

Capitolul 6. Funcții SQL

Am făcut, deja, frugal, cunoștință cu o serie de funcții pentru numere, siruri și date calendaristice. În acest capitol vom dezvolta subiectul, fără a epuiza, nici pe departe, impresionanta tipologie a funcțiilor disponibile. Cu atât mai mult cu cât portofoliul funcțiilor este destul de eterogen de la SGBD la SGBD. Până spre finalul lucrării vom avea în vedere, exclusiv, funcțiile standard (*buit-in functions*), urmând ca în capitolul 16 să cheltuim câteva pagini pe funcțiile definite de utilizator.

Prima ediție a cărții de față nu a avut un capitol special dedicat funcțiilor SQL. Am apelat la funcțiile sistem (predefinite) doar pe măsura nevoilor de interogare, comentându-le în exemple. Pentru a lămuri cât de cât discuția, trebuie subliniat că funcțiile din primele șase paragrafe (6.1-6.6) se aplică la nivel de linie, asupra valorii unui atribut sau unei expresii. Unii autori¹ le numesc *scalare* prin comparație cu funcțiile *agregat*, care se calculează la nivel de grup de înregistrări (sau la nivelul tuturor liniilor din tabelă care satisfac o eventuală condiție de filtrare). Adevărul este că și funcțiile agregat sunt oarecum scalare, întrucât returnează o valoare simplă (să-i zicem, cu stângere de inimă, *atomică*). Dacă ne raportăm la argumentele asupra cărora funcțiile operează, atunci categorizarea este acceptabilă.

După tipul datelor asupra cărora se aplică, putem delimita funcții dedicate numerelor, sirurilor de caractere, datelor calendaristice și intervalelor. La acestea mai putem adăuga funcții de conversie (a numerelor în siruri de caractere, a sirurilor în date calendaristice etc.), funcții pentru manipularea valorilor NULLe (vezi capitolul 8), funcții pentru colecții (vectori) și funcții sistem – cele care returnează anumiți parametri ai SGBD-ului, schemei sau sesiunii curente (ora sistemului, utilizatorul curent etc.). Categorizarea se poate face și mai fin, în sensul că, spre exemplu, putem face delimitarea între funcții aplicate asupra sirurilor ce returnează tot siruri (CONCAT, LTRIM, UPPER) și funcții aplicate asupra sirurilor ce returnează numere (CHARACTER_LENGTH, POSITION). La fel și cu multe funcții destinate datelor și intervalelor calendaristice.

¹ [Jones s.a. 05] p.33, [Sirbu 07] p.12

6.1. Funcții pentru numere

Nu are rost, pentru scopul lucrării de față, să detaliem funcțiile matematice de tipul ABS (ce returnează valoarea absolută dintr-un număr), COS, SIN, TAN, ACOS, ASIN, ATAN (cosinus, sinus, tangentă arccosinus, arcsinus, arctangentă) s.a.m.d.² Funcții precum ABS (pentru valoarea absolută a unui argument), MOD (pentru restul împărțirii a două numere) – vezi figura 6.1 – au o incidență relativ redusă în aplicațiile pentru afaceri. Iată un exemplu pe sintaxa Oracle:

```
SELECT ABS(15/7), ABS(-15/7), MOD(15,7), MOD(-15,7) FROM dual
```

ABS(15/7)	ABS(-15/7)	MOD(15,7)	MOD(-15,7)
2,14285714285714285714285714285714	2,14285714285714285714285714285714	1	-1

Figura 6.1. Exemplu de folosire a funcțiilor ABS și MOD (în Oracle SQL Developer).

Diferențele de la SGBD la SGBD pot să apară nu numai în titulatura unor funcții, după cum vom vedea în acest capitol, dar și la „precizia” afișării. Aceeași interogare executată în PostgreSQL (fără *FROM dual*) sau în DB2 (înlocuind DUAL cu *SYSIBM.SYSDUMMY1* sau *SYSIBM.DUAL*) extrage un rezultat ca în figura 6.2³ în care am fi tentați să credem că rezultatul funcției ABS este un întreg (INTEGER).

abs integer	abs integer	mod integer	mod integer
1	2	1	-1

Figura 6.2. Aceeași funcție ABS, precizie diferită în PostgreSQL pgAdmin

În realitate, diferența apare de la modul în care PostgreSQL-ul evaluează raportul 15/7. Ca exemplu suplimentar, calculăm procentul „unitar” de TVA

² Pentru o tratare detaliată a funcțiilor matematice în SQL, vezi [Jones s.a. 05], pp.69-86

³ Trebuie să precizăm că fiecare SGBD/utilitar de conectare pune la dispoziție comenzi pentru configurarea multor parametri de afișare și interpretare a datelor de tip numeric, date calendaristice etc.

pentru fiecare produs vândut în factura 1114, procent unitar care este produsul $ProcTVA * PretUnit$, aplicând funcția ABS:

```
SELECT DenPr, Cantitate, PretUnit, ProcTVA,
       PretUnit * ProcTVA AS ProcTVAUnitar1,
       ABS(-PretUnit * ProcTVA) AS ABS_ProcTVAUnitar
  FROM liniifact INNER JOIN produse ON liniifact.CodPr=produse.CodPr
 WHERE NrFact=1114
```

și vedem că rezultatul este unul cât se poate de normal – vezi figura 6.3 -, rezultat ce coincide cu cele obținute în urma lansării acestei interogări în SQLServer și DB2.

The screenshot shows a pgAdmin window titled "Query - sql2008 on postgres@localhost:5432". The query pane contains the following SQL code:

```
SELECT DenPr, Cantitate, PretUnit, ProcTVA,
       PretUnit * ProcTVA AS ProcTVAUnitar1,
       ABS(-PretUnit * ProcTVA) AS ABS_ProcTVAUnitar
  FROM liniifact INNER JOIN produse ON liniifact.CodPr=produse.CodPr
 WHERE NrFact=1114
```

The output pane displays the results in a table:

	denpr character varying	cantitate numeric(8,0)	pretunit numeric(9,2)	procvta numeric(2,2)	procTVAunitar1 numeric	abs_procTVAunitar numeric
1	Produs 2	70	1070.00	0.09	96.3000	96.3000
2	Produs 4	30	1705.00	0.09	153.4500	153.4500
3	Produs 5	700	7064.00	0.19	1342.1600	1342.1600

Figura 6.3. Funcția ABS în PostgreSQL pgAdmin

Echivalent funcției *modulo* (MOD), MS SQL Server folosește operatorul procent (%), așa încât prima interogare din acest paragraf poate funcționa în SQL Server doar în forma:

```
SELECT ABS(15/7), ABS(-15/7), 15 % 7, -15 % 7
```

Tot pentru încăzire, exemplificăm folosirea altor două funcții, POWER (pentru ridicarea la putere și SQRT (radical de ordinul 2 sau rădăcină pătrată) Numerele calculate sunt 2^{10} , $2^{0.5}$, 2^{-3} și $\sqrt{2}$ (ultimul este radical din 2), pe sintaxa Oracle:

```
SELECT POWER(2,10), POWER(2, 0.5), POWER(2, -3), SQRT(2)
      FROM dual
```

Interogarea funcționează în DB2, MS SQL Server, Oracle, și PostgreSQL (– vezi figura 6.4.), chiar dacă numărul de poziții fracționare afișat este diferit. Funcția POWER este implementată cu numele de POW în MySQL (nume folosibil și în PostgreSQL).

DB2:

```
SELECT POWER(2,10), POWER (2, 0.5), POWER(2, -3), SQRT(2) FROM sysibm.dual
```

	1	2	3	4	5
	1024	1.414	0	1.414	

Oracle:

```
SELECT POWER(2,10), POWER (2, 0.5), POWER(2, -3), SQRT(2) FROM dual
```

POWER(2,10) POWER(2,0,5) POWER(2,-3) SQRT(2)
1024 1,41421356237309504880168872420969807855 0,125 1,41421356237309504880168872420969807855

PostgreSQL:

```
SELECT POWER(2,10), POWER (2, 0.5), POWER(2, -3), SQRT(2)
```

	power double precis	power numeric	power double precis	sqrt double precis
1	1024	1.41421356237	0.125	1.41421356237

SQL Server:

```
SELECT POWER(2,10), POWER (2, 0.5), POWER(2, -3), SQRT(2)
```

	(No column name)	(No column name)	(No column name)	(No column name)
1	1024	1	0	1.4142135623731

Figura 6.4. Exemplu de folosire a funcțiilor POWER și SQRT în cele patru servere BD

Probabil că cele mai frecvent folosite dintre funcțiile numerice sunt ROUND, TRUNC, CEILING (sau CEIL) și FLOOR. Iată un exemplu executat în Oracle, obiectul funcțiilor fiind rezultatul împărțirii lui 7 la 11:

```
SELECT '7/11' , 7/11 AS functie FROM dual
UNION
SELECT 'CEILING(7/11)', CEIL (7/11) FROM dual
UNION
SELECT 'FLOOR(7/11)', FLOOR (7/11) FROM dual
UNION
SELECT 'ROUND(7/11, 2)', ROUND (7/11,2) FROM dual
UNION
SELECT 'TRUNC(7/11, 2)', TRUNC (7/11,2) FROM dual
```

Rezultatul din figura 6.5 arată (dacă mai era nevoie) că: prin CEIL (CEILING) am obținut cel mai apropiat număr întreg superior lui 0,636363..., adică 1; prin FLOOR am obținut cel mai apropiat număr întreg inferior lui 0,636363..., adică 0; iar diferența dintre ROUND și TRUNC este cea dintre rotunjire și trunchiere (la două poziții fractionare).

Figura 6.5. Comparație între funcțiile CEILING, FLOOR, ROUND și TRUNC

Din nou PostgreSQL-ul și DB2-ul fac notă discordantă din cauza modului de evaluare a raportului 7/11 - vezi figura 6.6.

```

Query - sql2008 on postgres@localhost:5432 *
File Edit Query Favourites Macros View Help
SELECT '7/11' , 7/11 AS functie UNION
SELECT 'CEILING(7/11)', CEIL (7/11) UNION
SELECT 'FLOOR(7/11)', FLOOR (7/11) UNION
SELECT 'ROUND(7/11, 2)', ROUND (7/11,2) UNION
SELECT 'TRUNC(7/11, 2)', TRUNC (7/11,2)

Output pane
Data Output Explain Messages History
?column? functie
text double precision
1 7/11 0
2 CEILING(7/11) 0
3 FLOOR(7/11) 0
4 ROUND(7/11, 2) 0
5 TRUNC(7/11, 2) 0

```

Figura 6.6. Alte rezultate în PostgreSQL (și DB2) pentru funcțiile CEILING, FLOOR, ROUND și TRUNC din cauza evaluării fracției 7/11

Ne mai liniștim dacă revenim la exemplul legat de calcularea TVA-ului unitar pentru produsele din factură 1114. Rezultatul interogării:

```

SELECT DenPr, Cantitate, PretUnit, ProcTVA,
       PretUnit * ProcTVA AS TVAUitar1,
       CEIL(PretUnit * ProcTVA) AS TVAUUn_CEIL,
       FLOOR(PretUnit * ProcTVA) AS TVAUUn_FLOOR,
       TRUNC(PretUnit * ProcTVA,0) AS TVAUUn_TRUNC0,
       TRUNC(PretUnit * ProcTVA,1) AS TVAUUn_TRUNC1,
       ROUND(PretUnit * ProcTVA,0) AS TVAUUn_ROUND0,
       ROUND(PretUnit * ProcTVA,1) AS TVAUUn_ROUND1
FROM liniifact INNER JOIN produse ON liniifact.CodPr=produse.CodPr
WHERE NrFact=1114

```

```

Query - sql2008 on postgres@localhost:5432 *
File Edit Query Favourites Macros View Help
SELECT DenPr, Cantitate, PretUnit, ProcTVA,
       PretUnit * ProcTVA AS TVAUitar1,
       CEIL(PretUnit * ProcTVA) AS TVAUUn_CEIL,
       FLOOR(PretUnit * ProcTVA) AS TVAUUn_FLOOR,
       TRUNC(PretUnit * ProcTVA,0) AS TVAUUn_TRUNC0,
       TRUNC(PretUnit * ProcTVA,1) AS TVAUUn_TRUNC1,
       ROUND(PretUnit * ProcTVA,0) AS TVAUUn_ROUND0,
       ROUND(PretUnit * ProcTVA,1) AS TVAUUn_ROUND1
FROM liniifact INNER JOIN produse ON liniifact.CodPr=produse.CodPr
WHERE NrFact=1114

Output pane
Data Output Explain Messages History
denpr cantitate pretunit procvta tvaunitar1 tvaun_ceil tvaun_floor tvaun_trunc0 tvaun_trunc1 tvaun_round0 tvaun_round1
characte numeric(10) numeric numeric numeric numeric numeric numeric numeric numeric numeric
1 Produs 2 70 1070.00 0.09 96.3000 97 96 96 96.3 96 96.3
2 Produs 4 30 1705.00 0.09 153.4500 154 153 153 153.4 153 153.5
3 Produs 5 700 7064.00 0.19 1342.1600 1343 1342 1342 1342.1 1342 1342.2

```

Figura 6.7. Funcțiile CEILING, FLOOR, ROUND și TRUNC în PostgreSQL

este unul cât se poate de corect – sau cel puțin aşa pare (vezi figura 6.7). Lucrurile stau identic în DB2. SQL Server prezintă două diferențe: în loc de CEIL este obligatoriu să se folosească CEILING, iar trunchierea se face tot cu funcția ROUND, adăugând un al treilea parametru (1):

```

SELECT DenPr, Cantitate, PretUnit, ProcTVA,
       PretUnit * ProcTVA AS TVAUnitar1,
       CEILING(PretUnit * ProcTVA) AS TVAUUn_CEILING,
       FLOOR(PretUnit * ProcTVA) AS TVAUUn_FLOOR,
       ROUND(PretUnit * ProcTVA,0,1) AS TVAUUn_TRUNC0,
       ROUND(PretUnit * ProcTVA,1,1) AS TVAUUn_TRUNC1,
       ROUND(PretUnit * ProcTVA,0) AS TVAUUn_ROUND0,
       ROUND(PretUnit * ProcTVA,1) AS TVAUUn_ROUND1
FROM liniifact INNER JOIN produse ON liniifact.CodPr=produse.CodPr
WHERE NrFact=1114

```

Trunchierea și rotunjirea se pot face nu numai la poziții fracționare, ci și la nivel de unități, zeci, sute, mii s.a.m.d. Înmulțim rezultatul diviziunii $7/11$ cu 10000 (adică 10^4), prilej de exemplificare a funcției POWER (ridicare la putere), păstrând funcțiile din SELECT-ul anterior (vezi figura 6.8):

```
SELECT '7 / 11 * 10000', 7/11 * POWER (10,4) AS functie FROM dual
UNION
SELECT 'CEILING(7 / 11 * 10000)', CEIL (7/11 * POWER (10,4)) FROM dual
UNION
SELECT 'FLOOR(7 / 11 * 10000)', FLOOR (7/11 * POWER (10,4)) FROM dual
UNION
SELECT 'ROUND(7/11 * POWER (10,4), -2)', ROUND (7/11 * POWER (10,4), -2)
FROM dual
UNION
SELECT 'TRUNC(7/11 * POWER (10,4), -2)', TRUNC (7/11 * POWER (10,4),-2)
FROM dual
```

Figura 6.8. Rotunjire și truchiere la nivelul sutelor (Oracle)

După cum vă așteptați, DB2 (interrogarea următoare), SQL Server și PostgreSQL ne furnizează un rezultat eronat – vezi figura 6.9:

```
SELECT '7 / 11 * 10000' AS Functie, 7/11 * POWER (10,4) AS Resultat
FROM SYSIBM.SYSDUMMY1
        UNION
SELECT 'CEILING(7 / 11 * 10000)', CEIL (7/11 * POWER (10,4))
```

```

FROM SYSIBM.SYSDUMMY1
    UNION
SELECT 'FLOOR(7 / 11 * 10000)', FLOOR (7/11 * POWER (10,4))
FROM SYSIBM.SYSDUMMY1
    UNION
SELECT 'ROUND(7/11 * POWER (10,4), -2)', ROUND (7/11 * POWER (10,4), -2)
FROM SYSIBM.SYSDUMMY1
    UNION
SELECT 'TRUNC(7/11 * POWER (10,4), -2)' AS "Functie",
      TRUNC (7/11 * POWER (10,4),-2) AS "Rezultat"
FROM SYSIBM.SYSDUMMY1

```

1	2
7 / 11 * 10000	0
CEILING(7 / 11 * 10000)	0
FLOOR(7 / 11 * 10000)	0
ROUND(7/11 * POWER (10,4), -2)	0
TRUNC(7/11 * POWER (10,4), -2)	0

Figura 6.9. Rotunjire și truchiere eronate la nivelul sutelor (DB2)

Unde mai punem că cele două coloane nu sunt denumite cum am dori, chiar dacă folosim clauza AS (din cauza UNION-ilor). În schimb, dacă aplicăm funcțiile la datele din tabelele LINIIFACT și PRODUSE (ca în figura 6.7), rezultatul este cât se poate de corect – vezi figura 6.10:

```

SELECT DenPr, Cantitate, PretUnit, ProcTVA,
       PretUnit * ProcTVA AS TVAUUnitar1,
       TRUNC(PretUnit * ProcTVA,-1) AS "TVAUn_TRUNC-1",
       ROUND(PretUnit * ProcTVA,-1) AS "TVAUn_ROUND-1",
       TRUNC(PretUnit * ProcTVA,-2) AS "TVAUn_TRUNC-2",
       ROUND(PretUnit * ProcTVA,-2) AS "TVAUn_ROUND-2"
FROM liniifact INNER JOIN produse ON liniifact.CodPr=produse.CodPr
WHERE NrFact=1114

```

DENPR	CANTITATE	PRETUNIT	PROCTVA	TVAUNITARI	TVAUH_TRUNC-1	TVAUH_ROUND-1	TVAUH_TRUNC-2	TVAUH_ROUND-2
Produs 2	70	1.070,00	0,09	96,3000	90,0000	100,0000	0,0000	100,0000
Produs 4	30	1.705,00	0,09	153,4500	150,0000	150,0000	100,0000	200,0000
Produs 5	700	7.064,00	0,19	1.342,1600	1.340,0000	1.340,0000	1.300,0000	1.300,0000

Figura 6.10. Rotunjire și truchiere corecte la nivelul sutelor (DB2)

Același rezultat îl obținem în MS SQL Server prin interogarea:

```

SELECT DenPr, Cantitate, PretUnit, ProcTVA,
       PretUnit * ProcTVA AS TVAUUnitar1,
       ROUND(PretUnit * ProcTVA,-1,1) AS [TVAUn_TRUNC-1],
       ROUND(PretUnit * ProcTVA,-1) AS [TVAUn_ROUND-1],
       ROUND(PretUnit * ProcTVA,-2,1) AS [TVAUn_TRUNC-2],
       ROUND(PretUnit * ProcTVA,-2) AS [TVAUn_ROUND-2]
FROM liniifact INNER JOIN produse ON liniifact.CodPr=produse.CodPr
WHERE NrFact=1114

```

6.2. Funcții pentru siruri de caractere

O bună parte dintre cele mai utile funcții pentru siruri de caractere au fost deja folosite în capitoalele 3 și 5 la definirea unor restricții, inserări de linii și interogări. Cele mai folosite din această categorie sunt funcțiile CONCAT, LOWER, UPPER, LENGTH, SUBSTR, TRIM și REPLACE⁴.

