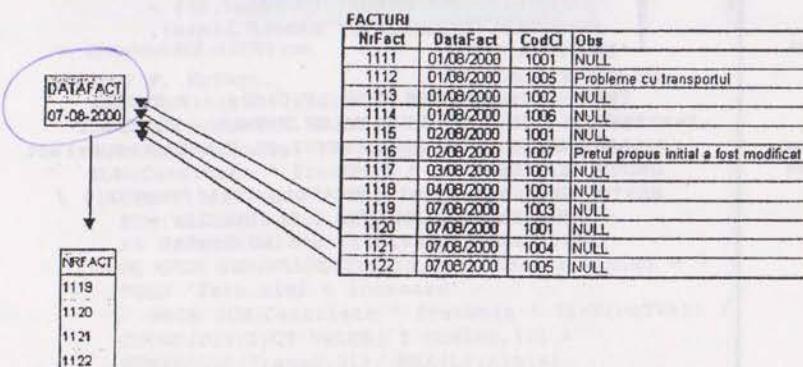


Figura 5.22. Schema execuției frazei SELECT ce prezintă o subconsultare

În rezultat a fost inclusă și factura de referință – 1120. Dacă se dorește excluderea acesteia, fraza SELECT se modifică astfel:

```
SELECT NrFact
FROM FACTURI
WHERE DataFact IN
    (SELECT DataFact
     FROM FACTURI
     WHERE NrFact=1120)
AND NrFact <> 1120
```

Ce facturi au fost emise în alte zile decât factura 1120?

Acum exemplul necesită folosirea operatorului de negație:

```
SELECT NrFact
FROM FACTURI
WHERE DataFact NOT IN
    (SELECT DataFact
     FROM FACTURI
     WHERE NrFact=1120)
```

Care sunt clienții căror li s-au trimis facturi în aceeași zi în care a fost întocmită factura 1120?

```
SELECT DenCl
FROM CLIENTI
WHERE CodCl IN
    (SELECT CodCl
     FROM FACTURI
     WHERE DataFact IN
         (SELECT DataFact
          FROM FACTURI
          WHERE NrFact=1120))
```

Rezultatul prezentat în figura 5.23 se obține folosind trei niveluri de interogare (faza principală, o subconsultare și o sub-subconsultare).

DENCL
Client 1 SRL
Client 3 SRL
Client 4
Client 5 SRL

Figura 5.23. Clienții pentru care există facturi emise în aceeași zi ca 1120

Dacă în DB2 și Oracle această ultimă interogare este executată fără probleme, în VFP „facem cunoștință” cu una dintre cele mai serioare limitări SQL – maximum două niveluri de consultare (faza principală și o interogare subordonată). Mesajul de eroare are numărul 1842: SQL: Subquery nesting is too deep.

se întâlnește de un obstacol pe care l-am cunoscut în capitolul precedent: operatorul DISTINCT nu poate fi întrebuințat decât o singură dată pe consultare/subconsultare. O variantă de lucru ar fi sevența următoare, dar rezultatul este unul mult mai puțin spectaculos, după cum se observă în figura 5.21.

```
SELECT F. NrFact, ;
    IIF(SUM(NVL(Transa,0))/ MAX(LF.Linie)=0, ;
        'Fara nici o incasare', ;
        IIF(SUM(Cantitate * PretUnit * (1+ProcTVA)) / ;
            COUNT(DISTINCT NVL(I.CodInc,1)) > ;
            SUM(NVL(Transa,0))/ MAX(LF.Linie), ;
            'Incasata parțial', 'INCASATA TOTAL ' )) ;
    AS Situație ;
FROM LINIIFACT LF ;
INNER JOIN PRODUSE P ON LF.CodPr = P.CodPr ;
INNER JOIN FACTURI F ON LF.NrFact = F.NrFact ;
LEFT OUTER JOIN INCASFACT I ON F.NrFact=I.NrFact ;
GROUP BY F.NrFact
```

NrFact	Situație
1111	INCASATA TOTAL
1112	Incasata parțial
1113	INCASATA TOTAL
1114	Fara nici o incasare
1115	Fara nici o incasare
1116	Fara nici o incasare
1117	INCASATA TOTAL
1118	INCASATA TOTAL
1119	Fara nici o incasare
1120	Incasata parțial
1121	Fara nici o incasare

Figura 5.21. Situația incasării facturilor – răspuns parțial în VFP

Aceasta nu înseamnă că problema este insolubilă în VFP, ci doar că trebuie să recurgem la o altă variantă, după cum vom vedea cu alt prilej.

- Soluția fără CASE, DECODE, IIF

Problema poate fi rezolvată și fără operatorii pentru codarea structurilor alternative, prin reunirea facturilor fără nici o tranșă incasată cu facturile incasate parțial și cu facturile incasate total. Din rațiuni de economie de spațiu, este prezentată în continuare numai varianta DB2 (SQL-92):

```
SELECT F. NrFact, ;
    SUM(Cantitate * PretUnit * (1+ProcTVA)) / ;
    COUNT(DISTINCT VALUE( I.CodInc,1)) AS Facturat,
    SUM(VALUE(Transa,0))/ MAX(LF.Linie) AS Incasat,
    SUM(Cantitate * PretUnit * (1+ProcTVA)) / ;
    COUNT(DISTINCT VALUE( I.CodInc,1)) -
    SUM(VALUE(Transa,0))/ MAX(LF.Linie)
    AS Diferenta,
    'Fara nici o incasare' AS Situație
```

```
FROM LINIIFACT LF
INNER JOIN PRODUSE P ON LF.CodPr = P.CodPr
INNER JOIN FACTURI F ON LF.NrFact = F.NrFact
LEFT OUTER JOIN INCASFACT I ON F.NrFact=I.NrFact
GROUP BY F.NrFact
HAVING SUM(VALUE(Transa,0))/ MAX(LF.Linie) = 0
UNION
SELECT F. NrFact,
    SUM(Cantitate * PretUnit * (1+ProcTVA)) / ;
    COUNT(DISTINCT VALUE( I.CodInc,1)) AS Facturat,
    SUM(VALUE(Transa,0))/ MAX(LF.Linie) AS Incasat,
    SUM(Cantitate * PretUnit * (1+ProcTVA)) / ;
    COUNT(DISTINCT VALUE( I.CodInc,1)) -
    SUM(VALUE(Transa,0))/ MAX(LF.Linie),
    'Incasata parțial'
FROM LINIIFACT LF
INNER JOIN PRODUSE P ON LF.CodPr = P.CodPr
INNER JOIN FACTURI F ON LF.NrFact = F.NrFact
LEFT OUTER JOIN INCASFACT I ON F.NrFact=I.NrFact
GROUP BY F.NrFact
HAVING SUM(Cantitate * PretUnit * (1+ProcTVA)) / ;
    COUNT(DISTINCT VALUE( I.CodInc,1)) >
    SUM(VALUE(Transa,0))/ MAX(LF.Linie)
    AND SUM(VALUE(Transa,0))/ MAX(LF.Linie) <> 0
UNION
SELECT F. NrFact,
    SUM(Cantitate * PretUnit * (1+ProcTVA)) / ;
    COUNT(DISTINCT VALUE( I.CodInc,1)) AS Facturat,
    SUM(VALUE(Transa,0))/ MAX(LF.Linie) AS Incasat,
    SUM(Cantitate * PretUnit * (1+ProcTVA)) / ;
    COUNT(DISTINCT VALUE( I.CodInc,1)) -
    SUM(VALUE(Transa,0))/ MAX(LF.Linie),
    'INCASATA TOTAL'
FROM LINIIFACT LF
INNER JOIN PRODUSE P ON LF.CodPr = P.CodPr
INNER JOIN FACTURI F ON LF.NrFact = F.NrFact
LEFT OUTER JOIN INCASFACT I ON F.NrFact=I.NrFact
GROUP BY F.NrFact
HAVING SUM(Cantitate * PretUnit * (1+ProcTVA)) / ;
    COUNT(DISTINCT VALUE( I.CodInc,1)) =
    SUM(VALUE(Transa,0))/ MAX(LF.Linie)
```

```

IIF(DOW(DataFact+20)=6, ;
    DataFact+22, ;
    IIF(DOW(DataFact+20)=1, ;
        DataFact+21, ;
        DataFact+20) ) AS Scadenta, ;
CDOW(IIF(DOW(DataFact+20)=6, ;
    DataFact+22, ;
    IIF(DOW(DataFact+20)=1, ;
        DataFact+21, ;
        DataFact+20) ) ) AS Zi_Scadenta ;
FROM FACTURI

```

Să se afișeze în dreptul fiecărei facturi: valoarea facturată, valoarea incasată, diferența de incasat, precum și un text din care să reiasă dacă factura este fără nici o incasare, incasată parțial sau incasată total (și cele trei valori).

Practic, se dorește o situație cum este cea din figura 5.20.

NrFact	FACUTURAT	INCASAT	DIFERENTA	SITUAȚIUNE
1111	5399625	5399625	0	INCASATA TOTAL
1112	1337560	487705	849855	Incasație parțial
1113	1160250	1160250	0	INCASATA TOTAL
1114	6786570	0	6786570	Fara nici o incasare
1115	1651125	0	1651125	Fara nici o incasare
1116	1383375	0	1383375	Fara nici o incasare
1117	2320500	2320500	0	INCASATA TOTAL
1118	2052750	2052750	0	INCASATA TOTAL
1119	7242935	0	7242935	Fara nici o incasare
1120	1066240	731557	334683	Incasație parțial
1121	5438300	0	5438300	Fara nici o incasare

Figura 5.20. Situația Incasării facturilor

- O varianta SQL-92/DB2 este:

```

SELECT F. NrFact,
    SUM(Cantitate * PretUnit * (1+ProcTVA)) / ;
    COUNT(DISTINCT NVL( I.CodInc,1)) AS Facturat,
    SUM(NVL(Transa,0))/ MAX(LF.Linie) AS Incasat,
    SUM(Cantitate * PretUnit * (1+ProcTVA)) / ;
    COUNT(DISTINCT NVL( I.CodInc,1)) - ;
    SUM(NVL(Transa,0))/ MAX(LF.Linie)
    AS Diferenta,
    CASE WHEN SUM(NVL(Transa,0))/ MAX(LF.Linie) = 0
    THEN 'Fara nici o incasare'
    WHEN SUM(Cantitate * PretUnit * (1+ProcTVA)) / ;
    COUNT(DISTINCT NVL( I.CodInc,1)) >
    SUM(NVL(Transa,0))/ MAX(LF.Linie)
    THEN 'Incasata parțial'
    ELSE 'INCASATA TOTAL'
    END AS Situațiune

```

```

FROM LINIIFACT LF
INNER JOIN PRODUSE P ON LF.CodPr = P.CodPr
INNER JOIN FACTURI F ON LF.NrFact = F.NrFact
LEFT OUTER JOIN INCASFACT I ON F.NrFact=I.NrFact
GROUP BY F.NrFact

```

și una cel puțin la fel de născutătoare în Oracle:

```

SELECT F. NrFact,
    SUM(Cantitate * PretUnit * (1+ProcTVA)) / ;
    COUNT(DISTINCT NVL( I.CodInc,1)) AS Facturat,
    SUM(NVL(Transa,0))/ MAX(LF.Linie) AS Incasat,
    SUM(Cantitate * PretUnit * (1+ProcTVA)) / ;
    COUNT(DISTINCT NVL( I.CodInc,1)) - ;
    SUM(NVL(Transa,0))/ MAX(LF.Linie)
    AS Diferenta,
    DECODE (SUM(NVL(Transa,0))/ MAX(LF.Linie),
    0, 'Fara nici o incasare',
    DECODE (
        SIGN(SUM(Cantitate * PretUnit * (1+ProcTVA)) / ;
        COUNT(DISTINCT NVL( I.CodInc,1)) - ;
        SUM(NVL(Transa,0))/ MAX(LF.Linie)),
        0, 'INCASATA TOTAL', 'Incasata parțial'))
    AS Situațiune
FROM LINIIFACT LF, PRODUSE P, FACTURI F, INCASFACT I
WHERE LF.CodPr = P.CodPr AND LF.NrFact = F.NrFact AND
F.NrFact=I.NrFact (+)
GROUP BY F.NrFact

```

Ne-am folosit de acest exemplu (și) pentru a include o funcție DECODE în alta.
Din păcate, în VFP, soluția:

```

SELECT F. NrFact,
    SUM(Cantitate * PretUnit * (1+ProcTVA)) / ;
    COUNT(DISTINCT NVL( I.CodInc,1)) AS Facturat, ;
    SUM(NVL(Transa,0))/ MAX(LF.Linie) AS Incasat, ;
    SUM(Cantitate * PretUnit * (1+ProcTVA)) / ;
    COUNT(DISTINCT NVL( I.CodInc,1)) - ;
    SUM(NVL(Transa,0))/ MAX(LF.Linie) ;
    AS Diferenta, ;
    IIF(SUM(NVL(Transa,0))/ MAX(LF.Linie)=0, ;
    'Fara nici o incasare', ;
    IIF(SUM(Cantitate * PretUnit * (1+ProcTVA)) / ;
    COUNT(DISTINCT NVL( I.CodInc,1)) > ;
    SUM(NVL(Transa,0))/ MAX(LF.Linie), ;
    'Incasata parțial', 'INCASATA TOTAL' )) ;
    AS Situațiune ;
FROM LINIIFACT LF ;
INNER JOIN PRODUSE P ON LF.CodPr = P.CodPr ;
INNER JOIN FACTURI F ON LF.NrFact = F.NrFact ;
LEFT OUTER JOIN INCASFACT I ON F.NrFact=I.NrFact ;
GROUP BY F.NrFact

```

- Soluția Oracle „tradicională” are nevoie de un mic truc pentru ca în DECODE să transformăm intervalele în liste. De aceea se scade 1 din NVL(SporCD, 0) și, dacă rezultatul are semnul +, inseamnă că s-a acordat sporul respectiv, iar în caz contrar că nu s-a acordat:

```
SELECT DECODE(SIGN( NVL(SporCD, 0)-1), 1,
  'Au spor CD', 'Nu au spor CD')
  AS Pozitionare,
  COUNT(*) AS NrClienti
FROM SPORURI
WHERE An=2000 AND Luna=7
GROUP BY DECODE(SIGN( NVL(SporCD, 0)-1), 1,
  'Au spor CD', 'Nu au spor CD')
```

- Soluția VFP:

```
SELECT IIF(SporCD > 0, 'Au spor CD', 'Nu au spor CD'), ;
  COUNT(*) AS Nr ;
FROM SPORURI ;
WHERE An=2000 AND Luna=7 ;
GROUP BY 1
```

Scadența fiecărei facturi emise este de 20 de zile. Dacă însă data-limită cade într-o sămbătă sau duminică, atunci scadența se mută în luna următoare. Care sunt zilele scadente în aceste condiții?

Soluția DB2 se bazează pe funcția DAYOFWEEK, care întoarce 1 dacă data-argument se referă la duminică, 2 pentru luni... 6 pentru sămbătă. DAYNAME afișează numele zilei, iar interogarea următoare va genera rezultatul din figura 5.19.

```
SELECT NrFact, DataFact,
  DataFact + 20 DAYS AS Scadental,
  DAYNAME(DataFact + 20 DAYS) AS NumeZil,
CASE
  WHEN DAYOFWEEK(DataFact + 20 DAYS) = 6
    THEN DataFact + 22 DAYS
  ELSE
    CASE WHEN DAYOFWEEK(DataFact + 20 DAYS) = 1
      THEN DataFact + 21 DAYS
      ELSE DataFact + 20 DAYS
    END
  END AS Scadenta,
  DAYNAME (CASE WHEN DAYOFWEEK(DataFact + 20 DAYS) = 6
    THEN DataFact + 22 DAYS
    ELSE
      CASE WHEN DAYOFWEEK(DataFact + 20 DAYS)=1
        THEN DataFact + 21 DAYS
        ELSE DataFact + 20 DAYS
      END
    END ) AS Zi_Scadenta
FROM FACTURI
```

NRFACT	DATAFACT	SCADENTA1	NUMEZIL	SCADENTA	ZI_SCADENTA
1111	2000-08-01	2000-08-21	Monday	2000-08-21	Monday
1112	2000-08-01	2000-08-21	Monday	2000-08-21	Monday
1113	2000-08-01	2000-08-21	Monday	2000-08-21	Monday
1114	2000-08-01	2000-08-21	Monday	2000-08-21	Monday
1115	2000-08-02	2000-08-22	Tuesday	2000-08-22	Tuesday
1116	2000-08-02	2000-08-22	Tuesday	2000-08-22	Tuesday
1117	2000-08-03	2000-08-23	Wednesday	2000-08-23	Wednesday
1118	2000-08-04	2000-08-24	Thursday	2000-08-24	Thursday
1119	2000-08-07	2000-08-27	Sunday	2000-08-28	Monday
1120	2000-08-07	2000-08-27	Sunday	2000-08-28	Monday
1121	2000-08-07	2000-08-27	Sunday	2000-08-28	Monday
1122	2000-08-07	2000-08-27	Sunday	2000-08-28	Monday

Figura 5.19. Scadența rectificată a încasării facturilor

Soluția Oracle 8i2 este una ceva mai simplă, un rol decisiv avându-l, pe lângă structura CASE, puternica funcție TO_CHAR:

```
SELECT NrFact, TO_CHAR(DataFact, 'YYYY-MM-DD') AS DataFact,
  TO_CHAR(DataFact + 20, 'YYYY-MM-DD') AS Scadental,
  TO_CHAR(DataFact + 20, 'DAY') AS NumeZil,
CASE WHEN TO_CHAR(DataFact + 20, 'DAY') = 'SATURDAY'
  THEN TO_CHAR(DataFact + 22, 'YYYY-MM-DD')
ELSE
  CASE WHEN TO_CHAR(DataFact + 20, 'DAY') = 'SUNDAY'
    THEN TO_CHAR(DataFact + 21, 'YYYY-MM-DD')
    ELSE TO_CHAR(DataFact + 20, 'YYYY-MM-DD')
  END
END AS Scadenta,
TO_CHAR(
CASE WHEN TO_CHAR(DataFact + 20, 'DAY') = 'SATURDAY'
  THEN DataFact + 22
ELSE
  CASE WHEN TO_CHAR(DataFact + 20, 'DAY') =
    'SUNDAY'
    THEN DataFact + 21
    ELSE DataFact + 20
  END
END,
'DAY') AS Zi_Scadenta
FROM FACTURI
```

Visual FoxPro prezintă funcțiile CDOW (Character Day Of the Week) și DOW (Day Of the Week), rezultatul din figura 5.19 fiind obținut după cum urmează:

```
SELECT NrFact AS Factura, DataFact, ;
  DataFact + 20 AS Scadental, ;
  CDOW(DataFact+20) AS NumeZil, ;
```

Rezultatul arată ca în figura 5.17, soluția fiind valabilă și în DB2. Pentru a răspunde exact la întrebare, se poate redacta o variantă SQL-92/DB2, după cum urmează (rezultatul final este prezentat în figura 5.18):

```
SELECT CASE CodPost
    WHEN '6600' THEN 'Din Iasi'
    ELSE 'Din afara Iasiului'
  END AS Pozitionare,
  COUNT(*) AS NrClienti
FROM CLIENTI
GROUP BY CASE CodPost WHEN '6600' THEN 'Din Iasi'
    ELSE 'Din afara Iasiului'
  END
```

DENCOD	CODCL	CODPOST	Pozitionare
Clien 1 SRL	1001	6600	Din Iasi
Clien 2 SA	1002	6600	Din Iasi
Clien 3 SRL	1003	6500	Din afara Iasiului
Clien 4	1004	5725	Din afara Iasiului
Clien 5 SRL	1005	1900	Din afara Iasiului
Clien 6 SA	1006	5550	Din afara Iasiului
Clien 7 SRL	1007	1900	Din afara Iasiului

Figura 5.17. Atribut calculat pe baza unei secvențe alternative

Pozitionare	NrClienti
Din Iasi	2
Din afara Iasiului	5

Figura 5.18. Numărul clientilor ieșeni și al celor din afara Iașiului

Oracle nu avea până la versiunea 8i structura CASE... WHEN... În schimb, putea fi folosită funcția DECODE, care are o logică similară. DECODE întoarce o valoare în funcție de conținutul unui argument:

```
DECODE (atribut, valoare_testată1, valoare_returnată1,
        valoare_testată2, valoare_returnată2,
        ...
        valoare_returnatăN)
```

Valoare_returnatăN este echivalenta OTHERWISE-ului dintr-o structură de tip CASE din orice 3GL. Pentru a obține tabelul din figura 5.17, soluția Oracle este:

```
SELECT DenCl, CodCl, CodPost,
       DECODE (CodPost,'6600 ','Din Iasi',
               'Din afara Iasiului')
  AS Pozitionare
 FROM CLIENTI
```

iar pentru răspunsul final la problema pusă:

```
SELECT DECODE (CodPost,'6600 ','Din Iasi',
               'Din afara Iasiului') AS Pozitionare,
          COUNT(*) AS NrClienti
   FROM CLIENTI
  GROUP BY DECODE (CodPost,'6600 ','Din Iasi',
                   'Din afara Iasiului')
```

O dată cu Oracle 8i însă, se poate vorbi de o nouă apropiere de SQL-92, deoarece este operațională și varianta:

```
SELECT CASE
    WHEN CodPost = '6600' THEN 'Din Iasi'
    ELSE 'Din afara Iasiului' END AS Pozitionare,
    COUNT(*) AS NrClienti
  FROM CLIENTI
  GROUP BY CASE
    WHEN CodPost = '6600' THEN 'Din Iasi'
    ELSE 'Din afara Iasiului' END
```

Visual FoxPro pune la dispoziție în asemenea situații IF-ul imediat – IIF-ul:

```
SELECT IIF(CodPost='6600 ', PADR('Din Iasi',20), ;
           'Din afara Iasiului') AS Pozitionare, ;
           COUNT(*) AS NrClienti ;
  FROM CLIENTI ;
  GROUP BY Pozitionare
```

Față de DB2 și Oracle, avantajul VFP este că în GROUP BY nu mai trebuie scrisă întreaga expresie – IIF-ul care desemnează coloana după care se face gruparea. De fapt, chiar este interzisă scrierea expresiei, în locul acesteia fiind indicat numele coloanei calculate (Pozitionare) sau numărul coloanei. Soluția următoare este echivalentă cu ultima consultare:

```
SELECT IIF(CodPost='6600 ', PADR('Din Iasi',20), ;
           'Din afara Iasiului') AS Pozitionare, ;
           COUNT(*) AS NrClienti ;
  FROM CLIENTI ;
  GROUP BY 1
```

Câți angajați au primit, pe luna iulie 2000, spor pentru condiții deosebite și câți nu?

- Soluția SQL-92/DB2/Oracle 8i:


```
SELECT CASE WHEN SporCD > 0
                  THEN 'Au spor CD'
                  ELSE 'Nu au spor CD' END,
             COUNT(*) AS Nr
      FROM SPORURI
      WHERE An=2000 AND Luna=7
      GROUP BY CASE WHEN SporCD > 0
                  THEN 'Au spor CD' ELSE 'Nu au spor CD' END
```

```

LEFT OUTER JOIN SPORURI S1 ON PERSONAL2.Marca=S1.Marca
AND 2000=S1.An AND 5 = S1.Luna
LEFT OUTER JOIN SPORURI S2 ON PERSONAL2.Marca=S2.Marca
AND 2000=S2.An AND 6 = S2.Luna
ORDER BY NumePren

```

• Soluția Oracle:

```

SELECT PERSONAL2.Marca, NumePren,
TO_CHAR(S1.SporNoapte, '99999999') AS SporNoapte_Mai,
TO_CHAR(S2.SporNoapte, '99999999') AS SporNoapte_Iunie
FROM PERSONAL2, SPORURI S1, SPORURI S2
WHERE PERSONAL2.Marca=S1.Marca (+) AND
S1.An(+)=2000 AND S1.Luna(+) = 5
AND PERSONAL2.Marca=S2.Marca (+) AND
S2.An(+)=2000 AND S2.Luna(+) = 6
ORDER BY NumePren

```

Elementul-cheie îl constituie prezența operatorului jonecții externe în dreptul atributelor An și Luna. Prin aceasta se includ în rezultat și liniile din tabela PERSONAL2 care nu prezintă corespondență după atributul Marca cu tabela SPORURI pentru cele două luni.

Să se obțină sporurile de noapte pentru al doilea trimestru al anului 2000, atât lunar, cât și cumulat. Sunt necesare trei instanțe ale tabelei SPORURI, frazele SELECT devenind supraponderale, după cum se va vedea în continuare.

• Soluția SQL-92/DB2:

```

SELECT PERSONAL2.Marca, NumePren,
VALUE(S1.SporNoapte,0) AS Spor_Noapte_Aprilie,
VALUE(S2.SporNoapte,0) AS Spor_Noapte_Mai,
VALUE(S3.SporNoapte,0) AS Spor_Noapte_Iunie,
VALUE(S1.SporNoapte,0) + VALUE(S2.SporNoapte,0) +
VALUE(S3.SporNoapte,0) AS Spor_Noapte_Trim_II
FROM PERSONAL2
LEFT OUTER JOIN SPORURI S1 ON PERSONAL2.Marca=S1.Marca
AND 2000=S1.An AND 4 = S1.Luna
LEFT OUTER JOIN SPORURI S2 ON PERSONAL2.Marca=S2.Marca
AND 2000=S2.An AND 5 = S2.Luna
LEFT OUTER JOIN SPORURI S3 ON PERSONAL2.Marca=S3.Marca
AND 2000=S3.An AND 6 = S3.Luna
ORDER BY NumePren

```

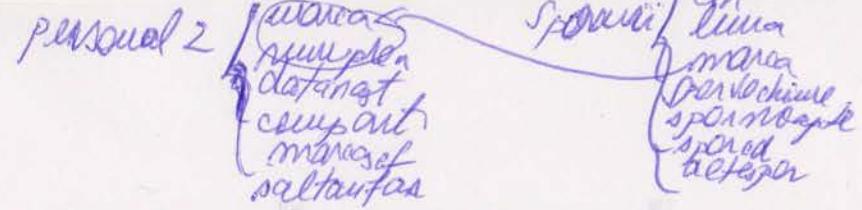
• Soluția VFP se obține din precedenta, înlocuind VALUE cu NVL.

• Soluția Oracle 8:

```

SELECT PERSONAL2.Marca, NumePren,
TO_CHAR(NVL(S1.SporNoapte,0), '9999999999')
AS Spor_Noapte_Aprilie,
TO_CHAR(NVL(S2.SporNoapte,0), '9999999999')
AS Spor_Noapte_Mai,
TO_CHAR(NVL(S3.SporNoapte,0), '9999999999')

```



```

AS Spor_Noapte_Iunie,
TO_CHAR(NVL(S1.SporNoapte,0)+NVL(S2.SporNoapte,0) +
NVL(S3.SporNoapte,0), '9999999999')
AS Spor_Noapte_Trim_II
FROM PERSONAL2, SPORURI S1, SPORURI S2, SPORURI S3
WHERE PERSONAL2.Marca=S1.Marca (+) AND
S1.An(+) = 2000 AND S1.Luna(+) = 4
AND PERSONAL2.Marca=S2.Marca (+) AND
S2.An(+) = 2000 AND S2.Luna(+) = 5
AND PERSONAL2.Marca=S3.Marca (+) AND
S3.An(+) = 2000 AND S3.Luna(+) = 6
ORDER BY NumePren

```

Cele trei soluții ar trebui să conducă la un rezultat asemănător celui din figura 5.16.

MARCA	NUMEPREN	SPOR_NOAPTE_APRILE	SPOR_NOAPTE_MAI	SPOR_NOAPTE_IUNIE	SPOR_NOAPTE_TRIM_II
1	ANGAJAT 1	0	0	0	0
10	ANGAJAT 10	0	0	80000	80000
2	ANGAJAT 2	450000	450000	0	900000
3	ANGAJAT 3	560000	0	0	560000
4	ANGAJAT 4	0	0	150000	150000
5	ANGAJAT 5	0	0	150000	150000
6	ANGAJAT 6	0	0	0	0
7	ANGAJAT 7	0	0	0	0
8	ANGAJAT 8	0	0	0	0
9	ANGAJAT 9	0	0	0	0

Figura 5.16. Sporurile de noapte pe trimestrul al II-lea, pe luni și cumulat

5.3. Structuri alternative: CASE, DECODE, IIF

SQL este un limbaj neprocedural. Cu toate acestea, începând cu standardul 92, SQL prezintă facilitatea codării structurilor alternative prin clauza CASE.

Căți dintre clienți sunt din Iași (cod poștal 6600) și căți din afara Iașului?

Incepem cu o versiune „ajutătoare”. Pentru a scrie în dreptul fiecărui client dacă e din Iași sau din afara Iașului, se folosește interogarea:

```

SELECT DenCl, CodCl, CodPost,
CASE CodPost
WHEN '6600' THEN 'Din Iasi'
ELSE 'Din afara Iasiului'
END
AS Poziționare
FROM CLIENTI

```

- Soluția SQL-92, DB2 și VFP (în VFP se înlocuiește VALUE cu NVL):

```
SELECT F.NrFact,
       SUM(Cantitate * PretUnit * (1+ProcTVA)) /
       COUNT(DISTINCT VALUE( I.CodInc,1)) AS Facturat,
       SUM(VALUE(Transa,0))/ MAX(LF.Linie) AS Incasat
  FROM LINIIFACT LF
 INNER JOIN PRODUSE P ON LF.CodPr = P.CodPr
 INNER JOIN FACTURI F ON LF.NrFact = F.NrFact
 LEFT OUTER JOIN INCASFACT I ON F.NrFact=I.NrFact
 GROUP BY F.NrFact
```

- Soluția Oracle 8:

```
SELECT F.NrFact, SUM(Cantitate * PretUnit * (1+ProcTVA)) /
       COUNT(DISTINCT NVL(I.CodInc,1)) AS Facturat,
       SUM(NVL(Transa,0)) / MAX(LF.Linie) AS Incasat
  FROM LINIIFACT LF, PRODUSE P, FACTURI F, INCASFACT I
 WHERE LF.CodPr = P.CodPr AND LF.NrFact = F.NrFact AND
       F.NrFact=I.NrFact (+)
 GROUP BY F.NrFact
```

Care au fost sporurile de noapte acordate angajașilor pe luniile mai și iunie 2000?

Situată obținută se referă la două luni. Există însă angajați care nu au acest spor pe una sau chiar pe ambele luni. Cu interogarea:

```
SELECT AN, Luna, PERSONAL2.Marcă, NumePren, SporNoapte
  FROM PERSONAL2, SPORURI
 WHERE PERSONAL2.Marcă=SPORURI.Marcă AND
       an=2000 AND Luna IN (5,6)
 ORDER BY NumePren, An, Luna
```

rezultatul arată ca în figura 5.13.

AN	LUNA	MARCA	NUMEPREN	SPORNOAPTE
2000	5	1	ANGAJAT 1	0
2000	6	1	ANGAJAT 1	0
2000	5	10	ANGAJAT 10	0
2000	6	10	ANGAJAT 10	80000
2000	5	2	ANGAJAT 2	450000
2000	6	2	ANGAJAT 2	0
2000	5	3	ANGAJAT 3	0
2000	6	4	ANGAJAT 4	150000
2000	6	5	ANGAJAT 5	150000

Figura 5.13. Sporurile de noapte pe mai și iunie – varianta 1 de afișare

Pentru acest exemplu, interesează însă formatul de prezentare din figura 5.14.

MARCA	NUMEPREN	SPORNOAPTE_MAI	SPORNOAPTE_IUNIE
1	ANGAJAT 1	0	0
10	ANGAJAT 10	0	80000
2	ANGAJAT 2	450000	0
3	ANGAJAT 3	0	
4	ANGAJAT 4		150000
5	ANGAJAT 5		150000
6	ANGAJAT 6		
7	ANGAJAT 7		
8	ANGAJAT 8		
9	ANGAJAT 9		

Figura 5.14. Sporurile de noapte pe mai și iunie – rezultatul dorit

Schimbăm alura SELECT-ului:

```
SELECT AN, Luna, PERSONAL2.Marcă, NumePren, SporNoapte
  FROM PERSONAL2 LEFT OUTER JOIN SPORURI
    ON PERSONAL2.Marcă=SPORURI.Marcă AND
      An=2000 AND Luna=5
 ORDER BY NumePren, An, Luna
```

Se obține astfel tabela din figura 5.15. Elementul îmbucurător este că în rezultat au fost inclusi toți angajații. Cei care nu erau angajați în această perioadă prezintă valori NULL pentru atrbutele An și Luna. Pentru afișarea pe coloane separate a sporurilor de noapte pe lunile mai și iunie (ca în figura 5.14) sunt necesare două joncțiuni externe ale tablei PERSONAL2 cu două instanțe ale tablei SPORURI.

AN	LUNA	MARCA	NUMEPREN	SPORNOAPTE
2000	5	1	ANGAJAT 1	0
2000	5	10	ANGAJAT 10	0
2000	5	2	ANGAJAT 2	450000
2000	5	3	ANGAJAT 3	0
		4	ANGAJAT 4	
		5	ANGAJAT 5	
		6	ANGAJAT 6	
		7	ANGAJAT 7	
		8	ANGAJAT 8	
		9	ANGAJAT 9	

Figura 5.15. Sporurile de noapte pe luna mai

- Soluția SQL-92/DB2/Visual FoxPro:

```
SELECT PERSONAL2.Marcă, NumePren,
       S1.SporNoapte AS SporNoapte_Mai,
       S2.SporNoapte AS SporNoapte_Iunie
  FROM PERSONAL2
```

Joncțiune externă totală

Nu există nici o posibilitate directă de a jonctiona total două tabele. Unica soluție este reunirea rezultatelor joncțiunii la stânga și la dreapta:

```
SELECT *
FROM R1, R2
WHERE R1.C (+) = R2.C
UNION
SELECT *
FROM R1, R2
WHERE R1.C = R2.C (+)
```

dreapta
sug

Cele trei caractere, (+), care simbolizează joncțiunea externă sunt plasate în dreptul tabelei din dreapta LEFT OUTER JOIN-ului și celei din stânga RIGHT OUTER JOIN-ului⁵³. Un element (suplimentar) creator de confuzii îl constituie faptul că simbolul (cele trei caractere) este plasat diferențiat, fie înaintea semnului =, în cazul joncțiunii externe la dreapta, fie imediat după atributul din dreapta semnului =, în cazul joncțiunii externe la stânga.

Următorul exemplu este desprins tot din algebra relațională (exemplul 20): Care sunt localitățile în care nu avem nici un client?

```
SELECT *
FROM LOCALITATI LEFT OUTER JOIN CLIENTI
ON LOCALITATI.CodPost = CLIENTI.CodPost
WHERE CLIENTI.CodPost IS NULL
```

CODPOST	LOC	JUD	CODCL	DENOL	CODFISCAL	ADRESA	CODPOST	TELEFON
5300	Focșani	VN						
5800	Suceava	SV						
6400	Bihor	VS						

Figura 5.11. Localitățile în care nu sunt clienți

Care este situația incasării facturii 1120?

Revenim la un exemplu din capitolul anterior, de la funcțiile SUM și MAX. Spuneam, la momentul respectiv, că soluția formulată este valabilă numai pentru facturile cu măcar o tranșă de incasare. Beneficind de avantajele joncțiunii externe, se poate redacta o soluție ameliorată care va funcționa în orice situație.

Pentru obținerea valorii facturate trebuie împărțită suma totală la numărul de tranșe de incasări (repetarea se producea din cauza joncțiunii). Atunci când nu există nici o tranșă, împărțirea trebuie să se facă la 1; astfel încât expresia valorii totale a facturii va fi: SUM (Cantitate * PretUnit * (1 + ProcTVA)) / COUNT (DISTINCT VALUE (I.CodInc, 1)).

In schimb, totalul tranșelor incasate (0 dacă nu există nici o tranșă) trebuie împărțit, pentru fiecare factură, la numărul de liniî din factură. De aici expresia: SUM (VALUE (Transa, 0)) / MAX (LF.Linie).

53. Vedi și [Fotache00-3].

• Soluția SQL-92 și DB2:

```
SELECT '1120' AS NrFact,
SUM(Cantitate * PretUnit * (1+ProcTVA)) /
COUNT(DISTINCT VALUE( I.CodInc,1)) AS Facturat,
SUM(VALUE(Transa,0)) / MAX(LF.Linie) AS Incasat
FROM LINIIFACT LF
INNER JOIN PRODUSE P ON LF.CodPr = P.CodPr
INNER JOIN FACTURI F ON LF.NrFact = F.NrFact
LEFT OUTER JOIN INCASFACT I ON F.NrFact=I.NrFact
WHERE F.NrFact = 1120
```

Așa cum am văzut, DB2 permite folosirea, în locul VALUE, a funcției COALESCE.

• Soluția echivalentă Visual FoxPro:

```
SELECT '1120' AS NrFact, SUM(Cantitate * PretUnit *
(1+ProcTVA)) /
COUNT(DISTINCT NVL(I.CodInc,1)) AS Facturat,
SUM(NVL(Transa,0)) / MAX(LF.Linie) AS Incasat ;
FROM LINIIFACT LF ;
INNER JOIN PRODUSE P ON LF.CodPr = P.CodPr ;
INNER JOIN FACTURI F ON LF.NrFact = F.NrFact ;
LEFT OUTER JOIN INCASFACT I ON F.NrFact=I.NrFact ;
WHERE F.NrFact = 1120
```

• Soluția Oracle 8:

```
SELECT '1120' AS NrFact, SUM(Cantitate * PretUnit *
(1+ProcTVA)) / COUNT(DISTINCT NVL(I.CodInc,1))
AS Facturat,
SUM(NVL(Transa,0)) / MAX(LF.Linie) AS Incasat
FROM LINIIFACT LF, PRODUSE P, FACTURI F, INCASFACT I
WHERE LF.CodPr = P.CodPr AND LF.NrFact = F.NrFact AND
F.NrFact=I.NrFact (+) AND F.NrFact = 1120
```

Care sunt valorile facturate și incasate ale fiecărei facturi?

Și acesta este un caz rezolvat incomplet în capitolul anterior (la prezentarea clauzei GROUP BY). Ambile soluții prezentate în continuare conduc către un rezultat de forma celui din figura 5.12.

NRFACT	FACTURAT	INCASAT
1111	5399625	5399625
1112	1337560	487705
1113	1160250	1160250
1114	5786570	0
1115	1651125	0
1116	1383375	0
1117	2320500	2320500
1118	2052750	2052750
1119	7242935	0
1120	1066240	731557
1121	5438300	0

Figura 5.12. Valorile facturate și incasate ale fiecărei facturi

MARCA	NUMEPREN.	COMPARTIMENT	TOTALSPORUR
1	ANGAJAT 1	DIRECTIUNE	600000
2	ANGAJAT 2	FINANCIAR	470000
3	ANGAJAT 3	MARKETING	450000
4	ANGAJAT 4	FINANCIAR	490000
5	ANGAJAT 5	FINANCIAR	560000
6	ANGAJAT 6	FINANCIAR	925000
7	ANGAJAT 7	FINANCIAR	1015000
8	ANGAJAT 8	MARKETING	1790000
9	ANGAJAT 9	MARKETING	1635000
10	ANGAJAT 10	RESURSE UMANE	1230000

Figura 5.10. Conversia valorilor nule în zero și evaluarea corectă a expresiei

În DB2, de obicei, funcției COALESCE îl este preferată o altă ce produce rezultate identice – VALUE. Prin urmare, ultima soluție se poate scrie astfel:

```
SELECT SPORURI.Marca, NumePren, Compart,
       VALUE (SporVechime,0) + VALUE(SporNoapte,0) +
       VALUE (SporCD,0) + VALUE (AlteSpor,0) AS TotalSporuri
  FROM PERSONAL2, SPORURI
 WHERE PERSONAL2.Marca=SPORURI.Marca AND
       An = 2000 AND Luna=7
```

Oracle și Visual FoxPro pun la dispoziție o altă funcție ce acționează după aceeași logică – NVL (de la NullValue), astfel încât varianta comună pentru interogarea anterioară este:

```
SELECT SPORURI.Marca, NumePren, Compart,
       NVL(SporVechime,0) + NVL(SporNoapte,0) +
       NVL(SporCD,0) + NVL(AlteSpor,0) AS TotalSporuri
  FROM PERSONAL2, SPORURI
 WHERE PERSONAL2.Marca=SPORURI.Marca AND
       An = 2000 AND Luna=7
```

Este important de reținut că funcțiile VALUE, COALESCE, NVL nu se aplică la nivel de expresie, ci fiecărui operand susceptibil de nulitate. De asemenea, un alt element interesant legat de valorile nule ține de conversia în sens invers, dintr-o valoare oarecare, în NULL.

Să se determine totalul sporurilor de noapte pentru luna iulie 2000, dar, în rezultat, să nu fie luate în calcul valoarea (valorile) 300.000 lei.

Soluția „clasică” este:

```
SELECT SUM(SporVechime)
  FROM SPORURI
 WHERE An = 2000 AND Luna=7 AND SporVechime <> 300000
O variantă ceva mai elegantă se redactează prin utilizarea funcției NULLIF prezentă în SQL-92, care, în interogarea de mai jos, convertește orice apariție a valorii 300 000 în NULL:
SELECT SUM(NULLIF(SporVechime, 300000))
  FROM SPORURI
 WHERE An = 2000 AND Luna=7
```

În Oracle nu există funcția NULLIF, dar în locul său se poate folosi REPLACE:

```
SELECT SUM(REPLACE(SporVechime, 300000, NULL))
  FROM SPORURI
 WHERE An = 2000 AND Luna=7
```

Visual FoxPro nu dispune de nici una dintre funcțiile de mai sus, dar, după cum vom vedea într-un viitor paragraf, conversia (substituirea) valorilor este posibilă prin IF-ul imediat (IFF-ul).

5.2. Juncțiunea externă

Standardul SQL-2 introduce operatorii necesari juncțiunii externe:

- LEFT OUTER JOIN pentru juncțiune externă la stânga,
- RIGHT OUTER JOIN pentru juncțiune externă la dreapta,
- FULL OUTER JOIN pentru juncțiune externă totală (în ambele direcții).

Dacă ne raportăm la exemplul teoretic din algebra relațională (capitolul 2, figura 2.24), atunci juncțiunile externe la stânga, la dreapta și totală dintre relațiile R1 și R2 se transcriu în standardul SQL-92 astfel:

Juncțiune externă la stânga

```
SELECT *
  FROM R1 LEFT OUTER JOIN R2 ON R1.C=R2.C
```

Juncțiune externă la dreapta

```
SELECT *
  FROM R1 RIGHT OUTER JOIN R2 ON R1.C=R2.C
```

Juncțiune externă totală

```
SELECT *
  FROM R1 FULL OUTER JOIN R2 ON R1.C=R2.C
```

Atât DB2, cât și Visual FoxPro au implementat acești operatori; însă, curios, nici până la versiunea 8 Oracle nu a implementat OUTER JOIN-ul. Totuși, juncțiunea externă este realizabilă, dar ceva mai greoi, astfel:

Juncțiune externă la stânga

```
SELECT *
  FROM R1, R2
 WHERE R1.C = R2.C (+)
```

Juncțiune externă la dreapta

```
SELECT *
  FROM R1, R2
 WHERE R1.C (+) = R2.C
```

Se observă un lucru curios: dacă reunim mulțimea angajaților născuți înainte de data fixată cu mulțimea celor născuți după această dată nu obținem relația inițială PERSONAL2 (figura 5.8).

```
SELECT *
FROM PERSONAL2
WHERE DataNast < '01-01-1970'
UNION
SELECT *
FROM PERSONAL2
WHERE DataNast >= '01-01-1970'
```

MARCA	NUMEPREN	DATANAST	COMPART	MARCASEF	SALTARIFAR
1	ANGAJAT 1	01-JUL-62	DIRECTIUNE		6000000
2	ANGAJAT 2	11-OCT-77	FINANCIAR	1	4500000
3	ANGAJAT 3	22-AUG-62	MARKETING	1	4500000
5	ANGAJAT 5	30-APR-65	FINANCIAR	2	4200000
6	ANGAJAT 6	09-NOV-65	FINANCIAR	5	3500000
8	ANGAJAT 8	31-DEC-60	MARKETING	3	2900000
9	ANGAJAT 9	28-FEB-76	MARKETING	3	4100000
10	ANGAJAT 10	29-JAN-72	RESURSE UMANE	1	3700000

Figura 5.8. Reuniunea persoanelor născute înainte de 1 ianuarie 1970 cu persoanele născute după această dată

Din tabela obținută în figura 5.8, lipsesc angajații care nu au precizată data nașterii, altfel spus, persoanele „fără vîrstă” (precum maestrul Gică Petrescu). Pentru recompoziția tabelei PERSONAL2, în reuniune mai trebuie adăugate și liniile pentru care DataNast IS NULL:

```
SELECT *
FROM PERSONAL2
WHERE DataNast < '01-01-1970'
UNION
SELECT *
FROM PERSONAL2
WHERE DataNast >= '01-01-1970'
UNION
SELECT *
FROM PERSONAL2
WHERE DataNast IS NULL
ORDER BY Marca
```

Acest exemplu este grăitor în privința logicii trivalente a modelului relațional în ceea ce privește valorile nule. În continuare, interesează un alt aspect al NULL-itațiilor: modul de evaluare a expresiilor în care unul sau mai mulți operanzi au valori nule.

Care este totalul sporurilor fiecărui angajat pe luna iulie 2000?

Soluția pare a fi de forma:

```
SELECT SPORURI.Marcă, NumePren, Compart,
SporVechime + SporNoapte + SporCD + AlteSpor
AS TotalSporuri
```

```
FROM PERSONAL2, SPORURI
WHERE PERSONAL2.Marcă=SPORURI.Marcă AND
An = 2000 AND Luna=7
```

Din păcate, rezultatul este incorrect – vezi figura 5.9 – deoarece, din prezentarea conținutului tabelei SPORURI (figura 5.3) reiese că, pe luna iulie 2000, ANGAJAT 1 are calculat spor de vechime (600.000 lei), iar ANGAJAT 4 are, pe aceeași lună, și spor de vechime (190.000 lei), și de noapte (150.000 lei), și alte sporuri (150.000 lei), iar aceste sporuri nu au fost luate în calcul la însumare.

MARCA	NUMEPREN	COMPART	TOTALSPORURI
1	ANGAJAT 1	DIRECTIUNE	
2	ANGAJAT 2	FINANCIAR	470000
3	ANGAJAT 3	MARKETING	450000
4	ANGAJAT 4	FINANCIAR	
5	ANGAJAT 5	FINANCIAR	560000
6	ANGAJAT 6	FINANCIAR	925000
7	ANGAJAT 7	FINANCIAR	1015000
8	ANGAJAT 8	MARKETING	1780000
9	ANGAJAT 9	MARKETING	1635000
10	ANGAJAT 10	RESURSE UMANE	1230000

Figura 5.9. Rezultatul unei expresii în care cel puțin un operand este NULL

Explicația este simplă: dacă, într-o expresie, unul dintre operanzi este NULL, atunci rezultatul evaluării întregii expresii este NULL. Fac excepție funcțiile statistice. Dacă, spre exemplu, vrem să calculăm:

Totalul sporurilor de noapte acordate pentru luna iulie 2000,

fraza:

```
SELECT SUM(SporNoapte) AS Total_SporuriNoapte_Luna_7
FROM SPORURI
WHERE An = 2000 AND Luna=7
```

calculează corect rezultatul: 2.160.000 lei.

Revenim la cazul cu probleme. Pentru a asigura corectitudinea totalului, ar trebui ca în expresie orice valoare nulă să fie considerată zero. Lucru realizabil, deoarece SQL-92 este „prevăzut” cu o funcție în acest sens – COALESCE:

```
SELECT SPORURI.Marcă, NumePren, Compart,
COALESCE(SporVechime,0) + COALESCE (SporNoapte,0) +
COALESCE (SporCD,0) + COALESCE (AlteSpor,0) AS
TotalSporuri
FROM PERSONAL2, SPORURI
WHERE PERSONAL2.Marcă=SPORURI.Marcă AND
An = 2000 AND Luna=7
```

Sumele obținute sunt în acest caz cele din figura 5.10.

AN	LUNA	MARCA	SPORVECHIME	SPORNOAPTE	SPORCD	ALTESPOR
2000	4	1	600000	0	0	320000
2000	4	2	300000	450000	0	170000
2000	4	3	450000	560000	1200000	570000
2000	5	1	600000	0	0	0
2000	5	2	300000	450000	0	170000
2000	5	3	450000	0	0	0
2000	5	10	370000	0	0	860000
2000	6	1	600000	0	0	0
2000	6	2	300000	0	0	150000
2000	6	4	190000	150000	680000	150000
2000	6	5	250000	150000	0	50000
2000	6	10	370000	80000	0	60000
2000	7	1	600000	0		
2000	7	2	300000	0	0	170000
2000	7	3	450000	0	0	0
2000	7	4	190000	150000		150000
2000	7	5	420000	0	0	140000
2000	7	6	350000	575000	0	0
2000	7	7	140000	675000	0	0
2000	7	8	290000	0	1500000	0
2000	7	9	300000	560000	775000	0
2000	7	10	370000	0	0	860000

Figura 5.3. Conținutul tabelei SPORURI

MARCA	NUMEPREN	COMPART	AN	LUNA
1	ANGAJAT 1	DIRECTIUNE	2000	7
4	ANGAJAT 4	FINANCIAR	2000	7

Figura 5.4. Angajații pentru care nu s-au operat sporurile pentru condiții deosebite

Care sunt angajații și lunile în care aceștia nu au primit spor pentru condiții deosebite?

Atât soluția, cât și rezultatul sunt sensibil diferite – vezi figura 5.5.

```
SELECT SPORURI.MARCA, NumePren, Compart, An, Luna
FROM PERSONAL2, SPORURI
WHERE PERSONAL2.MARCA=SPORURI.MARCA
AND SporCD = 0
ORDER BY An, Luna, NumePren
```

MARCA	NUMEPREN	COMPART	AN	LUNA
1	ANGAJAT 1	DIRECTIUNE	2000	4
2	ANGAJAT 2	FINANCIAR	2000	4
1	ANGAJAT 1	DIRECTIUNE	2000	5
10	ANGAJAT 10	RESURSE UMANE	2000	5
2	ANGAJAT 2	FINANCIAR	2000	5
3	ANGAJAT 3	MARKETING	2000	5
1	ANGAJAT 1	DIRECTIUNE	2000	6
10	ANGAJAT 10	RESURSE UMANE	2000	6
2	ANGAJAT 2	FINANCIAR	2000	6
5	ANGAJAT 5	FINANCIAR	2000	6
10	ANGAJAT 10	RESURSE UMANE	2000	7
2	ANGAJAT 2	FINANCIAR	2000	7
3	ANGAJAT 3	MARKETING	2000	7
5	ANGAJAT 5	FINANCIAR	2000	7
6	ANGAJAT 6	FINANCIAR	2000	7
7	ANGAJAT 7	FINANCIAR	2000	7

Figura 5.5. Angajații și lunile pentru care SporCD este zero (neNULL)

Care dintre angajați sunt născuți înainte de 1 ianuarie 1970 și care după această dată?

Persoanele născute înainte (figura 5.6):

```
SELECT *
FROM PERSONAL2
WHERE DataNast < '01-01-1970'
```

și după (figura 5.7):

```
SELECT *
FROM PERSONAL2
WHERE DataNast >= '01-01-1970'
```

MARCA	NUMEPREN	DATANAST	COMPART	MARCASEF	SALTARIAR
1	ANGAJAT 1	10-JUL-62	DIRECTIUNE	1	600000
3	ANGAJAT 3	22-AUG-65	MARKETING	1	450000
5	ANGAJAT 5	30-APR-65	FINANCIAR	2	420000
6	ANGAJAT 6	09-NOV-65	FINANCIAR	5	350000
8	ANGAJAT 8	31-DEC-60	MARKETING	3	290000

Figura 5.6. Persoane născute înainte de 1 ianuarie 1970

MARCA	NUMEPREN	DATANAST	COMPART	MARCASEF	SALTARIAR
2	ANGAJAT 2	11-OCT-77	FINANCIAR	1	450000
9	ANGAJAT 9	28-FEB-76	MARKETING	3	410000
10	ANGAJAT 10	29-JAN-72	RESURSE UMANE	1	370000

Figura 5.7. Persoane născute după 1 ianuarie 1970

valorilor nule (una din regulile SGBD-urilor relaționale) și, pentru a identifica clienții fără adresa introdusă, se folosea o variantă de genul:

```
SELECT * FROM CLIENTI WHERE Adresa = ""
```

sau

```
SELECT * FROM CLIENTI WHERE EMPTY(Adresa)
```

Este important de notat că, în vederea identificării valorilor nule, operatorul are forma IS NULL și nu =NULL. Prin execuția frazei SQL:

```
SELECT *
FROM CLIENTI
WHERE Adresa = NULL
```

se va obține o tabelă cu 0 (zero) linii. Rezultatul evaluării Adresa = NULL va fi întotdeauna FALSE. DB2 chiar afișează, la execuția acestei interogări, un convingător mesaj de eroare: SQL0401N The data types of the operands for the operation "=" are not compatible.

Logica operatorilor NOT, AND și OR este ilustrată în tabelul 5.1.

Tabelul 5.1. Rezultatele utilizării operatorilor NOT, AND și AND

Aplicarea operatorului NOT unei condiții

NOT	TRUE	FALSE	UNKNOWN
	FALSE	TRUE	UNKNOWN

Combinarea a două expresii utilizând operatorul AND

AND	TRUE	FALSE	UNKNOWN
TRUE	TRUE	FALSE	UNKNOWN
FALSE	FALSE	FALSE	FALSE
UNKNOWN	UNKNOWN	FALSE	UNKNOWN

Combinarea a două expresii utilizând operatorul OR

OR	TRUE	FALSE	UNKNOWN
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	UNKNOWN
UNKNOWN	TRUE	UNKNOWN	UNKNOWN

Deși baza de date prezentată este alcătuită deja dintr-un număr considerabil de tabele și atribute, introducem încă două tabele cu scop colateral temei „vânzări/incasări” – acela de a gestiona, o parte din datele privind drepturile bănești (salariu negociat și sporuri) ale angajaților firmei. Prima se numește PERSONAL2 și conține date generale despre angajați: marca; nume și prenume; data nașterii; compartiment; marca șefului (direct); salariu tarifar. A doua, SPORURI, evidențiază sporurile lunare primite de fiecare angajat: sporul de vechime (SporVechime), sporul pentru orele luate noaptea (SporNoapte), sporuri pentru condiții deosebite (SporCD) și sporuri diverse (AlteSpor).

Cu ajutorul valorii NULL se poate face diferență între angajații pentru care nu s-a calculat valoarea sporului pe luna curentă (și care vor avea în câmpul corespunzător valoarea NULL) și cei care nu au dreptul la un asemenea spor, adică valoarea este 0. În continuare este prezentat scriptul de creare a celor două tabele, iar în figurile 5.2 și 5.3 – conținuturile acestora.

. Listing 5.1. Script DB2/Oracle de creare a tabelelor SPORURI și PERSONAL2

```
DROP TABLE sporuri ;
DROP TABLE personal2 ;

CREATE TABLE personal2 (
    marca INTEGER CONSTRAINT pk_personal2 PRIMARY KEY,
    numepren VARCHAR2(40),
    datanast DATE,
    compart VARCHAR2(20),
    marcasef INTEGER CONSTRAINT fk_personal2
        REFERENCES personal2(marca),
    saltarifar INTEGER
);

CREATE TABLE sporuri (
    an INTEGER,
    luna INTEGER,
    marca INTEGER REFERENCES personal2 (marca),
    sporvechime INTEGER,
    spornoapte INTEGER,
    sporcd INTEGER,
    altespor INTEGER,
    PRIMARY KEY (an,luna,marca)
);
```

Datele din cele două tabele pot fi interpretate în maniera următoare: firma s-a înființat în aprilie 2000, când avea numai trei angajați; La momentul curent (iulie 2000) sunt 10 angajați.

Care sunt persoanele și lunile pentru care nu s-a calculat (nu se cunoaște) sporul pentru condiții deosebite?

Prin interogarea:

```
SELECT SPORURI.Marcă, NumePren, Compart, An, Luna
FROM PERSONAL2, SPORURI
WHERE PERSONAL2.Marcă=SPORURI.Marcă
    AND SporCD IS NULL
```

se obține situația din figura 5.4.

MARCA	NUMEPREN	DATANAST	COMPART	MARCASEF	SALTARIFAR
1	ANGAJAT 1	01-JUL-62	DIRECȚIUNE		600000
2	ANGAJAT 2	11-OCT-77	FINANCIAR	1	450000
3	ANGAJAT 3	22-AUG-62	MARKETING	1	450000
4	ANGAJAT 4		FINANCIAR	2	380000
5	ANGAJAT 5	30-APR-65	FINANCIAR	2	420000
6	ANGAJAT 6	09-NOV-65	FINANCIAR	5	3500000
7	ANGAJAT 7		FINANCIAR	5	2800000
8	ANGAJAT 8	31-DEC-60	MARKETING	3	290000
9	ANGAJAT 9	28-FEB-76	MARKETING	3	4100000
10	ANGAJAT 10	29-JAN-72	RESURSE UMANE	1	3700000

Figura 5.2. Conținutul tabelei PERSONAL2

Diviziunea relațională

Multe situații ce reclamă diviziunea relațională pot fi soluționate elegant cu ajutorul clauzelor GROUP BY și HAVING. În exemplul următor este vorba despre o intersecție „simulată” printr-un mecanism apropiat de diviziune.

In ce zile s-au vândut și produsul cu denumirea „Produs 1”, și cel cu denumirea „Produs 2”?

```
SELECT DISTINCT DataFact
  FROM PRODUSE, LINIIFACT, FACTURI
 WHERE PRODUSE.CodPr = LINIIFACT.CodPr AND
       LINIIFACT.NrFact = FACTURI.NrFact AND
       DenPr IN ('Produs 1', 'Produs 2')
 GROUP BY DataFact
 HAVING COUNT(DISTINCT LINIIFACT.CodPr) = 2
```

Pentru fiecare grup asociat unei zile de vânzări se numără câte produse, dintre *Produs 1* și *Produs 2*, au fost facturate. Funcția COUNT() din clauza HAVING poate „întoarce” maxim valoarea 2, ca în care data respectivă se încadrează în zilele căutate.

Una dintre facilitățile SQL ține de includerea în predicatul clauzei HAVING nu numai a constantelor și/sau variabilelor, ci și a altor consultări (subconsultări). Dar despre această facilitate vom vorbi în capitolul viitor.

