

# Capitolul 5

## SQL în Visual FoxPro

FoxPro a fost unul dintre primele SGBD-uri xBase care au implementat un important nucleu SQL. Visual FoxPro respectă, în continuare, multe din specificațiile standardului SQL-92, deși nu putem vorbi de un nivel similar marilor produse ale lumii SGBDR: Oracle, DB2, Sybase, Informix, SQL Server etc. Cu toate limitele sale, care vor fi punctate pe parcursul acestui capitol, prin posibilitatea includerii comenzilor SQL în programe VFP și folosirea cursorilor sau tabelilor virtuale, practic în Visual FoxPro se poate rezolva orice problemă de interogare și manipulare a datelor.

### 5.1. Principalele comenzi SQL din VFP

Comenzile SQL-92 implementate în Visual FoxPro 6 sunt prezentate în tabelul 6.1.

Tabelul 5.1. Comenzi SQL în Visual FoxPro

<i>Comandă</i>	<i>Scop</i>
Pentru manipularea datelor	
SELECT	Extragerea datelor din BD
INSERT	Adăugarea de noi linii într-o tabelă
DELETE	Ștergerea de linii dintr-o tabelă
UPDATE	Modificarea valorilor unor atribute
Pentru definirea bazei de date	
CREATE TABLE	Adăugarea unei noi table în BD
DROP TABLE	Ștergerea unei table din bază
ALTER TABLE	Modificarea structurii unei table
CREATE VIEW	Crearea unei table virtuale
DROP VIEW	Ștergerea unei table virtuale
CREATE CURSOR	Crearea unei table temporare
Pentru controlul tranzacțiilor	
BEGIN TRANSACTION	Marchează începutul unei tranzacții
END TRANSACTION	Marchează sfârșitul unei tranzacții
ROLLBACK	Abandonează tranzacția în curs

Din paragraful următor vom începe prezentarea comenzilor SQL cu opțiunile esențiale pentru crearea și modificarea structurii unei baze de date și a tabelilor acesteia – CREATE TABLE, ALTER TABLE, inclusiv a restricțiilor ce pot fi definite la nivel de atribut sau tabelă. Pentru popularea și actualizarea tabelilor vor fi prezentate clauzele comenzilor INSERT, UPDATE și DELETE.

## 5.2. Crearea și modificarea structurii tabelelor

Dacă în capitolul 2 au fost prezentate modalitățile asistate de creare a bazei de date, a tabelelor acesteia, inclusiv a restricțiilor, în acest capitol ne vom apropia de zona „profesională” a dezvoltării modulelor bazei de date în cadrul aplicațiilor VFP.

### 5.2.1. Crearea tabelelor și declararea restricțiilor

Comanda SQL utilizată pentru crearea unei tabele este `CREATE TABLE`, iar pentru modificarea ulterioară a structurii se folosește `ALTER TABLE`. Precizăm că este vorba de crearea structurii tablei. Popularea cu înregistrări și actualizarea ulterioară se realizează prin alte comenzi: `INSERT`, `UPDATE`, `DELETE`.

Formatul general al comenzii `CREATE TABLE` în Visual FoxPro este:

```
CREATE TABLE | DBF TableName [NAME LongTableName] [FREE]
    (FieldName1 FieldType [(nFieldWidth [, nPrecision])]
        [NULL | NOT NULL]
        [CHECK lExpression1 [ERROR cMessageText1]]
        [DEFAULT eExpression1]
        [PRIMARY KEY | UNIQUE]
        [REFERENCES TableName2 [TAG TagName1]]
        [NOCPTRANS]
    [, FieldName2 ...]
        [, PRIMARY KEY eExpression2 TAG TagName2
        |, UNIQUE eExpression3 TAG TagName3]
        [, FOREIGN KEY eExpression4 TAG TagName4 [NODUP]
            REFERENCES TableName3 [TAG TagName5]]
        [, CHECK lExpression2 [ERROR cMessageText2]])
| FROM ARRAY ArrayName
```

Iată, pe scurt, semnificația opțiunilor:

`CREATE TABLE | DBF TableName`

`TableName` reprezintă numele tablei ce urmează a fi creată. De remarcat că nu există nici o diferență între opțiunile `TABLE` și `DBF`. Prima se adresează celor atașați teoriei relaționale, iar cea de-a doua este mai aproape de utilizatorul „tradițional” al FoxPro (xBase-urilor, în general), pentru care o tabelă este un fișier cu extensia `.DBF`.

`NAME LongTableName`

Permite specificarea unui nume mai lung (până la 128 de caractere) pentru tabela creată. Pentru aceasta, este necesar ca baza de date să fie în prealabil deschisă, deoarece numele lungi sunt memorate în containerul asociat bazei (`.DBC`).

`FREE`

Indică faptul că tabela respectivă va fi independentă, deci nu va face parte din bază.

`(FieldName1 FieldType [(nFieldWidth [, nPrecision]])]`

Permite declararea, pentru fiecare atribut (câmp) al tabelii, a numelui, tipului, lungimii și, eventual, a numărului de poziții pentru reprezentarea părții fracționare (zecimale).

`NULL`

Prin specificarea sa, atributul este autorizat să conțină valori nule (`NULL`).

