

Capitolul 5. Sintaxa de bază a comenzii SELECT

Fără îndoială, cea mai gustată parte din SQL este cea legată de interogarea bazelor de date, adică de obținerea de informații dintre cele mai diverse, prin prelucrări, grupări etc. În SQL o interogare se formulează printr-o comandă¹ SELECT care are un format pe cât de simplu, pe atât de flexibil și puternic. Deși bazat pe algebra relațională, SQL-ul este cu mult mai complex. Este motivul pentru care paralela cu operatorii algebrici relaționali poate fi făcută doar până la un punct.

Din perspectiva prezentei lucrări, obiectivul principal al SQL constă în a oferi utilizatorului mijloacele necesare formulării unei consultări numai prin descrierea rezultatului dorit, cu ajutorul unei aserțiuni (expresie logică), fără a fi necesară și explicitarea modului efectiv în care se face căutarea în BD. Altfel spus, *utilizatorul califică (specifică) rezultatul, iar sistemul se ocupă de procedura de căutare*.

Prin comparație cu algebra relațională, SQL este nu numai mai generos, dar și-a luat o serie de libertăți care au stârnit mânia multor „puriști” ai modelului relațional, cum ar fi E.F. Codd², C.J. Date³ și F. Pascal⁴. Asupra unor „devieri” SQL-istice vom mai puncta pe parcurs. Unul dintre elementele-cheie care sugerează sorgintea relațională a SQL-ului este orientarea sa pe tupluri. Comanda SELECT pe care o vom întoarce pe toate părțile începând cu acest capitol, va obține, dintr-una, două sau mai multe tabele, un rezultat tabelar (o tabelă care e, de obicei și anonimă și temporară). SQL-ul nu lucrează la nivel de linie (individuală), ci de set de tupluri.

5.1. Trei clauze și un rezultat

Sintaxa frazei (comenzii) SELECT din SQL este grozav de simplă. Cele trei clauze principale sunt SELECT, FROM și WHERE⁵, din care numai primele două

¹ Vom vedea că sunt atât de multe opțiuni ce pot fi incluse într-o comandă SELECT, încât aceasta se poate întinde pe multe linii. Este, probabil, unul dintre motivele pentru care se folosește termenul *frază* SELECT

² Vezi [Codd 1989]

³ Vezi [Date 2004/05]

⁴ Vezi [Pascal 2000], dar și site-ul lui Fabian unde vitriolul este la el acasă: www.dbdebunk.com

⁵ La adresa <http://w3.one.net/~jhoffman/sqltut.htm> este un bun curs introductiv despre SQL

sunt obligatorii⁶. Presupunând că am dispune de o bază de date în care tabela STUDENȚI conține toți studenții facultății, pentru a obține lista celor înscriși în anul 3 la specializarea Informatică Economică comanda SELECT are sintaxa următoare:

```
SELECT NumePrenume
FROM studenti
WHERE an = 3 AND specializare = 'Informatică Economică '
```

Comanda încearcă să extragă din tabela STUDENTI toate liniile în care valoarea atributului An este 3 și, în același timp, cea a atributului Specializare este *Informatică Economică*. Pentru toate aceste linii extrase se afișează numai valoarea atributului NumePrenume.

Clauza SELECT corespunde deci operatorului proiecție din algebra relațională, fiind utilizată pentru desemnarea listei de atribute (coloanele) din rezultat. Clauza FROM este cea în care sunt enumerate tabelele din care vor fi extrase informațiile aferente consultării. Cu WHERE se desemnează predicatul selectiv al algebrei relaționale, relativ la atribute ale relațiilor care apar în clauza FROM. Prin execuția unei fraze SELECT se obține un rezultat de formă tabelară. Acesta poate fi o tabelă:

- temporară anonimă (creată implicit la execuția unei „fraze” SELECT ce nu are clauze speciale de stocare/direcționare a rezultatului);
- „obișnuită”:

```
CREATE TABLE tab_stud_an3_InfoEc AS
SELECT NumePrenume
FROM studenti
WHERE an = 3 AND specializare = 'Informatică Economică'
```

- temporară globală, cu structura stocată în dicționarul de date și conținut limitat la o sesiune sau la o tranzacție în care are loc invocarea tablei:

```
CREATE GLOBAL TEMPORARY TABLE gtt_stud_an3_InfoEc
AS...
```

- temporară locală care este și mai restrictivă decât tabele temporară globală în sensul dacă într-o sesiune/tranzacție logica aplicației presupune execuția mai multor blocuri de cod, fiecare bloc ce face referire la tabela temporară locală va avea o instanță proprie a tablei, independentă de celelalte blocuri:

```
CREATE LOCAL TEMPORARY TABLE ltt_stud_an3_InfoEc
AS...
```

⁶ În unele SCBD-uri (ex. PostgreSQL), chiar numai o clauză este obligatorie

- virtuală⁷:

```
CREATE VIEW view_stud_an3_InfoEc
```

```
AS...
```

În aplicații, uneori rezultatul este obținut și ca o variabilă masiv (tablou) sau set de înregistrări (recordset) necesare populării unei liste, combo (elemente ale unui formular) etc.

Atunci când clauza WHERE este omisă, se consideră implicit că predicatul P are valoarea logică "adevărat", astfel încât vor fi incluse în rezultat toate liniile din tabela, sau produsul cartezian al tabelelor, enumerată/enumerate în clauza FROM. Dacă, în locul coloanelor C1, C2, ... Cn, apare simbolul *, rezultatul va fi alcătuit din toate coloanele (atributele) relațiilor specificate în clauza FROM. De asemenea, atributele rezultatului preiau numele din tabela/tabelele specificate în FROM. Schimbarea numelui se realizează prin clauza AS.

În capitolul 2 era subliniată echivalența noțiunilor relație-tabelă. Conform restricției de unicitate, este interzisă repetarea unei tuplu într-o relație. SQL-ul este tolerant în această privință. Spre deosebire de algebra relațională, în SQL nu se elimină automat tuplurile identice (dublurile) din rezultat, ceea ce contravine uneia dintre poruncile fundamentale ale modelului relațional. Aceasta este vestea proastă. Vestea bună este că se poate folosi opțiunea DISTINCT:

```
SELECT DISTINCT C1, C2, ..., Cn
```

```
FROM R1, R2, ..., Rm
```

```
WHERE P
```

În concluzie, o frază SELECT, în forma de mai sus, corespunde mai multor operatori algebrici relaționali: selecție (clauza WHERE - P); proiecție (SELECT - Ci); produs cartezian (FROM - $R1 \otimes R2 \otimes \dots \otimes Rm$), și conduce la obținerea unui rezultat cu n coloane, fiecare coloană fiind un atribut din R1, R2, ..., Rm sau o expresie calculată pe baza unor atribute din R1, R2, ..., Rm.

Operatorii de bază folosiți pentru comparațiile dintre atribute/constante/expresii/funcții din clauza WHERE sunt cei arhicunoscuți:

- egal (=)
- diferit (<>)
- (strict) mai mic (<)
- mai mic sau egal (<=)
- (strict) mai mare (>)
- mai mare sau egal (>=)
-

⁷ Despre tabele temporare și virtuale o să discutăm un pic și prin capitolele 13 și 14.

Înainte de a trece la „conversia” în SQL a câtorva interogări din algebra relațională, pe baza exemplelor din capitolul 4, mai trebuie făcute câteva precizări. Mai întâi, deși nu e nicidecum obligatoriu, în cele mai multe interogări SQL vom scrie cuvintele rezervate (SELECT, FROM, WHERE, GROUP BY, HAVING, DISTINCT, SUM etc.) cu majuscule, numele tabelelor cu litere mici, iar numele atributelor cum s-o nimeri. Vom încerca să ilustrăm cu figuri rezultatele a cât mai multe interogări. Însă capturile de ecran vor fi preponderent din Oracle și PostgreSQL, ceea ce e oarecum nedrept pentru celelalte SGBD-uri luate în discuție. Pentru cei care nu au avut prea des prilejul de a lucra cu servere de baze de date, trebuie spus că ordinea de afișare a liniilor în rezultatul unei interogări poate fi uneori derutantă. În lipsa clauzei ORDER BY înregistrările sunt, inițial, afișate în ordinea introducerii în tabele. După actualizări (DELETE-uri, urmate de INSERT-uri sau UPDATE-uri), ordinea poate să pară ciudată, întrucât SGBD-ul „reciclează” spațiul eliberat la ștergerea unui tuplu⁸. Dar să revenim la exemplele din algebra relațională.

Exemplul 1 – selecție – paragraful 4.4.1 (figura 4.7)

Interogarea SQL care obține rezultatul din figura 4.7 este :

```
SELECT *  
FROM r1  
WHERE A > 20 AND C > 20
```

O variantă echivalentă este:

```
SELECT r1.* FROM r1 WHERE r1.A > 20 AND r1.C > 20
```

Din rațiuni de lizibilitate, de regulă, voi scrie fiecare clauză din comanda SELECT pe (cel puțin) o linie. Prefixarea atributelor (r1.A/r1.C/r1.*) nu este obligatorie în acest caz. Problema prefixării trebuie privită cu atenție atunci când în clauza FROM apar două sau mai multe tabele ce prezintă attribute cu nume identice, după cum vom vedea nu peste mult timp.

Folosind utilitarul SQL Developer distribuit gratuit de către cei de la Oracle, dialogul poate arăta ca în figura 5.1. Conexiunea dintre client și serverul de date a fost denumită SQL2008, iar rezultatul interogării are liniile numerotate (în exemplul nostru e o singură linie, așa încât nu prea a fost de lucru cu numerotarea).

⁸ Pentru detalii legate de organizarea fizică a înregistrărilor pe disc în Oracle, vezi și [Fotache s.a. 03]. Lucrurile stau oarecum similar în mai toate serverele majore de baze de date, orice carte de administrare de baze de date explicând organizarea bazei de date la nivelurile fizic și logic.

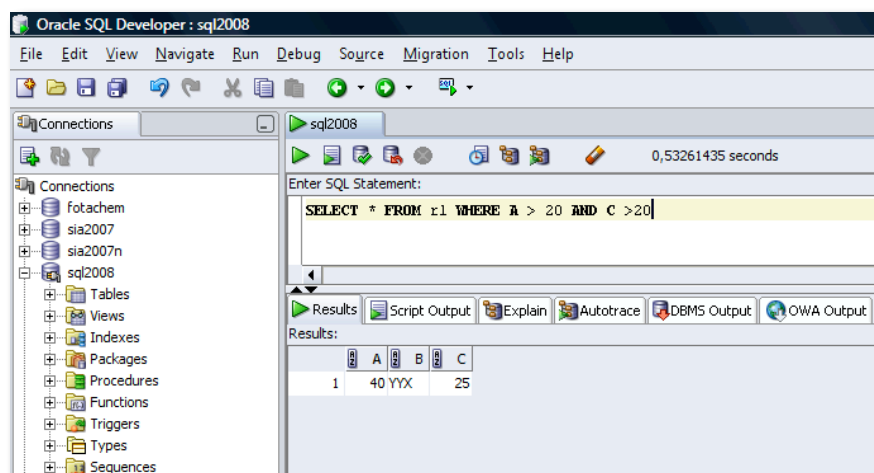


Figura 5.1. Interogare simplă executată în Oracle SQL Developer

Exemplul 2 – selecție (Care sunt județele din Moldova ?) - paragraful 4.4.1 (fig. 4.8)

Interogarea următoare funcționează fără modificări în toate SGBD-urile care pretind că au implementat SQL:

```
SELECT *
FROM judete
WHERE Regiune = 'Moldova'
```

După cum am văzut în precedentele capitole, utilitarul standard de conectare a unui client la serverul PostgreSQL-ul este pgAdmin. Execuția interogării de mai sus în pgAdmin se poate solda cu un rezultat de forma celui din figura 5.2.

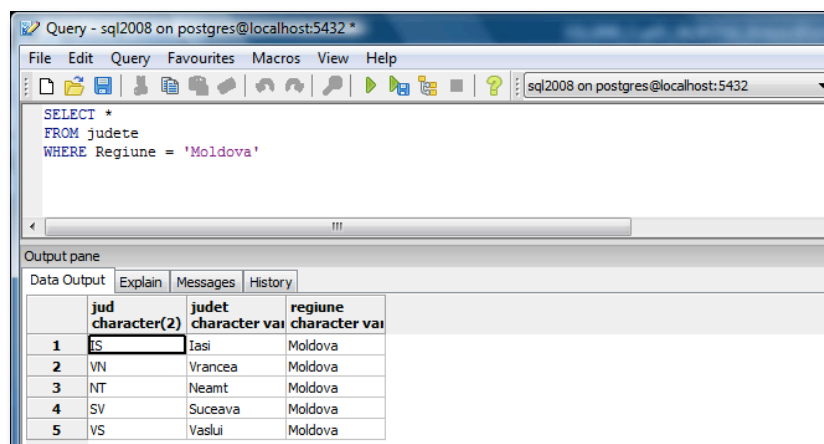


Figura 5.2. Interogare executată în pgAdmin (PostgreSQL)

Mai toate interfețele serverelor BD afișează un număr pentru fiecare linie din rezultat. Unele, precum Oracle SQL Developer sau Microsoft SQL Server Management Studio (vezi figura 5.3), afișează explicit valorile NULL.

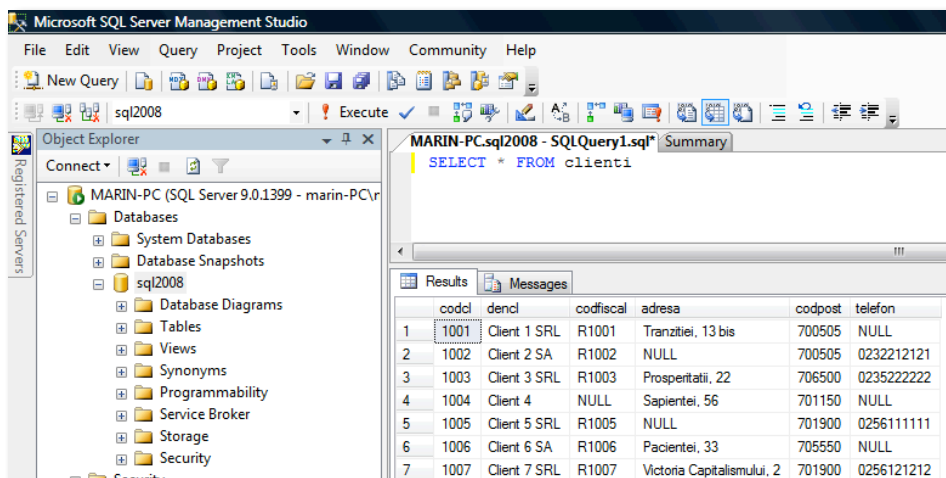


Figura 5.3. Afișarea explicită a valorilor NULL (cazul Microsoft SQL Server)

Altele, precum PostgreSQL sau DB2 lasă spații libere (vezi figura 5.4), astfel încât este destul de greu să ne dăm seama unde este un NULL sau unul sau mai multe spații.