Capitolul 5

SQL (CEVA MAI) AVANSAT

Precedentul capitol a avut drept obiectiv explicarea mecanismului de interogare a bazelor de date prin SQL. În cele ce urmează avansăm, treptat, spre ape mai adânci ale limbajului și vom atinge elemente care se apropie de lumea bună a SQL: valori nule, juncțiuni externe, structuri alternative și subconsultări.

5.1. Prelucrarea valorilor nule

Valoarea NULL a fost introdusă în primul capitol, la explicarea noțiunilor modelului relațional, ca posibilitate de reprezentare a informațiilor... inexistente sau inaplicabile⁵². Raportul Interim 75-02-09 înaintat ANSI X3 (SPARC Study Group 1975) a delimitat 14 tipuri de date incomplete ce ar putea apărea ca rezultate ale unor operații sau valori ale atributelor, printre care: depășiri ale capacitații de stocare, diviziune la zero, trunchierea sirurilor de caractere, ridicarea lui zero la puterea zero și alte erori computaționale, precum și valori necunoscute.

La popularea bazei de date, unui client nu i se cunoaște codul fiscal, unor clienți nu li se stă adresa sau telefonul. Aceste trei atribute aveau, pe una sau mai multe linii, valoarea NULL.

Așa cum a fost prezentat în capitolul 3, DB2 și Oracle permit declararea explicită a atributelor ce nu pot avea valori nule, în timp ce în VFP se pun în evidență atributele ce pot avea asemenea valori. Prezentăm câteva exemple dedicate operatorului IS NULL.

Pentru care dintre clienți nu se cunoaște adresa?

*Soluția e*asigură că se bazează pe utilizarea operatorului IS NULL, care extrage toate valorile NULL pentru un atribut:

```
SELECT *
  FROM CLIENTI
 WHERE Adresa IS NULL
```

CODCLI	DENCL	CODFISCAL	ADRESA	CODPOST	TELEFON
1002	Clerul 2 SA	R1002		6600	032-212121
1005	Clerul 5 SRL	R1005		1900	058-111111

Figura 5.1. Clienții fără adresă

Valoarea NULL nu se confundă cu valoarea 0 – pentru atributele numerice – sau cu valoarea ‘ ‘ – spațiu – pentru atributele de tip sir de caractere. Drept este că înainte de versiunile Visual, FoxPro nu avea incorporat mecanismul de tratament sistematic al

52. Vezi și [Fotache00-2].

In ce zile s-au emis mai multe facturi decât pe 2 august 2000?

Cu volumul de cunoștințe pe care-l avem, putem să redactăm o variantă elegantă, valabilă în SQL-99, DB2 și Oracle. Soluția următoare Oracle 8 obține rezultatul corect, după cum se observă în figura 4.49.

```
SELECT F1.DataFact AS Zi1, COUNT(DISTINCT F1.NrFact)
      AS Nr_Facturilor1,
     F2.DataFact AS Zi2, COUNT(DISTINCT F2.NrFact)
      AS Nr_Facturilor2
  FROM FACTURI F1, FACTURI F2
 WHERE F2.DataFact = TO_DATE('02/08/2000', 'DD/MM/YYYY')
 GROUP BY F1.DataFact, F2.DataFact
 HAVING COUNT(DISTINCT F1.NrFact) > COUNT(DISTINCT F2.NrFact)
```

ZI1	NR_FACTURILOR1	ZI2	NR_FACTURILOR2
01-AUG-00	4	02-AUG-00	2
07-AUG-00	4	02-AUG-00	2

Figura 4.53. Zilele în care s-au emis mai multe facturi decât pe 2 august

Să zăbovim asupra logicii acestei interogări. Mai întâi se efectuează produsul cartezian pentru două instanțe (F1 și F2) ale tabeliei FACTURI, eliminându-se pentru F2 liniile în care data este decât 2 august 2000. Rezultatul produsului cartezian urmat de selecție (nu este vorba nicidcum de jonecjiune):

```
SELECT *
  FROM FACTURI F1, FACTURI F2
 WHERE F2.DataFact = TO_DATE('02/08/2000', 'DD/MM/YYYY')
 ORDER BY F1.DataFact, F2.DataFact
 este cel din figura 4.54.
```

NRFAC	DATAFACT	CODCL	OBS	NRFAC	DATAFACT	CODCL	OBS
1111	01-AUG-00	1001		1115	02-AUG-00	1001	
1112	01-AUG-00	1005	Probleme cu transportul	1115	02-AUG-00	1001	
1113	01-AUG-00	1002		1115	02-AUG-00	1001	
1114	01-AUG-00	1006		1115	02-AUG-00	1001	
1114	01-AUG-00	1006		1116	02-AUG-00	1007	Pretul propus initial a fost modificat
1113	01-AUG-00	1002		1116	02-AUG-00	1007	Pretul propus initial a fost modificat
1112	01-AUG-00	1005	Probleme cu transportul	1116	02-AUG-00	1007	Pretul propus initial a fost modificat
1111	01-AUG-00	1001		1116	02-AUG-00	1007	Pretul propus initial a fost modificat
1115	02-AUG-00	1001		1115	02-AUG-00	1001	
1116	02-AUG-00	1007	Pretul propus initial a fost modificat	1116	02-AUG-00	1007	Pretul propus initial a fost modificat
1115	02-AUG-00	1001		1116	02-AUG-00	1007	Pretul propus initial a fost modificat
1116	02-AUG-00	1007	Pretul propus initial a fost modificat	1116	02-AUG-00	1001	
1117	03-AUG-00	1001		1115	02-AUG-00	1001	
1117	03-AUG-00	1001		1116	02-AUG-00	1007	Pretul propus initial a fost modificat
1118	04-AUG-00	1001		1116	02-AUG-00	1001	
1118	04-AUG-00	1001		1116	02-AUG-00	1007	Pretul propus initial a fost modificat
1119	07-AUG-00	1003		1115	02-AUG-00	1001	
1120	07-AUG-00	1001		1115	02-AUG-00	1001	
1119	07-AUG-00	1003		1116	02-AUG-00	1007	Pretul propus initial a fost modificat
1120	07-AUG-00	1001		1116	02-AUG-00	1007	Pretul propus initial a fost modificat
1122	07-AUG-00	1005		1116	02-AUG-00	1007	Pretul propus initial a fost modificat
1121	07-AUG-00	1004		1116	02-AUG-00	1007	Pretul propus initial a fost modificat
1121	07-AUG-00	1004		1115	02-AUG-00	1001	
1122	07-AUG-00	1005		1115	02-AUG-00	1001	

Figura 4.54. Rezultatul produsului cartezian pentru F2.DataFact='02/08/2000'

În pasul următor se constituie grupuri pentru fiecare combinație de valori (F1.DataFact, F2.DataFact). F2.DataFact are aceeași valoare, 02/08/2000, prin urmare grupurile se constituie în funcție de F1.DataFact. Primul grup acoperă primele 8 linii, în care F1.DataFact este 01/08/2000, al doilea 4 linii și.a.m.d. Pentru ca funcția COUNT să calculeze corect numărul fakturilor dintr-o zi, este necesară clauza DISTINCT.

Care sunt județele în care volumul vânzărilor este superior celui al județului Neamț?

Ideea rezolvării este preluată din exemplul anterior. Problema cea mai dificilă jine de luarea în calcul a singură dată a fiecărei linii dintr-o factură. Cum, la un moment dat, pot apărea în două facturi diferite liniile în care cantitățile și prețurile unitare să fie identice, am introduce, ca artificiu, împărțirea numărului facturii la el însuși și a numărului de linie la el însuși. Varianta următoare funcționează în Oracle 8:

```
SELECT J1.Judet, J2.Judet,
       SUM(DISTINCT (LF1.NrFact/LF1.NrFact) *
              (LF1.Linie/LF1.Linie) *
              LF1.Cantitate * LF1.PretUnit *
              (1 + P1.ProcTVA))
  AS Vinzari1,
       SUM(DISTINCT (LF2.NrFact/LF2.NrFact) *
              (LF2.Linie / LF2.Linie) *
              LF2.Cantitate.* LF2.PretUnit *
              (1 + P2.ProcTVA))
  AS Vinzari2
 FROM JUDETE J1, LOCALITATI L1,
       CLIENTI C1, FACTURI F1,
       LINIIIFACT LF1, PRODUSE P1,
       JUDETE J2, LOCALITATI L2,
       CLIENTI C2, FACTURI F2,
       LINIIIFACT LF2, PRODUSE P2
 WHERE J1.Jud=L1.Jud AND L1.CodPost=C1.CodPost AND
       C1.CodCl=F1.CodCl AND F1.NrFact=LF1.NrFact AND
       LF1.CodPr=P1.CodPr AND J1.Judet='Neamt' AND
       J2.Jud=L2.Jud AND L2.CodPost=C2.CodPost AND
       C2.CodCl=F2.CodCl AND F2.NrFact=LF2.NrFact AND
       LF2.CodPr=P2.CodPr
 GROUP BY J1.Judet, J2.Judet
 HAVING
       SUM(DISTINCT (LF1.NrFact/LF1.NrFact)*
              (LF1.Linie/LF1.Linie) *
              LF1.Cantitate * LF1.PretUnit * (1 + P1.ProcTVA))
       <
       SUM(DISTINCT (LF2.NrFact/LF2.NrFact) *
              (LF2.Linie / LF2.Linie) *
              LF2.Cantitate * LF2.PretUnit *
              (1 + P2.ProcTVA))
```

Denumireclient	Datafact	Vinzari
Client 1 SRL	01/08/2000	5535760.00
Client 1 SRL	02/08/2000	1692750.00
Client 1 SRL	03/08/2000	2379000.00
Client 1 SRL	04/08/2000	2104500.00
Client 1 SRL	07/08/2000	1093120.00
Client 1 SRL-Subtotal	//	12805120.00
Client 2 SA	01/08/2000	1189500.00
Client 2 SA-Subtotal	//	1189500.00
Client 3 SRL	07/08/2000	7425530.00
Client 3 SRL-Subtotal	//	7425530.00
Client 4	07/08/2000	5575400.00
Client 4-Subtotal	//	5575400.00
Client 5 SRL	01/08/2000	1371280.00
Client 5 SRL-Subtotal	//	1371280.00
Client 6 SA	01/08/2000	6957660.00
Client 6 SA-Subtotal	//	6957660.00
Client 7 SRL	02/08/2000	1418250.00
Client 7 SRL-Subtotal	//	1418250.00
TOTAL GENERAL	//	36742740.00

Figura 4.48. Vânzări pe clienți și zile, cu subtotaluri și total general – soluție VFP

Care este cea mai mare valoare a unei facturi?

Accastă interrogare este prea pretențioasă pentru nivelul de SQL la care ne situăm în prezent. Cu toate acestea, Oracle prezintă o facilitate grozavă prin care putem redacta un răspuns: includerea unei funcții în altă (rezultatul este prezentat în figura 4.49). Din păcate, nici DB2, nici VFP nu „suportă” această facilitate:

```
SELECT MAX(SUM(Cantitate * PretUnit * (1+ProcTVA)))
AS VaxMaxFact
FROM LINIIFACT LF, PRODUSE P
WHERE P.CodPr = LF.CodPr
GROUP BY NrFact
```

VAXMAXFACT
7242935

Figura 4.49. Valoarea maximă a unei facturi

Clauza HAVING

Cea mai simplă definiție: clauza HAVING este WHERE-ul ce operează la nivel de grupuri. Dacă WHERE acționează la nivel de tuplu, selectând acele linii care îndeplinesc o condiție specificată, HAVING permite specificarea unor condiții de selecție care se aplică grupurilor de linii create prin clauza GROUP BY.

Din rezultat sunt eliminate toate grupurile care nu satisfac condiția specificată. Formatul general este:

```
SELECT coloană 1, coloană 2, ..., coloană m
FROM tabelă
GROUP BY coloană-de-regrupare
HAVING caracteristică-de-grup
```

Care sunt zilele în care s-au întocmit cel puțin trei facturi?

```
SELECT DataFact, COUNT(*) AS Nr_Facturi
FROM FACTURI
GROUP BY DataFact
HAVING COUNT(*) >= 3
```

DATAFACT	NR_FACTURI
01-08-2000	4
07-08-2000	4

Figura 4.50. Zilele în care s-au întocmit trei sau mai multe facturi

Rezultatul este cel din figura 4.50. Logica operațiunilor este prezentată în figura 4.51.

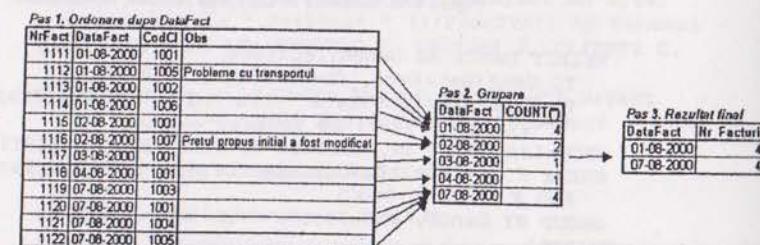


Figura 4.51. Logica execuției clauzelor GROUP BY și HAVING

Care sunt clienții pentru care vânzările depășesc 5 milioane (lei)?

```
SELECT DenCl AS Client,
SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari
FROM LINIIFACT LF, PRODUSE P, FACTURI F, CLIENTI C
WHERE P.CodPr = LF.CodPr AND LF.NrFact = F.NrFact
AND F.CodCl=C.CodCl
GROUP BY DenCl
HAVING SUM(Cantitate * PretUnit * (1+ProcTVA)) > 5000000
```

CLIENT	VINZARI
Client 1 SRL	12490240
Client 3 SRL	7242935
Client 4	5436300
Client 6 SA	6786570

Figura 4.52. Clienții cu achiziții de peste 5 milioane

PRODUS	REGIUNE	ZIUA	VINZARI
Produs 1	Moldova	01-AUG-00	595000
Produs 1	Moldova	03-AUG-00	1130500
Produs 1	Moldova	04-AUG-00	1660050
Produs 2	Banet	01-AUG-00	980560
Produs 2	Banet	02-AUG-00	1383375
Produs 2	Moldova	01-AUG-00	2988685
Produs 2	Moldova	02-AUG-00	1651125
Produs 2	Moldova	03-AUG-00	1190000
Produs 2	Moldova	04-AUG-00	392700
Produs 2	Moldova	07-AUG-00	2769725
Produs 3	Banet	01-AUG-00	357000
Produs 3	Moldova	07-AUG-00	333200
Produs 4	Moldova	01-AUG-00	564060
Produs 4	Moldova	07-AUG-00	833000
Produs 5	Moldova	01-AUG-00	9198700
Produs 5	Moldova	07-AUG-00	9811550

Figura 4.46. Gruparea vânzărilor pe produse, regiuni și zile

```

SELECT DenCl AS DenumireClient,
       TO_CHAR(DataFact,'DD-MM-YYYY') AS Data,
       TO_CHAR(SUM(Cantitate * PretUnit * (1+ProcTVA)),
               '999999999999') AS Vinzari
  FROM LINIIFACT LF, PRODUSE P, FACTURI F, CLIENTI C
 WHERE P.CodPr = LF.CodPr AND LF.NrFact = F.NrFact
   AND F.CodCl=C.CodCl
 GROUP BY DenCl, DataFact
 UNION
SELECT DenCl ||' - Subtotal' , NULL,
       TO_CHAR(SUM(Cantitate * PretUnit * (1+ProcTVA)),
               '999999999999')
  FROM LINIIFACT LF, PRODUSE P, FACTURI F, CLIENTI C
 WHERE P.CodPr = LF.CodPr AND LF.NrFact = F.NrFact
   AND F.CodCl=C.CodCl
 GROUP BY DenCl
 UNION
SELECT CHR(255)||'TOTAL GENERAL' , NULL,
       TO_CHAR(SUM(Cantitate * PretUnit * (1+ProcTVA)),
               '999999999999')
  FROM LINIIFACT LF, PRODUSE P
 WHERE P.CodPr = LF.CodPr

```

Cu modificările de rigoare, soluția este asemănătoare și în DB2. În VFP trebuie să se ia în considerare că dimensiunea unei coloane de tip sir de caractere depinde de rezultatul de numărul de caractere de pe prima linie a rezultatului. Astfel încât pentru denumirea clientului mai trebuie asigurat spațiu pentru cuvântul *Subtotal*, utilizându-se în acest scop funcția PADR().

DENUMIRECLIENT	DATA	VINZARI
Client 1 SRL	01-08-2000	5399625
Client 1 SRL	02-08-2000	1651125
Client 1 SRL	03-08-2000	2320500
Client 1 SRL	04-08-2000	2052750
Client 1 SRL	07-08-2000	1066240
Client 1 SRL - Subtotal		12490240
Client 2 SA	01-08-2000	1160250
Client 2 SA - Subtotal		1160250
Client 3 SRL	07-08-2000	7242935
Client 3 SRL - Subtotal		7242935
Client 4	07-08-2000	5438300
Client 4 - Subtotal		5438300
Client 5 SRL	01-08-2000	1337560
Client 5 SRL - Subtotal		1337560
Client 6 SA	01-08-2000	6786570
Client 6 SA - Subtotal		6786570
Client 7 SRL	02-08-2000	1363375
Client 7 SRL - Subtotal		1363375
YTOTAL GENERAL		35939230

Figura 4.47. Vânzările grupate pe clienți și zile, cu subtotaluri și total general – soluție Oracle

```

SELECT PADR(ALLT(DenCl),40) AS DenumireClient, ;
       DataFact, ;
       SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari ;
  FROM LINIIFACT LF, PRODUSE P, FACTURI F, CLIENTI C ;
 WHERE P.CodPr = LF.CodPr AND LF.NrFact = F.NrFact ;
   AND F.CodCl=C.CodCl ;
 GROUP BY DenCl, DataFact ;
 UNION ;
SELECT PADR(ALLT(DenCl)+'-Subtotal',40), ;
       ( / / ), ;
       SUM(Cantitate * PretUnit * (1+ProcTVA)) ;
  FROM LINIIFACT LF, PRODUSE P, FACTURI F, CLIENTI C ;
 WHERE P.CodPr = LF.CodPr AND LF.NrFact = F.NrFact ;
   AND F.CodCl=C.CodCl ;
 GROUP BY DenCl ;
 UNION ;
SELECT CHR(255)+'TOTAL GENERAL' , ;
       ( / / ), ;
       SUM(Cantitate * PretUnit * (1+ProcTVA)) ;
  FROM LINIIFACT LF, PRODUSE P ;
 WHERE P.CodPr = LF.CodPr

```

Rezultatul interogării este afișat în figura 4.48.

DENPR	UM	CANTITATIV	VALORIC
Produs 1	buc	300	3385550
Produs 2	kg	945	11356170
Produs 3	kg	80	690200
Produs 4	l	80	1397060
Produs 5	buc	2500	19010250

Figura 4.43. Includerea în rezultat a unității de măsură

Care este situația vânzărilor pe clienți și zile?

Interesează valoarea facturilor emise pe clienți și, pentru fiecare client, cantumul zilnic al acestora. Este momentul să folosim o veritabilă grupare după două attribute (spre deosebire de exemplul precedent, când gruparea celor două attribute a fost mai mult de nevoie, decât de voie).

```
SELECT DenCl AS DenumireClient, DataFact AS Data,
       SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari
  FROM LINIIFACT LF, PRODUSE P, FACTURI F, CLIENTI C
 WHERE P.CodPr = LF.CodPr AND LF.NrFact = F.NrFact
   AND F.CodCl = C.CodCl
 GROUP BY DenCl, DataFact
```

DENUMIRECLIENT	DATA	VINZARI
Client 1 SRL	01-AUG-00	5399625
Client 1 SRL	02-AUG-00	1651125
Client 1 SRL	03-AUG-00	2320500
Client 1 SRL	04-AUG-00	2052750
Client 1 SRL	07-AUG-00	1066240
Client 2 SA	01-AUG-00	1160250
Client 3 SRL	07-AUG-00	7242935
Client 4	07-AUG-00	5438300
Client 5 SRL	01-AUG-00	1337560
Client 6 SA	01-AUG-00	6786570
Client 7 SRL	02-AUG-00	1383375

Figura 4.44. Vânzările zilnice pentru fiecare client

Care este situația vânzărilor fiecărui produs pe fiecare regiune?

```
SELECT DenPr AS Produs, Regiune,
       SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari
  FROM LINIIFACT LF, PRODUSE P, FACTURI F, CLIENTI C,
       LOCALITATI L, JUDETE J
 WHERE P.CodPr = LF.CodPr AND LF.NrFact = F.NrFact
   AND F.CodCl=C.CodCl AND C.CodPost=L.CodPost AND
   L.Jud=J.Jud
 GROUP BY DenPr, Regiune
```

PRODUS	REGIUNE	VINZARI
Produs 1	Moldova	3385550
Produs 2	Banat	2363935
Produs 2	Moldova	8992235
Produs 3	Banat	357000
Produs 3	Moldova	333200
Produs 4	Moldova	1397060
Produs 5	Moldova	19010250

Figura 4.45. Vânzările pe regiuni ale fiecărui produs

Să se obțină situația vânzărilor pe produse, regiuni și zile.

Exemplul de față este o bună ocazie de a folosi gruparea după trei attribute și, implicit, o analiză a vânzărilor pe cele axe tradiționale: sortimentală, teritorială și calendaristică.

```
SELECT DenPr AS Produs, Regiune, DataFact AS Ziua,
       SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari
  FROM LINIIFACT LF, PRODUSE P, FACTURI F, CLIENTI C,
       LOCALITATI L, JUDETE J
 WHERE P.CodPr = LF.CodPr AND LF.NrFact = F.NrFact
   AND F.CodCl=C.CodCl AND C.CodPost=L.CodPost
   AND L.Jud=J.Jud
 GROUP BY DenPr, Regiune, DataFact
```

Rezultatul e ceva mai impresionant – vezi figura 4.46.

Fără îndoială, interogările de mai sus constituie un ajutor esențial în analiza vânzărilor oricărei firme (chiar și de stat). Ceea ce lipsește este un ingredient important al oricărui raport de acest gen – subtotalul. Vom recurge, așa cum v-ați obișnuit, la improvizații. Reluăm una dintre problemele anterioare, modificându-i ușor enunțul:

Să se obțină situația vânzărilor pe clienți și zile, afișându-se căte un subtotal la nivel de client și un total general.

Ideea improvizației este să „alipim”, pentru fiecare client, după vânzările zilnice, căte o linie care să conțină subtotalul. Iată varianta Oracle și rezultatul din figura 4.47.

Rezultatul este cel din figura 4.39.

DATAFACT	VALTOTALA
01-AUG-00	14664005
02-AUG-00	3034500
03-AUG-00	2320500
04-AUG-00	2052750
07-AUG-00	13747475

Figura 4.39. Vândările zilnice

Care sunt numărul de facturi și valoarea vânzărilor pentru fiecare client?

```
SELECT DenCl, COUNT(DISTINCT F.NrFact) AS NrFacturilor,
       SUM(Cantitate * PretUnit * (1+ProcTVA)) AS ValTotala
  FROM LINIIFACT LF, PRODUSE P, FACTURI F, CLIENTI C
 WHERE LF.CodPr = P.CodPr AND LF.NrFact = F.NrFact
   AND F.CodCl = C.CodCl
 GROUP BY DenCl
```

Analizând rezultatul din figura 4.40 și comparându-l cu tabela FACTURI, se observă că al cincilea client ar trebui să aibă două facturi, nu una. Această anomalie aparentă se datorează faptului că factura 1122 nu are nici o linie corespondentă în LINIIFACT, astfel încât această factură „cade” la joncțiune.

DENCL	NRFACTURILOR	VALTOTALA
Client 1 SRL	5	12490240
Client 2 SA	1	1160250
Client 3 SRL	1	7242935
Client 4	1	5438300
Client 5 SRL	1	1337550
Client 6 SA	1	6786570
Client 7 SRL	1	1383375

Figura 4.40. Numărul facturilor și valoarea vânzărilor pe clienți

Care este restul de incasat al fiecărei facturi?

Așa cum reiese din figura 4.41, soluția următoare oferă un rezultat incomplet. Lipsesc facturile care nu au nici o tranșă de incasare. Pentru rezolvarea cazului, avem nevoie de clauza HAVING, dar și de joncțiunea externă, după cum vom vedea în capitolul următor.

```
SELECT F.NrFact,
       SUM(Cantitate * PretUnit * (1+ProcTVA)) /
       COUNT(DISTINCT I.CodInc) AS Facturat,
       SUM(Transa) / MAX(LF.Linie) AS Incasat
  FROM LINIIFACT LF, PRODUSE P, FACTURI F, INCASFACT I
 WHERE P.CodPr = LF.CodPr AND LF.NrFact = F.NrFact
   AND F.NrFact=I.NrFact
 GROUP BY F.NrFact
```

NRFACT	FACTURAT	INCASAT
1111	5399625	5399625
1112	1337550	487705
1113	1160250	1160250
1117	2320500	2320500
1118	2052750	2052750
1120	1066240	731557

Figura 4.41. Valorile facturată și incasată pentru fiecare factură – rezultat incomplet

Care sunt vânzările, cantitativ și valoric, pentru fiecare produs?

```
SELECT DenPr,
       SUM(Cantitate) AS Cantitativ,
       SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Valoric
  FROM LINIIFACT LF, PRODUSE P, FACTURI F
 WHERE P.CodPr = LF.CodPr AND LF.NrFact = F.NrFact
 GROUP BY DenPr
```

Rezultatul – figura 4.42.

DENPR	CANTITATIV	VALORIC
Produs 1	300	3385550
Produs 2	945	11355170
Produs 3	80	690200
Produs 4	80	1397060
Produs 5	2500	19010250

Figura 4.42. Vânzările cantitative și valorice pe produse

O informație esențială care lipsește din figura 4.42 este unitatea de măsură, fără de care nu putem fi siguri dacă totalul cantitativ se referă la cutii, sticle, pachete, baxuri etc. Din păcate, varianta:

```
SELECT DenPr, UM,
       SUM(Cantitate) AS Cantitativ,
       SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Valoric
  FROM LINIIFACT LF, PRODUSE P, FACTURI F
 WHERE P.CodPr = LF.CodPr AND LF.NrFact = F.NrFact
 GROUP BY DenPr
```

nu funcționează, deoarece UM apare separat de atributul de grupare, nefiind inclus în funcție/funcțiile care se execută la nivelul grupului. Există însă un remediu simplu: includerea în clauza de grupare și a atributului UM. Altintinderi, gruparea este identică, deoarece DenPr este cheie candidată a tabelui PRODUSE, lucru observabil în figura 4.43.

```
SELECT DenPr, UM,
       SUM(Cantitate) AS Cantitativ,
       SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Valoric
  FROM LINIIFACT LF, PRODUSE P, FACTURI F
 WHERE P.CodPr = LF.CodPr AND LF.NrFact = F.NrFact
 GROUP BY DenPr, UM
```

• VFP:

```
SELECT MAX('Pret maxim='+STR(PretUnit,12)+', ;  
factura'+STR(NrFact,8)) AS Max_Pret_Produs_2, ;  
MIN('Pret minim='+STR(PretUnit,12)+', ;  
factura'+STR(NrFact,8)) AS Min_Pret_Produs_2 ;  
FROM LINIIFACT LF, PRODUSE P  
WHERE LF.CodPr = P.CodPr AND DenPr = 'Produs 2'
```

Care sunt cele mai mari două prețuri unitare (fără TVA) la care a fost vândut „Produs 2”?

După cum vedeați, o căutăm cu lumenarea. Iată soluția Oracle:

```
SELECT 'Produs 2: '||  
MAX('Primul PU: '||TO_CHAR(LF1.PretUnit,'99999999999')||  
' , al doilea PU: '||TO_CHAR(LF2.PretUnit,'99999999999')||  
' - factura primului:'||LF1.NrFact||  
' , factura-al doilea:'||LF2.NrFact)  
AS "Cele mai mari două PretUnit"  
FROM LINIIFACT LF1, LINIIFACT LF2, PRODUSE P  
WHERE LF1.CodPr = P.CodPr AND DenPr = 'Produs 2'  
AND LF1.CodPr = LF2.CodPr  
AND LF1.PretUnit > LF2.PretUnit
```

Pentru același calapod se redactează soluțiile DB2 și VFP, a căror sintaxă depinde regulile fiecărui SGBD de conversie între diferite tipuri de date.

Care este situația incasării facturii 1120?

A sosit momentul rezolvării problemei VFP care nu permite mai mulți DISTINCT pe un nivel de interogare. Soluția este foarte simplă. În loc să numărăm valorile distincte ale atributului Linie în LINIIFACT, determinăm maximul acestui atribut pentru factura luată în considerare. Obținem astfel varianta VFP, care, firește, funcționează și în celelalte două SGBD:

```
SELECT '1120' AS NrFact,  
SUM(Cantitate * PretUnit * (1+ProcTVA)) /  
COUNT(DISTINCT I.CodInc) AS Facturat,  
SUM(Transa) / MAX(LF.Linie) AS Incasat  
FROM LINIIFACT LF, PRODUSE P, FACTURI F, INCASFAT I  
WHERE LF.CodPr = P.CodPr AND LF.NrFact = F.NrFact  
AND F.NrFact=I.NrFact  
AND F.NrFact = 1120
```

4.6. Gruparea tuplurilor. GROUP BY și HAVING

Clauza GROUP BY formează grupe (grupuri) de tupluri ale unei relații, pe baza valorilor comune ale unui atribut. În frazele SELECT formulate până în acest paragraf, prin intermediul WHERE au fost selectate tupluri ale tablei. Prin asocierea unei clauze HAVING la GROUP BY este posibilă selectarea anumitor grupuri de tupluri ce îndeplinesc un criteriu, valabil numai la nivel de grup (nu și la nivel de linie).

Clauza GROUP BY

Rezultatul unei fraze SELECT ce conține această clauză se obține prin regruparea tuturor linioilor din tablelele enumerate în FROM, extrăgându-se căte o apariție pentru fiecare valoare distinctă a coloanei/grupului de coloane.

Formatul general este:

```
SELECT coloană 1, coloană 2, ..., coloană m  
FROM tabelă  
GROUP BY coloană-de-regrupare
```

Care este valoarea fără TVA a fiecărui factură emisă?

```
SELECT NrFact, SUM(Cantitate*PretUnit) as ValFaraTVA  
FROM LINIIFACT  
GROUP BY NrFact
```

Rezultatul se obține prin următoarea succesiune de operații:

1. Se ordonează linioile tablei LINIIFACT după atributul de grupare NrFact.
 2. Se constituie un grup pentru fiecare valoare distinctă a NrFact.
 3. Se execută funcția SUM(Cantitate*PretUnit) în cadrul fiecărui grup.
 4. Se obține rezultatul al căruia număr de linii coincide cu valorile distincte ale NrFact.
- Schimbul simplificăt de execuție a interogării este prezentată în figura 4.38.

Care este valoarea totală a vânzărilor pentru fiecare zi în care s-au emis facturi?

```
SELECT DataFact, SUM(Cantitate * PretUnit * (1+ProcTVA))  
AS ValTotala  
FROM LINIIFACT LF, PRODUSE P, FACTURI F  
WHERE LF.CodPr = P.CodPr AND LF.NrFact = F.NrFact  
GROUP BY DataFact
```

NrFact	Linie	CodPr	Cantitate	PretUnit
1111	1	1	50	10000
1111	2	2	75	10500
1111	3	5	500	6500
1112	1	2	80	10300
1112	2	3	40	7500
1113	1	2	100	9750
1114	1	2	70	10700
1114	2	4	30	15800
1114	3	5	700	6400
1115	1	2	150	9250
1116	1	2	125	9300
1117	1	2	100	10000
1117	2	1	100	9500
1118	1	2	30	11000
1118	2	1	150	9300
1119	1	2	35	10900
1119	2	3	40	7000
1119	3	4	50	14000
1120	4	5	750	6300
1120	1	2	80	11200
1121	1	5	550	6400
1121	2	2	100	10500

NrFact	ValFaraTVA
1111	4537500
1112	1124000
1113	975000
1114	5709000
1115	1387500
1116	1162500
1117	1950000
1118	1725000
1119	6086500
1120	8960000
1121	4570000

Figura 4.38. Schema de execuție a clauzei GROUP BY

Care este valoarea (fără TVA) medie a produselor vândute în factura 1111?

Rezultatul din figura 4.33 se obține prin interogarea:

```
SELECT 'Val. medie (fara TVA) a prod. din fact. 1111'
      AS Explicatii, AVG(Cantitate * PretUnit) AS ValMedie
  FROM LINIIFACT
 WHERE NrFact = 1111
```

EXPLICATII	VALMEDIE
Val. medie (fara TVA) a prod. din fact. 1111	1512500

Figura 4.33. Valoarea medie a produselor din factura 1111

Care este media valorilor (cu TVA) la care a fost vândut „Produs 1”?

```
SELECT 'Val. medie a vinzarilor prod. Produs 1'
      AS Explicatii,
      ROUND(AVG(Cantitate*PretUnit*(1+ProcTVA)),2) ValTotMedie
  FROM LINIIFACT LF, PRODUSE P, FACTURI F
 WHERE LF.CodPr = P.CodPr AND LF.NrFact = F.NrFact
       AND DenPr = 'Produs 1'
```

Adesea, prin aplicarea funcției AVG sau prin formularea unor expresii ce conțin rapoarte între atribute/constante, se obțin rezultate cu numeroase zecimale. În cazul de față, pentru prezentarea acestui disconfort vizual și intelectual, a fost preferată funcția ROUND, ce rotunjește media la două zecimale – vezi figura 4.34.

EXPLICATII	VALTOTMEDIE
Val. medie a vinzarilor prod. Produs 1	1126516.67

Figura 4.34. Funcția AVG combinată cu ROUND

Funcțiile MAX și MIN

Deosebit de utile în diverse tipuri de analiză, cele două funcții determină valoarea maximă, respectiv minimă, pentru o coloană (atribut). Se pot folosi și pentru atribute de tip sir de caractere, caz în care elementul de comparație este codul ASCII al caracterelor.

Care este localitatea cu ultima denumire, în ordine alfabetică?

```
SELECT MAX(Loc) AS UltimaLoc
  FROM LOCALITATI
```

ULTIMALOC
Vestul

Figura 4.35. Ultima localitate (alfabetic) din baza de date

Care este primul client și ultimul client (în ordinea numelui) din județul Iași?

```
SELECT MIN(DenC1) AS Primal_Client,
      MAX(DenC1) AS Ultimul_Client
```

```
FROM CLIENTI C, LOCALITATI L, JUDETE J
 WHERE C.CodPost = L.CodPost AND L.Jud = J.Jud
       AND Judet = 'Iași'
```

PRIMUL_CLIENT	ULTIMUL_CLIENT
Client 1 SRL	Client 4

Figura 4.36. Primul și ultimul client din județul Iași

Care este cel mai mare preț unitar (fără TVA) la care a fost vândut „Produs 1”?

```
SELECT MAX(PretUnit)
  FROM LINIIFACT LF, PRODUSE P
 WHERE LF.CodPr = P.CodPr AND DenPr = 'Produs 1'
```

Din păcate, dacă dorim să aflăm și în ce factură produsul are prețul unitar maxim, soluția:

```
SELECT MAX(PretUnit), NrFact
  FROM LINIIFACT LF, PRODUSE P
 WHERE LF.CodPr = P.CodPr AND DenPr = 'Produs 1'
       nu funcționează!
```

Până la subconsultările din capitolul viitor incercăm o soluție gen „cârpeală”, concatenând prețul cu numărul facturii. Varianta următoare este valabilă pentru Oracle:

```
SELECT 'Preț maxim='||MAX(TO_CHAR(LF.PretUnit,'99999999'))||
      ', apare în fact.'||NrFact) AS Produs1
  FROM LINIIFACT LF, PRODUSE P
 WHERE LF.CodPr = P.CodPr AND DenPr = 'Produs 1'
```

Interrogarea este operațională, cel puțin dacă ne luăm după rezultatul din figura 4.37, așa că n-are de ce să ne fie jenă de improvizație. Rezultatul este incomplet însă, atunci când prețul maxim apare în două sau mai multe фактури, deoarece interogarea de mai sus extrage numai una dintre фактуri (cea cu numărul cel mai mare).

PRODUS1
Pret maxim= 10000, apare în fact.1111

Figura 4.37. Prețul maxim pentru „Produs 1” și factură în care apare acest preț

Care este cel mai mare și cel mai mic preț unitar (fără TVA) la care a fost vândut „Produs 2”?

Dacă în ultimele exemple am pedalat pe Oracle, iată acum două variante DB2 și VFP.

• DB2:

```
SELECT MAX( 'Preț maxim=' CONCAT CAST (PretUnit AS CHAR(15))
          CONCAT ' , factura' CONCAT CAST (NrFact AS CHAR(10)))
          AS Max_Pret_Produs_2,
      MIN( 'Preț minim=' CONCAT CAST (PretUnit AS CHAR(15))
          CONCAT ' , factura' CONCAT CAST (NrFact AS CHAR(10)))
          AS Min_Pret_Produs_2
  FROM LINIIFACT LF, PRODUSE P
 WHERE LF.CodPr = P.CodPr AND DenPr = 'Produs 2'
```

```
FROM LINIIFACT LF, PRODUSE P, FACTURI F, INCASFACT I
WHERE LF.CodPr = P.CodPr AND LF.NrFact = F.NrFact
AND F.NrFact=I.NrFact
AND F.NrFact = 1111
```

va genera rezultatul din figura 4.28.

NRFACT	FATURAT	INCASAT
1111	5399625	16198875

Figura 4.28. Valoarea facturată și valoarea incasată (greșită!) pentru factura 1111

Valoarea incasată e complet anapoda, cam de trei ori mai mare decât cea facturată. Care e explicația? Răspunsul e mai ușor de aflat dacă vizualizăm rezultatul unei interogări „ajutătoare” ce generează tabela pe care se aplică cele două funcții SUM – figura 4.29.

```
SELECT DenPr, Cantitate, PretUnit,
Cantitate * PretUnit * (1+ProcTVA) AS Facturat,
Transa
FROM LINIIFACT LF, PRODUSE P, FACTURI F, INCASFACT I
WHERE LF.CodPr = P.CodPr AND LF.NrFact = F.NrFact
AND F.NrFact=I.NrFact
AND F.NrFact = 1111
```

DENPR	CANTITATE	PRETUNIT	FATURAT	TRANSA
Produs 1	50	10000	595000	5399625
Produs 2	75	10500	937125	5399625
Produs 5	500	6500	3867500	5399625

Figura 4.29. Rezultatul joncțiunii „componentelor” facturare și incasare pentru factura 1111

Întâmplarea face ca pentru această factură să existe o singură tranșă de incasare. Tranșă de 5.399.625 lei se repetă pentru fiecare linie a facturii 1111. Normal că funcția SUM întoarce o valoare triplă față de cea reală.

Ce-i de făcut? Să încercăm un artificiu. Împărțim suma tranșelor incasate la numărul liniilor facturii:

```
SELECT '1111' AS NrFact,
SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Facturat,
SUM(Transa)/COUNT(*) AS Incasat
FROM LINIIFACT LF, PRODUSE P, FACTURI F, INCASFACT I
WHERE LF.CodPr = P.CodPr AND LF.NrFact = F.NrFact
AND F.NrFact=I.NrFact
AND F.NrFact = 1111
```

Merge! Sau, cel puțin, așa arată figura 4.30.

NRFACT	FATURAT	INCASAT
1111	5399625	5399625

Figura 4.30. Facturarea și incasarea – factura 1111

Rămâne însă o sursă de crispare: ce se întâmplă cu facturile incasate în mai multe tranșe? Spre exemplu, factura 1117 are două linii și e incasată în trei tranșe. Dacă utilizăm interogarea „ajutătoare” (penultima), schimbând, bineînțeles, numărul facturii, echivalenta tabeliei din figura 4.29 se prezintă ca în figura 4.31.

DENPR	CANTITATE	PRETUNIT	FATURAT	TRANSA
Produs 2	100	10000	1190000	975410
Produs 1	100	9500	1130500	975410
Produs 2	100	10000	1190000	975410
Produs 1	100	9500	1130500	975410
Produs 2	100	10000	1190000	369680
Produs 1	100	9500	1130500	369680

Figura 4.31. Valori repetitive datorită mai multor tranșe de incasare – factura 1117

Există nu mai puțin de 3 (tranșe de incasare – INCASFACT) * 2 (linii în LINIIFACT) = 6 linii. Pe baza acestei observații ochiometrice, putem să încercăm o soluție finală, împărțind valoarea facturată la numărul tranșelor de incasare și valoarea incasată la numărul de linii ale facturii respective:

```
SELECT '1117' AS NrFact,
SUM(Cantitate * PretUnit * (1+ProcTVA)) /
COUNT(DISTINCT I.CodInc) AS Facturat,
SUM(Transa) / COUNT(DISTINCT(LF.Linie)) AS Incasat
FROM LINIIFACT LF, PRODUSE P, FACTURI F, INCASFACT I
WHERE LF.CodPr = P.CodPr AND LF.NrFact = F.NrFact
AND F.NrFact=I.NrFact
AND F.NrFact = 1117
```

NRFACT	FATURAT	INCASAT
1117	2320500	2320500

Figura 4.32. Valorile (corecte) factură și incasată – factura 1117

Rezultatul din figura 4.32 ne îndreptățește să sperăm că soluția e bună. Și, într-adevăr, este funcțională, dar numai pentru facturile care au cel puțin o tranșă de incasare! Pentru cealaltă categorie avem nevoie de cunoștințe ceva mai avansate (joncțiune externă).

Visual FoxPro nu ne dă, însă, pace. La execuția acestei variante apare ușurătorul mesaj de eroare: SQL: DISTINCT is invalid, eroare ce are numărul 1819 și se datorează, după cum scrie „la documentație”, faptului că se poate utiliza un singur DISTINCT pe nivel. Vom putea remedia situația folosind funcția MAX.

Funcția AVG

Ultimele exemple discutate se doresc o trecere lină la prezentul operator care, după cum îl spune și numele (în engleză), calculează media aritmetică a unei coloane într-o tabelă oarecare, prin divizarea sumei valorilor coloanei respective la numărul de valori nenele ale acesteia.

Dacă se dorește afișarea pe verticală, sub formă tabelară, a celor trei valori, se pot folosi şablonane (ce diferă de la SGBD la SGBD) împreună cu operatorul reunire, astfel încât rezultatul va arăta ca în figura 4.24. Am uitat să vă spun că soluția e valabilă în sintaxa de mai jos tot pentru Oracle 8.

```
SELECT '1' AS X,'Pentru factura 11111' AS TipValoare,
      '' AS Suma
FROM dual
UNION
SELECT '2','Valoarea fara TVA',
      TO_CHAR(SUM(Cantitate * PretUnit),'999999999')
FROM LINIIFACT LF, PRODUSE P
WHERE LF.CodPr = P.CodPr AND NrFact = 1111
UNION
SELECT '3','TVA',
      TO_CHAR(SUM(Cantitate * PretUnit * ProcTVA), '999999999')
FROM LINIIFACT LF, PRODUSE P
WHERE LF.CodPr = p.CodPr AND NrFact = 1111
UNION
SELECT '4','Valoarea totală',
      TO_CHAR(SUM(Cantitate*PretUnit*(1+ProcTVA)), '999999999')
FROM LINIIFACT LF, PRODUSE P
WHERE LF.CodPr = p.CodPr AND NrFact = 1111
```

X	TIPVALOARE	SUMA
1	Pentru factura 11111	
2	Valoarea fara TVA	4537500
3	TVA	862125
4	Valoarea totală	5399625

Figura 4.24. Cele trei valori ale facturii 11111 – afișare pe verticală

Funcția TO_CHAR convertește valoarea numerică obținută prin funcția SUM într-un sir de caractere; prin utilizarea şablonului, valorile vor fi aliniate la dreapta, pentru a fi mai ușor de comparat. Dacă tot am avut ocazia, am folosit și tabela DUAL, o tabelă utilă în Oracle atunci când se dorește extragerea prin SELECT a unei constante. Tabela DUAL are o singură linie și o singură coloană, servindu-ne în acest caz pentru afișarea așa-zisului antet: *Pentru factura 11111*.

Nu întotdeauna rezultatul reflectă ordinea firească de procesare a liniilor. Este motivul pentru care am introdus în interogare o coloană (X), cu un prim rol decorativ și un al doilea legat de afișarea liniilor în ordinea care interesează.

La cât se situează cifra vânzărilor pe data de 7 august 2000?

```
SELECT '7 aug. 2000' AS Data,
      SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari
FROM LINIIFACT LF, PRODUSE P, FACTURI F
WHERE LF.CodPr = P.CodPr AND LF.NrFact = F.NrFact
      AND DataFact = '07/08/2000'
```

DATA	VINZARI
7 aug. 2000	13747475

Figura 4.25. Vânzările zilei de 7 august 2000

Care este valoarea vânzărilor pentru „Client 1 SRL”?

```
SELECT 'Client 1 SRL' AS Client,
      SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari
FROM LINIIFACT LF, PRODUSE P, FACTURI F, CLIENTI C
WHERE LF.CodPr = P.CodPr AND LF.NrFact = F.NrFact AND
      F.CodCl = C.CodCl
      AND DenCl = 'Client 1 SRL'
```

CLIENT	VINZARI
Client 1 SRL	12490240

Figura 4.26. Vânzările către Client 1 SRL

Care este valoarea medie a prețului (inclusiv TVA) la care a fost vândut „Produs 1”?

Nu, n-am trecut la funcția AVG fără să vă fi anunțat din timp. Soluția se bazează pe raportul dintre suma valorilor și cantitatea însumată pentru acest produs.

```
SELECT SUM(Cantitate*PretUnit*(1+ProcTVA)) /
      SUM(Cantitate) AS PretUnitMediu
FROM LINIIFACT LF, PRODUSE P, FACTURI F
WHERE LF.CodPr = P.CodPr AND LF.NrFact = F.NrFact
      AND DenPr = 'Produs 1'
```

Care este situația încasării facturii 1120?

Rezolvarea acestei probleme presupune afișarea, pentru factura 1120, atât a valorii totale, obținută din tabelele LINIIFACT și PRODUSE, cât și a valorii încasate, obținută din INCASFACT. Prin urmare, pare firesc să încercăm varianta:

```
SELECT '1120' AS NrFact,
      SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Facturat,
      SUM(Transa) AS Incasat
FROM LINIIFACT LF, PRODUSE P, FACTURI F, INCASFACT I
WHERE LF.CodPr = P.CodPr AND LF.NrFact = F.NrFact
      AND F.NrFact=I.NrFact
      AND F.NrFact = 1120
```

NRFAC	FACTURAT	INCASAT
1120	1066240	731557

Figura 4.27. Valorile facturată și încasată ale facturii 1120

Rezultatul din figura 4.27 arată că lucrurile ar fi în regulă. Dar dacă înlocuim numărul facturii 1120 cu 1111, fraza:

```
SELECT '1111' AS NrFact,
      SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Facturat,
      SUM(Transa) AS Incasat
```

Pentru căi clienți se cunoaște adresa?

```
SELECT COUNT (Adresa) AS NrClienti
FROM CLIENTI
```

Rezultatul, 5, putea fi obținut și ceva mai complicat, folosind în clauza WHERE operatorul IS NULL, pe care-l tot amânam pentru capitolul următor.

Câte facturi au fost emise pe 7 august 2000?

```
SELECT COUNT(NrFact) AS NrFacturi
FROM FACTURI
WHERE DataFact = '07/08/2000'
```

Câte facturi au fost emise clienților din județul Vaslui?

```
SELECT COUNT(NrFact) AS NrFacturi
FROM FACTURI F, CLIENTI C, LOCALITATI L, JUDETE J
WHERE F.CodCl = C.CodCl AND C.CodPost=L.CodPost
AND L.Jud = J.Jud AND Judet='Vaslui'
```

Nu ne mai ostenim să redactăm și forma interogării cu INNER JOIN. Rămâne ca temă pentru acasă, oricare ar fi ea.

În care localități se află clienții firmei?

Tabela LOCALITĂȚI conține și orașe/comune în care nu se află, momentan, nici un client. De aceea, în locul soluției:

```
SELECT COUNT(CodPost) AS NrLocalit
FROM LOCALITATI
```

pare mai înțelept să folosim varianta:

```
SELECT COUNT(CodPost) AS NrLocalit
FROM CLIENTI
```

Necazul că rezultatul obținut, 7, este incorect, deoarece funcția COUNT numără toate valorile nenule. Există însă o clauză prin care o valoare să se ia în calcul o singură dată: DISTINCT.

Rezultatul corect (5) presupune varianta:

```
SELECT COUNT(DISTINCT CodPost) AS NrLocalit
FROM CLIENTI
```

Funcția SUM

SUM este una dintre cele mai utilizate funcții în aplicațiile economice, deoarece datele financiar-contabile și cele ale evidenței tehnico-operative sunt preponderent cantitative. Probabil că prea multe explicații teoretice despre modul în care operează această funcție sunt inutile, așa încât vom trece în revistă câteva exemple.

Care este valoarea fără TVA a facturii 1111?

```
SELECT SUM(Cantitate * PretUnit) AS ValFaraTVA
FROM LINIIFACT
WHERE NrFact = 1111
```

Rezultatul arată ca în figura 4.21.

VALFARATVA
4537500

Figura 4.21. Valoarea fără TVA a facturii 1111

Care este valoarea fără TVA a facturilor emise pe 7 august 2000?

```
SELECT SUM(Cantitate * PretUnit) AS ValFaraTVA_17aug2000
FROM LINIIFACT LF, FACTURI F
WHERE LF.NrFact = F.NrFact AND DataFact = '07/08/2000'
```

VALFARATVA_17AUO2000
11552500

Figura 4.22. Valoarea fără TVA a facturilor emise pe 17 august 2000

Care sunt cele trei valori: fără TVA, TVA și totală ale facturii 1111?

```
SELECT SUM(Cantitate * PretUnit) AS ValFaraTVA,
SUM(Cantitate * PretUnit * ProcTVA) AS TVA,
SUM(Cantitate * PretUnit + Cantitate * PretUnit *
ProcTVA) AS ValTotala
FROM LINIIFACT LF, PRODUSE P
WHERE LF.CodPr = P.CodPr
AND NrFact = 1111
```

În condițiile actuale, în care procentul TVA este unic – 19%, este corectă și varianta:

```
SELECT SUM(Cantitate * PretUnit) AS ValFaraTVA,
SUM(Cantitate * PretUnit * .19) AS TVA,
SUM(Cantitate * PretUnit + Cantitate * PretUnit * .19)
AS ValTotala
FROM LINIIFACT
WHERE NrFact = 1111
```

Soluția propusă este una atemporală, altfel spus, a-guvernamentală și a-relaxare fiscală. O vizualizare mai elegantă a rezultatului poate fi realizată în Oracle utilizând interogarea următoare, rezultatul fiind prezentat în figura 4.23 (analog se pot croi și variantele în DB2 și VFP):

```
SELECT 'Pentru factura 1111, valoarea fără TVA este ''|| SUM(Cantitate * PretUnit)|| ''
', TVA este ''|| SUM(Cantitate * PretUnit * ProcTVA)|| ''
', iar valoarea totală este ''|| SUM(Cantitate * PretUnit + Cantitate *
PretUnit * ProcTVA) AS Rezultat
FROM LINIIFACT, PRODUSE
WHERE LINIIFACT.CodPr=PRODUSE.CodPr AND NrFact = 1111
```

REZULTAT
Pentru factura 1111, valoarea fără TVA este 4537500, TVA este 862125, iar valoarea totală este 5399625

Figura 4.23. Cele trei valori ale facturii 1111 – afișare pe orizontală

Ca piatră de încercare, revenim la a doua soluție formulată în algebra relațională la exemplul 17:

În ce zile s-au vândut și produsul cu denumirea „Produs 1”, și cel cu denumirea „Produs 2”?

Joncționăm o instanță obținută prin joncțiunea PRODUSE-LINIIFACT-FACTURI (în care DenPr = 'Produs 1') cu o altă instanță a aceleiași combinații (în care DenPr = 'Produs 2').

- Soluție 2 – varianta 1 (generală):

```
SELECT DISTINCT F1.DataFact
FROM PRODUSE P1, LINIIFACT LF1, FACTURI F1,
PRODUSE P2, LINIIFACT LF2, FACTURI F2
WHERE
    P1.CodPr = LF1.CodPr AND LF1.NrFact = F1.NrFact
    AND (P1.DenPr = 'Produs 1') AND
    P2.CodPr = LF2.CodPr AND LF2.NrFact = F2.NrFact
    AND (P2.DenPr = 'Produs 2') AND
    F1.DataFact=F2.DataFact
```

- Soluție 2 – varianta 2 (non-Oracle):

```
SELECT DISTINCT F1.DataFact
FROM PRODUSE P1
    INNER JOIN LINIIFACT LF1 ON P1.CodPr = LF1.CodPr
    INNER JOIN FACTURI F1 ON LF1.NrFact = F1.NrFact
    INNER JOIN FACTURI F2 ON F1.DataFact=F2.DataFact
    INNER JOIN LINIIFACT LF2 ON LF2.NrFact = F2.NrFact
    INNER JOIN PRODUSE P2 ON LF2.CodPr = P2.CodPr
WHERE
    P1.DenPr = 'Produs 1' AND P2.DenPr = 'Produs 2'
```

Nici nu se putea încheiere mai triumfătoare pentru aşa un paragraf glorios.

4.5. Funcții-agregat: COUNT, SUM, AVG, MIN, MAX

Formatul general al unei fraze SELECT ce conține funcții-agregat este:

```
SELECT funcția-predefinită1, ... , funcția-predefinităN
FROM listă-tabele
WHERE condiții.
```

În lipsa opțiunii GROUP BY (vezi paragraful următor), dacă în clauza SELECT este prezentă o funcție-agregat, rezultatul va conține o singură linie.

Funcția COUNT

Funcția COUNT contorizează valorile nenule ale unei coloane sau numărul de linii dintr-un rezultat al unei interogări, altfel spus, în rezultatul unei consultări, COUNT numără câte valori diferite de NULL are o coloană specificată sau câte linii sunt.

Câți clienți are firma?

```
SELECT COUNT (*) AS NrClienti
FROM CLIENTI
```

Prezența asteriscului ca argument al funcției COUNT are ca efect numărarea liniilor tablei CLIENTI. Rezultatul este prezentat în figura 4.19.

NRCLIENTI
7

Figura 4.19. Câți clienți are firma ?

Folosind concatenarea, se poate obține un rezultat ceva mai elegant. Spre exemplu, în Oracle, prin interogarea următoare se obține rezultatul din figura 4.20 (la concatenare nu este necesară conversia operanzilor în siruri de caractere):

```
SELECT 'Firma are '||COUNT (*)||' clienti' AS Rezultat
FROM CLIENTI
```

REZULTAT
Firma are 7 clienti

Figura 4.20. Altă formă de prezentare (în Oracle) a rezultatului funcției COUNT

Pentru obținerea același rezultat în VFP fraza se scrie sub forma:

```
SELECT 'Firma are '+STR(COUNT(*),4)+' clienti' AS Rezultat
FROM CLIENTI
```

iar în DB2, în afara operatorului de concatenare CONCAT, este necesară conversia rezultatului funcției COUNT, care este INTEGER, în sir de caractere (CHAR):

```
SELECT 'Firma are ' CONCAT
    CAST (COUNT (*) AS CHAR(5)) CONCAT
    ' clienti' AS Rezultat
FROM CLIENTI
```

Tabela CLIENTI are cheie primară atributul CodCl, care nu poate avea valori nule; de aceea, în fel de corectă este și soluția:

```
SELECT COUNT (CodCl) AS NrClienti
FROM CLIENTI
```

Câte linii are produsul cartezian al tabelelor FACTURI și LINIIFACT?

```
SELECT COUNT(*)
FROM FACTURI, LINIIFACT
```

Acum am aflat și eu: 264.

```

SELECT DISTINCT DenCl
FROM PRODUSE P, LINIIFACT LF, FACTURI F, CLIENTI C
WHERE
P.CodPr = LF.CodPr AND LF.Nrfact = F.NrFact AND
F.CodCl = C.CodCl AND DenPr = 'Produs 2'
INTERSECT
SELECT DISTINCT DenCl
FROM PRODUSE P, LINIIFACT LF, FACTURI F, CLIENTI C
WHERE
P.CodPr = LF.CodPr AND LF.Nrfact = F.NrFact AND
F.CodCl = C.CodCl AND DenPr = 'Produs 3'
MINUS
SELECT DISTINCT DenCl
FROM PRODUSE P, LINIIFACT LF, FACTURI F, CLIENTI C
WHERE
P.CodPr = LF.CodPr AND LF.Nrfact = F.NrFact AND
F.CodCl = C.CodCl AND DenPr = 'Produs 5'

```

Tabelei PRODUSE î s-a asociat sinonimul P, LINIIFACT LF, FACTURI F, iar pentru CLIENTI C. Sinonimele prefixează (când este necesar) numele atributelor în clauzele **SELECT** și **WHERE** (eventual ORDER BY, GROUP BY).

Dacă pe calculatorul pe care procesez în acest moment documentul opțiunea WordCount funcționează rezonabil, atunci luăm de bune următoarele cifre: varianță fără sinonime numără 530 de caractere, în timp ce ultima este alcătuită din 431...

Există situații în care utilizarea sinonimelor n-are nimic de-a face cu lenea/comoditatea sau depresiile nervoase. În afara interogărilor corelate pe care le vom discuta într-un capitol viitor, o operație în care musai trebui folosite sinonimele este *jonctionarea tabeliei cu ea însăși*.

Revenim la *exemplul 19* din algebra relațională:

Ce facturi au fost emise în aceeași zi cu factura 1120?

Este, probabil, cel mai bun exemplu pentru prezentarea subconsultărilor; noi însă ne vom servi acum de acest caz spre a introduce noul tip de joncționare.

```

SELECT F2.NrFact
FROM FACTURI F1, FACTURI F2
WHERE F1.DataFact = F2.DataFact AND F1.NrFact=1120

```

Joncționarea tabelei cu ea însăși înseamnă, de fapt, joncționarea a două instanțe ale tabelei respective. Rezultatul joncționii F1 cu F2 este o tabelă „gospodărească”, după cum se observă în figura 4.17.