`NOT NULL`

Nu permite apariția valorilor nule pentru atributul respectiv. Automat, pentru atributele de tip cheie primară sau pentru care este utilizată opțiunea `UNIQUE` nu se admit valori `NULL`.

`CHECK lExpression1`

Servește la specificarea unei funcții-utilizator de validare la nivel de atribut (câmp)<sup>25</sup>. Astfel, funcția este verificată imediat după adăugarea unei noi înregistrări (linii) în tabelă. Dacă rezultatul evaluării funcției este `.F.` (`FALSE`), se declanșează o eroare.

`ERROR cMessageText1`

În cazul în care funcția de validare de la nivelul atributului nu se respectă (adică „întoarce” valoarea logică `FALSE`), pe ecran apare mesajul `cMessageText1`.

`DEFAULT eExpression1`

Specifică valoarea implicită a atributului (în funcție de tipul acestuia), valoare pe care o va conține acest câmp la adăugarea unei noi înregistrări în tabelă.

`PRIMARY KEY`

Declară atributul respectiv cheie primară, prin crearea unui index principal cu nume identic ca al atributului.

`UNIQUE`

Creează un „index candidat” – altfel spus, declară acest atribut cheie alternativă. Se previne astfel apariția a două valori identice în tabelă pentru acest câmp.

`REFERENCES TableName2 [TAG TagName1]`

Permite definirea unei restricții referențiale prin crearea unei legături permanente între tabele. `TableName2` este tabela-părinte a legăturii. Dacă nu se specifică `TAG TagName1`, relația este stabilită prin intermediul indexului primar al tabelii `TableName2`; dacă acesta (indexul primar) nu există, se generează un mesaj de eroare. `TableName2` nu poate fi o tabelă independentă.

`NOCPTRANS`

Este utilă pentru câmpuri de tip șir de caractere și memo, pentru a preveni conversia la un alt cod de pagină (*code page*).

---

25. Vezi capitolul 2, paragraful 2.5.1.

```
PRIMARY KEY eExpression2 TAG TagName2
```

Creează automat un index primar pentru tabela curentă. `eExpression2` este un câmp sau o combinație de câmpuri ale tabelului. Numele indexului poate conține până la 10 caractere. Firește, clauza `PRIMARY KEY` nu poate apărea decât o singură dată într-o comandă `CREATE TABLE`.

```
UNIQUE eExpression3 TAG TagName3
```

Creează un index candidat. `eExpression3` desemnează orice câmp sau combinație de câmpuri ale tabelului, atribut/combinatie care va îndeplini rolul de cheie alternativă. `TagName3` reprezintă numele indexului candidat care este alcătuit tot din maximum 10 caractere. Pentru o aceeași tabelă pot fi creați mai mulți indecși candidați.

```
FOREIGN KEY eExpression4 TAG TagName4 [NODUP]
```

Are ca rezultat crearea unui index obișnuit (*regular*) pentru tabela curentă (cheia de indexare fiind `eExpression4`, iar numele acestuia `TagName4`) și stabilirea unei relații permanente cu o tabelă-părinte. Prin `NODUP` se poate crea un index străin candidat (prin analogie cu indecșii candidați la „postul” de index primar).

```
REFERENCES TableName3 [TAG TagName5]
```

Specifică numele tabelului-părinte implicate în legătura creată prin opțiunea `FOREIGN KEY`. Indexul `TagName5` este cel prin care se realizează legătura dintre cele două tabele. Implicit este indexul primar al tabelului `TableName3`.

```
CHECK eExpression2 [ERROR cMessageText2]
```

Permite specificarea unei restricții (reguli de validare) la nivel de tabelă; `cMessageText2` este mesajul afișat în cazul nerespectării restricției.

```
FROM ARRAY ArrayName
```

Permite crearea unei tabele pe baza datelor conținute într-o variabilă de tip tablou.

În listingul 5.1 este prezentat programul pentru crearea, în VFP 6, a bazei de date VINZARI.

Listing 5.1. Crearea tabelor bazei de date în VFP 6

```
* In caz ca baza exista deja,  
* pentru crearea sa, trebuie inchisa  
CLOSE DATA ALL  
CLOSE TABLES ALL  
CLOSE ALL  
  
* se sterge BD impreuna cu tabelele care o alcatuiesc  
* Atentie !!! Comanda urmatoare distruge intreg continutul bazei !  
DELETE DATABASE vinzari DELETETABLES  
  
* A sosit vremea creatiei  
CREATE DATABASE vinzari
```

\* Secventa de creare a tabelelor

\* In tabela JUDETE

- \* - JUD este cheie primara
- \* - literele JUD trebuie sunt exclusiv majuscule
- \* - JUDET este cheie candidat (alternativa)
- \* - JUDET nu poate avea valori nule
- \* - prima litera din fiecare cuvint al denumirii judetului este obligatoriu majuscula
- \* - valoarea implicita a atributului REGIUNE este „MOLDOVA”
- \* - valoarea REGIUNE trebuie sa se incadreze in lista: „MOLDOVA”, „Banat”,....

```
CREATE TABLE judete ( ;
    jud CHAR(2) ;
        PRIMARY KEY ;
        CHECK (jud=LTRIM(UPPER(jud))) ;
        ERROR 'Indicativul judetului se scrie cu majuscule !', ;
    judet CHAR(25) ;
        UNIQUE ;
        NOT NULL ;
        CHECK (judet=LTRIM(PROPER(judet))) ;
        ERROR 'Prima litera din fiecare cuvint al'+CHR(13)+';
        'denumirii judetului este majuscula; '+CHR(13)+';
        'restul literelor sunt mici!'; ;
    regiune CHAR(15) ;
        DEFAULT 'Moldova' ;
        CHECK (INLIST(regiune, 'Banat', 'Transilvania', 'Dobrogea', 'Oltenia', ;
        'Muntenia', 'Moldova')) ;
        ERROR 'Regiunea poate avea o valoare numai din lista:'+CHR(13)+';
        'Banat, Transilvania, Dobrogea, Oltenia, Muntenia, Moldova' ;
)
```