Despre CONCAT se poate spune că, în afara acțiunii sale (concatenarea a două sau mai multe siruri de caractere), prezintă uneori (în Oracle și DB2) limitarea, destul de apăsătoare, de a avea numai două argumente. De exemplu, o patră interogare din paragraful 5.3 (cu rezultatul în figura 5.9) se poate scrie în Oracle/DB2 astfel:

```
SELECT CONCAT ('Judetul ', Judet) || CONCAT (' se afla in ', Regiune)
      AS Exemplu_Concatenare
   FROM judete
```

Alte dialecte (ex. MySQL) sunt mai darnice, acceptând oricâte argumente pentru CONCAT⁵, iar PostgreSQL și MS SQL Server nu au implementată această funcție.

Funcțiile LOWER, UPPER, implementate în toate cele patru servere BD, au fost întrebuiște serios chiar de la crearea tabelelor (atunci când doream ca valorile unui atribut să fie introduse numai cu litere mici sau majuscule). Iată un exemplu în care apare și o funcție mai puțin standardizată, și anume cea care transformă inițiala fiecărui cuvânt din sir în majusculă, restul literelor din fiecare cuvânt fiind transformate în litere mici. În Oracle și PostgreSQL funcția este INITCAP, dar în alte dialecte poate avea nume diferit (ex. în Visual FoxPro este PROPER), iar în altele (ex. DB2, MS SQL Server) nu este deloc. Rezultatul din Oracle este cel din figura 6.11.

```
SELECT ' Sirul initial', ' sIr DE TeSt' AS sir_Initial FROM dual
      UNION
SELECT ' MAJUSCULE', UPPER(' sIr DE TeSt') FROM dual
      UNION
SELECT ' litere mici', LOWER(' sIr DE TeSt') FROM dual
      UNION
```

⁴ Există diferențe apreciabile în enumerarea funcțiilor pentru siruri de caractere. Astfel, dacă în [Melton & Simon 02], pp.147-151, sunt prezentate: SUBSTRING, UPPER, LOWE, TRIM, TRANSLATE, CONVERT și OVERLAY, în [Jones s.a. 05], pp.65-69 sunt decrise: ASCII, CHR (CHAR), CONCAT, LOWER, UPPER, LENGTH (LEN) și REPLACE. Noi am inclus și cele care operează asupra sirurilor, dar returnează numere.

⁵ În schimb MySQL nu acceptă concatenarea cu operatorul || (sau +).

```
SELECT 'Initiala (initiale) majuscula. Restul, litere mici', INITCAP(' sIr DE TeSt')
FROM dual
```

'SIRULINITIAL'	SIR_INITIAL
Sirul initial	sIr DE TeSt
MAJUSCULE	SIR DE TEST
litere mici	sir de test
Initiala (initiale) majuscula. Restul, litere mici	Sir De Test

Figura 6.11. UPPER, LOWER, INITCAP

Tot în capitolul 3 am folosit și funcții de eliminare a spațiilor dintr-un sir de caractere - TRIM. Pentru a scăpa de spațiile de la începutul sirului foloseam LTRIM (de la LEFT TRIM), iar pentru a le elimina pe cele de la sfârșitul sirului foloseam RTRIM (RIGHT TRIM).

Standardele SQL:1999-2008 conțin doar funcția TRIM cu argumentele LEADING, TRAILING și BOTH pentru a indica modul în care se realizează „tunsul” spațiilor. Interrogarea următoare a fost executată în Oracle, de aici *FROM dual*-ul de rigoare:

```
SELECT '1: Sirul initial', '{' || ' sIr DE TeSt ' || '}' AS sir_Initial
FROM dual
UNION
SELECT '2: TRIM - LEADING', '{' ||
      TRIM( LEADING '' FROM (' sIr DE TeSt ')) || '}'
FROM dual
UNION
SELECT '3: TRIM - TRAILING', '{' ||
      TRIM( TRAILING '' FROM (' sIr DE TeSt ')) || '}'
FROM dual
UNION
SELECT '4: TRIM - BOTH', '{' || TRIM( BOTH '' FROM (' sIr DE TeSt '))
|| '}'
FROM dual
```

Pentru o mai bună vizualizare a spațiilor de la începutul și sfârșitul sirului (vezi figura 6.12), am folosit acoladele. Funcțiile operează similar în DB2 și PostgreSQL, nu însă și în MS SQL Server.

Results:	'1:SIRULINITIAL'	SIR_INITIAL
1 1: Sirul initial	{ sIr DE TeSt }	
2 2: TRIM - LEADING	{sIr DE TeSt }	
3 3: TRIM - TRAILING	{ sIr DE TeSt}	
4 4: TRIM - BOTH	{sIr DE TeSt}	

Figura 6.12. Funcția TRIM

În majoritatea SGBD-urilor, de ani buni se folosesc funcțiile la care am apelat și noi în capitolul 3: LTRIM și RTRIM, tăierea la ambele capete fiind realizată fie prin

TRIM (ex. Oracle, PostgreSQL), ALLTRIM (FoxPro) sau LTRIM(RTRIM(...)). Iată transcrierea în MS SQL Server a interogării de mai sus:

```

SELECT '1: Sirul initial', '{' + ' sIr DE TeSt   ' + '}' AS sir_Initial
      UNION
SELECT '2: TRIM - LEADING', '{' + LTRIM(' sIr DE TeSt   ') + '}' 
      UNION
SELECT '3: TRIM - TRAILING', '{' + RTRIM(' sIr DE TeSt   ') + '}' 
      UNION
SELECT '4: TRIM - BOTH', '{' + LTRIM( RTRIM(' sIr DE TeSt   ')) + '}'
```

Cel mai frecvent, funcția se folosește pentru eliminarea spațiilor. Însă, după cum probabil ati observat, după LEADING, TRAILING și BOTH spațiul a fost indicat explicit prin ' '. Putem specifica orice caracter în locul spațiului. De exemplu, **TRIM (BOTH 'E' FROM 'ESCHIVARE')** va returna 'SCHIVAR' .

Dacă funcția TRIM elimină spații (sau un alt caracter), multe dialecte SQL au o funcție opusă – PAD care “umple” cu spațiu sau alt caracter un sir de caractere până acesta (șirul) atinge o anumită lungime. Umplerea se poate face la stânga (LPAD, adică LEFT PAD) sau la dreapta (RPAD, de la RIGHT PAD). Astfel, dacă în tabela CLIENTI dorim să afișăm toate valorile atributului Adresa pe 30 caractere și, acolo unde lungimea este mai mică decât 30 să se umple cu asteriscurile necesare, putem folosi funcțiile RPAD și LPAD, al căror efect este vizibil în figura 6.13:

```

SELECT DenCl, Adresa,
       RPAD(adresa, 30, '*') AS Exemplu_RPAD,
       LPAD(adresa, 30, '*') AS Exemplu_LPAD
FROM clienti
```

DENCL	ADRESA	EXEMPLU_RPAD	EXEMPLU_LPAD
Client 1 SRL	Tranzitiei, 13 bis	Tranzitiei, 13 bis*****	*****Tranzitiei, 13 bis
Client 2 SA	(null)	(null)	(null)
Client 3 SRL	Prosperitatii, 22	Prosperitatii, 22*****	*****Prosperitatii, 22
Client 4	Sapientei, 56	Sapientei, 56*****	*****Sapientei, 56
Client 5 SRL	(null)	(null)	(null)
Client 6 SA	Pacientei, 33	Pacientei, 33*****	*****Pacientei, 33
Client 7 SRL	Victoria Capitalismului, 2	Victoria Capitalismului, 2****	****Victoria Capitalismului, 2

Figura 6.13. Funcții PAD

Din păcate, cei care se așteptau ca alinierea să fie perfectă (la urma urmei, am stabilit limita de reprezentare la 30 de caractere) au toate motivele să fie

dezamăgiți, întrucât în mediile grafice (cum este Oracle SQL Developer sau PostgreSQL pgAdmin) spațiul necesar reprezentării unui caracter diferă în funcție de forma acestuia. În vederea evitării acestui tip de probleme, MS SQL Server-ul și DB2 nu au implementat o asemenea funcție.

Lungimea unui sir de caractere poate fi determinată în standardul SQL prin funcția CHARACTER_LENGTH sau "prescurtarea" CHAR_LENGTH⁶. SGBD-urile sunt mai pragmatice, funcția fiind, simplu, LENGTH (în toate cele patru servere) sau LEN (MS SQL Server). și în privința funcției pentru extragerea unei porțiuni dintr-un sir de caractere – SUBSTRING – apar diferențe între standard și implementări. În standardele post-SQL:1999, pentru a extrage primele patru și ultimele patru caractere dintr-un sir –vezi figura 6.14 - sintaxa ar fi:

```
SELECT 'sIr DE TeSt' AS sir,
       CHARACTER_LENGTH('sIr DE TeSt') AS lungime,
       SUBSTRING('sIr DE TeSt' FROM 1 FOR 4) AS Primele_patru,
       SUBSTRING('sIr DE TeSt' FROM
                  CHARACTER_LENGTH('sIr DE TeSt')-3 FOR 4)
              AS Ultimele_patru
```

Results:			
SIR	LUNGIME	PRIMELE_PATRU	ULTIMELE_PATRU
1 sIr DE TeSt	11	sIr	TeSt

Figura 6.14. Funcțiile CHARACTER_LENGTH și SUBSTRING

DB2, PostgreSQL și Oracle prezintă funcția SUBSTR cu trei argumente: sirul propriu-zis, poziția de pe care începe extragerea și numărul de caractere extras:

```
SELECT 'sIr DE TeSt' AS sir, LENGTH('sIr DE TeSt') AS lungime,
       SUBSTR('sIr DE TeSt',1,4) AS Primele_patru,
       SUBSTR('sIr DE TeSt', LENGTH('sIr DE TeSt')-3,4) AS Ultimele_patru7
```

MS SQL Server folosește LEN pe post de LENGTH și SUBSTRING pe post de SUBSTR. De asemenea, dacă porțiunea extrasă se află la începutul sau la sfârșitul sirului de caractere, se pot folosi, în loc de SUBSTRING, funcțiile LEFT și RIGHT:

```
SELECT 'sIr DE TeSt' AS sir, LEN('sIr DE TeSt') AS lungime,
       LEFT('sIr DE TeSt',4) AS Primele_patru,
       RIGHT('sIr DE TeSt', 4) AS Ultimele_patru
```

⁶ [Melton & Simon 02], pp.144-145

⁷ Interogarea funcționează în PostgreSQL. Firește, în Oracle se adaugă FROM dual, iar în DB2 FROM sysibm.dual

Diferențe majore apar și în cazul funcției care caută aparția unui sir de caractere - *sir1* - în alt sir (tot de caractere) - *sir2* - și returnează poziția pe care apare *sir1* în *sir2* sau 0, când căutarea este fără succes. În SQL:1999 apare funcția POSITION. Spre exemplu, *POSITION ('T' IN 'sIr DE TeSt')* returnează 8, pentru că T apare pe a opta poziție din 'sIr DE TeSt'. SGBD-urile, însă, dispun de funcții echivalente, cum este INSTR – care este ceva mai elegantă, deoarece permite căutarea începând cu o poziție specificată și, în plus, caută a *n*-a apariție a lui *sir1* în *sir2*. Exemplul următor (executat în Oracle) caută și afișează prima, a doua și a treia apariție a literei e în sir de test, precum și prima apariție a succesiunii sirul în același sir de test. Căutarea se face începând cu primul caracter din *sir de test*.

```
SELECT 'sir de test' AS sir,
       INSTR('sir de test','e',1, 1) AS Poz1_e,
       INSTR('sir de test','e',1, 2) AS Poz2_e,
       INSTR('sir de test','e',1, 3) AS Poz3_e,
       INSTR('sir de test','sirul',1) AS Poz1_sirul
  FROM dual
```

După cum arată și rezultatul din figura 6.15, atunci când nu există o a *n*-a apariție a lui *sir1* în *sir2* (a treia apariție a lui e, prima apariție a lui *sirul*, rezultatul este zero.

SIR	POZ1_E	POZ2_E	POZ3_E	POZ1_SIRUL
1 sir de test	6	9	0	0

Figura 6.15. Funcția INSTR (echivalenta funcției POSITION)

În PostgreSQL se poate determina doar prima apariție a unui subșir într-un sir, cu ajutorul funcției POSITION:

```
SELECT 'sir de test' AS sir, POSITION('e' IN 'sir de test') AS Poz1_e
  în DB se poate folosi și POSSTR și LOCATE:
```

```
SELECT 'sir de test' AS sir,
       POSSTR('sir de test', 'e') AS Poz1_e,
       LOCATE('e', 'sir de test') AS Poz1_e_v2
  FROM SYSIBM.SYSDUMMY1
```

iar în MS SQL Server funcția se numește PATINDEX și folosește „joker”-ul % pe care l-am folosit în paragraful 5.6:

```
SELECT PATINDEX('%e%', 'sir de test')
```

Și dacă tot am atacat particularități ale funcțiilor de căutare în siruri de caractere, nu ar fi corect să nu amintim de funcția Oracle REGEXP_INSTR și „vecinele” sale – REGEXP_LIKE, REGEXP_COUNT, REGEXP_REPLACE și REGEXP_SUBSTR. Acest grup de funcții poate fi, la fel de bine, legat de paragraful 5.6. Oracle nu a implementat predicatul SIMILAR TO dedicat așa-numitelor *regular expressions*, însă își „reperează” onoarea cu cele cinci funcții. Pentru a extrage din tabela PERSOANE linile în care numele începe cu litera I, o variantă ce folosește funcția REGEXP_LIKE:

```
SELECT *
```

```
FROM persoane
WHERE REGEXP_LIKE(Prenume, 'T')
```

Partea cea mai interesantă o constituie multimea operatorilor și metasimbolurilor ce pot fi utilizati pentru construirea măștii de căutare. Spre exemplu, dacă dorim să extragem linile în care prenumele este unul dintre Ion, Ioan sau Ioana, şablonul este:

```
SELECT *
FROM persoane
WHERE REGEXP_LIKE(Prenume, 'I(o|oa)n(|a)$')
```

Simbolurile ^ și \$ marchează începutul și sfârșitul sirului, dar pot marca și „capetele” unei linii dintr-un sir care se întinde pe mai multe rânduri. Bara verticală introduce o structură alternativă, iar parantezele grupează expresii.