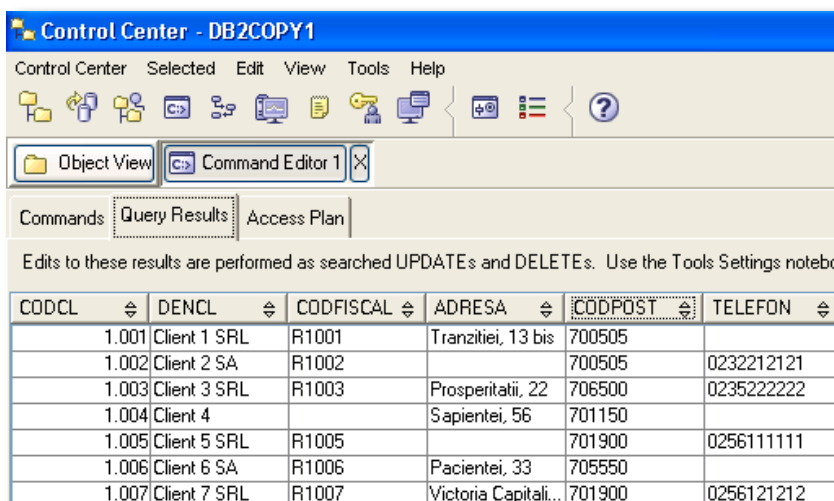


Figura 5.4. Afișarea defectoasă (de fapt, ne-afișarea) a valorilor NULL (cazul DB2 Control Center)

Exemplul 3 – selecție (Care sunt facturile emise în perioada 2-5 august 2007 ?) - paragraful 4.4.1 (figura 4.9)

Formatul standard al unei constante de tip dată calendaristică este *DATE'YYYY-MM-DD'*, așa încât interogarea se prezintă astfel:

```
SELECT *
FROM facturi
WHERE DataFact >= DATE'2007-08-02' AND DataFact <= DATE'2007-08-05'
```

În unele SGBD-uri, însă (ex. SQL Server), acest format de dată nu este recunoscut implicit, fiind necesară configurarea sesiunii curente pe formatul dorit sau utilizarea unor clauze sau funcții suplimentare. Altele, precum Oracle sau PostgreSQL, acceptă standardul *DATE'YYYY-MM-DD'*, dar folosesc, de multe versiuni încoace, funcții proprii, precum *TO_DATE*:

```
SELECT *
FROM facturi
WHERE DataFact >= TO_DATE('02/08/2007', 'DD/MM/YYYY') AND
DataFact <= TO_DATE('05/08/2007', 'DD/MM/YYYY')
```

În DB2 și SQL Server niciunul dintre formatele de mai sus nu este recunoscut, însă lucrul nu este prea deranjant, întrucât literalii (constantele) de tip dată calendaristică se încadrează între apostrofuri pe formatul *'YYYY-MM-DD'*:

```
SELECT *
FROM facturi
WHERE DataFact >= '2007-08-02' AND DataFact <= '2007-08-05'
```

Cel mai generos este PostgreSQL în care funcționează inclusiv această ultimă versiune a interogării.

Exemplul 4 – proiecție - paragraful 4.4.2 (figura 4.10)

```
SELECT A,C
FROM r1
```

Cele trei linii ale rezultatului din figura 4.10 se obțin întocmai și în SQL, întrucât valorile combinației (A,C) nu se repetă în R1.

Exemplul 5 (Care sunt regiunile țării preluate în bază ?) - paragraful 4.4.2 (figura 4.11)

Acum apare o primă diferențiere de rezultatul din algebra relațională. După cum aminteam câteva zeci de rânduri mai sus, spre deosebire de algebra relațională, SQL nu elimină automat dublurile. Tabela obținută prin consultarea:

```
SELECT Regiune
FROM judete
```

are structura și conținutul identice cu *R'* din figura 4.11). Pentru a obține răspunsul de forma tabeli *R* (o regiune să apară o singură dată în răspuns) se folosește clauza *DISTINCT*:

```
SELECT DISTINCT Regiune
FROM judete
```

Exemplul 6 (Care sunt: codul, denumirea și numărul de telefon ale fiecărui client ?) - paragraful 4.4.2 (figura 4.12)

```
SELECT CodCl, DenCl, Telefon  
FROM clienti
```

Nu este necesară clauza DISTINCT, deoarece CodCl este cheia primară a tabelului CLIENTI.

Exemplul 7 (Care este numărul de telefon al clientului Client 2 SA ?) - paragraful 4.4.3 (figura 4.13)

Dacă în algebra relațională aveam nevoie de înlănțuirea a doi operatori, SQL-ul este mai generos, dat fiind formatul comenzii SELECT:

```
SELECT Telefon  
FROM clienti  
WHERE DenCl = 'Client 2 SA'
```

Exemplul 8 (Care sunt denumirile și codurile poștale ale localităților (prezente în bază) din județele Iași (IS) și Vrancea (VN) ?) - paragraful 4.4.3 (figura 4.14)

```
SELECT Loc, CodPost  
FROM coduri_postale  
WHERE Jud = 'IS' OR Jud = 'VN'
```

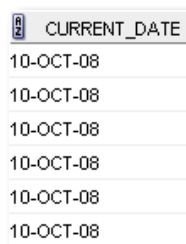
Paragraful 5.3 va fi dedicat coloanelor calculate pe baza unor expresii în care sunt implicate atribute din tabele în combinație cu diverse constante (literal) și funcții. Până atunci însă, ne întrebăm: ce se afișează la execuția următoarei interogări ?⁹

```
SELECT CURRENT_DATE FROM judete
```

CURRENT_DATE este funcția SQL prin care se returnează data sistemului (calculatorului). Majoritatea dialectelor SQL au clauza FROM obligatorie într-o comandă SELECT. Eu am scris, aproape la întâmplare, în clauza FROM una dintre tabele bazei de date. Cum procedează SGBD-ul ? Simplu: (1) se verifică existența tabelului JUDETE; (2) se extrag din tabela JUDETE numai liniile care satisfac condiția din clauza WHERE; aici clauza WHERE lipsește, deci se extrag toate liniile tabelului; (3) pentru toate liniile extrase la punctul anterior, se afișează coloanele din clauza SELECT; chiar dacă pare ciudat, SQL-ul nu se supără că în clauza SELECT nu apare nici un atribut din JUDETE. Pur și simplu, se va afișa data curentă pe un număr de rânduri egal cu numărul liniilor tabelului – vezi figura 5.5 (inutil de amintit

⁹ În Microsoft SQL Server varianta executabilă este: SELECT GETDATE() FROM judete.

că această tulburătoare interogare a fost executată ultima oară pe 10 octombrie 2008).



CURRENT_DATE
10-OCT-08
10-OCT-08
10-OCT-08
10-OCT-08
10-OCT-08
10-OCT-08
10-OCT-08

Figura 5.5. Interogare ce afișează (repetat) data curentă

Ca să reducem numărul liniilor din rezultat la unu, am putea folosi clauza DISTINCT:

```
SELECT DISTINCT CURRENT_DATE FROM judete
```

Deși funcțională, soluția este una destul de stupidă. Motiv pentru care diverși producători de SGBD-uri s-au gândit la variante mai acceptabile. Spre exemplu, în PostgreSQL este posibilă renunțarea la clauza FROM, interogarea:

```
SELECT CURRENT_DATE
```

fiind perfect funcțională. Microsoft SQL Server nu are implementată funcția CURRENT_DATE, ci numai CURRENT_TIMESTAMP, iar, ca și în PostgreSQL, clauza FROM poate lipsi:

```
SELECT GETDATE()
```

sau

```
SELECT CURRENT_TIMESTAMP
```

Alte SGBD-uri, însă, au recurs la soluția creării automate (la instalare) a unei tabele cu o singură linie și un singur atribut, astfel încât, în lipsa clauzei WHERE rezultatul va conține, cu siguranță, un singur rând. În Oracle, această tabelă se numește DUAL, iar în DB2 SYSIBM.SYSDUMMY1, interogarea fiind în Oracle:

```
SELECT CURRENT_DATE FROM dual
```

iar în DB2:

```
SELECT CURRENT_DATE FROM SYSIBM.SYSDUMMY1
```

De la versiunea 9.5, tabela DUAL există și în DB2 (în schema SYSIBM), așa că este corectă și sintaxa:

```
SELECT CURRENT_DATE FROM SYSIBM.dual
```

5.2.Reuniuni, intersecții, diferențe, produse carteziane

Cei patru operatori ansamblați din algebra relațională au fost transpuși în SQL într-o sintaxă destul de ușor de înțeles. Apar, totuși, câteva discuții legate de titulatura unor operatori în diverse dialecte, plus eterna problemă a liniilor duplicat.

5.2.1. Reuniunea

Un rezultat identic cu tabela R3 din figura 4.3 (paragraful 4.3.1) se obține prin fraza SELECT următoare:

```
SELECT *
FROM r1
UNION
SELECT *
FROM r2
```

La reuniunea a două tabele, SQL elimină automat liniile identice din rezultat. Dacă se dorește preluarea tuturor liniilor celor două relații, și, implicit, apariția de linii duplicate (ceea ce, reamintim, contrazice un principiu de bază al modelului relațional) se folosește clauza ALL astfel:

```
SELECT *
FROM r1
UNION ALL
SELECT *
FROM r2
```

Rezultatul (vezi figura 5.6) conține câte o linie pentru fiecare tuplu din cele două tabele.

R Z	A	R Z	B	R Z	C
	20	XYZ		30	
	30	XXZ		20	
	40	YYX		25	
	25	XYZ		30	
	40	YYX		25	
	30	XXZ		40	

Figura 5.6. Rezultatul operatorului UNION ALL

Revenim la exemplul 8 (*Care sunt denumirile și codurile poștale ale localităților (prezente în bază) din județele Iași (IS) și Vrancea (VN) ?*) - paragraful 4.4.3 (figura 4.14) pe care l-am discutat acum două pagini. Dacă soluția precedentă se baza pe un predicat ce folosea operatorul logic *sau* (OR), acum suntem în măsură să formulăm și o soluție bazată pe reuniune:

```
SELECT Loc, CodPost
FROM coduri_postale
WHERE Jud = 'IS'
UNION
SELECT Loc, CodPost
FROM coduri_postale
WHERE Jud = 'VN'
```

5.2.2. Intersecția

Raportându-ne la exemplul din paragraful 4.3.2 (figura 4.4), echivalentul tabeli R4 se obține în SQL prin:

```
SELECT *  
FROM r1  
INTERSECT  
SELECT *  
FROM r2
```

Standardele SQL permit, ca și în cazul reuniunii, prezența repetată în rezultat a unor linii (tupluri duplicate), bineînțeles atunci când cele două relații asupra cărora se aplică intersecția nu respectă restricția de unicitate:

```
SELECT *  
FROM r1  
INTERSECT ALL  
SELECT *  
FROM r2
```

Dacă tuplul t1 apare repetat și în R1 și în R2, mai precis, de n ori în R1 și de m ori în R2, în rezultatul operatorului INTERSECT t1 apare o singură dată, în timp ce utilizând INTERSECT ALL t1 va apărea de un număr de ori care este minimumul dintre n și m. INTERSECT ALL nu funcționează nici în Oracle, nici în MS SQL Server, în timp ce în DB2 și PostgreSQL INTERSECT ALL furnizează același rezultat ca INTERSECT !

Pentru o primă ilustrare a utilizării intersecției, transcriem soluția din algebra relațională formulată în paragraful 4.3.3, exemplul 9 (vezi figura 4.16) - *Care sunt codurile produselor care apar simultan și în factura 1111 și în factura 1117 ?*:

```
SELECT CodPr  
FROM liniifact  
WHERE NrFact = 1111  
INTERSECT  
SELECT CodPr  
FROM liniifact  
WHERE NrFact = 1117
```

Exemplele de mai sus funcționează în DB2, Oracle, și PostgreSQL, nu însă și în alte SGBD-uri (ex. Visual FoxPro sau Access) care nu are implementat operatorul INTERSECT, astfel încât intersecția trebuie realizată prin alte clauze și operatori.

5.2.3. Diferența

Operatorul așteptat ar fi MINUS, titulatură sub care a fost implementat în Oracle. În standardele SQL și în alte câteva SGBDR-uri (precum DB2), titulatura

operatorului pentru diferență (scădere relațională) nu e MINUS, ci EXCEPT, iar în alte SGBD-uri (Visual FoxPro, Access) nu există nici unul, nici altul. Tabela R5 din figura 4.5 (paragraful 4.3.3), calculată prin expresia $R1-R2$, se obține în SQL (și PostgreSQL/DB2/MS SQL Server) prin interogarea:

```
SELECT *
FROM r1
EXCEPT
SELECT *
FROM r2
```

Ca și în cazul reuniunii și intersecției, eliminarea tuplurilor duplicate se face automat, iar repetarea lor se asigură prin EXCEPT ALL. În DB2 și PostgreSQL, dacă INTERSECT ALL avea o funcționalitate identică INTERSECT-ului, EXCEPT ALL funcționează „corect”. În Oracle MINUS ALL nu este implementat, ca și EXCEPT ALL în SQL Server.

Modificăm enunțul ultimului exemplu din paragraful precedent: Care sunt codurile produselor care apar în factura 1111, dar nu apar în factura 1117 ?

```
SELECT CodPr
FROM liniifact
WHERE NrFact = 1111
EXCEPT
SELECT CodPr
FROM liniifact
WHERE NrFact = 1117
```

Execuția acestei consultări (în PostgreSQL) se soldează cu rezultatul din figura 5.7.

