# Diviziunea relațională

## Teorie și realități SQL în fiola de SQL (9)

– *Marin Fotache*

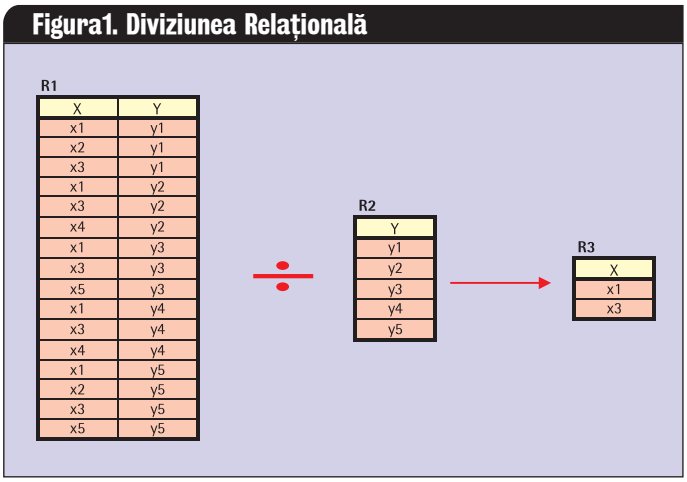
Cel mai interesant operator al algebrei relaționale este destul de bine ocolit în multe dintre lucrările apărute la noi care tratează problematica interogării bazelor de date relaționale. În alte cazuri este expediat rapid, eventual cu un exemplu eronat. Articolul de față este unul de popularizare, de demitizare a operatorului și de prezentare a câtorva dintre soluțiile SQL ce corespund diviziunii relaționale.

### Să ne formalizăm un pic

Diviziunea este cel mai complex și mai greu de explicat dintre operatorii relaționali. Codd l-a imaginat ca pe un operator invers produsului cartezian. În ceea ce ne privește, pentru a-l defini, pornim de la două relații `R1(X,Y)` și `R2(Y)`; prima are două atribute sau grupuri de atribute, notate `X` și `Y`, în timp ce a doua numai atributul sau grupul de atribute notat cu `Y` (definit pe același domeniu ca și în relația `R1`).

O primă restricție: relația `R2(Y)`, fiind numitorul diviziunii, nu poate 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 câtul diviziunii, fiind o relație notată `R3(X)`. Într-o altă formulare, $x_i \in R3$ dacă și numai dacă $\forall\ y_j \in Y \in R2 \rightarrow \exists (x_i, y_j) \in R1$. Să examinăm elementele din figura „Diviziunea Relațională".

**Figura1. Diviziunea Relațională**




Determinarea relației R3← 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 $y_5$?

Se 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 `y1` - *nu* va face parte din `R3`.

În urma raționamentului de mai sus, tabela `R3` va fi alcătuită din două tupluri. Ca element de verificare, produsul cartezian dintre câtul relațional și divizor trebuie să obțină un ansamblu de linii inclus în relația-deîmpărțit.

Deși pare un operator ceva mai abstract, diviziunea relațională este deosebit de utilă pentru formularea consultărilor în care apare clauza „∀" („oricare ar fi" sau „pentru toate").

### Exemple în algebra relațională

Pentru exemplele prezentate în acest material folosim următoarea schemă a bazei de date:

```
CLIENȚI {CodCl, DenCl, CodPost}
FACTURI {NrFact, DataFact, CodCl, Valoare}
```

**Exemplul 1** - *Care sunt clienții pentru care există cel puțin câte o factură emisă în fiecare zi?*

Într-o altă formulare, interesează *clienții* care au cumpărat „câte ceva" în *toate zilele* în care s-au efectuat vânzări ? Prin urmare, câtul va fi o tabelă cu un 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). Mergând pe calapodul prezentat, am putea nota R3(`DenCl`) și R2(`DataFact`). Cunoscând structura câtului și a divizorului, putem determina structura tabelei-deîmpărțit: R1(`DenCl, DataFact`). Relația R1 va conține denumirile clienților și zilele în care există măcar o factură pentru clientul respectiv, fiind obținută prin joncționarea tabelelor `FACTURI` și `CLIENȚI`.

Soluția poate fi redactată în următorii pași:

• construire relație-deîmpărțit:

```
R11← JONCȚIUNE (FACTURI, CLIENTI; CodCl)
R1 ← PROIECȚIE (R1; DenCl, DataFact)
```

• construire relație-numitor:

```
R2 ← PROIECȚIE (FACTURI;  DataFact)
```

• în fine, rezultatul:

```
R3 ←  R1  ÷  R2
```

Schema și conținutul relațiilor implicate în această soluție sunt prezentate în figura „Diviziunea relațională – exemplul 1".