\* In tabela LOCALITATI

- \* - ...
- \* - atributul JUD este cheie straina (parintele fiind tabela JUDETE)

```
CREATE TABLE localitati ( ;
    codpost CHAR(5) ;
        PRIMARY KEY ;
        CHECK (codpost=LTRIM(codpost)) ;
        ERROR 'Codul postal se scrie fara spatii la inceput !', ;
    loc CHAR(25) ;
        NOT NULL ;
        CHECK (loc=LTRIM(PROPER(loc))) ;
        ERROR 'Prima litera din fiecare cuvint al'+CHR(13)+';
        'denumirii localitatii este majuscula; '+CHR(13)+';
        'restul literelor sunt mici!'; ;
    jud CHAR(2) ;
        DEFAULT 'IS' ;
    FOREIGN KEY jud TAG jud REFERENCES judete TAG jud ;
)
```

```
CREATE TABLE clienti ( ;
    codcl NUMBER(6) ;
        PRIMARY KEY ;
        CHECK (codcl > 1000), ;
    dencl CHAR(30) ;
        CHECK (SUBSTR(dencl,1,1) = UPPER(SUBSTR(dencl,1,1))) ;
        ERROR 'Prima litera din denumirea clientului este ; '+CHR(13)+;
        'obligatoriu majuscula !', ;
    codfiscal CHAR(9) ;
        NULL ;
        CHECK (SUBSTR(codfiscal,1,1) = UPPER(SUBSTR(codfiscal,1,1))) ;
        ERROR 'Prima (eventuala) litera din codul fiscal ; '+CHR(13)+;
        'este obligatoriu majuscula !', ;
    adresa CHAR(40) ;
        NULL ;
        CHECK (SUBSTR(adresa,1,1) = UPPER(SUBSTR(adresa,1,1))) ;
        ERROR 'Prima (eventuala) litera din adresa ; '+CHR(13)+;
        'este obligatoriu majuscula !', ;
    codpost CHAR(5), ;
    telefon CHAR(10) ;
        NULL, ;
    FOREIGN KEY codpost TAG codpost REFERENCES localitati TAG codpost ;
) ;

CREATE TABLE persoane ( ;
    cnp CHAR(14) ;
        PRIMARY KEY ;
        CHECK (cnp=LTRIM(UPPER(cnp))) ;
        ERROR 'Codul numeric personal se scrie fara spatii la inceput !', ;
    nume CHAR(20) ;
        CHECK (nume=LTRIM(PROPER(nume))) ;
        ERROR 'Prima litera din fiecare cuvint al'+CHR(13)+;
        'numelui este majuscula; '+CHR(13)+;
        'restul literelor sunt mici!', ;
    prenume CHAR(20) ;
        CHECK (prenume=LTRIM(PROPER(prenume))) ;
        ERROR 'Prima litera din fiecare cuvint al'+CHR(13)+;
        'prenumelui este majuscula; '+CHR(13)+;
        'restul literelor sunt mici!', ;
    adresa CHAR(40) ;
        NULL ;
        CHECK (SUBSTR(adresa,1,1) = UPPER(SUBSTR(adresa,1,1))) ;
        ERROR 'Prima litera din adresa este obligatoriu majuscula !', ;
    sex CHAR(1) DEFAULT 'B' ;
        CHECK (INLIST(sex,'F','B')) ;
        ERROR 'Atributul Sex poate avea valorile F (de la Femeiesc)'+CHR(13)+;
        'sau B (de la Barbatesc) !', ;
    codpost CHAR(5), ;
    telacasa CHAR(10) NULL, ;
    telbirou CHAR(10) NULL, ;
    telemobil CHAR(10) NULL, ;
    email CHAR(20) NULL, ;
```

```

FOREIGN KEY codpost TAG codpost REFERENCES localitati TAG codpost :
);

* In tabela PERSCLIENTI
* - cheia primara este compusa: (CNP, CodCl)
CREATE TABLE perscienti ( ;
    cnp CHAR(14), ;
    codcl NUMBER(6), ;
    functie CHAR(25) ;
    CHECK (SUBSTR(functie,1,1) = UPPER(SUBSTR(functie,1,1))) ;
    ERROR 'Prima litera din functie este obligatoriu majuscula !', ;
    PRIMARY KEY cnp+STR(codcl,6)+functie TAG primaru, ;
    FOREIGN KEY cnp TAG cnp REFERENCES persoane TAG cnp, ;
    FOREIGN KEY codcl TAG codcl REFERENCES clienti TAG codcl ;
);

* Tabela PRODUSE
* - contine atributul IMAGE de tip GENERAL
CREATE TABLE produse ( ;
    codpr NUMBER(6) ;
    PRIMARY KEY ;
    CHECK (codpr > 0) ;
    ERROR 'Codul produsului trebuie sa fie mai mare de 100000 !', ;
    denpr CHAR(30) ;
    CHECK (SUBSTR(denpr,1,1) = UPPER(SUBSTR(denpr,1,1))) ;
    ERROR 'Prima litera din denumirea produsului este obligatoriu majuscula !', ;
    um CHAR(10), ;
    grupa CHAR(15) ;
    CHECK (SUBSTR(grupa,1,1) = UPPER(SUBSTR(grupa,1,1))) ;
    ERROR 'Prima litera din grupa este obligatoriu majuscula !', ;
    proctVA NUMBER(3,2) ;
    DEFAULT .22, ;
    imagine GENERAL ;
);

CREATE TABLE gestiuni ( ;
    gestiune CHAR(4) ;
    PRIMARY KEY, ;
    den_gest CHAR(25) ;
    CHECK (SUBSTR(den_gest,1,1) = UPPER(SUBSTR(den_gest,1,1))) ;
    ERROR 'Prima litera din denumirea gestiunii este obligatoriu majuscula !', ;
    adresa CHAR(40) ;
    NULL ;
    CHECK (SUBSTR(adresa,1,1) = UPPER(SUBSTR(adresa,1,1))) ;
    ERROR 'Prima litera din adresa este obligatoriu majuscula !', ;
    codpost CHAR(5), ;
    telefon CHAR(10) ;
    NULL, ;
    cnp CHAR(14), ;
    email CHAR(20) NULL, ;

