Capitolul 10. Subconsultări corelate

Toate subconsultările din capitolul anterior aveau regim de autonomie, ca să nu-i zicem izolare. Clauzele SELECT, WHERE şi HAVING făceau apel doar la tabele enumerate în clauza FROM sau la expresii-tabelă (clauza WITH). Vom vedea, în cele ce urmează, că putem referi, într-o subconsultare, atribute ale unor tabele din consultarea principală (sau subconsultări superioare), un mare avantaj care însă va da şi destule bătăi de cap.

Sunt unul dintre cei care cred că piatra de încercare a unui SQL-ist o constituie subconsultările corelate. Nu atât din lipsa alternativelor, întrucât mai toate rezolvările din acest capitol au soluții echivalente formulate sau sugerate în capitolele anterioare. Cel mai deconcertant aspect al interogărilor corelate este logica execuției acestora, care este una linie cu linie, nu ansamblistă, aşa cum am fost obişnuiți până acum. Maximul de adrenalină este rezervat subconsultărilor dublu corelate tratate în ultimul paragraf.

10.1. Subconsultări corelate în clauza SELECT

Am făcut cunoştință cu interogările scalare incluse în clauza SELECT în paragraful 9.5. Acum onorăm promisiunea făcută în finalul acelui paragraf, discutând despre subconsultări scalare incluse în clauza SELECT care conțin referințe către tabelele din clauza FROM a interogării principale.

Revenim, mai întâi, la o problemă simplă din paragraful 4.4.3, exemplul 9 (vezi figura 4.16) - *Care sunt codurile produselor care apar simultan şi în factura 1111 şi în factura 1117 ?* Alăturăm soluțiilor bazate pe intersecție, joncțiune şi subconsultări (necorelate), una care nu este pe deplin satisfăcătoare, însă interesantă ca idee:

```
SELECT lf1.CodPr,
       (SELECT CodPr FROM liniifact lf2
        WHERE lf2.CodPr=lf1.CodPr AND lf2.NrFact=1117) AS CodPr2
FROM liniifact lf1
WHERE NrFact = 1111
```

Prin clauzele FROM şi WHERE ale interogării principale vor fi selectate liniile facturii 1111 sub aliasul LF1. Prima coloană din rezultat conține codurile celor trei produse 1, 2 şi 5 (vezi figura 10.1). Pentru fiecarea dintre cele trei valori (pentru fiecare linie a rezultatului) se execută subconsultarea care se corelează cu fraza SELECT principală prin condiția *lf2.CodPr=lf1.CodPr*. Practic, pentru fiecare linie a rezultatului, ce corespunde codurilor produselor 1, 2 şi 5, subconsultările sunt:

- **SELECT CodPr FROM liniifact lf2 WHERE lf2.CodPr=1 AND lf2.NrFact=1117**
- **SELECT CodPr FROM liniifact lf2 WHERE lf2.CodPr=2 AND lf2.NrFact=1117**
- **SELECT CodPr FROM liniifact lf2 WHERE lf2.CodPr=5 AND lf2.NrFact=1117**

Prima „returnează” (deşi nu e funcție) 1, a doua 2, iar a treia nimic, adică NULL.

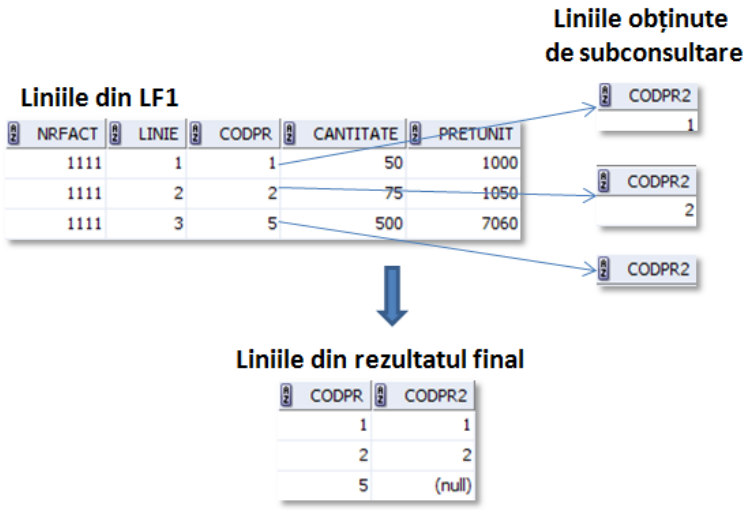
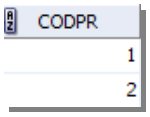


Figura 10.1. O subconsultare corelată în clauza SELECT

Rezervele legate de interogare se datorează faptului că rezultatul conține două coloane, în prima fiind codurile produselor care apar în factura 1111, iar în a doua, aceleaşi coduri, dacă produsele respective apar şi în factura 1117 (produsele 1 şi 2), sau NULL, în caz contrar (produsul 5). Prin urmare, pentru a răspunde punctual la întrebare, trebuie să mai facem o filtrare, eliminând produsele la care, pe coloana a doua, apare NULL. Filtrarea este pur vizuală întrucât, după cum am văzut în paragraful 9.5, coloana definită prin subconsultarea scalară în clauza SELECT nu poate fi referită în clauza WHERE, deci (vorba lui Cristoiu!) interogarea *SELECT lf1.CodPr, (SELECT CodPr FROM liniifact lf2 WHERE lf2.CodPr=lf1.CodPr AND lf2.NrFact=1117) AS CodPr2 FROM liniifact lf1 WHERE NrFact = 1111 AND CodPr2 IS NOT NULL* nu funcționează. Pentru a nu înregistra un nou eşec, includem interogarea funcțională de mai sus (ce conține subconsultarea scalară) în clauza FROM a unei interogări în care putem face filtrarea, rezultatul fiind cel din figura 10.2:

```
SELECT CodPr
FROM    (SELECT lf1.CodPr, (
                              SELECT CodPr
                              FROM liniifact lf2
                               WHERE lf2.CodPr=lf1.CodPr AND lf2.NrFact=1117
                                  ) AS CodPr2
            FROM liniifact lf1
            WHERE NrFact = 1111      ) t
WHERE CodPr2 IS NOT NULL
```



CODPR
1
2

Figura 10.2. Codurile produselor ce apar şi în factura 1111 şi în factura 1117

Prin comparație cu simplitatea problemei, complexitatea interogării poate părea, totuşi, exagerată. Pentru a recicla un pic codul, putem folosi această soluție pentru un exemplu legat de diferență:

Care sunt codurile produselor care apar în factura 1111, dar nu apar în factura 1117 ?

Singura modificare a interogării anterioare este eliminarea lui NOT de pe ultima linie:

```
SELECT CodPr
FROM    (SELECT lf1.CodPr, ( SELECT CodPr FROM liniifact lf2
                                WHERE lf2.CodPr=lf1.CodPr
                                  AND lf2.NrFact=1117
                             ) AS CodPr2
          FROM liniifact lf1
         WHERE NrFact = 1111 ) t
WHERE CodPr2 IS NULL
```

Care sunt localitățile în care se află sediul fiecărui client ?

Dacă precedentele două probleme erau legate de intersecție şi diferență, acesta este unul apropiat de joncțiune. Important este că tabela din clauza FROM a interogării principale este „copil", astfel încât subconsultarea extrage întotdeauna o singură linie (şi coloană) – vezi figura 10.3.

```
SELECT DenCl,
       (SELECT Loc
        FROM coduri_postale
        WHERE CodPost=clienti.CodPost
       ) AS Loc
FROM clienti ORDER BY 1
```

DENCL	LOC
Client 1 SRL	Iasi
Client 2 SA	Iasi
Client 3 SRL	Vaslui
Client 4	Pascani
Client 5 SRL	Timisoara
Client 6 SA	Roman
Client 7 SRL	Timisoara

Figura 10.3. Localitatea fiecărui client, obţinută printr-o subconsultare corelată

Care este valoarea vânzărilor din fiecare produs pentru luna septembrie 2007 ?

Având experienţa interogărilor anterioare, presupunem că interogarea următoare nu va întâmpină nicio rezistenţă[1]:

```
SELECT DenPr AS Produs,
        ROUND(
                COALESCE (
                        (SELECT SUM(Cantitate * PretUnit * (1+p1.ProcTVA))
                         FROM liniifact INNER JOIN facturi
                                 ON liniifact.NrFact=facturi.NrFact
                         WHERE DataFact BETWEEN '2007-09-01' AND '2007-09-30'
                                 AND CodPr=p1.CodPr
                        ), 0)
                ,0) AS Vinzari_Sept2007
FROM produse p1
ORDER BY DenPr
```

Aşa se şi întâmplă în PostgreSQL, Oracle (cu modificările legate de DATE) şi DB2 – vezi figura 10.4.

PRODUS	VINZARI_SEPT2007
Produs 1	324,989.0000
Produs 2	771,393.0000
Produs 3	58,013.0000
Produs 4	0.0000
Produs 5	4,284,714.0000
Produs 6	0.0000

Figura 10.4. Valoarea vânzărilor din sept. 2007 pentru fiecare produse, obţinută printr-o subconsultare corelată (DB2)

SQL Serverul ne cadoriseşte însă cu o eroare destul de tulburătoare – vezi figura 10.5. Dacă ar fi sa traducem mesajul, s-ar înţelege că în subconsultarea corelată singurul atribut extern (din tabele ale clauzei FROM din interogarea

[1] În Oracle trebuie adăugat cuvântul DATE înaintea constantelor de tip dată calendaristică.

principală) ce poate fi folosit în expresia SUM (...) ar fi cel după care se face corelarea, adică p1.CodPr...

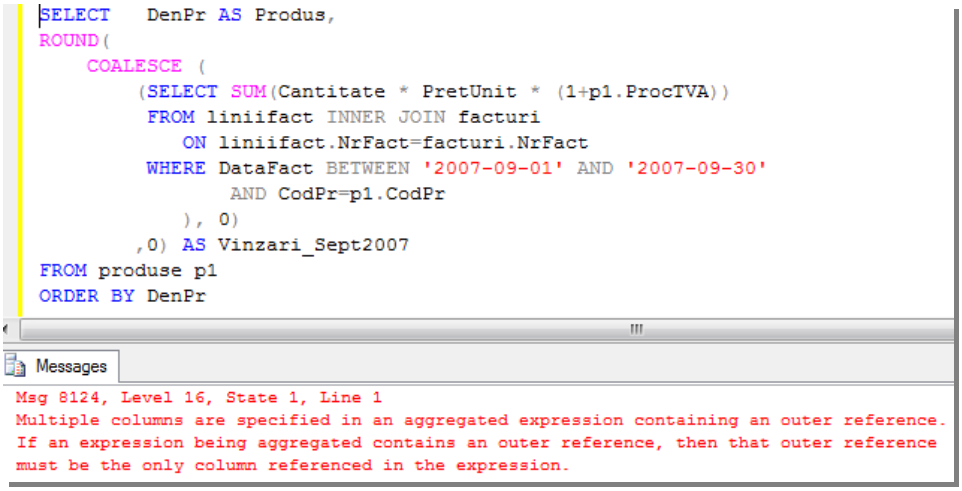


Figura 10.5. Eroare SQL Server la folosirea interogărilor corelate

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

Ne vom servi de o subconsultare scalară pentru a determina, pe fiecare linie (corespunzătoare unui produs) totalul vânzărilor, şi a calcula, astfel, raportul care interesează – vezi figura 10.6.

```
SELECT DenPr AS Produs,
        COALESCE(
                (SELECT SUM(Cantitate * PretUnit * (1+ProcTVA))
                 FROM liniifact lf
                 WHERE lf.CodPr=produse.CodPr
                ), 0)  AS Vinzari_Produs,
         (SELECT SUM(Cantitate * PretUnit * (1+ProcTVA))
          FROM liniifact lf INNER JOIN produse p ON lf.CodPr=p.CodPr
          ) AS Total_Vinzari,
          ROUND( COALESCE(
                          (
                (
        ( SELECT SUM(Cantitate * PretUnit * (1+ProcTVA))
         FROM liniifact lf
         WHERE lf.CodPr=produse.CodPr
        )
         * 100     )  /        (
                  SELECT SUM(Cantitate * PretUnit * (1+ProcTVA))
```

```
        FROM liniifact lf INNER JOIN produse p ON lf.CodPr=p.CodPr
                )
                        ),0 ), 2 )    AS Procent
FROM produse
ORDER BY DenPr
```

PRODUS	VINZARI_PRODUS	TOTAL_VINZARI	PROCENT
Produs 1	988533	49170335	2,01
Produs 2	2784023,5	49170335	5,66
Produs 3	251685	49170335	0,51
Produs 4	301657,5	49170335	0,61
Produs 5	44844436	49170335	91,2
Produs 6	0	49170335	0

Figura 10.6. Contribuţia fiecărui produs la totalul vânzărilor

Din nou SQL Server-ul se opune la acest gen de corelare, mesajul afişat fiind similar celui de la problema precedentă.