F1.NrFact	F1.DataFact	F1.CodCl	F1.Obs	F2.NrFact	F2.DataFact	F2.CodCl	F2.Obs
1111	1-Aug-00	1001		1111	1-Aug-00	1001	
1112	1-Aug-00	1005	Probleme cu transportul	1111	1-Aug-00	1001	
1113	1-Aug-00	1002		1111	1-Aug-00	1001	
1114	1-Aug-00	1006		1111	1-Aug-00	1001	
1111	1-Aug-00	1001		1112	1-Aug-00	1005	Probleme cu transportul
1112	1-Aug-00	1005	Probleme cu transportul	1112	1-Aug-00	1005	Probleme cu transportul
1113	1-Aug-00	1002		1112	1-Aug-00	1005	Probleme cu transportul
1114	1-Aug-00	1006		1112	1-Aug-00	1006	Probleme cu transportul
1111	1-Aug-00	1001		1113	1-Aug-00	1002	
1112	1-Aug-00	1005	Probleme cu transportul	1113	1-Aug-00	1002	
1113	1-Aug-00	1002		1113	1-Aug-00	1002	
1114	1-Aug-00	1006		1113	1-Aug-00	1002	
1111	1-Aug-00	1001		1114	1-Aug-00	1006	
1112	1-Aug-00	1005	Probleme cu transportul	1114	1-Aug-00	1006	
1113	1-Aug-00	1002		1114	1-Aug-00	1006	
1114	1-Aug-00	1006		1114	1-Aug-00	1006	
1111	1-Aug-00	1001		1115	2-Aug-00	1001	
1112	1-Aug-00	1005	Probleme cu transportul	1115	2-Aug-00	1001	
1113	1-Aug-00	1002		1115	2-Aug-00	1007	Pretul propus initial a fost modificat
1114	1-Aug-00	1006		1116	2-Aug-00	1007	Pretul propus initial a fost modificat
1111	1-Aug-00	1001		1116	2-Aug-00	1007	Pretul propus initial a fost modificat
1112	1-Aug-00	1005	Probleme cu transportul	1116	2-Aug-00	1007	Pretul propus initial a fost modificat
1113	1-Aug-00	1002		1117	3-Aug-00	1001	
1114	1-Aug-00	1006		1117	3-Aug-00	1001	
1111	1-Aug-00	1001		1118	4-Aug-00	1001	
1112	1-Aug-00	1005	Probleme cu transportul	1118	4-Aug-00	1001	
1113	1-Aug-00	1002		1119	7-Aug-00	1003	
1114	1-Aug-00	1006		1119	7-Aug-00	1003	
1115	2-Aug-00	1001		1119	7-Aug-00	1006	
1116	2-Aug-00	1007	Pretul propus initial a fost modificat	1115	2-Aug-00	1001	
1117	3-Aug-00	1001		1116	2-Aug-00	1007	Pretul propus initial a fost modificat
1118	4-Aug-00	1001		1117	3-Aug-00	1001	
1119	7-Aug-00	1003		1118	4-Aug-00	1001	
1120	7-Aug-00	1001		1119	7-Aug-00	1003	
1121	7-Aug-00	1004		1120	7-Aug-00	1001	
1122	7-Aug-00	1005		1120	7-Aug-00	1001	
1119	7-Aug-00	1003		1121	7-Aug-00	1001	
1120	7-Aug-00	1001		1121	7-Aug-00	1004	
1121	7-Aug-00	1004		1121	7-Aug-00	1004	
1122	7-Aug-00	1005		1121	7-Aug-00	1004	
1119	7-Aug-00	1003		1122	7-Aug-00	1005	
1120	7-Aug-00	1001		1122	7-Aug-00	1005	
1121	7-Aug-00	1004		1122	7-Aug-00	1005	
1122	7-Aug-00	1005		1122	7-Aug-00	1005	
1119	7-Aug-00	1003		1122	7-Aug-00	1006	
1120	7-Aug-00	1001		1122	7-Aug-00	1006	
1121	7-Aug-00	1004		1122	7-Aug-00	1006	
1122	7-Aug-00	1005		1122	7-Aug-00	1006	
1119	7-Aug-00	1003		1122	7-Aug-00	1006	
1120	7-Aug-00	1001		1122	7-Aug-00	1006	
1121	7-Aug-00	1004		1122	7-Aug-00	1006	
1122	7-Aug-00	1005		1122	7-Aug-00	1006	
1119	7-Aug-00	1003		1122	7-Aug-00	1006	
1120	7-Aug-00	1001		1122	7-Aug-00	1006	
1121	7-Aug-00	1004		1122	7-Aug-00	1006	
1122	7-Aug-00	1005		1122	7-Aug-00	1006	
1119	7-Aug-00	1003		1122	7-Aug-00	1006	
1120	7-Aug-00	1001		1122	7-Aug-00	1006	
1121	7-Aug-00	1004		1122	7-Aug-00	1006	
1122	7-Aug-00	1005		1122	7-Aug-00	1006	
1119	7-Aug-00	1003		1122	7-Aug-00	1006	
1120	7-Aug-00	1001		1122	7-Aug-00	1006	
1121	7-Aug-00	1004		1122	7-Aug-00	1006	
1122	7-Aug-00	1005		1122	7-Aug-00	1006	
1119	7-Aug-00	1003		1122	7-Aug-00	1006	
1120	7-Aug-00	1001		1122	7-Aug-00	1006	
1121	7-Aug-00	1004		1122	7-Aug-00	1006	
1122	7-Aug-00	1005		1122	7-Aug-00	1006	
1119	7-Aug-00	1003		1122	7-Aug-00	1006	
1120	7-Aug-00	1001		1122	7-Aug-00	1006	
1121	7-Aug-00	1004		1122	7-Aug-00	1006	
1122	7-Aug-00	1005		1122	7-Aug-00	1006	
1119	7-Aug-00	1003		1122	7-Aug-00	1006	
1120	7-Aug-00	1001		1122	7-Aug-00	1006	
1121	7-Aug-00	1004		1122	7-Aug-00	1006	
1122	7-Aug-00	1005		1122	7-Aug-00	1006	
1119	7-Aug-00	1003		1122	7-Aug-00	1006	
1120	7-Aug-00	1001		1122	7-Aug-00	1006	
1121	7-Aug-00	1004		1122	7-Aug-00	1006	
1122	7-Aug-00	1005		1122	7-Aug-00	1006	
1119	7-Aug-00	1003		1122	7-Aug-00	1006	
1120	7-Aug-00	1001		1122	7-Aug-00	1006	
1121	7-Aug-00	1004		1122	7-Aug-00	1006	
1122	7-Aug-00	1005		1122	7-Aug-00	1006	
1119	7-Aug-00	1003		1122	7-Aug-00	1006	
1120	7-Aug-00	1001		1122	7-Aug-00	1006	
1121	7-Aug-00	1004		1122	7-Aug-00	1006	
1122	7-Aug-00	1005		1122	7-Aug-00	1006	
1119	7-Aug-00	1003		1122	7-Aug-00	1006	
1120	7-Aug-00	1001		1122	7-Aug-00	1006	
1121	7-Aug-00	1004		1122	7-Aug-00	1006	
1122	7-Aug-00	1005		1122	7-Aug-00	1006	
1119	7-Aug-00	1003		1122	7-Aug-00	1006	
1120	7-Aug-00	1001		1122	7-Aug-00	1006	
1121	7-Aug-00	1004		1122	7-Aug-00	1006	
1122	7-Aug-00	1005		1122	7-Aug-00	1006	
1119	7-Aug-00	1003		1122	7-Aug-00	1006	
1120	7-Aug-00	1001		1122	7-Aug-00	1006	
1121	7-Aug-00	1004		1122	7-Aug-00	1006	
1122	7-Aug-00	1005		1122	7-Aug-00	1006	
1119	7-Aug-00	1003		1122	7-Aug-00	1006	
1120	7-Aug-00	1001		1122	7-Aug-00	1006	
1121	7-Aug-00	1004		1122	7-Aug-00	1006	
1122	7-Aug-00	1005		1122	7-Aug-00	1006	
1119	7-Aug-00	1003		1122	7-Aug-00	1006	
1120	7-Aug-00	1001		1122	7-Aug-00	1006	
1121	7-Aug-00	1004		1122	7-Aug-00	1006	
1122	7-Aug-00	1005		1122	7-Aug-00	1006	
1119	7-Aug-00	1003		1122	7-Aug-00	1006	
1120	7-Aug-00	1001		1122	7-Aug-00	1006	
1121	7-Aug-00	1004		1122	7-Aug-00	1006	
1122	7-Aug-00	1005		1122	7-Aug-00	1006	
1119	7-Aug-00	1003		1122	7-Aug-00	1006	
1120	7-Aug-00	1001		1122	7-Aug-00	1006	
1121	7-Aug-00	1004		1122	7-Aug-00	1006	
1122	7-Aug-00	1005		1122	7-Aug-00	1006	
1119	7-Aug-00	1003		1122	7-Aug-00	1006	
1120	7-Aug-00	1001		1122	7-Aug-00	1006	
1121	7-Aug-00	1004		1122	7-Aug-00	1006	
1122	7-Aug-00	1005		1122	7-Aug-00	1006	
1119	7-Aug-00	1003		1122	7-Aug-00	1006	
1120	7-Aug-00	1001		1122	7-Aug-00	1006	
1121	7-Aug-00	1004		1122	7-Aug-00	1006	
1122	7-Aug-00	1005		1122	7-Aug-00	1006	
1119	7-Aug-00	1003		1122	7-Aug-00	1006	
1120	7-Aug-00	1001		1122	7-Aug-00	1006	
1121	7-Aug-00	1004		1122	7-Aug-00	1006	
1122	7-Aug-00	1005		1122	7-Aug-00	1006	
1119	7-Aug-00	1003		1122	7-Aug-00	1006	
1120	7-Aug-00	1001		1122	7-Aug-00	1006	
1121	7-Aug-00	1004		1122	7-Aug-00	1006	
1122	7-Aug-00	1005		1122	7-Aug-00	1006	
1119	7-Aug-00	1003		1122	7-Aug-00	1006	
1120	7-Aug-00	1001		1122	7-Aug-00	1006	
1121	7-Aug-00	1004		1122	7-Aug-00	1006	
1122	7-Aug-00	1005		1122	7-Aug-00	1006	
1119	7-Aug-00	1003		1122	7-Aug-00	1006	
1120	7-Aug-00	1001		1122	7-Aug-00	1006	
1121	7-Aug-00	1004		1122	7-Aug-00	1006	
1122	7-Aug-00	10					

```

WHERE
PRODUSE.CodPr = LINIIFACT.CodPr AND
LINIIFACT.Nrfact = FACTURI.NrFact AND
FACTURI.CodCl = CLIENTI.CodCl AND
CLIENTI.CodPost = LOCALITATI.CodPost AND
LOCALITATI.Jud = JUDETE.Jud AND
DenPr = 'Produs 1' AND
DataFact BETWEEN '03/08/2000' AND '05/08/2000'
• Varianta 2 (exclusiv Oracle – atenție la constantele de tip dată calendaristică!):

```

```

SELECT DISTINCT Judet
FROM PRODUSE
INNER JOIN LINIIFACT ON PRODUSE.CodPr = LINIIFACT.CodPr
INNER JOIN FACTURI ON LINIIFACT.Nrfact = FACTURI.NrFact
INNER JOIN CLIENTI ON FACTURI.CodCl = CLIENTI.CodCl
INNER JOIN LOCALITATI ON
    CLIENTI.CodPost = LOCALITATI.CodPost
INNER JOIN JUDETE ON
    LOCALITATI.Jud = JUDETE.Jud
WHERE DenPr = 'Produs 1' AND
DataFact BETWEEN '03/08/2000' AND '05/08/2000'

```

Și în acest exemplu este recomandată folosirea clauzei DISTINCT.

Exemplul 17

(În ce zile s-au vândut și produsul cu denumirea „Produs 1”, și cel cu denumirea „Produs 2”?)

- Soluție 1 – varianta 1 (nu funcționează în VFP):

```

SELECT DISTINCT DataFact
FROM PRODUSE, LINIIFACT, FACTURI
WHERE
PRODUSE.CodPr = LINIIFACT.CodPr AND
LINIIFACT.Nrfact = FACTURI.NrFact AND
DenPr = 'Produs 1'
INTERSECT
SELECT DISTINCT DataFact
FROM PRODUSE, LINIIFACT, FACTURI
WHERE
PRODUSE.CodPr = LINIIFACT.CodPr AND
LINIIFACT.Nrfact = FACTURI.NrFact AND
DenPr = 'Produs 2'

```

- Soluție 1 – varianta 2 (SQL-92 și DB2):

```

SELECT DISTINCT DataFact
FROM PRODUSE
INNER JOIN LINIIFACT ON PRODUSE.CodPr = LINIIFACT.CodPr
INNER JOIN FACTURI ON LINIIFACT.Nrfact = FACTURI.NrFact
WHERE DenPr = 'Produs 1'
INTERSECT
SELECT DISTINCT DataFact

```

```

FROM PRODUSE INNER JOIN LINIIFACT
ON PRODUSE.CodPr = LINIIFACT.CodPr
INNER JOIN FACTURI
ON LINIIFACT.Nrfact = FACTURI.NrFact
WHERE DenPr = 'Produs 2'

```

Pentru variantele soluției 2 avem nevoie de ceea ce se numește joncțiunea tabelei cu ea însăși, lucru pe care il vom discuta în paragraful următor.

Exemplul 18

(Ce clienți au cumpărăt și „Produs 2”, și „Produs 3”, dar nu au cumpărăt „Produs 5”?)

- Soluția Oracle (și SQL-92 și DB2, dacă se înlocuiește MINUS cu EXCEPT):

```

SELECT DISTINCT DenCl
FROM PRODUSE, LINIIFACT, FACTURI, CLIENTI
WHERE
PRODUSE.CodPr = LINIIFACT.CodPr AND
LINIIFACT.Nrfact = FACTURI.NrFact AND
FACTURI.CodCl = CLIENTI.CodCl AND
DenPr = 'Produs 2'
INTERSECT
SELECT DISTINCT DenCl
FROM PRODUSE, LINIIFACT, FACTURI, CLIENTI
WHERE
PRODUSE.CodPr = LINIIFACT.CodPr AND
LINIIFACT.Nrfact = FACTURI.NrFact AND
FACTURI.CodCl = CLIENTI.CodCl AND
DenPr = 'Produs 3'
MINUS
SELECT DISTINCT DenCl
FROM PRODUSE, LINIIFACT, FACTURI, CLIENTI
WHERE
PRODUSE.CodPr = LINIIFACT.CodPr AND
LINIIFACT.Nrfact = FACTURI.NrFact AND
FACTURI.CodCl = CLIENTI.CodCl AND
DenPr = 'Produs 5'

```

4. Sinonime locale și joncțiunea unei tabele cu ea însăși

Lucrul cu nume lungi de tabele și atribute are marele avantaj al lejerității la citire și înțelegerei rapide a logicii de derulare a interogării. În schimb, suficienți informaticieni nu agreează risipa de caractere (și, implicit, de timp și nervi) presupuse de redactările „logoreice”. Ambele părți au dreptate în oarecare măsură (sesizări sindromul rabinului!). Astfel încât în frazele SELECT, tabelelor li se pot asocia sinonime sau aliasuri mai scurte. Pentru ilustrare, ultima interogare se poate scrie astfel:

4.3. Theta- și echi-joncțiunea

Dintre tipurile de joncțiune prezentate în capitolul 2, vom insista asupra theta-joncțiunii și echi-joncțiunii. SQL nu prezintă cluze sau operatori speciali pentru joncțiune, însă așa cum am văzut, o joncțiune este o combinație de *produs cartezian* și *selecție*. În consecință, pentru theta-joncționarea relațiilor R1 și R2 din exemplul 10 al algebrei relaționale (figura 2.17) se scrie:

```
SELECT *
FROM R1, R2
WHERE R1.A >= R2.E
```

iar pentru echi-joncționarea din exemplul 11 (figura 2.18):

```
SELECT *
FROM R1, R2
WHERE R1.A = R2.E
```

Joncțiunea naturală poate fi realizată numai prin specificarea numelor atributelor în clauza SELECT a frazei de interogare. În standardul SQL-92 și în implementările SQL ale multor SGBD-uri se poate folosi o varianta mai elegantă, înănd seamă că tot ce înseamnă theta- și echi-joncțiunea reprezintă, pentru SQL, INNER JOIN (joncțiune internă). Prin urmare, cele două soluții de mai sus pot fi recrise după cum urmează:

```
SELECT *
FROM R1 INNER JOIN R2 ON R1.A >= R2.E
```

respectiv

```
SELECT *
FROM R1 INNER JOIN R2 ON R1.A = R2.E
```

Surprinzător, cel puțin pentru mine, este că Oracle (Oracle 8) nu are implementată încă această sintaxă, astfel încât, de aici incolo, toate variantele ce utilizează INNER JOIN operează în DB2 și VFP, nu însă și în Oracle 8.

Reluăm, pentru comparație, exemple din algebra relațională.

Exemplul 13

(Să se obțină, pentru fiecare localitate: codul poștal, denumirea, indicativul și denumirea județului și regiunea din care face parte)

- Varianta 1 (generală):

```
SELECT CodPost, Loc, LOCALITATI.Jud, Judet, Regiune
FROM LOCALITATI, JUDETE
WHERE LOCALITATI.Jud = JUDETE.Jud
```

- Varianta 2 (exclusiv Oracle):

```
SELECT CodPost, Loc, LOCALITATI.Jud, Judet, Regiune
FROM LOCALITATI INNER JOIN JUDETE ON
LOCALITATI.Jud = JUDETE.Jud
```

Numai atributul Jud a fost prefixat de numele tabelei din care provine. Prefixarea este obligatorie atunci când câmpul există în două sau mai multe dintre tabelele enumerate în clauza FROM.

Exemplul 14

(Care sunt localitățile din Banat?)

- Varianta 1 (generală):

```
SELECT CodPost, Loc, LOCALITATI.Jud, Judet, Regiune
FROM LOCALITATI, JUDETE
WHERE LOCALITATI.Jud = JUDETE.Jud AND Regiune='Banat'
în clauza WHERE, predicatul de selecție pentru realizarea joncțiunii i s-a adăugat secvența de test a regiunii.
```

- Varianta 2 (exclusiv Oracle):

```
SELECT CodPost, Loc, LOCALITATI.Jud, Judet, Regiune
FROM LOCALITATI INNER JOIN JUDETE ON
LOCALITATI.Jud = JUDETE.Jud
WHERE Regiune='Banat'
```

Avantajul celei de-a doua variante este de separarea condiției ce definește regiune, de condiția legată de joncțiunea propriu-zisă.

Exemplul 15

(În ce zile s-a vândut produsul cu denumirea „Produs 1”?)

- Varianta 1 (generală):

```
SELECT DISTINCT DataFact
FROM PRODUSE, LINIIFACT, FACTURI
WHERE PRODUSE.CodPr = LINIIFACT.CodPr AND
LINIIFACT.NrFact = FACTURI.NrFact AND
DenPr = 'Produs 1'
```

- Varianta 2 (exclusiv Oracle):

```
SELECT DISTINCT DataFact
FROM PRODUSE
INNER JOIN LINIIFACT ON PRODUSE.CodPr = LINIIFACT.CodPr
INNER JOIN FACTURI ON LINIIFACT.NrFact = FACTURI.NrFact
WHERE DenPr = 'Produs 1'
```

De notat folosirea clauzei DISTINCT pentru eliminarea eventualelor dubluri.

Exemplul 16

(În ce județ s-a vândut produsul cu denumirea „Produs 1” în perioada 3-5 august 2000?)

- Varianta 1 (generală):

```
SELECT DISTINCT Judet
FROM PRODUSE, LINIIFACT, FACTURI, CLIENTI, LOCALITATI, JUDETE
```

Care sunt persoanele ce trebuie felicitate de Sfântul Ioan?

Firește, am fi tentați să redactăm varianta:

```
SELECT *
FROM PERSOANE
WHERE UPPER(Prenume) LIKE '%ION%'
```

Funcția UPPER face conversia valorilor atributului Prenume în majuscule. Rezultatul se vede figura 4.15.

CNP	NUME	PRENUME	ADRESA	SEX	CODPOST	TELACASA	TELBIROU	TELMOBIL	EMAIL
CNP2	Vasile	Ion		B	6600	234567	876543	094222223	ion@e.ro
CNP4	Lazer	Caraion	M.Eminescu, 42	B	6500	456789		094222225	
CNPS	Iurea	Simion	I.Creanga, 44 bis	B	6500	567890	543210		

Figura 4.15. Tentativă ratată de a extrage „Sfinții Ioni”

Interogarea nu și-a atins țintă, deoarece, pe de o parte, nu au fost extrase persoanele cu prenume ca Ioan, Ioana, Ioanid, iar pe de altă parte, au fost eronat extrase și persoanele cu prenumele Caraion și Simion.

Prin urmare, cele trei litere – ION trebuie plasate la începutul prenumelui. La acest şablon se adaugă și IOAN. Bun, dar dacă pe omul nostru îl cheamă Marius Ion (are două prenume)? Cele două şabioane trebuie să se găsească la începutul fiecărui cuvânt din atributul Prenume. Iar dacă avem în vedere că uneori cele două prenume se despărță prin cratimă (Marius-Ion), rezultă o mândrețe de interogare (răspunsul este prezentat în figura 4.16):

```
SELECT *
FROM PERSOANE
WHERE UPPER(Prenume) LIKE 'ION%' OR
      UPPER(Prenume) LIKE 'IOAN%' OR
      UPPER(Prenume) LIKE '% ION%' OR
      UPPER(Prenume) LIKE '% IOAN%' OR
      OR UPPER(Prenume) LIKE '%-ION%' OR
      UPPER(Prenume) LIKE '%-IOAN%'
```

CNP	NUME	PRENUME	ADRESA	SEX	CODPOST	TELACASA	TELBIROU	TELMOBIL	EMAIL
CNP2	Vasile	Ion		B	6600	234567	876543	094222223	ion@e.ro
CNP3	Popovici	Ioana	V.Micle, Bl.2, Ap.2	F	5725	345678		094222224	
CNP7	Pope	Ioanid	I.Ion, Bl.H2, Sc.C, Ap.45	B	1900	789012	321098		

Figura 4.16. Persoanele ce trebuie felicitate de Sf. Ioan

Varianta funcționează în această formă deopotrivă în DB2/Oracle/VFP. Deși rare, există cazuri când printre caracterele căutate în valorile unui atribut și de caracter se găsește chiar unul dintre cele două şabioane, sau %.

Dacă, spre exemplu, intereseză toți clienții care conțin simbolul % în numele lor (s-ar putea ca, la un moment dat, să avem în bază clienți de genul „Procentul vesel % SRL”). Soluția este:

```
SELECT *
FROM CLIENTI
WHERE DenCl LIKE '%\%%' ESCAPE '\'
```

Datorită primului și ultimului simbol procent, poziția caracterului căutat (în cazul nostru, chiar %) nu prezintă importanță: poate fi prima, ultima sau oricare între prima și ultima. Rezultatul corect este obținut grație unui caracter declarat prin clauza ESCAPE. Acesta este backslash-ul, dar poate fi oricare altul. Prin clauza ESCAPE s-a indicat SQL-ului că procentul ce urmează simbolului \ nu este joker, ci are regim de caracter oarecare, ce trebuie căutat în tabelă ca atare.

Operatorul IN

Atunci când se testează dacă valoarea unui atribut este încadrabilă într-o listă de valori date, în locul folosirii abundente a operatorului OR este mai elegant să se apeleze la serviciile operatorului IN.

Format general:

expresie1 IN (expresie2, expresie3, ...)

Rezultatul evaluării unui predicat ce conține acest operator va fi adevărat dacă valoarea expresiei1 este egală cu (cel puțin) una dintre valorile: expresie2, expresie3, ...

Care sunt localitățile din județele Iași (IS), Vaslui (VS) și Timiș (TM)?

- Fără operatorul IN:

```
SELECT *
FROM LOCALITATI
WHERE Jud = 'IS' OR Jud = 'VS' OR Jud = 'TM'
ORDER BY Jud, Loc
```

- Cu operatorul IN:

```
SELECT *
FROM LOCALITATI
WHERE Jud IN ('IS', 'VS', 'TM')
ORDER BY Jud, Loc
```

- Care sunt facturile întocmite pe 1, 3 și 7 august 2000?

```
SELECT *
FROM FACTURI
WHERE DataFact IN ('01/08/2000',
                   '03/08/2000', '07/08/2000')
```

Firește, sintaxa trebuie adaptată în funcție de felul în care fiecare SGBD lucrează cu atribute și constante de tip dată calendaristică.

Nifact	Datafact	Codcl	Oba
1115	02/08/2000	1001	NULL
1116	02/08/2000	1007	Pretul propus initial a fost modificat
1117	03/08/2000	1001	NULL
1118	04/08/2000	1001	NULL

Figura 4.10. Operator BETWEEN – rezultat Visual FoxPro 6

Să se obțină, în ordinea județelor, lista localităților cu indicativul județului cuprins între IS (Iași) și TM (Timiș).

```
SELECT Jud, Loc, CodPost
FROM LOCALITATI
WHERE Jud BETWEEN 'IS' AND 'TM'
ORDER BY Jud, Loc
```

JUD	LOC	CODPOST
IS	Iasi	6800
IS	Pascari	5725
NT	Roman	5550
SV	Suceava	5800
TM	Timisoara	1900

Figura 4.11. Operatorul BETWEEN aplicat atributelor sir de caractere

Din examinarea cu ochiul liber a celor trei figuri reiese că nici DB2 și nici SQL*Plus (Oracle 8) nu afișează valorile NULL în clar, ci spații.

Operatorul LIKE

De multe ori, când se dorește obținerea unor informații din bază suntem în postura, oarecum ingrată, de a nu ști cu exactitate cum se numește un client sau un produs, sau, pur și simplu, unei persoane îi cunoaștem numai unul dintre prenume etc. Sunt numai câteva situații pentru rezolvarea cărora a fost gândit operatorul LIKE.

Operatorul LIKE permite compararea unui atribut (expresii) cu un literal utilizând o „mască” construită cu ajutorul specificatorilor multipli % și _ Simbolurile procent și liniuță de jos (underscore, diferență de cratimă) sunt denumite și jokeri. Procentul substituie un sir de lungime variabilă, 0 – n caractere, în timp ce liniuța substituie un singur caracter.

Care dintre firmele-client sunt societăți cu răspundere limitată (SRL-uri)?

Întrucât, în tabela CLIENTI, denumirea fiecărui client se termină cu forma sa de societate (SA, SRL etc.), se poate reda consulta:

```
SELECT *
FROM CLIENTI
WHERE DenCl LIKE '%SRL'
```

Rezultatul se prezintă ca în figura 4.12.

CODCL	DENCL	CODFISCAL	ADRESA	CODPOST	TELEFON
1001	Client 1 SRL	R1001	Tranzitiei, 13 bis	6800	
1003	Client 3 SRL	R1003	Prosperitate, 22	6500	035-222222
1005	Client 5 SRL	R1005		1900	056-111111
1007	Client 7 SRL	R1007	Victoria Copilismul, 2	1900	056-121212

Figura 4.12. Clientii – SRL-uri

Care dintre clienți au numele (fără forma de societate și spațiul dinainte) din 8 caractere și sunt societăți pe acțiuni?

```
SELECT *
FROM CLIENTI
WHERE DenCl LIKE '_____ SA'
```

Cele șapte liniuțe (nu se observă, dar sunt șapte, parolă) plus spațiul care le urmează nu au efect vizibil/împresionant pentru baza noastră de date, deoarece în capitolul 1, leneș fiind, am denumit toți clienții de o manieră simplistă – Client 1 SRL, Client 2 SA etc. Dacă am avea mai mulți clienți (sau măcar un „Client 10 SA”), atunci interogarea ar avea cu totul alt farmec.

Ce persoane au numele conținând litera S pe a treia poziție?

```
SELECT *
FROM PERSOANE
WHERE Nume LIKE '_s%'
```

CNP	NUME	PRENUME	ADRESA	SEX	CODPOST	TELACASA	TELBIROU	TELEMOBIL	EMAIL
CNP2	Vasile	Ion		B	6800	234567	876543	094222223	Ion@ero
CNP6	Vasc	Simona	M.Eminescu, 13	F	5725		432109	094222227	

Figura 4.13. Persoane cu litera S pe a treia poziție a numelui

Rezultatul este vizualizat în figura 4.13. Atenție! – dacă există persoane al căror nume are litera S majusculă pe a treia poziție, acestea nu sunt extrase în rezultat. În asemenea situații, soluția de mai jos este ceva mai sigură:

```
SELECT *
FROM PERSOANE
WHERE Nume LIKE '_s%' OR Nume LIKE '%S%'
```

In numele căror persoane apare, măcar o dată, litera S (indiferent de poziție/pozitii)?

```
SELECT *
FROM PERSOANE
WHERE Nume LIKE '%s%' OR Nume LIKE '%S%'
```

CNP	NUME	PRENUME	ADRESA	SEX	CODPOST	TELACASA	TELBIROU	TELEMOBIL	EMAIL
CNP2	Vasile	Ion		B	6800	234567	876543	094222223	Ion@ero
CNP6	Vasc	Simona	M.Eminescu, 13	F	5725		432109	094222227	
CNP8	Bogacs	Idiko	I.V.Kleezu, 67	F	5550	890123	210987	094222229	

Figura 4.14. Persoane ai căror nume conține litera S

CODPOST	LOC.	JUD
6400	Birlad	VS
5300	Focșani	VN
6600	Iași	IS
5725	Pescari	IS
5550	Romeni	NT
5800	Suceava	SV
1900	Timișoara	TM
6500	Vaslui	VS

Figura 4.6. Localitățile ordonate alfabetic

Aranjarea se face implicit crescător (ASC). Prin opțiunea DESC, ordinea prezentării se inversează. În plus, se pot specifica mai multe coloane care să servească drept criterii suplimentare de ordonare. La valori egale ale primului atribut, intră în acțiune criteriul de „balotaj”, care este al doilea atribut etc.

Să se obțină, în ordinea descrescătoare a indicativului județelor, lista localităților în ordinea crescătoare a denumirii.

```
SELECT Jud, Loc, CodPost
FROM LOCALITATI
ORDER BY Jud DESC, Loc ASC
```

Rezultatul este cel din figura 4.7. Ultimul indicativ de județ (e vorba de ordine alfabetice) din tabela LOCALITĂȚI este VS (pentru Vaslui), deci primele localități afișate vor fi din acest județ; prin urmare, prima linie a rezultatului se referă la municipiul Bârlad, iar a doua la municipiul Vaslui.

JUD	LOC.	CODPOST
VS	Birlad	6400
VS	Vaslui	6500
VN	Focșani	5300
TM	Timișoara	1900
SV	Suceava	5800
NT	Romeni	5550
IS	Iași	6600
IS	Pescari	5725

Figura 4.7. Două criterii de ordonare

4.2. Operatorii BETWEEN, LIKE, IN

Pentru formularea predicatului de selecție, SQL permite utilizarea, pe lângă „clasicii” >, \geq , <, \leq , $=$, \neq , și a altor operatori, dintre care ne vom opri la: BETWEEN (intre, cuprins între), LIKE (ca și, la fel ca), IN (in), la care se adaugă IS NULL, ce va fi prezentat în capitolul 5.

Operatorul BETWEEN

Este util pentru definirea intervalor de valori. Ne reîntoarcem în capitolul 2, la exemplul 3 (*Care sunt facturile emise în perioada 2-5 august 2000?*).

Pe lângă soluția formulată anterior în acest capitol, vom prezenta noi variante DB2, Oracle și VFP. Profităm de ocazie pentru a prezenta și modul de vizualizare a rezultatelor cele trei SGBD-uri.

- Soluție DB2 v6.1 – vizualizare rezultat – figura 4.8.

```
SELECT *
FROM FACTURI
WHERE DataFact BETWEEN '02/08/2000' AND '05/08/2000'
```

NRFACT	DATAFACT	CODCL	OBS
1115	08/02/2000	1001	-
1116	08/02/2000	1007	Pretul propus initial a fost modificat
1117	08/03/2000	1001	-
1118	08/04/2000	1001	-

4 record(s) selected.

Figura 4.8. Operator BETWEEN – rezultat DB2 v6.1

- Soluția Oracle – rezultat figura 4.9. Specificarea constantelor de tip dată calendaristică fac necesară în Oracle funcția TO_DATE.

```
SELECT *
FROM FACTURI
WHERE DataFact BETWEEN TO_DATE('02/08/2000', 'DD/MM/YYYY')
AND TO_DATE('05/08/2000', 'DD/MM/YYYY');
```

NRFACT	DATAFACT	CODCL	OBS
1115	02-AUG-00	1001	
1116	02-AUG-00	1007	Pretul propus initial a fost modificat
1117	03-AUG-00	1001	
1118	04-AUG-00	1001	

Figura 4.9. Operator BETWEEN – rezultat Oracle 8 (SQL*Plus)

- Constantele de tip dată calendaristică trebuie incadrare de acolade în VFP. Ecranul de „răspuns” este cel din figura 4.10.

```
SELECT * ;
FROM FACTURI ;
WHERE DataFact BETWEEN {02/08/2000} AND {05/08/2000}
```

Complicându-ne și mai zdravăn, dorim să afișăm pentru fiecare factură ce dată va fi peste 1 an, două luni și 25 de zile de la momentul emiterii.

- Soluția DB2:

```
SELECT NrFact AS Factura, DataFact AS Data_Facturare,
       DataFact + 1 YEARS + 2 MONTHS + 25 DAYS AS
         O_Data_Viitoare
  FROM FACTURI
```

- Soluția Oracle necesită transformarea anilor în luni:

```
SELECT NrFact AS Factura, DataFact AS Data_Facturare,
       ADD_MONTHS(DataFact,14)+15 AS O_Data_Viitoare
  FROM FACTURI
```

- În VFP:

```
SELECT NrFact AS Factura, DataFact AS Data_Facturare,
       QMMONTH(DataFact,14)+15 AS O_Data_Viitoare ;
  FROM FACTURI
```

În ceea ce privește operațiunile de adunare și scădere între două date calendaristice, aici lucrurile se prezintă și mai diferențiat. Spre exemplu, interesează intervalul scurs între momentul curent și cel al emiterii fiecărei facturi. Interrogarea DB2:

```
SELECT NrFact AS Factura, DataFact AS Data_Facturare,
       CURRENT DATE - DataFact AS Timp_Scurs
  FROM FACTURI
```

furnizează valorile din figura 4.4. Trebuie avut în vedere că interrogarea a fost executată pe 5 februarie 2001.

FACTURA	DATA_FACTURARE	TIMP_SCURS
1111	2000-08-01	604
1112	2000-08-01	604
1113	2000-08-01	604
1114	2000-08-01	604
1115	2000-08-02	603
1116	2000-08-02	603
1117	2000-08-03	602
1118	2000-08-04	601
1119	2000-08-07	529
1120	2000-08-07	529
1121	2000-08-07	529
1122	2000-08-07	529

Figura 4.4. Intervalul dintre 05/02/2001 (data execuției interogării) și data fiecărei facturi (DB2)

Rezultatul scăderii a două date calendaristice conține numărul de ani, luni și zile. În figura 4.4, valoarea 604 înseamnă 6 luni și 4 zile, iar 529 – 5 luni și 29 de zile. Bineînțeles că apelând la funcții de conversie și extragere se poate obține un format de prezentare ceva mai agreabil.

În Oracle, avem nevoie de câteva artificii între data curentă (SYSDATE) și data facturării. Funcția MONTHS_BETWEEN calculează numărul lunilor cuprinse între două date calendaristice:

```
SELECT NrFact AS Factura, DataFact AS Data_Facturare,
       TRUNC(MONTHS_BETWEEN(TRUNC(SYSDATE), DataFact), 0)
         AS Luni_Scurse,
       ADD_MONTHS(TRUNC(SYSDATE), -
                  TRUNC(MONTHS_BETWEEN(TRUNC(SYSDATE), DataFact), 0))
                  - DataFact AS Zile_Scurse
  FROM FACTURI
```

Interrogarea generează un rezultat de forma celui din figura 4.5.

FACTURA	DATA_FACTURARE	LUNI_SCURSE	ZILE_SCURSE
1111	01-AUG-00	6	4
1112	01-AUG-00	6	4
1113	01-AUG-00	6	4
1114	01-AUG-00	6	14
1115	02-AUG-00	6	3
1116	02-AUG-00	6	3
1117	03-AUG-00	6	2
1118	04-AUG-00	6	1
1119	07-AUG-00	5	29
1120	07-AUG-00	5	29
1121	07-AUG-00	5	29
1122	07-AUG-00	5	29

Figura 4.5. Intervalul dintre 05 febr. 2001 (data execuției interogării) și data fiecărei facturi (Oracle)

Opsiunea ORDER BY

Una dintre caracteristicile modelului relațional este că nici ordinea atributelor, nici ordinea linilor în relații nu prezintă importanță din punctul de vedere al conținutului informațional. În practică însă, forma de prezentare a rezultatelor interogării este importantă. Spre exemplu, o listă a localităților este cu mult mai de folos decă se prezintă în ordine alfabetică. Ordonarea linilor în rezultatul unei interogări este posibilă prin clauza ORDER BY.

Să se obțină lista localităților în ordine alfabetică.

```
SELECT *
  FROM LOCALITATI
 ORDER BY LOC
```

Rezultatul se prezintă ca în figura 4.6.

A patra coloană este denumită ValFaraTVA, după cum a fost specificat în clauza AS. Valorile sale sunt determinate pe baza expresiei Cantitate * PretUnit.

Un alt tip de expresie este cel bazat pe concatenare, adică pe alipirea mai multor constante și variabile într-o coloană nouă. Operatorul SQL pentru concatenare este alcătuit din două bare verticale (||). Spre exemplu, interogarea următoare produce rezultatul din figura 4.2.

```
SELECT 'Judetul ' || Judet || ' se află în ' || Regiune
      AS Exemplu_Concatenare
FROM JUDETE
```

EXEMPLU_CONCATENARE
Judetul Iași se află în Moldova
Judetul Vrancea se află în Moldova
Judetul Neamț se află în Moldova
Judetul Suceava se află în Moldova
Judetul Vaslui se află în Moldova
Judetul Timiș se află în Banat

Figura 4.2. Concatenarea unor literali și atribute řir de caractere

Unele SGBD-uri, precum VFP, nu au implementat acest operator, folosind în acest scop unul specific (în cazul VFP este +).

În Oracle, se pot concatena direct, adică fără conversie prealabilă, literali, atribute řir de caractere, numerice etc. Alte SGBD-uri, precum DB2, necesită transformarea valorilor non-řir de caractere (attribute/constante numerice, dată calendaristică etc.), operařiune realizată prin funcþia CAST.

Spre exemplu, în Oracle, interogarea următoare este funcþională:

```
SELECT 'Factura ' || NrFact || ' a fost emisă pe data ' || 
       DataFact AS Concatenare_Oracle
FROM FACTURI
```

în timp ce DB2 nu o suportă, și asta nu neapărat din cauza faptului că am denumit coloana calculată Concatenare_Oracle. Folosind funcþia de conversie CAST, putem redaþta următoarea varianþă a interogării, mult mai pe gustul DB2:

```
SELECT 'Factura ' || CAST (NrFact AS CHAR(8)) || 
       ' a fost emisă pe data ' || CAST (DataFact AS 
                                         VARCHAR(10)) AS Concatenare_DB2
FROM FACTURI
```

Valorile atributului NrFact sunt transformate în řiruri de caractere de lungime fixă (8 poziþii), în timp ce ale DataFact vor fi convertite în řiruri de caractere de lungime variabilă (vorba vine, deoarece formatul datei este unitar pentru toate liniile rezultatului) de maximum 10 poziþii.

Visual FoxPro face notă discordantă, întrucât pentru conversia valorilor din numeric în řir de caractere se foloseþte funcþia STR(), iar din dată calendaristică, după caz, DTOC() sau DTOS():

```
SELECT 'Factura '+STR(NrFact,8)+' a fost emisă pe data '+;
       DTOC(DataFact) AS Concatenare_VFP1 ;
FROM FACTURI
```

În finalul discuþiei despre coloanele-expresii, mai zăbovim preþ de câteva rânduri asupra expresiilor de tip dată calendaristică. Modurile în care au fost implementate aceste funcþii sunt foarte eterogene, de la SGBD la SGBD. Să presupunem că orice factură trebuie incasată în maximum două săptămâni. Pentru a afiþa scadentele, soluþia DB2 este:

```
SELECT NrFact AS Factura, DataFact AS Data_Facturare,
       DataFact + 14 DAY AS Scadenta_Incasare
FROM FACTURI
```

FACTURA	DATA_FACTURARE	SCADENTA_INCASARE
1111	2000-08-01	2000-08-15
1112	2000-08-01	2000-08-15
1113	2000-08-01	2000-08-15
1114	2000-08-01	2000-08-15
1115	2000-08-02	2000-08-16
1116	2000-08-02	2000-08-16
1117	2000-08-03	2000-08-17
1118	2000-08-04	2000-08-18
1119	2000-08-07	2000-08-21
1120	2000-08-07	2000-08-21
1121	2000-08-07	2000-08-21
1122	2000-08-07	2000-08-21

Figura 4.3. O expresie

Data care ne interesează se obþine graþie expresiei DataFact + 14 DAY. Într-o expresie de tip DATE sau DATETIME, imediat după număr, trebuie indicată semnificaþia acestuia: YEAR (YEARS), MONTH (MONTHS), DAY (DAYS) sau, după caz, HOUR (HOURS), MINUTE (MINUTES), SECOND (SECONDS), MICROSECOND (MICROSECONDS).

La aceeaþi problemă, Oracle și Visual FoxPro permit soluþii mai simple (de data aceasta); DataFact fiind un atribut de tip DATE, implicit se consideră 14 ca fiind numărul zilelor:

```
SELECT NrFact AS Factura, DataFact AS Data_Facturare,
       DataFact + 14 AS Scadenta_Incasare
FROM FACTURI
```

Dacă am presupune că scadenþă ar fi peste două luni de la facturare, interogările ar trebui modificate astfel:

- DB2:

```
SELECT NrFact AS Factura, DataFact AS Data_Facturare,
       DataFact + 2 MONTHS AS Scadenta_Incasare
FROM FACTURI
```

- Oracle:

```
SELECT NrFact AS Factura, DataFact AS Data_Facturare,
       ADD_MONTHS(DataFact,2) AS Scadenta_Incasare
FROM FACTURI
```

- Funcþia Visual FoxPro corespondentă ADD_MONTHS este GOMONTH:

```
SELECT NrFact AS Factura, DataFact AS Data_Facturare, ;
       GOMONTH(DataFact,2) AS Scadenta_Incasare ;
FROM FACTURI
```

```
SELECT CodPr
FROM LINIIFACT
WHERE NrFact = 1117
```

Exemplul de mai sus funcționează în DB2 și Oracle, nu însă și în Visual FoxPro, care nu are implementat operatorul **INTERSECT**, astfel încât intersecția trebuie realizată prin alte clauze și operatori.

Diferență

Operatorul așteptat ar fi **MINUS**. În standardul SQL-92, și în câteva SGBDR-uri precum DB2, operatorul **MINUS** nu există, fiind substituit de **EXCEPT**, iar în alte SGBD-uri nu există nici unul, nici altul. Tabela R5 din figura 2.5, calculată prin expresia R1-R2, se obține în SQL prin interogarea:

```
SELECT *
FROM R1
EXCEPT
  SELECT *
  FROM R2
```

Oracle permite soluția:

```
SELECT *
FROM R1
MINUS
  SELECT *
  FROM R2
```

Ca și în cazul reuniunii și intersecției, eliminarea tuplurilor duplicate se face automat, iar repetarea lor se asigură prin **EXCEPT ALL**.

Modificăm enunțul exemplului 9 în următorul: **Care sunt codurile produselor care apar în factura 1111, dar nu apar în factura 1117?**

- Soluția DB2:**

```
SELECT CodPr
FROM LINIIFACT
WHERE NrFact = 1111
EXCEPT
  SELECT CodPr
  FROM LINIIFACT
  WHERE NrFact = 1117
```

- Soluția Oracle:**

```
SELECT CodPr
FROM LINIIFACT
WHERE NrFact = 1111
MINUS
  SELECT CodPr
  FROM LINIIFACT
  WHERE NrFact = 1117
```

Indiferent de variantă, rezultatul este o tabelă cu linie și o coloană, valoarea fiind 5 (produsul ce are codul 5 apare în prima factură, dar nu și în a două).

Ca și pentru **INTERSECT**, Visual FoxPro nu are implementat operator SQL pentru realizarea diferenței a două relații.

Pornind de la faptul că intersecția nu este un operator fundamental, fraza **SELECT** ce realizează intersecția se poate reformula cu ajutorul diferenței astfel (firește, în Oracle se înlocuiește **EXCEPT** cu **MINUS**):

```
SELECT *
FROM R1
EXCEPT
  (SELECT *
   FROM R1
   EXCEPT
    SELECT *
    FROM R2)
```

Produsul cartezian

SQL nu pune la dispoziție vreun operator special dedicat produsului cartezian. Nici nu este nevoie. Tabela R6 din figura 2.6 se obține, pur și simplu, prin enumerarea celor două relații în clauza **FROM**:

```
SELECT *
FROM R1, R2
```

Discuția despre operatorii juncțiune și diviziune se amână până la o dată ce va fi comunicată din timp. În continuare, prezentăm câteva facilități SQL privitoare la coloane calculate și ordonarea liniilor în rezultatul interogării.

Coloane-expresii

O facilitate importantă în multe interogări SQL ține de definirea, pe lângă atributele tabelelor, a unor coloane noi, pe baza unor expresii. Clauza **AS** permite denumirea coloanelor calculate sau redenumirea unor coloane ale tabelelor. Să luăm un exemplu banal: **Care este, pentru fiecare produs din factura 1111, codul, cantitatea, prețul unitar și valoarea fără TVA?**

```
SELECT CodPr, Cantitate, PretUnit, Cantitate * PretUnit
      AS ValFaraTVA
FROM LINIIFACT
WHERE NrFact = 1111
```

Rezultatul interogării este prezentat în figura 4.1.

CODPR	CANTITATE	PRETUNIT	VALFARATVA
1	50	10000	500000
2	75	10500	787500
5	500	6500	3250000

Figura 4.1. Exemplu de coloană calculată

Exemplul 4 – proiecție

```
SELECT A,C
FROM R
```

Exemplul 5 (Care sunt regiunile jării preluate în bază?)

După cum aminteam mai sus, spre deosebire de algebra relațională, SQL nu elimină automat dublurile. Tabela obținută prin consultarea:

```
SELECT Regiune
FROM JUDETE
```

are structura și conținutul identice cu R' (figura 2.11). Pentru a obține răspunsul de forma tabeliei R (o regiune să apară o singură dată în răspuns), se folosește clauza DISTINCT:

```
SELECT DISTINCT Regiune
FROM JUDETE
```

Exemplul 6 (Care sunt codul, denumirea și numărul de telefon ale fiecărui client?)

```
SELECT CodCl, DenumCl, Telefon
FROM CLIENTI
```

Nu este necesară clauza DISTINCT, deoarece CodCl este cheia primară a tabelei CLIENTI.

Exemplul 7 (Care este numărul de telefon al clientului Client 2 SA?)

```
SELECT Telefon
FROM CLIENTI
WHERE DenumCl = "Client 2 SA"
```

Exemplul 8 (Care sunt denumirile și codurile poștale ale localităților prezente în bază din județele Iași (IS) și Vrancea (VN)?)

```
SELECT Loc, CodPost
FROM LOCALITATI
WHERE Jud = 'IS' OR Jud = 'VN'
```

Reuniune, intersecție, diferență, produs cartezian

Primii trei operatori asamblăți prezintă operatori SQL dedicăți: UNION, INTERSECT, MINUS (EXTRACT), în timp ce produsul cartezian se calculează automat prin simpla enumerare a celor două tabele în clauza FROM.

Reuniunea

Un rezultat identic cu tabela R3 din figura 2.3 se obține prin fraza SELECT următoare:

```
SELECT *
FROM R1
UNION
SELECT *
FROM R2
```

La reuniunea a două tabele, SQL elimină automat liniile identice din rezultat. Dacă se dorește preluarea tuturor liniilor celor două relații și, implicit, apariția de liniile duplicate, se folosește clauza ALL astfel:

```
SELECT *
FROM R1
UNION ALL
SELECT *
FROM R2
```

Revenim la exemplul 8 din capitolul 2 (Care sunt denumirile și codurile poștale ale localităților (prezente în bază) din județele Iași (IS) și Vrancea (VN) ?).

Fraza SQL echivalentă soluției 2 (bazată pe reuniune) este:

```
SELECT Loc, CodPost
FROM LOCALITATI
WHERE Jud = 'IS'
UNION
SELECT Loc, CodPost
FROM LOCALITATI
WHERE Jud = 'VN'
```

Intersecția

Raportându-ne la exemplul din figura 2.3, echivalentul tabelei R4 se obține în SQL prin:

```
SELECT *
FROM R1
INTERSECT
SELECT *
FROM R2
```

SQL-92 permite, ca și în cazul reuniunii, prezența repetată în rezultat a unor lini (tuple duplicate), bineînțeles, atunci când cele două relații asupra cărora se aplică intersecția nu respectă restricția de unicitate:

```
SELECT *
FROM R1
INTERSECT ALL
SELECT *
FROM R2
```

Dacă tuplul t1 apare repetat și în R1, și în R2, mai precis, de n ori în R1 și de m ori în R2, în rezultatul operatorului INTERSECT t1 apare o singură dată, în timp ce utilizând INTERSECT_ALL t1 va apărea de un număr de ori care este minimul dintre n și m.

Pentru o primă ilustrare a utilizării intersecției, transcriem soluția din algebra relațională formulată la exemplul 9 (Care sunt codurile produselor care apar simultan și în factură 1111, și în factură 1117?).

```
SELECT CodPr
FROM LINIIFACT
WHERE NrFact = 1111
INTERSECT
```

Capitolul 4

ELEMENTE DE BAZĂ ALE INTEROGĂRILOR SQL

Lucrarea de față este dedicată preponderent modalităților prin care, pornind de la o schemă relațională, pot fi obținute diverse informații dintr-o bază de date. Procesul se numește interogare, iar formularea unei interogări înseamnă redactarea unei fraze SELECT. Prezentul capitol tratează fundamentele redactării frazelor SELECT și cele mai „populare” clauze ale acesteia.

4.1. Principalele clauze ale frazei SELECT

O frază SELECT are un format pe căt de simplu, pe atât de flexibil. Cele trei clauze principale sunt SELECT, FROM și WHERE⁵¹, dintre care numai primele două sunt obligatorii. Pentru început, vom face o paralelă cu operatorii algebrei relaționale prezenți în capitolul 2.

Selecția și proiecția

Clauza SELECT corespunde operatorului proiecție din algebra relațională, fiind utilizată pentru desemnarea listei de atribute (coloanele) din rezultat. Clauza FROM este cea în care sunt enumerate relațiile din care vor fi extrase informațiile aferente consultării. Prin WHERE se desemnează predicatul **selecțiv al algebrei relaționale**, relativ la atribute ale relațiilor care apar în clauza FROM.

La modul general (și simplist), o consultare simplă în SQL poate fi prezentată astfel:

```
SELECT C1, C2, ..., Cn
FROM R1, R2, ..., Rm
WHERE P
```

Prin execuția unei fraze SELECT se obține un rezultat de formă tabelară. Acesta poate fi o listă (text), o tabelă propriu-zisă sau o tabelă temporară (care, de obicei, nu poate fi actualizată), dar și o tabelă derivată (imagine). Uneori rezultatul poate fi obținut și ca o variabilă masiv (tablou).

51. La adresa <http://w3.one.net/~jhoffman/sqltut.htm> este un bun curs introductiv despre SQL.

- Ci – reprezintă coloanele (care sunt atribute sau expresii de atribute) rezultat;
- Rj – sunt relațiile ce trebuie parcuse pentru obținerea rezultatului;
- P – este predicatul (condiția) simplu sau compus ce trebuie îndeplinit de tupluri pentru a fi incluse în rezultat.

Atunci când clauza WHERE este omisă, se consideră implicit că predicatul P are valoarea logică „adevărat”, astfel încât vor fi incluse în rezultat toate liniile din tabela, sau produsul cartezian al tabelelor, enumerată/enumerate în clauza FROM.

Dacă, în locul coloanelor C1, C2, ..., Cn, apare simbolul *, rezultatul va fi alcătuit din toate coloanele (atributele) relațiilor specificate în clauza FROM. De asemenea, atributele rezultatului preiau numele din tabela/tabelele specificate în FROM. Schimbarea numelui se realizează prin clauza AS.

În capitolul 1 era subliniată echivalența noțiunilor relație-tabelă. Conform restricției de unicitate, într-o relație nu pot exista două lini îndepărtate. În SQL, tabela obținută dintr-o consultare poate conține două sau mai multe tupluri identice.

Spre deosebire de algebra relațională, în SQL nu se elimină automat tuplurile identice (dublurile) din rezultat. Pentru aceasta este necesară utilizarea opțiunii DISTINCT:

```
SELECT DISTINCT C1, C2, ..., Cn
FROM R1, R2, ..., Rm
WHERE P
```

În concluzie, o frază SELECT, în forma în care a fost prezentată, corespunde:

- unei selecții algebrice (clauza WHERE P)
- unei proiecții (SELECT Ci)
- unui produs cartezian (FROM = R1 \otimes R2 \otimes ... \otimes Rm)

și conduce la obținerea unui rezultat cu n coloane, fiecare coloană fiind: un atribut din R1, R2, ..., Rm sau expresie calculată pe baza unor atribute din R1, R2, ..., Rm.

Vom trece în SQL câteva interogări din algebra relațională, pe baza exemplelor din capitolul 2.

Exemplul 1 – selecție

```
SELECT *
FROM R1
WHERE A > 20 AND C > 20
```

Exemplul 2 – selecție (Care sunt județele din Moldova?)

```
SELECT *
FROM JUDETE
WHERE Regiune = "Moldova"
```

Exemplul 3 – selecție (Care sunt facturile emise în perioada 2-5 august 2000?)

Formatul standard al unei constante de tip dată calendaristică este YYYY-MM-DD, așa încât o interogare în SQL-92 poate avea forma:

```
SELECT *
FROM FACTURI
WHERE DataFact >= '2000/08/02' AND DataFact <= '2000/08/05'
```

Ștergerea linilor

Comanda SQL pentru ștergerea uneia sau mai multor linii dintr-o tabelă este **DELETE**. Formatul general este:

```
DELETE
  FROM nume-tabelă
 WHERE predicat
```

Din tabelă vor fi șterse toate liniile care îndeplinesc condiția specificată în predicatul din clauza WHERE.

Exemplul 1

Să se eliminate din tabela FACTURI factura cu numărul 1122.

```
DELETE FROM FACTURI
 WHERE NrFact = 1122
```

Varianta prezentată funcționează în toate cele trei SGBD-uri. Dacă pentru această factură există integrări-copil în LINIIFACT sau INCASFACT, SGBD-ul ține cont de opțiunea declarată la crearea tabeliei. Dacă s-a specificat (implicit sau explicit) ON DELETE RESTRICT, ștergerea este interzisă. În cazul ON DELETE CASCADE sunt șterse, o dată cu linia din FACTURI, toate liniile-copil din LINIIFACT și INCASFACT.

Exemplul 2

Să se eliminate din baza de date toate județele din Moldova.

```
DELETE FROM JUDETE
 WHERE Regiune = 'Moldova'
```

Ca și în cazul comenzii INSERT, pot fi șterse liniile pe baza unei condiții formulate printr-o subconsultare SQL, opțiune ce va fi prezentată în capitolul 6.

Modificarea valorilor unor attribute

Pentru a modifica valoarea uneia sau mai multor attribute pe una sau mai multe liniii dintr-o tabelă, se folosește comanda **UPDATE** cu formatul general (simplificat):

```
UPDATE tabelă
 SET atribut1 = expresie1 [, atribut2= expresie2 ...]
 WHERE predicat
```

Modificarea se va produce pe toate liniile tablei care îndeplinesc condiția formulată prin predicat.

Exemplul 3

Noul număr de telefon al clientului ce are codul 1001 este 032-313131. Să se opereze modificarea în baza de date.

Se actualizează atributul Telefon din tabela CLIENTI:

```
UPDATE CLIENTI
 SET Telefon = '032-313131'
 WHERE CodCl = 1001
```

Exemplul 4

În cadrul unei noi relaxări fiscale, se decide creșterea procentului TVA de la 19% la 22% pentru toate produsele.

Atributul modificat este ProcTVA din tabela PRODUSE, pe toate liniile:

```
UPDATE PRODUSE
 SET ProcTVA = 0.22
```

```

INSERT INTO facturi (nrfact, datafact, codcl)
    VALUES (1120, TO_DATE('07/08/2000', 'DD/MM/YYYY'), 1001);
INSERT INTO facturi (nrfact, datafact, codcl)
    VALUES (1121, TO_DATE('07/08/2000', 'DD/MM/YYYY'), 1004);
INSERT INTO facturi (nrfact, datafact, codcl)
    VALUES (1122, TO_DATE('07/08/2000', 'DD/MM/YYYY'), 1005);

INSERT INTO liniifact VALUES (1111, 1, 1, 50, 10000) ;
INSERT INTO liniifact VALUES (1111, 2, 2, 75, 10500) ;
INSERT INTO liniifact VALUES (1111, 3, 5, 500, 6500) ;
INSERT INTO liniifact VALUES (1112, 1, 2, 80, 10300) ;
INSERT INTO liniifact VALUES (1112, 2, 3, 40, 7500) ;
INSERT INTO liniifact VALUES (1113, 1, 2, 100, 9750) ;
INSERT INTO liniifact VALUES (1114, 1, 2, 70, 10700) ;
INSERT INTO liniifact VALUES (1114, 2, 4, 30, 15800) ;
INSERT INTO liniifact VALUES (1114, 3, 5, 700, 6400) ;
INSERT INTO liniifact VALUES (1115, 1, 2, 150, 9250) ;
INSERT INTO liniifact VALUES (1116, 1, 2, 125, 9300) ;
INSERT INTO liniifact VALUES (1117, 1, 2, 100, 10000) ;
INSERT INTO liniifact VALUES (1117, 2, 1, 100, 9500) ;
INSERT INTO liniifact VALUES (1118, 1, 2, 30, 11000) ;
INSERT INTO liniifact VALUES (1118, 2, 1, 150, 9300) ;
INSERT INTO liniifact VALUES (1119, 1, 2, 35, 10900) ;
INSERT INTO liniifact VALUES (1119, 2, 3, 40, 7000) ;
INSERT INTO liniifact VALUES (1119, 3, 4, 50, 14000) ;
INSERT INTO liniifact VALUES (1119, 4, 5, 750, 6300) ;
INSERT INTO liniifact VALUES (1120, 1, 2, 80, 11200) ;
INSERT INTO liniifact VALUES (1121, 1, 5, 550, 6400) ;
INSERT INTO liniifact VALUES (1121, 2, 2, 100, 10500) ;

INSERT INTO incasari VALUES (1234,
    TO_DATE('15/08/2000', 'DD/MM/YYYY'),
    'OP', '111', TO_DATE('10/08/2000', 'DD/MM/YYYY')) ;
INSERT INTO incasari VALUES (1235,
    TO_DATE('15/08/2000', 'DD/MM/YYYY'),
    'CHIT', '222', TO_DATE('15/08/2000', 'DD/MM/YYYY')) ;
INSERT INTO incasari VALUES (1236,
    TO_DATE('16/08/2000', 'DD/MM/YYYY'),
    'OP', '333', TO_DATE('09/08/2000', 'DD/MM/YYYY')) ;
INSERT INTO incasari VALUES (1237,
    TO_DATE('17/08/2000', 'DD/MM/YYYY'),
    'CEC', '444', TO_DATE('10/08/2000', 'DD/MM/YYYY')) ;
INSERT INTO incasari VALUES (1238,
    TO_DATE('17/08/2000', 'DD/MM/YYYY'),
    'OP', '555', TO_DATE('10/08/2000', 'DD/MM/YYYY')) ;
INSERT INTO incasari VALUES (1239,
    TO_DATE('18/08/2000', 'DD/MM/YYYY'),
    'OP', '666', TO_DATE('11/08/2000', 'DD/MM/YYYY')) ;

INSERT INTO incasfact VALUES (1234, 1111, 5399625) ;
INSERT INTO incasfact VALUES (1234, 1118, 1026375) ;
INSERT INTO incasfact VALUES (1235, 1112, 487705) ;
INSERT INTO incasfact VALUES (1236, 1117, 975410) ;
INSERT INTO incasfact VALUES (1236, 1118, 1026375) ;
INSERT INTO incasfact VALUES (1236, 1120, 731557) ;

```

```

INSERT INTO incasfact VALUES (1237, 1117, 975410) ;
INSERT INTO incasfact VALUES (1238, 1113, 1160250) ;
INSERT INTO incasfact VALUES (1239, 1117, 369680) ;
COMMIT ;

```

Ordinea valorilor din clauza VALUES trebuie să fie identică cu cea declarată la crearea (sau modificarea structurii) tabelelor. Modificarea este posibilă numai prin enumerarea, după numele tabeliei, a atributelor ce vor primi valorile specificate.