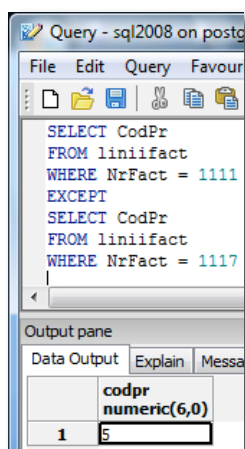


Figura 5.7. Codul produsului care apare în factura 1111, dar nu apare în 1117 (execuție în PostgreSQL pgAdmin)

Care sunt facturile care nu sunt încasate nici măcar parțial ?

Practic, ne interesează facturile ce nu prezintă nicio tranșă de încasare. Dacă tabela FACTURI conține facturile emise, tranșele de facturi încasate se află în tabela INCASFACT. Rezultatul acestei probleme (figura 5.8) poate fi aflat prin diferența dintre facturile emise și cele cu încasări. Asigurarea uni-compatibilității rezultatelor celor două fraze SELECT conectate prin EXCEPT va fi asigurată prin selectarea unui singur atribut – NrFact:

```
SELECT DISTINCT NrFact
FROM liniifact
EXCEPT
SELECT NrFact
FROM incasfact
```

NrFact
1114
1115
1116
1119
1121
1122
2111
2112
2113
2115
2116
2117
2118
2119
2121
2122
3111
3112
3113
3115
3116
3117
3118
3119

Figura 5.8. Facturile fără nici cea mai mică încasare

5.2.4. Produsul cartezian

Standardele SQL-89 și SQL-92 nu puneau la dispoziție vreun operator special dedicat produsului cartezian. Nici nu era mare nevoie. Tabela R6 din figura 4.6 (paragraful 4.3.4) se poate obține, pur și simplu, prin enumerarea celor două relații în clauza FROM:

```
SELECT *
FROM r1, r2
```

Din SQL:1999 apare însă operatorul CROSS JOIN implementat în toate cele patru servere de baze de date:

```
SELECT *
FROM r1 CROSS JOIN r2
```

5.3. Coloane-expresii

O facilitate importantă în multe interogări SQL ține de definirea, pe lângă attributele tabelelor, a unor coloane noi, pe baza unor expresii. Despre funcțiile aplicate atributelor sau constantelor și altor funcții vom discuta în capitolul următor. Clauza AS permite denumirea coloanelor calculate, sau redenumirea unor coloane ale tabelelor. Dacă numele coloanei conține spații sau alte caractere neautorizate, trebuie încadrat între ghilimele (nu între apostrofuri, precum literalii șir de caractere)¹⁰. Să luăm un exemplu banal: *Care este, pentru fiecare produs din factura 1111, codul, cantitatea, prețul unitar și valoarea fără TVA ?*

```
SELECT CodPr AS "Cod produs",
       Cantitate AS "Cantitate",
       PretUnit AS "Pret Unitar",
       Cantitate * PretUnit AS "Valoare fara TVA"
FROM liniifact
WHERE NrFact = 1111
```

Rezultatul interogării este prezentat în figura 5.9.

Cod produs	Cantitate	Pret Unitar	Valoare fara TVA
1	50	1000	50000
2	75	1050	78750
5	500	7060	3530000

Figura 5.9. Exemplu de coloană calculată

A patra coloana este denumită *Valoare fara TVA*, după cum a fost specificat în clauza AS. Valorile sale sunt determinate pe baza expresiei *Cantitate * PretUnit*. Acum, că am arătat cum se poate modifica numele unei coloane, vom reveni la starea de comoditate și vom folosi destul de rar spații sau alte caractere (decât underscore) pentru a evita folosirea ghilimelelor.

¹⁰ În SQL Server, în locul ghilimelelor se pot folosi parantezele drepte (Ex: [Pret Unitar])

Atenție, însă ! Dacă numele definit prin clauza AS poate fi folosit la ordonare, includerea sa în predicatul din clauza WHERE se soldează cu o eroare. Spre exemplu, dintre liniile facturii 1111 ne interesează doar cele în care valoarea fără TVA este mai mare decât 60000. Interogarea:

```
SELECT CodPr AS "Cod produs", Cantitate AS "Cantitate",  
       PretUnit AS "Pret Unitar", Cantitate * PretUnit AS "Valoare fara TVA"  
FROM liniifact  
WHERE NrFact = 1111 AND "Valoare fara TVA" > 60000
```

se va solda cu un mesaj de eroare – vezi figura 5.10.

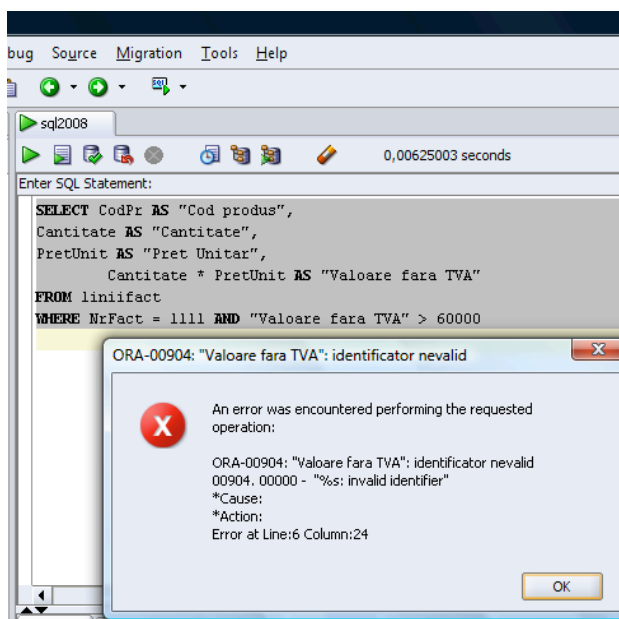


Figura 5.10. Eroare generată de folosirea numelui unei coloane calculate în clauza WHERE

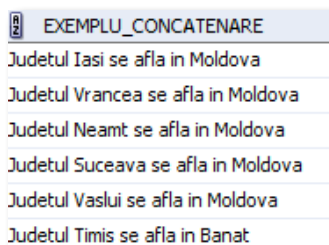
Versiunea corectă presupune includerea în predicatul de selecție nu a numelui, ci a expresiei ce definește coloana respectivă:

```
SELECT CodPr AS "Cod produs",  
       Cantitate AS "Cantitate",  
       PretUnit AS "Pret Unitar",  
       Cantitate * PretUnit AS "Valoare fara TVA"  
FROM liniifact  
WHERE NrFact = 1111 AND Cantitate * PretUnit > 60000
```

WHERE NrFact = 1111 AND Cantitate * PretUnit > 60000

Un alt tip de expresie este cel bazat pe concatenare, adică pe alipirea mai multor constante, variabile și attribute într-o coloană nouă. Operatorul SQL pentru concatenare este alcătuit din două bare verticale (||). Spre exemplu, interogarea următoare produce rezultatul din figura 5.11¹¹.

```
SELECT 'Judetul ' || Judet || ' se afla in ' || Regiune
      AS Exemplu_Concatenare
FROM judete
```



EXEMPLU_CONCATENARE
Judetul Iasi se afla in Moldova
Judetul Vrancea se afla in Moldova
Judetul Neamt se afla in Moldova
Judetul Suceava se afla in Moldova
Judetul Vaslui se afla in Moldova
Judetul Timis se afla in Banat

Figura 5.11. Concatenarea unor literali și attribute șir de caractere

SQL este un limbaj puternic tipizat, în sensul că pot fi combinați în expresii numai operatori de același tip. Conversia unui operator dintr-un tip în altul (acolo unde este posibil) se face cu ajutorul funcției CAST. În unele SGBD-uri, însă (ex. PostgreSQL și Oracle), există o mai mare libertate. Astfel, se pot concatena direct, adică fără conversie prealabilă, literali, attribute-șir de caractere, numerice etc. Alte SGBD-uri, precum DB2, necesită transformarea valorilor non-șir de caractere (attribute/constante numerice, dată calendaristică etc.), operațiune realizată prin funcția CAST. Spre exemplu, în Oracle și PostgreSQL interogarea următoare este funcțională:

```
SELECT 'Factura ' || NrFact || ' a fost emisa pe data ' ||
      DataFact AS Concatenare_Oracle_PgSQL
FROM facturi
```

în timp ce DB2 nu o acceptă. Folosind funcția de conversie CAST, putem redacta următoarea variantă a interogării care funcționează DB2, dar și în PostgreSQL și Oracle:

```
SELECT 'Factura ' || CAST (NrFact AS CHAR(8)) ||
      ' a fost emisa pe data ' || CAST (DataFact AS VARCHAR(10))
      AS Concatenare_DB2_PgSQL
FROM facturi
```

¹¹ În MS SQL Server, pentru concatenare se folosește simbolul +

Valorile atributului NrFact sunt transformate în șiruri de caractere de lungime fixă (8 poziții), în timp ce ale DataFact vor fi convertite în șiruri de caractere de lungime variabilă (vorba vine, deoarece formatul datei este unitar pentru toate liniile rezultatului) de maximum 10 poziții. După cum se observă în figura 5.12, dacă PostgreSQL convertește valoarea NrFact pe un șir de caractere de lungimea numărului de cifre, în DB2 la conversie se completează cu zerouri la stânga până la completarea celor opt poziții (*CAST NrFact AS VARCHAR(8)*).

concatenare_pgsq text	CONCATENARE_DB2
Factura 1111 a fost emisa pe data 2007-08-01	Factura 00001111 a fost emisa pe data 2007-08-01
Factura 1112 a fost emisa pe data 2007-08-01	Factura 00001112 a fost emisa pe data 2007-08-01
Factura 1113 a fost emisa pe data 2007-08-01	Factura 00001113 a fost emisa pe data 2007-08-01
Factura 1114 a fost emisa pe data 2007-08-01	Factura 00001114 a fost emisa pe data 2007-08-01
Factura 1115 a fost emisa pe data 2007-08-02	Factura 00001115 a fost emisa pe data 2007-08-02
Factura 1116 a fost emisa pe data 2007-08-02	Factura 00001116 a fost emisa pe data 2007-08-02
Factura 1117 a fost emisa pe data 2007-08-03	Factura 00001117 a fost emisa pe data 2007-08-03
Factura 1118 a fost emisa pe data 2007-08-04	Factura 00001118 a fost emisa pe data 2007-08-04
Factura 1119 a fost emisa pe data 2007-08-07	Factura 00001119 a fost emisa pe data 2007-08-07
Factura 1120 a fost emisa pe data 2007-08-07	Factura 00001120 a fost emisa pe data 2007-08-07
Factura 1121 a fost emisa pe data 2007-08-07	Factura 00001121 a fost emisa pe data 2007-08-07
Factura 1122 a fost emisa pe data 2007-08-07	Factura 00001122 a fost emisa pe data 2007-08-07
Factura 2111 a fost emisa pe data 2007-08-14	Factura 00002111 a fost emisa pe data 2007-08-14
Factura 2112 a fost emisa pe data 2007-08-14	Factura 00002112 a fost emisa pe data 2007-08-14
Factura 2113 a fost emisa pe data 2007-08-14	Factura 00002113 a fost emisa pe data 2007-08-14
Factura 2115 a fost emisa pe data 2007-08-15	Factura 00002115 a fost emisa pe data 2007-08-15

Figura 5.12. O diferență la conversia număr-șir de caractere între PostgreSQL și DB2

SQL Server folosește semnul plus pentru concatenare și, similar DB2-ului, reclamă transformarea explicită a argumentelor în șiruri de caractere:

```
SELECT 'Factura ' + CAST (NrFact AS CHAR(8)) +
      ' a fost emisa pe data ' + CAST (DataFact AS VARCHAR(11))
      AS Concatenare_SQLServer
FROM facturi
```

În finalul discuției despre coloanele-expresii, mai zăbovim preț de câteva rânduri la expresiile de tip dată calendaristică. Modurile în care au fost implementate aceste funcțiuni sunt foarte eterogene de la SGBD la SGBD. Să presupunem că orice factură trebuie încasată în maximum două săptămâni de la data emiterii. DataFact este un atribut de tip DATE, așa că numărul 14 din expresia ce calculează scadența este „tratat” ca număr de zile în toate cele patru SGBD-uri, cu excepția DB2:

```
SELECT NrFact AS Factura,
      DataFact AS Data_Facturare,
      DataFact + 14 AS Scadenta_Incasare
FROM facturi
```

Specificarea intervalelor calendaristice este o sarcină relativ ușoară. Astfel, interogarea următoare este redactată în PostgreSQL și furnizează același rezultat (vezi figura 5.13) în privința datei scadente:

```
SELECT NrFact AS Factura, DataFact AS Data_Facturare,
       DataFact + 14 AS Scadenta_Incasare1,
       DataFact + INTERVAL '14 DAYS' AS Scadenta_Incasare2,
       DataFact + INTERVAL '2 WEEKS' AS Scadenta_Incasare3
FROM facturi
```

factura numeric(8,0)	data_factura date	scadenta_incasare1 date	scadenta_incasare2 timestamp without time zone	scadenta_incasare3 timestamp without time zone
1111	2007-08-01	2007-08-15	2007-08-15 00:00:00	2007-08-15 00:00:00
1112	2007-08-01	2007-08-15	2007-08-15 00:00:00	2007-08-15 00:00:00
1113	2007-08-01	2007-08-15	2007-08-15 00:00:00	2007-08-15 00:00:00
1114	2007-08-01	2007-08-15	2007-08-15 00:00:00	2007-08-15 00:00:00
1115	2007-08-02	2007-08-16	2007-08-16 00:00:00	2007-08-16 00:00:00
1116	2007-08-02	2007-08-16	2007-08-16 00:00:00	2007-08-16 00:00:00
1117	2007-08-03	2007-08-17	2007-08-17 00:00:00	2007-08-17 00:00:00
1118	2007-08-04	2007-08-18	2007-08-18 00:00:00	2007-08-18 00:00:00
1119	2007-08-07	2007-08-21	2007-08-21 00:00:00	2007-08-21 00:00:00
1120	2007-08-07	2007-08-21	2007-08-21 00:00:00	2007-08-21 00:00:00
1121	2007-08-07	2007-08-21	2007-08-21 00:00:00	2007-08-21 00:00:00
1122	2007-08-07	2007-08-21	2007-08-21 00:00:00	2007-08-21 00:00:00