Să se afişeze numărul curent al fiecărei facturi emise în luna septembrie 2007.

Acest gen de problemă îşi are cea mai elegantă rezolvare prin folosirea funcţiei ROW_NUMBER(), una dintre funcţiile OLAP prezentate în capitolul următor. Până atunci, însă, putem formula o interogarea simplă bazată pe o subconsultare în clauza FROM. Soluţia este cu atât mai importantă pentru PostgreSQL în condiţiile în care în acest produs nu (prea) au fost implementate funcţii OLAP[2] şi nici opţiunea ROWNUM din Oracle. Ideea este următoare: prin clauzele FROM şi WHERE sunt selectate numai facturile din sept. 2007. Pentru fiecare linie (factură din septembrie) se vor număra, printr-o subconsultare (scalară şi corelată) din clauza SELECT câte facturi din septembrie au numărul mai mic decât cel al facturii curente (linia curentă din f2):

```
SELECT (
        SELECT COUNT(*) FROM facturi f2
        WHERE EXTRACT (YEAR FROM DataFact)=2007    AND EXTRACT (MONTH
                        FROM DataFact)=9 AND NrFact <= f1.NrFact
```

[2] Idee preluată din http://www.slideshare.net/xzilla/pro-postgresql-389802/

```
    ) AS NrCrt_Sept07, f1.*
FROM facturi f1
WHERE EXTRACT (YEAR FROM DataFact)=2007 AND
          EXTRACT (MONTH FROM DataFact)=9
```

nrcrt_sept07 bigint	nrfact integer	datafact date	codcl smallint	obs character varying(50)
1	3111	2007-09-01	1001	
2	3112	2007-09-01	1005	Probleme cu transportul
3	3113	2007-09-02	1002	
4	3115	2007-09-02	1001	
5	3116	2007-09-10	1007	Pretul propus initial a fost modificat
6	3117	2007-09-10	1001	
7	3118	2007-09-17	1001	

Figura 10.7. Numărul curent al fiecărei facturi din luna sept.2007

Reuşita interogării ne este confirmată de figura 10.7. Înlocuind funcţiile EXTRACT cu YEAR şi MONTH obţinem varianta DB2/SQL Server.

Să se afişeze numărul curent al fiecărei facturi în cadrul lunii în care a fost emisă.

Diferenţa faţă de problema anterioară ţine de *resetarea* numărului curent al facturilor la începutul fiecărei luni calendaristice – vezi figura 10.8.

nrcrt_pt_luna bigint	nrfact integer	datafact date	codcl smallint	obs character varying(50)
1	1111	2007-08-01	1001	
2	1112	2007-08-01	1005	Probleme cu transportul
3	1113	2007-08-01	1002	
4	1114	2007-08-01	1006	
5	1115	2007-08-02	1001	
6	1116	2007-08-02	1007	Pretul propus initial a fost mo
7	1117	2007-08-03	1001	
8	1118	2007-08-04	1001	
9	1119	2007-08-07	1003	
10	1120	2007-08-07	1001	
11	1121	2007-08-07	1004	
12	1122	2007-08-07	1005	
13	2111	2007-08-14	1001	
14	2112	2007-08-14	1005	Probleme cu transportul
15	2113	2007-08-14	1002	
16	2115	2007-08-15	1001	
17	2116	2007-08-15	1007	Pretul propus initial a fost mo
18	2117	2007-08-16	1001	
19	2118	2007-08-16	1001	
20	2119	2007-08-21	1003	
21	2121	2007-08-21	1004	
22	2122	2007-08-22	1005	
1	3111	2007-09-01	1001	
2	3112	2007-09-01	1005	Probleme cu transportul
3	3113	2007-09-02	1002	

Figura 10.8. Numărul curent al fiecărei facturi în luna în care a fost emisă

Logica interogării este aceeaşi, doar că se schimbă expresia de corelare:

```
SELECT (
        SELECT COUNT(*) FROM facturi f2 WHERE EXTRACT (YEAR
                FROM f2.DataFact) = EXTRACT (YEAR FROM f1.DataFact)
                 AND EXTRACT (MONTH FROM f2.DataFact) = EXTRACT (MONTH
                        FROM f1.DataFact) AND f2.NrFact <= f1.NrFact
        ) AS NrCrt_Pt_LunaCrt, f1.*
FROM facturi f1
```

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

Interesează un rezultat precum cel din figura 10.9. Dificultatea principală ține de sincronizarea dintre ziua curentă de vânzări şi ziua precedentă.

ZI	VINZARI_ZI_CURENTA	VINZARI_ZI_PRECEDENTA	DIFERENTA
01-08-2007	10599535	0	10599535
02-08-2007	277950	10599535	-10321585
03-08-2007	222050	277950	-55900
04-08-2007	201975	222050	-20075
07-08-2007	10610000,5	0	10610000,5
14-08-2007	4723931,5	0	4723931,5
15-08-2007	247757	4723931,5	-4476174,5
16-08-2007	467420	247757	219663
21-08-2007	10560939,5	0	10560939,5
01-09-2007	4596401,5	0	4596401,5
02-09-2007	238437,5	4596401,5	-4357964
10-09-2007	424704,5	0	424704,5
17-09-2007	179565	0	179565
07-10-2007	5819668	0	5819668

Figura 10.9. Vânzări zilnice, comparate cu cele ale precedentei zile cu vânzări

Iată soluția Oracle:

```
SELECT DataFact AS Zi,
        SUM(Cantitate * PretUnit * (1+ProcTVA))  AS Vinzari_Zi_Curenta,
        COALESCE        (
                (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 - INTERVAL '1' DAY
                        ),0)  AS Vinzari_Zi_Precedenta,
        SUM(Cantitate * PretUnit * (1+ProcTVA)) -
```

```
        COALESCE(
                (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 - INTERVAL '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
    ORDER BY DataFact
```

Prima interogare scalară furnizează totalul vânzărilor pentru ziua precedentă, ziua curentă fiind cea a grupului de referinţă (din SELECT-ul principal). A doua instanţă a interogării scalare permite calcularea diferenţei dintre vânzările zilei curente şi a celei precedente. Funcţia COALESCE converteşte eventuala valoare NULL datorată inexistenţei vânzărilor în data calendaristică precedentă (aşa cum este cazul zilelor de 1, 7, 14 şi 21 august 2007, 1, 10 şi 17 septembrie şi 7 octombrie) în zero.

Schimbând *INTERVAL '1' DAY* cu *1*, obţinem varianta care funcţionează în MS SQL Server, iar cu *1 DAY* o obţinem pe cea DB2. Lansarea interogării (nemodificate) în PostgreSQL ne dă prilejul unei noi suprize neplăcute. Rezultatul furnizat este altul – vezi figura 10.10. Pentru fiecare linie a rezultatului, vânzările din ziua curentă sunt identice vânzărilor din ziua precedentă.

zi date	vinzari_zi_curenta numeric	vinzari_zi_precedenta numeric	diferenta numeric
2007-08-01	10599535.0000	10599535.0000	0.0000
2007-08-02	277950.0000	277950.0000	0.0000
2007-08-03	222050.0000	222050.0000	0.0000
2007-08-04	201975.0000	201975.0000	0.0000
2007-08-07	10610000.5000	10610000.5000	0.0000
2007-08-14	4723931.5000	4723931.5000	0.0000
2007-08-15	247757.0000	247757.0000	0.0000
2007-08-16	467420.0000	467420.0000	0.0000
2007-08-21	10560939.5000	10560939.5000	0.0000
2007-09-01	4596401.5000	4596401.5000	0.0000
2007-09-02	238437.5000	238437.5000	0.0000
2007-09-10	424704.5000	424704.5000	0.0000
2007-09-17	179565.0000	179565.0000	0.0000
2007-10-07	5819668.0000	5819668.0000	0.0000

Figura 10.10. Un pic de năuceală proaspătă în PostgreSQL

Neavând o explicaţie ştiinţifică a fenomenului oarecum paranormal, bănuim, totuşi, că ni se trage de la modul de specificare a intervalului, prilej cu care

rememorăm o discuție purtată în paragraful 5.3. Aşa e, pentru că, dacă înlocuim *INTERVAL '1' DAY* cu *INTERVAL '1 DAYS'*:

```
SELECT DataFact AS Zi,
        SUM(Cantitate * PretUnit * (1+ProcTVA))  AS Vinzari_Zi_Curenta,
        COALESCE        (
                (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 - INTERVAL '1 DAYS'
                        ),0)  AS Vinzari_Zi_Precedenta,
        SUM(Cantitate * PretUnit * (1+ProcTVA)) -
                COALESCE(
                        (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 - INTERVAL '1 DAYS'
                        ),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 ORDER BY DataFact
```

rezultatul va fi unul corect şi în PostgreSQL.

O limită a corelării în clauza SELECT iese la iveală atunci când încercăm să simplificăm interogarea de mai sus cu ajutorul unei subconsultări în clauza FROM. Încercăm ca, printr-o subconsultare în clauza FROM, să creăm o tabelă ad-hoc – VINZARI_ZILNICE – la care să ne corelăm în clauza SELECT:

```
        SELECT Zi, Vinzari_Zi,
                COALESCE((SELECT Vinzari_Zi FROM vinzari_zilnice vz2
                        WHERE vz2.Zi = vinzari_zilnice.Zi - INTERVAL '1' DAY),
                                0) AS Vinzari_zi_precendenta,
                Vinzari_Zi - COALESCE((SELECT Vinzari_Zi FROM vinzari_zilnice vz2
                        WHERE vz2.Zi = vinzari_zilnice.Zi - INTERVAL '1' DAY),0)
                                AS Diferenta
        FROM
                (SELECT DataFact AS Zi,
                        SUM(Cantitate * PretUnit * (1+ProcTVA))  AS Vinzari_Zi
                FROM  facturi f
                        INNER JOIN liniifact lf ON f.NrFact=lf.NrFact
                        INNER JOIN produse p ON lf.CodPr=p.CodPr
                GROUP BY DataFact
```

```
        ORDER BY DataFact
        ) vinzari_zilnice
```

Interogarea s-ar simplifica destul de mult, însă nici Oracle, nici DB2, nici PostgreSQL nu recunosc în subconsultarea din clauza FROM tabela „ad-hoc" (SQL Server-ul nici atât) – vezi figura 10.11.