Reluăm unul dintre exemplele din paragraful 5.6 (figura 5.33): *Care sunt persoanele care stau la bloc, bloc al cărui nume să fie alcătuit dintr-o literă și o cifră* (ex. H2, B5, A9) ? Varianta Oracle de mai jos este mai bună decât cea din capitolul 5, întrucât SELECT-ul respectiv ar fi extras și blocurile al căror nume era alcătuit din două cifre:

```
SELECT *
FROM persoane
WHERE REGEXP_LIKE(Adresa, 'Bl.\w\d')
```

Caracterele \w substituie o literă, în timp ce \d o cifră. La fel de corectă este varianta în care o literă este substituită prin intervalul [A-Z]⁸, iar o cifră substituită prin intervalul [0-9]:

```
SELECT * FROM persoane WHERE REGEXP_LIKE(Adresa, 'Bl.[A-Z][0-9]')
```

Există foarte multe modalități de construire a expresiilor de căutare. Eu mă opresc aici, sperând că v-am trezit cât de căt curiozitatea ca să încercați ceva și pe cont propriu.

Am văzut în paragraful 5.4 că ordonarea se aplică nu numai la numere și date caleNDARISTICE, ci și la siruri de caractere. Elementul luat în calcul la ordonare este codul ASCII sub care este stocat orice caracter într-un calculator. Comanda următoare afișează (în Oracle) caracterele ce corespund codurilor ASCII cu numerele 40, 41, 49, 50, 65 și 97 (vezi figura 6.16).

```
SELECT 40 AS argument, CHR(40) AS "CHR(argument)" FROM dual
UNION SELECT 41, CHR(41) FROM dual
UNION SELECT 49, CHR(49) FROM dual
```

⁸ Literele din denumirea blocului se scriu cu majuscule întotdeauna.

```
UNION SELECT 50, CHR(50) FROM dual
UNION SELECT 65, CHR(65) FROM dual
UNION SELECT 97, CHR(97) FROM dual
```

ARGUMENT	CHR(argument)
40 (
41)	
49 1	
50 2	
65 A	
97 a	

Figura 6.16. Funcția CHR

Dacă eliminăm *FROM dual*, sintaxa este acceptată de PostgreSQL, iar, înlocuind *FROM dual* cu *FROM sysibm.sysdummy1* (sau cu *FROM sysibm.dual*) interogarea poate fi executată în DB2. În MS SQL Server interogarea este similară PostgreSQL-ului, doar că numele funcției e altul – CHAR.

Funcția inversă, care returnează codul ASCII al unui caracter dat este chiar ASCII. Despre funcții precum TRANSLATE/REPLACE, CONVERT sau OVERLAY și corespondențele lor în diverse dialecte nu ne vom ocupa în această lucrare⁹.

6.3. Funcții pentru date calendaristice

Din SQL:1999 a fost introdusă funcția EXTRACT prin care putem afla componentele unei date/moment calendaristice. Funcția este implementată în Oracle și PostgreSQL, dar nu și în DB2 și MS SQL Server. Cele mai întrebuintăte argumente sunt YEAR, MONTH și DAY. Iată un format simplu al unei interogări ce extrage, pe rând, anul, luna și ziua dintr-o dată calendaristică, interogare executată în Oracle și al cărei rezultat este cel din figura 6.17.

```
SELECT DATE'2007-09-17',
       EXTRACT (YEAR FROM DATE'2007-09-17') AS anul,
       EXTRACT (MONTH FROM DATE'2007-09-17') AS luna,
       EXTRACT (DAY FROM DATE'2007-09-17') AS Ziua
  FROM dual
```

⁹ Pentru detalii, vezi [Melton & Simon 02], [Jones s.a. 05]

	DATE'2007-09-17'	ANUL	LUNA	ZIUA
	17-09-2007	2007	9	17

Figura 6.17. Funcții EXTRACT aplicate asupra unei date calendaristice (Oracle)

EXTRACT poate fi aplicată și asupra unor literali/attribute/variabile de tip date-timp¹⁰ (TIMESTAMP), ca în exemplul următor (vezi figura 6.18):

```
SELECT TIMESTAMP'2007-09-17 18:30:21',
       EXTRACT (DAY FROM TIMESTAMP'2007-09-17 18:30:21') AS Ziua,
       EXTRACT (HOUR FROM TIMESTAMP'2007-09-17 18:30:21') AS Ora,
       EXTRACT (MINUTE FROM TIMESTAMP'2007-09-17 18:30:21') AS Minutele,
       EXTRACT (SECOND FROM TIMESTAMP'2007-09-17 18:30:21') AS Secundele
  FROM dual
```

TIMESTAMP'2007-09-17 18:30:21'	ZIUA	ORA	MINUTELE	SECUNDELE
17-09-2007 18:30:21,000000000	17	18	30	21

Figura 6.18. Funcția EXTRACT aplicată asupra unui literal TIMESTAMP

PostgreSQL-ul este mai generos cu funcția EXTRACT, fiind posibilă aflarea și a altor informații dintr-o dată sau dintr-un moment: secolul (CENTURY), deceniu (DECADE), ziua săptămânii (DOW – Day Of the Week), ziua din an (DOY – Day Of the Year), săptămâna din an (WEEK), trimestrul (QUARTER) etc. Iată asemenea informații pentru momentul *11 noiembrie 2007, ora 23:55:10*:

```
SELECT 'Momentul:' AS Parametru, ''|| TIMESTAMP'2007-11-10 23:55:10' AS Valoare
      UNION
SELECT 'Secol:', ''|| EXTRACT (CENTURY FROM TIMESTAMP'2007-11-10 23:55:10')
      UNION
SELECT 'Deceeniu:', ''|| EXTRACT (DECADE FROM TIMESTAMP'2007-11-10 23:55:10')
      UNION
SELECT 'Zi din sapt.:', ''|| EXTRACT (DOW FROM TIMESTAMP'2007-11-10 23:55:10')
      UNION
SELECT 'Zi din an:', ''|| EXTRACT (DOY FROM TIMESTAMP'2007-11-10 23:55:10')
      UNION
SELECT 'Sapt. din an:', ''|| EXTRACT (WEEK FROM TIMESTAMP'2007-11-10 23:55:10')
      UNION
SELECT 'Trimestru din an:', ''|| EXTRACT (QUARTER FROM
    TIMESTAMP'2007-11-10 23:55:10')
```

¹⁰ Pentru tipul dată-timp (TIMESTAMP) vom folosi, un pic forțat, termenul de tip *moment*.

Pentru a asigura afişarea pe mai multe linii (figura 6.19) am folosit operatorul UNION. Cum, însă, prima linie avea pe a doua coloană o valoare TIMESTAMP, iar celelalte valori NUMERIC, a trebuit să forțăm conversia lor în siruri de caractere prin concatenarea cu un spațiu.

parametru text	valoare text
Momentul:	2007-11-10 23:55:10
Deceniu:	200
Sapt. din an:	45
Secol:	21
Trimestrul din an:	4
Zi din an:	314
Zi din sapt.:	6

Figura 6.19. Generozitate PostgreSQl în materie de funcție EXTRACT

Aflarea altor informații, precum în ce zi a săptămânii cade o anumită dată presupune folosirea unor funcții specifice fiecărui SGBD, cum ar TO_CHAR În Oracle și PostgreSQL, DOW/CDOW în Visual FoxPro, DAYOFWEEK în DB2 etc. PostgreSQL-ul însă este destul de elegant, deoarece ziua săptămânii poate fi aflată tot cu EXTRACT, (DOW...), valoarea returnată fiind 0 pentru duminică, 1 pentru luni, ... 6 pentru sâmbătă. Unde mai punem că PostgreSQL poate obține asemenea informații și printr-o funcție proprietară – DATE_PART:

```
SELECT 'Momentul:' AS Parametru, ''||'TIMESTAMP'2007-11-10 23:55:10' AS Valoare
      UNION
SELECT 'Secol:', ''||'DATE_PART ('CENTURY', TIMESTAMP'2007-11-10 23:55:10')
      UNION
SELECT 'Deceniu:', ''||'DATE_PART ('DECADE', TIMESTAMP'2007-11-10 23:55:10')
      UNION
SELECT 'Zi din sapt.:', ''||'DATE_PART ('DOW', TIMESTAMP'2007-11-10 23:55:10')
      UNION
SELECT 'Zi din an:', ''||'DATE_PART ('DOY', TIMESTAMP'2007-11-10 23:55:10')
      UNION
SELECT 'Sapt. din an:', ''||'DATE_PART ('WEEK', TIMESTAMP'2007-11-10 23:55:10')
      UNION
SELECT 'Trim. din an:', ''||'DATE_PART ('QUARTER', TIMESTAMP'2007-11-10 23:55:10')
```

Interesant este și faptul că, în Oracle, o parte din funcțiile pe care noi le-am încadrat la matematice, TRUNC și ROUND, sunt utile și pentru date calendaristice (vezi figura 6.20):

```
SELECT DATE'2007-09-17',
       TRUNC(DATE'2007-09-17','MONTH') AS trunchiere_data,
       ROUND(DATE'2007-09-17','MONTH') AS rotunjire_data
FROM dual
```

<code>DATE'2007-09-17'</code>	<code>TRUNCIERE_DATA</code>	<code>ROTUNJIRE_DATA</code>
17-09-2007	01-09-2007	01-10-2007

Figura 6.20. Trunchierea și rotunjirea unei date calendaristice la nivel de lună (Oracle)

În PostgreSQL avem două vești rele și două bune. Prima (rea), este că nu putem folosi funcțiile ROUND și TRUNC pentru date calendaristice. A doua (bună) este că disponem de funcția DATE_TRUNC care se comportă similar funcției TRUNC din interogarea anterioară. Din păcate, nu avem și un DATE_ROUND, însă DATE_TRUNC poate folosi argumente inacceptabile în Oracle (ex. WEEK):

```
SELECT DATE'2007-09-19',
       DATE_TRUNC('MONTH', DATE'2007-09-19') AS trunchiere_luna,
       DATE_TRUNC('WEEK', DATE'2007-09-19') AS trunchiere_sapt
```

19 septembrie 2007 a picat într-o miercuri, aşa că trunchierea la nivel de săptămână a returnat data precedentei zile de luni (care marchează începutul săptămânii) – vezi figura 6.21.

date	trunchiere_luna timestamp with time zone	trunchiere_sapt timestamp with time zone
2007-09-19	2007-09-01 00:00:00+03	2007-09-17 00:00:00+03

Figura 6.21. Trunchieri ale unei date calendaristice la nivel de lună și săptămână (PostgreSQL)

În MS SQL Server veștile sunt la fel de amestecate. Cea mai rea este că nu există funcția EXTRACT, și nu e prima dată când luăm cunoștință de distanța produsului față de standardele SQL. Totodată, însă, cea mai mare parte a informațiilor aflate prin funcția EXTRACT poate fi obținută cu ajutorul a câtorva funcții proprietare. Conținutul din figura 6.17 se obține prin funcțiile YEAR, MONTH și DAY:

```
SELECT '2007-09-17',
       YEAR('2007-09-17') AS anul,
       MONTH ('2007-09-17') AS luna,
       DAY ('2007-09-17') AS Ziua
```

dar și cu mai generala funcție DATEPART care seamănă binișor cu „surata” sa din PostgreSQL:

```
SELECT '2007-09-17',
       DATEPART(YEAR, '2007-09-17') AS anul,
       DATEPART(MONTH, '2007-09-17') AS luna,
       DATEPART(DAY, '2007-09-17') AS Ziua
```

Tot funcția DATEPART ne garantează și informațiile obținute (în Oracle) în figura 6.18:

```
SELECT '2007-09-17',
       DATEPART(DAY, '2007-09-17 18:30:21') AS Ziua,
       DATEPART(HH, '2007-09-17 18:30:21') AS Ora,
       DATEPART(MI, '2007-09-17 18:30:21') AS Minutele,
       DATEPART(SS, '2007-09-17 18:30:21') AS Secundele
```

Alte informații extrase dintr-o dată calendaristică cu ajutorul funcției DATEPART trebuie să folosească argumentele:

- QUARTER (sau QQ sau Q) pentru trimestru;
- DAYOFYEAR (sau DY sau Y) pentru ziua din an;
- WEEK (sau WK sau WW) pentru săptămână;
- WEEKDAY (sau DW) pentru ziua din săptămână;
- DAYOFYEAR (sau DY sau Y) pentru ziua din an.

Pentru a vedea cu ce ziua începe săptămâna (prima zi a săptămânii) se poate folosi:

```
SELECT @@DATEFIRST
```

Fiind returnat un număr cuprins între 1 (pentru luni) și 7 (pentru duminică). Prima zi a săptămânii poate fi modificată prin comanda SET DATEFIRST, dar lucrul acesta nu prea ne interesează acum.

În DB2 calificarea unei constante de tip dată calendaristică s-a realizat până acum prin încadrarea datei (în format ISO) între apostrofuri. Același lucru se poate realiza și prin funcția DATE:

```
SELECT '2007-09-17' AS O_Data, DATE('2007-09-17') AS Aceeasi_Data
FROM sysibm.sysdummy1
```

Ca și în SQL Server, în DB2 există funcții speciale pentru extragerea zilei, lunii, anului dintr-o dată calendaristică și, pe lângă acestea, a orei, minutelor și secundelor dintr-un moment:

```
SELECT CURRENT_TIMESTAMP,
       YEAR(CURRENT_TIMESTAMP) AS anul,
       MONTH (CURRENT_TIMESTAMP) AS luna,
       DAY (CURRENT_TIMESTAMP) AS Ziua,
       HOUR (CURRENT_TIMESTAMP) AS Ora,
       MINUTE(CURRENT_TIMESTAMP) AS Min,
       SECOND (CURRENT_TIMESTAMP) AS Sec
FROM sysibm.sysdummy1
```

La acestea se mai adaugă:

- DAYNAME - pentru numele zilei din săptămână;
- DAYOFWEEK - pentru numărul zilei din săptămână (1-duminică, ... 7-sâmbătă) sau DAYOFWEEK_ISO pentru numărul-standard (ISO) al zilei din săptămână (1-luni, ..., 7-duminică);
- DAYOFYEAR - pentru ziua din an (numărul său);
- WEEK pentru săptămână (numărul său în cadrul anului), în condițiile în care prima zi a săptămânii este considerată duminică; WEEK_ISO funcționează identic, doar că prima zi a săptămânii este luna;
- MONTHNAME - pentru denumirea lunii;
- QUARTER - pentru trimestru;
- MICROSECONDS - pentru microsecundele dintr-un moment (TIMESTAMP).

6.4. Funcții pentru intervale

În paragraful 5.3, pentru a exemplifica câteva expresii cu date calendaristice, am folosit tipuri de date INTERVAL dedicate, după le spune și numele, exprimării intervalelor calendaristice. Mai toate SGBD-urile au însă funcții cu vechi state de serviciu în acest sens. Astfel, pentru a specifica intervale calendaristice de ordinul lunilor, în Oracle există funcția ADD_MONTHS, în Visual FoxPro GOMONTH, în SQL Server DATEADD etc.

În Oracle comanda:

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

este echivalentă cu:

```
SELECT NrFact AS Factura, DataFact AS Data_Facturare,
       ADD_MONTHS(DataFact,2) AS Scadenta_Incasare
  FROM facturi
```

În MS SQL Server același rezultat se obține folosind funcția DATEADD astfel:

```
SELECT NrFact AS Factura, DataFact AS Data_Facturare,
       DATEADD(MONTH, 2, DataFact) AS Scadenta_Incasare
  FROM facturi
```

Cu DATEADD suntem în stare să lucrăm cu orice gen de intervale pe care le-am ocolit în SQL Server până acum. Revenim la o problemă formulată în paragraful 5.3 - dorim să afișăm pentru fiecare factură ce dată va fi peste 1 an, două luni și 25 de zile de la momentul emiterii.

```
SELECT NrFact AS Factura, DataFact AS Data_Facturare,
       DATEADD(MONTH, 14, DataFact) + 25
  AS O_Data_Viitoare
  FROM facturi
```

Pentru aflarea numărului de luni dintre două date calendaristice, înainte de apariția tipurilor de date INTERVAL, în Oracle se putea folosi doar funcția MONTHS_BETWEEN. Reluăm una dintre interogările din paragraful 5.3 în a cărei clauză SELECT adăugăm un atribut calculat pe baza noii funcții:

```
SELECT NrFact, DATE'2007-11-22', DataFact,
       DATE'2007-11-22' - DataFact AS Zile_Scurse,
       (DATE'2007-11-22' - DataFact) YEAR TO MONTH
       AS AniLuni_Sc1,
       MONTHS_BETWEEN(DATE'2007-11-22', DataFact)
       AS Luni_Scurse
  FROM facturi
```

După cum se observă în figura 6.22, în lipsa unei funcții de rotunjire sau trunchiere, MONTHS_BETWEEN este din cale-afară de scrupuloasă cu zecimalele.