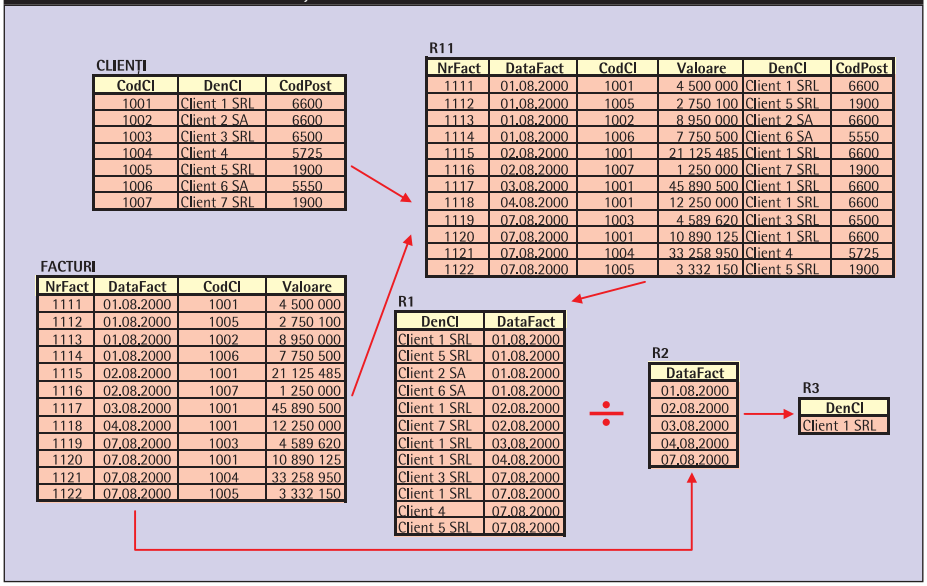
**Exemplul 2** - *Căror clienți li s-au emis facturi măcar în zilele în care s-au făcut vânzări Clientului 5 SRL ?*

Aceasta este o altă interogare care face necesară utilizarea diviziunii relaționale. Relația-cât conține denumirile clienților (`DenCl`) care îndeplinesc condiția, iar relația-divizor prezintă o singură coloană (`DataFact`) și un număr de linii ce reprezintă numărul zilelor în care s-au făcut vânzări clientului 5.

```
R11← JONCȚIUNE (CLIENȚI, FACTURI;  CodCl)
R1 ← PROIECȚIE (R11;  DenCl, DataFact)
R21← SELECȚIE (CLIENȚI;  DenCl = 'Client 5 SRL')
R22← JONCȚIUNE (R21, FACTURI;  CodCl)
R2 ← PROIECȚIE (R12;  DataFact)
R3 ←  R1  ÷  R2
```

**Realizarea diviziunii prin ceilalți operatori ai algebrei relaționale.** Diviziunea relațională nu este un operator fundamental; funcționalitatea sa

**Figura 2. Diviziunea relațională – exemplul 1**




poate fi realizată prin combinarea operatorilor: produs cartezian, diferență și proiecție.

Reluăm în discuție atât tabelele R1 și R2 din figura 1, 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 ← PROIECȚIE (R1; X)
- R12 ← R11 ⊗ R2. Tabela R12 cuprinde toate tuplurile posibile ($x_i$, $y_j$)
- R13 ← R12 - R1. Interesează ce tupluri din R12 lipsesc în R1
- Valorile xi din R13 nu apar în R1 în combinație cu toate valorile $y_j$ din R2. Se elimină dublurile prin R14 ← PROIECȚIE (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 diferența R3←R11-R14.

Figura 3 aduce, cât de cât, un plus de claritate.

Standardele SQL elaborate până în prezent nu conțin vreun operator special dedicat diviziunii. Cu toate acestea, există destule variante de rezolvare a unor probleme ce impun soluții de acest gen, după cum vom vedea în cele ce urmează.

**Soluții bazate pe GROUP BY și HAVING.** Dacă ne raportăm la cele două relații „teoretice" din figura 1, aflarea tuturor icșilor din R1 care apar în combinație în R1 cu toți igrecii din R2 este posibilă printr-o variantă de genul:

```
SELECT X
FROM R1
GROUP BY X
HAVING COUNT(*) =
   (SELECT COUNT(Y)
    FROM R2)
```

Pe acest calapod se formulează o soluție pentru exemplul 1 (Care sunt clienții pentru care există cel puțin câte o factură emisă în fiecare zi?):

```
SELECT DenCl
FROM CLIENTI C INNER JOIN FACTURI F ON C.CodCl=F.CodCl
GROUP BY DenCl
HAVING COUNT(DISTINCT DataFact) =
   (SELECT COUNT(DISTINCT DataFact)
    FROM FACTURI)
```

Aceste două interogări SQL92 funcționează în mai toate SGBD-urile de tip servere de baze de date (cu unele mici modificări - în Oracle, spre exemplu, nu există operatorul `INNER JOIN`), nu însă și în Visual FoxPro, deoarece în acest produs, și în multe SGBD-uri din categoria sa, nu sunt permise subconsultări în clauza `HAVING`.