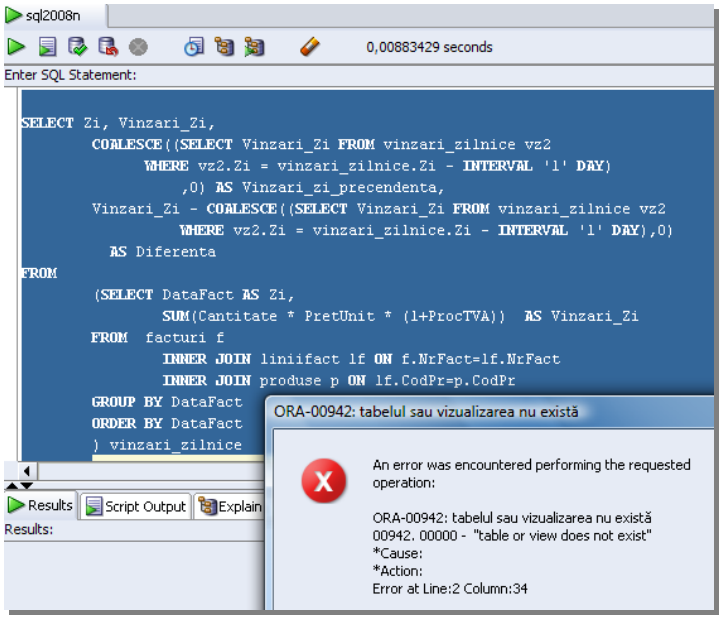


Figura 10.11. O subconsultare din clauza SELECT nu se poate corela (întotdeauna) cu o tabelă ad-hoc obţinută printr-o subconsultarea în clauza FROM

Pentru a nu înregistra o (altă) înfângere, vom transpune logica acestei interogări într-o alta similară (Oracle) ce foloseşte o expresie-tabelă.

```
WITH  vinzari_zilnice AS
        (SELECT DataFact AS Zi,
                SUM(Cantitate * PretUnit * (1+ProcTVA))  AS Vinzari_Zi
        FROM  facturi f
                INNER JOIN liniifact lf ON f.NrFact=lf.NrFact
                INNER JOIN produse p ON lf.CodPr=p.CodPr
        GROUP BY DataFact
        ORDER BY DataFact
        )
SELECT Zi, Vinzari_Zi,
        COALESCE((SELECT Vinzari_Zi FROM vinzari_zilnice vz2
       WHERE vz2.Zi = vinzari_zilnice.Zi - INTERVAL '1' DAY),
                0) AS Vinzari_zi_precendenta,
        Vinzari_Zi - COALESCE((SELECT Vinzari_Zi FROM vinzari_zilnice vz2
```

```
                    WHERE vz2.Zi = vinzari_zilnice.Zi - INTERVAL '1' DAY
                          ),0)  AS Diferenta
FROM vinzari_zilnice
```

Portarea acesteia în SQL Server ne dă prilejul descoperii unei noi probleme – vezi figura 10.12. Conform mesajului, ordonarea din expresia tabelă este invalidă în lipsa opțiunii TOP.

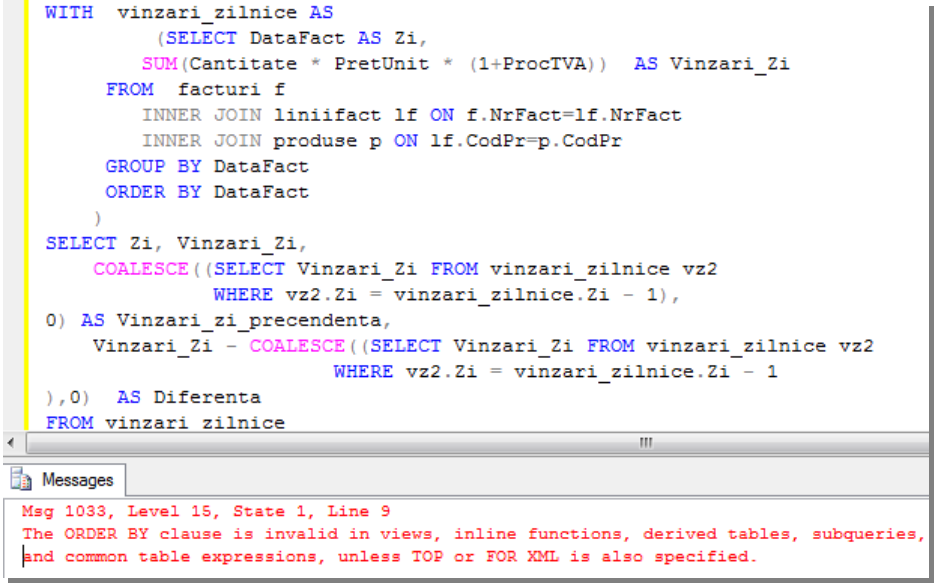


Figura 10.12. O problemă SQL Server de ordonare într-o expresie-tabelă

Ei bine, top a vrut, top a găsit ! Includem, din propria burtă, un *TOP 1000* pentru a salva sintactic interogarea:

```
WITH  vinzari_zilnice AS
        (SELECT TOP 1000 DataFact AS Zi,
                SUM(Cantitate * PretUnit * (1+ProcTVA))  AS Vinzari_Zi
         FROM  facturi f
                INNER JOIN liniifact lf ON f.NrFact=lf.NrFact
                INNER JOIN produse p ON lf.CodPr=p.CodPr
         GROUP BY DataFact
         ORDER BY DataFact
        )
SELECT Zi, Vinzari_Zi,
        COALESCE((SELECT Vinzari_Zi FROM vinzari_zilnice vz2
                        WHERE vz2.Zi = vinzari_zilnice.Zi - 1),
                        0) AS Vinzari_zi_precendenta,
        Vinzari_Zi - COALESCE((SELECT Vinzari_Zi FROM vinzari_zilnice vz2
```

```
                    WHERE vz2.Zi = vinzari_zilnice.Zi - 1
                ),0)  AS Diferenta
FROM vinzari_zilnice
```

Acum merge !

Să se afişeze penalizările pentru întârzierea la plată a facturilor, ştiind că acestea se calculează pentru fiecare zi de întârziere, pe tranşe, astfel:

- pentru tranşele neîncasate în maxim 10 zile de la data facturării, nu se calculează penalizări;
- pentru tranşele neîncasate în intervalul 11-12 zile de la data facturării, procentul de penalitate este 0,05% din valoarea neîncasată, pentru fiecare zi de întârziere;
- pentru tranşele neîncasate în intervalul 13-14 zile de la data facturării, procentul de penalitate este 0,1% din valoarea neîncasată, pentru fiecare zi de întârziere;
- pentru tranşele încasate în intervalul 15-20 zile de la data facturării, procentul este de 0,2% din valoarea neîncasată pentru fiecare zi de întârziere;
- pentru tranşele încasate în intervalul 21-40 zile de la data facturării, procentul este de 0,3% din valoarea neîncasată pentru fiecare zi de întârziere;
- pentru tranşele încasate cu întârziere de peste 40 zile de la data facturării, procentul este de 0,5% din valoarea neîncasată pentru fiecare zi de întârziere;

Inițial, această problemă a fost formulată în finalul paragrafului 9.4.1, moment în care am recunoscut că suntem incapabili să o rezolvăm. Pe calapodul soluțiilor anterioare, ne folosim de o subconsultare corelată pentru a determina încasările cumulate, până în momentul unei tranşe, şi, astfel, să aflăm restul de încasat:

```
SELECT NrFact AS "NrFact", DataFact AS "DataFact", Facturat AS "Facturat",
       DataInc AS "DataIncas", Incasat AS "Incasat", Zile_Intirz AS "Zile_Intirziere",
       Proc_Penaliz AS "Proc_Penalz",
            COALESCE((SELECT SUM(Incasat) FROM incasari i2
                  WHERE NrFact=t.NrFact AND DataInc < t.DataInc),0)
                        AS "Incas_Precedente"
FROM  (SELECT vinzari.NrFact, DataFact, Facturat,
            COALESCE(DataInc, CURRENT_DATE) AS DataInc,
            COALESCE(Incasat,0) AS Incasat,
            TRUNC(COALESCE(DataInc, CURRENT_DATE) - DataFact,0)
                  AS NrZile,
            TRUNC(COALESCE(DataInc, CURRENT_DATE) - DataFact - 10,0)
                  AS Zile_Intirz,
            CASE
```

```
            WHEN COALESCE(DataInc,CURRENT_DATE) - DataFact <= 10 THEN 0
            WHEN COALESCE(DataInc,CURRENT_DATE) - DataFact <= 12
                  THEN 0.0005
            WHEN COALESCE(DataInc,CURRENT_DATE) - DataFact <= 14
                  THEN 0.001
          WHEN COALESCE(DataInc,CURRENT_DATE) - DataFact <= 20
                  THEN 0.002
          WHEN COALESCE(DataInc,CURRENT_DATE) - DataFact <= 40
                  THEN 0.003
          ELSE 0.005
          END AS Proc_Penaliz
    FROM   (SELECT f.NrFact, DataFact, ROUND(SUM(Cantitate * PretUnit
                  * (1+ProcTVA)),0) AS Facturat
            FROM facturi f
                  INNER JOIN liniifact lf ON f.NrFact=lf.NrFact
                  INNER JOIN produse p ON lf.CodPr = p.CodPr
            GROUP BY f.NrFact, DataFact
           ) VINZARI
                        LEFT OUTER JOIN
           ( SELECT NrFact, DataInc, ROUND(SUM(Transa),0) AS Incasat
            FROM  incasfact INNER JOIN incasari
                  ON incasfact.CodInc=incasari.CodInc
            GROUP BY NrFact, DataInc
            ) INCASARI
                        ON VINZARI.NrFact = INCASARI.NrFact
    ORDER BY NrFact, DataInc          ) t
```

Dacă ar fi fost să ne luăm după figura 10.11, interogarea corelată menită să calculeze încasările precedente tranşei (pentru factura respectivă) - interogare subliniată prin caracterele italice („aplecate") - ar fi trebuit să nu funcţioneze. Ei bine, avem o veste bună şi una rea. Cea bună este că funcţionează în Oracle – vezi figura 10.13. Cea proastă este că funcţionează aiurea – vezi tot figura 10.13.

NrFact	DataFact	Facturat	DataIncas	Incasat	Zile_Intirziere	Proc_Penalz	Incas_Precedente
1111	01-08-2007	4346038	15-08-2007	53996	4	0,001	0
1112	01-08-2007	125516	15-08-2007	125516	4	0,001	0
1113	01-08-2007	106275	17-08-2007	106275	6	0,002	318825
1114	01-08-2007	6021707	12-03-2008	0	214	0,005	0
1115	02-08-2007	151238	12-03-2008	0	213	0,005	0
1116	02-08-2007	126713	12-03-2008	0	213	0,005	0
1117	03-08-2007	222050	16-08-2007	9754	3	0,001	19508
1117	03-08-2007	222050	17-08-2007	9754	4	0,001	29262
1117	03-08-2007	222050	18-08-2007	3696	5	0,002	18480
1118	04-08-2007	201975	15-08-2007	101975	1	0,0005	0
1118	04-08-2007	201975	16-08-2007	100000	2	0,0005	200000
1119	07-08-2007	5774499	12-03-2008	0	208	0,005	0
1120	07-08-2007	97664	16-08-2007	7315	-1	0	14630
1121	07-08-2007	4737838	12-03-2008	0	208	0,005	0