```

```
FOREIGN KEY codpost TAG codpost REFERENCES localitati TAG codpost, ;
FOREIGN KEY cnp TAG cnp REFERENCES persoane TAG cnp ;
)
```

\* In tabela FACTURI

- \* - valorile atributului DATAFACT trebuie sa se incadreze
- \* in intervalul 1 ian.2001 - 31 dec.2010

```
CREATE TABLE facturi ( ;
  nrfact NUMBER(8) ;
  PRIMARY KEY, ;
  datafact DATE ;
  DEFAULT DATE() ;
  CHECK (BETWEEN(datafact,{^2001/01/01},{^2010/12/31})); ;
  ERROR 'Baza de date functioneaza in intervalul 1 ian 2001 - 31 dec.2010 !', ;
  gestiune CHAR(4) NOT NULL, ;
  codcl NUMBER(6), ;
  obs CHAR(50) NULL, ;
  FOREIGN KEY gestiune TAG gsetiune REFERENCES gestiuni TAG gestiune, ;
  FOREIGN KEY codcl TAG codcl REFERENCES clienti TAG codcl ;
) ;
```

CREATE TABLE liniifact ( ;

```
  nrfact NUMBER(8), ;
  linie NUMBER(2) ;
  CHECK (linie > 0) ;
  ERROR 'Atributul linie trebuie sa fie mai mare ca zero !', ;
  codpr NUMBER(6), ;
  cantitate NUMBER(10), ;
  pretunit NUMBER (12), ;
  PRIMARY KEY STR(nrfact,8)+STR(linie,2) TAG primaru, ;
  FOREIGN KEY nrfact TAG nrfact REFERENCES facturi TAG nrfact, ;
  FOREIGN KEY codpr TAG codpr REFERENCES produse TAG codpr ;
)
```

\* In tabela INCASARI

- \* - valorile atributelor DATAINC si DATADOC trebuie sa se incadreze
- \* in intervalul 1 ian.2001 - 31 dec.2010
- \* - valoarea implicita a atributului DATAINC este data curenta (data sistemului)
- \* - valoarea implicita a atributului DATADOC este data curenta (data sistemului)
- \* minus 5; aceasta inseamna ca, de obicei, intre emiterea documentului
- \* de plata si data in care banii sosesc in cont trec 5 zile (e doar o presupunere !)

```
CREATE TABLE incasari ( ;
  codinc NUMBER(8) ;
  PRIMARY KEY, ;
  datainc DATE ;
  DEFAULT DATE() ;
  CHECK (BETWEEN(datainc,{^2001/01/01},{^2010/12/31})); ;
  ERROR 'Baza de date functioneaza in intervalul 1 ian.2001 - 31 dec.2010 !', ;
  coddoc CHAR(4) ;
  CHECK(coddoc=UPPER(LTRIM(coddoc))) ;
  ERROR 'Codul documentului se scrie cu majuscule !', ;
  nrdoc CHAR(16), ;
```



```

datadoc DATE :
    DEFAULT DATE() - 5 ;
    CHECK (BETWEEN(datadoc, {'^2001/01/01'}, {'^2001/12/31'})) ;
    ERROR 'Data documentului trebuie sa fie intre 1 ian.2000 si 31 dec.2010 !' ; ) ;

CREATE TABLE incasfact ( ;
    codinc NUMBER(8), ;
    nrfact NUMBER(8), ;
    transa NUMBER(16) ;
    NOT NULL, ;
    PRIMARY KEY STR(codinc,8)+STR(nrfact,8) TAG primaru, ;
    FOREIGN KEY codinc TAG codinc REFERENCES incasari TAG codinc, ;
    FOREIGN KEY nrfact TAG nrfact REFERENCES facturi TAG nrfact ;
) ;

```

În VFP este obligatoriu să se indice explicit ce atribute *pot* avea valori nule, iar la declararea cheilor primare compuse, atributele componente trebuie concatenate, lucru ce atrage necesitatea conversiei celor de alt tip (numerice, dată calendaristică, logice) în șiruri de caractere. Declararea cheilor primare, alternative și străine în VFP presupune crearea automată a indecșilor de tip PRIMARY, CANDIDATE sau REGULAR. Prin clauza TAG se poate da indexului respectiv un nume la alegere.