Pentru tabelele CLIENTI și FACTURI au fost incluse în clauza VALUES mai puține valori decât atributele talebelei. Este obligatorie, în aceste cazuri, precizarea atributelor care vor primi valorile. În Oracle și DB2, restul, adică cele nespecificate, vor avea, pe linile respective, valori NULL. În VFP, acestea sunt completate cu zero/spații (funcția EMPTY() întoarce .T., în timp ce ISNULL() .F.).

O problemă în VFP jine de lucrul cu atrbute de tip dată calendaristică. Listingul 3.5 prezintă, în acest sens, un extras din programul de populare a bazei de date, și anume comenziile INSERT pentru tabela FACTURI.

Listing 3.5. Constante de tip dată calendaristică în VFP

```

...
INSERT INTO facturi (nrfact, datafact, codcl) VALUES (1111, ;
    {'2000/08/01}, 1001)
INSERT INTO facturi VALUES (1112, (^2000/08/01}, 1005, ;
    'Probleme cu transportul', 0)
INSERT INTO facturi (nrfact, datafact, codcl) ;
    VALUES (1113, (^2000/08/01}, 1002)
INSERT INTO facturi (nrfact, datafact, codcl) ;
    VALUES (1114, (^2000/08/01}, 1006)
INSERT INTO facturi (nrfact, datafact, codcl) ;
    VALUES (1115, (^2000/08/02}, 1001)
INSERT INTO facturi VALUES (1116, (^2000/08/02}, ;
    1007, 'Prețul propus initial a fost modificat', 0)
INSERT INTO facturi (nrfact, datafact, codcl) ;
    VALUES (1117, (^2000/08/03}, 1001)
INSERT INTO facturi (nrfact, datafact, codcl) ;
    VALUES (1118, (^2000/08/04}, 1001)
INSERT INTO facturi (nrfact, datafact, codcl) ;
    VALUES (1119, (^2000/08/07}, 1003)
INSERT INTO facturi (nrfact, datafact, codcl) ;
    VALUES (1120, (^2000/08/07}, 1001)
INSERT INTO facturi (nrfact, datafact, codcl) ;
    VALUES (1121, (^2000/08/07}, 1004)
INSERT INTO facturi (nrfact, datafact, codcl) ;
    VALUES (1122, (^2000/08/07}, 1005)

```

Comanda INSERT permite și adăugarea unor linii rezultate dintr-o consultare SQL (SELECT). Aici apar câteva diferențe între VFP, Oracle și DB2, pe care le vom discuta în capitolul 6.

Listing 3.4. Script Oracle 8 de populare a tabelelor

```

DELETE FROM incasfact ;
DELETE FROM incasari ;
DELETE FROM liniifact ;
DELETE FROM facturi ;
DELETE FROM produse ;
DELETE FROM persclienti ;
DELETE FROM persoane ;
DELETE FROM clienti ;
DELETE FROM localitati ;
DELETE FROM judete ;

INSERT INTO judete VALUES ('IS', 'Iasi', 'Moldova') ;
INSERT INTO judete VALUES ('VN', 'Vrancea', 'Moldova') ;
INSERT INTO judete VALUES ('NT', 'Neamt', 'Moldova') ;
INSERT INTO judete VALUES ('SV', 'Suceava', 'Moldova') ;
INSERT INTO judete VALUES ('VS', 'Vaslui', 'Moldova') ;
INSERT INTO judete VALUES ('TM', 'Timis', 'Banat') ;

INSERT INTO localitati VALUES ('6600', 'Iasi', 'IS') ;
INSERT INTO localitati VALUES ('5725', 'Pascani', 'IS') ;
INSERT INTO localitati VALUES ('6500', 'Vaslui', 'VS') ;
INSERT INTO localitati VALUES ('5300', 'Focsani', 'VN') ;
INSERT INTO localitati VALUES ('6400', 'Birlad', 'VS') ;
INSERT INTO localitati VALUES ('5800', 'Suceava', 'SV') ;
INSERT INTO localitati VALUES ('5550', 'Roman', 'NT') ;
INSERT INTO localitati VALUES ('1900', 'Timisoara', 'TM') ;

INSERT INTO clienti VALUES (1001, 'Client 1 SRL', 'R1001',
                           'Tranzitiei, 13 bis', '6600', NULL) ;
INSERT INTO clienti (codcl, dencl, codfiscal, codpost, telefon)
VALUES (1002, 'Client 2 SA', 'R1002', '6600', '032-212121') ;
INSERT INTO clienti VALUES (1003, 'Client 3 SRL', 'R1003',
                           'Prosperitatii, 22', '6500', '035-222222') ;
INSERT INTO clienti (codcl, dencl, adresa, codpost)
VALUES (1004, 'Client 4', 'Sapientei, 56', '5725') ;
VALUES (1004, 'Client 4', 'Sapientei, 56', '5725')
INSERT INTO clienti VALUES (1005, 'Client 5 SRL', 'R1005', NULL,
                           '1900', '056-111111') ;
INSERT INTO clienti VALUES (1006, 'Client 6 SA', 'R1006',
                           'Pacientei, 33', '5550', NULL) ;
INSERT INTO clienti VALUES (1007, 'Client 7 SRL', 'R1007',
                           'Victoria Capitalismului, 2', '1900', '056-121212') ;

INSERT INTO persoane VALUES ('CNP1', 'Ioan', 'Vasile',
                            'I.L.Caragiale, 22', 'B', '6600', '123456', '987654',
                            '09422222', NULL) ;
INSERT INTO persoane VALUES ('CNP2', 'Vasile', 'Ion', NULL, 'B',
                            '6600', '234567', '876543', '09422223', 'Ion@.ro') ;
INSERT INTO persoane VALUES ('CNP3', 'Popovici', 'Ioana',
                            'V.Micle, Bl.I, Sc.B, Ap.2', 'F', '5725', '345678', NULL,
                            '09422224', NULL) ;

```

```

INSERT INTO persoane VALUES ('CNP4', 'Lazar', 'Caraiion',
                            'M.Eminescu, 42', 'B', '6500', '456789', NULL,
                            '09422225', NULL) ;
INSERT INTO persoane VALUES ('CNP5', 'Iurea', 'Simion',
                            'I.Creanga, 44 bis', 'B', '6500', '567890', '543210',
                            NULL, NULL) ;
INSERT INTO persoane VALUES ('CNP6', 'Vasc', 'Simona',
                            'M.Eminescu, 13', 'F', '5725', NULL, '432109',
                            '09422227', NULL) ;
INSERT INTO persoane VALUES ('CNP7', 'Popa', 'Ioanid',
                            'I.Ion, Bl.H2, Sc.C, Ap.45', 'B', '1900', '789012',
                            '321098', NULL, NULL) ;
INSERT INTO persoane VALUES ('CNP8', 'Bogacs', 'Ildiko',
                            'I.Viteazu, 67', 'F', '5550', '890123', '210987',
                            '09422229', NULL) ;
INSERT INTO persoane VALUES ('CNP9', 'Ioan', 'Vasilica',
                            'Garii, Bl.B4, Sc.A, Ap.1', 'F',
                            '1900', '901234', '109876', '09422230', NULL) ;

INSERT INTO persclienti VALUES ('CNP1', 1001, 'Director general') ;
INSERT INTO persclienti VALUES ('CNP2', 1002, 'Director general') ;
INSERT INTO persclienti VALUES ('CNP3', 1002, 'Sef aprovisionare') ;
INSERT INTO persclienti VALUES ('CNP4', 1003, 'Sef aprovisionare') ;
INSERT INTO persclienti VALUES ('CNP5', 1003, 'Director financiar') ;
INSERT INTO persclienti VALUES ('CNP6', 1004, 'Director general') ;
INSERT INTO persclienti VALUES ('CNP7', 1005, 'Sef aprovisionare') ;
INSERT INTO persclienti VALUES ('CNP8', 1006, 'Director financiar') ;
INSERT INTO persclienti VALUES ('CNP9', 1007, 'Sef aprovisionare') ;

INSERT INTO produse VALUES (1, 'Produs 1', 'buc', 'Tigari', .19) ;
INSERT INTO produse VALUES (2, 'Produs 2', 'kg', 'Bere', 0.19) ;
INSERT INTO produse VALUES (3, 'Produs 3', 'kg', 'Bere', 0.19) ;
INSERT INTO produse VALUES (4, 'Produs 4', 'l', 'Dulciuri', .19) ;
INSERT INTO produse VALUES (5, 'Produs 5', 'buc', 'Tigari', .19) ;

INSERT INTO facturi (nrfact, datafact, codcl)
VALUES (1111, TO_DATE('01/08/2000', 'DD/MM/YYYY'), 1001) ;
INSERT INTO facturi
VALUES (1112, TO_DATE('01/08/2000', 'DD/MM/YYYY'), 1005,
       'Probleme cu transportul') ;
INSERT INTO facturi (nrfact, datafact, codcl)
VALUES (1113, TO_DATE('01/08/2000', 'DD/MM/YYYY'), 1002) ;
INSERT INTO facturi (nrfact, datafact, codcl)
VALUES (1114, TO_DATE('01/08/2000', 'DD/MM/YYYY'), 1006) ;
INSERT INTO facturi (nrfact, datafact, codcl)
VALUES (1115, TO_DATE('02/08/2000', 'DD/MM/YYYY'), 1001) ;
INSERT INTO facturi
VALUES (1116, TO_DATE('02/08/2000', 'DD/MM/YYYY'), 1007,
       'Pretul propus initial a fost modificat') ;
INSERT INTO facturi (nrfact, datafact, codcl)
VALUES (1117, TO_DATE('03/08/2000', 'DD/MM/YYYY'), 1001) ;
INSERT INTO facturi (nrfact, datafact, codcl)
VALUES (1118, TO_DATE('04/08/2000', 'DD/MM/YYYY'), 1001) ;
INSERT INTO facturi (nrfact, datafact, codcl)
VALUES (1119, TO_DATE('07/08/2000', 'DD/MM/YYYY'), 1003) ;

```

Visual FoxPro este mai puțin pretențios la modificarea lungimii atributelor; acestea pot fi deopotrivă mărite sau micșorate; bineînțeles, la micșorare, trebuie avută în vedere trunchierea ce operează inevitabil.

ALTER TABLE PERSOANE ALTER COLUMN Nume CHAR(21) ch1(20)
dar și

ALTER TABLE PERSOANE ALTER COLUMN Nume CHAR(17) ch1(20)
măscata

Adăugarea/modificarea valorii implicate

Declararea valorii implicate a unui atribut în DB2 este posibilă numai la crearea tabeli sau la adăugarea atributului.

În Oracle lucrurile sunt mai simple. Dacă se dorește ca în liniile ce urmează să fie adăugate valoarea implicită a atributului Sex să fie 'F' (de la Femeie), comanda utilizată va fi:

ALTER TABLE PERSOANE MODIFY (Sex DEFAULT 'F')

Pentru anularea oricărei valori implicate se poate folosi:

ALTER TABLE PERSOANE MODIFY (Sex DEFAULT NULL)

Bineînțeles, atributele pentru care se declară NULL ca valoare implicită trebuie să „suportă” această (meta)valoare.

Și VFP permite modificarea valorilor implicate, prin:

ALTER TABLE PERSOANE ALTER COLUMN Sex SET DEFAULT 'F'

NULL și neNULL

Pentru unele atribută poate fi instituită la crearea tabeli obligativitatea valorilor nenele. În timp, aceasta poate fi modificată într-un sens sau în celălalt. DB2 nu este prea flexibil în acestă privință, în schimb Oracle și VFP da. Interzicerea valorilor nule pentru atributul Sex se realizează astfel:

- în Oracle:

ALTER TABLE PERSOANE MODIFY (Sex NOT NULL)

- în VFP:

ALTER TABLE PERSOANE ALTER COLUMN Sex NOT NULL

Invers, pentru a permite valori NULL pentru același atribut:

- în Oracle:

ALTER TABLE PERSOANE MODIFY (Sex NULL)

- în VFP:

ALTER TABLE PERSOANE ALTER COLUMN Sex NULL

Adăugarea/anularea restricțiilor

Toate restricțiile: cheie primară – PRIMARY KEY, unicitate – UNIQUE, referențială – FOREIGN KEY, de comportament – CHECK pot fi declarate ulterior creării tabeli și,

bineînțeles, anulate la un moment dat. Spre exemplu, formatul general al comenzi pentru dezactivarea cheii primare, comun celor trei SGBD-uri, este:

ALTER TABLE PERSOANE DROP PRIMARY KEY

De remarcat că, din cele trei produse, numai Oracle s-a „opus” vehement comenzi, motivând că există restricții referențiale declarate pe baza acestei chei primare. Pentru reînștiuirea cheii primare a tabelei PERSOANE comanda are forma:

ALTER TABLE PERSOANE ADD PRIMARY KEY (CNP)

Analog, prin ADD și DROP pot fi instituite/anulate și celelalte tipuri de restricții. Pentru o mai simplă referire, este utilă folosirea clauzei CONSTRAINT (în DB2 și Oracle) prin care se acordă un nume-utilizator fiecărei restricții. Altintinderi, pentru afilarea numelui restricției ce trebuie anulată este necesară consultarea catalogului sistem.

Stergerea tabelelor

Comanda DROP TABLE șterge o comandă din baza de date. Sintaxa acesteia nu ridică probleme deosebite:

DROP <nume tabelă> < comportament la ștergere >

unde:

< comportament la ștergere > : : = = RESTRICT | CASCADE

3.4. Inserarea, modificarea, ștergerea liniilor

SQL prezintă comenzi dedicate modificării conținutului unei tabele, înțelegând prin aceasta trei acțiuni prin care se actualizează baza:

1. adăugarea de noi linii la cele existente în tabelă,
2. ștergerea unor linii,
3. modificarea valorii unui atribut.

Adăugarea unei linii

Comanda SQL de adăugare de noi linii este INSERT, care are, în modul de lucru cel mai puțin pretențios, următorul format:

INSERT INTO tabelă [(atribut1, atribut2, ...)]
VALUES (valoare_atribut1, valoare_atribut2, ...)

În ceea ce privește scriptul de populare a tabelelor bazei de date cu valorile prezentate în primul capitol, diferențele dintre DB2, Oracle și VFP sunt minime, așa încât nu prezentăm decât varianta Oracle din listingul 3.4.

```

CHECK (BETWEEN(datadoc,
    BETWEEN(datafact,(^2000/01/01),(^2010/12/31)));
    ERROR 'Data documentului trebuie sa fie intre ;
    1 ian.2000 si 31 dec.2010 !';
);

CREATE TABLE incasfact (
    codinc NUMBER(8),
    nrfact NUMBER(8),
    transa NUMBER(16),
    NOT NULL,
    PRIMARY KEY STR(codinc,8)+STR(nrfact,8) TAG primar,
    FOREIGN KEY codinc TAG codinc,
    REFERENCES incasari TAG codinc,
    FOREIGN KEY nrfact TAG nrfact REFERENCES facturi TAG nrfact;
);

```

Nefiind un SGBDR de categoria grea, crearea bazei de date în VFP este un lucru ușor, fiind necesară o singură comandă, **CREATE DATABASE**. O opțiune interesantă la crearea tabelelor ține de faptul că, în afară de declararea restricțiilor, în VFP se poate stabili și mesajul afișat la violarea restricției respective.

De asemenea, în VFP se indică explicit ce atribute pot avea valori nule (în Oracle și DB2 se indică explicit cele care nu pot avea valori NULL). În plus, la atributele de tip numeric (reale), în numărul total de caractere o poziție (distincță) se contorizează pentru marca zecimală.

La declararea cheilor primare compuse, atributele componente trebuie concatenate, lucru ce atrage necesitatea conversiei celor de alt tip (numerice, dată calendaristică, logice) în siruri de caractere. Declararea cheilor primare, alternative și străine în VFP presupune crearea automată a indecsilor de tip PRIMARY, CANDIDATE sau REGULAR. Prin clauza TAG se poate da un nume la alegere indexului respectiv.

Din păcate, declararea cheilor străine nu înseamnă și instituirea restricției referențiale. Aceasta este una dintre cele mai ciudate „găselinje” VFP. Ca urmare, după crearea BD, fie trebuie „umblat” prin *Referential Integrity Builder* și, astfel, grafic, instituite regulile pentru prezervarea referențialității, fie trebuie create declanșatoare în acest scop (vezi capitolul 7).

Modificarea structurii tabelelor/restricțiilor în Oracle, DB2 și VFP

Schema unei baze de date reprezintă aspectul constant, invariabil sau, mai bine zis, puțin variabil în timp. O bună analiză și proiectare a aplicației (sistemuilui) diminuează riscul modificărilor de amploare în structura tabelor și restricții, conferind stabilitate bazei și diminuând sensibil eforturile de întreținere ulterioară instalării aplicației.

Cu toate acestea, apariția unui atribut nou, modificarea lungimii unui atribut sunt probleme de care orice analist/proiectant sau administrator/dezvoltator de aplicații și BD să cioneze cel puțin o dată (pe lună – e o glumă, firește!).

La aceste situații putem adăuga operațiunile diverse – **încarcarea bazei dintr-o altă aplicație, salvări, restaurări etc.** – operațiuni în care este necesară dezactivarea temporară a unor restricții și reactivarea lor ulterior. În plus, destui practicieni obișnuiesc să creeze mai întâi toate tabelele (declararea atributelor) și apoi să definească, printr-un script special, toate restricțiile bazei.

Astfel încât SQL prezintă o comandă dedicată modificării structurii bazei de date:

ALTER TABLE

ALTER TABLE <nume tabelă> <acțiune modificatoare a tabelei>

unde:

```

< acțiune modificatoare a tabelei > : : = =
    ADD [COLUMN] <definiție coloană>
    | ALTER [COLUMN] <nume coloană>
        <acțiune modificatoare a coloanei>
    | DROP [COLUMN] <nume coloană> <comportament la stergere>
    | ADD <definiție restricție a tabelei>
    | DROP CONSTRAINT <nume restricție>
        < comportament la stergere >

```

Comportamentul la stergere se referă la prohibirea sau propagarea în cascadă a stergerii în înregistrările/tabelele-copii:

< comportament la stergere > : : = = RESTRICT | CASCADE

Adăugarea unui nou atribut

Adăugarea atributului DataNast (data nașterii) în tabela PERSOANE se realizează identic în toate cele trei produse, DB2/Oracle/VFP:

ALTER TABLE PERSOANE ADD DataNast DATE

Stergerea unui atribut

Curios sau nu, nici DB2, nici Oracle nu permit stergerea unui atribut prin ALTER TABLE. În schimb, VFP este mai generos:

ALTER TABLE PERSOANE DROP COLUMN DataNast

Ba chiar, în VFP un atribut se poate și redenumi:

ALTER TABLE PERSOANE RENAME COLUMN DataNast TO DataMasterii

Modificarea tipului/lungimii unui atribut

DB2 permite modificarea lungimii numai pentru atributele de tip VARCHAR. Pentru a crește dimensiunea atributului Nume în tabela PERSOANE la 21 de caractere se folosește:

ALTER TABLE PERSOANE ALTER Nume SET DATA TYPE VARCHAR(21)

În Oracle, modificarea tipului unui atribut se poate face astfel: din CHAR în VARCHAR2 sau VARCHAR; din VARCHAR2 sau VARCHAR în CHAR, dar numai dacă respectivul atribut conține valoarea NULL în toate liniile tabelei sau dacă nu se modifică și lungimea atributului. Lungimea poate fi mărită pentru orice atribut fără probleme; în schimb, micșorarea sa poate avea loc numai când valorile atributului respectiv sunt NULL. Pentru atingerea aceluiși scop ca în precedentă ALTER TABLE din DB2, în Oracle comanda are forma:

ALTER TABLE PERSOANE MODIFY (Nume VARCHAR2(21))

```

CREATE TABLE persoane ( ;
cnp CHAR(14) ;
PRIMARY KEY ;
CHECK (cnp=LTRIM(UPPER(cnp))) ;
ERROR 'Codul numeric personal se scrie fara ;
spatii la inceput !', ;
nume CHAR(20) ;
CHECK (nume=LTRIM(PROPOR(nume))) ;
ERROR 'Prima litera din fiecare cuvint al'+CHR(13)+;
'numelui este majuscula; '+CHR(13)+;
'restul literelor sunt mici!', ;
prenume CHAR(20) ;
CHECK (prenume=LTRIM(PROPOR(prenume))) ;
ERROR 'Prima litera din fiecare cuvint al'+CHR(13)+;
'prenumele lui este majuscula; '+CHR(13)+;
'restul literelor sunt mici!', ;
adresa CHAR(40) ,
NULL ;
CHECK (SUBSTR(adresa,1,1) = UPPER(SUBSTR(adresa,1,1))) ;
ERROR 'Prima litera din adresa este obligatoriu ;
majuscula !', ;
sex CHAR(1) DEFAULT 'B' ;
CHECK (INLIST(sex,'F','B')) ;
ERROR 'Atributul Sex poate avea valorile F ;
(de la Femeiesc)'+CHR(13)+;
'sau B (de la Barbatesc) !', ;
codpost CHAR(5) ;
telcasca CHAR(10) NULL, ;
telbirou CHAR(10) NULL, ;
telmobil CHAR(10) NULL, ;
email CHAR(20) NULL, ;
FOREIGN KEY codpost TAG codpost ;
REFERENCES localitati TAG codpost ;
) ;

CREATE TABLE persclienti ( ;
cnp CHAR(14), ;
codcl NUMBER(6), ;
functie CHAR(25) ;
CHECK (SUBSTR(functie,1,1) = UPPER(SUBSTR(functie,1,1))) ;
ERROR 'Prima litera din functie este obligatoriu ;
majuscula !', ;
PRIMARY KEY cnp+STR(codcl,6)+functie TAG primar, ;
FOREIGN KEY cnp TAG cnp REFERENCES persoane TAG cnp, ;
FOREIGN KEY codcl TAG codcl REFERENCES clienti TAG codcl ;
) ;

CREATE TABLE produse ( ;
codpr NUMBER(6) ;
PRIMARY KEY ;
CHECK (codpr > 0) ;
ERROR 'Codul produsului trebuie sa fie mai mare ;
ca zero !', ;
denpr CHAR(30) ;

```

```

CHECK (SUBSTR(denpr,1,1) = UPPER(SUBSTR(denpr,1,1))) ;
ERROR 'Prima litera din denumirea produsului ;
este obligatoriu majuscula !', ;
um CHAR(10), ;
grupa CHAR(15) ;
CHECK (SUBSTR(grupa,1,1) = UPPER(SUBSTR(grupa,1,1))) ;
ERROR 'Prima litera din grupa este obligatoriu ;
majuscula !', ;
procTVA NUMBER(3,2) ;
DEFAULT .22 ;
)

CREATE TABLE facturi ( ;
nrfact NUMBER(8) ;
PRIMARY KEY, ;
datafact DATE ;
DEFAULT DATE() ;
CHECK (BETWEEN(datafact,(^2000/08/01),(^2010/12/31))) ;
ERROR 'Baza de date functioneaza in intervalul ;
1 aug.2000 - 31 dec.2010 !', ;
codcl NUMBER(6), ;
Obs CHAR(50) NULL, ;
FOREIGN KEY codcl TAG codcl REFERENCES clienti TAG codcl ;
) ;

CREATE TABLE liniifact ( ;
nrfact NUMBER(8), ;
linie NUMBER(2) ;
CHECK (linie > 0) ;
ERROR 'Atributul linie trebuie sa fie mai mare ;
ca zero !', ;
codpr NUMBER(6), ;
cantitate NUMBER(10), ;
pretunit NUMBER(12), ;
PRIMARY KEY STR(nrfact,8)+STR(linie,2) TAG primar, ;
FOREIGN KEY nrfact TAG nrfact REFERENCES facturi TAG nrfact, ;
FOREIGN KEY codpr TAG codpr REFERENCES produse TAG codpr ;
) ;

CREATE TABLE incasari ( ;
codinc NUMBER(8) ;
PRIMARY KEY, ;
datainc DATE ;
DEFAULT DATE() ;
CHECK (BETWEEN(datainc, ;
BETWEEN(datafact,(^2000/08/01),(^2010/12/31))) ;
ERROR 'Baza de date functioneaza in intervalul ;
1 aug.2000 - 31 dec.2010 !', ;
coddoc CHAR(4) ;
CHECK(coddoc=UPPER(LTRIM(coddoc))) ;
ERROR 'Codul documentului se scrie cu majuscu !', ;
nrdoc CHAR(16), ;
datadoc DATE ;
DEFAULT DATE() - 7 ;
)
```

NOCPTRANS

Este utilă pentru câmpuri de tip sir de caractere și memo, pentru a preveni conversia la un alt cod de pagină (cod page).

```
PRIMARY KEY eExpression2 TAG TagName2
```

Creează automat un index primar pentru tabela curentă. eExpression2 este un câmp sau o combinație de câmpuri ale tablei. Numele indexului poate conține până la 10 caractere. Firește, clauza PRIMARY KEY nu poate apărea decât o singură dată într-o comandă CREATE TABLE.

```
UNIQUE eExpression3 TAG TagName3
```

Creează un index candidat. eExpression3 desemnează orice câmp sau combinație de câmpuri ale tablei, atribut/combinare care va îndeplini rolul de cheie alternativă. TagName3 reprezintă numele indexului candidat al căruia nume este alcătuit tot din maximum 10 caractere. Pentru o aceeași tabelă pot fi creați mai mulți indecsi candidați.

```
FOREIGN KEY eExpression4 TAG TagName4 [NODUP]
```

Are ca rezultat crearea unui index obișnuit (*regular*) pentru tabela curentă (cheia de indexare fiind eExpression4, iar numele acestuia TagName4) și stabilirea unei relații permanente cu o tabelă-părinte. Prin NODUP se poate crea un index străin candidat (prin analogie cu indecsi candidați la „postul” de index primar).

```
REFERENCES TableName3 [TAG TagName5]
```

Specifică numele tablei-părinte implicate în legătura creată prin opțiunea FOREIGN KEY. Indexul TagName5 este cel prin care se realizează legătura dintre cele două table. Implicit este indexul primar al tablei TableName3.

```
CHECK eExpression2 [ERROR cMessageText2]
```

Permite specificarea unei restricții (reguli de validare) la nivel de tabelă. cMessageText2 este mesajul afișat în cazul nerespectării restricției.

```
FROM ARRAY ArrayName
```

Permite crearea unei tabele pe baza datelor conținute într-o variabilă de tip tablou. Pentru comparație, în listingul 3.3 este prezentat programul pentru crearea în VFP 6 a bazei de date VÂNZĂRI.

Listing 3.3. Crearea tabelelor bazei de date în VFP

```
CLOSE DATA
CLOSE TABLES ALL
DELETE DATABASE VINZARI DELETETABLES
CREATE DATABASE vinzari
CREATE TABLE judete (
    jud CHAR(2) ,
    PRIMARY KEY ,
    CHECK (jud=LTRIM(UPPER(jud))) ,
    ERROR 'Indicativul județului se scrie cu majuscule!',;
```

```
judet CHAR(25) ;
    UNIQUE ;
    NOT NULL ;
    CHECK (judet=LTRIM(PROPER(judet))) ;
    ERROR 'Prima literă din fiecare cuvint al'+CHR(13)+;
        'denumirii județului este majusculă; '+CHR(13)+;
        'restul literelor sunt mici!', ;
    regiune CHAR(15) ;
    DEFAULT 'Moldova' ;
    CHECK (INLIST(regiune, 'Banat', 'Transilvania',;
        'Dobrogea', 'Oltenia', 'Muntenia', 'Moldova')) ;
    ERROR 'Regiunea poate avea o valoare numai din lista:';
        +CHR(13)+'Banat, Transilvania, Dobrogea, ;
        Oltenia, Muntenia, Moldova' ;
)

CREATE TABLE localitati ( ;
    codpost CHAR(5) ,
    PRIMARY KEY ,
    CHECK (codpost=LTRIM(codpost)) ;
    ERROR 'Codul postal se scrie fără ;
        spații la început !', ;
    loc CHAR(25) ;
    NOT NULL ;
    CHECK (loc=LTRIM(PROPER(loc))) ;
    ERROR 'Prima literă din fiecare cuvint ;
        al'+CHR(13)+;
        'denumirii localitatii este majusculă;
        '+CHR(13)+;
        'restul literelor sunt mici!', ;
    jud CHAR(2) ;
    DEFAULT 'IS' ;
    FOREIGN KEY jud TAG jud REFERENCES judete TAG jud ;
)

CREATE TABLE clienti ( ;
    codcl NUMBER(6) ,
    PRIMARY KEY ,
    CHECK (codcl > 1000) ;
    dencl CHAR(30) ;
    CHECK(SUBSTR(dencl,1,1)=UPPER(SUBSTR(dencl,1,1))), ;
    codfiscal CHAR(9) ;
    NULL ;
    CHECK (SUBSTR(codfiscal,1,1)=;
        UPPER(SUBSTR(codfiscal,1,1))), ;
    adresa CHAR(40) ;
    NULL ;
    CHECK (SUBSTR(adresa,1,1) =;
        UPPER(SUBSTR(adresa,1,1))), ;
    codpost CHAR(5) ,
    telefon CHAR(10) ;
    NULL ;
    FOREIGN KEY codpost TAG codpost ;
    REFERENCES localitati TAG codpost ;
)
```

Nici în DB2 nu există, pentru restricțiile referențiale, opțiunea de actualizare în cascadă a valorilor atributului-părinte în înregistrările-copil. Clauza ON UPDATE prezintă două variante: ON UPDATE NO ACTION și ON UPDATE RESTRICT. Prima este periculoasă, deoarece lasă orfane înregistrările-copil.

Cât privește ștergerea unei înregistrări-părinte, se poate recurge la una din trei variante: ON DELETE NO ACTION (cea mai puțin recomandabilă), ON DELETE CASCADE și ON DELETE SET NULL. După cum am văzut în primul capitol, în virtutea restricției referențiale, este interzisă orice valoare nenulă într-o înregistrare-copil (pentru atributul cheie străină) care să nu se regăsească în tabela-părinte (tabela în care, de obicei, atributul de legătură este cheie primară sau candidată). Aceasta înseamnă că sunt admise înregistrările-copil orfane, dar numai la NULL-itatea cheii străine. Este ideea ultimei variante declarative – ON DELETE SET NULL: la ștergerea unei linii-părinte, în linile copil atributele de legătură vor primi automat valoarea NULL.

Crearea tabelelor și declararea restricțiilor în Visual FoxPro

Versiunea 3 și noua denumire, Visual FoxPro, marchează momentul unei transformări radicale, din toate punctele de vedere, a produsului FoxPro. Noțiunea de bază de date devine una efectivă și una de complezență. Restricțiile și procedurile stocate sunt memorate în containerul bazei, forma „foxistă” a dicționarului de date și a mult mai complicațialui catalog de sistem din serverele de bază de date.

Formatul general al comenzi CREATE TABLE în Visual FoxPro 3.0 este:

```
CREATE TABLE | DBF TableName1 [NAME LongTableName] [FREE]
  (FieldName1 FieldType [(nFieldWidth [, nPrecision])]
   [NULL | NOT NULL]
   [CHECK lExpression1 [ERROR cMessageText1]]
   [DEFAULT eExpression1]
   [PRIMARY KEY | UNIQUE]
   [REFERENCES TableName2 [TAG TagName1]]
   [NOCPTRANS]
   [, FieldName2 ...]
   [, PRIMARY KEY eExpression2 TAG TagName2]
   [, UNIQUE eExpression3 TAG TagName3]
   [, FOREIGN KEY eExpression4 TAG TagName4 [NODUP]
     REFERENCES TableName3 [TAG TagName5]]
   [, CHECK lExpression2 [ERROR cMessageText2]])
| FROM ARRAY ArrayName
```

Scurtă descriere a argumentelor:

CREATE TABLE | DBF TableName1

TableName1 reprezintă numele tabeliei ce urmează să fie creată. De remarcat că nu există nici o diferență între opțiunile TABLE și DBF. Prima se adresează celor atașați teoriei relaționale, iar cea de-a doua este mai aproape de utilizatorul „tradițional” al FoxPro (XBase-urilor, în general), pentru care o tabelă este un fișier cu extensia .DBF.

NAME LongTableName

Permite specificarea unui nume mai lung (până la 128 de caractere) pentru tabela creată. Pentru aceasta, este necesar ca baza de date să fie în prealabil deschisă, deoarece numele lungi sunt memorate în containerul asociat bazei (.DBC).

FREE

Indică faptul că tabela respectivă va fi independentă, deci nu va face parte din bază.

(FieldName1 FieldType [(nFieldWidth [, nPrecision])])

Permite declararea, pentru fiecare atribut (câmp) al tablei, a numelui, tipului, lungimii și, eventual, a numărului de poziții pentru reprezentarea părții fracționare (zecimale).

NULL

Prin specificarea sa, atributul este autorizat să conțină valori nule (NULL).

NOT NULL

Previne apariția valorilor nule pentru atributul respectiv. Automat, pentru atributele de tip cheie primară sau pentru care este utilizată opțiunea UNIQUE, nu se admit valori NULL.

CHECK lExpression1

Servește la specificarea unei funcții-utilizator de validare la nivel de atribut (câmp). Astfel, funcția este verificată imediat după adăugarea unei noi înregistrări (linii) în tabelă. Dacă rezultatul evaluării funcției este false, se declanșează o eroare.

ERROR cMessageText1

În cazul în care funcția de validare de la nivelul atributului nu se respectă (adică „întoarce” valoarea logică FALSE), pe ecran apare mesajul cMessageText1.

DEFAULT eExpression1

Specifică valoarea implicită a atributului (în funcție de tipul acestuia), valoare pe care o conține acest câmp la adăugarea unei noi înregistrări în tabelă.

PRIMARY KEY

Declară atributul respectiv cheie primară, prin crearea unui index principal cu nume identic cu același atribut.

UNIQUE

Creează un „index candidat”, altfel spus, declară acest atribut cheie alternativă. Se previne astfel apariția a două valori identice în tabelă pentru acest câmp.

REFERENCES TableName2 [TAG TagName1]

Permite definirea unei restricții referențiale prin crearea unei legături permanente între table. TableName2 este tabela-părinte a legăturii. Dacă nu se specifică TAG TagName1, relația este stabilită prin intermediul indexului primar al tablei TableName2; dacă acesta (indexul primar) nu există, se generează un mesaj de eroare. TableName2 nu poate fi o tabelă independentă.

```

loc VARCHAR(25)
    CONSTRAINT nn_loc NOT NULL,
jud CHAR(2)
    DEFAULT 'IS'
    CONSTRAINT fk_loc_jud REFERENCES judete(jud)
) ;

CREATE TABLE clienti (
codcl DECIMAL(6) NOT NULL
    CONSTRAINT pk_clienti PRIMARY KEY
    CONSTRAINT ck_codcl CHECK (codcl > 1000),
dencl VARCHAR(30)
    CONSTRAINT ck_dencl CHECK (SUBSTR(dencl,1,1) =
        UPPER(SUBSTR(dencl,1,1))),
codfiscal CHAR(9)
    CONSTRAINT ck_codfiscal CHECK (SUBSTR(codfiscal,1,1) =
        = UPPER(SUBSTR(codfiscal,1,1))),
adresa VARCHAR(40)
    CONSTRAINT ck_adresa_clienti CHECK
        (SUBSTR(adresa,1,1) = UPPER(SUBSTR(adresa,1,1))),
codpost CHAR(5)
    CONSTRAINT fk_cl_loc REFERENCES localitati(codpost),
telefon VARCHAR(10)
) ;

CREATE TABLE persoane (
cnp CHAR(14) NOT NULL
    CONSTRAINT pk_persoane PRIMARY KEY
    CONSTRAINT ck_cnp CHECK (cnp=LTRIM(UPPER(cnp))),
nume VARCHAR(20),
prenume VARCHAR(20),
adresa VARCHAR(40)
    CONSTRAINT ck_adresa_persoane CHECK
        (SUBSTR(adresa,1,1) = UPPER(SUBSTR(adresa,1,1))),
sex CHAR(1) DEFAULT 'B'
    CONSTRAINT ck_sex CHECK (sex IN ('F','B')),
codpost CHAR(5)
    CONSTRAINT fk_pers_loc REFERENCES
        localitati(codpost),
telacasa VARCHAR(10),
telbirou VARCHAR(10),
telmobil VARCHAR(10),
email VARCHAR(20)
) ;

CREATE TABLE persclienti (
cnp CHAR(14) NOT NULL
    CONSTRAINT fk_perscl_pers REFERENCES persoane(cnp),
codcl DECIMAL(6) NOT NULL
    CONSTRAINT fk_perscl_cl REFERENCES clienti(codcl),
functie VARCHAR(25) NOT NULL
    CONSTRAINT ck_functie CHECK (SUBSTR(functie,1,1) =
        UPPER(SUBSTR(functie,1,1))),
CONSTRAINT pk_persclienti
    PRIMARY KEY (cnp, codcl, functie)
) ;

```

```

CREATE TABLE produse (
codpr DECIMAL(6) NOT NULL
    CONSTRAINT pk_produse PRIMARY KEY
    CONSTRAINT ck_codpr CHECK (codpr > 0),
denpr VARCHAR(30) CONSTRAINT ck_denpr
    CHECK (SUBSTR(denpr,1,1) = UPPER(SUBSTR(denpr,1,1))),
un VARCHAR(10),
grupa VARCHAR(15)
    CHECK (SUBSTR(grupa,1,1) = UPPER(SUBSTR(grupa,1,1))),
procTVA DECIMAL(2,2) DEFAULT .22
) ;

CREATE TABLE facturi (
nrfact DECIMAL(8) NOT NULL
    CONSTRAINT pk_facturi PRIMARY KEY,
datafact DATE DEFAULT CURRENT DATE,
    CONSTRAINT ck_datafact CHECK (datafact >= '8/1/2000'
        AND datafact <= '12/31/2010'),
codcl DECIMAL(6)
    CONSTRAINT fk_fact_cl REFERENCES clienti(codcl),
obs VARCHAR(50)
) ;

CREATE TABLE liniifact (
nrfact DECIMAL(8) NOT NULL
    CONSTRAINT fk_lf_fact REFERENCES facturi(nrfact),
linie DECIMAL(2) NOT NULL
    CONSTRAINT ck_linie CHECK (linie > 0),
codpr DECIMAL(6) CONSTRAINT fk_lf_prod
    REFERENCES produse(codpr),
cantitate DECIMAL(10),
pretunit DECIMAL (12),
CONSTRAINT pk_liniifact PRIMARY KEY (nrfact,linie)
) ;

CREATE TABLE incasari (
codinc DECIMAL(8) NOT NULL
    CONSTRAINT pk_incasari PRIMARY KEY,
datainc DATE
    CONSTRAINT ck_datainc CHECK (datainc >= '8/1/2000'
        AND datainc <= '12/31/2010'),
coddoc CHAR(4)
    CONSTRAINT ck_coddoc CHECK (coddoc=UPPER(LTRIM(coddoc))),
nrdoc VARCHAR(16),
datadoc DATE
    CONSTRAINT ck_datadoc CHECK (datadoc >= '8/1/2000'
        AND datadoc <= '12/31/2010')
) ;

CREATE TABLE incasfact (
codinc DECIMAL(8) NOT NULL CONSTRAINT fk_if_inc
    REFERENCES incasari(codinc),
nrfact DECIMAL(8) NOT NULL CONSTRAINT fk_inc_fact
    REFERENCES facturi(nrfact),
transa DECIMAL(16) NOT NULL,
CONSTRAINT pk_incasfact PRIMARY KEY (codinc, nrfact)
) ;

```

```

CREATE TABLE facturi (
    nrfact NUMBER(8)
        CONSTRAINT pk_facturi PRIMARY KEY,
    datafact DATE DEFAULT SYSDATE
        CONSTRAINT ck_datafact CHECK (datafact >=
            TO_DATE('01/08/2000','DD/MM/YYYY')
        AND datafact <=
            TO_DATE('31/12/2010','DD/MM/YYYY')),
    codcl NUMBER(6)
        CONSTRAINT fk_facturi_clienti
            REFERENCES clienti(codcl),
    Obs VARCHAR2(50)
) ;

CREATE TABLE liniifact (
    nrfact NUMBER(8)
        CONSTRAINT fk_liniifact_facturi
            REFERENCES facturi(nrfact),
    linie NUMBER(2)
        CONSTRAINT ck_linie CHECK (linie > 0),
    codpr NUMBER(6) CONSTRAINT fk_liniifact_produse
        REFERENCES produse(codpr),
    cantitate NUMBER(10),
    pretunit NUMBER(12),
    CONSTRAINT pk_liniifact PRIMARY KEY (nrfact,linie)
) ;

CREATE TABLE incasari (
    codinc NUMBER(8)
        CONSTRAINT pk_incasari PRIMARY KEY,
    datainc DATE DEFAULT SYSDATE
        CONSTRAINT ck_datainc CHECK (datainc >=
            TO_DATE('01/08/2000','DD/MM/YYYY')
        AND datainc <=
            TO_DATE('31/12/2010','DD/MM/YYYY')),
    coddoc CHAR(4)
        CONSTRAINT ck_codedoc
            CHECK(codedoc=UPPER(LTRIM(coddoc))),
    nrdoc VARCHAR2(16),
    datadoc DATE DEFAULT SYSDATE - 7
        CONSTRAINT ck_datadoc CHECK
            (datadoc >= TO_DATE('01/01/2000','DD/MM/YYYY')
        AND datadoc <=
            TO_DATE('31/12/2010','DD/MM/YYYY'))
) ;

CREATE TABLE incasfact (
    codinc NUMBER(8) CONSTRAINT fk_incasfact_incasari
        REFERENCES incasari(codinc),
    nrfact NUMBER(8) CONSTRAINT fk_incasfact_facturi
        REFERENCES facturi(nrfact),
    transa NUMBER(16) NOT NULL,
    CONSTRAINT pk_incasfact PRIMARY KEY (codinc, nrfact)
) ;

```

Să detaliem câteva dintre restricțiile declarate. Astfel, în tabela JUDEȚE:

- atributul Jud este cheie primară (restricția pk_judete);
- valorile lui Jud sunt exclusiv majuscule și nu conțin spații la început (ck_jud), scopul fiind de a folosi clauza CHECK și funcțiile LTRIM și UPPER;
- Județ este cheie alternativă (un_judet);
- Județ nu poate avea valori nule (nn_judet);
- inițiala (sau, după caz, inițialele) denumirii județului – Județ – este majusculă; restul sunt litere mici (ck_judet);
- valoarea implicită pentru Regiune este 'Moldova';
- Regiune nu poate avea decât una dintre valorile: 'Banat', 'Transilvania', 'Dobrogea', 'Oltenia', 'Muntenia', 'Moldova' (ck_regiune).



Crearea tabelelor și declararea restricțiilor în DB2

În afară de micile diferențe legate de tipurile de date și lucrul cu atrbute de tip data calendaristică, se cuvine menționat faptul că în DB2, spre deosebire de Oracle, atrbutele din alcătuirea cheii primare trebuie declarate explicit ca fiind nenele (NOT NULL). Scriptul (ceva mai modest) de creare a tabelelor bazei de date este prezentat în listingul 3.2.

Listing 3.2. Crearea tabelelor în DB2

```

DROP TABLE incasfact ;
DROP TABLE incasari ;
DROP TABLE liniifact ;
DROP TABLE facturi ;
DROP TABLE produse ;
DROP TABLE persclienti ;
DROP TABLE persoane ;
DROP TABLE clienti ;
DROP TABLE localitati ;
DROP TABLE judete ;

CREATE TABLE judete (
    jud CHAR(2) NOT NULL
        CONSTRAINT pk_jud PRIMARY KEY
        CONSTRAINT ck_jud CHECK (jud=UPPER(jud)),
    judet VARCHAR(25)
        CONSTRAINT un_judet UNIQUE
        CONSTRAINT nn_judet NOT NULL,
    regiune VARCHAR(15)
        DEFAULT 'Moldova'
        CONSTRAINT ck_regiune CHECK (regiune IN
            ('Banat', 'Transilvania', 'Dobrogea',
            'Oltenia', 'Muntenia', 'Moldova'))
) ;

CREATE TABLE localitati (
    codpost CHAR(5) NOT NULL
        CONSTRAINT pk_loc PRIMARY KEY
        CONSTRAINT ck_codpost CHECK (codpost=UPPER(codpost)),
    ...
)

```

În listing 3.1 este prezentat scriptul Oracle 8 pentru crearea tabelelor și declararea restricțiilor bazei de date.

Listing 3.1. Script Oracle 8 de creare a tabelelor și declarare a restricțiilor

```

DROP TABLE incasfact ;
DROP TABLE incasari ;
DROP TABLE liniifact ;
DROP TABLE facturi ;
DROP TABLE produse ;
DROP TABLE persclienti ;
DROP TABLE persoane ;
DROP TABLE clienti ;
DROP TABLE localitati ;
DROP TABLE judete ;

CREATE TABLE judete (
    jud CHAR(2)
        CONSTRAINT pk_judete PRIMARY KEY
        CONSTRAINT ck_jud CHECK (jud=LTRIM(UPPER(jud))),
    judet VARCHAR2(25)
        CONSTRAINT un_judet UNIQUE
        CONSTRAINT nn_judet NOT NULL
        CONSTRAINT ck_judet CHECK
            (judet=LTRIM(INITCAP(judet))),
    regiune VARCHAR2(15)
        DEFAULT 'Moldova'
        CONSTRAINT ck_regiune CHECK (regiune IN ('Banat',
            'Transilvania', 'Dobrogea', 'Oltenia',
            'Muntenia', 'Moldova'))
) ;

CREATE TABLE localitati (
    codpost CHAR(5)
        CONSTRAINT pk_localitati PRIMARY KEY
        CONSTRAINT ck_codpost CHECK (codpost=LTRIM(codpost)),
    loc VARCHAR2(25)
        CONSTRAINT nn_loc NOT NULL
        CONSTRAINT ck_loc CHECK (loc=LTRIM(INITCAP(loc))),
    jud CHAR(2)
        DEFAULT 'IS'
        CONSTRAINT fk_localitati_jud REFERENCES judete(jud)
) ;

CREATE TABLE clienti (
    codcl NUMBER(6)
        CONSTRAINT pk_clienti PRIMARY KEY
        CONSTRAINT ck_codcl CHECK (codcl > 1000),
    dencl VARCHAR2(30)
        CONSTRAINT ck_dencl CHECK (SUBSTR(dencl,1,1) =
            UPPER(SUBSTR(dencl,1,1))),
    codfiscal CHAR(9)
        CONSTRAINT ck_codfiscal CHECK (SUBSTR(codfiscal,1,1) =
            UPPER(SUBSTR(codfiscal,1,1))),
```

```

    adresa VARCHAR2(40)
        CONSTRAINT ck_adresa_clienti CHECK
            (SUBSTR(adresa,1,1) = UPPER(SUBSTR(adresa,1,1))),
    codpost CHAR(5)
        CONSTRAINT fk_clienti_localitati
            REFERENCES localitati(codpost),
    telefon VARCHAR2(10)
) ;

CREATE TABLE persoane (
    cnp CHAR(14)
        CONSTRAINT pk_persoane PRIMARY KEY
        CONSTRAINT ck_cnp CHECK (cnp=LTRIM(UPPER(cnp))),
    nume VARCHAR2(20)
        CONSTRAINT ck_nume CHECK (nume=LTRIM(INITCAP(nume))),
    prenume VARCHAR2(20)
        CONSTRAINT ck_prenume CHECK
            (prenume=LTRIM(INITCAP(prenume))),
    adresa VARCHAR2(40)
        CONSTRAINT ck_adresa_persoane CHECK
            (SUBSTR(adresa,1,1) = UPPER(SUBSTR(adresa,1,1))),
    sex CHAR(1) DEFAULT 'B'
        CONSTRAINT ck_sex CHECK (sex IN ('F','B')),
    codpost CHAR(5)
        CONSTRAINT fk_persoane_localitati
            REFERENCES localitati(codpost),
    telacasa VARCHAR2(10),
    telbirou VARCHAR2(10),
    telmobil VARCHAR2(10),
    email VARCHAR2(20)
) ;

CREATE TABLE persclienti (
    cnp CHAR(14)
        CONSTRAINT fk_persclienti_persoane
            REFERENCES persoane(cnp),
    codcl NUMBER(6)
        CONSTRAINT fk_persclienti_clienti
            REFERENCES clienti(codcl),
    functie VARCHAR2(25)
        CONSTRAINT ck_functie CHECK (SUBSTR(functie,1,1) =
            UPPER(SUBSTR(functie,1,1))),
    CONSTRAINT pk_persclienti
        PRIMARY KEY (cnp, codcl, functie)
) ;

CREATE TABLE produse (
    codpr NUMBER(6)
        CONSTRAINT pk_produse PRIMARY KEY
        CONSTRAINT ck_codpr CHECK (codpr > 0),
    denpr VARCHAR2(30) CONSTRAINT ck_denpr
        CHECK (SUBSTR(denpr,1,1) = UPPER(SUBSTR(denpr,1,1))),
    um VARCHAR2(10),
    grupa VARCHAR2(15)
        CHECK (SUBSTR(grupa,1,1) = UPPER(SUBSTR(grupa,1,1))),
    proctva NUMBER(2,2) DEFAULT .22
) ;
```

În SQL-92 se poate specifica și modul în care va fi păstrată integritatea bazei de date la ștergerea unei linii-părinte sau modificarea unei chei primare ce prezintă înregistrări-copil. Pentru a interzice ștergerea unor facturi (înregistrări din FACTURI) pentru care există tupluri corespondente în LINIIFACT și, pe de altă parte, pentru ca la modificarea unui număr de factură (NrFact) în FACTURI să se modifice automat, în cascădă, toate linile-copil din LINIIFACT, forma comenzi se schimbă în:

```
CREATE TABLE LINIIFACT (
    NrFact NUMERIC(8) NOT NULL,
    Linie SMALLINT NOT NULL,
    CodPr NUMERIC(6) NOT NULL,
    Cantitate NUMERIC(10) NOT NULL,
    PretUnit NUMBER (12),
    PRIMARY KEY (NrFact, Linie),
    UNIQUE (NrFact, CodPr),
    FOREIGN KEY NrFact REFERENCES FACTURI(NrFact)
        ON DELETE RESTRICT
        ON UPDATE CASCADE,
    FOREIGN KEY CodPr REFERENCES PRODUSE(CodPr)
        ON DELETE RESTRICT
        ON UPDATE CASCADE )
```

Restricții-utilizator

În SGBD-urile actuale, restricțiile-utilizator, denumite și restricții de comportament, sunt implementate sub forma regulilor de validare la nivel de câmp (field validation rule), la nivel de înregistrare (record validation rule) sau, eventual, pot fi incluse în declanșatoare (triggers).

```
< definiție regulă de validare > : : = =
[ CONSTRAINT < nume restricție > ]
[CHECK] ( < condiție > )
```

Cu rezerva că aceste reguli pot avea forme complexe și pot intrebuința elemente avansate de SQL și/sau extensiile procedurale ale SQL în mediul respectiv, prezentăm în continuare o regulă de validare pentru atributul DataFact în tabela FACTURI și o altă pentru atributul Linie în LINIIFACT.

```
CREATE TABLE FACTURI (
    NrFact NUMERIC(8) NOT NULL PRIMARY KEY,
    DataFact DATE DEFAULT SYSDATE NOT NULL
        CHECK (DataFact >= '01/08/2000' AND DataFact <=
        '31/12/2010'),
    CodCl DECIMAL(6) DEFAULT 1001 NOT NULL
        REFERENCES CLIENTI(CodCl) ON DELETE RESTRICT
        ON UPDATE CASCADE,
    Obs VARCHAR(50) )

CREATE TABLE LINIIFACT (
    NrFact NUMERIC(8) NOT NULL,
    Linie SMALLINT NOT NULL CHECK (Linie > 0))
```

```
CodPr NUMERIC(6) NOT NULL,
Cantitate NUMERIC(10) NOT NULL,
PretUnit NUMBER (12),
PRIMARY KEY (NrFact, Linie),
UNIQUE (NrFact, CodPr),
FOREIGN KEY NrFact REFERENCES FACTURI(NrFact)
    ON DELETE RESTRICT ON UPDATE CASCADE,
FOREIGN KEY CodPr REFERENCES PRODUSE(CodPr)
    ON DELETE RESTRICT ON UPDATE CASCADE )
```

Ca urmare a clauzei CHECK, data unei facturi nu poate avea valori în afara intervalului 1 august 2000 – 31 decembrie 2010, iar atributul Linie trebuie să prezinte valori întregi mai mari ca 0.

Crearea tabelelor și declararea restricțiilor în Oracle 8

Ar fi o întreagă aventură dacă am încerca să discutăm formatul general al comenzi CREATE TABLE în Oracle, și aceasta, deoarece, în afară de attribute și restricții, comanda permite specificarea unei serii întregi de parametri legați de definirea organizării tablei, spațiul-tablelă, caracteristicile de stocare, clustere, paralelism, partionare etc.

Astfel încât nu limităm discuția la câteva considerente generale. Fiecarei restricții (cheie primară, unicitate, regulă la nivel de atribut etc.) i se poate da un nume, lucru util atunci când, la un moment dat (salvări, restaurări, încărcarea BD), vrem să dezactivăm una sau mai multe dintre acestea.

Pentru a ușura lucrul, se prefixeză numele fiecărei restricții cu tipul său:

- **pk** (PRIMARY KEY) pentru cheile primare
- **un** (UNIQUE) pentru cheile alternative
- **nn** (NOT NULL) pentru attributele obligatorii (ce nu pot avea valori nule)
- **ck** (CHECK) pentru reguli de validare la nivel de atribut
- **fk** (FOREIGN KEY) pentru cheile străine.

Am preferat tipul NUMBER pentru attributele numerice, deși, cel puțin pentru coduri, era mai indicat tipul INTEGER (PLS_INTEGER). Pentru șiruri de caractere cu lungime variabilă, Oracle recomandă utilizarea tipului VARCHAR2. Lucrul cu attribute de tip DATE (dată calendaristică) necesită funcția de conversie TO_DATE(). În Oracle nu există tipul de attribute LOGICAL sau BOOLEAN (doar în PL/SQL o variabilă poate fi declarată BOOLEAN).

Atributele unei table pot avea definite valori implicate, însă acestea nu pot fi generate prin funcții-utilizator, proceduri sau sevențe. Declararea unei chei străine se realizează prin clauza REFERENCES. În Oracle (cel puțin până la versiunea 8) nu există clauză de actualizare în cascădă din tablele-părinte în cele copii (UPDATE CASCADE), singura opțiune care se poate utiliza fiind ON DELETE CASCADE, utilă pentru ștergerea simultană a unei înregistrări-părinte și a tuturor înregistrărilor-copil. Firește, prin mecanismul referențial, opțiunile implicate sunt ON UPDATE RESTRICT și ON DELETE RESTRICT.

Mai trebuie amintit faptul că unele SGBD-uri, precum VFP, permit calculul valorilor implicate pe baza unei funcții-utilizator (procedură stocată), iar altele, precum Oracle, permit folosirea secvențelor în declanșatoare, utile mai ales pentru attribute-cheie, numerice.

Declararea restricțiilor

În SQL se pot declara toate tipurile de restricții ale unei BDR: **valori obligatorii (nenule), unicitate, restricții referențiale și restricții-utilizator**. Continând detalierea formului general al comenzi **CREATE TABLE** în sintaxa SQL-92:

```
< restricție a coloanei > : : = NOT NULL
| < definiție regulă de validare >
| < specificație privind unicitatea >
| < specificație referențială >
```

Valori nenule

Unele atribute, obligatoriu cele din cheia primară, nu pot prezenta valori nule. Pentru aceasta, la crearea tabeli și declararea atributului se folosește opțiunea **NOT NULL**. În unele produse (DB2, Oracle) trebuie indicate explicit atributele ce nu pot avea valori nule, în timp ce în Visual FoxPro este invers, trebuie declarate atributele susceptibile de valori **NULL**.

Și între Oracle și DB2 există o mică deosebire: **în Oracle, pentru un atribut-cheie nu este obligatorie clauza NOT NULL**, în timp ce în DB2 este. Iată noua formă a comenzi pentru crearea tabeli FACTURI:

```
CREATE TABLE FACTURI (
    NrFact NUMERIC(8) NOT NULL,
    DataFact DATE DEFAULT SYSDATE NOT NULL,
    CodCl DECIMAL(6) DEFAULT 1001 NOT NULL,
    Obs VARCHAR(50) )
```

Cheie primară/unicitate

```
< specificație privind unicitatea > : : = UNIQUE | PRIMARY KEY
```

Cheia primară unei relații este definită prin clauza **PRIMARY KEY**, plasată fie imediat după atributul-cheie, fie după descrierea ultimului atribut al tabeli. A doua variantă este întrebuiată cu precădere atunci când cheia primară a tabeli este compusă.