Figura 10.13. O eroare (SQL) de corelare

Dacă la facturile 1111 şi 1112 încasările precedente sunt corecte, pentru factura 1113 valoarea Incas_Precente este cel puțin curioasă. Ca să nu mai vorbim de situația facturii 1117. Ia să înlocuim interogarea corelată din clauza SELECT (cea *italic*-ată în interogarea de mai sus) cu una în care calculăm încasările precedente din tabelele INCASARI şi INCASFACT:

```
SELECT NrFact AS "NrFact", DataFact AS "DataFact", Facturat AS "Facturat", DataInc
       AS "DataIncas", Incasat AS "Incasat", Zile_Intirz AS "Zile_Intirziere",
       Proc_Penaliz AS "Proc_Penalz", COALESCE((SELECT SUM(Transa)
              FROM incasari i2 INNER JOIN incasfact if2 ON i2.CodInc=if2.CodInc
                     WHERE NrFact=t.NrFact AND DataInc < t.DataInc)
                            ,0) AS "Incas_Precedente"
FROM
       (SELECT vinzari.NrFact, DataFact, Facturat,
              COALESCE(DataInc, CURRENT_DATE) AS DataInc,
              COALESCE(Incasat,0) AS Incasat,
              TRUNC(COALESCE(DataInc, CURRENT_DATE) - DataFact,0)
                     AS NrZile,
              TRUNC(COALESCE(DataInc, CURRENT_DATE) - DataFact - 10,0)
                     AS Zile_Intirz,
              CASE
              WHEN COALESCE(DataInc,CURRENT_DATE) - DataFact <= 10 THEN 0
              WHEN COALESCE(DataInc,CURRENT_DATE) - DataFact <= 12
                     THEN 0.0005
              WHEN COALESCE(DataInc,CURRENT_DATE) - DataFact <= 14
                     THEN 0.001
            WHEN COALESCE(DataInc,CURRENT_DATE) - DataFact <= 20
                     THEN 0.002
```

```
            WHEN COALESCE(DataInc,CURRENT_DATE) - DataFact <= 40
                    THEN 0.003
            ELSE 0.005
            END AS Proc_Penaliz
    FROM
            (SELECT f.NrFact, DataFact, ROUND(SUM(Cantitate * PretUnit
                    * (1+ProcTVA)),0) AS Facturat
             FROM facturi f
                    INNER JOIN liniifact lf ON f.NrFact=lf.NrFact
                    INNER JOIN produse p ON lf.CodPr = p.CodPr
             GROUP BY f.NrFact, DataFact
            ) VINZARI
                            LEFT OUTER JOIN
            ( SELECT NrFact, DataInc, ROUND(SUM(Transa),0) AS Incasat
             FROM  incasfact INNER JOIN incasari
                    ON incasfact.CodInc=incasari.CodInc
             GROUP BY NrFact, DataInc
             ) INCASARI
                            ON VINZARI.NrFact = INCASARI.NrFact
    ORDER BY NrFact, DataInc
     ) t
```

Rezultatul este cât se poate de corect – vezi figura 10.14 – valorile încasărilor precedente fiind calculate cum se cuvine pentru facturile 1117 şi 1118, iar pentru factura 1120 numărul zilelor de întârziere este negativ (încasarea a fost mai devreme cu o zi.

NrFact	DataFact	Facturat	DataIncas	Incasat	Zile_Intirziere	Proc_Penalz	Incas_Precedente
1111	01-08-2007	4346038	15-08-2007	53996	4	0,001	0
1112	01-08-2007	125516	15-08-2007	125516	4	0,001	0
1113	01-08-2007	106275	17-08-2007	106275	6	0,002	0
1114	01-08-2007	6021707	12-03-2008	0	214	0,005	0
1115	02-08-2007	151238	12-03-2008	0	213	0,005	0
1116	02-08-2007	126713	12-03-2008	0	213	0,005	0
1117	03-08-2007	222050	16-08-2007	9754	3	0,001	0
1117	03-08-2007	222050	17-08-2007	9754	4	0,001	9754
1117	03-08-2007	222050	18-08-2007	3696	5	0,002	19508
1118	04-08-2007	201975	15-08-2007	101975	1	0,0005	0
1118	04-08-2007	201975	16-08-2007	100000	2	0,0005	101975
1119	07-08-2007	5774499	12-03-2008	0	208	0,005	0
1120	07-08-2007	97664	16-08-2007	7315	-1	0	0
1121	07-08-2007	4737838	12-03-2008	0	208	0,005	0

Figura 10.14. Modificarea (cu succes a) subconsultării corelate

Mai avem, însă, de lucru până la furnizarea răspunsului direct la întrebarea formulată. Mai întâi, pentru a afla întârzierea pentru fiecare tranşă, ar trebui să înmulţim - repet, pentru fiecare tranşă – procentul aferent numărului de zile de întârziere al tranşei cu diferenţa dintre valoarea totală a facturii şi încasările precedente tranşei din factură:

```
SELECT NrFact AS "NrFact", DataFact AS "DataFact", Facturat AS "Facturat", DataInc
      AS "DataIncas", Incasat AS "Incasat", Zile_Intirz AS "Zile_Intirziere",
      Proc_Penaliz AS "Proc_Penalz", COALESCE((SELECT SUM(Transa)
            FROM incasari i2 INNER JOIN incasfact if2 ON i2.CodInc=if2.CodInc
                  WHERE NrFact=t.NrFact AND DataInc < t.DataInc)
                        ,0) AS "Incas_Precedente",
      Proc_Penaliz * (Facturat - COALESCE((SELECT SUM(Transa) FROM
            incasari i2 INNER JOIN incasfact if2 ON i2.CodInc=if2.CodInc
            WHERE NrFact=t.NrFact AND DataInc < t.DataInc)
                        , 0)) AS "Penaliz_Transa"
FROM
      (...) t
```

Ceea ce obţinem (figura 10.15) este încă un pas către rezultatul final. Penalizările fiecărei tranşe par a fi corecte, dar sunt cele din momentul încasării tranşei respective. Dacă pentru facturile fără nicio încasare penalizarea se calculează corect (relativ la momentul curent[3]), mai există însă o serie de penalizări necalculate pentru sumele rămase de încasat din fiecare factură plătită parţial.

[3] Interogările pentru rezolvarea acestei probleme au fost executate pe 12 martie 2008 (vezi unde apare CURRENT_DATE)

NrFact	DataFact	Facturat	DataIncas	Incasat_Transa	Zile_Intirziere	Proc_Penalz	Incas_Precedente	Penaliz_Transa
1111	01-08-2007	4346038	15-08-2007	53996	4	0,001	0	4346,038
1112	01-08-2007	125516	15-08-2007	125516	4	0,001	0	125,516
1113	01-08-2007	106275	17-08-2007	106275	6	0,002	0	212,55
1114	01-08-2007	6021707	12-03-2008	0	214	0,005	0	30108,535
1115	02-08-2007	151238	12-03-2008	0	213	0,005	0	756,19
1116	02-08-2007	126713	12-03-2008	0	213	0,005	0	633,565
1117	03-08-2007	222050	16-08-2007	9754	3	0,001	0	222,05
1117	03-08-2007	222050	17-08-2007	9754	4	0,001	9754	212,296
1117	03-08-2007	222050	18-08-2007	3696	5	0,002	19508	405,084
1118	04-08-2007	201975	15-08-2007	101975	1	0,0005	0	100,9875
1118	04-08-2007	201975	16-08-2007	100000	2	0,0005	101975	50
1119	07-08-2007	5774499	12-03-2008	0	208	0,005	0	28872,495
1120	07-08-2007	97664	16-08-2007	7315	-1	0	0	0
1121	07-08-2007	4737838	12-03-2008	0	208	0,005	0	23689,19

Figura 10.15. Modificarea (cu succes a) subconsultării corelate

Cum singura problemă rămasă de rezolvat este includerea în rezultat a facturilor încasate parțial la data curentă (12 martie, în cazul meu), apelăm la un truc ieftin: includem în tabela ad-hoc INCASARI2 câte o linie pentru fiecare factură încasată parțial în care data încasării să fie data curentă, iar tranşa încasată să fie zero:

```
SELECT NrFact AS "NrFact", DataFact AS "DataFact", Facturat AS "Facturat",
        DataInc AS "DataIncas", Incasat AS "Incasat_Transa",
        Zile_Intirz AS "Zile_Intirziere", Proc_Penaliz AS "Proc_Penalz",
        COALESCE((      SELECT SUM(Transa)
                        FROM incasari i2 INNER JOIN incasfact if2
                                ON i2.CodInc=if2.CodInc
                        WHERE NrFact=t.NrFact AND DataInc < t.DataInc),0
                        ) AS "Incas_Precedente",
        Proc_Penaliz * (Facturat - COALESCE(
                (SELECT SUM(Transa)
                FROM incasari i2
                        INNER JOIN incasfact if2 ON i2.CodInc=if2.CodInc
                WHERE NrFact=t.NrFact AND DataInc < t.DataInc),0))
                        AS "Penaliz_Transa"
FROM
        (SELECT vinzari.NrFact, DataFact, Facturat,
                COALESCE(DataInc, CURRENT_DATE) AS DataInc,
                COALESCE(Incasat,0) AS Incasat,
                TRUNC(COALESCE(DataInc, CURRENT_DATE) - DataFact,0)
                        AS NrZile,
                TRUNC(COALESCE(DataInc, CURRENT_DATE) - DataFact - 10,0)
                        AS Zile_Intirz,
```

```
        CASE
        WHEN COALESCE(DataInc,CURRENT_DATE) - DataFact <= 10
                THEN 0
        WHEN COALESCE(DataInc,CURRENT_DATE) - DataFact <= 12
                THEN 0.0005
        WHEN COALESCE(DataInc,CURRENT_DATE) - DataFact <= 14
                THEN 0.001
        WHEN COALESCE(DataInc,CURRENT_DATE) - DataFact <= 20
                THEN 0.002
        WHEN COALESCE(DataInc,CURRENT_DATE) - DataFact <= 40
                THEN 0.003
        ELSE 0.005
        END AS Proc_Penaliz
FROM
        (SELECT f.NrFact, DataFact, ROUND(SUM(Cantitate *
                PretUnit * (1+ProcTVA)),0) AS Facturat
        FROM facturi f
                INNER JOIN liniifact lf ON f.NrFact=lf.NrFact
                INNER JOIN produse p ON lf.CodPr = p.CodPr
        GROUP BY f.NrFact, DataFact
        ) VINZARI
                        LEFT OUTER JOIN
        ( SELECT NrFact, DataInc, ROUND(SUM(Transa),0) AS Incasat
         FROM  incasfact INNER JOIN incasari
                ON incasfact.CodInc=incasari.CodInc
         GROUP BY NrFact, DataInc
                UNION
                SELECT NrFact, CURRENT_DATE, 0
                 FROM incasfact i
                 GROUP BY NrFact
                HAVING SUM(Transa) < (
                        SELECT SUM(Cantitate * PretUnit * (1+ProcTVA))
                        FROM liniifact lf INNER JOIN produse p
                                ON lf.CodPr = p.CodPr
                        WHERE NrFact = i.Nrfact
                                        )
        ) INCASARI2
                        ON VINZARI.NrFact = INCASARI2.NrFact
ORDER BY NrFact, DataInc
 ) t
```

Figura 10.16 este încurajatoare, întrucât apare câte o linie nouă pentru fiecare factură din care s-a încasat câte ceva, dar mai rămâne un rest de încasat cât de mic – facturile 1111 şi 1117 (probabil mai sunt şi altele, dar n-au încăput în figură).

NrFact	DataFact	Facturat	DataIncas	Incasat_Transa	Zile_Intirziere	Proc_Penalz	Incas_Precedente	Penaliz_Transa
1111	01-08-2007	4346038	15-08-2007	53996	4	0,001	0	4346,038
1111	01-08-2007	4346038	12-03-2008	0	214	0,005	53996	21460,21
1112	01-08-2007	125516	15-08-2007	125516	4	0,001	0	125,516
1113	01-08-2007	106275	17-08-2007	106275	6	0,002	0	212,55
1114	01-08-2007	6021707	12-03-2008	0	214	0,005	0	30108,535
1115	02-08-2007	151238	12-03-2008	0	213	0,005	0	756,19
1116	02-08-2007	126713	12-03-2008	0	213	0,005	0	633,565
1117	03-08-2007	222050	16-08-2007	9754	3	0,001	0	222,05
1117	03-08-2007	222050	17-08-2007	9754	4	0,001	9754	212,296
1117	03-08-2007	222050	18-08-2007	3696	5	0,002	19508	405,084
1117	03-08-2007	222050	12-03-2008	0	212	0,005	23204	994,23
1118	04-08-2007	201975	15-08-2007	101975	1	0,0005	0	100,9875
1118	04-08-2007	201975	16-08-2007	100000	2	0,0005	101975	50

Figura 10.16. Includerea în rezultat şi a penalizărilor pentru facturile încasate parţial la data curentă

Am ajuns la capăt: includem toată interogarea precedentă într-un FROM, şi aplicăm gruparea:

```
SELECT "NrFact", "DataFact", "Facturat", SUM ("Incasat_Transa") AS "Incasat",
        SUM("Penaliz_Transa") AS "Penalizari"
FROM
        (
                ... interogarea precedentă
        )
GROUP BY "NrFact", "DataFact", "Facturat"
ORDER BY 1
```

Comparând rezultatul din figura 10.17 cu cel din figura 10.16 avem motive să fim încrezători cu privire la corectitudinea rezultatului.