Figura 5.13. De trei ori scadență (primele linii din rezultat)

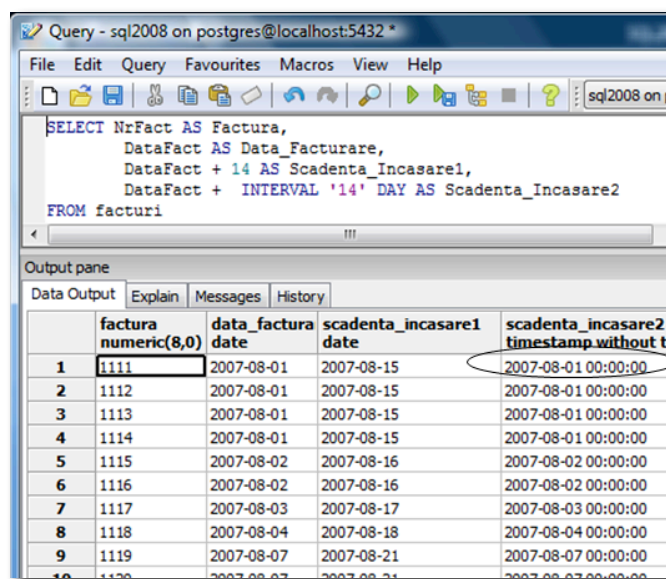
DB2 este mai pretențios, nepermițând nici expresia de la *Scadenta_Incasare1*, nici de la *Scadenta_Incasare3* (din cauza clauzei WEEKS), ci doar:

```
SELECT NrFact AS Factura,
       DataFact AS Data_Facturare,
       DataFact + 14 DAYS AS Scadenta_Incasare2
FROM facturi
```

În Oracle versiunea funcțională a interogării PostgreSQL de mai sus este:

```
SELECT NrFact AS Factura,
       DataFact AS Data_Facturare,
       DataFact + 14 AS Scadenta_Incasare1,
       DataFact + INTERVAL '14' DAY AS Scadenta_Incasare2
FROM facturi
```

intervalele de tip săptămâni (WEEK) nefiind implementate. Observați diferența la specificarea intervalului în zile. Aceeași interogare executată în PostgreSQL ne dă prilejul descoperirii unei ciudățenii a produsului – vezi figura 5.14. Deși nu ni se „spune” că am fi greșit undeva la sintaxă, a doua coloană-scadență (Scadență_Incasare2) are valori greșite – de fapt, valorile datei întocmirii fiecărei facturi. Din păcate, trebuie să avem grijă tot timpul, de aici încolo, ca în PostgreSQL la calificarea intervalelor să se folosească notația INTERVAL 'n DAYS | WEEKS...'.



```

SELECT NrFact AS Factura,
       DataFact AS Data_Facturare,
       DataFact + 14 AS Scadenta_Incasare1,
       DataFact + INTERVAL '14' DAY AS Scadenta_Incasare2
FROM facturi

```

	factura numeric(8,0)	data_factura date	scadenta_incasare1 date	scadenta_incasare2 timestamp without time zone
1	1111	2007-08-01	2007-08-15	2007-08-01 00:00:00
2	1112	2007-08-01	2007-08-15	2007-08-01 00:00:00
3	1113	2007-08-01	2007-08-15	2007-08-01 00:00:00
4	1114	2007-08-01	2007-08-15	2007-08-01 00:00:00
5	1115	2007-08-02	2007-08-16	2007-08-02 00:00:00
6	1116	2007-08-02	2007-08-16	2007-08-02 00:00:00
7	1117	2007-08-03	2007-08-17	2007-08-03 00:00:00
8	1118	2007-08-04	2007-08-18	2007-08-04 00:00:00
9	1119	2007-08-07	2007-08-21	2007-08-07 00:00:00
10	1120	2007-08-07	2007-08-21	2007-08-07 00:00:00

Figura 5.14. Un prim ghimpe în PostgreSQL

Dacă am presupune că scadența ar fi peste două luni de la facturare, interogarea ar trebui modificată astfel în PostgreSQL:

```

SELECT NrFact AS Factura, DataFact AS Data_Facturare,
       DataFact + INTERVAL '2 MONTHS' AS Scadenta_Incasare
FROM facturi

```

Varianta următoare funcționează și în Oracle și în PostgreSQL, dar rezultatul corect este furnizat numai în Oracle.

```

SELECT NrFact AS Factura, DataFact AS Data_Facturare,
       DataFact + INTERVAL '2' MONTH AS Scadenta_Incasare
FROM facturi

```

Din păcate, SQL Server-ul de la Microsoft nu are implementat tipul INTERVAL și nici clauzele de mai sus, operațiunile de adunare și scădere între date și intervale calendaristice fiind realizate cu funcții specifice precum DATEADD, DATEDIFF, CONVERT/CAST, după cum vom vedea în capitolul următor.

Complicându-ne și mai zdravăn, dorim să afișăm pentru fiecare factură ce dată va fi peste 1 an, două luni și 25 de zile de la momentul emiterii.

Soluția 1 - PostgreSQL:

```

SELECT NrFact AS Factura, DataFact AS Data_Facturare,
       DataFact + INTERVAL '1 YEAR' +
       INTERVAL '2 MONTH' + INTERVAL '25 DAY'
       AS O_Data_Viitoare
FROM facturi

```

Soluția 2 PostgreSQL:

```
SELECT NrFact AS Factura, DataFact AS Data_Facturare,
       DataFact + INTERVAL '1 YEAR 2 MONTH 25 DAY' AS O_Data_Viitoare
FROM facturi
```

Soluția Oracle:

```
SELECT NrFact AS Factura, DataFact AS Data_Facturare,
       DataFact + INTERVAL '1-2' YEAR TO MONTH +
       25 AS O_Data_Viitoare
FROM facturi
```

Soluția DB2:

```
SELECT NrFact AS Factura, DataFact AS Data_Facturare,
       DataFact + 1 YEAR + 2 MONTHS + 25 DAYS AS O_Data_Viitoare
FROM facturi
```

În ceea ce privește operațiunile de scădere a două date calendaristice, de regulă rezultatul se exprimă în zile. Spre exemplu, dacă ne interesează *numărul de zile scurs între 22 noiembrie 2007 și data emiterii fiecărei facturi* (figura 5.15 conține doar o parte dintre facturi), soluția Oracle/PostgreSQL este una simplă:

```
SELECT NrFact, DATE'2007-11-22', DataFact,
       DATE'2007-11-22' - DataFact AS Zile_Scuse
FROM facturi
```

R	NRFACT	R	DATE'2007-11-22'	R	DATAFACT	R	ZILE_SCURSE
1	1111	2	22-11-2007	3	01-08-2007	4	113
	1112	2	22-11-2007	3	01-08-2007	4	113
	1113	2	22-11-2007	3	01-08-2007	4	113
	1114	2	22-11-2007	3	01-08-2007	4	113
	1115	2	22-11-2007	3	02-08-2007	4	112
	1116	2	22-11-2007	3	02-08-2007	4	112
	1117	2	22-11-2007	3	03-08-2007	4	111
	1118	2	22-11-2007	3	04-08-2007	4	110
	1119	2	22-11-2007	3	07-08-2007	4	107
	1120	2	22-11-2007	3	07-08-2007	4	107
	1121	2	22-11-2007	3	07-08-2007	4	107
	1122	2	22-11-2007	3	07-08-2007	4	107

Figura 5.15. Diferența în zile dintre două date calendaristice

Dacă dorim același rezultat dar exprimat în ani și luni, operațiunea era destul de complicată până la standardul SQL:1999. Introducându-se tipurile de date `INTERVAL`, conversia se face de o manieră simplă, scriind imediat după expresia-scădere tipul intervalului (`YEAR TO MONTH` în cazul Oracle):

```
SELECT NrFact, DATE'2007-11-22', DataFact,
       DATE'2007-11-22' - DataFact AS Zile_Scuse,
       (DATE'2007-11-22' - DataFact) YEAR TO MONTH AS AniLuni_Sc
```

FROM facturi

Rezultatul Oracle (primele linii) este cel din figura 5.16.

NRFACT	DATE'2007-11-22'	DATAFACT	ZILE_SCURSE	ANILUNI_SC
1111	22-11-2007	01-08-2007	113	0-4
1112	22-11-2007	01-08-2007	113	0-4
1113	22-11-2007	01-08-2007	113	0-4
1114	22-11-2007	01-08-2007	113	0-4
1115	22-11-2007	02-08-2007	112	0-4
1116	22-11-2007	02-08-2007	112	0-4
1117	22-11-2007	03-08-2007	111	0-4
1118	22-11-2007	04-08-2007	110	0-4
1119	22-11-2007	07-08-2007	107	0-4
1120	22-11-2007	07-08-2007	107	0-4
1121	22-11-2007	07-08-2007	107	0-4
1122	22-11-2007	07-08-2007	107	0-4

Figura 5.16. Diferența în zile, ani și luni dintre două date calendaristice (Oracle)

Interogarea funcționează în Oracle, nu însă și în PostgreSQL. După cum vor vedea însă în paragraful 6.4, există variante elegante bazate pe funcția AGE. În DB2, diferența dintre două date calendaristice are aparența unui număr (de fapt, tipul este DURATION) care reprezintă anii, lunile și zilele scurse dintre cele două date. Astfel, în urma execuției interogării:

```
SELECT NrFact, '2007-11-22', DataFact,
       '2007-11-22' - DataFact AS Zile_Scurse1
FROM facturi
```

NRFACT	2007-11-22	DATAFACT	ZILE_SCURSE1
1.111	2007-11-22	01.08.2007	321
1.112	2007-11-22	01.08.2007	321
1.113	2007-11-22	01.08.2007	321
1.114	2007-11-22	01.08.2007	321
1.115	2007-11-22	02.08.2007	320
1.116	2007-11-22	02.08.2007	320
1.117	2007-11-22	03.08.2007	319
1.118	2007-11-22	04.08.2007	318
1.119	2007-11-22	07.08.2007	315
1.120	2007-11-22	07.08.2007	315
1.121	2007-11-22	07.08.2007	315
1.122	2007-11-22	07.08.2007	315
2.111	2007-11-22	14.08.2007	308
2.112	2007-11-22	14.08.2007	308
2.113	2007-11-22	14.08.2007	308
2.115	2007-11-22	15.08.2007	307
2.116	2007-11-22	15.08.2007	307
2.117	2007-11-22	16.08.2007	306
2.118	2007-11-22	16.08.2007	306
2.119	2007-11-22	21.08.2007	301

Figura 5.17. Diferența în ani, zile și luni dintre două date calendaristice în DB2

coloana *Zile_Scurse1* (vezi figura 5.17) trebuie „citită” de la dreapta la stânga, știind că primele două cifre reprezintă numărul de zile, următoarele două numărul de luni, iar următoarele două, numărul de ani. Astfel, 321 de pe prima linie înseamnă 3 luni și 21 de zile. Știind acum logica rezultatului, afișarea sub altă formă poate fi realizată prin funcții de extragere a unui șir de caractere.

Despre alte funcții dedicate gestionării datelor calendaristice și șirurilor de caractere vom discuta în capitolul următor.

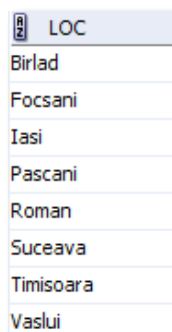
5.4.Ordonări

Una din caracteristicile modelului relațional este că nici ordinea atributelor, nici ordinea liniilor în relații nu influențează conținutului informațional al unei table. În practică, însă, forma de prezentare a rezultatelor interogării este importantă. Spre exemplu, o listă a localităților este cu mult mai de folos decât se prezintă în ordine alfabetică. Ordonarea liniilor în rezultatul unei interogări este posibilă prin clauza **ORDER BY**.

Să se obțină lista localităților în ordine alfabetică.

```
SELECT DISTINCT Loc  
FROM coduri_postale  
ORDER BY Loc
```

Rezultatul se prezintă ca în figura 5.18.



LOC
Birlad
Focsani
Iasi
Pascani
Roman
Suceava
Timisoara
Vaslui

Figura 5.18. Localitățile ordonate alfabetic

Aranjarea se face implicit crescător (ASC). Prin opțiunea DESC, ordinea prezentării se inversează. În plus, se pot specifica mai multe coloane care să servească drept criterii suplimentare de ordonare. La valori egale ale primului atribut, intră în acțiune criteriul de “balotaj” care este al doilea atribut etc.

Să se obțină, în ordinea descrescătoare a indicativului județelor, lista localităților în ordinea crescătoare a denumirii.

```
SELECT Jud, Loc, CodPost  
FROM coduri_postale
```

ORDER BY Jud DESC, Loc ASC

Rezultatul este cel din figura 5.19. Ultimul indicativ de județ (e vorba de ordine alfabetică) din tabela CODURI_POSTALE este VS (pentru Vaslui), deci primele localități afișate vor fi din acest județ; prin urmare, prima linie a rezultatului se referă la municipiul Bîrlad, iar a doua la municipiul Vaslui.

JUD	LOC	CODPOST
VS	Birlad	706400
VS	Vaslui	706510
VS	Vaslui	706500
VN	Focsani	705310
VN	Focsani	705300
TM	Timisoara	701900
SV	Suceava	705800
NT	Roman	705550
IS	Iasi	700515
IS	Iasi	700510
IS	Iasi	700505
IS	Pascani	701150

Figura 5.19. Două criterii de ordonare

Să se afișeze facturile din luna septembrie 2007 în ordinea alfabetică a observațiilor (atributul Obs):

```
SELECT *
FROM facturi
WHERE DataFact >= DATE'2007-09-01' AND DataFact <= DATE'2007-09-30'
ORDER BY Obs12
```

După cum se observă în rezultatul (Oracle) din figura 5.20, la început sunt liniile cu valorile nenule ale Obs, apoi apar toate cele în care valoarea atributului de ordonare este NULL¹³.

¹² În MS SQL Server și DB2 se elimină cele două cuvinte DATE.

¹³ În MS SQL Server primele linii sunt cele în care valorile atributului Obs sunt NULL, în timp ce DB2 ordonează liniile ca în Oracle.