Cât privește răspunsul la problema de la exemplul 2, interogarea următoare extrage din toate facturile emise numai zilele în care s-au făcut vânzări *Clientului 5 SRL*; prin gruparea acestor linii pe clienți și compararea, pentru fiecare grup, a numărului de zile cu numărul de zile de vânzare corespunzătoare clientului-reper, se obține rezultatul dorit.

```
SELECT DenCl
FROM CLIENTI C INNER JOIN FACTURI F ON
C.CodCl=F.CodCl
WHERE DataFact IN
    (SELECT DISTINCT DataFact
     FROM CLIENTI C INNER JOIN FACTURI
F ON C.CodCl=F.CodCl
    WHERE DenCl ='Client 5 SRL')
GROUP BY DenCl
HAVING COUNT(DISTINCT DataFact) =
   (SELECT COUNT(DISTINCT DataFact)
   FROM CLIENTI C INNER JOIN FACTURI F ON C.CodCl=F.CodCl
   WHERE DenCl ='Client 5 SRL')
```

**Diviziune prin diferență, joncțiune externă și subconsultări.** Și cu ajutorul operatorului `IN` (și `NOT IN`) se poate aborda problema *diviziunii* relaționale în SQL. Succesiunea de pași prezentată în figura 3 se realizează relativ simplu prin soluția SQL92/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, limpede, tonic și explicit ca al unui funcționar de la ghișeele Oficiilor Forțelor de Muncă: `SQL: Queries of this type are not supported (Error 1814).` Ce-i de făcut ? Salvarea vine de la joncțiunea externă:

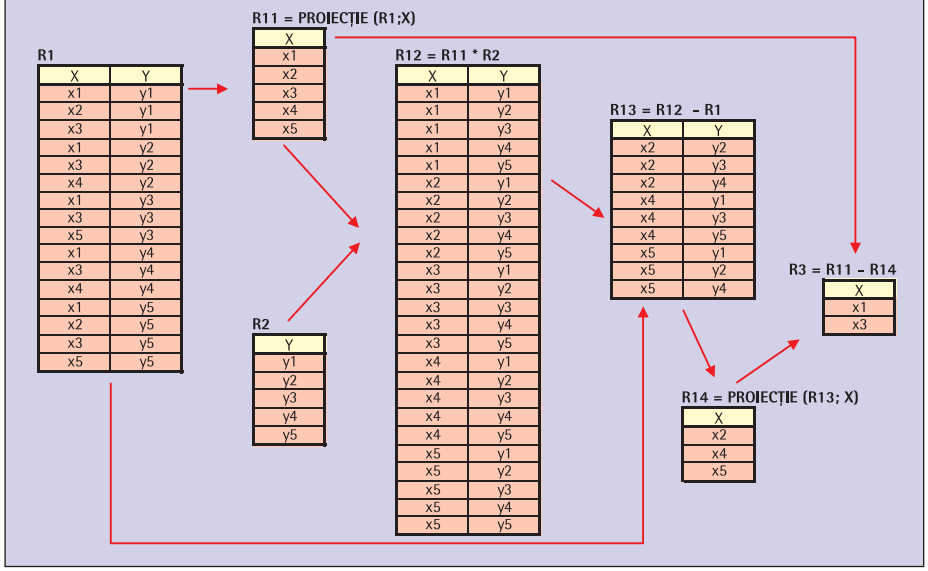
```
SELECT DISTINCT X ;
FROM R1 ;
WHERE X+'  ' NOT IN  ;
   (SELECT R1_1.X + NVL(R1_2.X,'  ') ;
   FROM R1 R1_1 ;
INNER JOIN R2 R2_1 ON 1=1 ;
        LEFT OUTER JOIN R1 R1_2  ;
ON R1_1.X=R1_2.X AND R2_1.Y=R1_2.Y)
```

Pseudo-joncțiunea internă este de fapt un produs cartezian, deoarece condiția de joncțiune este 1=1. Subconsultarea extrage icșii care au „goluri”, iar fraza SELECT principală efectuează scăderea lor din icșii tabelei R1.