NrFact	DataFact	Facturat	Incasat	Penalizari
1111	01-08-2007	4346038	53996	25806,248
1112	01-08-2007	125516	125516	125,516
1113	01-08-2007	106275	106275	212,55
1114	01-08-2007	6021707	0	30108,535
1115	02-08-2007	151238	0	756,19
1116	02-08-2007	126713	0	633,565
1117	03-08-2007	222050	23204	1833,66
1118	04-08-2007	201975	201975	150,9875
1119	07-08-2007	5774499	0	28872,495
1120	07-08-2007	97664	7315	451,745
1121	07-08-2007	4737838	0	23689,19
2111	14-08-2007	4442960	0	22214,8
2112	14-08-2007	153442	0	767,21

Figura 10.17.Centralizarea penalizărilor pe facturi

Este clar că situaţia clienţilor este deprimantă, însă am forţat procentele de penalizare doar pentru a vedea cum s-ar putea rezolva un asemenea gen de problemă. De-atâta oboseală, nu mai portăm interogarea în celelalte trei dialecte SQL.

10.2. Subconsultări corelate în clauza WHERE

Corelarea subconsultărilor în clauza WHERE reprezintă una dintre cele mai provocatoare teme în SQL, fiind mai dificilă, dar şi mai generoasă, decât corelarea subconsultărilor din clauza SELECT. Este momentul intrării în scenă a unuia dintre cei mai puternici şi temuţi operatori – EXISTS.

10.2.1. Operatorul EXISTS

Ca şi IN, operatorul EXISTS permite legarea frazei SELECT principale de una sau mai multe sub-consultări şi, astfel, formularea unor interogări complexe. Prin EXISTS se defineşte un predicat care are valoarea logică "adevărat" dacă sub-consultarea s-a concretizat într-o tabelă ce are cel puţin o linie.

În general, EXISTS poate înlocui cu succes operatorul IN. Reciproca nefiind valabilă[4], unii autori, precum C.J. Date îl preferă şi recomandă la formularea subconsultărilor. Pe de altă parte, însă, EXISTS este unul dintre cei mai dificili operatori. Logica sa se deosebeşte radical de cea „clasică" a lui IN.

Utilizând subconsultări necorelate, „conectate" la interogarea principală prin IN (sau =, >=, <, <=, <>), succesiunea paşilor se derulează astfel:

1. Se execută subconsultarea, rezultatul fiind un set de zero, una sau mai multe linii;
2. Din interogarea principală se extrag acele linii pentru valorile atributului/expresiei care sunt egale (sau diferite, mai mari s.a.m.d. - depinde de condiție) cu/decât cele obținute în rezultatul subconsultării.

La utilizarea lui EXISTS, se extrag linii tot ale tabelei/tabelelor frazei SELECT principală, dar pe baza existenței a cel puțin o linie în rezultatul subconsultării; în plus, în predicatul subconsultării se face referire şi la tupluri din interogarea principală (eu nu am înțeles nimic din fraza aceasta, dar dumneavoastră ?).

Predicatele din clauza WHERE operează la nivel de linie, şi 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 Larsen[5] pledează pentru folosirea lui EXISTS în detrimentul IN-ului pe considerentul optimizării[6]. IN presupune crearea unor tabele temporare ce, uneori, reclamă timp de execuție sensibil mai mare prin comparație cu logica *tuplu-cu-tuplu* EXISTS-ențială.

Pentru comparație, începem cu exemple formulate în capitolul precedent la prezentarea operatorului IN.

Ce facturi au fost emise în aceeaşi zi cu factura 1120 ?

Varianta de interogare bazată pe operatorul EXISTS se prezintă astfel:

```
SELECT *
FROM facturi f1
WHERE EXISTS
```

---

[4] [Date1986], p.144

[5] [Larsen1996]

[6] Cred, însă, că optimizarea e o chestiune internă a fiecărui SGBD. Deşi unele variante de interogări SQL sunt, oricum, mari consumatoare de resurse, indiferent pe serverul de baze de date pe care s-ar executa, există însă şi diferențe semnificative de la SGBD la SGBD în materie de performanțe.

```
(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 10.18.

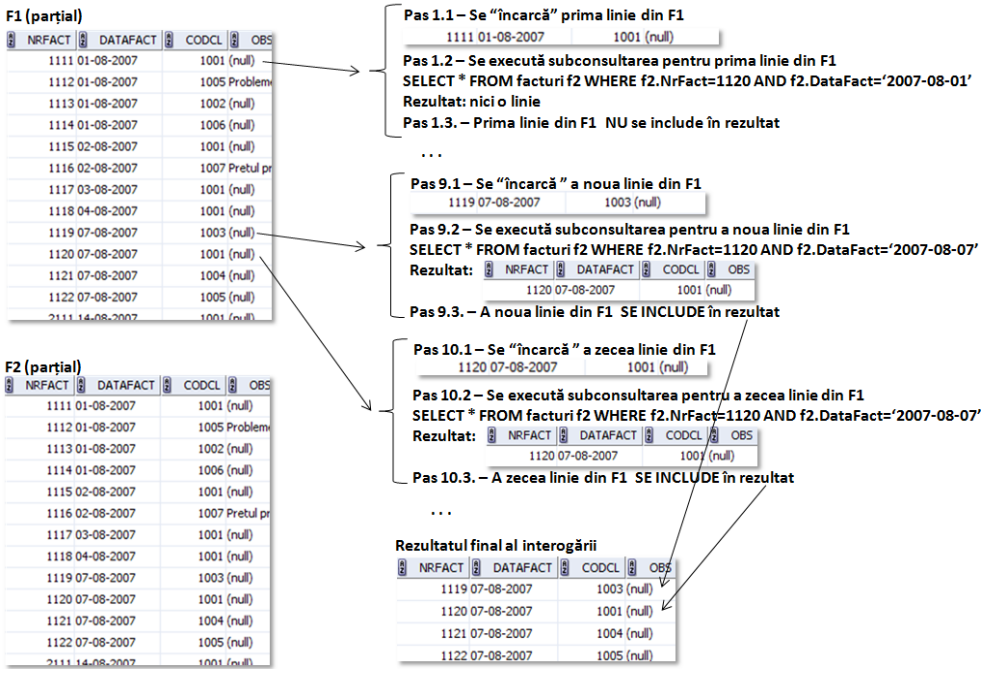


Figura 10.18. Logica operatorului EXISTS

Elementul de dificultate ține 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 sub-consultare care să satisfacă predicatul subconsultării. Dacă da, atunci nu aceasta se inserează în rezultat, ci linia din SELECT-ul principal ! Nu ştiu cum e mai nimerit să-i zicem, lucru în echipă, sau însuşirea de către SELECT-ul principal a meritelor SELECT-ului "subaltern".

Dacă se doreşte eliminarea din rezultat şi a facturii de referință, 1120, se modifică uşor interogarea:

```
SELECT *
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 şi de IN. Interogarea următoare este perfect similară, ca rezultat, anterioarei:

```
SELECT *
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 soluţii sunt echivalente:

```
SELECT *
FROM facturi f1
WHERE NOT EXISTS
        (SELECT *
        FROM facturi f2
        WHERE f2.NrFact=1120 AND f1.DataFact=f2.DataFact)
```

şi

```
SELECT *
FROM facturi f1
WHERE EXISTS
        (SELECT *
        FROM facturi f2
        WHERE f2.NrFact=1120 AND f1.DataFact<>f2.DataFact)
```

În subconsultarea corelată (cea care succede operatorului EXISTS) clauza SELECT poate conţine ca argument * sau un atribut care, de preferinţă, nu are valori nule. Cea mai simplă, sigură şi rapidă (pentru SBGD) variantă este însă folosirea unei constante. Spre exemplu, ultima variantă se poate schimba, aproape insesizabil, în:

```
SELECT *
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 şi este de bun augur pentru simplitate.

Care sunt clienţii cărora li s-au trimis facturi în aceeaşi zi în care a fost întocmită factura 1120 ?

```
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)
                        )
```

Soluția conține 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 județe s-a vândut produsul "Produs 2" ?

```
SELECT Judet
FROM judete j
WHERE EXISTS
        (SELECT 1
         FROM coduri_postale cp
         WHERE cp.Jud=j.Jud AND EXISTS
                (SELECT 1
                 FROM clienti c
                 WHERE c.CodPost=cp.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
                                                 DenPr = 'Produs 2'
                                        ) ) ) ) )
ORDER BY 1
```

Această soluție înlocuieşte atât joncțiunea, cât şi numeroasele tabele intermediare presupuse de subconsultările ne-corelate.

Prin EXISTS, fireşte, se poate realiza şi intersecţia a două relaţii. Dacă luăm în discuţie relaţiile R1 şi R2 din paragraful 4.3.2 (figura 4.4), intersecţia acestora se realizează şi 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)
```

În ce zile s-au vândut şi produsul cu denumirea "Produs 1" şi cel cu denumirea "Produs 2" ?

```
SELECT DISTINCT DataFact
FROM produse p1
        INNER JOIN liniifact lf1 ON p1.CodPr=lf1.CodPr
        INNER JOIN facturi f1 ON lf1.NrFact=f1.NrFact
WHERE DenPr = 'Produs 1' AND EXISTS
        (SELECT 1
        FROM produse p2
                INNER JOIN liniifact lf2 ON p2.CodPr=lf2.CodPr
                INNER JOIN facturi f2 ON lf2.NrFact=f2.NrFact
         WHERE p2.DenPr = 'Produs 2' AND f2.DataFact=f1.DataFact)
```

Sunt corelate două „instanţe" ale joncţiunii PRODUSE-LINIIFACT-FACTURI; prima conţine liniile 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 clienţi au cumpărat şi "Produs 2" şi "Produs 3", dar nu au cumpărat "Produs 5" ?

Acesta este exemplul în care sunt folosite, mânâ în mână, intersecţia şi diferenţa.