NRFACT	DATAFACT	CODCL	OBS
3116	10-09-2007	1007	Pretul propus initial a fost modificat
3112	01-09-2007	1005	Probleme cu transportul
3118	17-09-2007	1001	(null)
3117	10-09-2007	1001	(null)
3113	02-09-2007	1002	(null)
3111	01-09-2007	1001	(null)
3115	02-09-2007	1001	(null)

Figura 5.20. Ordonarea implicită pentru un atribut ce prezintă valori NULL

Standardul SQL prezintă două clauze, NULLS FIRST și NULLS LAST prin care ne putem exprima preferința în materie de poziționarea valorilor NULL la ordonare. Astfel, dacă în rezultat, preferăm să se înceapă cu valorile NULL, modificăm ușor interogarea de mai sus:

```
SELECT *
FROM facturi
WHERE datafact >= DATE'2007-09-01' AND datafact <= DATE'2007-09-30'
ORDER BY obs NULLS FIRST
```

Cum era de așteptat, ordinea este acum cea din figura 5.21. Interogarea funcționează în Oracle, nu însă în PostgreSQL înainte de versiunea 8.3 și nici în SQL Server și DB2 (toate versiunile), unde avem nevoie de o funcție de conversie a valorilor nule (COALESCE) sau o structura alternativă de tip CASE. Începând cu versiunea 8.3, în PostgreSQL lucrurile s-au remediat, interogarea de mai sus fiind funcțională.

O problemă (probabil) secundară ține de modul în care specificăm un criteriu de ordonare, în condițiile în care criteriul nu e un atribut, ci o expresie (coloană calculată). Ne interesează liniile facturii 1111, ordonate descrescător după valoarea fără TVA. Criteriul de ordonare se poate scrie în DB2, Oracle, PostgreSQL, SQL Server în mai multe moduri:

NRFACT	DATAFACT	CODCL	OBS
3111	01-09-2007	1001	(null)
3115	02-09-2007	1001	(null)
3117	10-09-2007	1001	(null)
3113	02-09-2007	1002	(null)
3118	17-09-2007	1001	(null)
3116	10-09-2007	1007	Pretul propus initial a fost modificat
3112	01-09-2007	1005	Probleme cu transportul

Figura 5.21. Modificarea ordonării pentru un atribut ce prezintă valori NULL

- Titulatura coloanei calculate, stabilită prin clauza AS:

```
SELECT liniifact.*, Cantitate * PretUnit AS ValLinie_FaraTVA
FROM liniifact
WHERE NrFact=1111
```


- ORDER BY ValLinie_FaraTVA DESC**
- Expresia propriu-zisă:
SELECT liniifact.*, Cantitate * PretUnit AS ValLinie_FaraTVA
FROM liniifact
WHERE NrFact=1111
ORDER BY Cantitate * PretUnit DESC
- Numărul de ordine al coloanei: știind că LINIIFACR are cinci atribute, expresia ar fi a șasea coloană:
SELECT liniifact.*, Cantitate * PretUnit AS ValLinie_FaraTVA
FROM liniifact
WHERE NrFact=1111
ORDER BY 6 DESC

5.5.Operatorii BETWEEN, OVERLAPS și IN

Pentru formularea predicatului de selecție, SQL permite utilizarea, pe lângă "clasicii" $>$, \geq , $<$, \leq , $=$, \neq , și a altor operatori, dintre care ne vom opri la: BETWEEN (între, cuprins între), OVERLAPS (se suprapune) și IN (în), la care se vor adăuga LIKE și SIMILAR, discutați în paragraful 5.6, și IS NULL prezentat în paragraful 8.2.

Operatorul BETWEEN este util pentru definirea intervalelor de valori. Ne reîntoarcem în capitolul 2 la exemplul 3 (*Care sunt facturile emise în perioada 2-5 august 2007 ?*) Pe lângă soluția formulată anterior în acest capitol, vom prezenta noi variante PostgreSQL și Oracle.

- Soluție „standard” - în care literalii de tip dată calendaristă sunt specificați prin clauza DATE.
SELECT *
FROM facturi
WHERE DataFact BETWEEN DATE'2007-08-02' AND DATE'2007-08-05'
- Soluție DB2/PostgreSQL/SQL Server - literalii de tip dată se pot scrie direct:
SELECT * FROM facturi
WHERE DataFact BETWEEN '2007-08-02' AND '2007-08-05'
- Soluția Oracle/PostgreSQL - specificare prin funcția TO_DATE.
SELECT * FROM facturi
WHERE DataFact BETWEEN TO_DATE('02/08/2007','DD/MM/YYYY')
AND TO_DATE('05/08/2007','DD/MM/YYYY')

Intervalul precizat începe obligatoriu cu limita minimă. Dacă am schimba interogarea de mai sus inversând cele două valori, varianta:

```
SELECT * FROM facturi
WHERE DataFact BETWEEN DATE'2007-08-05' AND DATE'2007-08-02'
```

nu returnează nimic.

BETWEEN poate fi folosit și pentru atribute de tip șir de caractere. Să se obțină, în ordinea județelor, lista localităților cu indicativul județului cuprins între IS (Iași) și TM (Timiș).

```
SELECT Jud, Loc, CodPost
FROM coduri_postale
WHERE Jud BETWEEN 'IS' AND 'TM'
ORDER BY Jud, Loc
```

Se poate folosi și negația operatorului. O consecință interesantă este că, atunci când intervalul calendaristic este precizat defectuos, interogarea extrage întreg conținutul tabelului, ca în acest exemplu:

```
SELECT *
FROM facturi
WHERE DataFact NOT BETWEEN DATE'2007-08-05' AND DATE'2007-08-02'
```

Pentru lucrul cu intervale calendaristice/temporale, SQL-ul prezintă un operator special – OVERLAPS. Acesta verifică dacă un interval calendaristic se suprapune (intersectează), fie și parțial, cu alt interval ce servește drept argument:

```
SELECT *
FROM facturi
WHERE (DataFact, DataFact + INTERVAL '20 DAYS') OVERLAPS
(DATE'2007-09-15', DATE'2007-10-02')
```

Interogarea extrage facturile pentru care intervalul format din ziua de facturare (DataFact) și următoarele 20 de zile se „intersectează”, măcar pentru o zi, cu intervalul 15 septembrie – 2 octombrie 2007. Rezultatul este cel din figura 5.22. Operatorul OVERLAPS este implementat, deocamdată, numai în PostgreSQL.

nrfact numeric(8,0)	datafact date	codcl numeric(6,0)	obs character varying(50)
3111	2007-09-01	1001	
3112	2007-09-01	1005	Probleme cu transportul
3113	2007-09-02	1002	
3115	2007-09-02	1001	
3116	2007-09-10	1007	Pretul propus initial a fost modificat
3117	2007-09-10	1001	
3118	2007-09-17	1001	

Figura 5.22. Facturile pentru care intervalul calendaristic dintre data emiterii și următoarele 20 de zile se suprapune cu intervalul 15 sept.-2 oct 2007.

Operatorul IN se recomandă în predicatele ce verifică dacă valoarea unui atribut este încadrabilă într-o listă de valori dată. În locul folosirii abundente a

operatorului OR, este mai elegant să se apeleze la serviciile operatorului IN. Format general: **expresie1 IN (expresie2, expresie3, ...)**

Rezultatul evaluării unui predicat ce conține acest operator va fi *adevărat* dacă valoarea expresiei1 este egală cu (cel puțin) una din valorile: expresie2, expresie3,

Care sunt localitățile din județele Iași (IS), Vaslui (VS) și Timiș (TM) ?

- Fără operatorul IN:

```
SELECT DISTINCT Loc, Jud
FROM coduri_postale
WHERE Jud = 'IS' OR Jud = 'VS' OR Jud = 'TM'
ORDER BY Jud, Loc
```
- Cu operatorul IN:

```
SELECT DISTINCT Loc, Jud
FROM coduri_postale
WHERE Jud IN ('IS', 'VS', 'TM')
ORDER BY Jud, Loc
```

Care sunt facturile întocmite pe 1, 3 și 7 august 2007 ?

```
SELECT *
FROM facturi
WHERE DataFact IN (DATE'2007-08-01', DATE'2007-08-03', DATE'2007-08-07')
```

Despre câteva probleme legate de valorile NULL o să avem ocazia să discutăm în capitolul 8. Deocamdată, să introducem o valoare NULL în lista-argument a operatorului. Rezultatele interogării:

```
SELECT *
FROM facturi
WHERE DataFact IN (DATE'2007-08-01', DATE'2007-08-03',
DATE'2007-08-07', NULL)
```

nu diferă cu nimic de cel al SELECT-ului fără NULL: (SELECT * FROM facturi WHERE DataFact IN (DATE'2007-08-01', DATE'2007-08-03', DATE'2007-08-07')). În schimb, dacă folosim NOT IN în combinație cu un NULL în listă, lucrurile stau oarecum diferit. Fraza:

```
SELECT *
FROM facturi
WHERE DataFact NOT IN (DATE'2007-08-01',DATE'2007-08-03',DATE'2007-08-07')
```

extrage toate facturile din *alte zile decât* 1, 3 și 7 august 2007, în timp ce rezultatul interogării:

```
SELECT *
FROM facturi
WHERE DataFact NOT IN (DATE'2007-08-01', DATE'2007-08-03',
```

DATE'2007-08-07', NULL)

nu conține nici o linie.

Includerea valorilor NULL în listă este controlată strict când construim și lansăm interogările direct, așa cum procedăm în această lucrare. Există însă aplicații în care lista de valori este alcătuită din variabile (preluate din formulare), iar în anumite circumstanțe, datorită datelor introduse de utilizator, este posibil ca cel puțin una dintre variabile să se afle în situația de nulitate.

DB-ul este reticent la introducerea valorilor NULL într-o listă-argument a operatorului IN, afișând un mesaj de eroare ca în figura 5.23.

```
----- Commands Entered -----
SELECT *
FROM facturi
WHERE DataFact IN ('2007-08-01', '2007-08-03', '2007-08-07', NULL);

-----
SELECT * FROM facturi WHERE DataFact IN ('2007-08-01', '2007-08-03', '2007-08-07', NULL)
SQL0206N "NULL" is not valid in the context where it is used.  SQLSTATE=42703
```

Figura 5.23. Opoziția (neconstructivă) DB-ului la inserarea unui NULL într-o listă

5.6.Comparații inexacte. LIKE și SIMILAR

Deseori suntem în postura, oarecum ingrată, de a nu ști cu exactitate cum se numește un client sau un produs, sau, pur și simplu, unei persoane îi cunoaștem numai unul dintre prenume etc. Sunt situații pentru rezolvarea cărora a fost gândit operatorul LIKE. Operatorul LIKE permite compararea unui atribut (expresii) cu un literal utilizând o “mască” (șablon) construită cu ajutorul specificatorilor multipli % și _. Simbolurile *procent* și *liniuță_de_jos* (*underscore*, diferită de cratimă sau *liniuță-de-unire*) sunt denumite și jokeri. Procentul substituie un șir de lungime variabilă, 0-n caractere, în timp ce linia un singur caracter.

Care din firmele-client sunt societăți cu răspundere limitată (SRL-uri) ?

Întrucât în tabela CLIENTI denumirea fiecărui client se termină cu forma sa de societate (SA, SRL etc.), se poate redacta consultarea:

```
SELECT *
FROM clienti
WHERE DenCl LIKE '%SRL'
```

Rezultatul obținut prin execuția interogării în (Microsoft) SQL Server Management Studio se prezintă ca în figura 5.24.

codcl	dencl	codfiscal	adresa	codpost	telefon
1001	Client 1 SRL	R1001	Tranzitiei, 13 bis	700505	NULL
1003	Client 3 SRL	R1003	Prosperitatii, 22	706500	0235222222
1005	Client 5 SRL	R1005	NULL	701900	0256111111
1007	Client 7 SRL	R1007	Victoria Capitalismului, 2	701900	0256121212

Figura 5.24. Clienții - SRLuri

Care dintre clienți au numele (fără forma de societate și spațiul dinainte) din 8 caractere și sunt societăți pe acțiuni ?

```
SELECT *
FROM clienti
WHERE DenCI LIKE '_____SA'
```

Cele șapte liniuțe-„underscore” (nu se observă, dar sunt șapte), plus spațiul care le urmează, nu au efect vizibil/impresionant pentru baza noastră de date, deoarece în capitolul 3, leneș fiind, am denumit toți clienții de o manieră simplistă - Client 1 SRL, Client 2 SA etc. Dacă am avea mai mulți clienți (sau măcar un “Client 10 SA”), atunci interogarea ar avea cu totul alt farmec.

Ce persoane au numele ce conține litera S pe a treia poziție ?

```
SELECT *
FROM persoane
WHERE Nume LIKE '__s%'
```

Rezultatul este vizualizat în figura 5.25. Atenție, dacă există persoane al căror nume are litera S majusculă pe a treia poziție, acestea nu sunt extrase în rezultat !

CNP	NUME	PRENUME	ADRESA	SEX	CODPOST	TELACASA	TELBIROU	TELMOBIL	EMAIL
CNP2	Vasile	Ion	(null)	B	700505	234567	876543	094222223	Ion@a.ro
CNP6	Vasc	Simona	M.Eminescu, 13 F	F	701150	(null)	432109	094222227	(null)

Figura 5.25. Persoane cu litera “s” pe a treia poziție a numelui

În asemenea situații, soluția de mai jos este ceva mai sigură:

```
SELECT *
FROM persoane
WHERE Nume LIKE '__s%' OR Nume LIKE '__S%'
```

În numele căror persoane apare, măcar o dată, litera S (indiferent de poziție/poziții) ?

De data aceasta, trebuie să folosim specificatorul % pe care îl plasăm și înainte și după litera S. Rezultatul este cel din figura 5.26.

```
SELECT *
FROM persoane
WHERE Nume LIKE '%s%' OR Nume LIKE '%S%'
```

CNP	NUME	PRENUME	ADRESA	SEX	CODPOST	TELACASA	TELBIROU	TELMOBIL	EMAIL
CNP2	Vasile	Ion	(null)	B	700505	234567	876543	094222223	Ion@a.ro
CNP6	Vasc	Simona	M.Eminescu, 13 F	F	701150	(null)	432109	094222227	(null)
CNP8	Bogacs	Ildiko	I.V.Viteazu, 67 F	F	705550	890123	210987	094222229	(null)

Figura 5.26. Persoane al căror nume conține litera S

Care sunt persoanele ce trebuie felicitate de Sfântul Ion ?