Pentru atributele de tip cheie candidată se poate folosi clauza **UNIQUE**, care va asigura respectarea unicății valorilor. Iată câteva variante de folosire a clăuzelor **PRIMARY KEY** și **UNIQUE**:

```
CREATE TABLE FACTURI (
    NrFact NUMERIC(8) NOT NULL PRIMARY KEY,
    DataFact DATE DEFAULT SYSDATE NOT NULL,
    CodCl DECIMAL(6) DEFAULT 1001 NOT NULL,
    Obs VARCHAR(50) )
```

```
CREATE TABLE JUDETE (
```

```
Jud CHAR(2) PRIMARY KEY,
Judet VARCHAR(25) NOT NULL UNIQUE,
Regiune VARCHAR(15) )
```

```
CREATE TABLE LINIIFACT (
    NrFact NUMERIC(8) NOT NULL,
    Linie SMALLINT NOT NULL,
    CodPr NUMERIC(6) NOT NULL,
    Cantitate NUMERIC(10) NOT NULL,
    PretUnit NUMBER(12),
    PRIMARY KEY (NrFact, Linie),
    UNIQUE (NrFact, CodPr) )
```

Pentru tabela **LINIIFACT**, cheia primară este combinația de atribute (**NrFact, Linie**); restricția de unicitate pentru cuplul (**NrFact, CodPr**) înseamnă că se interzice ca, pe o factură, un produs să apară repetat.

Restricții referențiale

```
< specificație referențială > : : =
[ CONSTRAINT < nume restricție >
    REFERENCES < nume tabelă referențiată > [ ( < coloană referențiată > ) ]
```

Declararea restricțiilor referențiale se realizează utilizând clauza **FOREIGN KEY**. Astfel, pentru stabilirea legăturii **LINIIFACT-FACTURI** comanda de creare are forma:

```
CREATE TABLE LINIIFACT (
    NrFact NUMERIC(8) NOT NULL,
    Linie SMALLINT NOT NULL,
    CodPr NUMERIC(6) NOT NULL,
    Cantitate NUMERIC(10) NOT NULL,
    PretUnit NUMBER(12),
    PRIMARY KEY (NrFact, Linie),
    UNIQUE (NrFact, CodPr),
    FOREIGN KEY NrFact REFERENCES FACTURI(NrFact),
    FOREIGN KEY CodPr REFERENCES PRODUSE(CodPr) )
```

Este deosebit de importantă corecta definirea relației.

```
CREATE TABLE LINIIFACT (
    NrFact NUMERIC(8) NOT NULL REFERENCES FACTURI(NrFact),
    Linie SMALLINT NOT NULL,
    CodPr NUMERIC(6) NOT NULL REFERENCES PRODUSE(CodPr),
    Cantitate NUMERIC(10) NOT NULL,
    PretUnit NUMBER(12),
    PRIMARY KEY (NrFact, Linie),
    UNIQUE (NrFact, CodPr) )
```

Pentru controlul accesului la BD

GRANT	Acordarea unor drepturi pentru utilizator
REVOKE	Revocarea unor drepturi pentru utilizator

Pentru controlul tranzacțiilor

COMMIT	Mar체ază sfârșitul unei tranzacții
ROLLBACK	Abandonează tranzacția în curs

În prezentul capitol, precum și în următoarele, accentul va fi pus pe comenzi din primele două categorii. Comenzi SQL pentru controlul accesului și al tranzacțiilor nu fac obiectul prezentei lucrări.

3.3. Crearea, modificarea și stergerea tabelelor. Restricții

Crearea unei baze de date are o componentă tehnică pronunțată, mai ales în cazul serverelor de baze de date: Oracle, DB/2, SQL Server, Informix etc. Chiar dacă există instrumente de administrare care ușurează considerabil această activitate, crearea unei baze de date necesită cunoștințe de administrare strict legate de produsul software de gestiune a bazei.

În cele ce urmează, vom eluda elementele tehnice/fizice (spațiile-tabelă, segmentele de rollback, fisierile de date, procesele sistem etc.), lăsându-le în seama administratorilor BD; vom considera baza de date creată din punct de vedere fizic, astfel încât ne interesează numai crearea tabelelor, modificarea structurii lor și declararea restricțiilor.

Crearea tabelelor și declararea atributelor

Comanda SQL utilizată pentru crearea unei tabele este **CREATE TABLE**. Precizăm că este vorba de crearea structurii tabelei. Popularea cu înregistrări și actualizarea ulterioară se realizează prin alte comenzi: **INSERT, UPDATE, DELETE**.

Formatul general al comenzi creațoare de tabele este:

CREATE TABLE <nume tabelă> (<listă elemente din tabelă>)

unde:

< listă elemente din tabelă > : : =
<element al tabelei> | <element al tabelei>, <listă elemente din tabelă>

< element al tabelei > : : =
< definiție coloană > | < definiție restricție tabelă >

Începem cu definițiile coloanelor care cuprind o serie de opțiuni pentru declararea numelui, tipului, lungimii, precum și restricțiilor:

```
< definiție coloană > : : =
  < nume coloană > < tip de date >
    [< clauză valoarea implicită >]
    [< restricție a coloanei >]
```

Pentru crearea unei tabele și declararea atributelor acesteia, fără nici o referire la restricții, comanda **CREATE TABLE** are, în cazul tabelei JUDEȚE, următoarea formă:

```
CREATE TABLE JUDETE
  (Jud CHAR(2),
   Judet VARCHAR(25),
   Regiune VARCHAR (15))
```

Întrucât atrbutele acestei tabele sunt exclusiv de tip sir de caractere, s-au folosit opțiunile CHAR și VARCHAR. CHAR este preferat pentru atrbutul Jud, deoarece indicativul auto al județului este format exclusiv din două litere (cu excepția Bucureștiului). Întrucât denumirile județului și regiunii au lungime variabilă, pentru aceste atrbute a fost preferat tipul VARCHAR.

Pentru ilustrarea și altor două tipuri de date, prezentăm comanda de creare a tabelei FACTURI:

```
CREATE TABLE FACTURI (
  NrFact NUMERIC(8),
  DataFact DATE,
  CodCl DECIMAL(6),
  Obs VARCHAR(50))
```

NrFact și CodCl sunt de tip numeric, în timp ce DataFact de tip dată calendaristică. Se mai cuvine de adăugat că **unele atrbute pot fi inițializate cu o valoare implicită la adăugarea unei linii în tabela respectivă**. Clauza este **DEFAULT**:

```
< clauză valoarea implicită > : : =
  [CONSTRAINT < nume restricție > ]
  DEFAULT < opțiune valoare implicită >
< opțiune valoare implicită > : : = < literal > | < valoare sistem > | NULL
```

Ultima comandă se poate scrie astfel:

```
CREATE TABLE FACTURI (
  NrFact NUMERIC(8),
  DataFact DATE DEFAULT SYSDATE,
  CodCl DECIMAL(6) DEFAULT 1001,
  Obs VARCHAR(50))
```

Ca urmare a clauzei **DEFAULT**, în orice linie adăugată în tabela FACTURI, valoarea implicită (dacă nu este specificată în comanda **INSERT**) a **DataFact** va fi data sistemului (data curentă), iar **CodCl** se va inițializa cu valoarea 1001.

3.2. Tipuri de date și comenzi principale SQL

Înainte de prezentarea comenzilor SQL, ne ocupăm de categoriile principale de date ce pot fi stocate și accesate într-o bază de date relațională. Lista completă este cu mult mai lungă. În plus, fiecare SGBD granularizează tipurile principale, astfel încât reprezentarea datelor să se facă într-un format cât mai apropiat de substanța informațiilor pe care le reflectă. Cu rea știință, au fost evitate, discret, tipurile definite de utilizator (UDT), în general, tot ce ține de gestionarea obiectelor.

Specific nivelului de intrare (*entry*) al SQL-92 sunt următoarele **categoriile de date** (împreună cu lungimea/precizia lor):

- **SMALLINT**: întregi – scurte (4 poziții, reprezentate pe 16 biți),
- **INTEGER sau INT**: întregi (9 poziții, 32 biți),
- **NUMERIC(p,s)** sau **DECIMAL(p,s)** sau **DEC(p,s)**: reale, cu un total de *p* poziții, din care *s* la partea fracționară,
- **FLOAT**: reale, virgulă mobilă (20 poziții pentru mantisă),
- **REAL**: real, virgulă mobilă (cu precizie mai mică decât FLOAT, dar la nivelul de intrare este identică),
- **DOUBLE PRECISION**: reale, virgulă mobilă, dublă precizie (30 de poziții pentru mantisă),
- **CHAR(n)** sau **CHARACTER(n)**: sir de caractere de lungime *n* (max. 240),
- **VARCHAR(n)** sau **CHAR VARYING(n)** sau **CHARACTER VARYING(n)**: sir de caractere de lungime variabilă (max. 254),
- **DATE**: dată calendaristică,
- **TIME**: ora etc.,
- **TIMESTAMP**: an, lună, zi, ora, minutul, secunda, plus o fracțiune zecimală dintr-o secundă.

Joe Celko tratează excelent fiecare categorie de dată, așa cum este prezentă în SQL-92⁵⁰. În ceea ce privește datele temporale, el operează o delimitare între:

- **evenimente**, pentru care există tipurile DATE, TIME, TIMESTAMP,
- **intervale** – perioade de timp dintre două evenimente (momente) – tipul INTERVAL (YEAR, MONTH, DAY, HOUR, MINUTE, SECOND) și
- **perioade** – sunt intervale cu un moment fix de început, pentru care este necesară o combinație de două câmpuri, fie TIMESTAMP – TIMESTAMP, fie TIMESTAMP – INTERVAL.

Cum lucrarea de față este consacrată nucleului SQL în trei dintre cele mai folosite SGBD-uri relaționale, DB2 6.1 (IBM), Oracle 8 (Oracle) și Visual FoxPro 6 (Microsoft), în tabelul 3.1 este prezentată o comparație sumară între acestea, din punctul de vedere al principalelor tipuri de date pe care le pot gestiona.

50. [Celko99].

Tabelul 3.1. Principalele tipuri de date utilizate în DB2, Oracle și Visual FoxPro

DB2	Oracle	Visual FoxPro
CHARACTER (n)	CHAR (n)	CHARACTER (n)
CHAR (n)		
CHARACTER VARYING (n)	VARCHAR (n)	
CHAR VARYING (n)	VARCHAR2 (n)	
VARCHAR (n)		
LONG VARCHAR (n)	LONG	
		LOGICAL
DECIMAL (p,s)	NUMBER (p,s)	NUMBER (p,s)
REAL (p,s)		
INTEGER	INTEGER	INTEGER
SMALLINT		
DOUBLE	FLOAT	DOUBLE
FLOAT	NUMBER	
DATE	DATE	DATE
TIME	DATE	DATETIME
TIMESTAMP	DATE (fără microsecunde)	

SQL este mai mult decât un limbaj de interogare, după cum am văzut în primul paragraf al acestui capitol. Acest lucru este ilustrat și din evocarea principalelor familii de comenzi, fiecare familie vizând o sarcină specifică: interogare/manipulare, definirea datelor, controlul accesului, gestiunea tranzacțiilor etc. Principalele comenzi ale SQL care se regresesc, într-o formă sau alta, în multe dintre SGBDR-urile actuale sunt:

Comandă	Scop
<i>Pentru manipularea datelor</i>	
SELECT	Extragerea datelor din bază
INSERT	Adăugarea de noi linii într-o tabelă
DELETE	Stergerea de linii dintr-o tabelă
UPDATE	Modificarea valorilor unor atrbute
<i>Pentru definirea bazei de date</i>	
CREATE TABLE	Adăugarea unei noi tabele în baza de date
DROP TABLE	Stergerea unei tabele din bază
ALTER TABLE	Modificarea structurii unei tabele
CREATE VIEW	Crearea unei tabele virtuale
DROP VIEW	Stergerea unei tabele virtuale

Partea 1. Cadru general. Descrie fiecare parte a standardului și conține informații comune tuturor părților.

Partea 2. Fundament. Definește sintaxa și semantica SQL în ceea ce privește definirea și manipularea bazei de date, inclusiv opțiuni privind gestiunea tipurilor abstrakte de date.

Partea 3. SQL/CLI (Call Level Interface): un ansamblu de funcții și proceduri pentru conectarea bazelor de date prin SQL în medii multi-utilizator și multi-platformă, ansamblu dezvoltat de SQL Access Group.

Partea 4. SQL/PSM (Persistent Stored Modules): specificații procedurale necesare în funcții și proceduri utilizator. În cele din urmă, elementele privind procedurile și funcțiile, precum chestiuni legate de invocarea rutinelor, au fost transferate în partea a 2-a.

Partea 5. SQL/Bindings: include *Dynamic SQL* și *Embedded SQL* din standardul SQL-92 (SQL-2). Se referă la modul în care SQL este inclus în limbajele de programare ne-obiectuale. Este de așteptat ca în viitoarea versiune a standardului SQL această parte să fie mutată tot în partea a 2-a.

Partea 6. SQL/XA: specificații elaborate de X/Open și dedicate platformei X Windows. Această parte a fost abandonată.

Partea 7. SQL/Temporal: adaugă noi facilități privind gestiunea timpului și datei calendaristice în SQL.

La această structură inițială, între timp au mai fost adăugate și alte părți precum⁴⁷:

SQL/OLAP. Sunt descrise funcțiile și operațiunile utilizate pentru prelucrări analitice, fiind publicate ca amendament la standardul SQL-99⁴⁸.

Partea 8. SQL/Objects – Extended Objects. Vizează modul în care SGBD-urile relaționale gestionează tipurile abstrakte de date în aplicații. Nici această componentă nu mai există astăzi, fiind transferată în întregime în *Fundament*.

Partea 9. SQL/MED (Management of External Data). Definește câteva elemente adiționale *Fundamentului* pentru accesarea unor surse (fișiere) de date non-SQL.

Partea 10: SQL/OLB (Object Language Bindings). Este inclusă numai în standardul ANSI și definitivată din 1998; cuprinde specificații privitoare la includerea frazelor SELECT în limbajul Java, fiind corespondentă unui alt standard ANSI, SQLJ (Partea 0).

Partea 11: SQL/Schemata. Se referă la definirea și extragerea informațiilor privind schema bazei de date; este parte din *Fundament*, dar, în viitor, această parte va fi de sine stătătoare.

SQL Routines using the Java Programming Language. Definitivată în 1999 și inclusă numai în standardul ANSI, această parte se referă la modul în care secvențe de cod Java pot fi incluse în bazele de date SQL.

Partea 12: SQL/Replication. Lucrările au demarat în 2000, fiind vizată standardizarea comenzilor pentru replicarea bazelor de date.

Dintre ameliorările nelegate strict de lucrul cu obiecte, merită amintite cele referitoare la: declanșatoare (*triggers*) și alte tipuri de proceduri stocate, suport pentru seturile de caractere naționale, noi predicate în clauza WHERE (FOR ALL, FOR SOME, SIMILAR TO), tabele virtuale actualizabile, roluri pentru definirea profilelor de securitate etc.

În 1997, organizația X3 a ANSI a fost redenumită NCITS – National Committee for Information Technology Standards, iar comitetul care se ocupă de standardizarea SQL se numește acum ANSI NCITS-H2.

Se poate spune că, o dată cu standardul publicat în 1999, SQL ieșe din sfera relaționalului și a normalizării relațiilor, aşa cum au fost formulate de Codd. În orice caz, SQL continuă să evolueze, iar organismele care-i guvernează standardizarea sunt preocupate de întăriminarea cerințelor pieței bazelor de date.

Din perspectiva prezentei lucrări, obiectivul principal al SQL constă în a oferi utilizatorului mijloacele necesare formulării unei consultări numai prin descrierea rezultatului dorit, cu ajutorul unei aserții (expresie logică), fără a fi necesară și explicitarea modului efectiv în care se face căutarea în BD. Altfel spus, *utilizatorul califică (specifică) rezultatul, iar sistemul se ocupă de procedura de căutare*.

Deși este considerat, în primul rând, un limbaj de interogare, SQL este mult mai mult decât un instrument de consultare a bazelor de date, deoarece permite, în egală măsură:

- Definirea datelor
- Consultarea BD
- Manipularea datelor din bază
- Controlul accesului
- Partajarea bazei între mai mulți utilizatori ai acesteia
- Menținerea integrității BD.

După Groff și Weinberg, principalele atuuri ale SQL sunt⁴⁹:

- a) Independența de producător, nefiind o tehnologie proprietară
- b) Portabilitate între diferite sisteme de operare
- c) Este standardizat
- d) „Filosofia” sa se bazează pe modelul relațional de organizare a datelor
- e) Este un limbaj de nivel înalt, cu structură ce se apropie de limba engleză
- f) Permite formularea de răspunsuri la numeroase interogări simple, ad-hoc, neprevăzute inițial
- g) Constituie suportul programatic pentru accesul la BD
- h) Permite multiple imagini asupra datelor bazei
- i) Este un limbaj relațional complet
- j) Permite definirea dinamică a datelor, în sensul modificării structurii bazei chiar în timp ce o parte dintre utilizatori sunt conectați la BD
- k) Constituie un excelent suport pentru implementarea arhitecturilor client-server.

47. Prelucrare din [Hernandez&Viesca00], pp. 62-63.

48. Vezi și [Fotache00 – 4] și [Fotache00 – 5].

49. [Groff&Weinberg94], pp. 6-7.

restricțiilor referențiale a generat discuții aprinse și critici vehemente, astfel încât rapid a fost publicat un set de specificații numit *Integrity Enhancement Feature* (elemente de ameliorare a integrității), prin care se pot defini chei primare și chei străine ca elemente componente ale schemei bazei de date.

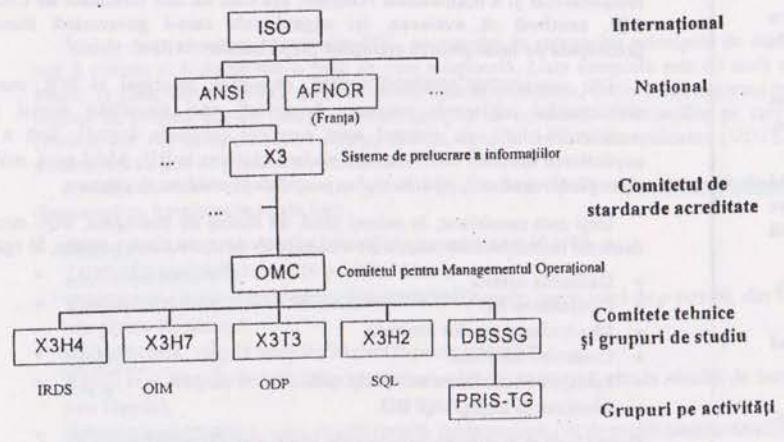


Figura 3.1. Structura organismelor de standardizare în domeniul bazelor de date

La trei ani de la publicarea SQL-86, prin revizuirea și extinderea sa, se „naște” SQL-89, care mai este denumit și SQL-1 – ANSI X3.135-1989, respectiv ISO 9075:1989. Deși recunoscut ca fundament al multor SGBDR-uri comerciale, și SQL-1 și-a atrăs numeroase critici. În plus, variantele comercializate de diferiți producători, deși esențialmente asemănătoare, erau (și sunt) incompatibile la nivel de detaliu. În afară de setul de specificații mai sus amintit, SQL-89 a inclus și specificații pentru apelul comenziilor și funcțiilor SQL din limbiile-gazdă, precum COBOL, Fortran și C.

Pentru a umple gurile SQL-1, ANSI și ISO au elaborat în 1992 versiunea SQL-2, ANSI X3.135-1992 (*Database Language SQL*), respectiv ISO/IEC 9075:1992, specificațiile fiind prezentate la un nivel mult mai detaliat (dacă SQL-1 se intindea pe numai 100 de pagini, SQL-92 a fost publicat în aproape 600). Dintre numeroasele facilități aduse de SQL-92, merită amintire cu deosebire: joncțiunea externă (OUTER JOIN), atribute zi-oră și de alte tipuri, raportare standardizată a erorilor, un set standard de tabele din catalog (dicționarul de date), modificarea schemei bazei (DROP, ALTER, GRANT, REVOKE), SQL dinamic, modificări și ștergeri referențiale în cascadă, amânarea verificării restricțiilor, niveluri de coherență a tranzacțiilor etc.

Standardul SQL-92 definește trei niveluri de conformitate:

- *Entry* – intrare, de bază (opțiunile din SQL-89 corectate);
- *Intermediate* – intermediar (ce include aproximativ jumătate dintre facilități);
- *Full* – deplin.

Fiecare firmă își declară nivelul de conformitate al SGBD-ului în raport cu SQL-92. Spre exemplu, nucleul SQL din Oracle 8 este conform cu nivelul de bază (*entry*), dar, după declarațiile producătorului, prezintă multe elemente suplimentare specifice celorlalte trei niveluri superioare.

Certificarea nivelului de conformitate cădea, până nu demult, în sarcina unui organism independent, *National Institute for Standards and Technology* (NIST), care utilizează *Federal Information Processing Standards* (FIPS), FIPS PUB 127-2. Din 1997 însă, FIPS și-a declinat implicarea în activitatea de certificare⁴⁵.

Pe lângă ANSI, ale cărui standarde au cea mai largă audiență, mai există și alte organisme de standardizare SQL. X/Open este un grup de firme europene care a adoptat SQL ca nucleu al unei întregi serii de standarde menite să asigure realizarea unui mediu general pentru aplicații portabile, grefat pe sistemul de operare UNIX. IBM a avut un aport incontestabil la apariția și maturizarea SQL, fiind un producător cu mare influență în lumea SGBD-urilor, iar produsul său, DB2, este unul din standardele *de facto* ale SQL.

În 1989, un grup de producători de instrumente dedicate bazelor de date au format *SQL Access Group*, în vederea realizării conexiunilor dintre SGBDR-urile fiecărui, pe baza unor specificații comune, din care un prim set a fost publicat în 1991 sub titulatura RDA (*Remote Database Access*). Specificațiile RDA n-au reușit să se impună pe piața SGBD-urilor. La insistențele firmei Microsoft, SQL Access Group și-a concentrat eforturile pentru elaborarea unei interfețe-standard pentru SQL. Pe baza unui set de propunerii inițiat de companie, în 1992, au rezultat specificațiile CLI (*Call Level Interface*). Având drept reper CLI, Microsoft elaborează și implementează în același an un set propriu, ODBC (*Open DataBase Connectivity*), care a devenit standard în materie de interfață SQL pentru accesarea diferitelor baze de date.

Cel mai recent standard este SQL-3, care a fost publicat în cea mai mare parte în iulie 1999. Complexitatea superioară față de precedator este sugerată și de numărul de pagini, aproape 2000 (față de 600 ale SQL-92). Scaderea finalizării sale a fost repetat amănată. Principalele orientări ale SQL-3 vizează transformarea acestuia într-un limbaj complet, în vederea definirii și gestionării obiectelor complexe și persistente. Aceasta include:

- generalizare și specializare,
- moșteniri multiple,
- polimorfism,
- encapsulare,
- tipuri de date definite de utilizator,
- suport pentru sisteme bazate pe gestiunea cunoștințelor,
- expresii privind interogări recursive și instrumente adecvate de administrare a datelor.

Defalcarea inițială (1993) a standardului SQL operată de comitetele ANSI și ISO a avut în vedere șapte componente⁴⁶:

45. Vezi și [Gorman97].

46. http://www.jcc.com/SQLPages/jccs_sql.htm.

- INTERSECT (R1, R2)
- INTERSECȚIE (R1, R2)

pentru diferență:

- R1 - R2
- REMOVE (R1, R2)
- MINUS (R1, R2)
- DIFERENȚĂ (R1, R2)

pentru produs cartesian:

- R1 x R2
- PRODUCT (R1, R2)
- TIMES (R1, R2)
- PROD (R1, R2)

pentru selecție:

- condiție (R)
- R [condiție]
- RESTRICT (R, condiție)
- SEL (R, condiție)

pentru proiecție:

- $\prod_{A, B, C, \dots, Z} (R)$
- R [A, B, ..., Z]
- PROJECT (R, A, B, ...)
- PROI (R, A, B, ..., Z)

pentru joncțiune:

- $R1 \bowtie_{condiție} R2$
- JOIN (R1, R2, condiție)

pentru diviziune:

- R1 + R2
- DIVISION (R1, R2)

Reprezentarea grafică a interogărilor

În multe lucrări de specialitate sunt utilizate o serie de simboluri grafice asociate operatorilor relaționali. Dintre acestea, cele mai importante sunt prezentate în figura 2.29.

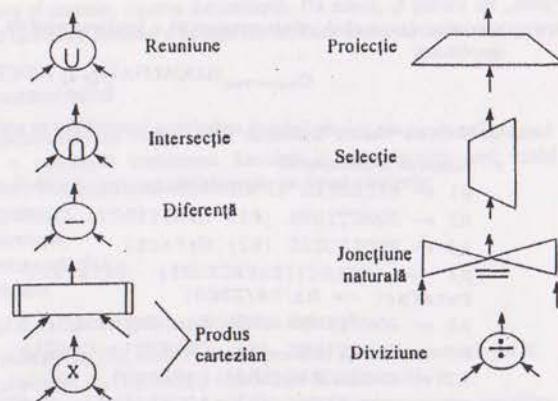


Figura 2.29. Simboluri pentru reprezentarea grafică a operatorilor algebrei relaționale

Aceste simboluri permit reprezentarea, în condiții de lizibilitate sensibil îmbunătățită, a logicii de derulare a unei interogařiri formulate prin operatorii relaționali. Pentru ilustrarea disponibilității simbolurilor în concordanță cu logica soluției, reluăm cele patru exemple folosite în acest paragraf.

Exemple

- Soluția de la exemplul 7 – figura 2.30:

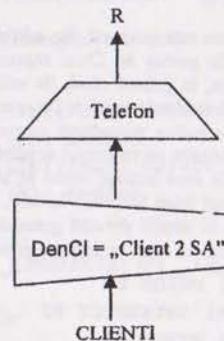


Figura 2.30. Reprezentare grafică – soluție ex. 7

- Soluția de la exemplul 9 – figura 2.31:

Este drept, atunci când trebuie reprezentată o joncțiune naturală, se poate recurge la o redare simplificată:

$$\sigma_{\text{Regiune} = \text{Banat}}(\text{LOCALITATI} \bowtie \text{JUDETE})$$

O altă observație ține de faptul că optimizarea interogărilor nu prezintă aproape deloc importanță.

- Soluția de la exemplul 16

```
R1 ← SELECȚIE (PRODUSE; DenPr = "Produs 1")
R2 ← JONCȚIUNE (R1, LINIIIFACT; CodPr)
R3 ← PROIECTIE (R2; NrFact)
R4 ← SELECȚIE (FACTURI; DataFact >= 03/08/2000 AND DataFact <= 03/08/2000)
R5 ← JONCȚIUNE (R3, R4; NrFact)
R6 ← JONCȚIUNE (R5, CLIENTI; CodCl)
R7 ← PROIECTIE (R6; CodPost)
R8 ← JONCȚIUNE (R7, LOCALITATI; CodPost)
R9 ← PROIECTIE (R8; Jud)
R10 ← JONCȚIUNE (R9, JUDETE; Jud)
R ← PROIECTIE (R10; Judet):
```

$$\prod_{\text{Judet}} (\sigma_{\text{DenPr} = \text{"Produs 1"} \wedge \text{DataFact} \geq 03/08/2000 \wedge \text{DataFact} \leq 03/08/2000} (\text{PRODUSE} \bowtie \text{LINIIIFACT} \bowtie \text{FACTURI} \bowtie \text{CLIENTI} \bowtie \text{LOCALITATI} \bowtie \text{JUDETE}))$$

Nu spun că acest stil de notare ar fi din cale-afară de dificil, dar eu, umul, consider că varianta „noastră” de redare este mult mai lejeră.

Gramatica BNF

Notația următoare este preluată din ediția a 4-a a lucrării lui C.J. Date³⁶ și, cu oarecare amendamente din partea lui Date, reprezintă gramatica *Backus Naur Form*. Avantajele acestei notații țin, în primul rând, de utilizarea unor cuvinte din limba engleză, în locul abstracterilor simboluri matematice. De asemenea, interogările se scriu liniar, fiind astfel mai lizibile. Pentru a indica precedența operațiilor se folosesc parantezele. Un avantaj major față de notația folosită pe parcursul acestui capitol ține de faptul că pune în evidență modul în care operațiile sunt incluse unele în altele, fiind mai aproape și de logica SQL (care folosește o singură frază SELECT în care, eventual, sunt incluse subconsultări).

Fără a intra în detaliî privind gramatica BNF tratată *in extenso* în lucrarea autorului american, amintim că reprezentarea operatorilor se face astfel:

- *reuniune*: R1 UNION R2
- *intersecție*: R1 INTERSECT R2
- *diferență*: R1 MINUS R2
- *produs cartezian*: R1 TIMES R2
- *selecție*, denumită și *restricție*: R WHERE P, unde P este un predicat aplicat asupra relației R
- *proiecție*: R [A, B, ... Z], unde A, B, ... Z sunt attribute ale R

36. [Date86].

- *theta-joncțiunea* se reprezintă diferit de *joncțiunea naturală*, ca un operator compus dintr-un produs cartezian urmat de o selecție. Astfel, pentru (theta)joncțiunea relațiilor R1 și R2 prin predicatul: R1.A > R2.C, se folosește notația:
R1 TIMES R2) WHERE R1.A > R2.C
- pentru *joncțiunea naturală* lucrurile sunt mai simple; astfel, *joncțiunea naturală* dintre R1 și R2 prin atributul C (atributul comun) se simbolizează pur și simplu: R1 JOIN R2.
- *diviziune*: R1 DIVIDED BY R2

Exemple

- Soluția de la exemplul 7:

(CLIENTI WHERE DenCl = "Client 2 SA") [Telefon]

- Soluția de la exemplul 9:

((LINIIIFACT WHERE NrFact = 1111) [CodPr]) INTERSECT
((LINIIIFACT WHERE NrFact = 1111) [CodPr])

- Soluția 1 de la exemplul 14:

((JUDETE JOIN LOCALITATI) WHERE Regiune = "Banat")

- Soluția de la exemplul 16

((((((PRODUSE WHERE DenPr = "Produs 1") JOIN LINIIIFACT) [NrFact]) JOIN FACTURI) WHERE DataFact >= 03/08/2000 AND DataFact <= 03/08/2000) JOIN CLIENTI) JOIN LOCALITATI) JOIN JUDETE) [Judet])

Probabil cel mai mare dezavantaj al eccestei notații ține de faptul că s-ar putea să greșim la numărul parantezelor...

Notații diverse

Literatura de specialitate a propus diverse notații, mai mult sau mai puțin apropiate de cele prezentate până acum. Dacă e să ne referim numai la cările publicate la noi³⁷ (bazate, ca și prezența, pe lucrări de circulație internațională), putem aminti:

pentru reuniune:

- R1 ∪ R2
- OR (R1, R2)
- APPEND (R1, R2)
- UNION (R1, R2)
- REUNIUNE (R1, R2)

pentru intersecție:

- R1 ∩ R2
- AND (R1, R2)

37. [Lungu §.a. 95], [Popescu96], [Dollinger98], [Florescu §.a. 99] (în treacăt fie spus, exemplul de la diviziunea relațională, p. 143, e cel puțin tulburător). [Grama&Filip00].

Figura 2.28 aduce, cât de căt, un plus de claritate.

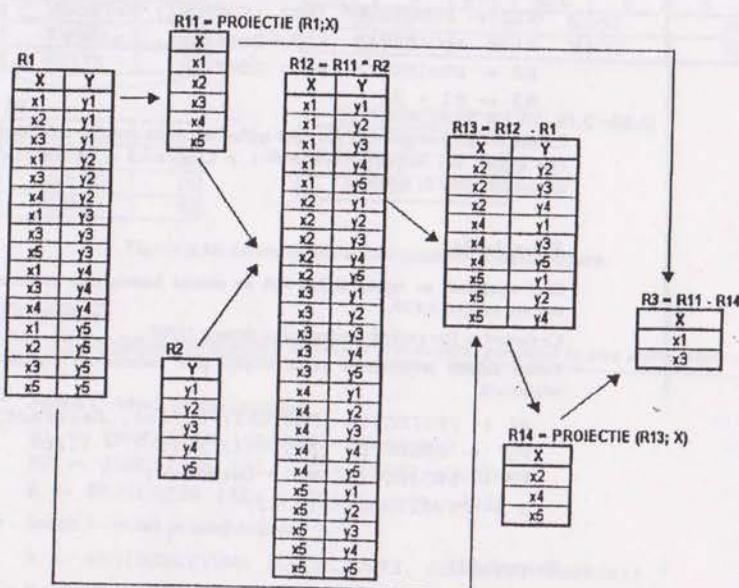


Figura 2.28. Simularea diviziunii prin alți operatori relaționali

Dintre operatorii prezentați pe parcursul ultimelor două paragrafe, selecția, proiecția, produsul cartezian, reunirea și diferența sunt operatori *fundamentali* sau *ireductibili*, în sensul că nici unul nu poate fi definit prin intermediul celorlalți.

- Joncțiunea, intersecția și diviziunea sunt operatori *derivați*;
- Joncțiunea se exprimă cu ajutorul produsului cartezian urmat de selecție;
- Intersecția se exprimă cu ajutorul diferenței, conform relației $R1 \cap R2 = R1 - (R1 - R2)$;
- Diviziunea poate fi exprimată, așa cum am văzut ceva mai sus, prin intermediul produsului cartezian, proiecției și diferenței.

2.4. Alte notații și reprezentarea grafică ale interogărilor

Încă de la începutul prezentării operatorilor algebrei relaționale, precizam că notația pe care o vom folosi este una dintre cele mai comode (nu și cea mai ușoară). Modelul relațional este, în ciuda materialului de față, unul matematizat. Algebra relațională este, așa

cum îi spune și numele, riguros formalizată. De aceea, și pentru un „look” mai științific, majoritatea lucrărilor folosesc o notație matematică sau... aproximativ matematică.

Notația matematică

Există o multitudine de lucrări care utilizează această notație. În cazul de față, sursa principală o constituie traducerea franceză a unei lucrări mai vechi de Korth și Silberschatz³⁵. Notația prezentată folosește, pe lângă clasicele:

- \cup – reunire,
- \cap – intersecție,
- \times – produs cartezian,
- $+$ – diviziune

și simboluri din alfabetul grecesc, după cum urmează:

- pentru selecție: $\sigma_P(R)$, unde P este un predicat aplicat asupra relației R,
- pentru proiecție: $\Pi_S(R)$, unde S este o listă de atribute din R,
- pentru joncțiune: $R1 \bowtie_{R1.A=R2.C} R2$ – indică (theta)joncțiunea relațiilor R1 și R2 prin predicatul: $R1.A > R2.C$.

Exemple

Soluția de la exemplul 7:

$R1 \leftarrow \text{SELECTIE} (\text{CLIENTI}; \text{DenCl} = "Client 2 SA")$
 $R2 \leftarrow \text{PROIECTIE} (R1; \text{Telefon})$

Potrivit notării matematice, rezolvarea arată:

$$\Pi_{\text{Telefon}}(\sigma_{\text{DenCl} = "Client 2 SA"}(\text{CLIENTI}))$$

După cum se observă, întreaga interogare se scrie într-o singură expresie.

- Soluția de la exemplul 9:
- $R1 \leftarrow \text{SELECTIE} (\text{LINIIFACT}; \text{NrFact} = 1111)$
 $R2 \leftarrow \text{PROIECTIE} (R1; \text{CodPr})$
 $R3 \leftarrow \text{SELECTIE} (\text{LINIIFACT}; \text{NrFact} = 1117)$
 $R4 \leftarrow \text{PROIECTIE} (R3; \text{CodPr})$
 $R5 \leftarrow R2 \cap R4$:

$$\Pi_{\text{CodPr}}(\sigma_{\text{NrFact} = 1111}(\text{LINIIFACT})) \cap \Pi_{\text{CodPr}}(\sigma_{\text{NrFact} = 1117}(\text{LINIIFACT}))$$

- Soluția I de la exemplul 14:

$R1 \leftarrow \text{JONCȚIUNE} (\text{LOCALITATI}, \text{JUDETE}; \text{Jud})$
 $R \leftarrow \text{SELECTIE} (R1; \text{Regiune} = "Banat")$

$$\sigma_{\text{Regiune} = "Banat"}(\text{LOCALITATI}) \bowtie_{\text{LOCALITATI.Jud} = \text{JUDETE.Jud}} \text{JUDETE}$$

35. [Korth&Silberschatz88].

Soluția poate fi redactată în următorii pași:

- construire relație-deimpărțit:
 $R11 \leftarrow \text{JONCȚIUNE} (\text{FACTURI}, \text{CLIENTI}; \text{CodCl})$
 $R1 \leftarrow \text{PROIECTIE} (R11; \text{DenCl}, \text{DataFact})$
- construire relație-numitor:
 $R2 \leftarrow \text{PROIECTIE} (\text{FACTURI}; \text{DataFact})$
- în fine, apoteoză:
 $R3 \leftarrow R1 + R2$

Schema și conținutul relațiilor implicate în această soluție sunt prezentate în figura 2.27.

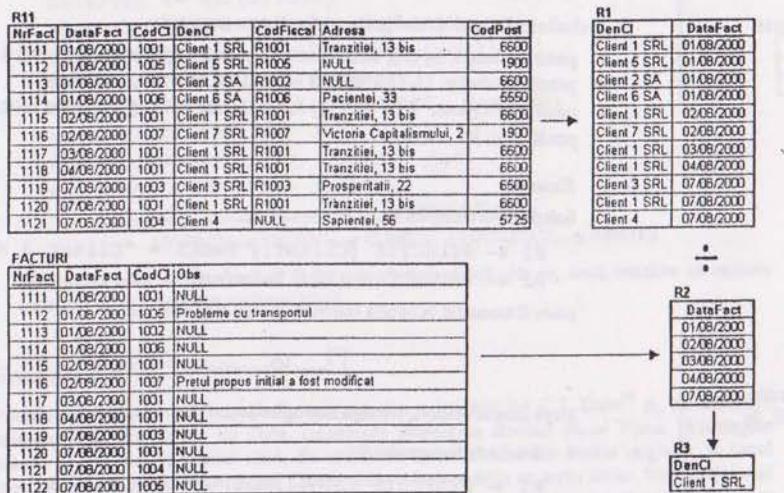


Figura 2.27. Diviziunea relațională – exemplul 22

Poate o să spună că n-a meritat efortul, dar acest gen de soluții se aplică la o gamă largă de probleme. Să discutăm câteva speje.

Exemplul 23

În ce zile s-au vândut și produsul cu denumirea „Produs 1”, și cel cu denumirea „Produs 2”?

Vă gândiți, probabil, că ați mai văzut undeva acest enunț. Este chiar textul problemei de la exemplul 17. La acel moment am formulat două soluții: una „clasică”, bazată pe intersecție, și una mai neconvențională ce utilizează juncțiunea. Adăugăm la acestea o alta, încadrabilă probabil tot în categoria „neconvenționale”.

• Soluție 3

$R11 \leftarrow \text{SELECTIE} (\text{PRODUSE}; \text{DenPr} = \text{"Produs 1"} \text{ OR } \text{DenPr} = \text{"Produs 2"})$

$R12 \leftarrow \text{JONCȚIUNE} (R11, \text{LINIIFACT}; \text{CodPr})$
 $R13 \leftarrow \text{JONCȚIUNE} (R12, \text{FACTURI}; \text{NrFact})$
 $R1 \leftarrow \text{PROIECTIE} (R13; \text{DataFact}, \text{CodPr})$
 $R2 \leftarrow \text{PROIECTIE} (R11; \text{CodPr})$
 $R3 \leftarrow R1 + R2$

Relația-divizor este alcătuită din două tupluri ce conțin codurile celor două produse. Deimpărțitul este alcătuit din atributele DataFact și CodProd și conține toate produsele vândute în fiecare din zilele de facturare.

Exemplul 24

Deși reprezintă un regres vizibil față de nivelul interogărilor precedente, revenim discret la enunțul exemplului 19.

Ce facturi au fost emise în aceeași zi cu factura 1120?

Pentru această problemă se poate incopi și o variantă de rezolvare bazată pe diviziunea relațională.

$R1 \leftarrow \text{PROIECTIE} (\text{FACTURI}; \text{NrFact}, \text{DataFact})$
 $R2' \leftarrow \text{SELECTIE} (\text{FACTURI}; \text{NrFact} = 1120)$
 $R2 \leftarrow \text{PROIECTIE} (R2'; \text{DataFact})$
 $R \leftarrow \text{DIVIZIUNE} (R1, R2)$

Exemplul 25

Cărora clienți le-au fost vândute toate produsele firmei?

$R11 \leftarrow \text{JONCȚIUNE} (\text{LINIIFACT}, \text{FACTURI}; \text{NrFact})$
 $R1 \leftarrow \text{PROIECTIE} (R11; \text{CodCl}, \text{CodPr})$
 $R2 \leftarrow \text{PROIECTIE} (\text{PRODUSE}; \text{CodPr})$
 $R3 \leftarrow \text{DIVIZIUNE} (R1, R2)$
 $R \leftarrow \text{JONCȚIUNE} (R3, \text{CLIENTI}; \text{CodCl})$

Diviziunea relațională nu este un operator fundamental; funcționalitatea sa poate fi realizată prin combinarea operatorilor: produs cartezian, diferență și proiecție.

Se reiau în discuție atât tabelele R1 și R2 din figura 2.26, cât și problema: care dintre valorile x_i apar în tupluri, în R1, cu toate valorile y_j din R2?

O soluție a problemei poate fi constituită din următorii pași:

- $R11 \leftarrow \text{PROIECTIE} (R1; X)$
- $R12 \leftarrow R11 \otimes R2$. Tabela R12 cuprinde toate tuplurile posibile (x_i, y_j)
- $R13 \leftarrow R12 - R1$. Interesează ce tupluri din R12 lipsesc în R1
- Valorile x_i din R13 nu apar în R1 în combinație cu toate valorile y_j din R2. Se elimină dublurile prin $R14 \leftarrow \text{PROIECTIE} (R13; X)$
- Valorile x_i din R14 sunt cele care nu prezintă tupluri obținute prin combinații cu toate valorile y_j . Deoarece interesează cele care prezintă combinațiile respective, rezultatul se obține prin diferență $R3 \leftarrow R11 - R14$.

R1

A	B	C
20 XYZ	30	
30 XXZ	20	
40 YYX	25	

R2

C	D	E
25 XYZ	30	
40 YYX	25	
30 XXZ	40	

R = JONCTIUNE (R1,R2; R1.C=R2.C)

A	B	R1.C	R2.C	D	E
20 XYZ	30		30 XXZ		40
40 YYX	25		25 XYZ		30

R = SEMIJONCTIUNE (R1,R2; R1.C=R2.C)

A	B	C
20 XYZ	30	
40 YYX	25	

Figura 2.25. Diferența dintre echijoncțiune și semijoncțiune

Exemplul 21

Care sunt localitățile (codul poștal, denumirea și indicativul județului) în care există măcar un client?

- Soluția 1 – bazată pe echijoncțiune


```
R1 ← PROIECȚIE (CLIENTI; CodPost)
R2 ← JONCTIUNE (R1, LOCALITATI; CodPost)
R ← PROIECȚIE (R2; CodPost, Loc, Jud)
```
- Soluția 2 – bazată pe semijoncțiune


```
R ← SEMIJONCTIUNE (LOCALITATI, CLIENTI; CodPost)
```

Ca și cum nu ar fi fost de ajuns, în „literatura” relațională mai există un tip de joncțiune – autojoncțiunea (închiderea tranzitivă), utilă în procesul de elaborare a bazelor de date relaționale, la normalizarea relațiilor³⁴, pe care o lăsăm însă în plată teoriei.

Diviziunea

Este cel mai complex și mai greu de explicat dintre operatorii prezenți în acest capitol. Codd l-a imaginat ca operator invers al produsului cartezian. Pentru a-l defini, se pornește de la două relații $R1(X, Y)$ și $R2(Y)$; prima are, care variază, două attribute sau grupe de attribute, notate X și Y , în timp ce a doua numai atributul sau grupul de attribute notat cu Y (definit pe același domeniu ca și în relația $R1$).

O primă restricție: relația $R2(Y)$, fiind numitorul diviziunii, nu trebuie să fie vidă.

Diviziunea relațională $R1 + R2$ are ca rezultat o relație definită ca ansamblul sub-tuplurilor $R1(X)$ pentru care produsul (lor) cartezian cu $R2(Y)$ este un subansamblu al $R1(X, Y)$. Rezultatul expresiei $R1 + R2$ reprezintă cădoul diviziunii, fiind o relație ce poate fi notată $R3(X)$. Într-o altă formulare, $x_i \in R3$ dacă și numai dacă $\forall y_i \in Y \in R2 \rightarrow \exists$

34. [Lungu s.a. 95], pp. 114-115.

Pentru simplificarea prezentării, în continuare am considerat X și Y două attribute, deși, după cum reiese din preambul, acestea pot fi grupe (ansambluri) de attribute. Să examinăm elementele din figura 2.26.

R1

X	Y
x1	y1
x2	y1
x3	y1
x1	y2
x3	y2
x4	y2
x1	y3
x3	y3
x5	y3
x1	y4
x3	y4
x4	y4
x1	y5
x2	y5
x3	y5
x5	y5

R2

Y
y1
y2
y3
y4
y5

R3

X
x1
x3

Figura 2.26. Diviziunea relațională

Determinarea relației $R3 \leftarrow R1 + R2$ este sinonimă cu rezolvarea problemei: care dintre x_1, x_2, x_3, x_4 și x_5 apar în $R1$, în tupluri împreună cu toate valorile lui Y din $R2$, respectiv y_1, y_2, y_3, y_4 și y_5 ?

Să parcurg pe rând valorile x_i ale atributului X din relația $R1$:

- x_1 apare cu y_1 (în tuplul 1), cu y_2 (în tuplul 4), cu y_3 (în tuplul 7), cu y_4 (în tuplul 10) și cu y_5 (în tuplul 13). Deci x_1 îndeplinește condiția și va fi inclus în relația $R3$;
- x_2 apare cu y_1 (în tuplul 2) dar nu apare cu y_2 – nu va face parte din $R3$;
- x_3 apare cu y_1 (în tuplul 3), cu y_2 (în tuplul 5), cu y_3 (în tuplul 8), cu y_4 (în tuplul 11) și cu y_5 (în tuplul 15) – îndeplinește condiția și va fi tuplu în $R3$;
- x_4 nu apare cu y_1 – nu va face parte din $R3$.
- x_5 nu apare cu y_1 – nu va face parte din $R3$.

În urma raționamentului de mai sus, tabela $R3$ va fi alcătuită din două tupluri. Deși pare un operator ceva mai metafizic, diviziunea relațională este deosebit de utilă pentru formularea consultărilor în care apare clauza „ \forall ” („oricare ar fi” sau „pentru toate”).

Exemplul 22

Care sunt clienții pentru care există cel puțin căte o factură emisă în fiecare zi?

Într-o altă formulare, ne interesează clienții care au cumpărat „căte ceva” în *toate zilele* în care s-au efectuat vânzări. Prin urmare, cătul va fi o tabelă cu singur atribut, *DenCl* (denumirea clientului), iar divizorul va fi o relație alcătuită numai din atributul *DataFact* (conține toate zilele în care s-au efectuat vânzări). Dacă am merge pe calapodul prezentat, am putea nota $R3(DenCl), R2(DataFact)$. Cunoscând structura cătului și a divizorului, putem determina structura tabelei-deîmpărțit: $R1(DenCl, DataFact)$. Relația $R1$ va conține denumirile clientilor și zilele în care există măcar o factură pentru clientul respectiv.

```
R1 ← SELECTIE (FACTURI; NrFact = 1120)
R2 ← PROIECTIE (R1; DataFact)
R3 ← JONCTIUNE (FACTURI, R2; DataFact)
R ← PROIECTIE (R3; NrFact)
```

Alte două tipuri de jonctiune: jonctiunea externă și semijonctiunea

Cele trei tipuri de jonctiune prezentate (theta, echi, naturală) prezintă două extensii, jonctiunea externă și semijonctiunea, dintre care prima are o importanță deosebită, mai ales prin transpunerea sa în SQL și redactarea unor interogări elegante pentru probleme ceva mai complexe.

Jonctiunea externă

Ideea de bază a jonctiunii externe este de a include în rezultat și tupluri din una dintre relații, sau din ambele relații, care prezintă valori ale atributului de legătură ce nu se regăsesc în celelalte relații.

Dacă precedentele tipuri de jonctiune sunt comutative, în cazul jonctiunii externe trebuie specificat din care relație se extrag liniile fără corespondent în celelalte relații. De aceea, există jonctiunea externă la stânga și jonctiunea externă la dreapta. La acestea se adaugă jonctiunea externă totală (denumită și plină sau deplină), care reprezintă reunirea celor două.

Apelând la aceleași tabele „abstrakte”, R1 și R2, diferența dintre jonctiunea internă (echi-jonctiunea) și cele trei tipuri de jonctiune externă apare cu mai multă claritate în figura 2.24.

Exemplul 20

Care sunt localitățile în care nu avem nici un client?

- Soluția 1 se bazează pe diferența dintre tabela tuturor localităților (din tabela LOCALITĂȚI) și tabela localităților în care există clienți (tabela CLIENTI).
- ```
R1 ← PROIECTIE (LOCALITATI; CodPost)
R2 ← PROIECTIE (CLIENTI; CodPost)
R3 ← R1 - R2
R ← JONCTIUNE (R3, LOCALITATI; CodPost)
```

| R - JONCTIUNE (R1,R2; R1.C=R2.C) |     |      |      |     |    |
|----------------------------------|-----|------|------|-----|----|
| A                                | B   | R1.C | R2.C | D   | E  |
| 20                               | XYZ | 30   | 30   | XXZ | 40 |
| 40                               | YYX | 25   | 25   | XYZ | 30 |

| R - JONCTIUNE EXTERNA LA STINGA (R1,R2; R1.C=R2.C) |     |      |      |      |      |
|----------------------------------------------------|-----|------|------|------|------|
| A                                                  | B   | R1.C | R2.C | D    | E    |
| 20                                                 | XYZ | 30   | 30   | XXZ  | 40   |
| 30                                                 | XXZ | 20   | NULL | NULL | NULL |
| 40                                                 | YYX | 25   | 25   | XYZ  | 30   |

| R - JONCTIUNE EXTERNA LA DREAPTA (R1,R2; R1.C=R2.C) |      |      |      |     |    |
|-----------------------------------------------------|------|------|------|-----|----|
| A                                                   | B    | R1.C | R2.C | D   | E  |
| 20                                                  | XYZ  | 30   | 30   | XXZ | 40 |
| 40                                                  | YYX  | 25   | 25   | XYZ | 30 |
| NULL                                                | NULL | NULL | 40   | YYX | 25 |

| R - JONCTIUNE EXTERNA TOTALA (R1,R2; R1.C=R2.C) |      |      |      |      |      |
|-------------------------------------------------|------|------|------|------|------|
| A                                               | B    | R1.C | R2.C | D    | E    |
| 20                                              | XYZ  | 30   | 30   | XXZ  | 40   |
| 30                                              | XXZ  | 20   | NULL | NULL | NULL |
| 40                                              | YYX  | 25   | 25   | XYZ  | 30   |
| NULL                                            | NULL | NULL | 40   | YYX  | 25   |

Figura 2.24. Diferența dintre echi-jonctiune și jonctiunile externe

- Soluția 2 utilizează proaspăta jonctiune externă.

```
R1 ← JONCTIUNE EXTERNA LA STINGA (LOCALITATI, CLIENTI;
LOCALITATI.CodPost = CLIENTI.CodPost)
R ← SELECTIE (R1; CodCl IS NULL)
```

Pentru a identifica localitățile fără clienti, s-au extras, din jonctiunea externă la stânga a relațiilor LOCALITĂȚI și CLIENTI, numai liniile în care unul din atributele preluate din CLIENTI (noi ne-am opriți asupra primului, CodCl) are valoarea NULL.

### Semijonctiunea

Semijonctiunea este unul dintre cele mai „marginalizate” tipuri de jonctiune. Implementarea sa în SGBD-urile comerciale este extrem de rară. A fost introdusă din dorința de a optimiza procesul de interogare (consultare). Calculul semijonctiunii a două tabele presupune selecțarea numai a liniilor din prima tabelă care apar în jonctiune cu liniile din a două tabele – vezi figura 2.25.

**Exemplul 17**

*În ce zile s-au vândut și produsul cu denumirea „Produs 1”, și cel cu denumirea „Produs 2”?*

Rezultatul conține atributul DataFact. Predicatul de selecție se aplică tabeliei PRODUSE.

- Soluție 1

```
R1 ← SELECȚIE (PRODUSE; DenPr = "Produs 1")
R2 ← JONCȚIUNE (R1, LINIIFACT; CodPr)
R3 ← JONCȚIUNE (R2, FACTURI; NrFact)
R4 ← PROIECTIE (R3; DataFact)
R5 ← SELECȚIE (PRODUSE; DenPr = "Produs 2")
R6 ← JONCȚIUNE (R5, LINIIFACT; CodPr)
R7 ← JONCȚIUNE (R6, FACTURI; NrFact)
R8 ← PROIECTIE (R7; DataFact)
R ← R4 ∩ R8
```

- Soluție 2

```
R1 ← SELECȚIE (PRODUSE; DenPr = "Produs 1")
R2 ← JONCȚIUNE (R1, LINIIFACT; CodPr)
R3 ← JONCȚIUNE (R2, FACTURI; NrFact)
R4 ← SELECȚIE (PRODUSE; DenPr = "Produs 2")
R5 ← JONCȚIUNE (R5, LINIIFACT; CodPr)
R6 ← JONCȚIUNE (R6, FACTURI; NrFact)
R7 ← JONCȚIUNE (R3, R6; DataFact)
R ← PROIECTIE (R7; DataFact)
```

Prima variantă este una „cuminte”. Se intersectează relația zilelor în care s-a vândut primul produs (R4) cu relația zilelor în care s-a facturat produsul 2 (R8).

A doua e ceva mai insolită. Pur și simplu, în loc de intersecție folosim joncțiunea. Cum, spre deosebire de intersecție, relațiile jonctionate nu trebuie să fie unicompatibile, putem să operăm direct joncțiunea între R3 și R6.

Reținem ideea: putem să simulăm intersecția a două relații prin joncțiune. Pentru a fi mai convingător, rogu-vă să examinați figura 2.23, în care, pornind de la relațiile R1 și R2, se obține relația-intersecție R direct prin operatorul intersecție (stânga figurii) și mai pe ocolite, folosind joncțiunea (dreapta).

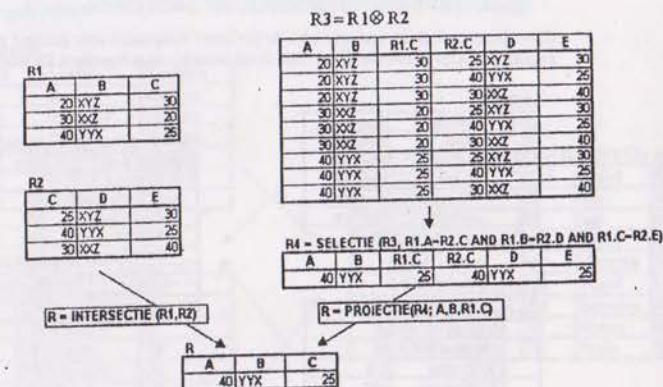


Figura 2.23. Intersecția prin joncțiune

**Exemplul 18**

*Ce clienți au cumpărat „Produs 2” și „Produs 3”, dar nu au cumpărat „Produs 5”?*

Bună întrebare! Lucrurile nu sunt însă atât de complicate precum par, deoarece folosim elemente din exemplul anterior (intersecția) plus operatorul... diferență.

```
R1 ← SELECȚIE (PRODUSE; DenPr = "Produs 2")
R2 ← JONCȚIUNE (R1, LINIIFACT; CodPr)
R3 ← JONCȚIUNE (R2, FACTURI; NrFact)
R4 ← JONCȚIUNE (R3, CLIENTI; CodCl)
R5 ← PROIECTIE (R4; DenCl)
R6 ← SELECȚIE (PRODUSE; DenPr = "Produs 3")
R7 ← JONCȚIUNE (R6, LINIIFACT; CodPr)
R8 ← JONCȚIUNE (R7, FACTURI; NrFact)
R9 ← JONCȚIUNE (R8, CLIENTI; CodCl)
R10 ← PROIECTIE (R9; DenCl)
R11 ← SELECȚIE (PRODUSE; DenPr = "Produs 5")
R12 ← JONCȚIUNE (R11, LINIIFACT; CodPr)
R13 ← JONCȚIUNE (R12, FACTURI; NrFact)
R14 ← JONCȚIUNE (R13, CLIENTI; CodCl)
R15 ← PROIECTIE (R14; DenCl)
R ← R5 ∩ R10 - R15
```

**Exemplul 19**

*Ce facturi au fost emise în aceeași zi cu factura 1120?*

Spre deosebire de interogările de până acum, condiția de selecție este una indirectă. În prealabil, trebuie determinată ziua în care a fost întocmită factura cu numărul 1120. Apoi trebuie extrase, din relația FACTURI, liniile pentru care DataFact are valoarea zilei facturii-reper.

Care dintre cele două variante este de preferat? Răspunsul este facilitat de reprezentarea grafică a pașilor parcursi în fiecare dintre cele două soluții – vezi figurile 2.21 și 2.22.

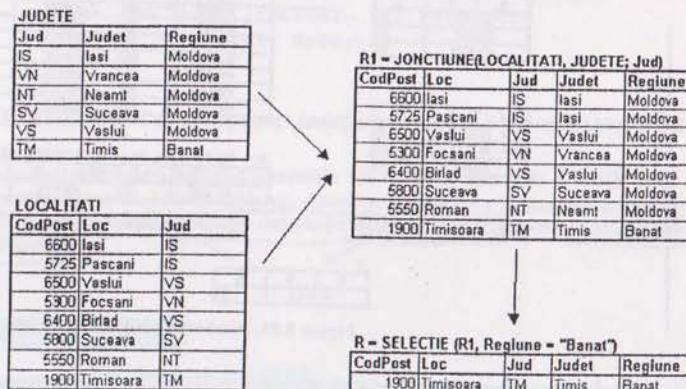


Figura 2.21. Plan de execuție – exemplul 14, soluția 1

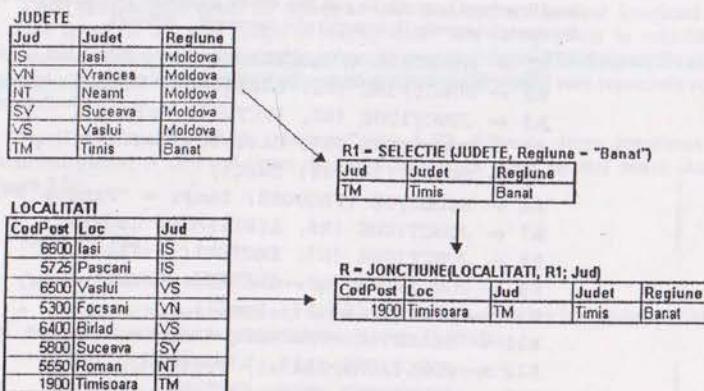


Figura 2.22. Plan de execuție – exemplul 14, soluția 2

A doua variantă pare mai bună decât prima, deoarece jonctiunea operează asupra a două relații mai reduse ca dimensiuni. Diferența este că atât mai vizibilă atunci când relația JUDEȚE conține toate județele țării, iar LOCALITĂȚI are câteva sute de înregistrări. Iar dacă ne gândim că înaintea oricărui jonctiuni se calculează produsul cartezian, apare drept firească ideea amânării jonctiunii, astfel încât aceasta să opereze asupra unor tabele cu un număr căt mai mic de linii și coloane.