```
SELECT DISTINCT DenCl
FROM produse p1
        INNER JOIN liniifact lf1 ON p1.CodPr=lf1.CodPr
        INNER JOIN facturi f1 ON lf1.NrFact=f1.NrFact
        INNER JOIN clienti c1 ON f1.CodCl=c1.CodCl
WHERE DenPr = 'Produs 2' AND EXISTS
        (SELECT 1
         FROM produse p2
                INNER JOIN liniifact lf2 ON p2.CodPr=lf2.CodPr
                INNER JOIN facturi f2 ON lf2.NrFact=f2.NrFact
         WHERE DenPr = 'Produs 3' AND CodCl=c1.CodCl
        )
```

```
AND NOT EXISTS
      (SELECT 1
       FROM produse p3
                INNER JOIN liniifact lf3 ON p3.CodPr=lf3.CodPr
                INNER JOIN facturi f3 ON lf3.NrFact=f3.NrFact
       WHERE DenPr = 'Produs 5' AND CodCl=c1.CodCl
         )
```

Fraza SELECT principală este corelată la două subconsultări dispuse "în linie", secvențial. S-au folosit şi EXISTS şi 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. Fireşte, cea mai la îndemână variantă foloseşte o subconsultare necorelată:

```
SELECT DenCl
FROM clienti
WHERE CodCl IN
        (SELECT CodCl
         FROM facturi
         GROUP BY CodCl
         HAVING COUNT(NrFact) = 2)
```

Şi mai simplă decât aceasta este soluția fără nicio subconsultare:

```
SELECT DenCl
FROM clienti INNER JOIN facturi ON clienti.CodCl=facturi.CodCl
GROUP BY DenCl
HAVING COUNT(NrFact) = 2
```

Varianta pe care o consider cea mai interesantă nu foloseşte, pentru corelarea subconsultării cu fraza SELECT principală, nici EXISTS, nici IN, ci egalitatea:

```
SELECT DenCl
FROM clienti
WHERE 2 = ( SELECT COUNT(*)
            FROM facturi
```

WHERE facturi.CodCl=clienti.CodCl)

Logica interogării, exprimată nu atât vizual, cât „în proză" este:

- Etapa 1:
  - se "încarcă" prima linie[7] din tabela CLIENTI: *(1001, 'Client 1 SRL', 'R1001', 'Tranzitiei, 13 bis', '700505', NULL)* ;
  - se numără, cu funcția COUNT, câte linii din FACTURI au CodCl = 1001 (*SELECT COUNT(*) FROM facturi WHERE facturi.CodCl=1001*); rezultatul este 13.
  - se compară 2 din SELECT-ul principal cu rezultatul subconsultării corelate: $12 \neq 2$, deci „Client 1 SRL" *nu* se include în rezultat.
- Etapa 2:
  - se "încarcă" a doua linie din CLIENTI: (*1002,'Client 2 SA', 'R1002', NULL, '700505', '0232212121'*);
  - se numără, cu funcția COUNT, câte linii din FACTURI au CodCl = 1002 (*SELECT COUNT(*) FROM facturi WHERE facturi.CodCl=1002*). Rezultatul este 3;
  - se compară 3 din SELECT-ul principal cu rezultatul subconsultării corelate: $3 \neq 2$, prin urmare, nici ‚Client 2 SA' *nu* se include în rezultat.
- Etapa 3:
  ...
- Etapa 4:
  - se "încarcă" a patra linie din CLIENTI: (*1004, 'Client 4', NULL, 'Sapientei, 56', '701150', NULL*);
  - se numără, cu funcția COUNT, câte linii din FACTURI au CodCl = 1004 (*SELECT COUNT(*) FROM facturi WHERE facturi.CodCl=1004*). Rezultatul este 2;
  - se compară 2 din SELECT-ul principal cu rezultatul subconsultării corelate: $2 = 2$, prin urmare, „Client 4" *va fi inclus* în rezultat.

...

Analog etapele 5, 6 şi 7 (avem doar 7 clienți în tabela cu acelaşi nume)

....

[7] Nimic nu garantează că, în realitate aceasta ar fi ordinea de încărcare dar, de dragul exemplului, vom face această supoziție.

Rezultatul final este cel din figura 10.19.

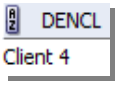


Figura 10.19. Singurul client cu două facturi

Care sunt cele mai mari cinci preţuri unitare la care s-au efectuat vânzări ?

Această problemă a mai fost formulată în paragraful 9.2. Revenim cu o soluţie nou-nouţă, bazată pe subinterogări corelate, soluţie care obţine un rezultat identic celui din figura 9.10.

```
SELECT *
FROM liniifact lf1
WHERE 5 >
        (SELECT COUNT(DISTINCT lf2.PretUnit)
         FROM liniifact lf2
         WHERE lf2.PretUnit > lf1.PretUnit)
ORDER BY PretUnit DESC, NrFact
```

Ideea este grozav de ingenioasă (păcat că nu este a mea): se parcurge linie cu linie prima instanţă a tabelei LINIIFACT – LF1. Pentru fiecare linie de LF1 se numără în a doua instanţă a LINIIFACT – LF2 câte linii prezintă preţul unitar mai mare decât cel de pe linia curentă din LF1. Dacă numărul calculat prin *COUNT(DISTINCT lf2.PretUnit)* este mai mic decât 5, atunci în rezultat se extrage linia curentă din LF1.

Cum nici eu n-am înţeles prea bine explicaţiile de mai sus, să analizăm interogarea mai pe îndelete.

Pas 1: se ia în discuţie primul tuplu din LINIIFACT (LF1)[8]. PretUnit are valoarea 1000. Se calculează numărul valorilor distincte ale PretUnit din LF2 pentru care *LF2.PretUnit* > *LF1.PretUnit*, adică se execută interogarea: *SELECT COUNT(DISTINCT lf2.PretUnit) FROM liniifact lf2 WHERE lf2.PretUnit* > *1000*.

[8] Ordinea în care se parcurg tuplurile din LINIIFACT (instanţa LF1) nu are nici cea mai mică importanţă. Pentru a nu compromite iremediabil explicaţiile, vor presupune că tuplurile sunt ordonate după valorile atributelor NrFact şi Linie.

Rezultatul este 8. Prin urmare, în LINIIFACT există opt preţuri unitare peste 1000. 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 1050. Funcţia COUNT din subconsultare (*SELECT COUNT(DISTINCT lf2.PretUnit) FROM liniifact lf2 WHERE lf2.PretUnit > 1050*) „returnează" 9 - în LINIIFACT există nouă linii cu preţuri unitare peste 1050. Cum 9 > 5, nici al doilea tuplu din LF1 *nu este inclus* în rezultat.

Pas 3: se "încarcă" al treilea uplu din LINIIFACT (LF1) în carte PretUnit este 7060. COUNT întoarce 1; 1 < 5, deci al treilea tuplu din LF1 *este inclus* în rezultatul final.

….

În total sunt 56 de paşi, corespunzători tuplurilor din LINIIFACT.

În care facturi există linii non-consecutive ?

În niciuna, decocamdată. Haideţi, însă, să introducem trei noi facturi în care liniile să fie „presărate" cu destule goluri între ele. Pentru scurtarea discuţiei ne vom limita la sintaxa Oracle/PostgreSQL.

```
INSERT INTO facturi (nrfact, datafact, codcl) VALUES (4111, DATE'2007-10-01', 1001);
INSERT INTO facturi (nrfact, datafact, codcl, obs)
        VALUES (4112, DATE'2007-10-01', 1005, 'Probleme cu transportul');
INSERT INTO facturi (nrfact, datafact, codcl) VALUES (4113, DATE'2007-10-02', 1002);

INSERT INTO liniifact (nrfact, linie, codpr, cantitate, pretunit)
        VALUES (4111, 1, 1, 57, 1000) ;
INSERT INTO liniifact (nrfact, linie, codpr, cantitate, pretunit)
        VALUES (4111, 3, 2, 79, 1050) ;
INSERT INTO liniifact (nrfact, linie, codpr, cantitate, pretunit)
        VALUES (4111, 5, 5, 510, 7060) ;
INSERT INTO liniifact (nrfact, linie, codpr, cantitate, pretunit)
        VALUES (4112, 2, 2, 85, 1030) ;
INSERT INTO liniifact (nrfact, linie, codpr, cantitate, pretunit)
        VALUES (4112, 5, 3, 65, 750) ;
INSERT INTO liniifact (nrfact, linie, codpr, cantitate, pretunit)
        VALUES (4113, 2, 2, 120, 975) ;
INSERT INTO liniifact (nrfact, linie, codpr, cantitate, pretunit)
        VALUES (4113, 3, 1, 110, 925) ;
```

În factura 4111 lipsesc liniile 2 şi 4, în 4112 liniile 1, 3 şi 4, iar în 4113 linia 1. Bizuindu-ne pe interogările corelate şi operatorul EXISTS, putem formula o soluţie destul de simplă:

```
        SELECT *
```

```
FROM liniifact lf1
WHERE linie > 1 AND NOT EXISTS
        (SELECT 1
         FROM liniifact lf2 WHERE NrFact=lf1.NrFact AND linie=lf1.linie-1)
ORDER BY Nrfact, Linie
```

În SELECT-ul principal (LF1) sunt luate în „discuție" liniile mai mari ca 1, iar, dintre acestea se extrag, prin intermediul subconsultării, cele care nu au linia precedentă (adică, dacă linia curentă este 2, se verifică dacă există, în aceeaşi factură, linia 1; dacă linia curentă este 3, se verifică dacă, în aceeaşi factură, există linia 2 s.a.m.d.). Rezultatul este corect – vezi figura 10.20.

NRFACT	LINIE	CODPR	CANTITATE	PRETUNIT
4111	3	2	79	1050
4111	5	5	510	7060
4112	2	2	85	1030
4112	5	3	65	750
4113	2	2	120	975

Figura 10.20. Linii din facturi în care nu există "precedentele" lor

Dacă am fi fost mai modeşti, adică ne-ar fi interesat doar facturile în care liniile sunt „şvaițer", puteam recurge şi la varianta:

```
SELECT NrFact
FROM liniifact
GROUP BY NrFact
HAVING COUNT(*) <> MAX(Linie)
```

Revenim la conținutul „standard" al celor două tabele ştergând liniile proaspăt adăugate:

```
DELETE FROM liniifact WHERE NrFact >= 4111 ;
DELETE FROM facturi WHERE NrFact >= 4111 ;
```

10.2.2. Corelări şi diviziuni

Din interogările menite să ilustreze folosirea operatorului EXISTS nu puteau lipsi cele legate de diviziunea relațională. Începem cu un prim exemplu (teoretic) din paragraful 4.4.7. Pentru a afla valorile lui X aflate în RD1 în combinație cu toate valorile lui Y din RD2 se poate recurge şi la soluția:

```
SELECT DISTINCT X
FROM rd1 T1_1
WHERE EXISTS
        (SELECT 1
         FROM rd1 T1_2
```

```
        WHERE T1_2.X = T1_1.X
        GROUP BY T1_2.X
        HAVING COUNT(DISTINCT T1_2.Y) =
                (SELECT COUNT(Y)
                 FROM rd2)
                )
```

Se parcurge, pe rând, fiecare linie din RD1. De fiecare dată, se verifică dacă, pentru X-ul curent, în RD1 apar toți igrecii din RD2, verificare realizată prin GROUP BY şi HAVING.

Ce produse au fost vândute tuturor clienților ? (exemplu 26, paragraf 4.4.7)?

Pentru fiecare produs din portofoliul firmei (fiecare linie din tabela PRODUSE) se verifică dacă există linia care s-ar obține numai dacă numărul clienților la care a fost vândut ar fi egal cu numărul înregistrărilor din tabela CLIENTI:

```
SELECT DISTINCT DenPr
FROM produse p1
WHERE EXISTS (
        SELECT 1
        FROM liniifact lf
                INNER JOIN facturi f ON lf.NrFact=f.NrFact
         WHERE lf.CodPr = p1.CodPr
         GROUP BY CodPr
          HAVING COUNT(DISTINCT CodCl) =
                                (SELECT COUNT(*) FROM clienti)
                )
```

Care sunt clienții pentru care există cel puțin câte o factură emisă în fiecare zi cu vânzări din perioada 10-30 septembrie 2007? (exemplu 23, paragraf 4.4.7)?

```
SELECT DISTINCT DenCl
FROM clienti
WHERE EXISTS (
        SELECT 1
        FROM facturi
        WHERE DataFact BETWEEN DATE'2007-09-10' AND DATE'2007-09-30'
                AND CodCl=clienti.CodCl
       GROUP BY CodCl
       HAVING COUNT(DISTINCT DataFact) =
                (SELECT COUNT(DISTINCT DataFact)
                 FROM facturi
                 WHERE DataFact BETWEEN DATE'2007-09-10' AND DATE'2007-09-30'
                )
                )
```

Ultima consultare (de jos), cea din clauza HAVING, calculează numărul total al zilelor în care sunt vânzări în intervalul 10-30 septembrie, număr care este 2. Subconsultarea "mijlocie" extrage o linie (şi o coloană) numai dacă numărul de zile de vânzări pentru clientul curent (clienti.CodCl) este egal cu 2 (rezultatul subconsultării de jos).

Care sunt facturile ce conţin măcar produsele din factura 1117? - exemplul 27 din paragraful 4.4.7.