Firește am fi tentați să redactăm varianta:

```

SELECT *
FROM persoane
WHERE UPPER(Prenume) LIKE '%ION%'

```

Funcția UPPER face conversia valorilor atributului Prenume în majuscule (vezi capitolul următor). Rezultatul este însă cel din figura 5.27.

cnp	nume	prenume	adresa	sex	codpost	telacasa	telbirou	telmobil	email
CNP2	Vasile	Ion	NULL	B	700505	234567	876543	094222223	lon@a.ro
CNP4	Lazar	Caraion	M.Eminescu, 42	B	706500	456789	NULL	094222225	NULL
CNP5	Iurea	Simion	I.Creanga, 44 bis	B	706500	567890	543210	NULL	NULL

Figura 5.27. Tentativă ratată de a extrage "Sfinții Ioni"

Interogarea nu și-a atins ținta, deoarece, pe de o parte, nu au fost extrase persoanele cu prenume ca Ioan, Ioana, Ioanid, iar, pe de altă parte, au fost eronat extrase și persoanele cu prenumele Caraion și Simion. Prin urmare, cele trei litere - ION trebuie să fie plasate la începutul prenumelui. La acest șablon se adaugă și IOAN. Bun, dar dacă pe omul nostru îl cheamă Marius Ion (are două prenume) ? Cele două șabloane trebuie să se găsească la începutul fiecărui cuvânt din atributul Prenume. Iar dacă avem în vedere că uneori cele două prenume se despart prin cratimă (Marius-Ion), rezultă o mândrețe de interogare (răspunsul este prezentat în figura 5.28) care funcționează în toate cele patru servere BD.:

```

SELECT *
FROM persoane
WHERE UPPER(Prenume) LIKE 'ION%' OR UPPER(Prenume) LIKE 'IOAN%'
      OR UPPER(Prenume) LIKE '% ION%' OR
      UPPER(Prenume) LIKE '% IOAN%' OR
      UPPER(Prenume) LIKE '%-ION%' OR
      UPPER(Prenume) LIKE '%-IOAN%'

```

cnp	nume	prenume	adresa	sex	codpost	telacasa	telbirou	telmobil	email
CNP2	Vasile	Ion	NULL	B	700505	234567	876543	094222223	lon@a.ro
CNP3	Popovici	Ioana	V.Micle, Bl.I, Sc.B, Ap.2	F	701150	345678	NULL	094222224	NULL
CNP7	Popa	Ioanid	I.Ion, Bl.H2, Sc.C, Ap.45	B	701900	789012	321098	NULL	NULL

Figura 5.28. Persoanele ce trebuie felicitate de Sf. Ion

SQL Server poate folosi în șablonul clauzei LIKE, pe lângă % și _ și paranteze drepte și caracterul ^. Ne interesează toate persoanele ale căror prenume începe cu litera I sau V. În afara variantei standard:

```

SELECT * FROM persoane
WHERE UPPER(Prenume) LIKE 'I%' OR UPPER(Prenume) LIKE 'V%'

```

SQL Server acceptă și varianta ceva mai scurtă:

```

SELECT *
FROM persoane
WHERE UPPER(Prenume) LIKE '[I,V]%'

```

Iar dacă ne-ar interesa toate persoanele, cu excepția celor ale căror nume încep fie cu litera I, fie cu litera V, varianta generală ar fi:

```
SELECT *
FROM persoane
WHERE UPPER(Prenume) NOT LIKE 'I%' AND
      UPPER(Prenume) NOT LIKE 'V%'
```

în timp ce SQL Server acceptă și sintaxa:

```
SELECT * FROM persoane WHERE UPPER(Prenume) LIKE '[^I,V]%'
```

Tot aici putem nota operatorul ILIKE din PostgreSQL care are aceeași funcționalitate ca LIKE, doar că nu se mai ține cont dacă literele sunt majuscule sau minuscule și, astfel, nu avem nevoie de funcția UPPER. Interogarea „Sfinților Ion” se poate rescrie ceva mai simplu:

```
SELECT *
FROM persoane
WHERE Prenume ILIKE 'ION%' OR Prenume ILIKE 'IOAN%' OR
      Prenume ILIKE '% ION%' OR Prenume ILIKE '% IOAN%'
      OR Prenume ILIKE '%-ION%' OR Prenume ILIKE '%-IOAN%'
```

Deși rare, există cazuri când printre caracterele căutate în valorile unui atribut șir de caractere se găsește chiar unul dintre cele două șabloane, _ sau %. Dacă, spre exemplu, interesează toți clienții care conțin simbolul % în numele lor (s-ar putea ca, la un moment dat, să avem în bază clienți de genul “Procentul vesel % SRL”). Soluția este:

```
SELECT *
FROM clienti
WHERE DenCI LIKE '%!%%' ESCAPE '!'
```

Datorită primului și ultimului simbol procent, poziția caracterului căutat (în cazul nostru, chiar %) nu prezintă importanță: poate fi prima, ultima sau oricare între prima și ultima. Rezultatul corect este obținut grație unui caracter declarat prin clauza ESCAPE. Acesta este, în interogarea noastră, semnul mirării (!), dar putea fi oricare altul. Prin clauza ESCAPE s-a indicat SQL-ului că procentul ce urmează simbolului ! nu este joker, ci are regim de caracter oarecare, ce trebuie căutat în tabelă ca atare.

Standardul SQL:1999 a introdus un mecanism mai elegant de căutare preluat din sistemul de operare UNIX (Posix), mecanism bazat pe operatorul SIMILAR și implementat doar în PostgreSQL. Cele două caractere folosite în construirea „măștilor” sunt tot procent și underscore. Modul de contruire a expresiei de căutare este însă mult mai generos, principalele caractere disponibile fiind:

- | - pentru structuri alterative;
- * - pentru repetarea șirului (de caractere) precedent de zero sau mai multe ori;
- + - pentru repetarea șirului precedent de unu sau mai multe ori;

- () - pentru gruparea mai multor șiruri de caractere într-o unitate logică;
- [] - pentru indicarea unei clase de caractere.

Să începem cu câteva chestiuni simple. Aflarea *firmelor-client ce sunt societăți cu răspundere limitată* se poate realiza ca la începutul paragrafului, înlocuind LIKE cu SIMILAR TO:

```
SELECT *
FROM clienti
WHERE DenCl SIMILAR TO '%SRL'
```

Expresia-șablon poate folosi paranteze:

```
SELECT * FROM clienti WHERE DenCl SIMILAR TO '(%SRL)'
```

sau

```
SELECT * FROM clienti WHERE DenCl SIMILAR TO '%(SRL)'
```

Prin folosirea caracterului , | ' (bară verticală) evităm folosirea operatorului OR. Reluând ultima soluție care ar extrage candidații la felicitări de Sf. Ion, prin folosirea noului caracter interogarea se simplifică vizibil, cel puțin ca lungime:

```
SELECT *
FROM persoane
WHERE UPPER(Prenume) SIMILAR TO
'ION%|IOAN%| % ION%| % IOAN| %-ION%| %-IOAN%'
```

Deși nu prea are rost, putem folosi și parantezele:

```
SELECT * FROM persoane
WHERE UPPER(Prenume) SIMILAR TO
'(ION%)|(IOAN%)| (% ION%)| (% IOAN)| %-ION%| %-IOAN%'
```

Continuăm cu o problemă ceva mai exotică. Vrem să aflăm *clienții care au sediul la un număr care conține numai cifrele 2, 3 sau 5*. Știind că numărul urmează unui spațiu de după numele străzii, construim șablonul după cum urmează:

```
SELECT *
FROM clienti
WHERE Adresa SIMILAR TO '% [235]+'
```

Ne interesează doar cifrele 2, 3 și 5, deci excludem combinațiile acestora cu alte cifre. Lucrul acesta se indică prin plasarea lor între paranteze drepte. Semnul plus din finalul șablonului indică repetarea de oricâte ori (de 1-n ori) sau, altfel spus, se vor selecta numai liniile în care cifra 2, 3 sau 5 apare măcar o dată în combinație cu ea însăși sau celelate două. Rezultatul interogării PostgreSQL conține trei linii, ca în figura 5.29.

codcl numeric(6,0)	dendcl character varying(40)	codfiscal character(9)	adresa character varying(40)	codpost character(6)	telefon character varying(12)
1003	Client 3 SRL	R1003	Prosperitatii, 22	706500	035-222222
1006	Client 6 SA	R1006	Pacientei, 33	705550	
1007	Client 7 SRL	R1007	Victoria Capitalismului, 2	701900	056-121212

Figura 5.29. Clienții ce au numărul de la adresa alcătuit numai din una sau mai multe dintre cifrele 2, 3 sau 5

Dacă nu am fi folosit în finalul șablonului semnul +, adică:

```
SELECT *
FROM clienti
WHERE Adresa SIMILAR TO '% [235]'
```

rezultatul ar fi conținut doar un rând (vezi figura 5.30), întrucât doar Clientul 7 are numărul de la adresă alcătuit dintr-o singură cifră (2).

codcl numeric(6,0)	dendcl character varying(40)	codfiscal character(9)	adresa character varying(40)	codpost character(6)	telefon character varying(15)
1007	Client 7 SRL	R1007	Victoria Capitalismului, 2	701900	056-121212

Figura 5.30. Clienții ce au numărul de la adresa alcătuit numai din una dintre cifrele 2, 3 sau 5

Semnul + indică repetarea de 1-*n* ori a cifrei/cifrelor din șablon, în timp ce asteriscul (*) indică repetarea de 0-*n* ori. Posix („sortimentul” de Unix din care provine mecanismul SIMILAR TO) are opțiuni pentru a indica numărul exact de repetări ale caracterului specificat, însă nici standardele SQL, și nici PostgreSQL-ul nu au încă această facilitățe.

O problemă a soluțiilor de mai sus iese la iveală dacă ne propunem să extragem clienții cu adrese la numere alcătuite numai din cifrele 1 și 3. Interogarea:

```
SELECT *
FROM clienti
WHERE Adresa SIMILAR TO '(% [13]+)'
```

extrage doar clientul 6 (vezi figura 5.31), deși clientul 1 are sediul pe strada Tranzitei, nr. 13 bis.

codcl numeric(6,0)	dendcl character varying(40)	codfiscal character(9)	adresa character varying(40)	codpost character(6)	telefon character varying(15)
1006	Client 6 SA	R1006	Pacientei, 33	705550	

Figura 5.31. Un singur client cu numărul de la adresă alcătuit numai din una sau din ambele cifre 1 și 3 (deși sunt doi)

Necazul decurge din faptul că numărul 13 se continuă cu „bis”, așa că avem nevoie de modificarea șablonului:

```
SELECT *
FROM clienti
WHERE Adresa SIMILAR TO '(% [13]+)|(% [13]+ bis)'
```

De data aceasta rezultatul este corect, după cum se poate observa în figura 5.32. Am folosit secvența alternativă (operatorul „|”).

codcl numeric(6,0)	dendcl character varying(40)	codfiscal character(9)	adresa character varying(40)	codpost character(6)	telefon character varying(15)
1001	Client 1 SRL	R1001	Tranzitei, 13 bis	700505	
1006	Client 6 SA	R1006	Pacientei, 33	705550	

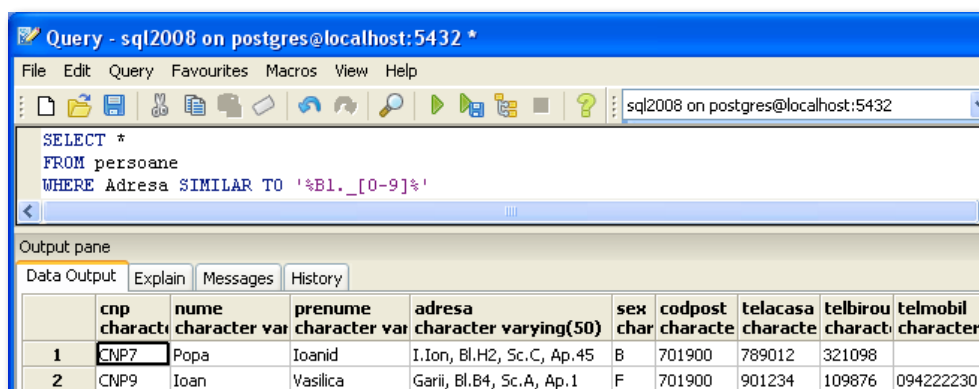
Figura 5.32. Cei doi clienți cu numărul de la adresă alcătuit numai din una sau din ambele cifre 1 și 3

Care sunt persoanele care stau la bloc, blocuri ale căror nume este alcătuit dintr-o literă și o cifră (ex. H2, B5, A9) ?

Pentru litera din denumirea blocului se poate folosi jokerul *underscore*. În schimb, substituirea unei cifre se poate face incluzând între paranteze drepte intervalul 0-9, ca în interogarea următoare al cărei rezultat este afișat în figura 5.33.

```
SELECT *
FROM persoane
WHERE Adresa SIMILAR TO '%B1._[0-9]%'
```

Din păcate, celelalte trei servere nu au implementat SIMILAR TO, ci doar câteva funcții cu acțiune similară.



Query - sql2008 on postgres@localhost:5432 *

```
SELECT *
FROM persoane
WHERE Adresa SIMILAR TO '%B1._[0-9]%'
```

Output pane

Data Output Explain Messages History

	cnp	nume	prenume	adresa	sex	codpost	telacasa	telbirou	telmobil
	character	character var	character var	character varying(50)	char	character	character	character	character
1	CNP7	Popa	Ioanid	I.Ion, Bl.H2, Sc.C, Ap.45	B	701900	789012	321098	
2	CNP9	Ioan	Vasilica	Garii, Bl.B4, Sc.A, Ap.1	F	701900	901234	109876	094222230

Figura 5.33. Predicatul SIMILAR TO pentru extragerea în PostgreSQL a domiciliilor în blocuri ale căror nume cuprinde o literă și o cifră

5.7. Joncțiuni interne

SQL prezintă clauze sau operatori speciali pentru aproape toate tipurile de joncțiune discutate în capitolul 4. O joncțiune internă este o combinație de produs cartezian și selecție. În consecință, pentru theta-joncționarea relațiilor R1 și R2 din exemplu 10 al algebrei relaționale (paragraful 4.4.4, figura 4.17) se poate scrie:

```
SELECT *
FROM r1, r2
WHERE r1.A >= r2.E
```

iar pentru echi-joncționarea din exemplul 11 (figura 4.18):

```
SELECT *
FROM r1, r2
WHERE r1.A = r2.E
```

Toate cele patru servere au implementată opțiunea propusă de SQL-92, INNER JOIN (joncțiune internă). Astfel, cele două soluții de mai sus pot fi rescrise după cum urmează:

```
SELECT *  
FROM r1 INNER JOIN r2 ON r1.A >= r2.E
```

respectiv:

```
SELECT *  
FROM r1 INNER JOIN r2 ON r1.A >= r2.E
```

Reluăm, pentru comparație, exemple din algebra relațională.