Din păcate, declararea cheilor străine nu înseamnă și instituirea restricției referențiale. Aceasta este una dintre cele mai ciudate „găselnițe” VFP. Ca urmare, după crearea BD, fie trebuie „umblat” prin *Referential Integrity Builder* și, astfel, grafic, instituite regulile pentru prezervarea referențialității (vezi capitolul 2), fie trebuie create declanșatoare în acest scop (capitolul 6).

### 5.2.2. Modificarea structurii tabelor/restricțiilor în VFP

Schema unei baze de date reprezintă aspectul constant, invariabil sau, mai bine zis, puțin variabil în timp. O bună analiză și proiectare a aplicației (sistemului) diminuează riscul modificărilor de amploare în structura tabelor și restricțiilor, conferind stabilitate bazei și diminuând sensibil eforturile de întreținere ulterioară instalării aplicației. Cu toate acestea, pe parcursul utilizării bazei de date pot apărea situații în care este necesară adăugarea unui nou atribut, ștergerea unui atribut inutil, modificarea lungimii etc.

Comanda pentru modificarea structurii unei tabele este ALTER TABLE, ale cărei opțiuni mai importante le vom parcurge în continuare.

#### *Adăugarea unui nou atribut*

Adăugarea atributului DataNast (data nașterii) în tabela PERSOANE se realizează astfel:

```
ALTER TABLE PERSOANE ADD DataNast DATE
```

#### *Ștergerea unui atribut*

Dacă ne propunem să ștergem atributul pe care tocmai l-am adăugat, trebuie doar să folosim comanda:

```
ALTER TABLE PERSOANE DROP COLUMN DataNast
```

**Redenumirea unui atribut**

Spre deosebire chiar de SGBD-uri consacrate, în VFP un atribut se poate și redenumi:

```
ALTER TABLE PERSOANE RENAME COLUMN DataNast TO DataNasterii
```

**Modificarea tipului/ lungimii unui atribut**

Dacă se dorește creșterea la 21 a numărului de caractere alocate valorilor atributului

Prenume și reducerea la 17 a lungimii atributului Nume, vom utiliza:

```
ALTER TABLE PERSOANE ALTER COLUMN Prenume CHAR(21)
```

```
ALTER TABLE PERSOANE ALTER COLUMN Nume CHAR(17)
```

**Adăugarea/modificarea valorii implicite**

Dacă dorim ca valoarea implicită a atributului Sex să fie 'F' (deoarece sunt mai multe femei decât bărbați), acest lucru este posibil și astfel:

```
ALTER TABLE PERSOANE ALTER COLUMN Sex SET DEFAULT 'F'
```

**NULL și neNULL**

Interzicerea valorilor nule pentru atributul Sex se realizează astfel:

```
ALTER TABLE PERSOANE ALTER COLUMN Sex NOT NULL
```

Invers, pentru a permite valori NULL pentru același atribut:

```
ALTER TABLE PERSOANE ALTER COLUMN Sex NULL
```

**Adăugarea/anularea restricțiilor**

Toate restricțiile: cheie primară – PRIMARY KEY, unicitate – UNIQUE, referențială – FOREIGN KEY, de comportament – CHECK pot fi declarate ulterior creării tablei și, bineînțeles, anulate la un moment dat. Spre exemplu, formatul general al comenzii pentru dezactivarea cheii primare este:

```
ALTER TABLE PERSOANE DROP PRIMARY KEY
```

Pentru reinstituirea cheii primare a tablei PERSOANE comanda are forma:

```
ALTER TABLE PERSOANE ADD PRIMARY KEY (CNP)
```

Analog, prin ADD și DROP pot fi instituite/anulate și celelalte tipuri de restricții, formatele generale ale comenzii ALTER TABLE fiind:

```
ALTER TABLE TableName1
ADD | ALTER [COLUMN] FieldName1
FieldType [(nFieldWidth [, nPrecision])]
[NULL | NOT NULL]
[CHECK lExpression1 [ERROR cMessageText1]]
[DEFAULT eExpression1]
[PRIMARY KEY | UNIQUE]
[REFERENCES TableName2 [TAG TagName1]]
[NOCPTRANS]
[NOVALIDATE]
```

sau

```
ALTER TABLE TableName1
ALTER [COLUMN] FieldName2
[NULL | NOT NULL]
[SET DEFAULT eExpression2]
[SET CHECK lExpression2 [ERROR cMessageText2]]
[DROP DEFAULT]
[DROP CHECK]
[NOVALIDATE]
```

sau

```
ALTER TABLE TableName1
[DROP [COLUMN] FieldName3]
[SET CHECK lExpression3 [ERROR cMessageText3]]
[DROP CHECK]
[ADD PRIMARY KEY eExpression3 TAG TagName2 [FOR
lExpression4]]
[DROP PRIMARY KEY]
[ADD UNIQUE eExpression4 [TAG TagName3 [FOR lExpression5]]]
[DROP UNIQUE TAG TagName4]
[ADD FOREIGN KEY [eExpression5] TAG TagName4 [FOR
lExpression6]
REFERENCES TableName2 [TAG TagName5]]
[DROP FOREIGN KEY TAG TagName6 [SAVE]]
[RENAME COLUMN FieldName4 TO FieldName5]
[NOVALIDATE]
```

Unul dintre elementele de noutate față de formatul comenzii CREATE TABLE îl reprezintă apariția clauzei NOVALIDATE, prin prezența căreia se permite operarea unor modificări ale structurii chiar dacă acestea ar viola integritatea datelor din bază.