```
SELECT DISTINCT NrFact
FROM facturi
WHERE EXISTS  (
        SELECT 1
        FROM liniifact
        WHERE CodPr IN (SELECT CodPr FROM liniifact WHERE NrFact = 1117)
                AND NrFact=facturi.NrFact
        GROUP BY NrFact
        HAVING COUNT(DISTINCT CodPr) =
                (SELECT COUNT(DISTINCT CodPr)
                 FROM liniifact
                 WHERE NrFact=1117)
              )
```

Atmosfera din acest paragraf devenise lâncedă, aproape plictisitoare, când am observat accidental că, în Oracle, o eroare importantă scapă neobservată (şi nepedepsită). La prima rulare, interogarea de mai sus conţinea o greşeală în ultima subconsultare. În loc de

SELECT COUNT(DISTINCT CodPr) FROM *liniifact* **WHERE NrFact=1117**

iniţial era

SELECT COUNT(DISTINCT CodPr) FROM *facturi* **WHERE NrFact=1117.**

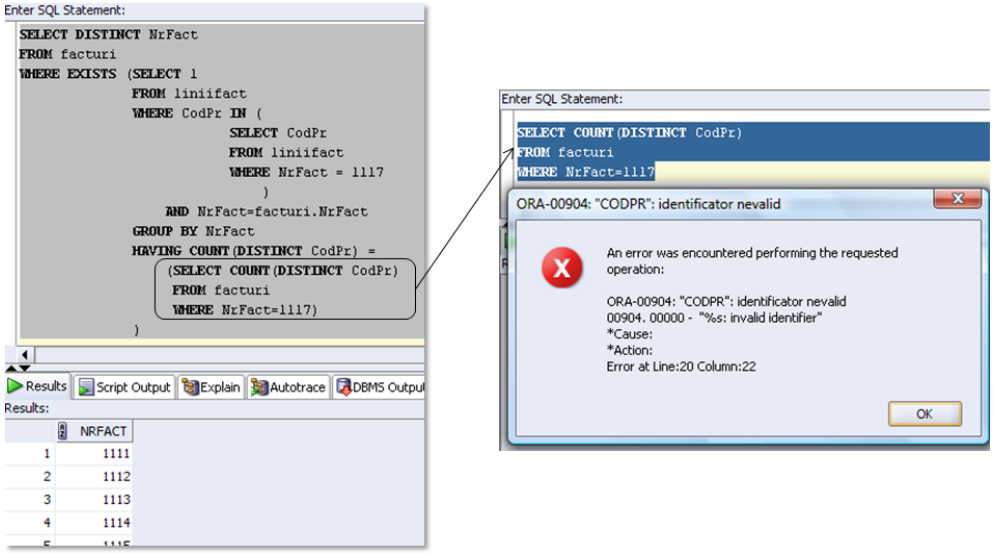


Figura 10.21. Alte surprize...suprize

Eroarea este grosolană deoarece tabela FACTURI nu are nici în clin, nici în mânecă, cu atributul CodPr, iar dacă am executa "singură" această subconsultare, am recepționa în plină figură un meritat mesaj de mustrare - vezi partea dreaptă a figurii 10.21. Ei bine, consultarea noastră "mare" se execută fără mustrări de conştiință – vezi partea stângă a figurii 10.21 – rezultatul fiind o bramburească.

Pentru ca lucrurile să fie şi mai interesante, aceeaşi problemă apare şi în PostgreSQL, DB2 şi în MS SQL Server. Aşa că mă întreb dacă chiar este o eroare (ştiți gluma cu numărul oamenilor care îți spun că eşti beat). Trecem peste nodul comun din papura celor de la IBM, Oracle, PostgreSQL şi Microsoft, şi exemplificăm corelarea fără EXISTS, luând în seamă o problemă aparent ciudată:

Care sunt produsele vândute la prețuri unitare superioare cu 14% mediei grupei din care fac parte, care sunt prețurile respective şi facturile în care apar.

Tabela PRODUSE conține, printre altele, şi un atribut tratat cu oarecare răceală până acum – Grupa. Aflăm media prețurilor unitare dintr-o grupă şi, apropiindu-ne de enunțul problemei, media prețurilor plus 14%, prin:

```
SELECT Grupa,
       TRUNC(AVG(PretUnit),2) AS "Media",
       TRUNC(AVG(PretUnit) * 1.14,2) AS "Media + 14%"
FROM liniifact lf1 INNER JOIN produse p1 ON lf1.CodPr=p1.CodPr
GROUP BY Grupa
```

Rezultatul – vezi figura 10.22 – ne interesează doar pentru a verifica dacă interogarea următoare este corectă. Oricum sintaxa era valabilă numai în Oracle şi PostgreSQL, în timp ce următoarea este valabilă în toate cele patru dialecte.

GRUPA	Media	Media + 14%
Dulciuri	1483,75	1691,47
Bere	968,82	1104,45
Tigari	3884	4427,76

Figura 10.22. Media preţurilor unitare de vânzare calculată pe grupe de produse

```
SELECT Grupa, DenPr, lf1.*
FROM liniifact lf1 INNER JOIN produse p1 ON lf1.CodPr=p1.CodPr
WHERE PretUnit >
        (SELECT AVG(PretUnit)
         FROM liniifact lf2 INNER JOIN produse p2 ON lf2.CodPr=p2.CodPr
         WHERE p2.Grupa = p1.Grupa
        ) * 1.14
ORDER BY Grupa, DenPr, PretUnit DESC
```

Ne interesează preţurile unitare ale produselor, dar şi facturile în care apar, aşa că liniile după care se face corelarea sunt cele ale joc̦iunii LINIIFACT-PRODUSE. Noutatea, în economia acestui paragraf, o aduce operatorul de corelare dintre consultarea principală şi subconsultarea ei, operator care este >. Cum clauza SELECT a subconsultării conţine funcţia agregat AVG (subconsultarea este scalară), interogarea funcţionează – vezi figura 10.23.

GRUPA	DENPR	NRFACT	LINIE	CODPR	CANTITATE	PRETUNIT
Bere	Produs 2	1120	1	2	80	1120
Dulciuri	Produs 4	1114	2	4	30	1705
Tigari	Produs 5	2121	1	5	550	7064
Tigari	Produs 5	1114	3	5	700	7064
Tigari	Produs 5	1121	1	5	550	7064
Tigari	Produs 5	1111	3	5	500	7060
Tigari	Produs 5	2111	3	5	510	7060
Tigari	Produs 5	3111	3	5	510	7060
Tigari	Produs 5	3119	4	5	755	6300
Tigari	Produs 5	2119	4	5	755	6300
Tigari	Produs 5	1119	4	5	750	6300

Figura 10.23. Produse vândute la preţuri superioare cu mai mult de 14% decât media grupei din care fac parte

10.3. Alte tipuri de corelări simple

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

În problema din paragraful 10.1 (vezi şi figura 10.6), diferenţa era calculată între ziua curentă şi ziua calendaristică 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 şi cele din precedenta zi de vânzări, adică între luni şi vineri s.a.m.d., după cum se observă în figura 10.24.

ZI	Vinzari_Zi_Curenta	Vinzari_Zi_Precedenta	Diferenta
01-08-2007	10599535	0	10599535
02-08-2007	277950	10599535	-10321585
03-08-2007	222050	277950	-55900
04-08-2007	201975	222050	-20075
07-08-2007	10610000,5	201975	10408025,5
14-08-2007	4723931,5	10610000,5	-5886069
15-08-2007	247757	4723931,5	-4476174,5
16-08-2007	467420	247757	219663
21-08-2007	10560939,5	467420	10093519,5
01-09-2007	4596401,5	10560939,5	-5964538
02-09-2007	238437,5	4596401,5	-4357964
07-09-2007	5819668	238437,5	5581230,5
10-09-2007	424704,5	5819668	-5394963,5
17-09-2007	179565	424704,5	-245139,5

Figura 10.24. Diferenţele dintre două zile consecutive de vânzări

```
SELECT DataFact AS Zi,
        SUM(Cantitate * PretUnit * (1+ProcTVA))  AS "Vinzari_Zi_Curenta ",
        (SELECT COALESCE(SUM(Cantitate * PretUnit * (1+ProcTVA)) ,0)
         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 ",
        SUM(Cantitate * PretUnit * (1+ProcTVA)) - COALESCE(
                (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
ORDER BY DataFact
```

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

Noutatea acestei soluții ține de corelarea, de data aceasta, a frazei SELECT principale cu subconsultarea subconsultării scalare (cea din clauza SELECT) – vezi figura 10.25. Nu ştiu ce ziceți dumneavoastră, dar mie-mi place…

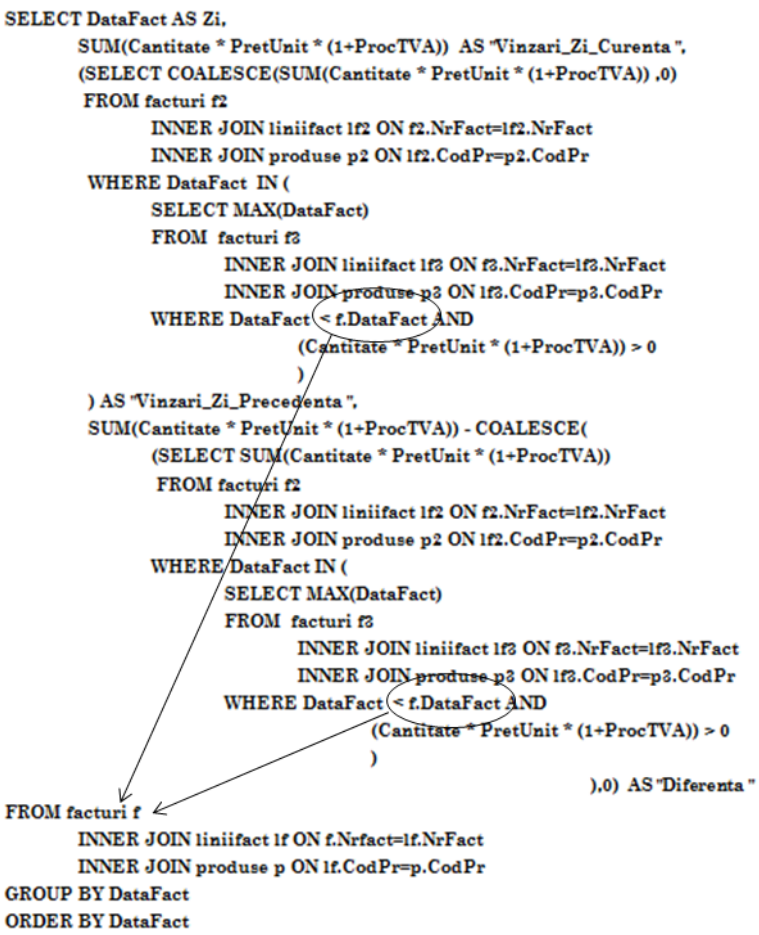


Figura 10.25. Corelarea liniilor tabelei FACTURI din SELECT-ul principal cu o subconsultare a subconsultării scalare

Care sunt facturile cu valori mai mari decât media zilei în care au fost întocmite?

E timpul unui nou experiment. Încercăm să corelăm o subconsultare din clauza HAVING cu grupurile interogării principale:

```
SELECT f1.NrFact, SUM(Cantitate * PretUnit * (1 + ProcTVA)) AS Valoare
FROM facturi f1
        INNER JOIN liniifact lf1 ON f1.NrFact=lf1.NrFact
        INNER JOIN produse p1 ON lf1.CodPr=p1.CodPr
GROUP BY f1.NrFact
HAVING SUM(Cantitate * PretUnit * (1 + ProcTVA)) >=
        (SELECT SUM(Cantitate * PretUnit * (1 + ProcTVA)) /
                COUNT(DISTINCT f2.NrFact)
```