Exemplul 13 (paragraful 4.4.4) - *Să se obțină, pentru fiecare localitate, codurile poștale, denumirea, indicativul și denumirea județului și regiunea din care face parte*

Varianta 1 (SQL-89):

```
SELECT CodPost, Loc, coduri_postale.Jud, Judet, Regiune  
FROM coduri_postale, judete  
WHERE coduri_postale.Jud = judete.Jud
```

Varianta 2 (SQL-92):

```
SELECT CodPost, Loc, coduri_postale.Jud, Judet, Regiune  
FROM coduri_postale INNER JOIN judete  
ON coduri_postale.Jud = judete.Jud
```

Numai atributul Jud a fost prefixat de numele tabelului din care provine. Operațiunea este obligatorie atunci când câmpul există în două sau mai multe din tabelele enumerate în clauza FROM. Astfel, dacă nu prefixăm atributul Jud în clauza SELECT vom obține un mesaj de genul *Coloană definită ambiguu / Column ambiguously defined* (în funcție de limba specificată la instalare) – vezi figura 5.34. O bilă albă pentru Oracle SQL Developer care afișează mesajul și în română și în engleză.

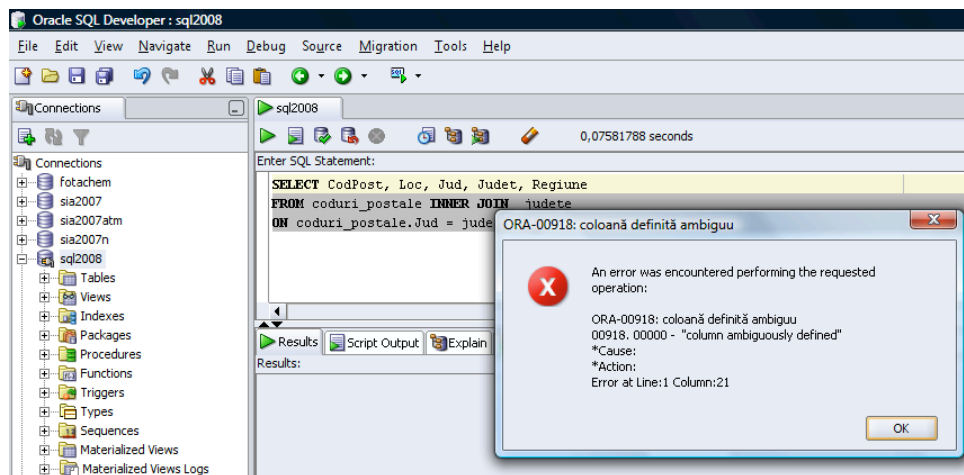


Figura 5.34. Eroare (de ambiguitate) la ne-specificarea tabelii pentru un atribut ce apare în două sau mai multe relații din clauza INNER JOIN

Exemplul 14 (paragraful 4.4.4 – figura 4.20) - *Care sunt codurile poștale ale localităților din Banat ?*

Varianta SQL-1:

```
SELECT CodPost, Loc, coduri_postale.Jud, Judet, Regiune
FROM coduri_postale, judete
WHERE coduri_postale.Jud = judete.Jud AND Regiune= 'Banat'
```

În clauza WHERE, predicatului de selecție pentru realizarea joncțiunii i s-a adăugat secvența de test a regiunii.

Varianta SQL-2:

```
SELECT CodPost, Loc, coduri_postale.Jud, Judet, Regiune
FROM coduri_postale INNER JOIN judete ON coduri_postale.Jud = judete.Jud
WHERE Regiune='Banat'
```

Avantajul celei de-a doua variante ține de separarea condiției ce ține de regiune de condiția legată de joncțiunea propriu-zisă.

Exemplul 15 (paragraful 4.4.4) - *În ce zile s-a vândut produsul cu denumirea "Produs 1" ?*

```
SELECT DISTINCT DataFact
FROM produse
INNER JOIN liniifact ON produse.CodPr = liniifact.CodPr
INNER JOIN facturi ON liniifact.NrFact = facturi.NrFact
WHERE DenPr = 'Produs 1'
```

De notat folosirea clauzei DISTINCT pentru eliminarea eventualelor dubluri.

Exemplul 16 (paragraful 4.4.4) - *În ce județe s-a vândut produsul cu denumirea "Produs 1" în perioada 3-5 august 2007 ?*

```
SELECT DISTINCT Judet
FROM produse
INNER JOIN liniifact ON produse.CodPr = liniifact.CodPr
INNER JOIN facturi ON liniifact.NrFact = facturi.NrFact
INNER JOIN clienti ON facturi.CodCl = clienti.CodCl
INNER JOIN coduri_postale
ON clienti.CodPost=coduri_postale.CodPost
INNER JOIN judete ON coduri_postale.Jud = judete.Jud
WHERE DenPr = 'Produs 1' AND DataFact BETWEEN
DATE'2007-08-03' AND DATE'2007-08-05'
```

Și în acest exemplu este recomandată folosirea clauzei DISTINCT¹⁴.

Exemplul 17 (paragraful 4.4.4) - *În ce zile s-au vândut și produsul cu denumirea "Produs 1" și cel cu denumirea "Produs 2" ?*

```
SELECT DISTINCT DataFact
FROM produse
    INNER JOIN liniifact ON produse.CodPr = liniifact.CodPr
    INNER JOIN facturi ON liniifact.NrFact = facturi.NrFact
WHERE DenPr = 'Produs 1'
INTERSECT
SELECT DISTINCT DataFact
FROM produse
    INNER JOIN liniifact ON produse.CodPr = liniifact.CodPr
    INNER JOIN facturi ON liniifact.NrFact = facturi.NrFact
WHERE DenPr = 'Produs 2'
```

Interesant este că putem include predicatul de selecție chiar în clauza de joncțiune:

```
SELECT DISTINCT DataFact
FROM produse
    INNER JOIN liniifact
        ON produse.CodPr = liniifact.CodPr AND DenPr = 'Produs 1'
    INNER JOIN facturi ON liniifact.NrFact = facturi.NrFact
INTERSECT
SELECT DISTINCT DataFact
FROM produse INNER JOIN liniifact
    ON produse.CodPr = liniifact.CodPr AND DenPr = 'Produs 2'
INNER JOIN facturi ON liniifact.NrFact = facturi.NrFact
```

Ar mai fi o soluție bazată pe joncțiunea tabeli cu ea-însăși, lucru pe care îl vom discuta nu peste mult timp în acest paragraf.

Facem ce facem și ne legăm de lucruri minore. În cazul în care joncționăm două tabele, iar în clauza SELECT folosim asteriscul (*) pentru a semnaliza includerea tuturor atributelor în rezultat, ce nume vor primi attributele care sunt prezente în două sau mai multe dintre tabelele din clauza FROM ? Regula este... că nu e nicio regulă. Execuția interogării:

```
SELECT *
```

¹⁴ Nu uitați să eliminați cele două cuvintele DATE pentru ca interogarea să funcționeze în DB2 și SQL Server.

FROM judete INNER JOIN coduri_postale ON judete.Jud=coduri_postale.Jud

În Oracle SQL Developer duce la o titulatură a coloanelor ca în figura 5.35. Fiind grijuliu în unicitatea denumirii fiecărui atribut, SQL Developer atribuie „celui de-al doilea Jud” titulatura JUD_1, iar DB2-ul titulatura JUD1.

JUD	JUDET	REGIUNE	CODPOST	LOC	JUD_1
IS	Iasi	Moldova	700505	Iasi	IS
IS	Iasi	Moldova	700510	Iasi	IS
IS	Iasi	Moldova	701150	Pascani	IS
IS	Iasi	Moldova	700515	Iasi	IS
NT	Neamt	Moldova	705550	Roman	NT
SV	Suceava	Moldova	705800	Suceava	SV

Figura 5.35. Selectarea tuturor atributelor din două tabele jonctionate în Oracle SQL Developer

În PostgreSQL pgAdmin, nu e nici o preocupare pentru unicitatea denumirii coloanelor în rezultat, așa încât toate coloanele care se repetă au același nume – vezi figura 5.36. Similar stau lucrurile și în SQL Server Management Studio. Am uitat să vă spun că în DB2 și SQL Server în loc de INNER JOIN se poate folosi, simplu, JOIN.

Pentru moment, problema denumirii coloanelor este una fără semnificații majore, însă la definirea ad-hoc, în clauza FROM a unei tabele pe baza unei sub-consultări, denumirile coloanelor trebuie gestionate mai cu grijă.

jud character(2)	judet character var	regiune character var	codpost character(6)	loc character var	jud character(2)
IS	Iasi	Moldova	700505	Iasi	IS
IS	Iasi	Moldova	700510	Iasi	IS
IS	Iasi	Moldova	700515	Iasi	IS
IS	Iasi	Moldova	701150	Pascani	IS
VS	Vaslui	Moldova	706500	Vaslui	VS

Figura 5.36. Selectarea tuturor atributelor din două tabele jonctionate în PostgreSQL pgAdmin

În prima ediție a cărții am evitat cu grijă atât exemple de theta-joncțiune, cât și clauza NATURAL JOIN dedicată, cum era de așteptat, joncțiunii interne naturale. În privință joncțiunii naturale trebuie restabilit adevărul istoric, și anume că în SQL există operatorul NATURAL JOIN pe care l-am descris în paragraful 4.4.4 (exemplul 12, figura 4.19). Avantajul acestui operator ține atât de lungimea interogării (automat joncționarea se realizează după toate atributele comune tabelor din clauza FROM), cât și de păstrarea în rezultat doar a unei singure apariții pentru fiecare atribut comun (de joncționare). Revenim la interogarea din figura 5.35. Folosind operatorul NATURAL JOIN, implementat în DB2, Oracle, PostgreSQL (nu, însă, și în MS SQL Server):

SELECT *

FROM judete NATURAL JOIN coduri_postale

obținem un rezultat ca în figura 5.37.

JUD	JUDET	REGIUNE	CODPOST	LOC
IS	Iasi	Moldova	700505	Iasi
IS	Iasi	Moldova	700510	Iasi
IS	Iasi	Moldova	701150	Pascani
IS	Iasi	Moldova	700515	Iasi
NT	Neamt	Moldova	705550	Roman
SV	Suceava	Moldova	705800	Suceava
TM	Timis	Banat	701900	Timisoara
VN	Vrancea	Moldova	705310	Focsani
VN	Vrancea	Moldova	705300	Focsani
VS	Vaslui	Moldova	706400	Birlad
VS	Vaslui	Moldova	706500	Vaslui
VS	Vaslui	Moldova	706510	Vaslui

Figura 5.37. O joncțiune internă (care funcționează) în Oracle

Exemplul 17 (paragraful 4.4.4) - *Ce clienți au cumpărat și "Produs 2" și "Produs 3", dar nu au cumpărat "Produs 5" ?*

Cu noul operator pentru joncțiune naturală, efortul de scriere al interogării se diminuează în oarecare măsură:

```

SELECT DISTINCT DenCl
FROM produse NATURAL JOIN liniifact NATURAL JOIN facturi
      NATURAL JOIN clienti
WHERE DenPr = 'Produs 2'
INTERSECT
SELECT DISTINCT DenCl
FROM produse NATURAL JOIN liniifact NATURAL JOIN facturi
      NATURAL JOIN clienti
WHERE DenPr = 'Produs 3'
EXCEPT
SELECT DISTINCT DenCl
FROM produse NATURAL JOIN liniifact NATURAL JOIN facturi
      NATURAL JOIN clienti
WHERE DenPr = 'Produs 5'

```

Dacă e să dăm crezare figurii 5.38 interogarea merge fără cusur. Așa stau lucrurile în PostgreSQL.



```

dend
character varying(30)
Client 5 SRL

```

Figura 5.38. O joncțiune internă (care funcționează) în Oracle

Executată în Oracle SQL Developer, și schimbând EXCEPT cu MINUS, aceeași interogare nu obține nimic – vezi figura 5.39.

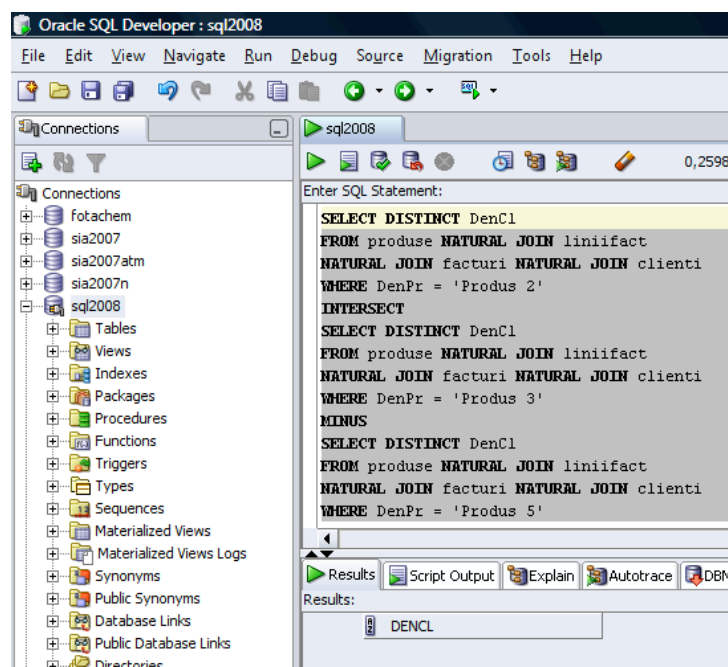


Figura 5.39. O joncțiune internă care nu funcționează în Oracle

Gândindu-mă la celebrele X-Files sau alte ciudățenii, am încercat alte variante de joncțiune naturală în Oracle. Dintre rezultatele tulburătoare obținute, vă supun atenției două interogări plasate față în față care ar trebui, teoretic, să obțină același număr de linii:

<pre> SELECT * FROM produse NATURAL JOIN liniifact NATURAL JOIN facturi NATURAL JOIN clienti WHERE DenPr = 'Produs 1' AND datafact BETWEEN DATE'2007-08-01' AND DATE'2007-08-03' </pre>	<pre> SELECT dencl FROM produse NATURAL JOIN liniifact NATURAL JOIN facturi NATURAL JOIN clienti WHERE DenPr = 'Produs 1' AND datafact BETWEEN DATE'2007-08-01' AND DATE'2007-08-03' </pre>
---	---

Ei bine, nici vorbă ! Interogarea din stânga (SELECT *) extrage două linii în rezultat, în timp ce a doua (SELECT dencl) extrage nu mai puțin de 441 de linii !!! Nu am avut curiozitatea să văd ce funcție recursivă au putut aplica cei de la Oracle ca să obțină așa mândrețe de rezultat. Am inclus însă în figura 5.40 primele 8,5 coloane din rezultatul primei interogări (în stânga) și primele 4,5 linii din rezultatul celei de-a doua.