### 5.2.3. Ștergerea tabelor

Comanda de ștergere a unei tabele din bază este DROP TABLE și are o sintaxă extrem de simplă:

```
DROP TABLE tabelă
```

## 5.3. Comenzi de actualizare

Nucleul SQL din Visual FoxPro prezintă toate cele trei comenzi pentru actualizarea conținutului unei tabele: INSERT, UPDATE și DELETE.

### 5.3.1. Adăugarea unei linii

Comanda SQL de adăugare de noi linii este INSERT, care are, în modul de lucru cel mai puțin pretențios, următorul format:

```
INSERT INTO tabelă [(atribut1, atribut2, ...)]  
VALUES (valoare_atribut1, valoare_atribut2, ...)
```

Ordinea valorilor din clauza VALUES trebuie să fie identică cu cea declarată la crearea (sau modificarea structurii) tabelului. Modificarea este posibilă numai prin enumerarea, după numele tabelului, a atributelor ce vor primi valorile specificate.

Dacă, spre exemplu, pentru tabelele CLIENTI și FACTURI au fost incluse în clauza VALUES mai puține valori decât atributele tabelului, VFP va genera o eroare. Este obligatorie, în aceste cazuri, precizarea atributelor care vor primi valorile (vezi listingul 5.2). În Oracle și DB2, restul, adică cele nespecificate, vor avea, pe liniile respective, valori NULL. În VFP, acestea sunt completate cu valori vide (funcția EMPTY() întoarce .T., în timp ce ISNULL() - .F.).

### 5.3.2. Ștergerea liniilor

Comanda SQL pentru ștergerea uneia sau mai multor linii dintr-o tabelă este DELETE. Formatul general este:

```
DELETE FROM nume-tabelă WHERE predicat
```

În tabelă vor fi *marcate pentru ștergere* toate liniile care îndeplinesc condiția specificată în predicatul din clauza WHERE. Firește, când clauza WHERE lipsește, vor fi marcate pentru ștergere toate liniile tabelului.

#### Exemplul 1

*Să se elimine din tabela FACTURI toate facturile emise de gestiunea 002.*

```
DELETE FROM FACTURI ;  
WHERE gestiune = '002'
```

#### Exemplul 2

*Să se elimine din baza de date toate județele din Moldova.*

```
DELETE FROM JUDETE ;  
WHERE Regiune = 'Moldova'
```

### 5.3.3. Popularea bazei de date prin comenzi SQL

Folosind comenzile de ștergere și inserare, în listingul 5.2 este prezentat programul pentru popularea bazei de date. La început se folosesc comenzile de ștergere, pentru a preveni eventuala duplicare a înregistrărilor.