```
FROM facturi f2
        INNER JOIN liniifact lf2 ON f2.NrFact=lf2.NrFact
        INNER JOIN produse p2 ON lf2.CodPr=p2.CodPr
WHERE f2.DataFact=f1.DataFact
)
```

Din păcate trucul nu funcționează în niciunul dintre SGBD-urile testate – vezi cazul PostgreSQL în figura 10.26. Noroc că mesajul de eroare este unul civilizat.

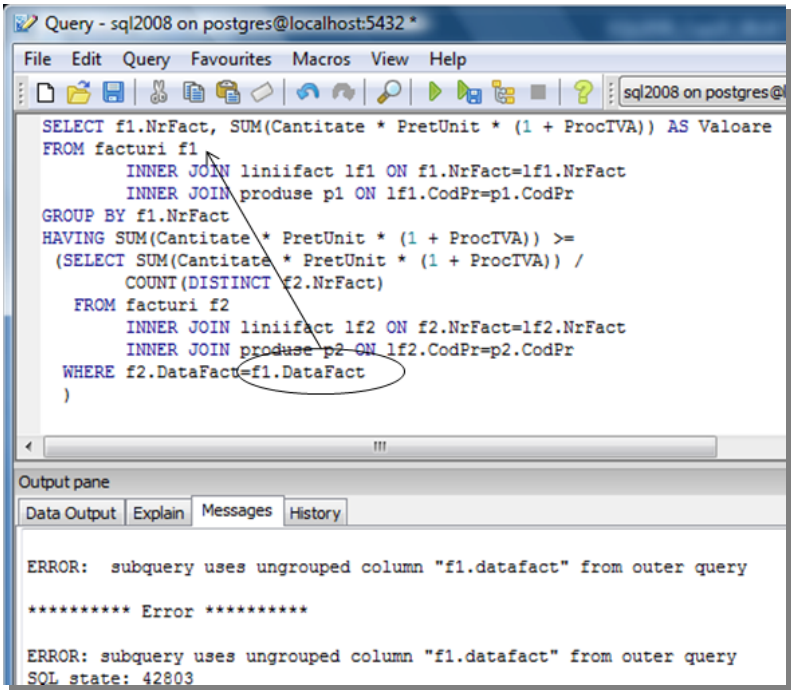


Figura 10.26. O corelare eşuată

10.4. Corelare dublă

Dubla corelare constituie o facilitate bine ocolită de către SQLişti (eu, unul, am reuşit s-o evit cu brio până în acest paragraf şi aşa voi face şi după aceea). Sheryl Larsen estima în 1998 că mai puţin de 10% dintre dezvoltatorii de aplicaţii în DB2 aveau cunoştinţă de interogările dublu corelate. Mărturisesc că în articolul lui Sheryl[9] am găsit cea mai bună explicaţie a acestui mecanism.

În cazul corelării simple, scenariul de execuţie este unul de tip top-bottom-top; la corelarea dublă lucrurile stau cu mult mai interesant: top-bottom-middle-bottom-middle-top. Mai simplu nici că se poate !

Dacă n-ar fi fost în joc diviziunea relaţională, n-am fi ajuns aici cu "hermeneutica" SQL. Revenim la cele două tabele, RD1 şi RD2, pe care le-am folosit în capitolul 4 la prezentarea diviziunii relaţionale. Valorile lui X care se găsesc în RD1 în combinaţie cu toate valorile lui Y din RD2 pot fi aflate şi prin interogarea următoare:

```
SELECT DISTINCT X
FROM rd1 T1_1
WHERE NOT EXISTS
        (SELECT 1
        FROM rd2
        WHERE NOT EXISTS
                (SELECT 1
                 FROM rd1 T1_2
                 WHERE T1_2.X=T1_1.X AND T1_2.Y=rd2.Y
                 )
        )
```

Execuţia frazei SQL se derulează astfel:

1. „Faza" 1:
   1.1. se "încarcă" prima linie din T1_1: *(x1, y1)* şi prima linie din RD2: *y1*
      1.1.1. se testează dacă există în T1_2 o linie în care *X=x1 (T1_1.X) şi Y=y1 (RD2.Y)*. Există !, astfel încât se trece la următoarea linie din RD2

[9] [Larsen1998]

1.1.2. se testează dacă există în T1_2 o linie în care *X=x1 (T1_1.X) şi Y=y2 (R2.Y)*. Există !, deci se "avansează" încă o linie în RD2
1.1.3. se testează dacă există în T1_2 o linie în care *X=x1 (T1_1.X) şi Y=y3 (R2.Y)*. Există !, deci se "avansează" încă o linie în RD2
1.1.4. se testează dacă există în T1_2 o linie în care *X=x1 (T1_1.X) şi Y=y5 (R2.Y)*. Există !, deci se "avansează" încă o linie în RD2
1.1.5. se testează dacă există în T1_2 o linie în care *X=x1 (T1_1.X) şi Y=y5 (R2.Y)*. Există !, deci se încearcă încarcarea următoarei linii din în RD2
1.1.6. Întrucât am ajuns la sfârşitul tabelei RD2 şi toate valorile din RD2 s-au regăsit în T1_2 în combinaţie cu x1,

1.2. Subconsultarea de jos (bottom) întoarce TRUE către subconsultarea din mijloc (middle). Subconsultarea din mijloc recepţionează rezultatul SELECT-ului de jos şi îl pasează cu aceeaşi valoare logică, sau cu valoare schimbată (dacă apare NOT), interogării principale. În această situaţie, SELECT-ul mijlociu primeşte de la cel de jos valoarea logică TRUE, însă, deoarece operatorul de conexiune mijloc-jos este NOT EXISTS, va întoarce SELECT-ului principal FALSE.
1.3. Fraza SELECT principală recepţionează 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** !

2. „Faza" 2:
2.1. se "încarcă" al doilea tuplu din T1_1: *(x2, y1)* şi primul tuplu din RD2 : *y1*
2.1.1. se testează dacă există în T1_2 o linie în care *X=x2 (T1_1.X) şi Y=y1 (RD2.Y)*. Există !, astfel încât se trece la următoarea linie din RD2
2.1.2. se testează dacă există în T1_2 o linie în care *X=x2 (T1_1.X) şi Y=y2 (RD2.Y)*. Nu există !, iar încărcarea celorlalte linii din RD2 este abandonată şi
2.2. subconsultarea de jos (bottom) întoarce FALSE către subconsultarea din mijloc (middle). SELECT-ul mijlociu recepţionează valoarea logică FALSE, şi, datorită operatorului NOT EXISTS, "pasează" SELECT-ului principal TRUE.
2.3. Fraza SELECT principală recepţionează 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 RD1. Clauza DISTINCT asigură preluarea 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 combinaţie, pe (măcar) o linie în RD1.*

În ce zile s-au vândut şi produsul cu denumirea "Produs 1" şi cel cu denumirea "Produs 2" ?

Soluţia bazată pe corelare dublă este probabil cea mai complicată, mult prea complicată pentru aşa o bagatelă de problemă:

```
SELECT DISTINCT DataFact
FROM facturi f1
        INNER JOIN liniifact lf1 ON f1.NrFact=lf1.NrFact
        INNER JOIN produse p1 ON lf1.CodPr=p1.CodPr
WHERE NOT EXISTS
        (SELECT 1
         FROM produse p1
         WHERE DenPr IN ('Produs 1', 'Produs 2') AND NOT EXISTS
                (SELECT 1
                 FROM facturi f2
                        INNER JOIN liniifact lf2 ON f2.NrFact=lf2.NrFact
                        INNER JOIN produse p2 ON lf2.CodPr=p2.CodPr
                 WHERE f2.DataFact=f1.DataFact AND p2.CodPr=p1.CodPr
                )
        )
```

Extrageţi clienţii pentru care există cel puţin câte o factură emisă în fiecare zi.

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

```
SELECT DISTINCT DenCl
FROM clienti c1 INNER JOIN facturi f1 ON c1.CodCl=f1.CodCl
WHERE DataFact BETWEEN DATE'2007-09-10' AND DATE'2007-09-30'
                AND NOT EXISTS
        (SELECT 1
         FROM facturi f2
         WHERE DataFact BETWEEN DATE'2007-09-10' AND DATE'2007-09-30'
                        AND NOT EXISTS
                (SELECT 1
                 FROM clienti c3 INNER JOIN facturi f3
                        ON c3.CodCl=f3.CodCl
                WHERE DataFact BETWEEN DATE'2007-09-10'
                        AND DATE'2007-09-30' AND
                c3.CodCl=c1.CodCl AND f3.DataFact=f2.DataFact
        )
  )
```

Toate soluţiile ce întrebuinţează dubla corelare de până acum au o concurenţă care le pune în umbră. Variantele ne-bazate pe dubla corelare sunt mai simple,

atât ca dimensiune, cât şi ca mod de înţelegere. Este timpul să scoatem din mânecă o problemă de diviziune relaţională care să demonstreze adevărata forţă a dublei corelări şi să justifice generosul efort intelectual pentru a o înţelege:

Ce facturi conţin măcar produsele din factura 1117 ?

```
SELECT DISTINCT NrFact
FROM liniifact lf1
WHERE NOT EXISTS
        (SELECT 1
        FROM liniifact lf2
        WHERE NrFact = 1117
        AND NOT EXISTS
                (SELECT 1
                FROM liniifact lf3
                WHERE lf3.CodPr =lf2.CodPr AND lf3.NrFact=lf1.NrFact ) )
```

Fraza SELECT analizată răspunde, de fapt, la întrebarea: *Pentru care facturi nu există nici un produs ce apare în factura 1117, dar nu este prezent în factura respectivă ?*

Fără dubla corelare, pot fi formulate şi soluţii bazate pe EXCEPT (sau minus în Orcle) şi NOT IN precum:

```
SELECT NrFact FROM liniifact
        EXCEPT
SELECT DISTINCT lf1.NrFact FROM liniifact lf1, liniifact lf2
WHERE lf2.NrFact = 1117 AND  (lf1.Nrfact, lf2.CodPr) NOT IN
        (SELECT NrFact, CodPr FROM liniifact)
```

şi (SQL Server):

```
SELECT NrFact FROM liniifact
        EXCEPT
SELECT DISTINCT lf1.NrFact FROM liniifact lf1, liniifact lf2
WHERE lf2.NrFact = 1117 AND
        CAST (lf1.Nrfact AS CHAR(8)) + CAST (lf2.CodPr AS CHAR(6)) NOT IN
                (SELECT CAST (NrFact AS CHAR(8)) + CAST (CodPr AS CHAR(6))
                 FROM liniifact)
```

Care sunt clienţii cărora li s-au vândut cel puţin produsele vândute clientului CLIENT 4 ?

Analizând enunţul problemei, *"Care sunt clienţii ?…"*, rezultă că atributul pentru corelarea consultării principale (top) şi celei de jos (bottom) este CodCl, iar din partea condiţională a enunţului *"… cel puţin produsele…"* pentru corelarea mijloc (middle) – jos (bottom) se utilizează atributul CodPr.

```
SELECT DenCl
FROM clienti c1
WHERE NOT EXISTS
```

```
 (SELECT 1
 FROM clienti c2
          INNER JOIN facturi f2 ON c2.CodCl=f2.CodCl
          INNER JOIN liniifact lf2 ON f2.NrFact=lf2.NrFact
WHERE DenCl='Client 4' AND NOT EXISTS
          (SELECT 1
          FROM  facturi f3
                    INNER JOIN liniifact lf3 ON f3.NrFact=lf3.NrFact
           WHERE c1.CodCl=f3.CodCl AND lf3.CodPr=lf2.CodPr)
)
```