NRFACT	DATE'2007-11-22'	DATAFACT	ZILE_SCURSE	ANILUNI_SC1	LUNI_SCURSE
1111	22-11-2007	01-08-2007	113 0-4	3,67741935483870967741935483870967741935	
1112	22-11-2007	01-08-2007	113 0-4	3,67741935483870967741935483870967741935	
1113	22-11-2007	01-08-2007	113 0-4	3,67741935483870967741935483870967741935	
1114	22-11-2007	01-08-2007	113 0-4	3,67741935483870967741935483870967741935	
1115	22-11-2007	02-08-2007	112 0-4	3,64516129032258064516129032258064516129	
1116	22-11-2007	02-08-2007	112 0-4	3,64516129032258064516129032258064516129	
1117	22-11-2007	03-08-2007	111 0-4	3,61290322580645161290322580645161290323	
1118	22-11-2007	04-08-2007	110 0-4	3,58064516129032258064516129032258064516	
1119	22-11-2007	07-08-2007	107 0-4	3,48387096774193548387096774193548387097	

Figura 6.22. Funcția Oracle MONTHS_BETWEEN

Spre deliciul utilizatorilor (adică al nostru), funcția EXTRACT poate fi aplicată nu numai asupra unei date, ci și asupra unui interval de timp. Astfel, pentru a calcula numărul de luni dintre două date putem recurge la soluții precum cele din interogarea:

```
SELECT NrFact,
       DATE'2008-11-22', DataFact,
       DATE'2008-11-22' - DataFact AS Zile_Scurse,
       (DATE'2008-11-22' - DataFact) YEAR TO MONTH AS AnLuni_Sc,
       EXTRACT (YEAR FROM (DATE'2008-11-22' - DataFact) YEAR TO MONTH
              ) * 12 +
       EXTRACT (MONTH FROM
              (DATE'2008-11-22' - DataFact) YEAR TO MONTH
              ) AS Luni_Sc1,
       TRUNC(MONTHS_BETWEEN (DATE'2008-11-22', DataFact),2) AS Luni_Sc2
FROM facturi
```

O parte din liniile rezultatului este prezentată în figura 6.23.

PostgreSQL-ul nu atât de generos la exprimarea directă a diferenței a două date calendaristice în ani, luni sau alte tipuri de date INTERVAL. Sprijinul vine însă de la o funcție proprietară – AGE care, după cum îi spune numele, calculează intervalul scurs de la o anumită dată până la momentul curent (atunci când funcția are un singur parametru) sau intervalul scurs între două date/momente (atunci când funcția are doi parametri). Mai mult, funcția AGE poate fi argument al funcției EXTRACT, aşa încât libertatea de mișcare este aproape totală în materie de intervale. Pentru exemplificare, dorim să exprimăm intervalul scurs între data emiterii fiecărei dintre facturi și 22 noiembrie 2007 în zile, în ani-luni-zile, numai în ani și numai în luni:

```
SELECT NrFact, DATE'2008-11-22', DataFact,
       DATE'2008-11-22' - DataFact AS Zile_Scurse,
       AGE(DATE'2008-11-22',DataFact) AS Interval_Scurs,
       EXTRACT (YEAR FROM AGE(DATE'2008-11-22',DataFact)) AS Interval_Ani,
       EXTRACT (YEAR FROM AGE(DATE'2008-11-22',DataFact)) * 12 +
       EXTRACT (MONTH FROM AGE(DATE'2008-11-22',DataFact)) AS Interval_Luni
FROM facturi
```

NRFAC	DATE'2008-11-22'	DATAFACT	ZILE_SCURSE	ANILUNI_SC	LUNI_SC1	LUNI_SC2
1111	22-11-2008	01-08-2007	479	1-4	16	15,67
1112	22-11-2008	01-08-2007	479	1-4	16	15,67
1113	22-11-2008	01-08-2007	479	1-4	16	15,67
1114	22-11-2008	01-08-2007	479	1-4	16	15,67
1115	22-11-2008	02-08-2007	478	1-4	16	15,64
1116	22-11-2008	02-08-2007	478	1-4	16	15,64
1117	22-11-2008	03-08-2007	477	1-4	16	15,61
1118	22-11-2008	04-08-2007	476	1-4	16	15,58
1119	22-11-2008	07-08-2007	473	1-4	16	15,48
1120	22-11-2008	07-08-2007	473	1-4	16	15,48
1121	22-11-2008	07-08-2007	473	1-4	16	15,48
1122	22-11-2008	07-08-2007	473	1-4	16	15,48
2111	22-11-2008	14-08-2007	466	1-3	15	15,25
2112	22-11-2008	14-08-2007	466	1-3	15	15,25
2113	22-11-2008	14-08-2007	466	1-3	15	15,25
2115	22-11-2008	15-08-2007	465	1-3	15	15,22
2116	22-11-2008	15-08-2007	465	1-3	15	15,22

Figura 6.23. Funcții (Oracle) EXTRACT, TRUNC și MONTHS_BETWEEN

Rezultatul este cel din figura 6.24. Pentru calculul numărului de luni an înmulțit numărul anilor cu 12 la care am adăugat numărul lunilor extras cu EXTRACT (MONTH...).

nrfact numeric(8,0)	date date	datafact date	zile_scurse integer	interval_surs interval	interval_ani double precis	interval_luni double precis
1111	2008-11-22	2007-08-01	479	1 year 3 mons 21 days	1	15
1112	2008-11-22	2007-08-01	479	1 year 3 mons 21 days	1	15
1113	2008-11-22	2007-08-01	479	1 year 3 mons 21 days	1	15
1114	2008-11-22	2007-08-01	479	1 year 3 mons 21 days	1	15
1115	2008-11-22	2007-08-02	478	1 year 3 mons 20 days	1	15
1116	2008-11-22	2007-08-02	478	1 year 3 mons 20 days	1	15
1117	2008-11-22	2007-08-03	477	1 year 3 mons 19 days	1	15
1118	2008-11-22	2007-08-04	476	1 year 3 mons 18 days	1	15
1119	2008-11-22	2007-08-07	473	1 year 3 mons 15 days	1	15
1120	2008-11-22	2007-08-07	473	1 year 3 mons 15 days	1	15
1121	2008-11-22	2007-08-07	473	1 year 3 mons 15 days	1	15
1122	2008-11-22	2007-08-07	473	1 year 3 mons 15 days	1	15
2111	2008-11-22	2007-08-14	466	1 year 3 mons 8 days	1	15
2112	2008-11-22	2007-08-14	466	1 year 3 mons 8 days	1	15
2113	2008-11-22	2007-08-14	466	1 year 3 mons 8 days	1	15

Figura 6.24. Funcții (PostgreSQL) AGE și EXTRACT

Și SQL Server are tot o funcție proprietară pentru calculul intervalului dintre două date calendaristice – DATEDIFF.

```
SELECT NrFact, '2007-11-22', DataFact,
       DATEDIFF(YEAR, DataFact, '2007-11-22') AS Ani_Scursi,
       DATEDIFF(YEAR, DataFact, '2007-11-22') * 12 +
          DATEDIFF(MONTH, DataFact, '2007-11-22') AS Luni_Scurse,
       DATEDIFF(DAY, DataFact, '2007-11-22') AS Zile_Scurse
```

FROM facturi

Rezultatul este cel din figura 6.25.

NrFact	(No column name)	DataFact	Ani_Scursi	Luni_Scurse	Zile_Scurse
1111	2007-11-22	2007-08-01 00:00:00	0	3	113
1112	2007-11-22	2007-08-01 00:00:00	0	3	113
1113	2007-11-22	2007-08-01 00:00:00	0	3	113
1114	2007-11-22	2007-08-01 00:00:00	0	3	113
1115	2007-11-22	2007-08-02 00:00:00	0	3	112
1116	2007-11-22	2007-08-02 00:00:00	0	3	112
1117	2007-11-22	2007-08-03 00:00:00	0	3	111
1118	2007-11-22	2007-08-04 00:00:00	0	3	110
1119	2007-11-22	2007-08-07 00:00:00	0	3	107
1120	2007-11-22	2007-08-07 00:00:00	0	3	107

Figura 6.25. Funcția MS SQL Server DATEDIFF

În DB2, după cum am văzut în paragraful 5.3, la o dată calendaristică poate fi adunat un interval de tip an, lună, zi etc. fără a folosi explicit clauza INTERVAL. Tot acolo am văzut că diferența dintre două calendaristice este un număr care conține, de la dreapta la stânga, numărul de zile, de luni și de ani. Pentru calculul numărului de zile dintre două date se poate folosi o altă funcție – DAYS. Această funcție returnează numărul de zile corespunzător unei date calendaristice. Prin scăderea a două funcții DAYS obținem tocmai intervalul în zile dintre cele două date:

```
SELECT NrFact, '2007-11-22' AS Data_Reper, DataFact,
       DataFact - '2007-11-22' AS Ani_Luni_Zile_Scurse,
       DAYS('2007-11-22') - DAYS(DataFact) AS Zile_Scurse
```

FROM facturi

Ultimele două coloane ale rezultatului din figura 6.26 sunt edificatoare.

NRFAC	DATA_REPER	DATAFACT	ANI_LUNI_ZILE_SCURSE	ZILE_SCURSE
1.111	2007-11-22	01.08.2007	-321	113
1.112	2007-11-22	01.08.2007	-321	113
1.113	2007-11-22	01.08.2007	-321	113
1.114	2007-11-22	01.08.2007	-321	113
1.115	2007-11-22	02.08.2007	-320	112
1.116	2007-11-22	02.08.2007	-320	112
1.117	2007-11-22	03.08.2007	-319	111
1.118	2007-11-22	04.08.2007	-318	110
1.119	2007-11-22	07.08.2007	-315	107
1.120	2007-11-22	07.08.2007	-315	107
1.121	2007-11-22	07.08.2007	-315	107
1.122	2007-11-22	07.08.2007	-315	107
2.111	2007-11-22	14.08.2007	-308	100
2.112	2007-11-22	14.08.2007	-308	100

Figura 6.26. Două diferențe între două date calendaristice în DB2

Cea mai spumoasă funcție DB2 dedicată extragerii unor informații dintr-un interval cuprins între două momente este TIMESTAMPDIFF, care are două

argumente, dintre care primul semnalizează tipul intervalului, și poate avea valorile:

- 1 pentru fracțiuni de secundă;
- 2 pentru secunde;
- 4 pentru minute;
- 8 pentru ore;
- 16 pentru zile;
- 32 pentru săptămâni;
- 64 pentru luni;
- 128 pentru trimestre și
- 256 pentru ani,

iar al doilea este chiar diferența dintre cele două momente, diferență ce trebuie transformată în sir de caractere (funcția CHAR):

```
SELECT TIMESTAMP('2008-03-27 14:12:55') AS Moment_Start,
       TIMESTAMP('2006-12-10 23:44:32') AS Moment_Stop ,
       TIMESTAMPDIFF(16, CHAR(
           TIMESTAMP('2008-03-27 14:12:55') - TIMESTAMP('2006-12-10 23:44:32'))
       ) AS Zile_Scurse,
       TIMESTAMPDIFF(32, CHAR(
           TIMESTAMP('2008-03-27 14:12:55') - TIMESTAMP('2006-12-10 23:44:32'))
       ) AS Sapt_Scurse,
       TIMESTAMPDIFF(64, CHAR(
           TIMESTAMP('2008-03-27 14:12:55') - TIMESTAMP('2006-12-10 23:44:32'))
       ) AS Luni_Scurse
FROM sysibm.sysdummy1
```

Diferența dintre cele două momente este exprimată, pe rând, în zile, săptămâni și ani – vezi figura 6.27.

MOMENT_START	MOMENT_STOP	ZILE_SCURSE	SAPT_SCURSE	LUNI_SCURSE
27.03.2008 14:12:55 000000	10.12.2006 23:44:32 000000	471	67	15

Figura 6.27. Diferența (exprimată, pe rând, în zile, săptămâni și luni) dintre două momente în DB2

6.5. Conversii între tipuri de date

Funcția de conversie prezentă în mai toate limbajele de programare cunoscute este CAST pe care am întrebuiat-o deja în paragraful 5.3 când precizam că în standardele SQL și unele SGBD-uri (ex. IBM DB2, SQL Server) concatenarea unor siruri de caractere cu numere sau date calendaristice reclamă conversia numerelor/datelor și intervalelor calendaristice în siruri. Afisările legate de scadența fiecărei facturi emisă în luna septembrie 2007, text care să folosească funcțiile CAST, TRIM,..., plus EXTRACT în clauza WHERE, precum și o expresie dată calendaristică + interval.

```
SELECT 'Scadenta facturii ' || TRIM( TRAILING FROM
```

```

        CAST (NrFact AS CHAR(8))
        ) || ' (trimisa clientului ' || dencl ||
') are scadenta pe ' || CAST (
        (DataFact + INTERVAL '14' DAY)
        AS CHAR(10)
        )
AS Scadente_facturi_Sept2007
FROM facturi f INNER JOIN clienti c ON f.CodCl=c.CodCl
WHERE EXTRACT (YEAR FROM datafact) = 2007 AND
EXTRACT (MONTH FROM datafact)=9

```

A sosit momentul să recunoaștem că fără funcția CAST (sau CONVERT) în MS SQL Server nu putem obține rezultatul din figura 6.19. Fiecare SELECT „reunit” din interogarea următoare funcționează izolat:

```

SELECT 'Momentul:' AS Parametru, '2007-11-10 23:55:10' AS Valoare
UNION
SELECT 'Zi din sapt.:', DATEPART (WEEKDAY, '2007-11-10 23:55:10')
UNION
SELECT 'Zi din an:', DATEPART (DAYOFYEAR, '2007-11-10 23:55:10')
UNION
SELECT 'Sapt. din an:', DATEPART (WEEK, '2007-11-10 23:55:10')
UNION
SELECT 'Trim. din an:', DATEPART (QUARTER, '2007-11-10 23:55:10')

```

În schimb când lansăm întreaga interogare de mai sus obținem un mesaj de eroare – vezi figura 6.28 – din cauza faptului că trucul concatenării cu un spațiu nu transformă automat a doua coloană într-un sir de caractere.

```

SELECT 'Momentul:' AS Parametru, CAST ('2007-11-10 23:55:10' AS VARCHAR) AS Valoare
UNION
SELECT 'Zi din sapt.:', DATEPART (WEEKDAY, '2007-11-10 23:55:10')
UNION
SELECT 'Zi din an:', DATEPART (DAYOFYEAR, '2007-11-10 23:55:10')
UNION
SELECT 'Sapt. din an:', DATEPART (WEEK, '2007-11-10 23:55:10')
UNION
SELECT 'Trim. din an:', DATEPART (QUARTER, '2007-11-10 23:55:10')

```

Results Messages

Msg 245, Level 16, State 1, Line 2
Conversion failed when converting the varchar value '2007-11-10 23:55:10' to data type int.

Figura 6.28. O mică problemă la reuniune în MS SQL Server

Cu ajutorul funcției CAST lucrurile se rezolvă destul de elegant:

```

SELECT 'Momentul:' AS Parametru,
CAST ('2007-11-10 23:55:10' AS VARCHAR) AS Valoare
UNION
SELECT 'Zi din sapt.:',
CAST (DATEPART (WEEKDAY, '2007-11-10 23:55:10') AS VARCHAR)

```

```

        UNION
SELECT 'Zi din an:',  

       CAST (DATEPART (DAYOFYEAR, '2007-11-10 23:55:10') AS VARCHAR)
        UNION
SELECT 'Sapt. din an:', CAST (DATEPART (WEEK, '2007-11-10 23:55:10') AS VARCHAR)
        UNION
SELECT 'Trim. din an:',  

       CAST (DATEPART (QUARTER, '2007-11-10 23:55:10') AS VARCHAR)

```

Dacă ar fi să rescriem interogarea de mai sus cu, exclusiv, funcțiile proprietare SQL Server, am putea apela la: CONVERT (pe post de CAST) și STR (pentru conversia unui număr în sir de caractere). Funcțiile DATEPART, DAY, MONTH și YEAR returnează numere. Pentru afișarea numelui unei componente dintr-o dată calendaristică, în loc de DATEPART se întrebunează DATENAME. Noua variantă (cu adăugiri) a interogării (cu rezultatul în figura 6.29) este:

```

SELECT 'Momentul:' AS Parametru,  

       CONVERT (VARCHAR, '2007-11-10 23:55:10') AS Valoare
        UNION
SELECT 'Zi din sapt.:', STR (DATEPART (WEEKDAY, '2007-11-10 23:55:10'))
        UNION
SELECT 'Zi din an:', STR (DATEPART (DAYOFYEAR, '2007-11-10 23:55:10'))
        UNION
SELECT 'Sapt. din an:', STR (DATEPART (WEEK, '2007-11-10 23:55:10'))
        UNION
SELECT 'Trim. din an:', STR (DATEPART (QUARTER, '2007-11-10 23:55:10'))
        UNION
SELECT 'Numele zilei:', DATENAME (WEEKDAY, '2007-11-10 23:55:10')

```

Parametru	Valoare
Momentul:	2007-11-10 23:55:10
Numele zilei:	Saturday
Sapt. din an:	45
Trim. din an:	4
Zi din an:	314
Zi din sapt.:	7

Figura 6.29. Câteva funcții de conversie a componentelor unei date calendaristice în SQL Server

Una dintre cele mai puternice funcții de conversie din Oracle și PostgreSQL este TO_CHAR, la care se adaugă TO_DATE și TO_NUMBER. Spun puternică, pentru că, știind tipologia caracterelor folosite în şablonul acestor funcții, pot fi obținute informații diverse. Să luăm câteva exemple. Înainte de implementarea opțiunii DATE'YYYY-MM-DD', și în Oracle și în PostgreSQL cel mai sigur mod de specificare a unui literal de tip dată calendaristică presupunea folosirea funcției TO_DATE. Clauza WHERE din comanda SELECT anterioară se putea scrie:

```
SELECT ...
```

```

FROM ...
WHERE DataFact BETWEEN TO_DATE('01/09/2007', 'DD/MM/YYYY')
AND TO_DATE('30/09/2007', 'DD/MM/YYYY')

```

Ordinea componentelor unei date calendaristice este indicată de şablon, utilizatorul având o mare libertate de mişcare. Preţul plătit este lungimea, şablonul fiind considerabil mai lung prin comparaţie cu *DATE'yyyy-mm-dd'*. Cele mai de efect opţiuni pot fi gustate, însă, la operaţiunea inversă, de conversie din dată calendaristică în sir de caractere - **TO_CHAR** - conversie care coincide cu extragerea unor informaţii utile din data/momentul argument. Astfel, similar figurii 6.17 (paragraful 6.3) se poate obţine rezultatul (Oracle) din figura 6.30 a interogării următoare:

```

SELECT DATE'2007-09-17',
       TO_CHAR(DATE'2007-09-17', 'yyyy') AS anul,
       TO_CHAR(DATE'2007-09-17', 'mm') AS luna,
       TO_CHAR(DATE'2007-09-17', 'dd') AS ziua
  FROM dual

```

	DATE'2007-09-17'	ANUL	LUNA	ZIUA
	17-09-2007	2007	09	17

Figura 6.30. Test: identificaţi trei diferenţe faţă de figura 6.17

Cei cu dioptrii foarte bune vor remarcă faptul că anul, luna și ziua sunt aliniate la stânga în figura 6.30, în timp ce în figura 6.17 erau aliniate la dreapta. Aceasta deoarece rezultatul unei funcţii **TO_CHAR** este un sir de caractere, deci va fi aliniat la stânga, în timp ce rezultatele funcţiei **EXTRACT** erau numere. Pentru a „transforma” anul, luna și ziua în numere încadrăm funcţia **TO_CHAR** într-o funcţie **TO_NUMBER**:

```

SELECT DATE'2007-09-17',
       TO_NUMBER(TO_CHAR(DATE'2007-09-17', 'yyyy')) AS anul,
       TO_NUMBER(TO_CHAR(DATE'2007-09-17', 'mm')) AS luna,
       TO_NUMBER(TO_CHAR(DATE'2007-09-17', 'dd')) AS ziua
  FROM dual

```

Atunci când, la conversia în număr, dorim un anumit număr de cifre pe care să se facă reprezentarea, funcţia **TO_NUMBER** poate fi folosită împreună cu un şablon în care să se indice număr poziţiilor pentru partea întreagă și pentru partea fracţională.