Listing 5.2. Script Visual FoxPro de populare a tabelului (cu valorile din capitolul 2)

```
* Program de populare a BAZEI DE DATE  
*  
* se deschide baze de date (in caz ca  
* nu e deja deschisa  
IF !DBUSED('vinzari')  
OPEN DATABASE vinzari EXCLUSIVE  
ENDIF
```

```
DELETE FROM incasfact
SELE incasfact
ZAP
```

```
DELETE FROM incasari
SELE incasari
ZAP
```

```
DELETE FROM liniifact
SELE liniifact
ZAP
```

```
DELETE FROM facturi
SELE facturi
ZAP
```

```
DELETE FROM gestiuni
SELE gestiuni
ZAP
```

```
DELETE FROM produse
SELE produse
ZAP
```

```
DELETE FROM perscianti
SELE perscianti
ZAP
```

```
DELETE FROM persoane
SELE persoane
ZAP
```

```
DELETE FROM clienti
SELE clienti
ZAP
```

```
DELETE FROM localitati
SELE localitati
ZAP
```

```
DELETE FROM judete
SELE judete
ZAP
```

```
INSERT INTO judete VALUES ('IS', 'Iasi', 'Moldova')
INSERT INTO judete VALUES ('VN', 'Vrancea', 'Moldova')
INSERT INTO judete VALUES ('NT', 'Neamt', 'Moldova')
INSERT INTO judete VALUES ('SV', 'Suceava', 'Moldova')
INSERT INTO judete VALUES ('VS', 'Vaslui', 'Moldova')
INSERT INTO judete VALUES ('TM', 'Timis', 'Banat')
```

```
INSERT INTO localitati VALUES ('6600', 'Iasi', 'IS')
INSERT INTO localitati VALUES ('5725', 'Pascani', 'IS')
INSERT INTO localitati VALUES ('6500', 'Vaslui', 'VS')
INSERT INTO localitati VALUES ('5300', 'Focsani', 'VN')
INSERT INTO localitati VALUES ('6400', 'Birlad', 'VS')
INSERT INTO localitati VALUES ('5800', 'Suceava', 'SV')
INSERT INTO localitati VALUES ('5550', 'Roman', 'NT')
INSERT INTO localitati VALUES ('1900', 'Timisoara', 'TM')
```

```

INSERT INTO clienti VALUES (1001, 'Client 1 SRL', 'R1001', 'Tranzitiei, 13 bis', ;
'6600', NULL)
INSERT INTO clienti (codcl, dencl, codfiscal, codpost, telefon) ;
VALUES (1002, 'Client 2 SA', 'R1002', '6600', '032-212121')
INSERT INTO clienti VALUES (1003, 'Client 3 SRL', 'R1003', 'Prosperitatii, 22', ;
'6500', '035-222222')
INSERT INTO clienti (codcl, dencl, adresa, codpost) ;
VALUES (1004, 'Client 4', 'Sapientei, 56', '5725')
INSERT INTO clienti VALUES (1005, 'Client 5 SRL', 'R1005', NULL, ;
'1900', '056-111111')
INSERT INTO clienti VALUES (1006, 'Client 6 SA', 'R1006', 'Pacientei, 33', ;
'5550', NULL)
INSERT INTO clienti VALUES (1007, 'Client 7 SRL', 'R1007', 'Victoria Capitalismului, 2', ;
'1900', '056-121212')

INSERT INTO persoane VALUES ('CNP1', 'Ioan', 'Vasile', 'I.L.Caragiale, 22', 'B', ;
'6600', '123456', '987654', '094222222', NULL)
INSERT INTO persoane VALUES ('CNP2', 'Vasile', 'Ion', NULL, 'B', ;
'6600', '234567', '876543', '094222223', 'Ion@a.ro')
INSERT INTO persoane VALUES ('CNP3', 'Popovici', 'Ioana', 'V.Micle, Bl.I, Sc.B,Ap.2', 'F', ;
'5725', '345678', NULL, '094222224', NULL)
INSERT INTO persoane VALUES ('CNP4', 'Lazar', 'Caraion', 'M.Eminescu, 42', 'B', ;
'6500', '456789', NULL, '094222225', NULL)
INSERT INTO persoane VALUES ('CNP5', 'Iurea', 'Simion', 'I.Creanga, 44 bis', 'B', ;
'6500', '567890', '543210', NULL, NULL)
INSERT INTO persoane VALUES ('CNP6', 'Vasc', 'Simona', 'M.Eminescu, 13', 'F', ;
'5725', NULL, '432109', '094222227', NULL)
INSERT INTO persoane VALUES ('CNP7', 'Popa', 'Ioanid', 'I.Ion, Bl.H2, Sc.C, Ap.45', 'B', ;
'1900', '789012', '321098', NULL, NULL)
INSERT INTO persoane VALUES ('CNP8', 'Bogacs', 'Ildiko', 'I.V.Viteazu, 67', 'F', ;
'5550', '890123', '210987', '094222229', NULL)
INSERT INTO persoane VALUES ('CNP9', 'Ioan', 'Vasilica', 'Garii, Bl.B4, Sc.A, Ap.1', 'F', ;
'1900', '901234', '109876', '094222230', NULL)
INSERT INTO persoane VALUES ('CNP10', 'Tucaliuc', 'Simion', NULL, 'B', ;
'6600', '222222', '212121', NULL, 'simion@mail.dntis.ro')
INSERT INTO persoane VALUES ('CNP11', 'Vasiliiu', 'Mihai', 'Stefan cel Mare, 34', 'B', ;
'5800', '258965', '256985', '093555555', 'mihai@assist.ro')
INSERT INTO persoane VALUES ('CNP12', 'Ciubotaru', 'Toader', 'Carpati, 12', 'B', ;
'5725', '454545', '254785', NULL, NULL)

INSERT INTO perscienti VALUES ('CNP1', 1001, 'Director general')
INSERT INTO perscienti VALUES ('CNP2', 1002, 'Director general')
INSERT INTO perscienti VALUES ('CNP3', 1002, 'Sef aprovizionare')
INSERT INTO perscienti VALUES ('CNP4', 1003, 'Sef aprovizionare')
INSERT INTO perscienti VALUES ('CNP5', 1003, 'Director financiar')
INSERT INTO perscienti VALUES ('CNP6', 1004, 'Director general')
INSERT INTO perscienti VALUES ('CNP7', 1005, 'Sef aprovizionare')
INSERT INTO perscienti VALUES ('CNP8', 1006, 'Director financiar')
INSERT INTO perscienti VALUES ('CNP9', 1007, 'Sef aprovizionare')

```

\* cimpurile GENERAL nu accepta valori NULL

```

INSERT INTO produse (codpr, denpr, um, grupa, proctva) VALUES (1, 'Produs 1', 'buc', 'Tigari', .19)
INSERT INTO produse (codpr, denpr, um, grupa, proctva) VALUES (2, 'Produs 2', 'kg', 'Bere', 0.19)
INSERT INTO produse (codpr, denpr, um, grupa, proctva) VALUES (3, 'Produs 3', 'kg', 'Bere', 0.19)
INSERT INTO produse (codpr, denpr, um, grupa, proctva) VALUES (4, 'Produs 4', 'l', 'Dulciuri', .19)
INSERT INTO produse (codpr, denpr, um, grupa, proctva) VALUES ;
(5, 'Produs 5', 'buc', 'Tigari', .19)