Coborând cu picioarele pe pământ, trebuie spus că discuția noastră are un caracter de principiu, deoarece algebra relațională este totuși un limbaj... curat teoretic.

### Exemplul 15

*In ce zile s-a vândut produsul cu denumirea „Produs 1”?*

Elementul de noutate îl reprezintă interesul pentru o informație ce provine dintr-o relație (atributul DataFact din FACTURI) pe baza unei condiții aplicate altor relații (atributul DenPr din PRODUSE), iar cele două relații, FACTURI și PRODUSE, nu sunt în raportul părinte-copil. În aceste cazuri, este necesară atragerea altor relații, până se completează „lanțul”. Interrogarea ce rezolvă problema ridicată în acest exemplu necesită și tabela LINIIFACT.

- Soluție 1 – neoptimizată

```
R1 ← JONCTIUNE (PRODUSE, LINIIFACT; CodPr)
R2 ← JONCTIUNE (R1, FACTURI; NrFact)
R3 ← SELECTIE (R2; DenPr = "Produs 1")
R ← PROIECȚIE (R3; DataFact)
```

- Soluție 2 – optimizată la sângie

```
R1 ← SELECTIE (PRODUSE; DenPr = "Produs 1")
R2 ← PROIECȚIE (R1; CodPr)
R3 ← JONCTIUNE (R2, LINIIFACT; CodPr)
R4 ← PROIECȚIE (R3; NrFact)
R5 ← JONCTIUNE (R4, FACTURI; NrFact)
R ← PROIECȚIE (R4; DataFact)
```

În cea de-a două soluție, fideli principiului „jonctiunii celei mai economicoase”, am eliminat numai tuplurile, dar și atributele de prisos înaintea calculării relației intermediare.

### Exemplul 16

*In ce județe s-a vândut produsul cu denumirea „Produs 1” în perioada 3-5 august 2000?*

Relația-resultat trebuie să conțină valori ale atributului Judet din tabela JUDEȚE. Predicatul de selecție se aplică însă în alte două tabele: PRODUSE, în care DenPr = „Produs 1”, și FACTURI, ale cărui tupluri trebuie să verifice condiția: DataFact >= 03/08/2000 AND DataFact <= 03/08/2000.

```
R1 ← SELECTIE (PRODUSE; DenPr = "Produs 1")
R2 ← JONCTIUNE (R1, LINIIFACT; CodPr)
R3 ← PROIECȚIE (R2; NrFact)
R4 ← SELECTIE (FACTURI; DataFact >= 03/08/2000 AND DataFact <= 03/08/2000)
R5 ← JONCTIUNE (R3, R4; NrFact)
R6 ← JONCTIUNE (R5, CLIENTI; CodCl)
R7 ← PROIECȚIE (R6; CodPost)
R8 ← JONCTIUNE (R7, LOCALITATI; CodPost)
R9 ← PROIECȚIE (R8; Jud)
R10 ← JONCTIUNE (R9, JUDETE; Jud)
R ← PROIECȚIE (R10; Judet)
```

**Exemplul 12 – Joncțiune naturală**

Joncțiunea naturală presupune nu numai ca operatorul de comparație să fie semnul de egalitate, ci și denumirea identică a atributelor de legătură dintre cele două tabele.

$R \leftarrow \text{JONCȚIUNE} (R1, R2; R1.C = R2.C)$

Datorită faptului că ambele atrbute au același număr, se poate considera că tabela-rezultat păstrează numai unul dintre cele două atrbute, ca în figura 2.19, și se poate recurge la o notație simplificată:

$R \leftarrow \text{JONCȚIUNE} (R1, R2; C)$

| R1     |   |    | R $\leftarrow R1 \otimes R2$ |   |      |        |   |    |
|--------|---|----|------------------------------|---|------|--------|---|----|
| A      | B | C  | A                            | B | R1.C | R2.C   | D | E  |
| 20 XYZ |   | 30 | 20 XYZ                       |   | 30   | 25 XYZ |   | 30 |
| 30 XXZ |   | 20 | 20 XYZ                       |   | 30   | 40 YYX |   | 25 |
| 40 YYX |   | 25 | 20 XYZ                       |   | 30   | 30 XXZ |   | 40 |

| R2     |   |    | R - SELECTIE (R', R1.C = R2.C) |   |        |        |    |
|--------|---|----|--------------------------------|---|--------|--------|----|
| C      | D | E  | A                              | B | C      | D      | E  |
| 25 XYZ |   | 30 | 20 XYZ                         |   | 30 XXZ | 40     |    |
| 40 YYX |   | 25 | 40 YYX                         |   | 25     | 25 XYZ | 30 |
| 30 XXZ |   | 40 | 40 YYX                         |   | 25     | 40 YYX | 25 |

Figura 2.19. Joncțiune naturală – exemplul 12

De ce se insistă atât de mult pe importanța joncțiunii? În primul rând pentru că permite recompozerea relației universale inițiale. Modelul relațional se bazează pe spargerea bazei în relații, astfel încât nivelul redundanței datelor și problemele la actualizarea tabelelor să fie redus la minim. Cele mai multe interogări însă, operează cu date și predicate aplicate simultan atrbutele din două sau mai multe tabele. Cum selecția este un operator *unar* (poate fi aplicată unei singure relații), este necesară fusionarea prealabilă a celor două, trei... relații și obținerea unei relații-agregat, la care se aplică predicatul suplimentar de selecție.

Fuzionarea este posibilă prin joncțiune. Prin joncționarea tuturor relațiilor dintr-o bază de date se obține relația *universală* (cea inițială, atotcuprinzătoare). Nu ne permitem să reconstituim structura și conținutul relației universale ale bazei de date VÂNZĂRI, însă, prin exemplu, vom încerca să demonstrăm utilitatea joncțiunii.

**Exemplul 13**

Să se obțină, pentru fiecare localitate: codul poștal, denumirea, indicativul județului, denumirea județului și regiunea din care face parte.

Practic, tabelei LOCALITĂȚII li trebuie „alipite” la dreapta informațiile din tabela JUDEȚE. Problema este dificil de formulat, însă rezolvarea sa este căt se poate de simplă: joncționarea celor două relații – vezi figura 2.20.

$R \leftarrow \text{JONCȚIUNE} (\text{LOCALITATI}, \text{JUDETE}; \text{Jud})$

## JUDETE

| Jud | Judet   | Regiune |
|-----|---------|---------|
| IS  | Iași    | Moldova |
| VN  | Vrancea | Moldova |
| NT  | Neamț   | Moldova |
| SV  | Suceava | Moldova |
| VS  | Vaslui  | Moldova |
| TM  | Timiș   | Banat   |

| R - JONCȚIUNE(LOCALITATI, JUDETE; Jud) |           |     |         |         |
|----------------------------------------|-----------|-----|---------|---------|
| CodPost                                | Loc       | Jud | Judet   | Regiune |
| 6600                                   | Iasi      | IS  | Iasi    | Moldova |
| 5725                                   | Pascani   | IS  | Iasi    | Moldova |
| 6500                                   | Vaslui    | VS  | Vaslui  | Moldova |
| 5300                                   | Focșani   | VN  | Vrancea | Moldova |
| 6400                                   | Birlad    | VS  | Vaslui  | Moldova |
| 5800                                   | Suceava   | SV  | Suceava | Moldova |
| 5550                                   | Roman     | NT  | Neamț   | Moldova |
| 1900                                   | Timișoara | TM  | Timiș   | Banat   |

## LOCALITATI

| CodPost | Loc       | Jud |
|---------|-----------|-----|
| 6600    | Iasi      | IS  |
| 5725    | Pascani   | IS  |
| 6500    | Vaslui    | VS  |
| 5300    | Focșani   | VN  |
| 6400    | Birlad    | VS  |
| 5800    | Suceava   | SV  |
| 5550    | Roman     | NT  |
| 1900    | Timisoara | TM  |

Figura 2.20. Joncțiunea tabelelor LOCALITĂȚI și JUDEȚE – exemplul 13

Una din întrebările „clasice” pentru verificarea modului în care a fost sau nu înțeleasă joncțiunea este: *Câte linii are tabela rezultat al joncțiunii?* În cazul nostru (de obicei în BDR) răspunsul este: *câte linii are tabela copil*. Răspunsul este corect numai atunci când se respectă integritatea referențială, altfel spus, numai atunci când toate valorile cheii strâni se regăsesc în tabela-părinte.

Ca de obicei, lucrurile sunt mai complicate în realitate, deoarece joncțiunea nu se instituie musai între o tabelă-părinte și una copil. Chiar primele „exemple, cele „teoretice” – figurile 2.17, 2.18, 2.19 – folosesc două relații, R1 și R2, despre care nu se face nici o afirmație privind filiația.

**Exemplul 14**

Care sunt localitățile din Banat?

- Soluția 1

După calapodul exemplului anterior:

$R1 \leftarrow \text{JONCȚIUNE} (\text{LOCALITATI}, \text{JUDETE}; \text{Jud})$   
 $R \leftarrow \text{SELECTIE} (R1; \text{Regiune} = "Banat")$

- Soluția 2

Mai întâi se aplică selecția asupra tabelci JUDETE, iar tabela intermedieră se joncționează cu LOCALITATI:

$R1 \leftarrow \text{SELECTIE} (\text{JUDETE}; \text{Regiune} = "Banat")$   
 $R \leftarrow \text{JONCȚIUNE} (\text{LOCALITATI}, R1; \text{Jud})$

$R3 \leftarrow \text{SELECTIE} (\text{LINIIFACT}; \text{NrFact} = 1117)$   
 $R4 \leftarrow \text{PROIECTIE} (R3; \text{CodPr})$   
 $R5 \leftarrow R2 \cap R4$

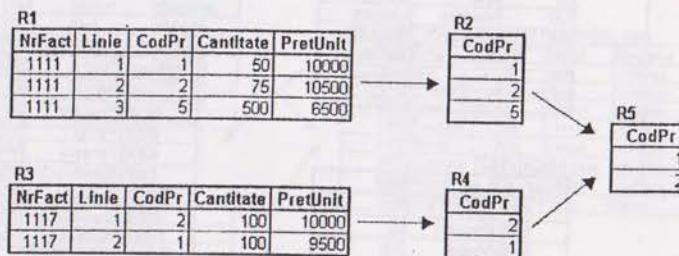


Figura 2.16. Exemplul 9 – soluție

### Joncțiunea

În paragraful anterior am văzut că produsul cartezian permite fuzionarea a două tabele într-o tabelă mamut ce conține toate atributele și linile obținute prin combinarea fiecărui tuplu dintr-o relație cu fiecare tuplu din celalăt. Tot atunci ne exprimam regretul sincer că operatorul produs cartezian nu poate fi folosit, de unul singur, în interogări, dar că acest „neajuns” va fi compensat din plin de operatorul derivat *joncțiune*.

Dacă produsul cartezian este o fuziune necondiționată a două tabele, *joncțiunea* reprezintă fuziunea a două relații care au o proprietate comună. Fie două relații, note:  $R1(A_1, A_2, \dots, A_n)$  și  $R2(B_1, B_2, \dots, B_p)$ . Fie  $A_i$  și  $B_j$  două atrbute definite pe același domeniu și  $\theta$  ansamblul operatorilor de comparație:  $\{=, >, \geq, <, \leq, \neq\}$ , ce pot fi aplicati celor două atrbute  $A_i$  și  $B_j$ .

*Joncțiunea relației  $R1$ , prin  $A_i$ , cu relația  $R2$ , prin  $B_j$ , notată*

$R1(A_i \theta B_j) R2$  sau  $R1 \bowtie_{\theta} R2$

este relația ale cărei tupluri sunt obținute prin concatenarea fiecărui tuplu al relației  $R1$  cu tuplurile relației  $R2$ , pentru care este verificată condiția  $\theta$  instituită între  $A_i$  și  $B_j$ .

$R1(A_i \theta B_j) R2 = \{ t | t \in R1 \otimes R2 \text{ și } t(A_i) \theta t(B_j) \}$

*Joncțiunea este echivalentă unui produs cartezian urmat de o selecție. Joncțiunea definită mai sus este cunoscută în lucrările de specialitate ca *theta-joncțiune*. În lucrul cu BDR se utilizează cu precădere *echi-joncțiunea*, ce reprezintă un caz particular al theta-joncțiunii, atunci când  $\theta$  este operatorul de egalitate ( $=$ ). Formal, echi-joncțiunea se definește astfel:*

$R1(A_i = B_j) R2 = \{ t | t \in R1 \otimes R2 \text{ și } t(A_i) = t(B_j) \}$

| R1 |     |    | R2 |     |      | R1 ⊗ R2 |     |    |
|----|-----|----|----|-----|------|---------|-----|----|
| A  | B   | C  | A  | B   | R1.C | R2.C    | D   | E  |
| 20 | XYZ | 30 | 20 | XYZ | 30   | 25      | XYZ | 30 |
| 30 | XXZ | 20 | 20 | XYZ | 30   | 40      | YYX | 25 |
| 40 | YYX | 25 | 20 | XYZ | 30   | 30      | XXZ | 40 |
|    |     |    | 30 | XXZ | 20   | 25      | XYZ | 30 |
|    |     |    | 30 | XXZ | 20   | 40      | YYX | 25 |
|    |     |    | 40 | YYX | 25   | 25      | XYZ | 30 |
|    |     |    | 40 | YYX | 25   | 40      | YYX | 25 |
|    |     |    | 40 | YYX | 25   | 30      | XXZ | 40 |

| R = SELECTIE (R', A >= E) |     |      |      |     |    |
|---------------------------|-----|------|------|-----|----|
| A                         | B   | R1.C | R2.C | D   |    |
| 30                        | XXZ | 20   | 25   | XYZ | 30 |
| 30                        | XXZ | 20   | 40   | YYX | 25 |
| 40                        | YYX | 25   | 25   | XYZ | 30 |
| 40                        | YYX | 25   | 40   | YYX | 25 |
| 40                        | YYX | 25   | 30   | XXZ | 40 |

Figura 2.17. Mecanismul de (theta)-joncțiune – exemplul 10

Apelăm și la o altă notație, mai ușor de reprezentat și suficient de inteligibilă.  
 $R \leftarrow \text{ECHI-JONCȚIUNE} (R1, R2; A_i = B_j)$

### Exemplul 10 – Theta-Joncțiune

Începem exemplificările cu aceleași două tabele folosite în precedentul paragraf,  $R1$  și  $R2$ . Rezultatul joncțiunii (theta-joncțiunii) exprimată prin expresia:

$R \leftarrow \text{JONCȚIUNE} (R1, R2; R1.A >= R2.E)$

va fi obținut în doi pași, după cum este descris în figura 2.17.

### Exemplul 11 – Echi-joncțiune

Operatorul de comparație dintre cele două atrbute este, obligatoriu, semnul de egalitate.

$R \leftarrow \text{JONCȚIUNE} (R1, R2; R1.A = R2.E)$

| R1 |     |    | R2 |     |    | R1 ⊗ R2 |     |      |      |     |    |
|----|-----|----|----|-----|----|---------|-----|------|------|-----|----|
| A  | B   | C  | C  | D   | E  | A       | B   | R1.C | R2.C | D   | E  |
| 20 | XYZ | 30 | 20 | XYZ | 30 | 25      | XYZ | 30   | 25   | XYZ | 30 |
| 30 | XXZ | 20 | 20 | XYZ | 30 | 40      | YYX | 25   | 40   | YYX | 25 |
| 40 | YYX | 25 | 20 | XYZ | 30 | 30      | XXZ | 40   | 30   | XXZ | 40 |
|    |     |    | 30 | XXZ | 20 | 25      | XYZ | 30   | 25   | XYZ | 30 |
|    |     |    | 30 | XXZ | 20 | 40      | YYX | 25   | 40   | YYX | 25 |
|    |     |    | 40 | YYX | 25 | 25      | XXZ | 40   | 30   | XXZ | 40 |
|    |     |    | 40 | YYX | 25 | 40      | YYX | 25   | 40   | YYX | 25 |
|    |     |    | 40 | YYX | 25 | 30      | XXZ | 40   | 30   | XXZ | 40 |

| R = SELECTIE (R', A = E) |     |      |      |     |    |
|--------------------------|-----|------|------|-----|----|
| A                        | B   | R1.C | R2.C | D   |    |
| 30                       | XXZ | 20   | 25   | XYZ | 30 |
| 40                       | YYX | 25   | 30   | XXZ | 40 |

Figura 2.18. Echi-joncțiune – exemplul 11

**Exemplul 6**

Care sunt: codul, denumirea și numărul de telefon ale fiecărui client?

Tabela care interesează este CLIENTI, din care se decupează trei coloane: CodCl, DenCl și Telefon (figura 2.12).

$R \leftarrow \text{PROIECȚIE} (\text{CLIENTI}; \text{CodCl}, \text{DenCl}, \text{Telefon})$

| R     |              |            |
|-------|--------------|------------|
| CodCl | DenCl        | Telefon    |
| 1001  | Client 1 SRL | NULL       |
| 1002  | Client 2 SA  | 032-212121 |
| 1003  | Client 3 SRL | 035-222222 |
| 1004  | Client 4     | NULL       |
| 1005  | Client 5 SRL | 056-111111 |
| 1006  | Client 6 SA  | NULL       |
| 1007  | Client 7 SRL | 056-121212 |

Figura 2.12. Rezultat-proiecție – exemplul 6

**Înlăturarea consultărilor**

După cum aminteam în paragraful introductiv al capitolului, rezultatul unei consultări este o relație (tabelă) nouă. Pe baza acestui fapt, se pot înlături două sau mai multe operațiuni, redactându-se astfel interogări complexe.

**Exemplul 7**

Care este numărul de telefon al clientului Client 2 SA?

Soluția este una foarte simplă. Cu ajutorul selecției se decupează din relația CLIENTI numai linia corespunzătoare clientului „incriminat”. Se obține o relație nou-nouă denumită (de noi) R1. Asupra lui R1 se aplică o proiecție, deoarece interesează numai numărul de telefon; astfel, R2 conține răspunsul la problema luată în discuție (vezi figura 2.13).

$R1 \leftarrow \text{SELECTIE} (\text{CLIENTI}; \text{DenCl} = \text{"Client 2 SA"})$   
 $R2 \leftarrow \text{PROIECȚIE} (R1; \text{Telefon})$

| R1    |             |           |        |         |            |
|-------|-------------|-----------|--------|---------|------------|
| CodCl | DenCl       | CodFiscal | Adresa | CodPost | Telefon    |
| 1002  | Client 2 SA | R1002     | NULL   | 6600    | 032-212121 |

| R2      |            |
|---------|------------|
| Telefon | 032-212121 |

Figura 2.13. Înlăturarea unei selecții cu o proiecție – exemplul 7

**Exemplul 8**

Care sunt denumirile și codurile poștale ale localităților (prezente în bază) din județele Iași (IS) și Vrancea (VN)?

Tabela „interrogată” este LOCALITATI. Pentru a răspunde la întrebarea pe care tot noi am formulat-o, putem alege între următoarele două soluții:

## • Soluția 1 – figura 2.14

$R1 \leftarrow \text{SELECȚIE} (\text{LOCALITATI}; \text{Jud} = \text{"IS"} \text{ OR } \text{Jud} = \text{"VN"})$   
 $R2 \leftarrow \text{PROIECȚIE} (R1; \text{Loc}, \text{CodPost})$

| R1      |         |     | R2      |         |
|---------|---------|-----|---------|---------|
| CodPost | Loc     | Jud | Loc     | CodPost |
| 6600    | Iasi    | IS  | Iasi    | 6600    |
| 5725    | Pascani | IS  | Pascani | 5725    |
| 5300    | Focșani | VN  | Focșani | 5300    |

Figura 2.14. Exemplul 8 – soluția 1

## • Soluția 2 – figura 2.15

$R1 \leftarrow \text{SELECȚIE} (\text{LOCALITATI}; \text{Jud} = \text{"IS"})$   
 $R2 \leftarrow \text{PROIECȚIE} (R1; \text{Loc}, \text{CodPost})$   
 $R3 \leftarrow \text{SELECȚIE} (\text{LOCALITATI}; \text{Jud} = \text{"VN"})$   
 $R4 \leftarrow \text{PROIECȚIE} (R3; \text{Loc}, \text{CodPost})$   
 $R5 \leftarrow R2 \cup R4$

Prima soluție este, prin simplitate, cea mai tentantă. Este însă un prim caz în care pentru rezolvarea unei probleme pot fi formulate două (sau mai multe) soluții.

**Exemplul 9**

Care sunt codurile produselor care apar deopotrivă în factura 1111 și în factura 1117?

| R1      |         |     | R2      |         |
|---------|---------|-----|---------|---------|
| CodPost | Loc     | Jud | Loc     | CodPost |
| 6600    | Iasi    | IS  | Iasi    | 6600    |
| 5725    | Pascani | IS  | Pascani | 5725    |

| R3      |         |     | R4      |         |
|---------|---------|-----|---------|---------|
| CodPost | Loc     | Jud | Loc     | CodPost |
| 5300    | Focșani | VN  | Focșani | 5300    |

| R5      |         |
|---------|---------|
| Loc     | CodPost |
| Iasi    | 6600    |
| Pascani | 5725    |
| Focșani | 5300    |

Figura 2.15. Exemplul 8 – soluția 2

Tabela din care vor fi extrase datele este LINIIFACT. Soluția se bazează pe intersecția relației care conține produsele prezente în factura 1111 (R2) cu relația produselor prezente în factura 1117 (R4), după cum reiese din figura 2.16.

$R1 \leftarrow \text{SELECTIE} (\text{LINIIFACT}; \text{NrFact} = 1111)$   
 $R2 \leftarrow \text{PROIECȚIE} (R1; \text{CodPr})$

- R1 este noua relație obținută în urma selecției, care va avea aceeași schemă relațională ca R - R1 (A1, A2, ..., An).
  - ⇒ <expresie-logică> poate fi scrisă mai analitic astfel:
    - ⇒ <expresie-logică> = (termen1) și/sau (termen2) ... și/sau (termenk), unde termen j = expresie, Ø expresie<sub>j</sub>.
    - ⇒ expresie<sub>1</sub> sau expresie<sub>2</sub> sunt expresii calculate plecând de la atributele A<sub>i</sub> ale relației R.
    - ⇒ Ø poate fi unul dintre operatorii pentru comparație.

#### Exemplul 1

Pentru a pune în operă savanția notăție de mai sus, luăm în discuție o primă problemă: *Care sunt liniile din R1 pentru care valorile atributelor A și C sunt mai mari decât 20?* Ajungem, astfel, la notația:

$R \leftarrow \text{SELECTIE} (R1; A > 20 \text{ AND } C > 20)$

Tabela R este prezentată în figura 2.7.

| R | A  | B   | C  |
|---|----|-----|----|
|   | 40 | YYX | 25 |

Figura 2.7. Rezultat-selectie – exemplul 1

#### Exemplul 2

*Care sunt județele din Moldova?*

Mai întâi se identifică în baza de date tabela (sau tabelile) din care se extrage rezultatul. În acest exemplu, aceasta este JUDEȚE. Apoi se stabilesc atributele (atributul) asupra căror se va aplica predicatul de selecție. Se obține soluția:

$R \leftarrow \text{SELECTIE} (\text{JUDEȚE}; \text{Regiune} = \text{"Moldova"})$

| R  | Jud     | Judet   | Regiune |
|----|---------|---------|---------|
| IS | Iasi    | Moldova |         |
| VN | Vrancea | Moldova |         |
| NT | Neamt   | Moldova |         |
| SV | Suceava | Moldova |         |
| VS | Vaslui  | Moldova |         |

Figura 2.8. Rezultat-selectie – exemplul 2

#### Exemplul 3

*Care sunt facturile emise în perioada 2-5 august 2000?*

Tabela în care va opera operatorul de selecție este FACTURI. Predicatul de selecție utilizează atributul DataFact:

$R \leftarrow \text{SELECTIE} (\text{FACTURI}; \text{DataFact} \geq 02/08/2000 \text{ AND } \text{DataFact} \leq 05/08/2000)$

| R | NFact | DataFact   | CodCl | Obs                                    |
|---|-------|------------|-------|----------------------------------------|
|   | 1115  | 02/06/2000 | 1001  | NULL                                   |
|   | 1116  | 02/06/2000 | 1007  | Pretul propus initial a fost modificat |
|   | 1117  | 03/06/2000 | 1001  | NULL                                   |
|   | 1118  | 04/06/2000 | 1001  | NULL                                   |

Figura 2.9. Rezultat-selectie – exemplul 3

#### Proiecția (Select)

Prin proiecție, o relație poate fi „decupată” pe verticală. Dacă selecția extrage dintr-o tabelă anumite linii, pe baza condiției îndeplinite de valorile unora dintre atrbute, **proiecția** permite selectarea într-o tabelă rezultat numai a coloanelor (atributelor) dorite dintr-o relație.

Formal, fie o relație R (A<sub>1</sub>, A<sub>2</sub>, ..., A<sub>n</sub>). Proiecția relației R asupra unui subansamblu alcătuit din atrbute proprii este o relație care se obține după parcurgerea a doi pași:

- eliminarea dintre A<sub>i</sub> a celor atrbute care nu sunt specificate și
- suprimarea dublurilor (tuplurile identice).

Se notează:

$R1 \leftarrow \text{PROIECȚIE} (R; A_j, A_k, \dots, A_x)$

Spre deosebire de R, schema relației R1 este alcătuită numai din atrbutele indicate: R1 (A<sub>j</sub>, A<sub>k</sub>, ..., A<sub>x</sub>). Dacă după extragerea coloanelor nu există tupluri (linii) identice, R1 va avea același număr de linii ca și relația R. În caz contrar, numărul lor va fi mai mic, în funcție de numărul dublurilor.

#### Exemplul 4

Incepem, ca de obicei, cu un exemplu ceva mai arid. *Care sunt valorile combinației atrbutelor A și C în relația R1?*

$R \leftarrow \text{PROIECȚIE} (R1; A, C)$

Tabela R are două coloane, A și C, și trei linii, ca în figura 2.10.

| R | A  | C  |
|---|----|----|
|   | 20 | 30 |
|   | 30 | 20 |
|   | 40 | 25 |

Figura 2.10. Rezultat-proiecție – exemplul 4

#### Exemplul 5

*Ce regiuni ale jării sunt preluate în bază?*

Tabela în care se află răspunsul este JUDEȚE. Singura coloană care interesează este Regiune (figura 2.11).

$R \leftarrow \text{PROIECȚIE} (\text{JUDEȚE}; \text{Regiune})$

În primul pas se face decupajul pe verticală, obținându-se o relație notată R', apoi se elimină dublurile, rezultatul final fiind relația R.

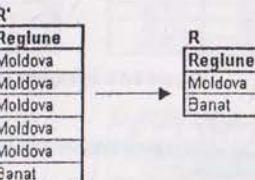


Figura 2.11. Rezultat-proiecție – exemplul 5

| R1     |   |    | R2     |   |    |
|--------|---|----|--------|---|----|
| A      | B | C  | C      | D | E  |
| 20 XYZ |   | 30 | 25 XYZ |   | 30 |
| 30 XXZ |   | 20 | 40 YYX |   | 25 |
| 40 YYX |   | 25 | 30 XXZ |   | 40 |

R1 - R2

| A      | B | C  |
|--------|---|----|
| 20 XYZ |   | 30 |
| 30 XXZ |   | 20 |

Figura 2.6. Diferența a două relații

### Produsul cartezian

Produsul cartezian dintre două relații R1 și R2, denumit de Codd joncțiune încrușită (CROSS JOIN), este ansamblul tuturor tuplurilor obținute prin concatenarea fiecărei linii din tabela R1 cu toate liniile tabelii R2. Formal, dacă notăm cele două relații: R1 (A<sub>1</sub>, A<sub>2</sub>, ..., A<sub>n</sub>) și R2 (B<sub>1</sub>, B<sub>2</sub>, ..., B<sub>m</sub>), produsul cartezian este definit astfel:

$$R1 \otimes R2 = \{ (t_1, t_2) \mid t_1 \in R1 \text{ și } t_2 \in R2 \}$$

Spre deosebire de celelalte trei operațiuni precedente, produsul cartezian nu face apel la noțiunea de relații incompatibile, iar relația-resultat cumulează atributele celor două relații-argument. În figura 2.6 este ilustrat rezultatul produsului cartezian a tabelelor R1 și R2.

| R1     |   |    | R2     |   |    |
|--------|---|----|--------|---|----|
| A      | B | C  | C      | D | E  |
| 20 XYZ |   | 30 | 25 XYZ |   | 30 |
| 30 XXZ |   | 20 | 40 YYX |   | 25 |
| 40 YYX |   | 25 | 30 XXZ |   | 40 |

R1 ⊗ R2

| A      | B | R1.C | R2.C   | D | E  |
|--------|---|------|--------|---|----|
| 20 XYZ |   | 30   | 25 XYZ |   | 30 |
| 20 XYZ |   | 30   | 40 YYX |   | 25 |
| 20 XYZ |   | 30   | 30 XXZ |   | 40 |
| 30 XXZ |   | 20   | 25 XYZ |   | 30 |
| 30 XXZ |   | 20   | 40 YYX |   | 25 |
| 30 XXZ |   | 20   | 30 XXZ |   | 40 |
| 40 YYX |   | 25   | 25 XYZ |   | 30 |
| 40 YYX |   | 25   | 40 YYX |   | 25 |
| 40 YYX |   | 25   | 30 XXZ |   | 40 |

Figura 2.6. Produsul cartezian

Se notează:

$$R6 \leftarrow R1 \otimes R2$$

Tabela-resultat R6 are o nouă structură – şase atrbute (trei preluate din R1 și trei din R2). Întrucât există un atrbut cu nume comun, C, pentru a diferenția cele două aparriții, acestea sunt prefixate, în antetul tabeliei, cu numele relației din care provine.

Prima linie din R6 este obținută prin „alipirea” primului tuplu din R1 cu primul tuplu din R2, a doua din primul tuplu din R1 cu al doilea din R2 etc. Cum R1 are 3 tupluri, iar R2 tot 3, relația-resultat al produsului cartezian are  $3 * 3 = 9$  tupluri.

Nu prea există situații care să reclame folosirea directă și exclusivă a produsului cartezian. Cel mai important „merit” al acestuia în algebra relațională este că permite „alipirea” a două relații, fundamentând astfel operatorul-cheie care este joncțiunea.

### 2.3. Operatorii relaționali

Cei patru operatori prezentați în paragraful precedent sunt generali, spre deosebire de următorii, care sunt specifici algebrei relaționale. De obicei, gruparea operatorilor relaționali se face astfel:

- operatori unari de restricție, care permit decupajul unei relații, pe orizontală – **SELECȚIA** și pe verticală – **PROIECTIA**;
- operatori binari de extensie: **JONCȚIUNEA** și **DIVIZIUNEA**.

O altă deosebire majoră față de paragraful precedent este că vom putea recurge și la exemple concrete din baza de date „cobaï” prezentată în capitolul anterior.

#### Selectia ( $\wedge$ hec)

Selectia triează dintr-o relație (tabelă) numai tuplurile ce satisfac o condiție specificată printr-un predicat.

Ca preambul, definim noțiunea de **formulă F asupra unei relații R ca o expresie logică compusă din:**

- operanzi care sunt nume de atrbute sau constante;
- operatori de comparație aritmetică:  $>$ ,  $\geq$ ,  $<$ ,  $\leq$ ,  $=$ ,  $\neq$ ;
- operatori logici: **ȘI**, **SAU**, **NON**.

Selectia unei relații R, printr-o condiție F, notată SF(R), poate fi definită:

$$SF(R) = \{ \text{tuplu } t \mid t \in R \text{ și } F(t) = \text{adevărată} \}$$

O notare ceva mai pământecă este:

$$R1 \leftarrow \text{SELECTIE } (R; \langle \text{expresie-logică} \rangle)$$

dar, de dragul științei, vom detalia:

- R este relația R (A<sub>1</sub>, A<sub>2</sub>, ..., A<sub>n</sub>) asupra căreia se aplică selecția (A<sub>i</sub> sunt atrbutele sale).

Conținutul tabelei-reuniune R3 este prezentat în figura 2.3. Primele trei tupluri din rezultat sunt preluate din R1, iar ultimele două din R2. R3 are numai cinci tupluri deoarece un tuplu este comun tabelelor R1 și R2. Algebra relațională elimină automat dublurile (tuplurile identice), astfel încât restricția de unicitate este asigurată după orice operație.

| R1 |     |    | R2 |     |    |
|----|-----|----|----|-----|----|
| A  | B   | C  | C  | D   | E  |
| 20 | XYZ | 30 | 25 | XYZ | 30 |
| 30 | XXZ | 20 | 40 | YYX | 25 |
| 40 | YYX | 25 | 30 | XXZ | 40 |

| R1 ∪ R2 |     |    |
|---------|-----|----|
| A       | B   | C  |
| 20      | XYZ | 30 |
| 30      | XXZ | 20 |
| 40      | YYX | 25 |
| 25      | XYZ | 30 |
| 30      | XXZ | 40 |

Figura 2.3. Reuniunea a două relații

Există suficiente situații informaționale care fac uz de reuniunea a două tabele. Să luăm două exemple:

- tabelele ce reflectă tranzacții economice pot fi descompuse (sparte) în funcție de anul (luna sau cincinalul, deceniu) la care se referă; dacă ne raportăm la baza noastră de date, tabela LINIIFACT poate fi ruptă în alte două: LF\_2000\_2001, care ar conține facturile emise în anii 2000 și 2001, și LINIIFACT, ce conține numai înregistrările aferente anului calendaristic 2002. Începând cu 2002, orice situație statistică privind vânzările în perioada 2000-2002, 2000-2003 etc. necesită reuniunea celor două tabele, LF\_2000\_2001 și LINIIFACT.
- pentru a afla care sunt clienții care au cumpărat cel puțin unul din produsele *Produs 1* și *Produs 2*, se poate proceda la reuniunea tabelei ce conține clienții care au cumpărat *Produs 1* cu tabela clienților care au cumpărat *Produs 2*.

Reuniunea este comutativă. Singura problemă neclară ar fi legătura de numele atributelor în relația rezultat. În acest sens, se poate institui regula potrivit căreia numele atributelor relației-reuniune sunt numele primei relații participante în operație. Aceasta nu are importanță asupra comutativității, deoarece conținutul tabelei rezultat este identic, indiferent care este prima relație enumerată.

### Intersecția

Intersecția a două relații unicompabile, R1 și R2, poate fi definită astfel:

$$R1 \cap R2 = \{ \text{tuplu } t \mid t \in R1 \text{ și } t \in R2 \}$$

Se notează:

$$R4 \leftarrow R1 \cap R2.$$

Conținutul tabelei-intersecție R4 este prezentat în figura 2.4. Cum numai un tuplu este absolut identic și în R1 și R2, tabela rezultat este alcătuită dintr-o singură linie.

| R1 |     |    | R2 |     |    |
|----|-----|----|----|-----|----|
| A  | B   | C  | C  | D   | E  |
| 20 | XYZ | 30 | 25 | XYZ | 30 |
| 30 | XXZ | 20 | 40 | YYX | 25 |
| 40 | YYX | 25 | 30 | XXZ | 40 |

| R1 ∩ R2 |     |    |
|---------|-----|----|
| A       | B   | C  |
| 40      | YYX | 25 |

Figura 2.4. Intersecția a două relații

Exemple de informații care fac necesară recurgerea la intersecție:

- pentru a afla care sunt clienții care au cumpărat și *Produs 1*, și *Produs 2*, se poate proceda la intersecția tabelei clienților care au cumpărat *Produs 1* cu tabela alcătuită din clienții care au cumpărat *Produs 2*;
- zilele în care s-au făcut vânzări și clientului *Client 1 SRL*, și clientului *Client 2 SA*;
- persoanele de la firmele-client care cumulează posturile de *Director vânzări* și *Sef aprovizionare*.

Ca și reuniunea, intersecția este comutativă, iar numele atributelor relației-intersecție sunt extrase din prima relație participantă în operație.

### Diferența

Diferența a două relații unicompabile, noteate R1 și R2, este definită astfel:

$$R1 - R2 = \{ \text{tuplu } t \mid t \in R1 \text{ și } t \notin R2 \}$$

Se notează:

$$R5 \leftarrow R1 - R2.$$

Conținutul tabelei-diferență R5 (figura 2.5) conține numai tuplurile din prima relație, R1, care nu se regăsesc în a doua relație, R2. Așadar, din rezultat este eliminat al treilea tuplu din R1, deoarece valorile acestuia există și în R2 (al doilea tuplu din R2).

Exemple de informații care fac necesară recurgerea la diferență:

- care sunt clienții care au cumpărat *Produs 1* dar nu au cumpărat *Produs 2*?
- care sunt zilele în care s-au făcut vânzări clientului *Client 1 SRL*, dar nu există nici o factură către *Client 2 SA*?

Spre deosebire de reuniune și intersecție, diferența nu este este comutativă. Atributele relației-diferență sunt cele ale primei relații (descăzutul), iar tuplurile care sunt extrase din relația-descăzut nu se regăsesc în relația-scăzător. În plus, nu există restricții privind cardinalitatea (numărul de tupluri) celor două relații, adică nu este musaj ca relația-descăzut să conțină mai multe tupluri decât cea scăzător.

Elementul definitoriu pentru limbajele de manipulare bazate pe calculul predicatorilor il reprezintă noțiunea de variabilă, care poate fi asociată fie tuplurilor, fie domeniilor.

O altă grupare delimită limajele non-grafice de cele grafice. Primele permit reprezentarea unei consultări „in linie”, prin dispunerea succesivă a operatorilor, atributelor și relațiilor. Cele grafice permit redactarea consultării în mod interactiv, prin afișarea pe ecran a unui sistem de meniu și elemente de dialog din care opțiunile pot fi selectate și modificate cu ajutorul mouse-ului (sau claviaturii); în plus, se mai poate opera o delimitare și pentru limajele grafice în funcție de utilizarea explicită sau implicită a variabilelor de domeniu.

Deși nu exagerat de recentă, o clasificare încă valabilă (în parte) a limajelor relaționale poate fi prezentată ca în figura 2.1<sup>32</sup>.

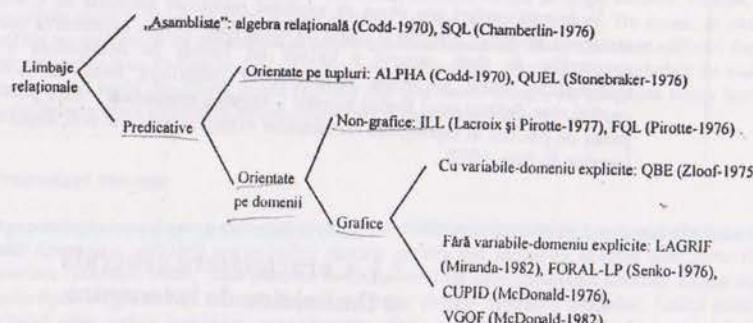


Figura 2.1. O clasificare a limajelor relaționale

Există o serie de caracteristici comune tuturor limajelor:

- operatorii relaționali se aplică relațiilor luate în întregime, adică tuturor tuplurilor care alcătuiesc relațiile respective;
- rezultatul fiecărui operator (rezultatul consultării) este o nouă relație ce poate servi ca argument într-o altă consultare și.a.m.d.;
- logica operatorilor se bazează pe valorile atributelor, ceea ce constituie, de altminteri, suportul singurul mod de acces în BD. Intrucât consultările mono- și multi-relații sunt efectuate exclusiv prin compararea valorilor atributelor (definite pe domenii compatibile), accesul total independent de limaj este asigurat.

Înaintea redactării unei consultări într-un limaj relațional trebuie parcursă o fază de analiză, pentru determinarea atributelor rezultatului, legăturilor dintre tabele și eventualelor condiții (restricții) ce trebuie respectate.

Limbajul algebraic relațional cuprinde două tipuri de operatori: asamblări – REUNIUNE, INTERSECTIE, DIFERENȚĂ, PRODUS CARTEZIAN – și relaționali – SELECTIE, PROIECTIE, JONCTIUNE și DIVIZIUNE.

O altă clasificare distinge operatorii fundamentali, ireductibili (reuniunea, diferența, produsul cartezian, selecția, proiecția), de cei derivați, a căror funcționalitate poate fi realizată prin combinarea operatorilor fundamentali (intersecția, jonctiunea, diviziunea).

32. [Miranda&Busta90], vol. 2, p. 30.

Pentru cele ce urmează se notează cu:

- $t$  sau  $r$ , un tuplu al unei relații (linie a unei tabele) și
- $t(A)$ , un sub-tuplu al relației  $R$ , relativ la atributul  $A$  (valoarea atributului  $A$  în linia  $t$ ).

Algebra, ca și calculul relațional, servește ca punct de referință în caracterizarea unui limaj ca fiind complet sau incomplet, din punct de vedere relațional. Dacă un limaj permite exprimarea tuturor operatorilor enumerați mai sus și oferă cel puțin facilități algebrei relaționale, se spune că acesta este un limaj relațional complet<sup>33</sup>.

Mai trebuie amintit că în literatura de specialitate există o multitudine de reprezentări ai operatorilor algebrei relaționale. Notația pe care o vom utiliza în cele ce urmează este cea mai simplă și ușor de înțeles (*părere mea, vorba lui...*).

## 2.2. Operatorii asamblări

Trei dintre operatorii asamblări – reuniunea („ $\cup$ ”), intersecția („ $\cap$ ”) și diferența („ $-$ ”) – pot opera numai cu două relații unicompatibile.

Fie  $R1 (A_1, A_2, \dots, A_n)$  și  $R2 (B_1, B_2, \dots, B_m)$  două relații. Se spune despre  $R1$  și  $R2$  că sunt unicompatibile dacă:

- $n = m$ ,
- $\forall i \in \{1, 2, \dots, n\}, A_i$  și  $B_i$  sunt de același tip sintactic (aceasta nu înseamnă că trebuie să prezinte, neapărat, domenii identice de definire).

Relațiile  $R1$  și  $R2$  din figura 2.2 sunt unicompatibile deoarece:

1. ambele au același număr de atribute;
2. atributele  $A, B, C$  din  $R1$  (le putem nota și  $R1.A, R1.B, R1.C$ ) corespund sintactic (sunt de același tip) atributelelor  $C, D$  și  $E$  din  $R2$  ( $R2.C, R2.D, R2.E$ ).

| R1 |     |    | R2 |     |    |
|----|-----|----|----|-----|----|
| A  | B   | C  | C  | D   | E  |
| 20 | XYZ | 30 | 25 | XYZ | 30 |
| 30 | XXZ | 20 | 40 | YYX | 25 |
| 40 | YYX | 25 | 30 | XXZ | 40 |

Figura 2.2. Două relații uicompatibile

### Reuniunea

Reuniunea a două relații uicompatibile,  $R1$  și  $R2$ , este definită astfel:

$$R1 \cup R2 = \{ \text{tuplu } t \mid t \in R1 \text{ sau } t \in R2 \}.$$

Se notează:

$$R3 \leftarrow R1 \cup R2.$$

33. [Salch94], p. 35.

- derivată poate fi operat nu numai în interogări, dar și în operațiuni de inserare/modificare/ștergere.
9. *Independența fizică a datelor.* Aplicațiile-program și activitățile de tip „consolă” (interogări directe) rămân neschimbate, din punct de vedere logic, la modificarea în structurile de stocare și metodele de acces.
  10. *Independența logică a datelor.* Această regulă permite modificarea dinamică a modelului logic al bazei de date, cum ar fi, spre exemplu, o descompunere (spargere) sau jonctionare de tabele care nu atrage pierderi de informații.
  11. *Independența mecanismului de integritate al bazei.* Restricțiile de integritate sunt definite printr-un limbaj relațional și stocate în dicționarul de date (catalog), și nu în aplicațiile ce exploatează BD. Dintre aceste restricții trebuie să fie implementabile măcar două, ceea ce entității și cea referențială.
  12. *Regula independenței distribuirii datelor.* Limbajul relațional operează asupra datelor care sunt plasate atât pe calculatorul pe care se derulează activitățile, dar și pe alte calculatoare legate în rețea, de o manieră transparentă pentru utilizator.
  13. *Regula non-subversiunii.* Dacă un sistem relațional prezintă un limbaj de nivel scăzut (apropiat de „mașină”), care permite prelucrarea, pe rând, a fiecărui înregistrare (linii sau tuplu), acest limbaj nu poate fi utilizat pentru a încălca restricțiile declarate printr-un alt limbaj, de nivel înalt, al SGBD.

Cum găsirea unor SGBD-uri care să indeplinească toate „poruncile” se dovedește o muncă aproape zadarnică (nu cunoște nici un SGBD în totală conformitate cu toate cele 13 reguli), s-a încercat formularea unor *cerințe minime* pentru care un sistem de gestiune a bazelor de date să fie declarat „relațional”. Acestea ar fi:

- ① Toate datele bazei sunt organizate în relații.
- ② Între tabele nu există pointeri care să fie gestionati de utilizatori.
- ③ Sunt implementați operatorii relaționali: selecție, proiecție și jonctionare naturală.

Un sistem este *complet relațional* dacă, în plus:

- ④ Sunt implementați toți operatorii algebrici relaționali.
- ⑤ Sunt respectate restricțiile de unicitate a cheii și cea referențială.

Sistemele care îndeplinesc numai condițiile ① și ③ sunt *pseudorelaționale*, iar dintre cele pseudorelaționale, cele care îndeplinesc ③ numai în raport cu funcția de interogare sunt denumite SGBD-uri cu *interfață relațională*.

În afară celor 13 porunci, Codd a mai specificat 9 reguli structurale, 3 reguli de integritate și 18 reguli de manipulare. Ulterior, a transformat poruncile în veritabilă Constituție, formulând, în a doua versiune a modelului relațional, 333 de reguli. Dacă se mai străduiește un pic și mai extrage încă un rând de specificații, obținem 666 de reguli, iar atunci se poate spune că modelul relațional este unul dumnezeiesc.

## Capitolul 2

### ALGEBRA RELAȚIONALĂ

În primul capitol am văzut că modelul relațional formulat de Codd are la bază trei elemente: structuri, operații și reguli de integritate. Pentru exprimarea operațiilor aplicabile structurilor de date, autorul a definit un limbaj de manipulare a datelor puternic matematizat, bazat pe teoria ansamblurilor (seturilor), numit *DSL/Alpha*<sup>31</sup>. Prezentul capitol este dedicat unui limbaj teoretic – algebra relațională – care poate constitui un bun punct de plecare în înțelegerea chestiunilor esențiale ale celui mai important limbaj dedicat bazelor de date – SQL.

#### 2.1. Caracterizare generală a limbajelor de interogare

Gestiunea bazelor de date relaționale are ca obiectiv principal acoperirea nevoilor informaționale ale conducerii firmei la toate nivelurile. Până la consacrarea SGBDR-urilor, extragerea informațiilor dorite din baza de date se realiza prin aplicații dezvoltate exclusiv cu limbaje *procedurale*, în care se precizează atât datele dorite, cât și metodele de căutare și extragere a acestora. Actuala generalizare a SGBDR-urilor se află într-o strânsă relație cu elaborarea și implementarea unor limbaje performante pentru manipularea BD – limbajele de interogare.

Limbajele relaționale sunt *neprocedurale*: utilizatorul definește datele ce trebuie extrase din BD, sarcina căutării și extragerii fiind „rezervată” exclusiv SGBD-ului. De asemenea, în unele lucrări, acestea sunt considerate *limbaje închise*, deoarece o consultare generează o nouă relație ce poate fi utilizată, la rândul său, ca argument în alte consultări și.a.m.d.

Pornind de la cele două modalități de definire a relației, pe o parte, ca predicat aplicat asupra unor domenii și, pe altă parte, ca ansamblu de tupluri, limbajele de manipulare a datelor sunt grupate în două mari categorii: *limbaje predicative* – fondate pe teoria predicatelor și *limbaje asambliste* – fondate pe teoria ansamblurilor (tuplurilor).

Limbajele *predicative* sunt divizate în alte două sub-clase:

- cele care au la bază *calculul relațional asupra tuplurilor* (limbaje orientate pe tupluri) și
- cele în care *calculul relațional se aplică asupra domeniilor* (limbaje orientate pe domenii).

31. Vezi și [Date99-1].

Pe de altă parte, nu toate tabelele derivate sunt atât de problematice precum cea discutată anterior, astfel încât propagarea modificărilor dintr-o relație virtuală poate fi destul de simplă. După cum vom vedea în capitolile viitoare, restricțiile la care trebuie să se supună o tabelă derivată potru a putea fi modificabilă diferă de la un SGBD la altul.

Câteva dintre SGBDR-urile actuale propun un alt mecanism oarecum apropiat de cel al tabelei virtuale, anume cel de *cliușeu sau imagine concretă*<sup>28</sup>. Această noțiune este fondată pe conținutul unei „*imaginii*” calculate pe baza definiției sale și stocată în baza de date.

O „*imagine concretă*” reflectă starea bazei de date la un moment *t* când a fost „calculated”, cliușel fiind deci determinat periodic de către sistem. Actualizarea bazei de date va fi reflectată în cliușeu abia în momentul primului calcul de după aceasta. Firește, la un volum mare al bazei de date, timpul de calcul poate fi destul de lung. De aceea, se poate institui un mecanism de schimbare în cliușeu numai a tuplurilor ce au suferit modificări după calculul precedent. Probleme mai delicate apar la ștergerea unor tupluri în tabelele de bază, deoarece un tuplu din cliușeu poate proveni din mai multe relații, iar ștergerea unuia intr-o astfel de relație nu trebuie să atragă neapărat ștergerea din cliușeu.

### Proceduri stocate

**O procedură stocată este o secvență de program (cod) care face parte integrantă din baza de date.** Avantajele utilizării procedurilor stocate decurg din faptul că acestea sunt parte din structura (schema) bazei, fiind păstrate în dicționarul de date (catalogul sistem). Există mai multe tipuri de proceduri stocate: funcții-utilizator pentru validarea tabelelor, funcții pentru calculul unor valori implicate, proceduri/funcții de validare la nivel de linie sau tabelă, funcții/proceduri de calcul a unor expresii complexe etc.

**Declanșatorul (trigger)** este un tip special de procedură stocată care este executată automat când un eveniment predefinit (înserare, actualizare sau ștergere) modifică o tabelă. Utilitatea declanșatoarelor este evidentă la formularea unor restricții mai complexe decât „suportă” comenzi CREATE/ALTER TABLE, actualizarea automată a unor attribute calculate, jurnalizarea modificărilor suferite de baza de date, păstrarea integrității referențiale etc.

Există diferențe sensibile între SGBD-uri și în ceea ce privește sintaxa declanșatoarelor, deoarece acestea sunt redactate în extensiile procedurale ale SQL care prezintă diferențieri majore de la un producător la altul. Poate fi evocată, spre exemplu, diversitatea declanșatoarelor în Oracle, DB2 etc. față de Visual FoxPro.

Astfel, dacă în VFP există numai trei tipuri de declanșatoare, pentru adăugare, pentru modificare și pentru ștergere, în schimb, în Oracle, pentru fiecare din cele trei operațiuni există o primă separare între declanșatoare lansate *înaintea* operației (actualizării) și cele după operație. La această delimitare se mai adaugă și cea dintre *trigger* la nivel de linie, ce intră în acțiune la fiecare inserare/modificare/ștergere a unei linii, și cele la nivel de comandă, care se execută o singură dată, indiferent către linii afectează comanda de actualizare respectivă (*row* versus *statement*).

Detalii despre declanșatoare în Visual FoxPro și Oracle vor fi prezentate în capitolul 8.

28. Vezi [Salch94], p. 64.

### 1.7. Regulile modelului relațional

În 1985, Codd a publicat în revista *Computerworld* un articol în care formulează 12 reguli, plus regula 0, de fundamente, care ar trebui respectate de un SGBD pentru a fi declarat relațional<sup>29</sup>. Mai mult, autorul s-a angajat cu inverșunare împotriva celor care nu folosesc abuziv sintagma *relațional* pentru produsele lor, chiar două firme fiind aduse în pragul falimentului de atacurile „demascatoare” ale lui Codd, sau cel puțin așa spune legenda<sup>30</sup>.

Discutabile, ca orice reguli, „cele 13 porunci” constituie un etalon plauzibil de a caracteriza funcționalitățile unui SGBD și a-l încadra pe scara relațională.

1. **Regula de fundament.** Orice sistem declarat relațional trebuie să fie capabil să administreze întreaga bază de date exclusiv prin funcțiuni relaționale.
2. **Regula informației.** Toate informațiile sunt reprezentate explicit, la nivel logic, într-un singur mod, ca valori în anumite tabele.
3. **Regula accesului garantat.** Orice dată elementară (reprezentată printr-o valoare atomică) este accesibilă, fără ambiguitate, prin cunoașterea numelui tabelei din care face parte, a denumirii atributului care o reprezintă și a valorii cheii primare a tuplului pe care se găsește.
4. **Prelucrarea sistematică a valorilor nule.** Un SGBDR utilizează valorile nule (diferite de valorile vide, spații sau 0) în vederea reprezentării informațiilor care lipsesc, necunoscute sau inaplicabile. Poate avea valori NULLe orice tip de atribut.
5. **Regula catalogului actualizabil în timp real.** Descrierea unei BDR este stocată, la nivel logic, într-un catalog (dicționar de date) alcătuit din tabele-sistem ce pot fi consultate de o manieră similară tabelelor de date obișnuite.
6. **Regula sub-limbajului de date** se referă la existența cel puțin a unui limbaj (ca SQL, de exemplu) „dotat” cu o serie de comenzi cu sintaxă bine definită, prin care să se realizeze:
  - definirea (descrierea) datelor;
  - definirea sub-schemelor („machete” sau tabele virtuale);
  - manipularea datelor (interactiv sau prin program);
  - definirea și implementarea restricțiilor de integritate;
  - autorizarea accesului la date;
  - gestiunea tranzacțiilor.
 Limbajul trebuie să prezinte o sintaxă liniară și să poată fi utilizat atât interactiv, cât și din cadrul aplicațiilor.
7. **Regula actualizării tabelelor virtuale.** Orice tabelă derivată teoretic actualizabilă trebuie să aibă același regim în cadrul SGBD-ului.
8. **Inserarea, modificarea și ștergerea** pot fi efectuate prin intermediul unui limbaj de nivel înalt, orientat pe lucrul simultan cu un ansamblu de linii. Ca efect, o tabelă de bază sau

29. O prezentare a celor 13 reguli se găsește în lucrarea [Pascu&Pascu94], pp. 22-27. Vezi și [Lungu §.a. 95], pp. 133-135.

30. Vezi și *Relational Philosopher*, în [Celko99], pp. 34-35.

## 1.6. Alte noțiuni ale SGBD-urilor relaționale

Este greu de inventariat toate sintagmele vehiculate de dezvoltatorii de aplicații cu baze de date, cu atât mai mult cu cât unele sunt specifice fiecărui produs. În cele ce urmează ne vom opri, pentru câteva minute, la tabelele derivate (view-urile) și procedurile stocate.

### Tabele virtuale (view-uri)

O altă noțiune (alunecoasă) a modelului relațional este cea căreia, prin traducerea din „originalul” view (imagină, vedere), își asociază multiple titulaturi în literatura de specialitate și practica din țara noastră: *imagină, relație (tabelă) virtuală, relație (tabelă) derivată sau relație (tabelă) dinamică*.

O relație virtuală stabilește o legătură semantică între relații statice și/sau alte relații dinamice, nefiind definită explicit, prin tupluri proprii, ca o relație de bază (statică), ci printr-o expresie relațională. Conținutul (instantierea) relației virtuale depinde, la un moment oarecare dat, de conținutul tabelelor de bază din care derivă. Pentru a înțelege mai bine diferența, explicația se poate rezuma astfel: *tabela virtuală este cea pentru care pe disc  
se memorează numai schema, nu și conținutul*.

Tabelele virtuale oferă oricărui utilizator al unei baze de date posibilitatea prezentării datelor în funcție de nevoile sale specifice. De asemenea, rațiuni de securitate și confidențialitate a anumitor informații pot conduce la izolarea unor date față de utilizatorii neautorizați, lucru deplin posibil prin intermediul *imaginilor*. Pornind de la aceleași tabele de bază, se pot crea un mare număr de tabele virtuale, în funcție de situație.

Tabelele derivate sunt suportul creării schemelor externe. O dată definită, o tabelă virtuală poate fi privită ca o tabelă de bază oarecare. De asemenea, ca și relațiile de bază, și „imaginile” pot fi actualizate, ceea ce atrage modificarea tabelelor statice din care derivă. În aceste situații apar o serie de probleme. Este clar că modificarea conținutului tabelelor de bază presupune modificarea conținutului unei tabele derivate. Însă ce informații pot fi modificate în tabelele de bază, pornind de la modificările unei tabele virtuale? Soluțiile implementate în SGBDR-urile actuale diferă de la caz la caz.

Pentru a intra în câteva detalii, să imaginăm o tabelă derivată denumită *INCASARI\_CLIENTI* – figura 1.25 –, ce conține documentul de incasare, suma incasată (corespunzătoare documentului) și clientul care a efectuat plată.

| INCASARI_CLIENTI |        |       |             |            |
|------------------|--------|-------|-------------|------------|
| Client           | CodDoc | NrDoc | DataDoc     | Suma_Plata |
| Client 1 SRL     | CEC    | 444   | 10-Aug-2000 | 863636     |
| Client 1 SRL     | OP     | 111   | 10-Aug-2000 | 5689636    |
| Client 1 SRL     | OP     | 333   | 9-Aug-2000  | 2420124    |
| Client 1 SRL     | OP     | 666   | 11-Aug-2000 | 327316     |
| Client 2 SA      | OP     | 555   | 10-Aug-2000 | 1027295    |
| Client 5 SRL     | CHIT   | 222   | 15-Aug-2000 | 431818     |

Figura 1.25. O tabelă derivată

Utilitatea unei asemenea relații virtuale poate fi certificată de un angajat al compartimentului financiar care se ocupă, printre altele, și cu evidența incasărilor de la clienți. Pe că de utilă, pe atât de problematică devine *INCASARI\_CLIENTI* atunci când se pune problema actualizării sale și propagării modificărilor în tabelele de bază din care provine, *CLIENTI*, *FACTURI*, *INCASARI* și *INCASFACT*.