Haideţi să luăm în discuţie o problemă „resimtă” în prima ediţie a cărţii, şi încă actuală în *PostgreSQL pgAdmin* şi *MS SQL Server Management Studio* la afişarea rezultatelor în interogări în care intervin calcule aritmetice (de exemplu, pentru determinarea valorii unei linii dintr-o factură sau a unei facturi, sau la calculul TVA). Modul în care se prezintă coloanele calculate nu e deloc entuziasmant. Este drept, e o problemă de afişare, deci de interfaţă/utilitar de conectarea la bază, nu de SQL ! Să luăm un exemplu în care trebuie să determinăm, pentru fiecare linie a facturii cu numărul 1111, valoarea fără TVA şi „TVA-ul” liniei. Acestea presupun jonctionarea LINIIFACT cu PRODUSE, întrucât procentul care se aplică la vânzarea unui produs se află în tabela PRODUSE (atributul ProcTVA). Interogarea:

```

SELECT lf.*, Cantitate * PretUnit AS ValFaraTVA,
       Cantitate * PretUnit * ProcTVA AS TVA_linie
  FROM liniifact lf INNER JOIN produse p ON lf.CodPr=p.CodPr
 WHERE NrFact = 1111

```

executată în Oracle SQL*Plus (sau SQL Developer) obține un rezultat care doar cu indulgență poate fi catalogat drept tabelar (figura 6.31) aceasta din pricina pozitiei fracționare diferite de la linie la linie, ceea ce strică alinierea coloanei TVA_Linie.

NRFACT	LINIE	CODPR	CANTITATE	PRETUNIT	VALFARATVA	TVA_LINIE
1111	1	1	50	1000	50000	9500
1111	2	2	75	1050	78750	7087,5
1111	3	5	500	7060	3530000	670700

Figura 6.31. Modul de afișare a coloanelor numerice în Oracle SQL*Plus

Ei bine, pentru a mai salva din stilul de afișare (și, implicit, aparențele) am folosit în multe figuri¹¹ funcția TO_CHAR, în genul:

```

SELECT lf.*,
       TO_CHAR(Cantitate * PretUnit, '9999999.99') AS ValFaraTVA,
       TO_CHAR(Cantitate * PretUnit * ProcTVA, '9999999.99') AS TVA_linie
  FROM liniifact lf INNER JOIN produse p ON lf.CodPr =p.CodPr
 WHERE NrFact = 1111

```

Precizând șapte cifre la partea întreagă (de șapte ori cifra 9) și două pozitii fracționare (cei doi de 9 care urmează punctului), am forțat, odată cu conversia, alinierea sub formă tabelară – vezi figura 6.32. Din păcate, artificiul funcționează numai în Oracle SQL*Plus, pentru că interfața este în mod text, fiecare caracter având aceeași lungime, în timp ce interfața Oracle SQL Developer este grafică, spațiul rezervat reprezentării unui caracter fiind diferit, în funcție de formă (un I este mult mai îngust decât un M).

¹¹ În prima ediție a cărții interogările Oracle au fost executate în SQL*Plus

NRFACT	LINIE	CODPR	CANTITATE	PRETUNIT	VALFARATUA	TUA_LINIE
1111	1	1	50	1000	50000.00	9500.00
1111	2	2	75	1050	78750.00	7087.50
1111	3	5	500	7060	3530000.00	670700.00

Figura 6.32. Ameliorare modului de afișare a coloanelor numerice în Oracle SQL*Plus cu ajutorul funcției TO_CHAR

Dacă în şablonanele funcțiilor TO_DATE/TO_CHAR dedicate datelor calendaristice foloseam DD pentru zile, MM pentru luni și YYYY pentru an, pentru ore se poate folosi fie HH, fie HH24 (HH afișează orele pe formatul 0-12AM, apoi 0-12PM, în timp ce HH24 permite formatul 0-24), pentru minute MI, iar pentru secunde SS. Spre exemplu, următoarea interogare a fost pregătită pentru lansarea în execuție pe 22 noiembrie 2007 pe la 18:55 (e seară și vremea-i mohorită, numai bună pentru sinucigași):

```
SELECT TO_CHAR(CURRENT_TIMESTAMP,
    'DD-MM-YYYY HH24:MI:SS.SSSSS') AS Data_Ora_Foarte_Exacte
FROM dual
```

Până la lansarea în execuție a interogării a trecut aproape un minut, aşa că rezultatul este cel din figura 6.33.

DATA_ORA_FOARTE_EXACTE
22-11-2007 18:56:01.68161

Figura 6.33. Afișarea tuturor componentelor datei/orei curente (la ora execuției)

După cum știți, de mai bine de zece ani a fost introdus în România sistemul de identificare al persoanelor bazat pe codul numeric personal (CNP-ul). CNP-ul este o „cheie inteligentă”, în sensul că înglobează câteva informații despre „posesor”:

- prima cifră indică sexul: dacă este 1 sau 5 este vorba un băiat/bărbat; dacă este 2 sau 6 este o fată/femeie (mai sunt și alte situații, dar nu le tratăm);
- următoarele șase cifre indică data nașterii (două pentru an, două pentru lună și două pentru zi; dacă persoana este născută înainte de 2000, atunci „cifra sexului” este 1 sau 2; pentru cei născuți după 2000, „cifra sexului” este 5 sau 6;
- etc.

Pentru a afla data nașterii pacienților născuți înainte de anul 2000, formulăm o interogare în care extragem cifrele de la 2 la 7 inclusiv. După cum se observă în

figura 6.34, la extragere secolul implicit este cel actual, astfel încât persoanele par sănscute în viitor.

```
SELECT NumePacient, cnp,
       TO_DATE(SUBSTR(cnp, 2, 6), 'YYMMDD') AS Data_Nasterii_CNP
  FROM pacienti
 WHERE SUBSTR(cnp, 1, 1) IN ('1','2')
```

NUMEPACIENT	CNP	DATA_NASTERII_CNP
Stroe Mihaela	2601228390834	28-12-2060
Buzatu Corneliu	165012370514	12-05-2065
Bagdasar Adela	2601002250611	02-10-2060
Popescu Maria-Mirabela	2721231300888	31-12-2072
Cazan Ana Maria	2690202200345	02-02-2069

Figura 6.34. Tentativă de extragere a datei din CNP pentru cei sănscuți în secolul trecut

Soluția pare simplă: scădem 100 de ani din data obținută mai sus:

```
SELECT NumePacient, cnp,
       TO_DATE(SUBSTR(cnp, 2, 6), 'YYMMDD') -
       INTERVAL '100' YEAR AS Data_Nasterii_CNP
  FROM pacienti
 WHERE SUBSTR(cnp, 1, 1) IN ('1','2')
```

Paradoxal, însă, Oracle tratează această variantă cu oarecare răceală, dacă este să ne luăm după mesajul din figura 6.35.

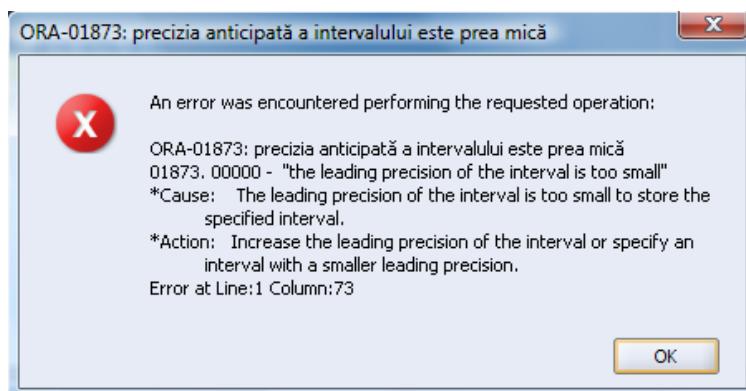


Figura 6.35. Surprize, surprize... în Oracle (episodul 2)

Nedumerirea este cu atât mai mare cu cât, dacă în locul INTERVAL, folosim funcția „proprietară” Oracle ADD_MONTHS cu argumentul -1200 (de luni), lucrurile sunt în regulă – vezi figura 6.36:

```
SELECT NumePacient, cnp,
       ADD_MONTHS(TO_DATE(SUBSTR(cnp, 2, 6), 'YYMMDD'),
                  -1200)
          AS Data_Nasterii_CNP
  FROM pacienti
 WHERE SUBSTR(cnp, 1, 1) IN ('1','2')
```

NUMEPACIENT	CNP	DATA_NASTERII_CNP
Stroe Mihaela	2601228390834	28-12-1960
Buzatu Corneliu	1650512370514	12-05-1965
Bagdasar Adela	2601002250611	02-10-1960
Popescu Maria-Mirabela	2721231300888	31-12-1972
Cazan Ana Maria	2690202200345	02-02-1969

Figura 6.36. Extragerea corectă a datei din CNP pentru cei născuți în secolul trecut

Iată o interogare în care folosim și alte formate de extragere și afișare a lunii dintr-o dată calendaristică:

```
SELECT NrFact, DataFact,
       TO_CHAR(DataFact, 'MM') AS Luna_Nr,
       TO_CHAR(DataFact, 'MONTH') AS Luna_Sir_Intreaga_Majuscule,
       TO_CHAR(DataFact, 'month') AS Luna_Sir_Intreaga_Minuscule,
       TO_CHAR(DataFact, 'MON') AS Luna_Sir_Prescurtat
  FROM facturi
 WHERE NrFact = 1111
```

Clauzele *MONTH* și *month* determină afișarea (vezi figura 6.37) numelui întreg al lunii (în engleză sau română, depinde de modul în care a fost instalat sistemul de operare, serverul Oracle și SQL Developerul), în timp ce *MON* (sau *mon*) afișează numele prescurtat, din trei litere, al lunii.

NRFAC	DATAFACT	LUNA_NR	LUNA_SIR_INTREAGA_MAJUSCULE	LUNA_SIR_INTREAGA_MINUSCULE	LUNA_SIR_PRESCURTAT
1111 01-08-2007	08	AUGUST	august	august	AUG

Figura 6.37. Modalități de vizualizare a unei luni dintr-o dată calendaristică

Următoarea interogare afișează, oarecum în premieră, numele zilei din săptămână în care pică o anumită dată calendaristică, folosindu-se drept şablon *DAY* – vezi figura 6.38:

```
SELECT NrFact, DataFact,
       TO_CHAR(DataFact, 'DD') AS Zi_Din_Luna,
       TO_CHAR(DataFact, 'DAY') AS Zi_Sapt_char
  FROM facturi
 WHERE TO_CHAR(DataFact, 'MM-YYYY') = '09-2007'
```

În plus, am simplificat mult și predicatul de extragere numai a facturilor din luna septembrie 2007.

NRFAC	DATAFACT	ZI_DIN_LUNA	ZI_SAPT_CHAR
3111 01-09-2007	01	SÂMBĂTĂ	
3112 01-09-2007	01	SÂMBĂTĂ	
3113 02-09-2007	02	DUMINICĂ	
3115 02-09-2007	02	DUMINICĂ	
3116 10-09-2007	10	LUNI	
3117 10-09-2007	10	LUNI	
3118 17-09-2007	17	LUNI	

Figura 6.38. Vizualizarea zilei din săptămână

După cum spuneam în paragraful 6.3, PostgreSQL-ul are un argument pentru funcția EXTRACT care permite aflarea zilei din săptămână, dar numai sub formă numerică (de la 0 pentru duminică până la 6 pentru sâmbătă). Pentru ilustrare, adăugăm în clauza SELECT a interogării anterioare o coloană în care folosim această opțiune (rezultatul este în figura 6.39):

```
SELECT NrFact, DataFact,
       TO_CHAR(DataFact, 'DD') AS Zi_Din_Luna,
       TO_CHAR(DataFact, 'DAY') AS Zi_Sapt_char,
       EXTRACT (DOW FROM DataFact) AS Zi_Sapt_num
  FROM facturi
 WHERE TO_CHAR(DataFact, 'MM-YYYY') = '09-2007'
```

nrfact numeric(8,0)	datafact date	zi_din_luna text	zi_sapt_char text	zi_sapt_num double precis
3111	2007-09-01	01	SATURDAY	6
3112	2007-09-01	01	SATURDAY	6
3113	2007-09-02	02	SUNDAY	0
3115	2007-09-02	02	SUNDAY	0
3116	2007-09-10	10	MONDAY	1
3117	2007-09-10	10	MONDAY	1
3118	2007-09-17	17	MONDAY	1

Figura 6.39. Funcția EXTRACT (DOW...) în PostgreSQL

Atenție la predicatele în care se compară ziua săptămânii obținută cu ajutorul funcției TO_CHAR ! Dacă am dori să afișăm numai facturile întocmite luna în luna septembrie 2007, am fi tentați să redactăm SELECT-ul astfel:

```
SELECT NrFact, DataFact,
       TO_CHAR(DataFact, 'DD') AS Zi_Din_Luna,
       TO_CHAR(DataFact, 'DAY') AS Zi_Sapt_char
  FROM facturi
 WHERE TO_CHAR(DataFact, 'MM-YYYY') = '09-2007'
       AND TO_CHAR(DataFact, 'DAY') IN ('LUNDI', 'MONDAY')
```

Ei bine, interogarea nu extrage niciodată nimic, iar răspunsul e frustrant: TO_CHAR convertește ziua într-un sir de caractere de lungime fixă. Cum nu prea avem noroc să nimerim din prima lungimea de comparație, cel mai sănătos este să folosim o funcție care elimină spațiile de la final – TRIM (TRAILING...).

```
SELECT NrFact, DataFact,
       TO_CHAR(DataFact, 'DD') AS Zi_Din_Luna,
       TO_CHAR(DataFact, 'DAY') AS Zi_Sapt_char
  FROM facturi
 WHERE TO_CHAR(DataFact, 'MM-YYYY') = '09-2007' AND
       TRIM(TRAILING FROM TO_CHAR(DataFact, 'DAY')) IN ('LUNDI', 'MONDAY')
```

În DB2, după cum am văzut în paragrafele anterioare, obținerea oricărei informații dintr-o dată calendaristică sau moment presupune folosirea unei funcții speciale: YEAR, MONTH, DAY, HOUR, MINUTE, SECOND, MICROSECOND, DAYOFWEEK etc. De asemenea, exprimarea unui interval precizat prin diferență

dintre două date/momente se realizează prin funcția `TIMESTAMPDIFF`. Pentru conversia dintr-un tip în altul, funcția `CAST` este, în general, suficientă. Pe lângă acestea, mai pot fi puse la lucru și câteva alte funcții (mai mult sau mai puțin) „proprietare”, dintre care amintim:

- `CHAR` convertește un număr, dată sau moment în sir de caractere¹²;
- `DATE` convertește un sir de caractere într-o dată calendaristică sau extrage data dintr-un moment;
- `INTEGER` convertește un sir de caractere, dată calendaristică sau moment într-un număr întreg;
- `TIME` convertește un sir de caractere într-o „oră” sau extrage ora-minutul-secunda-fracțiunea de secundă dintr-un moment;
- `TIMESTAMP` convertește o expresie într-un moment.