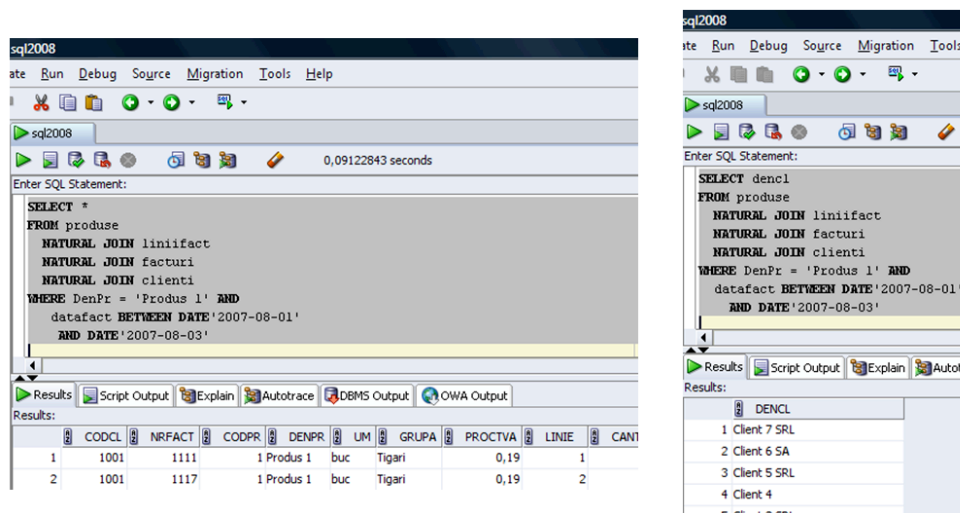


Figura 5.40. Surprise, surprize... în Oracle

Pentru exemplificarea theta-joncțiunii revenim la exemplul 20 din finalul paragrafului 4.4.4. Echivalent soluției propuse în algebra relațională, se poate redacta următoarea interogare SQL:

```
SELECT dataora_examinare, numepacient, numedocitor
FROM triaj INNER JOIN garzi ON dataora_examinare
BETWEEN inceput_garda AND sfirsit_garda
INNER JOIN doctori ON garzi.iddoctor = doctori.iddoctor
INNER JOIN pacienti ON triaj.idpacient = pacienti.idpacient
ORDER BY 1
```

Rezultatul este cel din figura 5.41.

DATAORA_EXAMINARE	NUMEPACIENT	NUMEDOCTOR
Jan 3, 2008 7:18:00 AM 000000	Stroe Mihaela	Georgescu Mircea
Jan 3, 2008 8:45:00 AM 000000	Buzatu Corneliu	Georgescu Mircea
Jan 3, 2008 12:45:00 PM 000000	Spineanu Marius	Georgescu Mircea
Jan 3, 2008 8:45:00 PM 000000	Bagdasar Adela	Zahir Tudorel
Jan 4, 2008 1:28:00 AM 000000	Ionascu Ionelia	Zahir Tudorel
Jan 4, 2008 10:45:00 AM 000000	Spineanu Marius	Bostan Vasile
Jan 4, 2008 11:20:00 AM 000000	Stroescu Mihaela Oana	Bostan Vasile
Jan 4, 2008 10:45:00 PM 000000	Cazan Ana Maria	Popescu Ionela
Jan 5, 2008 6:18:00 AM 000000	Ionascu Ionelia	Georgescu Mircea

Figura 5.41. Rezultatul interogării în care este implicată theta-joncțiunea tabelelor TRIAJ și GARZI

Dacă despre joncțiunile externe ne vom ocupa abia în capitolul 8, despre semijoncțiune lămurim lucrurile pe loc, dacă este, într-adevăr, ceva de lămurit. Pentru exemplificare, reluăm exmplul 22 din paragraful 4.4.6 - *Care sunt codurile poștale (plus denumirea localității și indicativul județului) în care există măcar un client?*

Soluția SQL este mai mult decât banală, întrucât presupune prefixarea asteriscului, în clauza SELECT, cu numele tabelului din care ne interesează valorile – CODURI_POSTALE:

```
SELECT coduri_postale.*
FROM coduri_postale INNER JOIN clienti
ON coduri_postale.CodPost = clienti.CodPost
```

Dacă tabela CODURI_POSTALE are 12 linii, rezultatul din figura 5.42 are doar 7.

CODPOST	LOC	JUD
700505	Iasi	IS
700505	Iasi	IS
701150	Pascani	IS
701900	Timisoara	TM
701900	Timisoara	TM
705550	Roman	NT
706500	Vaslui	VS

Figura 5.42. Semi-joncțiunea dintre CODURI_POSTALE și CLIENTI

5.8. Sinonime locale și joncțiunea unei tabeli cu ea însăși

Lucrul cu nume lungi de tabele are marele avantaj ale lejerității la „citire” și înțelegerii rapide a logicii de derulare a interogării. În schimb, suficienți informaticieni nu agreează risipa de caractere (și, implicit, de timp și nervi) presupuse de redactările „logoreice”. Ambele părți au dreptate în oarecare măsură (sesizați sindromul rabinului !). Astfel încât, în frazele SELECT, tabelilor li se pot asocia sinonime sau aliasuri mai scurte:

```
SELECT *
FROM produse P
INNER JOIN liniifact LF ON P.CodPr = LF.CodPr
INNER JOIN facturi F ON LF.NrFact = F.NrFact
```

Tabelei PRODUSE i s-a asociat sinonimul P, LINIIFACT LF, iar pentru FACTURI F. Sinonimele prefixează (când este necesar) numele atributelor în clauzele SELECT și WHERE (eventual ORDER BY, GROUP BY).

Există situații în care utilizarea sinonimelor n-are cu nimic de-a face cu lenea/comoditatea sau depresiile nervoase. În afara interogărilor mai mult sau mai

puțin corelate, pe care le vom discuta în capitole viitoare, o operațiune în care musai trebuie folosite sinonimele este joncționarea tabeli cu ea-însăși sau, cum spunem noi, moldovenii, cu *dânsa-însăși*¹⁵.

Care sunt județele din regiunea țării în care se află județul Iași ?

O asemenea problemă se rezolvă dintr-o clipită folosind subconsultări. Noi, însă, vrem morțiș să folosim doar ceea ce am discutat în acest capitol. Iată o soluție interesantă:

```
SELECT j1.*  
FROM judete j1  
      INNER JOIN judete j2 ON j1.Regione=j2.Regione  
WHERE j2.Judet='Iasi'
```

Să discutăm pe etape ceea ce se întâmplă. Joncțiunea a două instanțe ale tabeli JUDETE pe baza egalității valorilor atributului Regione (vezi figura 5.43)

```
SELECT *  
FROM judete j1  
      INNER JOIN judete j2 ON j1.Regione=j2.Regione
```

va obține un rezultat în care pe fiecare linie se află două județe (sau chiar același județ) care sunt din aceeași regiune¹⁶.

¹⁵ Asta este una din glumele nesărate din prima ediție.

¹⁶ Adevărul este că, pentru a obține exact titlatura coloanelor din figură, interogarea are forma:
SELECT j1.Jud AS "j1.Jud", j1.Judet AS "j1.Judet", j1.Regione AS "j1.Regione", j2.Jud AS "j2.Jud", j2.Judet
AS "j2.Judet", j2.Regione AS "j2.Regione" FROM judete j1 INNER JOIN judete j2 ON
j1.Regione=j2.Regione

j1.Jud	j1.Judet	j1.Regione	j2.Jud	j2.Judet	j2.Regione
VS	Vaslui	Moldova	IS	Iasi	Moldova
SV	Suceava	Moldova	IS	Iasi	Moldova
NT	Neamt	Moldova	IS	Iasi	Moldova
VN	Vrancea	Moldova	IS	Iasi	Moldova
IS	Iasi	Moldova	IS	Iasi	Moldova
VS	Vaslui	Moldova	VN	Vrancea	Moldova
SV	Suceava	Moldova	VN	Vrancea	Moldova
NT	Neamt	Moldova	VN	Vrancea	Moldova
VN	Vrancea	Moldova	VN	Vrancea	Moldova
IS	Iasi	Moldova	VN	Vrancea	Moldova
VS	Vaslui	Moldova	NT	Neamt	Moldova
SV	Suceava	Moldova	NT	Neamt	Moldova
NT	Neamt	Moldova	NT	Neamt	Moldova
VN	Vrancea	Moldova	NT	Neamt	Moldova
IS	Iasi	Moldova	NT	Neamt	Moldova
VS	Vaslui	Moldova	SV	Suceava	Moldova
SV	Suceava	Moldova	SV	Suceava	Moldova
NT	Neamt	Moldova	SV	Suceava	Moldova
VN	Vrancea	Moldova	SV	Suceava	Moldova
IS	Iasi	Moldova	SV	Suceava	Moldova
VS	Vaslui	Moldova	VS	Vaslui	Moldova
SV	Suceava	Moldova	VS	Vaslui	Moldova
NT	Neamt	Moldova	VS	Vaslui	Moldova
VN	Vrancea	Moldova	VS	Vaslui	Moldova
IS	Iasi	Moldova	VS	Vaslui	Moldova
TM	Timis	Banat	TM	Timis	Banat

Figura 5.43. Auto-joncțiunea tabelului JUDETE după atributul Regione

Știind că pe o linie avem două județe (sau unul) din aceeași regiune, filtrăm una dintre instanțele tabelului JUDETE (a doua, în cazul nostru), păstrând numai liniile în care j2.Judet este Iași – vezi figura 5.44:

```
SELECT *
```

```
FROM judete j1 INNER JOIN judete j2 ON j1.Regione=j2.Regione
```

```
WHERE j2.Judet='Iasi'
```

j1.Jud	j1.Judet	j1.Regione	j2.Jud	j2.Judet	j2.Regione
IS	Iasi	Moldova	IS	Iasi	Moldova
VN	Vrancea	Moldova	IS	Iasi	Moldova
NT	Neamt	Moldova	IS	Iasi	Moldova
SV	Suceava	Moldova	IS	Iasi	Moldova
VS	Vaslui	Moldova	IS	Iasi	Moldova

Figura 5.44. Filtrarea liniilor din auto-joncțiune

Acum vedem că cel de-al doilea „rând de coloane” – cel care corespunde instanței j2 – este de prisos, așa încât prefixăm asteriscul cu aliasul (sinonimul local) j1, obținând înregistrările din figura 5.45.

JUD	J1	JUDET	J2	REGIUNE
IS		Iasi		Moldova
VN		Vrancea		Moldova
NT		Neamt		Moldova
SV		Suceava		Moldova
VS		Vaslui		Moldova

Figura 5.45. Rezultat final – „colegii” de regiune ai județului Iași

Acum, dacă tot ne-am chinuit așa de tare, revenim la exemplul 19 din algebra relațională: *Ce facturi au fost emise în aceeași zi cu factura 1120 ?*

Pe calapodul soluției de mai sus, redactăm una care, pe deasupra, elimină din rezultat factura 1120 (vezi figura 5.46), deci răspunde cu exactitate enunțului:

```
SELECT F2.NrFact
FROM facturi F1 INNER JOIN facturi F2
ON F1.DataFact = F2.DataFact AND F1.NrFact=1120
WHERE F2.NrFact <> 1120
```

Pentru a (mai) dilua din plictiseală am introdus o parte din predicatul de selecție (*F1.NrFact=1120*) chiar în clauza FROM.

NRFAC
1119
1121
1122

Figura 5.46. Facturile emise în aceeași zi cu 1120

Despre intersecția realizată prin joncțiune am discutat la exemplul 17 din paragraful 4.4.4 - *În ce zile s-au vândut și produsul cu denumirea "Produs 1" și cel cu denumirea "Produs 2" ?*

Joncționăm o instanță obținută prin joncțiunea PRODUSE-LINIIFACT-FACTURI, în care DenPr = 'Produs 1', cu o altă instanță a aceleiași combinații de tabele, în care DenPr = 'Produs 2'.

```
SELECT DISTINCT F1.DataFact
FROM produse P1
INNER JOIN liniifact LF1 ON P1.CodPr = LF1.CodPr
INNER JOIN facturi F1 ON LF1.NrFact = F1.NrFact
INNER JOIN facturi F2 ON F1.DataFact=F2.DataFact
INNER JOIN liniifact LF2 ON LF2.NrFact = F2.NrFact
INNER JOIN produse P2 ON LF2.CodPr = P2.CodPr
```

```
WHERE P1.DenPr = 'Produs 1' AND P2.DenPr = 'Produs 2'
```

Inserând filtrarea după cele două produse în clauza FROM se obține o formă ușor schimbată a consultării:

```
SELECT DISTINCT F1.DataFact
FROM produse P1
    INNER JOIN liniifact LF1 ON P1.CodPr = LF1.CodPr
        AND P1.DenPr = 'Produs 1'
    INNER JOIN facturi F1 ON LF1.NrFact = F1.NrFact
    INNER JOIN facturi F2 ON F1.DataFact=F2.DataFact
    INNER JOIN liniifact LF2 ON LF2.NrFact = F2.NrFact
    INNER JOIN produse P2 ON LF2.CodPr = P2.CodPr
        AND P2.DenPr = 'Produs 2'
```