Iată câteva dintre probleme:

- modificarea atributului *Suma\_Plata* este imposibil de operat în tabela de bază, *INCASFACT*. O linie din tabela derivată corespunde uneia sau mai multor linii din tabela de bază (în funcție de căte facturi sunt achitate prin documentul de plată respectiv). Dacă pentru ordinul de plată 111 din 10 august 2000 se dorește modificarea (corecția) sumei din 5.689.636 în 5.600.000, nu se poate cunoaște tranșa și factura unde s-a comis eroarea și, normal, unde trebuie operată modificarea.
- modificarea celui de-al cincilea tuplu din (Client 2 SA, OP, 555, 10-Aug-2000, 1027295) în (Client 5 SRL, OP, 555, 10-Aug-2000, 1027295), adică modificarea clientului pentru Ordinul de plată 222 poate fi operată în trei moduri în tabele de bază:
  - fie se modifică în *FACTURI* pentru factura 1113 valoarea atributului *CodCl* din 1002 în 1005;
  - fie se modifică în linia din *INCASFACT* codul incasării din 1238 în 1235;
  - fie se modifică în *INCASFACT* valoarea atributului *NrFact* din 1113 în 1112, păstrând neschimbă codul incasării.

Chiar dacă nu la fel de plauzibile, cele trei variante generează o „stare” de confuzie pentru SGBD.

- inserarea unei linii în tabela derivată este o acțiune temerară, dar fără prea mulți sorți de izbândă: oricare ar fi cele trei tabele de bază unde are loc inserarea, cel puțin un atribut important (ce nu poate avea valori NULL) nu are valori specificate, astfel încât se încalcă una dintre restricții (de entitate sau comportament). Spre exemplu, convenim că inserarea trebuie să se propage numai în *INCASFACT*. Nu se cunoaște însă numărul facturii incasate. Cum *NrFact* este un component al cheii primare a tabelei, este clar că operațiunea va fi interzisă de SGBD.
- stergerea unei linii din tabela derivată ridică problema identificării liniei sau liniilor din una sau mai multe tabele de bază.

Păstrarea integrității BD în momentul actualizării unei tabele derivate se poate rezolva, în principiu, prin „apelarea” la una dintre următoarele trei soluții<sup>27</sup>:

- Actualizarea unei relații dinamice se face exclusiv pornind de la tabelele de bază din care derivă. Este cea mai restrictivă și în contradicție cu regulile ce definesc un SGBDR, aşa cum au fost ele definite de Codd.
- Definirea unor proceduri generale de corespondență, pornindu-se de la expresia relațională de creare a tabelei derivate, proceduri pe baza cărora SGBD-ul va „traduce” actualizarea tabelei virtuale în modificări ale tabelelor de bază.
- Controlul actualizării relației derivate prin intermediul unui sub-sistem de integritate al bazei de date. Această soluție este considerată ca fiind cea mai eficace pentru păstrarea coerentei și securității întregului ansamblu de relații ce alcătuiesc baza de date, fie ele statice sau dinamice.

27. După [Saleh94], p. 60.

- data documentului de incasare nu poate fi anterioară datei de început a aplicației: 1 august 2000 (DataDoc >= 01-08-2000);
- codul documentului se scrie numai cu majuscule.

| INCASARI |            |        |       |            |
|----------|------------|--------|-------|------------|
| CodInc   | DataInc    | CodDoc | NrDoc | DataDoc    |
| 1234     | 15/08/2000 | OP     | 111   | 10/08/2000 |
| 1235     | 15/08/2000 | CHIT   | 222   | 15/08/2000 |
| 1236     | 16/08/2000 | OP     | 333   | 09/08/2000 |
| 1237     | 17/08/2000 | CEC    | 444   | 10/08/2000 |
| 1238     | 17/08/2000 | OP     | 555   | 10/08/2000 |
| 1239     | 18/08/2000 | OP     | 666   | 11/08/2000 |

Figura 1.23. Tabela INCASARI

Tabela INCASFACT (figura 1.24) detaliază tabela precedentă și indică ce facturi (tranșe din facturi) sunt achitate prin fiecare incasare. Un client poate plăti mai multe facturi odată (printr-o singură plată). Pe de altă parte, orice factură poate fi plătită în una sau mai multe tranșe, în funcție de banii de care dispune clientul la momentul întocmirii documentului de plată. Atributele tabelei sunt:

- CodInc – codul incasării;
- NrFact – factură pentru care se incasează valoarea integrală sau numai o tranșă;
- Transa – tranșă din factură (sau întreaga valoare) care se incasează prin documentul primar ce stă la baza incasării.

| INCASFACT |        |         |
|-----------|--------|---------|
| CodInc    | NrFact | Transa  |
| 1234      | 1111   | 5399625 |
| 1234      | 1118   | 1026375 |
| 1235      | 1112   | 487705  |
| 1236      | 1117   | 975410  |
| 1236      | 1118   | 1026375 |
| 1236      | 1120   | 731557  |
| 1237      | 1117   | 975410  |
| 1238      | 1113   | 1160250 |
| 1239      | 1117   | 369680  |

Figura 1.24. Tabela INCASFACT

Cheia primară este combinația CodInc+NrFact, deoarece la o incasare se pot achita mai multe facturi, iar pe de altă parte, o factură poate fi plătită în mai multe tranșe. CodInc și NrFact sunt chei străine.

#### Comentarii suplimentare privind restricțiile referențiale

Instituirea unei restricții referențiale într-o bază de date interzice apariția de valori nenele în tabele-copil care nu se regăsesc în tabelele-părinte (linii orfane). De aceea, la inserarea unei linii într-o tabelă-copil sau modificarea unei chei străine, SGBD-ul trebuie să verifice dacă noile valorile se regăsesc în linii-părinte.

Inserarea unei linii într-o tabelă-părinte, ca și ștergerea de linii dintr-o tabelă-copil nu prezintă nici un pericol „referențial”.

La ștergerea unei linii dintr-o tabelă-părinte trebuie precizat cum se va prezerva restricția referențială: fie prin ștergerea în cascadă a tuturor înregistrărilor-copil (ON DELETE CASCADE), fie, pur și simplu, prin interzicerea ștergerii (ON DELETE RESTRICT).

În fine, la modificarea unei chei primare/alternative pentru care există, în alte tabele, înregistrări-copil trebuie precizată acțiunea întreprinsă de SGBD: modificarea în cascadă a tuturor liniilor-copil (ON UPDATE CASCADE) sau interzicerea modificării, dacă există cel puțin o înregistrare-copil (ON UPDATE RESTRICT).

Pentru exemplificare, să luăm cazul tabelei FACTURI și trei situații legate de actualizarea acestora.

- **Inserarea unei linii.** Deoarece FACTURI este legată (ca tabelă-copil) prin restricție referențială de tabela CLIENTI (părinte), trebuie verificat dacă valoarea CodCl există în CLIENTI. Dacă nu, inserarea trebuie proibită.
- **Modificarea unei linii.** Aici sunt două situații, corespunzătoare posturilor de părinte și copil a tabelei modificate. Astfel:
  - dacă se modifică valoarea atributului CodCl, trebuie verificat dacă aceasta se regăsește în CLIENTI și, dacă nu, operațiunea trebuie anulată;
  - dacă se modifică valoarea lui NrFact, atunci trebuie testat dacă există înregistrări-copil în tabelele LINIIFACT și INCASFACT. Dacă da,
    - fie se interzice modificarea (ON UPDATE RESTRICT),
    - fie se modifică în cascadă toate liniile-copil – valorile NrFact din LINIIFACT și INCASFACT (ON UPDATE CASCADE).

Fără, nu e obligatoriu ca acțiunea să fie identică pentru ambele tabele-copil. Cu alte cuvinte, se poate recurge, pentru LINIIFACT, la ON UPDATE CASCADE, iar pentru INCASFACT, la ON UPDATE RESTRICT.

- **Ștergerea unei linii.** Atunci trebuie testat dacă există înregistrări-copil în tabelele LINIIFACT și INCASFACT. Dacă da,
  - fie se interzice ștergerea (ON DELETE RESTRICT),
  - fie se șterg în cascadă toate liniile-copil din LINIIFACT și INCASFACT (ON DELETE CASCADE).

Există diferențe semnificative între SGBD-uri în privința definirii acțiunilor ce trebuie întreprinse pentru respectarea integrităților referențiale. Unele produse – firește, dintre cele mai bine cotate – permit, la declararea restricțiilor referențiale (este de obicei simultană cu crearea tabelelor), instituirea oricărui reguli din cele patru: ON UPDATE CASCADE, ON UPDATE RESTRICT, ON DELETE CASCADE, ON DELETE RESTRICT.

Altele, precum Oracle 8, permit declararea acțiunilor numai pentru ștergere, ON DELETE CASCADE și ON DELETE RESTRICT. Pentru modificarea în cascadă sunt necesare declanșatoare speciale construite în limbajul de dezvoltare – PL/SQL.

Cât privește Visual FoxPro 6, al treilea SGBD pe care il vom folosi în exemplele din capitolele următoare, este interesant faptul că toate acțiunile pot fi definite, dar numai grafic, prin Referential Integrity Builder. În schimb, dezvoltatorii de aplicații trebuie să scrie și să implementeze declanșatoare în care să se descrie explicit ce trebuie întreprins în fiecare situație ce pune în pericol restricțiile referențiale.

|    |               |              |           |      |
|----|---------------|--------------|-----------|------|
| 67 | Vodca Scandic | cutie 1 l    | Spirtoase | 0.22 |
| 68 | Vodca Scandic | cutie 200 ml | Spirtoase | 0.22 |

Figura 1.20. Diferențierea (în afară de cod) numai prin UM

Este dificil de spus care variantă este mai bună. Eu, unul, aș倾ina pentru cea de-a doua, caz în care atributul DenPr nu este cheie candidată.

Tabela FACTURI (figura 1.21) conține câte o linie pentru fiecare factură emisă, factură ce reflectă o vânzare (către un client). Atribute:

- NrFact – numărul facturii;
- DataFact – data întocmirii facturii;
- CodCl – codul clientului căruia i s-au vândut produsele/serviciile consemnate în factură;
- Obs – observații; e folosit relativ rar, pentru a introduce eventuale detalii sau probleme care au apărut în legătură cu o factură.

| FACTURI |            |       |                                        |
|---------|------------|-------|----------------------------------------|
| NrFact  | DataFact   | CodCl | Obs                                    |
| 1111    | 01/08/2000 | 1001  | NULL                                   |
| 1112    | 01/08/2000 | 1005  | Probleme cu transportul                |
| 1113    | 01/08/2000 | 1002  | NULL                                   |
| 1114    | 01/08/2000 | 1006  | NULL                                   |
| 1115    | 02/08/2000 | 1001  | NULL                                   |
| 1116    | 02/08/2000 | 1007  | Pretul propus initial a fost modificat |
| 1117    | 03/08/2000 | 1001  | NULL                                   |
| 1118    | 04/08/2000 | 1001  | NULL                                   |
| 1119    | 07/08/2000 | 1003  | NULL                                   |
| 1120    | 07/08/2000 | 1001  | NULL                                   |
| 1121    | 07/08/2000 | 1004  | NULL                                   |
| 1122    | 07/08/2000 | 1005  | NULL                                   |

Figura 1.21. Tabela FACTURI

Cheia primară este atributul NrFact. Explicația este una simplă: gestionarea facturilor emise este, conform legii, strictă. Nu pot exista două facturi (care reflectă două vânzări) cu un același număr. Sau, de fapt, pot exista, dar aceasta înseamnă o ilegalitate destul de gravă (se mai poate incă prin economia subterană). CodCl este cheie străină (tabela-părinte este CLIENTI). Ca restricție-utilizator suplimentară, nu pot fi introduse facturi întocmite înainte de 1 august 2000 (DataFact >= {01/08/2000}).

Tabela LINIIFACT (figură 1.22) detaliază tabela precedentă. Un tuplu se referă la un produs/serviciu vândut și consemnat într-o factură emisă. Pentru fiecare factură vor fi atâtea linii câte produse/servicii au fost consemnate la vânzarea respectivă. Atribute:

- NrFact – numărul facturii;
- Linie – numărul liniei din factură respectivă;
- CodPr – codul produsului/serviciului vândut;
- Cantitate – cantitatea vândută;
- PretUnit – prețul unitar (fără TVA) la care s-a făcut vânzarea.

Cheia primară este combinația NrFact+Linie. NrFact și CodPr sunt chei străine. Pentru a determina valoarea de incasat a unei facturi (inclusiv TVA), la valoarea fără TVA pentru fiecare linie (obținută prin produsul Cantitate \* PretUnit) trebuie adăugată TVA colectată, obținută prin aplicarea procentului de TVA al produsului/serviciului (atributul ProcTVA din PRODUSE) la valoarea fără TVA.

Tabela INCASARI (figura 1.23) reprezintă un nomenclator al incasărilor. Printr-o incasare, un client își stinge una sau mai multe obligații de plată, adică achită una sau mai multe facturi. Documentul primar pe baza căruia se consemnează incasarea poate fi ordinul de plată, cecul, chitanța etc. Atributele tăbelei sunt:

| LINIIFACT |       |       |           |          |
|-----------|-------|-------|-----------|----------|
| NrFact    | Linie | CodPr | Cantitate | PretUnit |
| 1111      | 1     | 1     | 50        | 10000    |
| 1111      | 2     | 2     | 75        | 10500    |
| 1111      | 3     | 5     | 500       | 6500     |
| 1112      | 1     | 2     | 80        | 10300    |
| 1112      | 2     | 3     | 40        | 7500     |
| 1113      | 1     | 2     | 100       | 9750     |
| 1114      | 1     | 2     | 70        | 10700    |
| 1114      | 2     | 4     | 30        | 15800    |
| 1114      | 3     | 5     | 700       | 6400     |
| 1115      | 1     | 2     | 150       | 9250     |
| 1116      | 1     | 2     | 125       | 9300     |
| 1117      | 1     | 2     | 100       | 10000    |
| 1117      | 2     | 1     | 100       | 9500     |
| 1118      | 1     | 2     | 30        | 11000    |
| 1118      | 2     | 1     | 150       | 9300     |
| 1119      | 1     | 2     | 35        | 10900    |
| 1119      | 2     | 3     | 40        | 7000     |
| 1119      | 3     | 4     | 50        | 14000    |
| 1119      | 4     | 5     | 750       | 6300     |
| 1120      | 1     | 2     | 80        | 11200    |
| 1121      | 1     | 5     | 550       | 6400     |
| 1121      | 2     | 2     | 100       | 10500    |

Figura 1.22. Tabela LINIIFACT

- CodInc – codul incasării este un număr intern, util pentru a diferenția o incasare de celelalte;
- DataInc – data incasării – data la care banii au intrat în contul sau casieria firmei;
- CodDoc – codul documentului justificativ al incasării: OP – ordin de plată, CHIT – chitanță, CEC – filă cec;
- NrDoc – numărul documentului justificativ;
- DataDoc – data la care a fost întocmit documentul justificativ; din momentul întocmirii documentului de plată până la data la care banii ajung efectiv în cont/casierie trec câteva zile sau săptămâni (datorită circuitului documentelor între firme și bănci).

Cheia primară este atributul CodInc. Ca restricții-utilizator pot fi instituite:

- data incasării nu poate precedea pe cea a întocmirii documentului (DataDoc <= DataInc);

Tabela **JUDEȚE** (figura 1.13) conține informații generale despre județele în care sunt clienți. Fiecare linie a tabelei descrie un județ. Atributele sunt:

- **Jud** – indicativul auto al județului (alcătuit din două litere, majuscule);
- **Judet** – denumirea județului;
- **Regiune** – regiunea istorică (provincia) din care face parte județul.

| JUDEȚE |         |         |
|--------|---------|---------|
| Jud    | Judet   | Regiune |
| IS     | Iasi    | Moldova |
| VN     | Vrancea | Moldova |
| NT     | Neamt   | Moldova |
| SV     | Suceava | Moldova |
| VS     | Vaslui  | Moldova |
| TM     | Timis   | Banat   |

Figura 1.13. Tabela JUDEȚE

Cheia primară este atributul **Jud**. Atributul **Judet** este cheie alternativă. Pot fi instituite o serie de restricții-utilizator:

- Jud este alcătuit numai din majuscule (eventual un spațiu, pentru București);
- Fiecare cuvânt din Judet începe cu majusculă; restul literelor sunt mici;
- Regiune începe cu majusculă; restul literelor sunt mici;
- Regiune poate avea numai una din valorile: Banat, Dobrogea, Muntenia, Oltenia, Transilvania, Moldova.

Tabela **LOCALITĂȚI** (figura 1.14) conține câte o linie pentru fiecare oraș sau comună. Atenție: satele nu sunt preluate în această tabelă, deoarece, într-o comună, fiecare sat component are același cod poștal ca și comuna. Atribute:

- **CodPost** – codul poștal al comunei sau orașului;
- **Loc** – denumirea comunei/orașului;
- **Jud** – indicativul auto al județului.

| LOCALITĂȚI |           |     |
|------------|-----------|-----|
| CodPost    | Loc       | Jud |
| 6600       | Iasi      | IS  |
| 5725       | Pascani   | IS  |
| 6500       | Vaslui    | VS  |
| 5300       | Focșani   | VN  |
| 6400       | Birlad    | VS  |
| 5800       | Suceava   | SV  |
| 5550       | Roman     | NT  |
| 1900       | Timisoara | TM  |

Figura 1.14. Tabela LOCALITĂȚI

Cheia primară este atributul **CodPost**. Atributul **Jud** este cheie străină, tabela-părinte fiind **JUDEȚE** (prin atributul **Jud**). Pot fi instituite o serie de restricții-utilizator:

- Jud este alcătuit numai din majuscule (eventual un spațiu, pentru București);
- Fiecare cuvânt din Localitate începe cu majusculă; restul literelor sunt mici.

Tabela **CLIENTI** (figura 1.15) grupează date generale ale clientilor firmei pentru care s-a constituit baze de date (o linie – un client). Atribute:

- **CodCl** – codul clientului;
- **DenCl** – denumirea clientului (persoană juridică);
- **CodFiscal** – codul fiscal;
- **Adresa** – adresa sediului firmei-client;
- **CodPost** – codul poștal al comunei sau orașului;
- **Telefon** – telefonul (principal) al clientului.

| CLIENTI |              |           |                            |         |            |
|---------|--------------|-----------|----------------------------|---------|------------|
| CodCl   | DenCl        | CodFiscal | Adresa                     | CodPost | Telefon    |
| 1001    | Client 1 SRL | R1001     | Tranzitiei, 13 bis         | 6600    | NULL       |
| 1002    | Client 2 SA  | R1002     | NULL                       | 6600    | 032-212121 |
| 1003    | Client 3 SRL | R1003     | Prosperitatii, 22          | 6500    | 035-222222 |
| 1004    | Client 4     | NULL      | Sapientei, 56              | 5725    | NULL       |
| 1005    | Client 5 SRL | R1005     | NULL                       | 1900    | 056-111111 |
| 1006    | Client 6 SA  | R1006     | Pacientei, 33              | 5550    | NULL       |
| 1007    | Client 7 SRL | R1007     | Victoria Capitalismului, 2 | 1900    | 056-121212 |

Figura 1.15. Tabela CLIENTI

Cheia primară este atributul **CodCl**. Atributul **CodPost** este cheie străină, tabela-părinte fiind **LOCALITĂȚI** (prin atributul **CodPost**). **DenCl** și **Adresa** încep cu majuscule. După cum punctam anterior, valorile nule indică o lipsă de informație. Pentru primul client nu se cunoaște telefonul, celui de-al doilea adresa etc.

Tabela **PERSOANE** (figura 1.16) are ca obiectiv stocarea datelor despre persoanele-cheie de la firmele-client: directori generali, directori financiari, șefi ai compartimentelor comerciale (aprovisionare și/sau vânzări) etc. Probabil că vi se pare ciudat, dar ceea ce intenționăm cu această tabelă (și următoarea) este să sprinjnim *fidelizarea* clientului. Nu costă (mai) nimic dacă de Sfântul Ioan se trimită căte o felicitare tuturor Ionilor și Ioanelor din partea firmei noastre. Iar pentru ca lucrurile să fie și mai bine puse la punct, ar fi trebuit să preluăm și informații precum: data nașterii, starea civilă, numele și vîrstă copiilor, pasiuni în materie de muzică, arte pastice, literatură, sport etc. Schema luată în considerare s-a oprit la următoarele atrbute:

- **CNP** – codul numeric al persoanei;
- **Nume**;
- **Prenume**;
- **Adresa**;
- **Sex**;
- **CodPost**;
- **TelAcasă** – numărul telefonului de acasă;
- **TelBirou** – numărul telefonului (fix) de la birou;
- **TelMobil** – numărul „mobilului”;
- **Email** – adresa e-mail.

## 1.5. Schema și conținutul unei baze de date.

### Exemplu

Există două aspecte complementare de abordare a bazelor de date relaționale: *schema* (structura, intensia) și *conținutul* (instantierea, extensia)<sup>23</sup>.

**Conținutul unei relații este reprezentat de ansamblul tuplurilor ce o alcătuiesc la un moment dat.** Pe parcursul exploatarii bazei, conținutul poate crește exponențial, în funcție de volumul și complexitatea operațiunilor consumante.

**O schema relațională poate fi definită ca un ansamblu de relații asociate semantic prin domeniul lor de definiție și prin restricții de integritate<sup>24</sup>.** Este independentă de timp și reprezintă componenta permanentă a relațiilor.

După C.J. Date, schema bazei de date (intensia) cuprinde<sup>25</sup>:

- **Structura titulaturilor** alcătuitură din numele relațiilor și cele ale atributelor (fiecare atribut fiind asociat domeniului său) și
- **Restricțiile de integritate**, care sunt de trei feluri:

- a) **Restricțiile cheilor primare.** După cum a fost prezentat, fiecare cheie primară este supusă restricțiilor de unicitate, compozitie minimală și valori nenele.
- b) **Restricții referențiale**, care decurg din existența cheilor strâine.
- c) **Alte restricții.** În această categorie sunt incluse restricțiile definite de utilizator (de comportament), dependențele dintre atrbute etc.

De multe ori este suficientă cunoașterea numai a schemei simplificate. **Schema simplificată a unei baze de date relaționale cuprinde numele tabelelor și enumerarea atributelor acestora, atrbutele-chei primare fiind subliniate.**

În capitolele care vor urma, va fi utilizată cu precădere o bază de date „martor” sau, mai bine spus, „cobiai”, denumită VÂNZĂRI, ce conține următoarele tabele:

```

JUDETE {Jud, Judet, Regiune}
LOCALITATI {CodPost, Loc, Jud}
PERSOANE{CNP, Nume, Prenume, Adresa, Sex, CodPost, TelAcasa,
 TelBirou, TelMobil, EMail}
CLIENTI{CodCl, DenCl, CodFiscal, Adresa, CodPost, Telefon}
PERSCLIENTI{CNP, CodCl, Functie}
PRODUSE{CodPr, DenPr, UM, Grupa, ProcTVA}
FACTURI{NrFact, DataFact, CodCl, Obs}
LINIIFACT{NrFact, Linie, CodPr, Cantitate, PretUnit}
INCASARI{CodInc, DataInc, CodDoc, NrDoc, DataDoc}
INCASFACt{CodInc, NrFact, Transa}

```

23. În multe dintre lucrările redactate în limba engleză se folosesc noțiunile de „intension” și „extension” (spre exemplu, vezi [Date96]). De aici traducerea „intensie” și „extensie” din [Lungu §.a. 95].

24. [Saleh94], p. 29.

25. [Date86], pp. 90-91.

Discuția detaliată despre fiecare o amână peste numai câteva zeci de rânduri, când, pentru un plus de claritate, va fi prezentat și conținutul fiecăreia.

Schema unei baze de date poate fi reprezentată grafic, avantajul principal fiind o mai bună sugestivitate. Iată câteva reguli pentru reprezentarea grafică a unei BDR<sup>26</sup>:

- O tabelă se reprezintă pe două linii, pe prima fiind scris numai numele relației, iar pe cea de-a doua numele atributelor.
- Ordinea prezentării atributelor nu are importanță. Totuși, în scopul ușurării înțelegerei schemei, coloanele care desemnează cheia primară se dispun succesiv (bineînteles, atunci când sunt mai multe). La fel și coloanele care constituie o cheie străină. În general, cheia primară este plasată la marginea stângă a tăbelei (prima sau primele coloane).
- Numele coloanelor ce alcătuiesc cheia primară se subliniază cu o linie continuă, iar cele care alcătuiesc identificatorii secundari se subliniază cu linie punctată.
- Numele unei coloane facultative este scris între paranteze.
- Dacă o cheie străină este alcătuire din mai multe coloane, se utilizează acolada pentru a le grupa.
- O restricție referențială este reprezentată printr-o săgeată care pleacă de la numele coloanei de referință (sau de la vârful acoladei, în cazul unui grup de referință) și are vârful în atrbutele tăbelei la care se face referință (tabela-pârinte).

Pentru baza de date VÂNZĂRI, reprezentarea grafică a schemei simplificate este cea din figura 1.12.

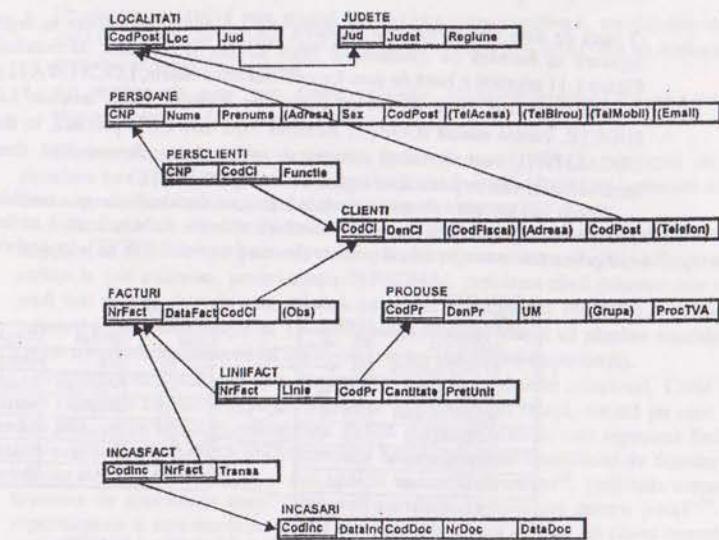


Figura 1.12. Schema simplificată a bazei de date VÂNZĂRI

„cutii Whiskas 10000”. Autorul afirmă, pe bună dreptate, că la nivelul conceptual la care se înregistrează informațiile, nu există nici o valoare prin care cele patru rânduri să fie diferențiate. Astfel încât, pentru a respecta canoanele modelului relațional, este necesară introducerea unei coloane suplimentare fără prea mare relevanță, dar care să asigure unicitatea fiecărui tuplu.

Este adevarat că E.F. Codd a propus ulterior consiliului ANSI X3H2 o funcție specială care să măsoare gradul de repetabilitate a valorii unuia sau mai multor atribute, funcție care ar semăna întru câtva cu funcția COUNT() din SQL folosită împreună cu GROUP BY<sup>20</sup>. Dacă am privi cazul celor patru conserve Whiskas prin prisma celor trei abordări, SQL le privește ca patru cutii separate, dar echivalente, modelul Codd (original)/Date le consideră o singură clasă de cutii, iar pentru ultimul model al lui Codd avem de-a face cu o colecție de patru cutii.

Pe lângă noțiunile cheie candidată, cheie primară, cheie alternativă, modelul relațional utilizează și sintagma *supercheie*. După Celko<sup>21</sup>, o supercheie poate fi definită ca un set de coloane ce îndeplinește condiția de cheie (unicitate), dar care conține cel puțin un subset care este el însuși cheie. Cu alte cuvinte, din cele trei condiții ale cheii primare, o supercheie o îndeplinește numai pe prima (unicitate), fără a avea însă compozitia minimă și fără a pune problema restricției de entitate (valorile nule ale atributelor componente).

### Restricția referențială

O bază de date relațională este alcătuită din relații (tabele) aflate în legătură. Stabilirea legăturii se bazează pe mecanismul cheii străine și, implicit, al restricției referențiale. Figura 1.11 prezintă o bază de date formată din două tabele, LOCALITĂȚI și JUDEȚE.

Atributul Jud joacă un rol de „agent de legătură” între tabelele LOCALITĂȚI și JUDEȚE. Pentru relația JUDEȚE, atributul Jud este cheie primară, în timp ce în tabela LOCALITĂȚI, Jud reprezintă coloana de referință sau cheie străină, deoarece numai pe baza valorilor sale se poate face legătură cu relația JUDEȚE.

Cheile străine sau coloanele de referință sunt deci atrbute sau combinații de atrbute care pun în legătură linii (tupluri) din relații diferite. Tabela în care atrbutul de legătură este primară se numește *tabelă-părinte* (în cazul nostru JUDEȚE), iar cealaltă *tabelă-copil*.

| LOCALITĂȚI |           |      | JUDEȚE |         |         |
|------------|-----------|------|--------|---------|---------|
| CodPost    | Loc       | Jud  | Jud    | Judet   | Regiune |
| 6600       | Iasi      | IS   | IS     | Iasi    | Moldova |
| 5725       | Pascani   | IS   | IS     | Vrancea | Moldova |
| 6500       | Vaslui    | VS   | VN     | Neamt   | Moldova |
| 5300       | Focșani   | VN   | NT     | Suceava | Moldova |
| 6400       | Birlad    | VS   | SV     | Vaslui  | Moldova |
| 5800       | Suceava   | SV   | VS     | Timis   | Banat   |
| 5550       | Roman     | NULL | TM     |         |         |
| 1900       | Timisoara | TM   |        |         |         |

Figura 1.11. Mecanismul de legare a tabelelor

20. Vezi [Celko99], p. 49.

21. [Celko99], p. 52.

Legat de noțiunea de cheie străină apare conceptul de *restricție referențială*. O restricție de integritate referențială apare atunci când o relație face referință la o altă relație. C.J. Date definește restricția de integritate referențială astfel<sup>22</sup>:

*Fie D un domeniu primar și R1 o relație ce prezintă un atrbut A definit pe D – se poate scrie R1(..., A:D, ...). În orice moment, orice valoare a lui A în R1 trebuie să fie:*

- a) ori nulă,
- b) ori egală cu o valoare a lui V, unde V este cheie primară sau candidată într-un tuplu al unei relații oarecare R2 (R1 și R2 nu trebuie să fie neapărat distinse), cheie primară definită pe același domeniu primar D – R2(V:D, ...).

Într-o altă formulare, se iau în discuție două tabele (relații) T1 și T2. Ambele prezintă atrbutul sau grupul de atrbute notat CH, care pentru T1 este cheie primară, iar pentru T2 cheie străină. Dacă în T2 se interzice apariția de valori nenele ale CH care nu există în nici un tuplu din T1, se spune că între T2 și T1 s-a instituit o restricție referențială.

Instituirea restricției referențiale între tabela JUDEȚE (părinte) și LOCALITĂȚI (copil) permite cunoașterea, pentru fiecare localitate, a denumirii județului și a regiunii din care face parte. Dacă în LOCALITĂȚI ar exista vreo linie în care valoarea atrbutului Jud ar fi, spre exemplu, XY, este clar că acea linie ar fi orfană (nu ar avea linie corespondentă în tabela-părinte JUDEȚE).

### Observații

1. Pentru mulți utilizatori și profesioniști ai bazelor de date, denumirea de „relațional” desemnează faptul că o bază de date este alcătuită din tabele puse în legătură prin intermediul cheilor străine. Aceasta este, de fapt, o doar acceptiune a termenului de BDR, prima, cea „clasică”, având în vedere percepția fiecărei linii dintr-o tabelă ca o relație între clase de valori.
2. Majoritatea SGBD-urilor prezintă mecanisme de declarare și gestionare automată a restricțiilor referențiale, prin actualizări în cascadă și interzicerea valorilor care ar încâlcea aceste restricții.
3. Respectarea restricțiilor referențiale este una dintre cele mai complicate sarcini pentru dezvoltatorii de aplicații ce utilizează baze de date. Din acest punct de vedere, tentația este de a „sparge” baza de date în căi mai puține tabele cu putință, altfel spus, de a avea relații căt mai „corpolente”. Gradul de fragmentare al bazei fine de normalizarea bazei de date, care, ca parte a procesului de proiectare a BD, se bazează pe dependențele funcționale, multivaloare și de joncțiune între atrbute.

### Restricții-utilizator

Restricțiile-utilizator mai sunt denumite și restricții de comportament sau restricții ale organizației. De obicei aceste restricții iau forma unor reguli de validare la nivel de atrbut, la nivel de linie/tabelă sau al unor reguli implementate prin declanșătoare (triggers). Spre exemplu, se poate institui o regulă care interzice emiterii unei noi facturi (o nouă vânzare) dacă datorile firmei-client sunt mai mari de 2.000.000.000 lei, iar directorul acesteia nu este membru în partidul de guvernământ sau coalitia de guvernare.

22. [Date86], p. 89.

| BIBLIOTECA  |                  |                        |                              |         |      |                                            |
|-------------|------------------|------------------------|------------------------------|---------|------|--------------------------------------------|
| ISBN        | Titlu            | Cote                   | Autori                       | Editura | An   | CuvinteCheie                               |
| 973-333-566 | Instrumente CASE | III 13421<br>III 13422 | Oprea<br>Dumitriu<br>Meșniță | Alfa    | 1998 | Analiză SI<br>Proiectare SI<br>CASE<br>DFD |
| ...         |                  |                        |                              |         |      |                                            |

Figura 1.10. Relație în care există valori ne-atomice

### Restricția de unicitate

Într-o relație nu pot exista două linii identice (două linii care prezintă aceleași valori pentru toate atributele). Mai mult, majoritatea relațiilor prezintă un atribut, sau o combinație de atribut, care diferențiază cu siguranță un tuplu de toate celelalte tupluri ale relației.

**Cheia primară a unei relații (tabele) este un atribut sau un grup de atribuție care identifică fără ambiguitate fiecare tuplu (linie) al relației (tabeli).**

După Date, există trei restricții pe care trebuie să le verifice cheia primară:

1. **Unicitate:** o cheie identifică un singur tuplu (linie) al relației.
2. **Compoziție minimală:** atunci când cheia primară este compusă, nici un atribut din cheie nu poate fi eliminat fără distrugerea unicității tuplului în cadrul relației. În cazuri-limită, o cheie poate fi alcătuire din toate atributele relației.
3. **Valori non-nule:** valorile atributului (sau ale ansamblului de atrbute) ce desemnează cheia primară sunt întotdeauna specificate, deci nenule. Mai mult, nici un atribut din compoziția cheii primare nu poate avea valori nule. Această a treia condiție se mai numește și **restricție a entității**.

**Domeniul unui atribut care este cheie primară într-o relație este denumit domeniu primar. Dacă într-o relație există mai multe combinații de atrbute care conferă unicitate tuplului, acestea sunt denumite chei candidate. O cheie candidată care nu este identificator primar este cunoscută și sub numele de cheie alternativă.**

Celko inventariază patru proprietăți dezirabile pentru o cheie<sup>17</sup>:

- Familiaritate: valorile cheii să fie ușor de înțeles pentru utilizatori.
- Stabilitate: valorile cheii nu trebuie să fie volatile.
- Minimalitate.
- Simplitate: sunt de preferat chei scurte și simple.

Tot el atrage atenția asupra faptului că cele patru proprietăți pot intra uneori în conflict. O cheie stabilă poate să se dovedească complexă și dificil de gestionat. Identificarea cheii primare pentru o relație face parte (împreună cu stabilirea tabelelor prin gruparea atrbutelor, stabilirea celorlalte restricții) din procesul de proiectare a bazei de date. Uneori, precum în cazul tabelei JUDEȚE, stabilirea cheii primare nu ridică probleme. Cum fiecare județ are un indicativ auto unic, rezultă că atributul Jud candidează la postul de cheie a relației. La fel se poate spune și despre denumirea judejului. Rezultă că relația JUDEȚE

17. [Celko99], p. 247.

rezintă două chei candidate: Jud și Județ. Alegerea uneia dintre cheile candidate are în vedere criterii precum lungimea, ușurința în reținere și/sau editare. Fiind mai scurt, desemnăm atributul Jud drept cheie primară, situație în care Județ devine cheie alternativă.

Există suficiente cazuri în care cheia primară este compusă din două, trei și.a.m.d. sau, la extrem, toate atrbutele relației. Să luăm spre analiză o relație PERSONAL care conține date generale despre angajații firmei. Fiecare tuplu al relației se referă la un angajat, atrbutele fiind: Nume, Prenume, DataNașterii, Vechime, SalariuTarifar.

- Atributul Nume nu poate fi cheie, deoarece chiar și intr-o întreprindere de talie mijlocie, este posibil să existe doi angajați cu același nume.
- Dacă apariția a două persoane cu nume identice este posibilă, atunci apariția a două persoane cu același Prenume este probabilă.
- Nici unul dintre atrbutele DataNașterii, Vechime, SalariuTarifar nu poate fi „înregistrat” cu funcția de identificator.
- În acest caz, se încearcă gruparea a două, trei, patru și.a.m.d. atrbute, până când se obține combinația care va permite diferențierea clară a oricăriei linii de toate celelalte.
- Combinăția Nume+Adresă pare, la prima vedere, a îndeplini „cerințele” de identificator. Ar fi totuși o problemă: dacă în aceeași firmă (organizație) lucrează împreună soțul și soția? Ambii au, de obicei, același nume de familie și, tot de obicei, același domiciliu. Este adevărat, cazul ales nu este prea fericit. Dar este suficient pentru „compromiterea” combinației.
- Următoarea tentativă este grupul Nume+Prenume+Adresă, combinăție neoperantă dacă în organizație lucrează tată și un fiu (sau mama și o fiică) care au același nume și prenume și domiciliu comun.
- Ar rămâne de ales una dintre soluțiile: Nume+Prenume+Adresă+Vechime, Nume+Prenume+Adresă+DataNașterii.

Oricare dintre cele două combinații prezintă riscul violării restricției de entitate, deoarece este posibil ca, la preluarea unui angajat în bază, să nu î se cunoască adresa sau data nașterii, caz în care atrbutul respectiv ar avea valoarea NULL.

Dificultățile de identificare fără ambiguitate a angajaților au determinat firmele ca, la angajare, să aloce fiecărei persoane un număr unic, denumit Marcă. Prin adăugarea acestui atrbut la cele existente, pentru relația PERSONAL problema cheii primare este rezolvată mult mai simplu. Actualmente, sarcina este simplificată și prin utilizarea codului numeric personal (CNP), combinăție de 13 cifre care prezintă avantajul că rămâne neschimbător pe tot parcursul vieții persoanei (și după, dar asta are mai puțină importanță).

Problema unicității este una discutabilă. Clasicii modelului relațional, Codd și Date, s-au pronunțat împotriva repetării tuplurilor în cadrul unei relații, cerință pe care standardele SQL nu au luat-o în considerare. Există și situații reale în care repetarea linii este inevitabilă. David Beech a prezentat într-o lucrare înaintată Consiliului de Standardizare a bazelor de date ANSI X3H2, și mai apoi în revista Datamation<sup>18</sup>, problema consacrată în literatura de specialitate drept „argumentul cutiilor cu mâncare pentru pisici”<sup>19</sup>: într-un supermagazin în care casele de marcat sunt legate la o bază de date, un client cumpără patru cutii Whiskas pentru pisici care costă, fiecare (cutie), 10.000 lei; pe bon apare de patru ori

18. Beech, D. – „New Life for SQL”, Datamation, 1 februarie 1989.

19. [Celko99], p. 45.

Valoarea NULL este diferită însă (cel puțin teoretic) de valorile 0 sau spațiu. Importanța este majoră în expresii și funcții, după cum vom vedea în capitoalele consacrate limbajului SQL.

În încheiere, principalele caracteristici ale unei relații sunt sistematizate după cum urmează:

- În cadrul unei baze de date, o relație prezintă un nume distinct de al celorlalte relații.
- Valoarea unui atribut într-un tuplu oarecare conține o singură valoare (o valoare atomică).
- Fiecare atribut are un nume distinct.
- Orice valoare a unui atribut face parte din domeniul pe care a fost definit acesta.
- Ordinea disponirii atributelor în relație nu prezintă importanță.
- Fiecare tuplu este distinct, adică nu pot exista două tupluri identice.
- Ordinea tuplurilor nu influențează conținutul informațional al relației.

Facem astfel trecerea la următorul paragraf, dedicat restricțiilor unei baze de date relationale.

#### 1.4. Restricții ale bazei de date

Există diverse moduri de abordare și clasificare a restricțiilor ce pot fi instituite într-o bază de date. În cele ce urmează vor fi prezentate: restricția de domeniu, de atomicitate, de unicitate, referențială și restricțiile-utilizator.

##### Restricția de domeniu

După cum am văzut în paragraful anterior, un atribut este definit printr-un nume și un domeniu. Orice valoare a atributului trebuie să se încadreze în domeniul definit. Există mai multe moduri de percepție a acestei restricții.

O parte dintre informaticieni substituie domeniul tipului atributului: numeric, și de caractere, dată calendaristică, logic (boolean) etc. și, eventual, lungimii (numărul maxim de poziții/carcătere pe care se poate „întinde” un atribut). După cum se observă, este luat în calcul numai aspectul sintactic al domeniului. Faptul că indicativul unui județ poate fi una dintre valorile: IS, TM, B etc. reprezintă o *restricție de comportament* sau, mai simplu, o *restricție definită de utilizator*.

Cea de-a doua categorie privește domeniul deopotrivă sintactic și semantic. Astfel, domeniul sintactic al atributului JUD (indicativul județului) este un sir de două caractere, obligatoriu litere (sau o literă și un spațiu, pentru București), și, chiar mai restrictiv, literele sunt obligatoriu majuscule. Din punct de vedere semantic, indicativul poate lua una dintre valorile: IS, TM...

Majoritatea SGBD-urilor permit definirea tuturor elementelor ce caracterizează domeniul (sintactic și semantic) atributului JUD prin declararea tipului și lungimii atributului și prin așa-numitele reguli de validare la nivel de câmp (*field validation rule*). Sunt însă și produse la care domeniul poate fi definit explicit, sintactic și semantic, dându-i-se nume la care vor fi legate atributele în momentul creării tabelelor.

##### Atomicitate

Conform teoriei bazelor de date relaționale, orice atribut al unei tabele oarecare trebuie să fie *atomic*, în sensul imposibilității descompunerii sale în alte atribut. Implicit, toate domeniile unei baze de date sunt măsuai atomice (adică elementare). În aceste condiții, se spune că baza de date se află în *prima formă normală* sau *prima formă normalizată* (1NF)<sup>16</sup>.

Astăzi, atomicitatea valorii atributelor a devenit o „țintă predilectă a „atacurilor dușmanoase” la adresa modelului relațional, datorită imposibilității înglobării unor structuri de date mai complexe, specifice unor domenii ca: proiectare asistată de calculator, baze de date multimedia etc. Mulți autori, dintre care merită amintiți cu deosebire Chris J. Date și Hugh Darwen, se opun ideii de atomicitate formulată de Codd.

Prinvele vizate de „stigmatul” neatomicității sunt atributele compuse. Un exemplu de atribut compus (non-atomic) este Adresa. Fiind alcătuitor din Stradă, Număr, Bloc, Scără, Etaj, Apartament, discuția despre atomicitatea adresei pare de prisos, iar descompunerea sa imperativă. Trebuie să ne raportăm însă la obiectivele bazei. Fără îndoială că pentru BD a unei filiale CONEL sau ROMTELECOM, sau pentru poliție, preluarea separată a fiecărui element constituent al adresei este foarte importantă. Pentru un importator direct însă, pentru un mare en-grosist sau pentru o firmă de producție lucrurile stau într-o cu totul altă lumină. Partenerii de afaceri ai acestora sunt persoane juridice, iar adresa interesează numai la nivel general, caz în care atributul Adresa nu este considerat a fi non-atomic.

Alte exemple de atribut ce pot fi considerate, în funcție de circumstanțe, simple sau compuse: DataOperăriiBancare (Data+Ora), BuletinIdentitate (Seria+Număr), NrInmatricularAuto (privit global sau pe cele trei componente: număr, județ, combinație trei de litere).

O relație (tabelă) în 1NF nu trebuie să conțină atributе care se repetă ca grupuri (*grupuri repetitive*). Într-o altă formulare, toate liniile unei tabele trebuie să conțină același număr de atributе. Fiecare celulă a tabelei (intersecția unei coloane cu o linie) – altfel spus, valoarea unui atribut pe o linie (înregistrare) – trebuie să fie atomică.

După [Connolly și a. 96], un *grup repetitive* este un atribut sau grup de atributе dintr-o tabelă care apare cu valori multiple pentru o singură apariție a cheii primare a tabelei nenormalizate.

Să luăm exemplul tabelei BIBLIOTECĂ din figura 1.10. Tabela gestionează informații despre cărțile existente în depozitul bibliotecii facultății X. Depozitul conține două exemplare ale cărții cu ISBN 973-333-566. Cartea a fost scrisă de trei autori și li sunt asociate patru cuvinte-cheie. Relația nenormalizată conține trei grupuri repetitive: Cote, Autori și CuvinteCheie.

16. Pentru o discuție *in extenso* a atomicității și a primei forme normalizate, vezi [Fotache2000-1].

După cum vom vedea într-un alt capitol, algebra relațională (și SQL-ul) utilizează conceptual de domeniu compatibil pentru operatorii reuniune, intersecție și diferență.

Dacă notăm cu  $D_1$  domeniul atributului Jud, cu  $D_2$  domeniul atributului Județ și cu  $D_3$  domeniul pentru Regiune, se poate spune că fiecare linie a tabelii JUDEȚE este un tuplu de patru elemente:

$$(v_1, v_2, v_3), \text{ unde } v_1 \in D_1, v_2 \in D_2 \text{ și } v_3 \in D_3,$$

Relația JUDEȚE corespunde unui subansamblu din ansamblul tuturor tuplurilor posibile alcătuite din patru elemente, ansamblu care este produsul cartezian

$$\prod_{i=1}^n D_i$$

În general, orice relație  $R$  poate fi definită ca un subansamblu al produsului cartezian de  $n$  domenii  $D_i$ :

$$R \subseteq D_1 \times D_2 \times D_3 \times \dots \times D_n,$$

$n$  fiind denumit *gradul* sau *ordinul* relației. Relațiile de grad 1 sunt *unare*, cele de grad 2 – *binare*, ..., cele de grad  $n$  – *n-are*. Aceasta definiție pune în evidență aspectul constant al relației, de independență în timp (se spune că în acest caz relația este definită ca un *predicat*).

O a doua definiție abordează o relație  $R$  ca un *ansamblu* de  $m$ -uplete ( $m$ -tupluri) de valori:

$$R = \{t_1, t_2, \dots, t_k, \dots, t_m\}, \text{ unde } t_i = (d_{i1}, d_{i2}, \dots, d_{in}),$$

în care:

$d_{i1}$  este o valoare în  $D_1$ ,  $d_{i2}$  este o valoare în  $D_2$ , ...,  $d_{in}$  este o valoare în  $D_n$ ;

$n$  – reprezintă *ordinul* lui  $R$ ;

$m$  – *cardinalitatea* lui  $R$ .

Pentru un plus de claritate, vezi figura 1.8.

Reprezentarea sub formă de tabelă, deci ca ansamblu de tupluri, pune în evidență aspectul dinamic, variabil al relației. Abordarea predicativă (prima definiție) sau asamblistă (a doua definiție) reprezintă un criteriu important de delimitare a limbajelor de interogare a bazelor de date relaționale.

Așând în vedere cele două definiții ale relației, se poate reformula conceptual de *atribut*, ca fiind elementul care determină *ansamblul de valori luate de fiecare domeniu al relației*<sup>15</sup>. Precizarea fără ambiguitate a rolului jucat de fiecare domeniu în cadrul relației impune ca numele atributelor să fie diferite. O relație poate fi simbolizată și astfel:

$$R(A_1, A_2, \dots, A_n) \text{ sau } (A_1 : D_1, A_2 : D_2, \dots, A_n : D_n),$$

unde  $A_i$  sunt atributele relației.

15. [Salch94], p. 29.

| R     | $D_1$    | $D_2$    | $D_3$    | ... | $D_l$    | ... | $D_n$    |
|-------|----------|----------|----------|-----|----------|-----|----------|
| $t_1$ | $d_{11}$ | $d_{12}$ | $d_{13}$ | ... | $d_{1l}$ | ... | $d_{1n}$ |
| $t_2$ | $d_{21}$ | $d_{22}$ | $d_{23}$ | ... | $d_{2l}$ | ... | $d_{2n}$ |
| $t_3$ | $d_{31}$ | $d_{32}$ | $d_{33}$ | ... | $d_{3l}$ | ... | $d_{3n}$ |
| ...   | ...      | ...      | ...      | ... | ...      | ... | ...      |
| $t_k$ | $d_{k1}$ | $d_{k2}$ | $d_{k3}$ | ... | $d_{kl}$ | ... | $d_{kn}$ |
| ...   | ...      | ...      | ...      | ... | ...      | ... | ...      |
| $t_m$ | $d_{m1}$ | $d_{m2}$ | $d_{m3}$ | ... | $d_{ml}$ | ... | $d_{mn}$ |

Figura 1.8. Ilustrarea celei de-a doua definiții a unei relații

Reținem corespondența noțiunilor *relație-tabelă*, *tuplu-linie* și *atribut-colonă*.

Numele de tabele pe care le conține o bază de date, atributele „adunate” în fiecare tabelă, domeniul fiecărui dintre atrbute prezintă diferențe majore de la o bază la alta, chiar dacă uneori reflectă același tip de procese. Întrăm astfel în sfera proiectării bazelor de date, a dependențelor și normalizării.

Înainte de finalul paragrafului, să examinăm o altă tabelă, CLIENTI, prezentată în figura 1.9.

| CodCl | DenCl        | CodFiscal | Adresa                     | CodPost | Telefon    |
|-------|--------------|-----------|----------------------------|---------|------------|
| 1001  | Client 1 SRL | R1001     | Tranzitiei, 13 bis         | 6600    | NULL       |
| 1002  | Client 2 SA  | R1002     | NULL                       | 6600    | 032-212121 |
| 1003  | Client 3 SRL | R1003     | Prosperității, 22          | 6500    | 035-222222 |
| 1004  | Client 4     | NULL      | Sapientei, 56              | 5725    | NULL       |
| 1005  | Client 5 SRL | R1005     | NULL                       | 1900    | 056-111111 |
| 1006  | Client 6 SA  | R1006     | Pacientei, 33              | 5550    | NULL       |
| 1007  | Client 7 SRL | R1007     | Victoria Capitalismului, 2 | 1900    | 056-121212 |

Figura 1.9. Valori NULLe într-o relație

Relația conține informații despre clienții firmei (firma este cea pentru care s-a proiectat, creat și implementat baza de date). Fiecare linie se referă la un singur client (persoană juridică). Se observă că atrbutele CodFiscal, Adresa și Telefon conțin, pe alocuri, o valoare curioasă notată NULL. Valoarea NULL este considerată o metavaloare și indică faptul că, în acel loc, informația este necunoscută sau inaplicabilă. În tabela CLIENTI, Clientului 1 SRL (cod 1001) nu î se cunoaște telefonul, pentru Client 2 SA nu se cunoaște adresa, pentru Client 4 (cod 1004) nu se cunoște nici codul fiscal, nici telefonul etc.

Pentru cei deprinși să lucreze cu SGBD-uri sub MS-DOS, valorile respective sunt fie 0 (pentru atrbute numerice), fie ‘’ (pentru siruri de caractere), fiind extrase din tabele prin funcții precum EMPTY().

a sute, chiar mii de utilizatori la o aceeași bază de date. Dacă, tradițional, piața acestui segment era rezervată Unix-ului, în prezent Windows NT și Novell Netware fac față celor mai multe cerințe.

Se mai cuvine adăugat că eforturi substanțiale sunt canalizate pe linia unui mariaj al relaționalului de obiectual. Includerea de noi tipuri de date, extensiile ale nucleului relațional „tradițional” al bazelor de date către obiecte, suportul pentru Web și comerț electronic sunt doar câteva direcții pe care s-au înscris majoritatea SGBDR-urilor actuale pentru a-și conserva sau ameliora poziția de pe piață.

### 1.3. Relații/tabele, domenii și atrbute

La modul simplist, o bază de date relațională (BDR) poate fi definită ca un ansamblu de relații (tabele), fiecare tabelă (sau tabel), alcătuită din liniile (tupluri), are un nume unic și este stocată pe suport extern (de obicei disc). La intersecția unei liniilor cu o coloană se găsește o valoare atomică.

O relație conține informații omogene legate de anumite entități, procese, fenomene: CĂRTI, STUDENȚI, LOCALITĂȚI, PERSONAL, FACTURI etc. Spre exemplu, în figura 1.6 este reprezentată tabela JUDEȚE.

| JUDEȚE |         |         |
|--------|---------|---------|
| Jud    | Judet   | Regiune |
| IS     | Iasi    | Moldova |
| VN     | Vrancea | Moldova |
| NT     | Neamt   | Moldova |
| SV     | Suceava | Moldova |
| VS     | Vaslui  | Moldova |
| TM     | Timis   | Banat   |

Figura 1.6. Tabela JUDEȚE

În teoria relațională se folosește termenul *relație*. Practică, însă, a consacrat termenul *tabelă* (engl. *table*). Pentru introducerea în cititor a senzatiei de pragmatism, cu care a fost impregnată prezenta lucrare, în SQL vom folosi preponderent noțiunea de tabelă. Cei care au de gând să scrie o lucrare teoretică (inevitabil, științifică!) sunt consiliați să apeleze la celălalt termen.

Un tuplu sau o linie este o succesiune de valori de diferite tipuri. În general, o linie regroupează informații referitoare la un obiect, eveniment etc., altfel spus, informații referitoare la o entitate; o carte (un titlu sau un exemplar din depozitul unei biblioteci, în funcție de circumstanțe), un/o student(ă), o localitate (oraș sau comună), un angajat al

firmei, o factură emisă etc. Figura 1.7 conține al treilea tuplu din tabela JUDEȚE, tuplu referitor la județul Neamț.

|    |       |         |
|----|-------|---------|
| NT | Neamț | Moldova |
|----|-------|---------|

Figura 1.7. Un tuplu al tăbelei JUDEȚE

Linia de mai sus este alcătuită din trei valori ce desemnează indicativul (auto) al județului, numele județului și regiunea din care face parte (una dintre provinciile istorice), valori specifice județului Neamț.

Theoretic, orice tuplu reprezintă o *relație între clase de valori* (în cazul nostru, între trei clase de valori); de aici provine sintagma *baze de date relaționale*, în sensul matematic al relației, de asociere a două sau mai multe elemente.

Fără deosebire, toate tuplurile relației au același format (structură), ceea ce înseamnă că în tabela JUDEȚE, din care a fost extrasă linia din figura 1.7, fiecare linie este constituită dintr-o valoare a indicativului, o valoare a denumirii și o valoare a regiunii.

Ordinea tuplurilor nu prezintă importanță din punctul de vedere al conținutului informațional al tabeliei.

**Ansamblul valorilor de același tip corespunde unui atribut** al entităților, atribut ale căruia valori alcătuiesc una dintre coloanele tăbelei. Tabela din figura 1.6 prezintă trei coloane; prin urmare, se spune că relația JUDEȚE conține trei atribut: Jud, Judet și Regiune.

Fiecare atribut este caracterizat printr-un *nume* și un *domeniu de valori* pe care le poate lua. **Domeniul poate fi definit ca ansamblul valorilor acceptate (autorizate) pentru un element component al relației:**

- într-o tabelă destinată datelor generale ale angajaților, pentru atributul Sex, domeniul este alcătuit din două valori: Femeiesc și Bărbațesc;
- pentru atributul Regiune, domeniul, deși limitat, este ceva mai mare; valorile autorizate pot fi: Dobrogea, Banat, Transilvania, Oltenia, Muntenia, Moldova.
- domeniul atributului Jud este alcătuit din indicatele auto (două caractere) ale fiecărui județ (plus București).
- domeniul unui atribut precum PrețUnitar, care se referă la prețul la care a fost vândut un produs/serviciu, este cu mult mai larg, fiind alcătuit din orice valoare cuprinsă între 1 și 999999999 lei (depinde de inflație și, implicit, de FMI).

Fără deosebire, în funcție de specificul bazei de date, domeniul poate fi extins sau restrâns, după cerințe. De exemplu, la regiunile luate în discuție mai sus mai pot fi adăugate și provincii precum: Bucovina, Maramureș, Crișana.

**Integritatea domeniului** privește controlul sintactic și semantic al unei date oarecare și face referință la modul său de definire.

**Două domenii sunt declarate compatibile dacă sunt comparabile din punct de vedere semantic și sintactic**, adică ansamblurile care le definesc nu sunt disjuncte. Prin definiție, două domenii identice sau incluse unul în celălalt sunt compatibile.

Pentru două tabele: FURNIZORI cu atrbutele: Nume, Adresa și Localitate și CLIENTI cu atrbutele: Nume, Adresa, Localitate și CodFiscal, se poate spune că domeniile atrbutorilor Nume sunt compatibile (ambele cuprind mulțimea firmelor românești; de fapt, un sir de caractere de o anumită lungime). Același lucru se poate spune despre domeniile atrbutorilor Adresa și Localitate ale celor două relații.

Ştiinţific Peterlee (Marea Britanie), patronat de firma IBM; RDMS (*Relational Data Management System*) – General Motors; INFORMIX (denumită iniţial *Relational Data Systems*) etc.

Până la jumătatea anilor '80, SGBD-urile relaţionale au stat în umbra celor hierarhice şi reiea. Motivul principal l-a constituit viteză mai mică, parametru esenţial în sistemele „confruntante” cu accesul simultan a sute sau chiar mii de utilizatori. În a doua parte a anilor '80, performanţele SGBDR-urilor au fost net ameliorate. Căştigul de viteză, care s-a adăugat atuului principal, interogarea ad-hoc prin utilizarea limbajelor de generaţia a IV-a (SQL, QBE, Quel), a largit continuu segmentul ocupat de SGBDR-uri, acesta fiind estimat în 1997 la aproximativ 60% din totalul pieţei bazelor de date<sup>12</sup>.

Practic, pentru toate platformele (mainframe-uri, minicalculatoare, staţii de lucru, PC-uri) şi sistemele de operare există o întreagă gamă de SGBDR-uri deosebit de performante, printre „protagoniştii” numărându-se produsele firmelor: Oracle, IBM, Sybase, Informix, Microsoft etc.