Iată o interogare ce folosește câteva dintre aceste funcții și rezultatul său (figura 6.40):

```
SELECT CHAR(CURRENT_TIMESTAMP) AS "CHAR(CURRENT_TIMESTAMP)",
       DATE(CURRENT_TIMESTAMP) AS "DATE(CURRENT_TIMESTAMP)",
       INTEGER(DATE(CURRENT_TIMESTAMP)) AS
              "INTEGER(DATE(CURRENT_TIMESTAMP))",
       TIMESTAMP(DATE('2008-03-23'), TIME('23:59:58')) AS "Un_TIMESTAMP"
FROM sysibm.sysdummy1
```

CHAR(CURRENT_TIMESTAMP)	DATE(CURRENT_TIMESTAMP)	INTEGER(DATE(CURRENT_TIMESTAMP))	Un_TIMESTAMP
2008-03-27 08:56:54.937000	27.03.2008	20080327	23.03.2008 23:59:58 000000

Figura 6.40. Funcții `CHAR`, `DATE`, `INTEGER` și `TIMESTAMP` în DB2

6.6. Alte funcții-sistem

Funcția `CURRENT_DATE`, pe care am mai folosit-o de câteva ori, poate fi încadrată și la categoria funcții-sistem. Returnează, în mai toate SGBD-urile, data curentă pe care o „știe” SGBD-ul, și are numeroase rufe prin dialectele SQL: `SYSDATE`¹³ - Oracle), `DATE()` - FoxPro, `GETDATE()` - MS SQL Server etc.

`CURRENT_TIME` se folosește pentru aflarea orei (mai mult sau mai puțin exacte) curente, plus fusul orar, fiind implementată în Oracle și PostgreSQL. Fără

¹² Am folosit deja această funcție la prezentarea `TIMESTAMPDIFF`.

¹³ Nu uitați ca tipul `DATE` în Oracle este, de fapt, un `TIMESTAMP`.

zona ce indică fusul orar (GMT+2 pentru ora Bucureștiului), funcția devine LOCALTIME (PostgreSQL) – vezi figura 6.41.

The screenshot shows a PostgreSQL query window titled "Query - sql2008 on postgres@localhost:5432". The query is:

```
SELECT CURRENT_TIME, LOCALTIME
```

The results pane displays two rows of data:

	timetz time with time zone	time time without time zone
1	08:20:52.955+02	08:20:52.955

Figura 6.41. Diferența dintre CURRENT_TIME și LOCALTIME (PostgreSQL)

The screenshot shows a PostgreSQL query window titled "Query - sql2008 on postgres@localhost:5432". The query is:

```
SELECT CURRENT_TIMESTAMP, LOCALTIMESTAMP, NOW(), TIMEOFDAY()
```

The results pane displays four columns of data:

	now timestamp with time zone	timestamp timestamp without time zone	now timestamp with time zone	timeofday text
1	2008-02-05 08:57:19.753+02	2008-02-05 08:57:19.753	2008-02-05 08:57:19.753+02	Tue Feb 05 08:57:19.768000 2008 EET

Figura 6.42. Momente curente (în PostgreSQL)

Diferența dintre „general” și local se menține și în cazul funcțiilor care furnizează momentul curent, CURRENT_TIMESTAMP și LOCAL_TIMESTAMP. În PostgreSQL echivalentul funcției CURRENT_TIMESTAMP este NOW() și, în plus, mai dispunem și de funcția TIMEOFDAY() ce returnează momentul curent ca un sir de caractere – vezi figura 6.42.

În MS SQL Server ora locală și cea „universală” (ex. GMT) se obțin cu funcțiile GETDATE și GETUTCDATE – vezi figura 6.43.

The screenshot shows an MS SQL Server query window with the following query:

```
SELECT GETDATE() AS getdate, GETUTCDATE() AS getutcdate
```

The results pane displays two columns of data:

	getdate	getutcdate
1	2008-02-25 09:44:04.430	2008-02-25 07:44:04.430

Figura 6.43. Momente curente (în MS SQL Server)

Toate funcțiile din acest paragraf puteau fi tratate în secțiunile dedicate datelor și intervalelor. Următoarele, deși tot „curente” (CURRENT...) oferă informații necesare în multe aplicații, mai ales în cele client-server și web:

- CURRENT_DATABASE (PostgreSQL) și DB_NAME (MS SQL Server) furnizează numele bazei de date curente;
- CURRENT_SCHEMA (PostgreSQL), SCHEMA_NAME() (SQL Server) sau CURRENT SCHEMA (DB2) furnizează numele schemei în care se lucrează;
- CURRENT_USER (PostgreSQL, MS SQL Server), USER (Oracle, MS SQL Server, DB2, PostgreSQL), SESSION_USER (PostgreSQL, MS SQL Server) furnizează numele utilizatorului curent conectat la baza de date.

Modul lor de apelare, parametrii și, uneori, chiar funcționalitatea, pot diferi ușor de la SGBD la SGBD. Figura 6.44 ilustrează modul de execuție al interogării în PostgreSQL:

```
SELECT CURRENT_DATABASE(), CURRENT_SCHEMA(),
       CURRENT_USER, USER, SESSION_USER
```

The screenshot shows the pgAdmin III interface with a query window titled "Query - sql2008 on postgres@localhost:5432 *". The query entered is:

```
SELECT CURRENT_DATABASE(), CURRENT_SCHEMA(), CURRENT_USER, USER, SESSION_USER
```

The results pane displays the output of the query:

	current_database_name	current_schema_name	current_user_name	current_user_name	session_user_name
1	sql2008	public	postgres	postgres	postgres

Figura 6.44. Baza de date, schema și utilizatorul, toate curente, în PostgreSQL

6.7. Funcții-agregat

Denumirea acestui tip de funcții sugerează faptul că acestea se aplică unui set de valori și returnează o singură valoare (de cele mai multe ori numerică). Cele mai folosite sunt: COUNT, SUM, MIN și MAX. Formatul general al unei fraze SELECT ce conține funcții agregat este:

```
SELECT funcția-predefinită1, ... , funcția-predefinităN
      FROM listă-tabele
      WHERE condiții.
```

În lipsa opțiunii GROUP BY (tratată în capitolul următor), dacă în clauza SELECT este prezentă o funcție agregat, rezultatul va conține o singură linie.

6.7.1. COUNT

Funcția COUNT contorizează valorile nenule ale unei coloane sau numărul de linii dintr-un rezultat al unei interogări, altfel spus, în rezultatul unei consultări, COUNT numără câte valori diferite de NULL are o coloană specificată sau câte linii sunt.

Câți clienți are firma ?

```
SELECT COUNT (*) AS NrClienti
  FROM clienti
```

Prezența asteriscului ca argument al funcției COUNT are ca efect numărarea liniilor tăbelei CLIENTI. Rezultatul este prezentat în figura 6.45.

NRCLIENTI
7

Figura 6.45. Numărul clienților

Tăbea CLIENTI are cheie primară atributul CodCl care nu poate avea valori nule; de aceea, corectă este și soluția:

```
SELECT COUNT (CodCl) AS NrClienti
  FROM clienti
```

Folosind concatenarea, se poate obține un rezultat de forma unui mesaj (vezi figura 6.46).

```
SELECT 'Firma are '||COUNT (*)||' clienti' AS Rezultat
  FROM clienti
```

REZULTAT
Firma are 7 clienti

Figura 6.46. Altă formă de prezentare (în Oracle) a rezultatului funcției COUNT

Reamintim că, în DB2 sau SQL Server, pe lângă operatorul de concatenare (|| sau +) sau funcția CONCAT, este necesară conversia rezultatului funcției COUNT care este INTEGER în sir de caractere (CHAR), așa că în DB2 putem scrie:

```
SELECT CONCAT ('Firma are ', CONCAT (
    CAST (COUNT (*) AS VARCHAR(5)), ' clienti')
) AS Rezultat
  FROM clienti
```

iar în MS SQL Server:

```
SELECT 'Firma are ' + CAST (COUNT (*) AS VARCHAR(5)) + ' clienti'
      AS Rezultat
  FROM clienti
```

Câte linii are produsul cartezian al tăbeelor FACTURI și LINIIFACT ?

Suntem la rubrica „Răspundem ascultătorilor” (însă nu la Radio Erevan).

```
SELECT COUNT(*)
  FROM facturi, liniifact
```

sau

```
SELECT COUNT(*)
  FROM facturi CROSS JOIN liniifact
```

Oricare din aceste două interogări returnează 1680 (în condițiile în care liniile sunt exact cele din scriptul de inserare din capitolul 3).

Pentru câți clienți se cunoaște adresa ?

```
SELECT COUNT (Adresa) AS NrClienti_cu_adresa_cunoscuta
FROM clienti
```

Rezultatul, 5 (fig. 6.47), putea fi obținut și ceva mai complicat, folosind în clauza WHERE operatorul IS NULL pe care-l tot amânam pentru capitolul 8.

NrClienti_cu_adresa_cunoscuta
5

Figura 6.47. Numărul de linii din CLIENTI cu adresă nenulă (SQL Server)

Câte facturi au fost emise pe 7 august 2007 ?

```
SELECT COUNT(NrFact) AS NrFacturi
FROM facturi
WHERE DataFact = DATE'2007-08-07'
```

Vă spun eu: patru.

Câte facturi au fost emise clientilor din județul Vaslui ?

Problema este că numărul facturilor poate fi determinat din tabela FACTURI, însă condiția de selecție, *Judet = „Vaslui”*, este aplicabilă tabeliei JUDETE. Întrucât cele două tabele nu pot fi jonctionate direct, se introduc în clauza FROM „intermediarii” CLIENTI și CODURI_POSTALE:

```
SELECT COUNT(NrFact) AS NrFacturi
FROM facturi f
    INNER JOIN clienti c ON f.codcl=c.codcl
    INNER JOIN coduri_postale cp ON c.codpost=cp.codpost
    INNER JOIN judete j ON cp.jud=j.jud
WHERE Judet='Vaslui'
```

Câte localități sunt preluate în baza de date ?

Problema este un pic mai complicată deoarece nu există o tabelă în care numele localității să fie cheie primară sau alternativă. Tabela CODURI_POSTALE poate conține mai multe linii pentru același orașe, deoarece orașele ceva mai mari prezintă mai multe coduri poștale.

Dacă ținem cont că o acceașă localitate (în mediul rural, mai ales) poate să apară în mai multe județe, interogarea pare a fi:

```
SELECT COUNT(Loc || Jud) AS NrLocalit
```

```
FROM coduri_postale14
```

R	NRLOCALIT_GRESIT
	12

Figura 6.48. Folosirea eronată a funcției COUNT pentru aflarea numărului de localități

Necazul e că rezultatul obținut, 12 (vezi figura 6.48), este incorrect, deoarece funcția COUNT numără toate valorile nenule ale atributului sau expresiei argument. Din fericiere, există însă o clauză prin care o valoare să se ia în calcul o singură dată: DISTINCT. Rezultatul corect presupune varianta:

```
SELECT COUNT(DISTINCT Loc || Jud) AS NrLocalit
FROM coduri_postale
```

Valoarea furnizată de această interogare – vezi figura 6.49 – este cea corectă ținând cont de liniile introduse în tabelă în capitolul 3.

NRLOCALIT	▼
	8

Figura 6.49. Opțiunea DISTINCT a funcției COUNT (rezultat DB2)

6.7.2. SUM

Funcția SUM este una dintre cele mai utilizate funcții în aplicațiile economice, deoarece datele financiar-contabile și cele ale evidenței tehnico-operative sunt preponderent cantitative. Probabil că prea multe explicații teoretice despre modul în care operează această funcție sunt inutile, aşa încât vom trece în revistă câteva exemple.

Care este valoarea fără TVA a facturii 1111 (figura 6.50) ?

Valoarea unei facturi este dată de suma valorilor liniilor sale, fiecare linie descriind vânzarea unui produs sau serviciu. Cum valoarea fără TVA a fiecărei linii din factură este produsul *Cantitate * PretUnit*, comanda SELECT are forma:

```
SELECT SUM(Cantitate * PretUnit) AS ValFaraTVA
FROM liniifact WHERE NrFact = 1111
```

R	VALFARATVA_1111
	3658750

Figura 6.50. Valoarea fără TVA a facturii 1111

¹⁴ Nu uitați că în MS SQL Server operatorul de concatenare nu e ||, ci +.

Care este valoarea fără TVA a facturilor întocmite pe 7 august 2007 ?

Dacă valoarea fără TVA a facturilor se obține din LINIIFACT, condiția de selecție se aplică asupra atributului DataFact din FACTURI, așa că cele două tabele trebuie jonctionate, rezultatul fiind cel din figura 6.51.

```
SELECT SUM(Cantitate * PretUnit) AS ValFaraTVA_7aug2007
FROM liniifact lf INNER JOIN facturi f ON lf.NrFact=f.NrFact
WHERE DataFact = DATE'2007-08-07'
```

VALFARATVA_7AUG2007
8941450

Figura 6.51. Vânzările fără TVA din 7 august 2007

Care sunt cele trei valori: fără TVA, TVA și totală (cu TVA) ale facturii 1111 ?

Procentul de TVA care se aplică la vânzarea unui produs se află în tabela PRODUSE (atributul ProcTVA), așa că în clauza FROM va fi adăugată pentru joncțiune și această tabelă (valorile obținute sunt cele din figura 6.52):

```
SELECT SUM(Cantitate * PretUnit) AS ValFaraTVA,
       SUM(Cantitate * PretUnit * ProcTVA) AS TVA,
       SUM(Cantitate * PretUnit + Cantitate * PretUnit * ProcTVA)
             AS ValTotala
FROM liniifact lf
      INNER JOIN facturi f ON lf.NrFact=f.NrFact
      INNER JOIN produse p ON lf.CodPr=p.CodPr
WHERE f.NrFact=1111
```

VALFARATVA	TVA	VALTOTALA
3,658,750.00	687,287.5000	4,346,037.5000

Figura 6.52. Cele trei valori ale facturii 1111

În condițiile în care firma pentru care se folosește baza noastră de date comercializează exclusiv produse la care procentul TVA este unic - 19% ar fi corectă și varianta:

```
SELECT SUM(Cantitate * PretUnit) AS ValFaraTVA,
       SUM(Cantitate * PretUnit * .19) AS TVA,
       SUM(Cantitate * PretUnit + Cantitate * PretUnit * .19) AS ValTotala
FROM liniifact WHERE NrFact = 1111
```

În plus, nici nu ar mai fi nevoie de atributul ProcTVA în PRODUSE. Nu recomand renunțarea la acest atribut, început, după un timp, procentele TVA ale unor produse se vor modifica.

Pentru a ne mai dezmemori mouse-ul, vom afișa cele trei valori pe verticală, ca în figura 6.53, prilej de ceva reuniuni. Iată versiunea Oracle/PostgreSQL¹⁵:

```

SELECT '1' AS "",'Valoarea fara TVA' AS "Factura 1111",
       TRUNC(SUM(Cantitate * PretUnit)) AS Suma
  FROM liniifact lf INNER JOIN produse p ON lf.CodPr=p.CodPr
 WHERE NrFact = 1111
UNION
SELECT '2','TVA', TRUNC(SUM(Cantitate * PretUnit * ProcTVA))
  FROM liniifact lf
        INNER JOIN facturi f ON lf.NrFact=f.NrFact
        INNER JOIN produse p ON lf.CodPr=p.CodPr
 WHERE lf.NrFact = 1111
UNION
SELECT '3','Valoarea totala', TRUNC(SUM(Cantitate*PretUnit*(1+ProcTVA)))
  FROM liniifact lf
        INNER JOIN facturi f ON lf.NrFact=f.NrFact
        INNER JOIN produse p ON lf.CodPr=p.CodPr
 WHERE lf.NrFact = 1111

```

¹⁵ În DB2 funcția TRUC are, obligatoriu un al doilea argument care semnifică la câte poziții fractionare să se facă trunchierea, iar în MS SQL Server varianta funcțională a acestei interogări este:

```

SELECT '1' AS "",'Valoarea fara TVA' AS "Factura 1111",
       ROUND(SUM(Cantitate * PretUnit),0,1) AS Suma
  FROM liniifact lf INNER JOIN produse p ON lf.CodPr=p.CodPr
 WHERE NrFact = 1111
UNION
SELECT '2','TVA', ROUND(SUM(Cantitate * PretUnit * ProcTVA),0,1)
  FROM liniifact lf INNER JOIN facturi f ON lf.NrFact=f.NrFact INNER JOIN produse p
        ON lf.CodPr=p.CodPr
 WHERE lf.NrFact = 1111
UNION
SELECT '3','Valoarea totala', ROUND(SUM(Cantitate*PretUnit*(1+ProcTVA)),0,1)
  FROM liniifact lf INNER JOIN facturi f ON lf.NrFact=f.NrFact
        INNER JOIN produse p ON lf.CodPr=p.CodPr
 WHERE lf.NrFact = 1111

```

	Factura 1111	SUMA
1	Valoarea fara TVA	3658750
2	TVA	687287
3	Valoarea totala	4346037

Figura 6.53. Cele trei valori ale facturii 1111 - afişare pe verticală

Nu întotdeauna rezultatul reflectă ordinea firească de procesare a liniilor. Este motivul pentru care am introdus în interogare o coloană (fără nume, de fapt, cu „numele” spațiu), cu un prim rol decorativ și un al doilea legat de afișarea liniilor în ordinea care ne interesează. Dacă nu am fi fost mulțumiți de ordinea afișării, am fi adăugat interogării clauza *ORDER BY 1*.