Echivalent cu această ultimă soluție, se poate formula o alta DB2/Oracle ce folosește atât joncțiunea externă, cât și subconsultarea în clauza FROM:

```
SELECT DISTINCT X
FROM R1
WHERE X NOT IN
   (SELECT DISTINCT PROD_CART.X
   FROM (SELECT 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)
```

**Figura 3. Simularea diviziunii prin alți operatori relaționali**




Mai merită menționată și o solutie ceva mai punctuală (valabilă numai în DB2) care beneficiază de suportul expresiilor-tabelă:

```
WITH
PRODUS_CARTEZIAN AS
   (SELECT DISTINCT T1.X, T2.Y
    FROM T1, T2)
SELECT  DISTINCT X
FROM T1
WHERE X NOT IN
   (SELECT DISTINCT X
   FROM PRODUS_CARTEZIAN
   WHERE (X,Y) NOT IN
        (SELECT X, Y
        FROM T1) )
```

Revenind la exemplul 1, pe baza logicii de mai sus se pot formula următoarele interogări:

• Soluția DB2 (și Oracle, cu MINUS-ul de rigoare):

```
SELECT DenCl
FROM CLIENTI
WHERE CodCl IN
   (SELECT CodCl
    FROM CLIENTI C
   EXCEPT
        SELECT DISTINCT C.CodCl
        FROM CLIENTI C, FACTURI F1, FACTURI F2
        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 1=1 ;
    LEFT OUTER JOIN FACTURI F3 ON  ;
        F1.CodCl=F3.CodCl AND
        F2.DataFact=F3.DataFact)
```

Cât privește exemplul 2, prezentăm numai o soluție DB2 care, totuși, poate fi transcrisă ușor în sintaxa Oracle:

```
SELECT DenCl
FROM CLIENTI INNER JOIN FACTURI ON
   CLIENTI.CodCl=FACTURI.CodCl
EXCEPT
SELECT DISTINCT R1.DenCl
FROM
(SELECT DISTINCT DenCl
FROM CLIENTI INNER JOIN FACTURI
ON CLIENTI.CodCl=FACTURI.CodCl) R1,
(SELECT DISTINCT DataFact
FROM CLIENTI INNER JOIN FACTURI
ON CLIENTI.CodCl=FACTURI.CodCl
WHERE DenCl='Client 5 SRL') R2
WHERE (R1.DenCl, R2.DataFact) NOT IN
        (SELECT DenCl, DataFact
        FROM CLIENTI INNER JOIN FACTURI ON
CLIENTI.CodCl=FACTURI.CodCl)
```

**Diviziunea prin interogări corelate.** În fiola SQL de luna trecută a fost prezentat pe îndelete mecanismul interogărilor corelate, utilizând cu preponderență operatorul EXISTS. Fără a mai explicita analitic logica operațiunilor, prezentăm câteva soluții de acest gen. Începem cu cele două relații R1 și R2, soluția SQL92/DB2/Oracle fiind:

```
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 acest model se construiesc interogări-răspuns la exemplele formulate.

Sunt corelate două instanțe ale joncțiunii CLIENȚI-FACTURI; prima conține liniile legate de Client 1 SRL, iar a doua de Client 2 SA. Întrucât interesează zilele în care s-au întocmit facturi pentru ambii clienți, atributul de corelare este DataFact.

## Diviziunea și dubla corelare în SQL

Dubla corelare constituie o facilitate prea puțin folosită în aplicațiile cu baze de date. 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 Sheryl am găsit cea mai bună explicație a acestui mecanism.

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 !

Revenim la cele două tabele, R1 și R2. 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=R2.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 „avansează" 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 „avansează" 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=y4 (R2.Y). Există !, deci se „avansează" 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: Întrucât am ajuns la sfârșitul tabelei 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 pasează această valoare logică sau contrara sa interogării principale. În cazul nostru 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 preluarea în rezultat a fiecărei valori o singură dată.

Dacă privim mai îndeaproape interogarea, observăm că, de fapt, aceasta răspunde la întrebarea: pentru care dintre valorile lui X nu există nici un Y care să nu apară în combinație cu X-ul respectiv în R1?

Apelând la dubla corelare, pentru exemplul 1 se poate redacta o soluție precum:

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

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 zilele…"* pentru corelarea mijloc (middle) – jos (bottom) se utilizează atributul DataFact.

## Concluzii

Există numeroase situații informaționale pentru rezolvarea cărora operatorul diviziune relațională se dovedește a fi grozav de util. În prezentul articol au fost expuse câteva probleme și tipuri de soluții legate de acest operator. Ar mai fi de spus că, pe vremuri, în System R (primul SGBDR) al IBM, exista pentru realizarea diviziunii și operatorul CONTAINS. Nefiind luat în considerare de nici unul dintre standardele SQL, și aproape ignorat în produsele ce domină astăzi piața, l-am trecut sub tăcere.

*Marin Fotache este conferențiar la Catedra de Informatică Economică, UAIC Iași, Facultatea de Economie și Administrarea Afacerilor. Poate fi contactat pe email la: fotache@uaic.ro*

## Bibliografie

[Larsen98] Larsen, S.
– *The SQL Double Double*, DB2 Magazine On Line, Spring 1998 ■ 14