O adevărată „democratizare” a SGBDR-urilor s-a produs o dată cu dezvoltarea explozivă a PC-urilor. Produse precum dBase, RBase, Clipper, FoxPro, Paradox, Access etc. s-au vândut (şi copiat ilegal) în milioane de exemplare. Destinate iniţial utilizatorilor de PC-uri cu un redus bagaj de cunoştinţe în domeniul bazelor de date, astăzi aceste produse sunt dotate cu seturi puternice de instrucţiuni şi funcţii şi prezintă o interfaţă de lucru deosebit de prietenosă, atâtui ce conferă o mare dinamică acestui segment al pieţei, denumit şi cel al *SGBD-urilor micro*.

Exploataabile pe platformele de lucru MS-DOS/Windows, reprezintă zona „de jos” care vizează depoziştii ne-profesionişti în ale informaticii, dar şi dezvoltatori de aplicaţii la scară medie. Totodată, în retelele complexe (de întreprindere), SGBD-urile *micro* pot fi utilizate ca medii de dezvoltare pentru staţiile *client*, prin care se asigură accesul utilizatorilor la BD administrate printr-un SGBDR „profesional”.

Începuturile acestei familii de produse-program se pierd în negura istoriei... anilor '70. Un anume Wayne Ratcliff a elaborat un SGBD simplu, dedicat microcalculatoarelor. La acel moment, acestea erau echipate cu microprocesoare pe 8 biţi (Intel 8080, Zilog Z80, Motorola 6800 etc.), memorie internă de ordinul a zeci de kiloocteţi, unităţi de dischete de 256 Ko şi alte „facilităţi” care ar strănge în spate astăzi pe orice utilizator de PC dotat cu procesor Pentium II sau III. Produsul este preluat de firma Ashton-Tate (la care George Tate era director), care îl redenumeşte *dBase*. Versiunea *dBase II* a avut parte de o intrare triumfală în familia programelor pentru microcalculatoare. Paradoxal sau nu, nu a existat un *dBase I*, Tate considerând că lansarea produsului cu sufîxul II ar sugera utilizatorului că produsul este deja matur. Şi aşa a fost... După *dBase II* urmează versiunea *III*, apoi *III plus*, care va cunoaşte gloria, fiind etalonul „de facto” pe piaţă nou-apărută a SGBD-urilor pentru PC-uri.

Visul frumos al firmei Ashton-Tate s-a diluat şi apoi spulberat o dată cu versiunea *dBase IV*. Intens mediatisată şi aşteptată destul de mult, prima versiune, *dBase IV 1.0*, a fost „presărată” cu erori atât de hilare, încât produsul a devenit rapid emblema decenului opt şi începutului de deceniu 9 pentru programele făcute în pripă şi care prezintă multe erori sau, cum sugestiv sună în limba engleză, *quick and dirty* („în grabă şi murdar”). Este adevărat, „stafeta” a fost preluată de alte produse (Corel Draw 5 este numai un exemplu), dar concurenţa e acerbă, chiar case mari (în primul rând Microsoft) fiind jinta ironiilor datorate „bug-urilor” de care nu duc lipsă pachetele de programe pe care le produc.

12. Preluare din *01 Réseaux*, nr. 39 (iunie), 1997, p. 107.

Pentru Ashton-Tate, dBase IV 1.0 a fost începutul declinului. Datorită dificultăţilor financiare, în 1991 firma a fost preluată de Borland (astăzi Inprise), care n-a reușit, nici ea, să-şi revină (financiar) din „socul” Ashton-Tate.

Între timp, au apărut şi câştigat poziţii avansate produsele competitoare. Unul dintre cele mai cunoscute este FoxPro. La mijlocul deceniu lui 8, la Seattle (SUA), firma Fox Software a atrăs atenţia prin produsul său principal, FoxBase, care era o „clonă” dBase. La scurt timp însă, FoxBase-ul a surclasat „maestrul” dBase la aproape toate capitolurile: viteză, interactivitate, pluralitate de platforme (încă din anii '80 a fost disponibilă o versiune FoxBase pentru Macintosh) etc.

Ulterior FoxBase devine FoxPro, anul 1991 fiind anul în care marea majoritate a publicaţiilor de specialitate au „incoronat”, la categoria SGBD-uri pentru microcalculatoare, pe FoxPro 2.0, iar altele, precum *PC Magazine*, *InfoWorld*, *PC Computing*, *Byte*, l-au desemnat drept „cel mai bun produs-program al anului 1991”<sup>13</sup>. Este şi motivul pentru care în 1992, firma Microsoft, care la acea dată nu avea un SGBD de forţă la această categorie, a înghijit Fox Software şi, începând cu versiunea 2.5, FoxPro poartă sigla gigantului din Redmont (statul Washington). Unii spun că de aici începe şi declinul FoxPro.

Dintre ceilalţi concurenţi puternici ai dBase-ului amintim doar Paradox (Borland), RBase (Microrim), Clipper (Nantucket, companie fondată de doi ex-Ashton-Tate şi cumpărată ulterior de Computer Associates), dBFast (Computer Associates), Approach (Lotus, achiziţionată în 1995 de IBM), Access (Microsoft) etc.

Lumea SGBD-urilor *micro* a suferit schimbări majore atât în fizionomie, dar şi în modul efectiv de lucru, mai ales pentru cele din clasa XBase. Visual Objects (Computer Associates), Visual FoxPro, Visual dBase seamănă din ce în ce mai puţin cu ceea ce a fost altădată un produs XBase, fiind instrumente care înglobează tehnologii recente: client-server, replicare, programare orientată pe obiecte, dezvoltarea rapidă a aplicaţiilor (RAD) etc.

Este greu de făcut o ierarhizare, însă piaţa a consacrat la această categorie produsul Access al firmei Microsoft. Ajuns la versiunea 2000, inclus în suita Office cu acelaşi sufix, Access este ieftin, intuitiv şi bine integrat cu ceilalţi „pilieri”, Excel şi Word. Astfel incărtăria sa de utilizare este superioară oricărui alt produs competitor.

Urcând la categoria superioară SGBD-urilor *micro*, se cuvine de amintit că piaţă a cochetat pe la mijlocul anilor '90 cu sintagma *servere de date pentru grupuri de lucru*<sup>14</sup>. Acest concept a fost pus în practică de firma Microsoft, prin comercializarea la un preț foarte avantajos a produsului SQL Server, un SGBD ce permitea iniţial accesul simultan la baza de date a unui număr de 20-30 utilizatori. Fără a se ridica la pretenţiile unui SGBD Unix, SGBD-urile pentru grupurile de lucru au fost declarate instrumentul ideal pentru firmele mici şi mijlocii, acolo unde Paradox, FoxPro, Access îşi arătau limitele în materie de volum de date ce poate fi gestionat, securitate, păstrarea coerentei în caz de avarie etc. Prin lansarea unor produse sub eticheta *Workgroup Databases*, marii producători de SGBD-uri au creat o anumită confuzie în rândul utilizatorilor şi chiar al distribuitorilor.

În ultimul timp, SGBD-urile pentru grupuri de lucru au fost assimilate celor de „categoria grea”: Oracle, DB2, Sybase, Informix, SQL Server etc. Acestea sunt orientate pe gestiunea informatizată a organizaţiilor mari şi foarte mari, asigurându-se accesul simultan

13. Preluare din *PC World România*, iunie 1993, p. 33.

14. Vezi T. Lévy-Abégouli, „SGBD Workgroup: un concept très marketing”, *01 Informatique*, nr. 1354/1995, pp. 24-25; „Workgroup Databases Redux” (M. Ricciuti), *InfoWorld*, vol. 17, nr. 46/1995, p. 43.

Modelarea realității se concretizează în *tabele de valori* numite relații, avându-se în vedere că:

- relație are un *nume*;
- coloană reprezintă un *atribut*;
- linie reprezintă un *n-uplet (tuple)* de valori ale celor *n* attribute din relație;
- ordinea liniilor și coloanelor în cadrul unei tabele nu este relevantă pentru conținutul informațional.

Fiecare linie a unei tabele reprezintă o entitate sau un fapt al realității, în timp ce o coloană reprezintă o proprietate a acestei entități sau fapt. Introducând un plus de rigore, se cuvine să remarcăm că entitatea poate fi nu numai un obiect concret/proces (o persoană, un lucru oarecare), dar și o relație între obiecte (persoane, lucruri), cum ar fi: contractele de afaceri, căsătoriile (există totuși câteva deosebiri între acestea și precedentele), structura ierarhică a unei organizații etc. În plus, nu e întotdeauna evident care dintre informații sunt entități, care attribute și care asociații (relații).

Dintre coloane, o parte are drept scop identificarea liniilor (*identificator sau cheie primară*); altele constituie suportul de referință către liniile din alte tabele (*coloane de referință sau chei strânsă*).

Ansamblul valorilor stocate în tabele reprezintă *conținutul* bazei de date, conținut ce poate fi modificat prin operațiuni de actualizare: introducerea unor linii (tuple) noi, stergerea unor linii, modificarea valorii unor attribute.

În figura 1.5<sup>9</sup> este prezentată o schemă incompletă de evoluție a bazelor de date, din punctul de vedere al modelului intern de organizare a datelor. Prima etapă este reprezentată de sisteme care utilizează modelul *file-based* (pre-baze de date), a doua – modelul *rețea*, a treia – modelul *relațional*, iar ultima, în curs de maturizare, este etapa SGBD-urilor *orientate pe obiecte*. Schemei își pot reproba absența unui model important, din care a derivat cel *rețea*, și anume modelul *ierarhic*.

Față de modelele ierarhice și *rețea*, modelul relational prezintă câteva avantaje<sup>10</sup>:

- propune structuri de date ușor de utilizat;
- ameliorează independența logică și fizică;
- pune la dispoziția utilizatorilor limbaje ne-procedurale;
- optimizează accesul la date;
- îmbunătățește integritatea și confidențialitatea datelor;
- ia în calcul o largă varietate de aplicații;
- abordează metodologic definirea structurii bazei de date.

Modelului relational îl este asociată *teoria normalizării*, care are ca scop prevenirea comportamentului aberrant al relațiilor în momentul actualizării lor, eliminarea datelor redundante și înlesnirea înțelegerei legăturilor semantice dintre date.

9. Prelucrare din M. Sărbu – „Dincolo de relațional?”, PC Report, nr. 35 (august), 1995

10. [Adiba&Delobel82], citată în [Saleh94], p. 24. Pentru detalii legate de obiectivele vizate de Codd atunci când a formulat modelul relational, vezi și [Date99-2].

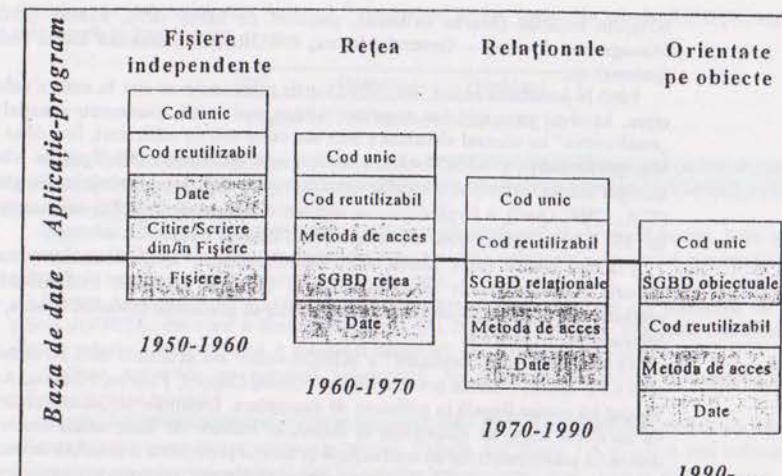


Figura 1.5. Evoluția bazelor de date

Articolele lui Codd au avut un ecou semnificativ în lumea cercetătorilor. Firma IBM a inițiat un proiect major care trebuia să se materializeze în implementarea unui sistem de gestiune a bazelor de date relaționale, numit System/R. Proiectul s-a derulat în laboratorul Santa Teresa din San Jose, California. Prima fază – anii 1974 și 1975 – s-a concretizat într-un prototip al SGBDR-ului, prototip care a fost rescris în anii 1976 și 1977 pentru a permite interogarea simultană a mai multor tabele și accesul multi-utilizator. IBM a distribuit produsul pe parcursul anilor 1978 și 1979 la câțiva colaboratori ai săi pentru evaluări, dar în 1979 firma abandonă proiectul System/R, considerându-l nefezabil<sup>11</sup>. System/R a fost însă punctul de plecare pentru realizarea unuia dintre cele mai cunoscute SGBDR-uri, DB2.

Între timp, în 1977, un grup de ingineri de la Menlo Park, California, a fondat compania Relational Software Inc., cu scopul declarat de a dezvolta un SGBD bazat pe limbajul SQL. Produsul, denumit Oracle, a fost lansat în 1979 și a reprezentat primul SGBDR comercial.

Dincolo de faptul că Oracle precedă cu doi ani primul SGBDR comercial al firmei IBM, orientarea sa a fost extrem de bine gândită, rulând pe minicalculatele VAX (Digital Equipment Corporation), sensibil mai ieftine decât mainframe-urile IBM, și, deci, cu un segment de piată mai mare. Astfel, succesul a fost vizibil. Astăzi, firma, redenumită Oracle Corporation, este liderul produselor software dedicate mediilor de lucru cu bazele de date.

Primul SGBDR comercializat de IBM a fost SQL/DS (DS – Data System), anunțat în 1981 și comercializat în 1982. În 1983, IBM a lansat DB2, dedicat inițial mainframe-urilor sale, dar care în timp a fost portat pe toate sistemele de operare importante.

Alte SGBD-uri din perioada de pionierat a relaționalului au fost: INGRES (Interactive Graphics and Retrieval System) – al Universității Berkeley; QUERY BY EXAMPLE – Centrul de cercetări T.J. Watson; PRTV (Peterlee Relational Test Vehicle) – Centrul

11. Pentru amănunte privind odiseea System R, vezi și [http://www.mcjones.org/System\\_R/SQL\\_Reunion\\_95](http://www.mcjones.org/System_R/SQL_Reunion_95).

privește nivelul extern, utilizatorul nefiind realmente interesat de modul în care informația se găsește fizic pe suportul de înregistrare.

Rezolvarea acestor probleme cade sub incidența gestionarului bazei de date, care este un modul de programe ce realizează interfața dintre datele interne (de pe suport) conținute în bază și programele (sau comenziile) de consultare și actualizare; principalele sale sarcini pot fi grupate astfel:

- Interacțiunea cu gestionarul de fișiere. Datele brute sunt stocate pe disc prin intermediul sistemului de gestiune a fișierelor, care este o componentă a sistemului de operare al calculatorului. Gestionarul BD traduce instrucțiunile diverselor DML în instrucțiuni-sistem, la nivel elementar, fiind „responsabil” și de buna desfășurare a operațiilor de scriere/citire a datelor în/din bază.
- Validitatea (corectitudinea) datelor. Datele stocate trebuie să satisfacă anumite restricții de integritate, specificate de administratorul bazei. După implementarea mecanismului de integritate, gestionarul verifică dacă toate actualizările se derulează cu respectarea restricțiilor.
- Securitatea datelor. Accesul la date trebuie să fie selectiv, gestionarul bazei fiind cel care asigură respectarea drepturilor de acces ale fiecărui utilizator.
- Salvare și restaurare. În sistemele informatici sunt supuse accidentelor: distrugerea suprafeței magnetice, întreruperi în furnizarea energiei electrice, erori ale programelor etc. sunt inerente chiar și în ţări super dezvoltate. În multe cazuri, datele ce erau în curs de prelucrare în momentul accidentului se alterează sau se pierd în totalitate. Gestionarul are dificila misiune de a detecta la timp avariile și de a restaura datele pierdute în forma și conținutul dinaintea accidentului. Aceasta se realizează prin intermediul unor programe speciale de salvare-restaurare.
- Prelucrări simultane (concurente). Întrucât în același timp pot lucra cu baza doi sau mai mulți utilizatori, trebuie asigurată coerența datelor prin afectarea unor niveluri de prioritate la nivel de operații și utilizatori.

#### **Administratorul bazei de date**

Din cele prezentate până acum reiese clar că un sistem de gestiune a bazei de date reprezintă un ansamblu de programe ce asigură controlul complet al datelor, dar și al aplicațiilor care le exploatează. Administratorul unei baze de date este persoana responsabilă de sistem în ansamblul său. Rolul acestuia este determinant în câteva activități foarte importante.

- Definirea arhitecturii bazei de date se realizează prin scrierea definițiilor care vor fi transformate de compilatorul DDL într-un ansamblu de tabele stocate permanent în dicționarul de date.
- Definirea modalităților în care va fi structurată memoria externă și a metodelor de acces la date. Acestea devin operaționale prin redactarea unor specificații scrise ca instrucții ale limbajului de definire (descriere) a datelor.
- Modificarea arhitecturii și organizării fizice a bazei de date poate fi realizată prin intermediul instrucțiunilor DDL, obținându-se astfel actualizările corespondente ale dicționarului de date.
- Autorizarea accesului la date se acordă fiecărui utilizator al bazei de date, administratorul fiind cel care decide asupra datelor ce pot fi consultate și actualizate de fiecare utilizator sau grup de utilizatori.

- Specificarea restricțiilor de integritate. Aceste restricții sunt stocate pe disc și consultate de gestionarul bazei la fiecare actualizare.

În plus, administratorul bazei de date este cel care: asigură legătura cu utilizatorii; definește procedurile de verificare a drepturilor de acces și a procedurilor de validare a integrității datelor; definește strategia de salvare (înregistrarea copiilor de siguranță)/restaurare a bazei; monitorizează performanțele bazei și o adaptează la modificările ulterioare ale sistemului informațional.

După cum se poate observa, sarcina unui administrator este deosebit de complexă, iar rolul său este primordial în exploatarea bazei de date. De aceea, raportul ANSI/SPARC 1975 delimitizează trei categorii de administratori:

- Administratorul întreprinderii, responsabil cu exploatarea și identificarea globală a aplicațiilor prezente și viitoare utilizate în organizația respectivă;
- Administratorul aplicațiilor, responsabil cu dezvoltarea sub-schemelor externe pentru aplicații și utilizatori;
- Administratorul datelor precizează necesarul și disponibilitatea datelor la nivelul schemei interne.

#### **Utilizatorii bazelor de date**

Cu riscul de a simplifica lucrurile excesiv, ne putem opri numai la patru categorii de utilizatori ce pot interacționa cu sistemul:

**Programatorii de aplicații.** Sunt informaticienii care interacționează cu baza de date prin intermediul instrucțiunilor DML integrate în programe scrise într-un limbaj-gazdă (Cobol, Pascal, C, Visual Basic, Java) sau în limbajul SGBD-lui respectiv.

**Utilizatorii ocazionali.** Aceștia sunt persoane care interacționează cu sistemul nu prin intermediul programelor de aplicații, ci printr-un limbaj de consultare specific bazei. Fiecare interogare este transmisă unui procesor de consultare având rolul de a prelucra instrucțiunile DML și a le face „inteligibile” pentru gestionarul bazei.

**Utilizatorii curenți** sunt persoane care efectuează operații de rutină asupra bazei de date, utilizând aplicațiile disponibile.

**Utilizatorii specializați** reprezintă un ansamblu de specialiști care dezvoltă aplicații-program ce lucrează cu baze de date, dar în afara cadrului general al SGBD-urilor. Acei sunt inclusi specialiștii în sisteme expert, proiectare asistată de calculator, modelare etc.

## **1.2. Un pic de istorie recentă și relațională**

Codd a definit modelul relațional al datelor printr-o serie de *structuri* de date (relații alcătuite din tupluri), *operații* aplicate asupra structurilor de date (selecție, proiecție, juncție etc.) și *reguli de integritate* care să asigure consistența datelor (chei primare, restricții referențiale și.a.).

Aceste cinci module interacționează cu o serie de componente fizice ale bazei:

- Fișierele de date reprezintă suportul propriu-zis al bazei.
  - Dicționarul de date înmagazinează informații relative la structura bazei, fiind solicitat în toate operațiunile de consultare și actualizare.
  - Îndecesă, un număr suficient pentru mărirea vitezei de acces la date.
- Sintetic, structura unui sistem de lucru cu o bază de date este prezentată în figura 1.4<sup>8</sup>.

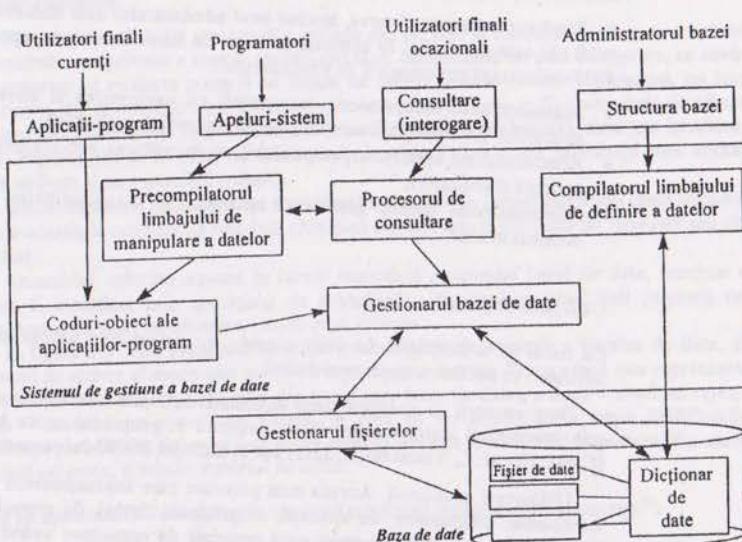


Figura 1.4. Structura unui sistem de lucru cu o bază de date

### Limbaje de definire a datelor

Arhitectura unei baze de date este specificată printr-o serie de definiții redactate sub formă de instrucții scrise într-un anumit limbaj – limbajul de definire a datelor – DDL (Data Definition Language). Execuția acestor definiții se materializează într-un ansamblu de tabele care sunt memorate într-un fișier special, denumit dicționar (sau repertoar) de date.

Un dicționar de date conține deci metadate, adică date relative la alte date, fiind consultat înaintea oricărei citiri sau modificări a datelor din bază.

Principalele funcții ale DDL sunt:

- Descrierea logică a bazei de date și sub-schemelor proprii fiecărui grup de utilizatori.
- Identificarea schemei, sub-schemelor și diverselor agregări de date.

8. Preluat din [Korth&Silberschatz88], p. 19.

- Specificarea fișierelor de date și a legăturilor logice dintre acestea. Pe baza acestor specificații se poate realiza accesul la date chiar și în condițiile coexistenței mai multor modele de organizare într-o același BD.
- Definirea restricțiilor semantice la care sunt supuse datele. Acestea reprezintă ansamblul valorilor permise ale fiecărei date, eventual formula de calcul a unei date pe baza valorilor altor date. Respectarea acestor restricții asigură coerența bazei.
- Definirea cheilor de acces rapid și a cheilor confidențiale (parolelor).
- Definirea metodelor de exploatare a fișierelor ce vor fi utilizate în aplicații pentru selectarea înregistrărilor.
- Definirea procedurilor speciale de criptare, în vederea generării cheilor de acces.
- Definirea modului de indexare sau de localizare a entităților.
- Determinarea tipului unei date, în sensul de dată de bază sau derivată (calculată printr-o expresie, pe baza valorilor altor date).

Dată fiind complexitatea structurilor de memorare și metodelor de acces la acestea, la nivel elementar fișierele de date și dicționarul de date sunt accesibile numai specialiștilor. În schimb, pentru descrierea schemei BD, marea majoritate a limbajelor de interogare prezintă comenzi adecvate, ce pot fi utilizate și de ne-informaticieni.

### Limbaje de manipulare a datelor

Prin manipularea datelor se înțelege efectuarea uneia dintre următoarele operații:

- extragerea unor date din bază (consultare);
- scrierea de noi date în bază (adăugare);
- ștergerea datelor perimale sau eronate (uneori chiar și a celor corecte);
- modificarea valorii unor date.

Un limbaj de manipulare a datelor (DML – Data Manipulation Language) este utilizat pentru a prelucra datele în funcție de structura lor. La nivel fizic, prin DML se vizează identificarea și implementarea unor algoritmi performanți de acces la date, în timp ce la nivel extern DML trebuie să faciliteze dialogul utilizatorului cu baza în vederea obținerii informațiilor dorite.

În mod curent, termenul *consultare* sau *interrogare* desemnează acțiunea de căutare și identificare a datelor necesare, dintre cele aflate în bază. Ansamblul instrucțiunilor DML pentru căutarea și identificarea datelor constituie limbajul de consultare.

### Gestionarul bazei

O bază de date consumă un volum considerabil de memorie, astfel încât poate fi stocată numai pe disc, memoria internă (RAM) a calculatorului fiind insuficientă (și volatilă). Programele de exploatare fac frecvent transferuri de date între memoria internă și disc. Deoarece viteza de transfer este mult mai mică, prin comparație cu viteza de lucru a procesorului, găsirea unor soluții, la nivel fizic, pentru transferul rapid al datelor prezintă o importanță deosebită.

Pe de altă parte, obiectivul esențial al bazelor de date este de a facilita exploatarea datelor, în vederea obținerii de către utilizatori a informațiilor necesare. Acest obiectiv

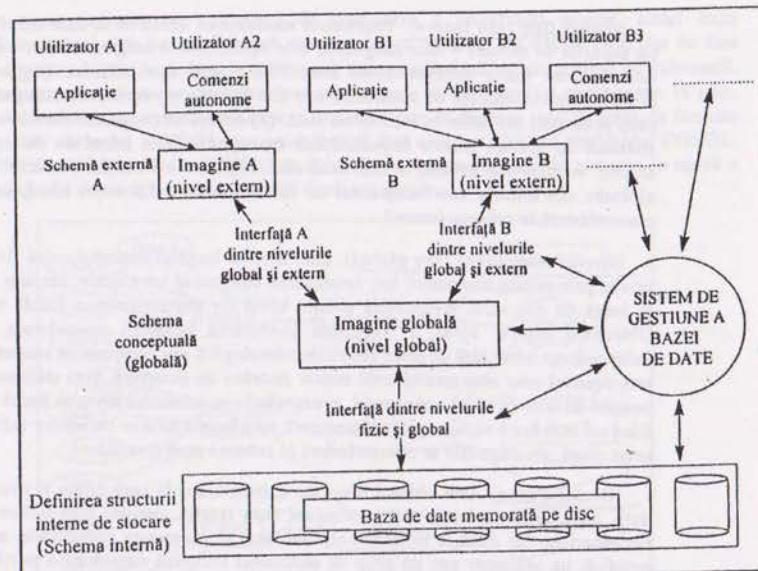


Figura 1.3. Schematizarea unui sistem de lucru cu o bază de date

Pozibilitatea modificării structurii la un nivel, fără a afecta structura nivelului sau nivelurilor superioare, se numește autonomie a datelor stocate în bază. Se poate vorbi de două paliere de autonomie.

**Autonomia fizică** reprezintă posibilitatea modificării arhitecturii bazei la nivel intern, fără ca aceasta să necesite schimbarea schemei conceptuale și rescrierea programelor pentru exploatarea bazei de date. Asemenea modificări sunt necesare uneori pentru ameliorarea performanțelor de lucru (viteză de acces, mărimea fișierelor etc.). Tot autonomia fizică este cea care asigură portarea bazei de date de pe un sistem de calcul pe altul fără modificarea schemei conceptuale și a programelor.

**Autonomia logică** presupune posibilitatea modificării schemei conceptuale a bazei (modificare datorată necesității rezolvării unor noi cerințe informaționale) fără a rescrie programele de exploatare. Autonomia logică a datelor este mai greu de realizat decât autonomia fizică, deoarece programele de exploatare sunt dependente, în foarte mare măsură, de structura logică a datelor pe care le consultă și actualizează, în ciuda existenței dicționarului de date. Firește, un element important îl reprezintă și anvergura modificării schemei conceptuale.

### Sisteme de gestiune a bazelor de date

Apărute în anii '60, Sistemele de Gestiune a Bazelor de Date (prescurtat SGBD-uri) reprezintă un ansamblu de programe ce permit utilizatorilor să interacționeze cu o bază de date, în vederea creării, actualizării și interogării acesteia. SGBD-ul este cel care asigură și supravezează: introducerea de informații în baza de date; actualizarea și extragerea datelor din bază; autorizarea și controlul accesului la date; păstrarea independenței dintre structura bazei și programe<sup>6</sup>.

Obiectivul esențial al unui SGBD este furnizarea unui mediu eficient, adaptat utilizatorilor care doresc să consulte sau să actualizeze informațiile conținute în bază. Bazele de date sunt concepute pentru a prelucra un volum mare de informații. Gestionarea acestora impune nu numai o structurare riguroasă a datelor, dar și o raționalizare a procedurilor de acces și prelucrare.

Principalele funcții ale unui SGBD vizează:

- descrierea ansamblului de date la nivelurile fizic și conceptual;
- crearea (initializarea) și exploatarea (consultarea și actualizarea) bazei de date;
- controlul integrității bazei;
- confidențialitatea informațiilor conținute în bază;
- accesul simultan al mai multor utilizatori la informații;
- securitatea în funcționare;
- furnizarea unui set de comenzi și instrucțiuni necesare atât utilizatorilor pentru consultarea directă a bazei, prin intermediul unui limbaj de manipulare, cât și programatorilor, pentru redactarea programelor de lucru cu baza de date;
- revizia și restructurarea bazei;
- monitorizarea performanțelor.

Un SGBD prezintă, în general, următoarele module<sup>7</sup>:

- ① Gestionarul fișierelor, care se ocupă cu afectarea spațiilor de memorie pe disc și cu structurile fizice de date care servesc la reprezentarea informațiilor pe suport.
- ② Gestionarul bazei de date face legătura datelor „fizice” din bază cu aplicațiile-program de consultare și actualizare.
- ③ Procesorul de consultare „traduce” instrucțiunile limbajului de consultare în instrucțiuni elementare, „inteligibile” pentru gestionarul bazei. Mai mult, acesta optimizează consultarea, pentru a obține rezultatele într-un timp cât mai scurt.
- ④ Modulele DML (Data Manipulation Language) realizează conversia instrucțiunilor limbajului de manipulare a datelor (DML) inserate într-un program de aplicație în proceduri curente ale limbajului-gazdă, interacționând cu procesorul de consultare în vederea producerii secvențelor de cod adecvate.
- ⑤ Modulele Limbajului de Definire a Datelor — DDL (Data Definition Language) „traduc” (prin compilare sau interpretare) și execută instrucțiunile DDL, obținându-se ansamblul de tabele ce reprezintă metadatele stocate în dicționarul de date.

6. [G. Dodescu §.a. 87], p. 511.

7. [Korth&Silberschatz88], pp. 17-18.

Avantajele organizării informațiilor în baze de date decurg tocmai din existența acestui fișier de descriere globală a bazei, denumit, în general, *dicționar de date* (denumit și repertoar de date sau catalog de sistem). Extragerea și modificarea datelor – altfel spus, lucrul cu fișierele de date – se derulează exclusiv prin intermediul dicționarului în care se găsesc informații privitoare la structura datelor și restricțiile îndeplinite de acestea. Iată câteva dintre avantaje:

- Un grad redus de redundanță a datelor.
- Evitarea, în mai mare măsură, a inconsistentei datelor.
- Facilitarea partajării informațiilor între toți utilizatorii acestora din cadrul organizației.
- Suport pentru standardizare.
- Implementarea unor mecanisme ameliorate privind asigurarea securității informațiilor.
- Îmbunătățirea integrității datelor.
- Un mai bun suport pentru rezolvarea conflictelor ce apar la încercările de modificare simultană a unei același date (de către doi sau mai mulți utilizatori).
- Structurile de date sunt mai aproape de realitate și mai ușor de manipulat.
- Este permisă legătura cu diverse limbi-gazdă.
- Întreprinderea poate fi abordată global, luându-se în considerare și interacțiunile dintre compartimente (producție, marketing, personal, finanțe, contabilitate).
- Datele fiind separate de programele de consultare și actualizare a lor, procesul de dezvoltare a aplicațiilor-program este sensibil ameliorat, efortul de scriere a programelor (codarea) diminuându-se considerabil.
- Sistemele informatiche ce utilizează baze de date sunt mai flexibile, reflectă mai bine specificul firmei, fiind adaptabile la modificările ulterioare ale mediului economic.

O bază de date (BD) reprezintă un ansamblu structurat de fișiere care grupează datele prelucrate în aplicațiile informatic ale unei persoane, grup de persoane, întreprinderi, instituții etc. Formal, *BD poate fi definită ca o colecție de date aflate în interdependență, împreună cu descrierea datelor și a relațiilor dintre ele, iar după G.D. Everest, o BD reprezintă o colecție de date utilizată într-o organizație, colecție care este automatizată, partajată, definită riguros (formalizată) și controlată la nivel central*<sup>4</sup>.

Bazele de date evoluează în timp, în funcție de volumul și complexitatea proceselor, fenomenelor și operațiunilor pe care le reflectă. Ansamblul informațiilor stocate în bază la un moment dat constituie *conținutul* sau instanța sau realizarea acesteia. Organizarea bazei de date se reflectă în *schema* sau structura sa, ce reprezintă un ansamblu de instrumente pentru descrierea datelor, a relațiilor dintre acestea, a semanticii lor și a restricțiilor la care sunt supuse. În timp ce volumul prezintă o evoluție spectaculoasă în timp, schema unei baze rămâne relativ constantă pe tot parcursul utilizării acesteia.

#### Niveluri de abstractizare a datelor

Într-un sistem informatic ce utilizează BD, organizarea datelor poate fi analizată din mai multe puncte de vedere și pe diferite paliere. De obicei, abordarea se face pe trei niveluri: *fizic* sau *intern*, *conceptual* sau *global* și *extern*.

4. [Everest86], p. 11.

**Nivelul fizic (sau intern).** Reprezintă modalitatea efectivă în care datele sunt „scrise” pe suportul de stocare – disc magnetic, disc optic, bandă magnetică etc. Structura datelor este descrisă foarte detaliat, fiind accesibilă numai specialiștilor (ingineri de sistem, programatori în limbi de asamblare sau alte limbi apropriate de „mașină”). Cele două părți principale ale bazei la acest nivel sunt: (1) un set de programe care interacționează cu sistemul de operare pentru îmbunătățirea managementului bazei de date și (2) fișierele stocate în memoria externă a calculatorului. Fișierele ce conțin datele propriu-zise sunt alcătuite din articole sau înregistrări cu format comun. La acest nivel, structura BD se concretează în *schema internă*.

**Nivelul conceptual (sau global).** Este nivelul imediat superior celui fizic, datele fiind privite prin prisma semanticii lor; interesează conținutul lor efectiv, precum și relațiile care le leagă de alte date. Reprezintă primul nivel de abstractizare a lumii reale observate. Obiectivul acestui nivel îl constituie modelarea realității considerate, asigurându-se independența bazei față de orice restricție tehnologică sau echipament anume. Întreaga bază este descrisă prin intermediul unui număr restrâns de structuri. Toți utilizatorii își exprimă nevoile de date la nivel conceptual, prezentându-le administratorului bazei de date, acesta fiind cel care are o vizionare globală necesară satisfacerii tuturor cerințelor informaționale. La acest nivel, structura BD se concretează în *schema conceptuală*.

**Nivelul extern.** Este ultimul nivel de abstractizare la care poate fi descrisă o bază de date. Structurile de la nivelul conceptual sunt relativ simple, însă volumul lor poate fi deconcertant. Iar dacă la nivel conceptual baza de date este abordată în ansamblul ei, în practică, un utilizator sau un grup de utilizatori lucrează numai cu o porțiune specifică a bazei, în funcție de departamentul în care își desfășoară activitatea și de atribuții. Simplificarea interacțiunii utilizatori-bază, precum și creșterea securității bazei sunt deziderate ale unui nivel superior de abstractizare, care este nivelul extern. Astfel, structura BD se prezintă sub diferite machete, cunoscute uneori și ca *sub-scheme*, scheme externe sau imagini, în funcție de nevoile fiecărui utilizator sau grup de utilizatori.

Luând în considerare cele trei niveluri de abstractizare, schematizarea unui sistem de lucru cu o bază de date se poate face ca în figura 1.3<sup>5</sup>.

Accesul utilizatorului la informațiile din bază este posibil numai prin intermediul sistemului de gestiune a bazei de date. În general, interfața utilizator-SGBD se poate realiza în două moduri:

- Printr-un mecanism de apel (cuvânt-cheie, ca de exemplu CALL) inserat în programele scrise într-un limbaj „tradițional” (C, COBOL etc.), acesta fiind cazul SGBD-urilor cu limbaj-gazdă.
- Prin comenzi speciale utilizate autonom (în afara aplicațiilor-program), în cazul SGBD-urilor autonome.

5. Prelucrare a schemei din [Date86], p. 33.

Se poate projecția un mecanism de menținere a integrității datelor, astfel încât actualizarea unei date într-un fișier să atragă automat actualizarea tuturor fișierelor de date în care apare aceasta, însă, în sistemele mari, care gestionează volume uriașe de informații, implementarea unui asemenea mecanism este extrem de complexă și costisitoare. În plus, fișierele de date sunt uneori proiectate și implementate la distanțe mari în timp, în formate diferite: de exemplu, FIȘIER1 este posibil să fi fost creat cu ajutorul limbajului COBOL, FIȘIER2 în FORTRAN, iar FIȘIER3 în BASIC. În asemenea condiții, punerea în operație a mecanismului de menținere a integrității devine o utopie.

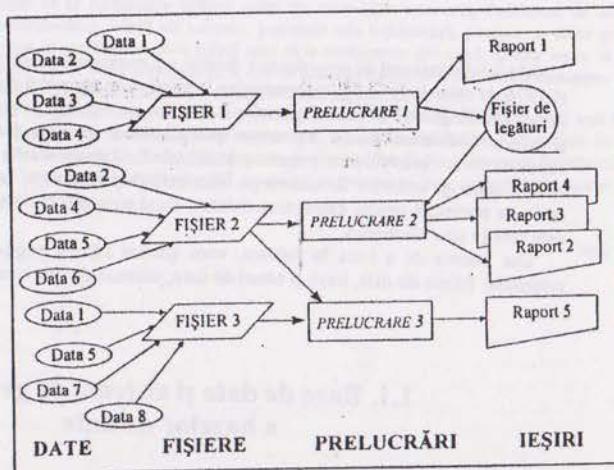


Figura 1.1. Sistem informatic bazat pe organizarea datelor în fișiere independente

Chiar numai și din cele prezentate mai sus, se pot desprinde câteva dezavantaje ale organizării datelor după modelul *file-based*<sup>2</sup>:

- Redundanță și inconsistență datelor: o aceeași dată apare în mai multe fișiere; în aceste cazuri există riscul modificării acesteia într-un fișier fără a face modificările și în toate celelalte fișiere.
- Dificultatea accesului. Într-o întreprindere, o aceeași informație este exploatață de mai mulți utilizatori. Spre exemplu, pentru departamentul care se ocupă cu gestionarea stocurilor, intrările de materiale trebuie ordonate pe magazii (depozite) și repere, în timp ce pentru departamentul care se ocupă cu decontările cu partenerii de afaceri ai întreprinderii, intrările trebuie ordonate pe furnizori ai materialelor. Or, fișierele tradiționale nu facilitează accesarea datelor după mai multe criterii, specific diferenților utilizatori sau grupuri de utilizatori.
- Izolarea datelor: când datele sunt stocate în formate diferite, este dificil de scris programe care să realizeze accesarea datelor într-o manieră globală a tuturor celor implicate în derularea unei tranzacții.

2. [Korth&Silberschatz88], pp. 3-5; [Everest86], pp. 31-34.

- Complexitatea deosebită a actualizațiilor. O actualizare presupune adăugarea, modificarea sau ștergerea unor informații din fișiere. Cum prelucrările se desfășoară în timp real, de la mai multe terminale (în mediile multi-utilizator), pot apărea situații conflictuale atunci când doi utilizatori doresc modificarea simultană a unei același date. Rezolvarea acestui gen de conflicte presupune existența unui program-supervizor al prelucrărilor, care este greu de realizat cu o multitudine de fișiere, create la distanță în timp și în formate diferite.
- Probleme de securitate în de dificultatea creării unui mecanism care să protejeze pe deplin datele din fișiere de accesul neautorizat.
- Probleme legate de integritatea datelor. Informațiile stocate în fișiere sunt supuse la numeroase restricții semantice. Toate aceste restricții alcătuiesc mecanismul de integrare a datelor, deosebit de complex în mediile de lucru multi-utilizator și eterogene.
- Inabilitatea de a obține răspunsuri rapide la probleme ad-hoc simple.
- Costul ridicat se datorează gradului mare de redundanță a datelor, eforturilor deosebite ce trebuie depuse pentru interconectarea diferitelor tipuri de fișiere de date și pentru asigurarea funcționării sistemului în condițiile respectării unui nivel minim de integritate și securitate a informațiilor.
- Inflexibilitatea față de schimbările ulterioare, ce sunt inerente oricărui sistem informațional.
- Modelarea inadecvată a lumii reale.

Sintagma *bază de date* apare pentru prima dată în titlul unei conferințe organizate la Santa Monica (California) în 1964 de System Development Corporation<sup>3</sup>. Majoritatea lucrărilor consideră ca moment al consacrării termenului anul 1969, când CODASYL publică, în cadrul unei conferințe dedicate limbajelor de gestiune a datelor, primul raport tehnic [CODASYL69] în care este prezentat conceptul de „bază de date”. Față de modelul *file-based*, nouitatea o constituie existența unui fișier de descriere globală a bazei, astfel încât să se poată asigura independenta programelor față de date, după cum o arată și figura 1.2.

BAZA DE DATE

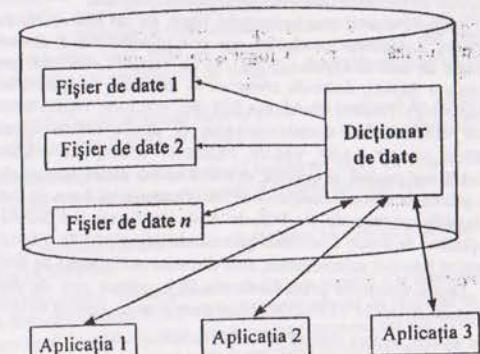


Figura 1.2. Schemă de principiu a unei baze de date

3. Conferința s-a intitulat *Development and Management of a Computer-Centered Data Base*.

Una din criticile pe care mi le asum este că nu am prezentat nici un dialect din categoria SGBD-ilor gratuite, astăzi din ce în ce mai populare: PostgreSQL, MySQL, Interbase etc. Poate în viitoarea ediție a cărții, după ce prezenta se va fi epuizată din anticariat și de la buchiniști (vă dăți seama că lucrul acesta nu o să se petreacă prea curând, chiar dacă tirajul ar fi foarte mic).

Capitolul 4 este unul de introducere în interogările SQL, de prezentare a principalelor clauze ale frazei SELECT. În parte, exemplele sunt cele din algebra relațională, dar discuția depășește cu mult cadrul celor opti operatori din capitolul 2. Astfel, sunt discutate: coloanele calculate (împreună cu opțiuni de concatenare și operații cu date calendaristice), ordonarea linioilor în rezultatul unei interogări, operatori pentru lucru cu intervale (BETWEEN), liste (IN) și săboane (LIKE), sinonime locale și autojoinuri, funcții de agregare (COUNT, SUM, AVG, MIN, MAX), precum și clauzele de grupare (GROUP BY și HAVING).

Capitolul 5 se aplicație asupra cătorva „delicatese” SQL: valori nule, junciune externă, structuri alternative generalizate (CASE), subconsultări (operatorul IN), operatorii ALL, SOME și ANY și utilizarea subconsultărilor în clauza HAVING a unei fraze SELECT. Soluțiile formulate pun în evidență avantajele și limitele fiecăruia dintre cele trei produse – DB2, Oracle și Visual FoxPro. În finalul capitolului sunt prezentate câteva variante de refugiu în SQL, în sensul că, atunci când dialectul produsului nu permite anumite elemente de finețe, se poate recurge la varianta neortodoxă de a salva rezultatele consultărilor în tabele virtuale sau temporare și folosirea acestora ca argumente în alte fraze SELECT, altfel spus, secvențializarea interogărilor. Un bonus al paragrafului îl constituie prezentarea expresiilor-tabelă din DB2.

După cum sugrează și titulatura, capitolul 6 ridică frazele SELECT pe „cele mai înalte culmi” ale SQL. Primul paragraf este dedicat uneia dintre cele mai dificile probleme în formularea consultărilor: corelarea simplă și, mai ales, cea dublă. Prezentarea mecanismului corelărilor este însoțită de exemple ce pun în valoare utilitatea operatorului EXISTS. Al doilea paragraf este mult mai puțin încordat, evocând o facilitate preferată de mulți dintre cei ce lucrează în SQL – includerea frazelor SELECT în clauza FROM a unei consultări/subconsultări, opțiune ce furnizează soluții pentru multe probleme informaționale aparent insolubile.

Reprezentarea structurilor ierarhice (arborescente) sunt tratate lapidat într-un paragraf special, împreună cu o serie de facilități Oracle în acest sens (clauzele CONNECT BY și START WITH). Paragraful final al capitolului prezintă câteva ingrediente ale rețetei de actualizare a tabelelor prin comenzi UPDATE ce utilizează subconsultări corelate.

Singurul capitol care se apropie, timid, de cel mai recent standard SQL, SQL-99, este cel de-al șaptelea. Apropierea nu vizează, așa cum probabil s-ar fi cunoscut, orientarea pe obiecte (în principal tipurile de date abstractive – ADT), ci un domeniu mult mai pragmatic, cel al analizei multidimensionale a datelor, domeniu consacrat în literatura de specialitate drept OLAP (*On Line Analytical Processing*). Nucleul OLAP din SQL-99: ROLLUP, CUBE, GROUPING, GROUPING SETS, RANK, ROW\_NUMBER etc. – este completat de câteva funcții specifice Oracle 8i2: PERCENT\_RANK, FIRST\_VALUE, LAST\_VALUE, PERCENT\_RANK, LAG și LEAD.

Ultimul capitol, al optulea, dezvoltă câteva dintre elementele prezentate în precedentele capitole, cu privire la obiectele componente ale schemei unei baze de date și întră în câteva detalii legate de extensiile procedurale ale SQL în două dintre cele trei SGBD-uri, VFP și Oracle. În completarea extensiilor de creație a tabelelor prin combinația CREATE TABLE / SELECT și declararea restricțiilor CHECK folosind subconsultări, sunt discutate modalitățile de creație și actualizare a tabelelor virtuale.

Dintre extensiile procedurale ale SQL, câteva zeci de pagini sunt cheltuite cu procedurile și funcțiile stocate în VFP și Oracle, precum și un tip special de proceduri stocate – declanșatoarele.

Vă urez lectură plăcută și vise frumoase...

## Capitolul 1

### NOTIUNI ALE MODELULUI RELAȚIONAL

Modelul relațional de organizare a datelor s-a conturat în două articole publicate în 1969 și 1970 de către E.F. Codd, matematician la centrul de cercetări din San Jose (California) al firmei IBM<sup>1</sup>. În acel moment, tehnologia bazelor de date era centrală pe modelele *ierarhic* și *rețea*, modele ce depind într-o mai mare măsură de organizarea internă a datelor pe suportul de stocare. Codd a propus o structură de date tabelară, independentă de tipul de echipamente și software de sistem pe care este implementată, structură „înzestrată” cu o serie de operatori pentru extragerea datelor. Deși puternic matematizat, modelul relațional este relativ ușor de înțeles.

Dar înainte de a intra în subiect, vom cheltui câteva pagini cu teoria referitoare la noțiunile: fișiere de date, baze și bânci de date, sisteme de gestiune a bazelor de date.

#### 1.1. Baze de date și sisteme de gestiune a bazelor de date

Conceptul de „bază de date” a apărut în a două parte a anilor '60. La momentul respectiv, în sistemele informative ale organizațiilor (atâtă căte erau – vorbesc de sisteme informative), informațiile erau organizate în fișiere de date (secvențiale, indexate etc.), create cu ajutorul unor programe scrise în limbaje din a III-a generație: COBOL, Fortran etc. (figura 1.1).

Specific acestui mod de lucru, cunoscut ca *file-based* sau *flat files* (fișiere independente), este faptul că fiecare dată (Data1, Data2,... DataN) este descrisă (nume, tip, lungime), autonom, în toate fișierele în care apare. Mai mult, descrierea fiecăruia fișier de date (câmpurile care-l alcătuiesc, tipul și lungimea fiecăruia, modul de organizare – secvențial, indexat, relativ etc.) este obligatorie în toate programele care îl „citesc” sau modifică. Între FIŞIER1, FIŞIER2, ... FIŞIERn nu există nici o relație definită explicit.

Spre exemplu, Data2 este prezentă în două fișiere de date, FIŞIER1 și FIŞIER2. Dacă, prin program, se modifică formatul sau valoarea acesteia în FIŞIER1, modificarea nu se face automat și în FIŞIER2; prin urmare, o aceeași dată, Data2, va prezenta două valori diferite în cele două fișiere, iar necazurile bat la ușă: informațiile furnizate de sistemul informatic sunt redundante și prezintă un mare risc de pierdere a coerentei.

<sup>1</sup>. Extrase din lucrarea [Codd70] se găsesc la adresa <http://www.acm.org/classics/nov95/>. De asemenea, o excelentă analiză a articolelor lui Codd este în [Date98].

Întrucât am reușit (până acum) să imbin lucrul la catedră și munca patriotică din facultate cu o serie de contracte/participări la realizarea unor aplicații (nu puține au fost eșecurile), îndrăznesc să o recomand și practicienilor, celor care dezvoltă aplicații cu baze de date, dar și celor care au baze de date implementate de ceva timp și vor să obțină tot soiul de informații și nu au la dispoziție un informatician sau sunt certați cu acesta.

O lucrare serioasă capată cu adevărat prestigiu dacă prezintă se termină cu mulțumiri. Întrucât aveam de acasă, ca și laureajii premiilor Oscar, o listă cu nume, dar am rătăcit-o pe undevo, nu-mi rămâne decât să le mulțumesc tuturor celor pe care i-am cunoscut, indiferent de media (vizual/auditiv/e-mail/reviste), relații de rudenie, prietenie sau indiferență, precum și celor pe care li voi cunoaște până la următoarea carte (când sper să le mulțumesc din nou). Unora vreau să le cer scuze dacă plăcerea lor de a mă cunoaște nu a fost la înălțimea plăcerii mele de a-i cunoaște. Timpul nu e pierdut. Ce-ați zice să mai încercați o dată, cu cartea aceasta în față?

Mă simt obligat să mărturisesc că o parte din materialul bibliografic parcurs în anii începuturilor mele în SQL (și acum învăț SQL, deși nu mai am sporul de altădată) a fost preluat de pe Internet, precum și prin fondurile Grantului 376 finanțat de Banca Mondială și Guvernul României. Mulțumesc fondatorilor Internet-ului și directorului grantului cu pricina – care, întâmplător, este o persoană pe care o cunosc binișor.

Autorul  
Iași, aprilie 2001

## CONTINUTUL LUCRĂRII

Cartea începe destul de anot, cu inevitabilă terorie legată de bazele de date: cum au apărut, la ce sunt folosite, care sunt componentele unui sistem de lucru cu bazele de date, ce este un sistem de gestiune a bazelor de date (SGBD) și care îl sunt modulele principale etc. Pentru a mai atenua din durerile „teoretice”, a fost inserată și o istorie română a SGBD-urilor relaționale. Aceasta, de fapt, pentru a pregăti un nou calup teoretic, cel al noțiunilor modelului relațional și al restricțiilor ce pot fi definite pentru o bază de date. Din păcate, aceste paragrafe chiar trebuie citite, deoarece sintagmele explicate aici sunt folosite în capitoile următoare. Spre finalul primului capitol este prezentată în detaliu baza de date „mărtor” a lucrării, ceea ce pare vom opera comenzi SQL.

Al doilea capitol este jumătate teoretic și restul practic. Teoretic, deoarece prezintă coordonatele esențiale ale algebrei relaționale, limbaj curat... teoretic ce fundamentează SQL. Sunt prezentate în extenso operatorii ce permit extragerea informațiilor dintr-o bază de date, cei asamblăți – reunirea, intersecția, diferența și produsul cartezian – și cei relaționali – selecția, proiecția, joncționarea (de fapt joncționurile, că sunt mai multe tipuri) și diviziunea. Aerul practic se datorează exemplelor care însoțesc fiecare operator prezentat, exemple ce utilizează baza de date descrisă în capitolul 1.

Al treilea capitol este și cel care intră – și adevărat, nu chiar frontal – în problematica SQL. Nu chiar frontal, deoarece începe, aproape inevitabil pentru o carte scrisă de un universitar, cu o scurtă istorie a limbajului. Cu acest prilej sunt conturate și direcțiile de evoluție ale SQL. După enumerarea principalelor tipuri de date gestionabile prin SQL, discuția se concentrează asupra modalităților de creare a tabelelor și mai ales de declarare (cu grec am reușit să evit cuvântul *implementare*) a restricțiilor unei baze de date. Formatul general din standartul SQL-92 este permanent însoțit de variantele operaționale în DB2, Oracle și Visual FoxPro, lucru ce se va repeta în capitoile ce vor urma.

Apropo, de ce DB2, Oracle și Visual FoxPro și nu alte produse? În primul rând, pentru că sunt singurle dialecte SQL pe care le știu (cât de căt). În celelalte rânduri, pentru că:

- DB2 este produsul din care primele două standarde SQL au imprumutat cel mai mult; de altminteri, IBM este probabil firma-cheie atât în ceea ce privește apariția modelului relațional, cât și SQL.
- Oracle definește, de ani buni, cea mai mare cotă din piața bazelor de date și nu sunt semne vizibile că ar trece curând pe locul doi.
- Visual FoxPro este primul SGBD „democrat” din țara noastră – și aceasta nu ține atât de faptul că a apărut și a fost utilizat după 1990, că mai ales de aria sa de utilizare în țara noastră, incomparabil mai mare decât pe plan mondial. Dacă, aşa cum spuneau cei din Școala Ardeleană, *de la Rim ne tragem*, se poate spune că și cel puțin o generație de dezvoltatori de aplicații cu baze de date din FoxPro se trage... Este drept că VFP nu stă chiar pe roze, ca evoluție mondială a numărului de dezvoltatori ce-l utilizează, dar mai este, pentru multe firme mici și mijlocii, o soluție ce nu trebuie ridiculizată.

Iar pe de altă parte, n-am reușit să rezist tentației de a alătura (numai într-o carte) trei produse ale unor firme (IBM, Oracle și Microsoft) între care relațiile sunt cordiale o dată sau de două ori pe deceniu.

## CUVÂNT, OARECUM, ÎNAINTE

Un amic, al căruia succes publicistic îl invidiez de ani buni, m-a convins că, uneori, titlul este la fel de important ca și conținutul unei cărți. Distinsul „cetitor” care intră în librărie sau anticariat, hotărât să nu se atingă de bruma din portmoneu, trebuie capturat cu orice preț de o copertă viu colorată, de un titlu sprințar și convins să cumpere ori, în cel mai râu caz, poate pleca chinuit de păcatul de a fi lăsat cartea în raft (sentimentul se numește, în termeni tehniici, frustrare).

Astfel, prima mea grija a fost găsirea unui titlu strănic. *SQL pentru to(n)ți* ar fi fost prea jignitor și,oricum, nerealist. Nu cred că poți să SQL și să fil, simultan, tot. Vă asigur că dacă vă descurcați cu SQL la un nivel acceptabil, sunteți oricum, dar nu și tot.

Nici *Total despre SQL* nu mi s-a părut grozav. În primul rând, deoarece a fost deja folosit de Corina și Adrian Pascu. Apoi, ar fi fost o exagerare. Un tratat complet despre SQL s-ar intinde pe câteva mii de pagini și, chiar și aşa, ar mai rămâne destule puncte neatinse sau tratate superficial, motiv pentru care mi-am reprimat urgent bruma de vocație academică.

Anxious din cale-afără, am profitat de primul vin roșu ieșit în cale și mi-am provocat prietenii la un *wine-storming*. Le-am povestit problema și pe data au inceput săurgă sugestiile, criticele și propunerile. *Olecuță de SQL* e prea moldovenesc. Polirom este o editură cu o bătăie lungă și largă și nu s-ar cuveni o regionalizare a jării tocmai pe criterii SQL-istice.

O primă propunere serioasă a fost *Esență de SQL*. Mi-a plăcut, dar parcă prea sună a elită – or, cel puțin prin titlu, carteabu să se adreseze maselor largi de studenți, informaticieni și altora dispuși sau silicii să lucreze cu bazele de date. Încălziți fiind, ne-am amintit de prestația lui Al Pacino din *Parfum de femeie*, așa că următoarea idee a fost *Parfum de SQL*. Am realizat că exageraserem, deoarece unui cărtitor versat i-ar fi trecut prin cap idea că mai bine sună *Aftershave de SQL*.

Cum apucaseam să mă laud, cu câteva luni în urmă, cu rubrica din *Net Report* (ex-PC Report), unul dintre mesenii a sugerat ca titlul cărții să fie chiar *Fiola de SQL*. Există deja câteva *mii* (zeci!, l-am corectat eu) de posibili cătitori care au avut de această expresie. M-am opus net (report), din mai multe motive. Nu mi s-a părut corect față de cei de la revistă (acum mă suspectez că am avut în vedere și o eventuală culegere de fioli într-o altă carte, idee care, la acest moment, nu mi pare chiar realistă). Și apoi, unde s-a mai pomenit ca o fiolă să se întândă pe câteva sute de pagini? Poate *Butoiul cu SQL* – dar și asta sună ca un atac (bahic) la persoana studentului/informaticianului/specialiștului/cumpărătorului.

...și uite așa am ajuns la prea puțin sofisticatul titlu *SQL*. Pur și simplu. Adio regionalisme, esențe, deodorante și fioli. Pentru a mai adăuga un pic de semnificație și a întări câteva categorii clare de utilizatori/cumpărători ai cărții, am mai adăugat subtitul *Dialecte DB2, Oracle și Visual FoxPro*. Asta e. Să nu ziceți că titlul n-are imaginea. N-ăs suportă lovitura...

Carteaua de față este destinată unor grupuri de utilizatori ce folosesc sau sunt pe cale de a folosi bazele de date. Pentru studenții specializațiilor Informatică Economică și Birotică (de la Facultatea de Economie și Administrația Afacerilor) ar putea fi un ajutor neprejurit în efortul supraomenesc de a promova anul de studii în care apare disciplina *Baze de date* (glumeam, cred că am, în rândul studenților, o rată de sinucideri care l-ar dezamăgi pe Cioran).

Studenții de alte facultăți (Informatică, Cibernetică, Calculatoare) o pot folosi chiar dacă-i supără pe profesorii lor titulari.