La cât se situează cifra vânzărilor pe data de 7 august 2007 ?

Cifra de afaceri reprezentă, de fapt, valoarea totală a vânzărilor, deci valoarea cu tot cu TVA a facturilor emise pe 7 august 2007 (figura 6.54):

```
SELECT '7 aug. 2007' AS Data,
       SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari
  FROM liniifact lf
 INNER JOIN facturi f ON lf.NrFact=f.NrFact
 INNER JOIN produse p ON lf.CodPr=p.CodPr
 WHERE DataFact = DATE'2007-08-07'
```

DATA	VINZARI
7 aug. 2007	10610000,5

Figura 6.54. Vânzările cu TVA din 7 august 2007

Care este valoarea vânzărilor pentru “Client 1 SRL” (figura 6.55) ?

```
SELECT SUM(Cantitate * PretUnit * (1 + ProcTVA)) AS Vinzari_Client1
  FROM liniifact lf
 INNER JOIN produse p ON lf.CodPr=p.CodPr
 INNER JOIN facturi f ON lf.NrFact=f.NrFact
 INNER JOIN clienti c ON f.CodCl=c.CodCl
 WHERE DenCl = 'Client 1 SRL'
```

VINZARI_CLIENT1
15061538

Figura 6.55. Vânzările către Client 1 SRL

Să se afișeze valoarile (fără TVA, TVA, cu TVA) liniilor facturii 1111, plus o linie de total.

Practic, suntem interesați de un raport precum cel din figura 6.56. Raportul are două tipuri de rânduri, cel corespunzător liniilor din tabela LINIIFACT și cel de totalizare.

Linia	Produs	Cantitate	Pret Unitar	Val. fara TVA	TVA linie	Val cu TVA linie
1	Produs 1	50	1000	50000	9500	59500
2	Produs 2	75	1050	78750	7087	85837
3	Produs 5	500	7060	3530000	670700	4200700
101	TOTAL factura	(null)	(null)	3658750	687287	4346037

Figura 6.56. Un rând de total pe lângă liniile facturii 1111

Obținerea lor presupune două SELECT-uri conectat prin UNION. Pentru ca reuniunea să funcționeze, rezultatele celor două SELECT-uri trebuie să fie unicompabile, adică să prezintă același număr de coloane, iar prima coloană rezultat a primului SELECT să fie de același tip cu prima coloană rezultat al celui de-al doilea SELECT s.a.m.d.

```

SELECT linie AS "Linia", DenPr AS "Produs", Cantitate AS "Cantitate",
       TRUNC(PretUnit) AS "Pret Unitar",
       TRUNC(Cantitate * PretUnit) AS "Val. fara TVA",
       TRUNC(Cantitate * PretUnit * ProcTVA) AS "TVA linie",
       TRUNC(Cantitate * PretUnit * (1 + ProcTVA)) AS "Val cu TVA linie"
  FROM liniifact lf INNER JOIN produse p ON lf.CodPr=p.CodPr
 WHERE NrFact = 1111
UNION
SELECT 101, 'TOTAL factura',
        NULL,
        NULL,
        TRUNC(SUM(Cantitate * PretUnit)),
        TRUNC(SUM(Cantitate * PretUnit * ProcTVA)),
        TRUNC(SUM(Cantitate * PretUnit * (1 + ProcTVA)))
  FROM liniifact lf INNER JOIN produse p ON lf.CodPr=p.CodPr
 WHERE NrFact = 1111
 ORDER BY 1

```

Cele două valori NULL din SELECT-ul corespunzător rândului de total au fost necesare pentru că nu se pot totaliza cantitățile și prețurile unitare, atâtă vreme cât produsele sunt diferite. Clauza ORDER BY forțează ordonarea după prima coloană. Atenție, nici în Oracle, nici în PostgreSQL, nu funcționează în interogarea de mai sus sintaxa *ORDER BY linie*¹⁶!. În schimb, funcționează în MS SQL Server, dar aici trebuie să înlocuim TRUNC cu ROUND cu un parametru suplimentar:

```

SELECT linie AS "Linia", DenPr AS "Produs", Cantitate AS "Cantitate",
       ROUND(PretUnit,0,1) AS "Pret Unitar",

```

¹⁶ Ar funcționa, totuși, ORDER BY "Linia"

```

        ROUND(Cantitate * PretUnit,0,1) AS "Val. fara TVA",
        ROUND(Cantitate * PretUnit * ProcTVA,0,1) AS "TVA linie",
        ROUND(Cantitate * PretUnit * (1 + ProcTVA),0,1) AS "Val cu TVA linie"
FROM liniifact lf INNER JOIN produse p ON lf.CodPr=p.CodPr
WHERE NrFact = 1111
      UNION
SELECT 101, ' TOTAL factura ', NULL,
       NULL,
       ROUND(SUM(Cantitate * PretUnit),0,1),
       ROUND(SUM(Cantitate * PretUnit * ProcTVA),0,1),
       ROUND(SUM(Cantitate * PretUnit * (1 + ProcTVA)),0,1)
FROM liniifact lf INNER JOIN produse p ON lf.CodPr=p.CodPr
WHERE NrFact = 1111
ORDER BY linie

```

Dacă în loc de NULL am vrea să apară spațiu, lucrurile ar degenera, deoarece spațiul este tip sir de caractere iar cantitățile și prețurile unitare sunt numerice – vezi figura 6.57.

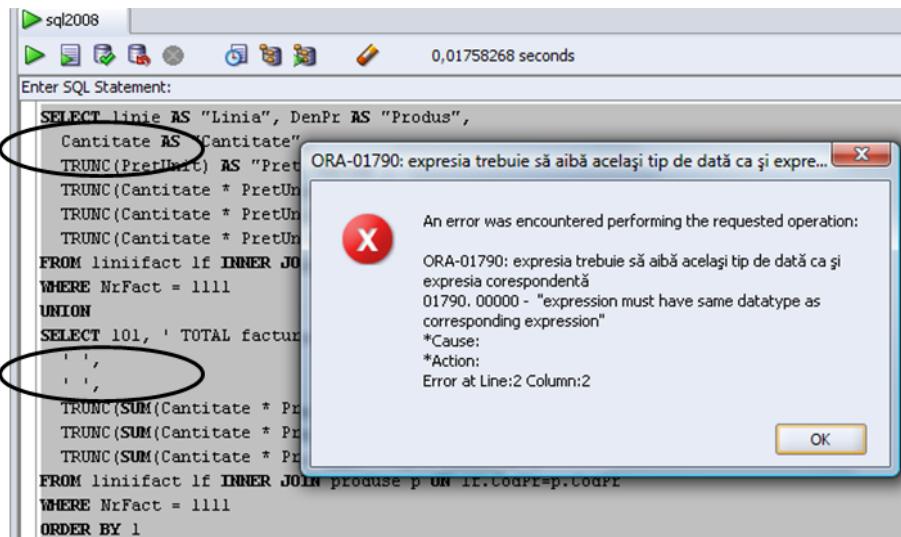


Figura 6.57. Nepotrivire (de tip) la reuniune

Care este valoarea medie a prețului (inclusiv TVA) la care a fost vândută o unitate din "Produs 1" ?

Nu, n-am trecut la funcția AVG fără să vă fi anunțat din timp. Soluția se bazează pe raportul dintre suma valorilor și cantitatea însumată pentru acest produs (figura 6.58). Iată și interogarea:

```

SELECT SUM(Cantitate*PretUnit*(1+ProcTVA)) / SUM(Cantitate)
      AS "Pret Unitar Mediu (tare)"
FROM liniifact lf
      INNER JOIN produse p ON lf.CodPr=p.CodPr
      INNER JOIN facturi f ON lf.NrFact=f.NrFact

```

```
WHERE DenPr = 'Produs 1'
```

	Pret Unitar Mediu (tare)
	1131,044622425629061784897025171624714

Figura 6.58. Preț mediu calculat prin raportul între suma valorii și suma cantităților

Ne bazăm pe faptul că este vorba de același produs, deci are sens să cumulăm cantitățile.

Care este situația încasării facturii 1120 ?

Rezolvarea acestei probleme presupune afișarea, pentru factura 1120, atât a valorii totale, obținută din tabelele LINIIFACT și PRODUSE, cât și a valorii încasate, obținută din INCASFAC. Prin urmare, pare firesc să încercăm varianta:

```
SELECT '1120' AS NrFact,
       SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Facturat,
       SUM(Transa) AS Incasat
  FROM liniifact lf
    INNER JOIN produse p ON lf.CodPr=p.CodPr
    INNER JOIN facturi f ON lf.NrFact=f.NrFact
    INNER JOIN incasfact i ON f.NrFact=i.NrFact
 WHERE F.NrFact = 1120
```

NRFAC	FACTURAT	INCASAT
1120	97664	7315

Figura 6.59. Valorile facturată și încasată ale facturii 1120

Rezultatul din figura 6.59 arată că lucrurile ar fi în regulă. Dar dacă înlocuim numărul facturii 1120 cu 1111, fraza:

```
SELECT '1111' AS NrFact,
       SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Facturat,
       SUM(Transa) AS Incasat
  FROM liniifact lf
    INNER JOIN produse p ON lf.CodPr=p.CodPr
    INNER JOIN facturi f ON lf.NrFact=f.NrFact
    INNER JOIN incasfact i ON f.NrFact=i.NrFact
 WHERE F.NrFact = 1111
```

va genera rezultatul din figura 6.60.

NRFAC	FACTURAT	INCASAT
1111	4346037,5	161988

Figura 6.60. Valoarea facturată și valoarea încasată (greșită !) pentru factura 1111

Valoarea încasată, 161988, e complet anapoda, valoarea încasată „reală” din această factură fiind 53996. Interesant este că valoarea greșită este de trei ori mai mare decât cea facturată. Care e explicația ? Răspunsul e mai ușor de aflat dacă vizualizăm rezultatul unei interogări “ajutătoare” ce generează tabela pe care se aplică cele două funcții SUM - vezi figura 6.61.

```
SELECT lf.*, Cantitate * PretUnit * (1+ProcTVA) AS Val_fact_linie,
```

```

Transa AS Incasat
FROM liniifact lf
    INNER JOIN produse p ON lf.CodPr=p.CodPr
    INNER JOIN facturi f ON lf.NrFact=f.NrFact
    INNER JOIN incasfact i ON f.NrFact=i.NrFact
WHERE F.NrFact = 1111

```

NRFAC	LINIE	CODPR	CANTITATE	PRETUNIT	VAL_FACT_LINIE	INCASAT
1111	1	1	50	1000	59500	53996
1111	2	2	75	1050	85837,5	53996
1111	3	5	500	7060	4200700	53996

Figura 6.61. Rezultatul joncțiunii “componentelor” *facturare și încasare* pentru factura 1111

Întâmplarea face ca pentru această factură să existe o singură tranșă de încasare. Tranșa de 53996 se repetă pentru fiecare linie a facturii 1111. Normal că funcția SUM întoarce o valoare triplă față de cea reală. Ce-i de făcut? Să încercăm un artificiu. Împărțim sumă tranșelor încasate la numărul liniilor facturii:

```

SELECT '1111' AS NrFact, SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Facturat,
        SUM(Transa) / COUNT(*) AS Incasat
FROM liniifact lf

```

```

    INNER JOIN produse p ON lf.CodPr=p.CodPr
    INNER JOIN facturi f ON lf.NrFact=f.NrFact
    INNER JOIN incasfact i ON f.NrFact=i.NrFact
WHERE F.NrFact = 1111

```

Merge! Sau, cel puțin, aşa arată figura 6.62.

NRFAC	FACTURAT	INCASAT
1111	4346037,5	53996

Figura 6.62. Valoarile (încurajatoare) corecte pentru factura 1111

Rămâne însă o sursă de crizpare: ce se întâmplă cu facturile încasate în mai multe tranșe? Spre exemplu, factura 1117 are două linii și e încasată în trei tranșe. Dacă utilizăm interogarea “ajutătoare” (penultima), schimbând, bineînțeles, numărul facturii, echivalenta tabelei din figura 6.61 se prezintă ca în figura 6.63.

NRFAC	LINIE	CODPR	CANTITATE	PRETUNIT	VAL_FACT_LINIE	INCASAT
1117	1	2	100	1000	109000	9754
1117	1	2	100	1000	109000	9754
1117	1	2	100	1000	109000	3696
1117	2	1	100	950	113050	9754
1117	2	1	100	950	113050	9754
1117	2	1	100	950	113050	3696

Figura 6.63. Valori repetitive datorită mai multor tranșe de încasare – factura 1117

Există nu mai puțin de 3 (tranșe de încasare - INCASFAC) * 2 (linii în LINIIFACT) = 6 linii. Pe baza acestei observații ochiometrice, putem să încercăm o

soluție finală, împărțind valoarea facturată la numărul tranșelor de încasare și valoarea încasată la numărul de linii ale facturii respective:

```
SELECT '1117' AS NrFact,
       SUM(Cantitate * PretUnit * (1+ProcTVA)) /
       COUNT(DISTINCT i.CodInc) AS Facturat,
       SUM(Transa) / COUNT(DISTINCT lf.Linie) AS Incasat
  FROM liniifact lf
    INNER JOIN produse p ON lf.CodPr=p.CodPr
    INNER JOIN facturi f ON lf.NrFact=f.NrFact
    INNER JOIN incasfact i ON f.NrFact=i.NrFact
 WHERE f.NrFact = 1117
```

NRFAC	FACTURAT	INCASAT
1117	222050	23204

Figura 6.64. Valorile (corecte) facturată și încasată – fact.1117

Rezultatul din figura 6.64 ne îndreptățește să sperăm că soluția e bună. Și, într-adevăr, este funcțională, dar numai pentru facturile care au cel puțin o tranșă de încasare ! Pentru cealaltă categorie avem nevoie de cunoștințe ceva mai avansate (jocuri externe) – vezi capitolul 8. În altă ordine de idei, dacă SQL-ul ar fi fost atât de încâlcit la asemenea probleme nu prea complicate, numărul utilizatorilor ar fi fost incomparabil mai mic. Noroc cu opțiunile din capitolele următoare care ne fac viața suportabilă (cel puțin, în SQL).

6.7.3. AVG

Funcția AVG calculează media aritmetică a unei coloane într-o tabelă oarecare, prin divizarea sumei valorilor coloanei respective la numărul de valori nenule ale acesteia. Nefiind prea multe de adăugat, vom discuta doar două exemple destul de simple.

Care este valoarea (fără TVA) medie a liniilor din factura 1111 (figura 6.65) ?

```
SELECT 'Val. medie (fara TVA) a prod. din fact. 1111' AS Explicatii,
       TRUNC(AVG(Cantitate * PretUnit),2) AS ValMedie
  FROM liniifact
 WHERE NrFact = 111117
```

¹⁷ Sintaxă Oracle/PostgreSQL. În DB2 și SQL Server funcția TRUNC se folosește așa cum a fost arătat la interogarea ce obține liniile din figura 6.53.

EXPLICATII	VALMEDIE
Val. medie (fara TVA) a prod. din fact. 1111	1219583,33

Figura 6.65. Valoarea medie a unei linii din factura 1111

Care este media valorilor (cu TVA) la care a fost vândut "Produs 1" ?

```
SELECT 'Val. medie a vinzarilor Produs 1' AS Explicatii,
       ROUND(AVG(Cantitate*PretUnit*(1+ProcTVA)),2) AS "Valoare (cu TVA) medie"
  FROM liniifact lf INNER JOIN produse p ON lf.CodPr=p.CodPr
 WHERE DenPr = 'Produs 1'
```

Adesea, prin aplicarea funcției AVG, sau prin formularea unor expresii ce conțin rapoarte între atribute/constante, se obțin rezultate cu numeroase zecimale. În cazul de față, pentru preîntâmpinarea acestui disconfort vizual și intelectual, a fost preferată funcția ROUND ce rotunjește media la două zecimale (vezi figura 6.66)¹⁸.

Explicatii	Valoare (cu TVA) medie
Val. medie a vinzarilor Produs 1	109837.000000

Figura 6.66. Media valorilor (cu TVA) la care a fost vândut, într-o factură, Produs 1

6.7.4. MIN & MAX

Funcțiile MAX și MIN sunt deosebit de utile în diverse tipuri de analiză a datelor, determinând valoarea maximă, respectiv minimă, pentru o coloană (atribut). Se pot folosi și pentru atribute de tip sir de caractere, caz în care elementul de comparație este codul ASCII/UTF al caracterelor.

Care este localitatea cu ultima denumire, în ordine alfabetică ?

Atributul care ne interesează este Loc din tabela CODURI_POSTALE, iar valoarea determinată este Vaslui (figura 6.67).

```
SELECT MAX(Loc) AS "Ultima_Localitate_Alfabetic"
      FROM coduri_postale
```

¹⁸ Pare dubios că, deși facem rotunjirea la două zecimale, coloana-medie are şase zecimale. Încă o dată, este o chestiune doar de afișare, calculul făcându-se, într-adevăr, la precizia de două zecimale. Dacă nu mă credeți, încercați cu produsele 2 sau 5.

Ultima_Localitate_Alfabetic	
Vaslui	

Figura 6.67. (A fost sau n-a fost) Vaslui, ultima localitate (doar din punct de vedere alfabetic!)

Care este primul client și ultimul client (în ordinea numelui) din județul Iași (figura 6.68) ?

Nu e nici o problemă dacă o clauză SELECT conține două sau mai multe funcții agregat, atâtă vreme cât nu apar alte coloane (ne-agregat).

```
SELECT MIN(DenCl) AS Primul_Client,
       MAX(DenCl) AS Ultimul_Client
  FROM clienti c
    INNER JOIN coduri_postale cp ON c.codpost=cp.codpost
    INNER JOIN judete j ON cp.jud=j.jud
 WHERE Judet = 'Iasi'
```

PRIMUL_CLIENT	ULTIMUL_CLIENT
Client 1 SRL	Client 4

Figura 6.68. Primul și ultimul clienti (d.p.d.v. alfabetic) din județul Iași

Care este cel mai mare preț unitar (fără TVA) la care a fost vândut Produs 1 ?

```
SELECT MAX(PretUnit)
  FROM liniifact lf INNER JOIN produse p ON lf.CodPr = p.CodPr
 WHERE DenPr = 'Produs 1'
```

MAX(PRETUNIT)
1000

Figura 6.69. Prețul unitar maxim la care a fost vândut Produsul 1

Din păcate, dacă dorim să aflăm și în ce factură produsul are prețul unitar maxim, soluția:

```
SELECT MAX(PretUnit), NrFact
  FROM liniifact lf INNER JOIN produse p ON lf.CodPr = p.CodPr
 WHERE DenPr = 'Produs 1'
```

nu funcționează (vezi figura 6.70), deoarece în clauza SELECT apare o funcție agregat împreună cu un atribut „simplu” în lipsa clauzei de grupare (vezi capitolul următor).

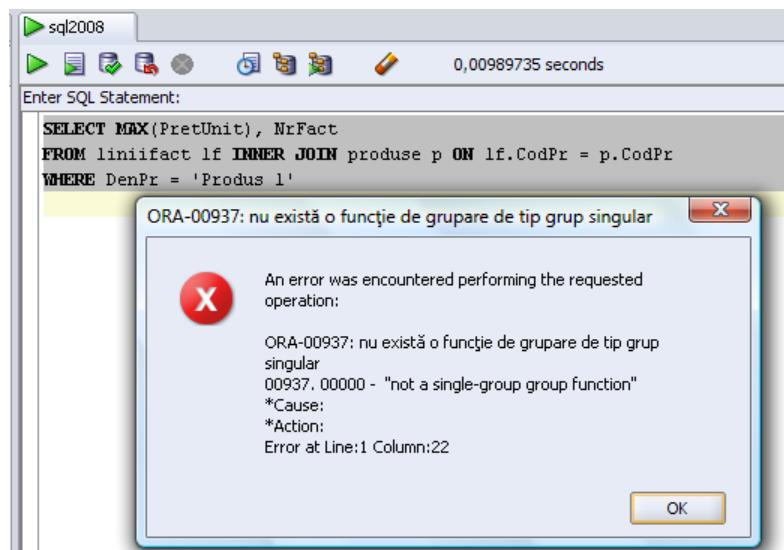


Figura 6.70. În lipsa clauzei GROUP BY este interzisă apariția în clauza SELECT a unei funcții agregat împreună cu un atribut/expresie ne-agregat

Până la subconsultările din capitolul 9 încercăm o soluție gen "cârpeală" Oracle/PostgreSQL, concatenând prețul cu numărul facturii, plus ceva text, iar funcția MAX o vom aplica la acest sir:

```
SELECT MAX('Pretul maxim=' || LF.PretUnit || 
           ', apare in factura '|| NrFact)
       AS Mesaj_cam_lung
FROM liniifact lf INNER JOIN produse p ON lf.CodPr = p.CodPr
WHERE DenPr = 'Produs 1'19
```

MESAJ_CAM_LUNG
Pretul maxim=950, apare in factura 3117

Figura 6.71. Cârpeli Oracle (vol.1)

¹⁹ În MS SQL Server interogarea are forma: SELECT MAX('Pretul maxim=' + CAST (LF.PretUnit AS VARCHAR) +' , apare in factura ' + CAST (NrFact AS VARCHAR)) AS Mesaj_cam_lung FROM liniifact lf INNER JOIN produse p ON lf.CodPr = p.CodPr WHERE DenPr = 'Produs 1'

Rezultatul din figura 6.71 este eronat, deoarece produsul 1 s-a vândut la un preț mai mare decât 950 (1000 lei, în facturile 1111, 2111 și 3111), după cum se vede în figura 6.72, al cărei conținut a fost extas prin interogarea:

```
SELECT lf.*  
FROM liniifact lf INNER JOIN produse p ON lf.CodPr = p.CodPr  
WHERE DenPr = 'Produs 1'  
ORDER BY pretunit DESC
```

NRFAC	LINIE	CODPR	CANTITATE	PRETUNIT
1111	1	1	50	1000
2111	1	1	57	1000
3111	1	1	57	1000
1117	2	1	100	950
3117	2	1	110	950
2117	2	1	110	950
1118	2	1	150	930
2118	2	1	120	930
3118	2	1	120	930

Figura 6.72. Cârpeli (vol.1)

Ne întrebăm, firește, de la ce ni se trage rușinea din figura 6.71. Ei bine, de la concatenare! Pentru a înțelege mai bine, să introducem sirul de caractere ce constituie argumentul funcției MAX chiar în interogarea de mai sus:

```
SELECT lf.* , 'Pretul maxim=' || LF.PretUnit ||  
    , ' apare in factura '|| NrFact AS Mesaj_la_fel_de_lung  
FROM liniifact lf INNER JOIN produse p ON lf.CodPr = p.CodPr  
WHERE DenPr = 'Produs 1'  
ORDER BY pretunit DESC
```

De data aceasta, figura 6.73 este destul de limpede pentru a înțelege modalitatea de ordonare. Practic, la concatenare se ia în calcul prima cifră semnificativă a prețului unitar, astfel încât 9 (de la 950) este mai mare decât 1 (de la 1000).

NRFAC	LINIE	CODPR	CANTITATE	MESAJ_LA_FEL_DE_LUNG
3117	2	1	110	950 Pretul maxim=950, apare in factura 3117
2117	2	1	110	950 Pretul maxim=950, apare in factura 2117
1117	2	1	100	950 Pretul maxim=950, apare in factura 1117
3118	2	1	120	930 Pretul maxim=930, apare in factura 3118
2118	2	1	120	930 Pretul maxim=930, apare in factura 2118
1118	2	1	150	930 Pretul maxim=930, apare in factura 1118
3111	1	1	57	1000 Pretul maxim=1000, apare in factura 3111
2111	1	1	57	1000 Pretul maxim=1000, apare in factura 2111
1111	1	1	50	1000 Pretul maxim=1000, apare in factura 1111

Figura 6.73. La concatenare, se ia în calcul prima cifră semnificativă a prețului

Pentru a rezolva problema, forțăm conversia prețului unitar în sir de caractere pe un număr prestabilit de poziții. Funcția PostgreSQL/Oracle TO_CHAR este cea

care va asigurarea „umplerea” zerourilor nesemnificative cu spații și, astfel, viabilitatea ideii:

```
SELECT lf.*, 'Pretul maxim=' || TO_CHAR(LF.PretUnit, '999999.99')
      || ', apare in factura '|| NrFact AS Mesaj_la_fel_de_lung
FROM liniifact lf INNER JOIN produse p ON lf.CodPr = p.CodPr
WHERE DenPr = 'Produs 1'
ORDER BY Mesaj_la_fel_de_lung DESC
```

	NRFACT	LINIE	CODPR	CANTITATE	PRETUNIT	MESAJ_LA_FEL_DE_LUNG
	3111	1	1	57	1000	Pretul maxim= 1000.00, apare in factura 3111
	2111	1	1	57	1000	Pretul maxim= 1000.00, apare in factura 2111
	1111	1	1	50	1000	Pretul maxim= 1000.00, apare in factura 1111
	3117	2	1	110	950	Pretul maxim= 950.00, apare in factura 3117
	2117	2	1	110	950	Pretul maxim= 950.00, apare in factura 2117
	1117	2	1	100	950	Pretul maxim= 950.00, apare in factura 1117
	3118	2	1	120	930	Pretul maxim= 930.00, apare in factura 3118
	2118	2	1	120	930	Pretul maxim= 930.00, apare in factura 2118
	1118	2	1	150	930	Pretul maxim= 930.00, apare in factura 1118

Figura 6.74. Folosirea funcției TO_CHAR pentru ordonarea corectă a prețului

Ordinea este acum cea din figura 6.74. În MS SQL Server, în locul funcției TO_CHAR se folosește STR:

```
SELECT MAX('Pretul maxim=' + STR(LF.PretUnit,10,2) +
           ', apare in factura ' + STR(NrFact,10)) AS Mesaj_cam_lung
FROM liniifact lf INNER JOIN produse p ON lf.CodPr = p.CodPr
WHERE DenPr = 'Produs 1'
```

Firește, în loc de TO_CHAR, puteam recurge la eleganta CAST. Lucrurile nu sunt chiar aşa de simple, întrucât ordinea rândurilor din rezultatul interogării²⁰:

```
SELECT lf.*, 'Pretul maxim=' || CAST (LF.PretUnit AS CHAR(12))
      || ', apare in factura '|| NrFact AS Mesaj_la_fel_de_lung
FROM liniifact lf INNER JOIN produse p ON lf.CodPr = p.CodPr
WHERE DenPr = 'Produs 1'
ORDER BY Mesaj_la_fel_de_lung DESC
```

arată la fel de deprimant ca în figura 6.73. Remedierea situației presupune o amețitoare combinație LPAD (TRIM (...)) prin care forțăm alinierea la dreapta a

²⁰ Sintaxă Oracle/PostgreSQL. Valabil și pentru următoarele două interioigări.

prețului unitar și completarea cu spații la stânga pentru umplerea eventualelor poziții datorate zerourilor nesemnificative:

```
SELECT lf.* , 'Pretul maxim=' || LPAD(
    TRIM (TRAILING FROM
        CAST(LF.PretUnit AS CHAR(12))
        ),
        12, ' ')
    || ', apare in factura '|| NrFact AS Mesaj_la_fel_de_lung
FROM liniifact lf INNER JOIN produse p ON lf.CodPr = p.CodPr
WHERE DenPr = 'Produs 1'
ORDER BY Mesaj_la_fel_de_lung DESC
```

În fine, ajungem și la final, considerând expresia concatenată argument al funcției MAX:

```
SELECT MAX ('Pretul maxim=' ||
    LPAD(
        TRIM (TRAILING FROM CAST(LF.PretUnit AS CHAR(12))
        ),
        12, ' ')
    || ', apare in factura '|| NrFact) AS PretUnitar_Maxim
FROM liniifact lf INNER JOIN produse p ON lf.CodPr = p.CodPr
WHERE DenPr = 'Produs 1'
```

Soluția funcționează, cel puțin dacă ne luăm după rezultatul din figura 6.75, să că nu-are de ce să ne fie jenă (prea tare) de improvizare. Rezultatul este incomplet, însă, atunci când prețul maxim apare în două sau mai multe facturi, deoarece interogarea de mai sus extrage numai una dintre facturi.

PRETUNITAR_MAXIM	
Pretul maxim=	1000, apare in factura 3111

Figura 6.75. Prețul maxim pentru Produs 1 și factura în care apare acest preț

Trebuie spus, însă, că în DB2 versiunea inițială a interogării – cea care folosește funcția CAST funcționează ireproșabil datorită faptului că, la conversie, implicit se completează cu zerouri nesemnificative – vezi figura 6.76:

```
SELECT MAX('Pretul maxim=' || CAST( LF.PretUnit AS CHAR(10)) ||
    ', apare in factura '|| CAST ( NrFact AS CHAR(10)))
AS Mesaj_cam_lung
FROM liniifact lf INNER JOIN produse p ON lf.CodPr = p.CodPr
WHERE DenPr = 'Produs 1'
```

MESAJ_CAM_LUNG
Pretul maxim=0001000.00, apare in factura 00003111.

Figura 6.76. Varianta DB2 de afișare a prețului maxim pentru Produs 1 și facturii în care apare acest preț

Care este cel mai mare și cel mai mic preț unitar (fără TVA) la care a fost vândut Produs 2 ?

Suntem plini ochi de know-how-ul de la problema anterioară, aşa că ne distrăm un pic (rezultatul fiind cel din figura 6.77):

```
SELECT MAX('Pret maxim=' || LPAD( TRIM (TRAILING FROM CAST(LF.PretUnit AS CHAR(12))), 12, ' ') || ', factura:' || CAST (NrFact AS CHAR(10))) AS Max_Pret_Produs_2, MIN ('Pret minim=' || LPAD( TRIM (TRAILING FROM CAST(LF.PretUnit AS CHAR(12))), 12, ' ') || ', factura:' || CAST (NrFact AS CHAR(10))) AS Min_Pret_Produs_2
FROM liniifact lf INNER JOIN produse p ON lf.CodPr = p.CodPr
WHERE DenPr = 'Produs 2'
```

MAX_PRET_PRODUS_2	MIN_PRET_PRODUS_2
Pret maxim= 1120, factura:1120	Pret minim= 925, factura:1115

Figura 6.77. Prețurile maxime și minime ale Produs 2 și facturile în care apar aceste prețuri

Din nou trebuie să precizăm că, în cazul în care cele două prețuri s-au înregistrat în mai multe facturi, interogarea nu indică decât una dintre acestea și, în plus, interogarea funcționează ca atare în PostgreSQL și Oracle.

Care sunt cele mai mari două prețuri unitare (fără TVA) la care a fost vândut Produs 2 ?

Cu remarca răuțacioasă legată de incompletitudinea rezultatului, profităm de problemă pentru a auto-jonctiona tabela LINIIFACT. Atenție la predicatul din INNER JOIN (avem un alt exemplu de theta-jonctiune) !

```
SELECT 'Produs 2: ' ||
       MAX('Primul PU: ' || TO_CHAR(LF1.PretUnit,'999999.99') || ', al doilea PU: ' || TO_CHAR(LF2.PretUnit,'99999999') ||
            ' - factura primului:' || LF1.NrFact || ', factura-al doilea:' || LF2.NrFact)
                        AS "Cele mai mari două PretUnit"
FROM liniifact LF1
      INNER JOIN liniifact LF2
        ON LF1.CodPr = LF2.CodPr AND LF1.PretUnit > LF2.PretUnit
      INNER JOIN produse P ON LF1.CodPr = P.CodPr AND DenPr = 'Produs 2'
```

Cele mai mari două PretUnit
Produs 2: Primul PU: 1120.00, al doilea PU: 1100 - factura primului:1120, factura-al doilea:3118

Figura 6.77. Cele mai mari două prețuri la care s-a vândut Produs 2 și facturile în care apar aceste prețuri

Acestei variante îi putem reproşa (ca multor altora din această carte) că este prea oraclisto-postgreslistă, aşa că putem să formulăm o variantă mai generală, executabilă și în DB2:

```
SELECT 'Produs 2: '|| MAX('Primul PU:'|| CAST (LF1.PretUnit AS CHAR(10)) ||
    ', al doilea PU:'|| CAST (LF2.PretUnit AS CHAR(10)) ||
    '|| ' - factura primului:'|| CAST (LF1.NrFact AS CHAR(8))|| |
    ', factura-al doilea:'|| CAST (LF2.NrFact AS CHAR(8)))
        AS "Cele mai mari două PretUnit"
FROM liniifact LF1
    INNER JOIN liniifact LF2
        ON LF1.CodPr = LF2.CodPr AND LF1.PretUnit > LF2.PretUnit
    INNER JOIN produse P ON LF1.CodPr = P.CodPr AND DenPr = 'Produs 2'
```

Care este situația încasării facturii 1117 ?

Reluăm ultima variantă de rezolvare a acestei probleme, introducând acum și funcția MAX:

```
SELECT '1117' AS NrFact,
    SUM(Cantitate * PretUnit * (1+ProcTVA)) / COUNT(DISTINCT i.CodInc)
        AS Facturat,
    SUM(Transa) / MAX(lf.Linie) AS Incasat
FROM liniifact lf
    INNER JOIN produse p ON lf.CodPr=p.CodPr
    INNER JOIN facturi f ON lf.NrFact=f.NrFact
    INNER JOIN incasfact i ON f.NrFact=i.NrFact
WHERE f.NrFact = 1117
```