```

\* popularea cu poze. Din pacate nu exista o comanda SQL in acest sens

```
SELECT produse
```

```
GO TOP
```

```
nNrFisiereTIF = ADIR(aTIFuri, ".tif")
```

```
IF nNrFisiereTIF > 0
```

```
    FOR i = 1 to nNrFisiereTIF
```

```
        APPEND GENERAL Image FROM aTIFuri(i,1)
```

```
        SKIP
```

```
    ENDFOR
```

```
ENDIF
```

```
INSERT INTO gestiuni VALUES ('001', 'Depozit Pacurari', 'Sos. Pacurari, nr 145 bis', '6600', ;
```

```
'CNP10', '212121', 'x.gest001@mail.dntis.ro')
```

```
INSERT INTO gestiuni VALUES ('002', 'Depozit Suceava', 'Str. Lunetistului nr.33', '5800', ;
```

```
'CNP11', '256985', 'x.gest002@mail.dntis.ro')
```

```
INSERT INTO gestiuni VALUES ('003', 'Sectie Pascani', 'Bd. Trompetistului, 56', '5725', ;
```

```
'CNP12', '254785', 'x.gest003@mail.dntis.ro')
```

```
INSERT INTO facturi (nrfact, datafact, gestiune, codcl) VALUES (1111, {'^2001/08/01'}, '001', 1001)
```

```
INSERT INTO facturi VALUES (1112, {'^2001/08/01'}, '001', 1005, 'Probleme cu transportul')
```

```
INSERT INTO facturi (nrfact, datafact, gestiune, codcl) VALUES (1113, {'^2001/08/01'}, '001', 1002)
```

```
INSERT INTO facturi (nrfact, datafact, gestiune, codcl) VALUES (1114, {'^2001/08/01'}, '002', 1006)
```

```
INSERT INTO facturi (nrfact, datafact, gestiune, codcl) VALUES (1115, {'^2001/08/02'}, '002', 1001)
```

```
INSERT INTO facturi VALUES (1116, {'^2001/08/02'}, '001', 1007, ;
```

```
'Pretul propus initial a fost modificat')
```

```
INSERT INTO facturi VALUES (1117, {'^2001/08/03'}, '002', 1001, NULL)
```

```
INSERT INTO facturi VALUES (1118, {'^2001/08/04'}, '003', 1001, NULL)
```

```
INSERT INTO facturi VALUES (1119, {'^2001/08/07'}, '003', 1003, NULL)
```

```
INSERT INTO facturi VALUES (1120, {'^2001/08/07'}, '003', 1001, NULL)
```

```
INSERT INTO facturi VALUES (1121, {'^2001/08/07'}, '001', 1004, NULL)
```

```
INSERT INTO facturi VALUES (1122, {'^2001/08/07'}, '003', 1005, NULL)
```

```
INSERT INTO liniifact VALUES (1111, 1, 1, 50, 10000)
```

```
INSERT INTO liniifact VALUES (1111, 2, 2, 75, 10500)
```

```
INSERT INTO liniifact VALUES (1111, 3, 5, 500, 6500)
```

```
INSERT INTO liniifact VALUES (1112, 1, 2, 80, 10300)
```

```
INSERT INTO liniifact VALUES (1112, 2, 3, 40, 7500)
```

```
INSERT INTO liniifact VALUES (1113, 1, 2, 100, 9750)
```

```
INSERT INTO liniifact VALUES (1114, 1, 2, 70, 10700)
```

```
INSERT INTO liniifact VALUES (1114, 2, 4, 30, 15800)
```

```
INSERT INTO liniifact VALUES (1114, 3, 5, 700, 6400)
```

```
INSERT INTO liniifact VALUES (1115, 1, 2, 150, 9250)
```

```
INSERT INTO liniifact VALUES (1116, 1, 2, 125, 9300)
```

```
INSERT INTO liniifact VALUES (1117, 1, 2, 100, 10000)
```

```
INSERT INTO liniifact VALUES (1117, 2, 1, 100, 9500)
```

```
INSERT INTO liniifact VALUES (1118, 1, 2, 30, 11000)
```

```
INSERT INTO liniifact VALUES (1118, 2, 1, 150, 9300)
```

```
INSERT INTO liniifact VALUES (1119, 1, 2, 35, 10900)
```

```
INSERT INTO liniifact VALUES (1119, 2, 3, 40, 7000)
```

```
INSERT INTO liniifact VALUES (1119, 3, 4, 50, 14000)
```

```
INSERT INTO liniifact VALUES (1119, 4, 5, 750, 6300)
```

```
INSERT INTO liniifact VALUES (1120, 1, 2, 80, 11200)
```

```
INSERT INTO liniifact VALUES (1121, 1, 5, 550, 6400)
```

```
INSERT INTO liniifact VALUES (1121, 2, 2, 100, 10500)
```

```

INSERT INTO incasari VALUES (1234, {'^2001/08/15'}, 'OP', '111', {'^2000/08/10'})
INSERT INTO incasari VALUES (1235, {'^2001/08/15'}, 'CHIT', '222', {'^2000/08/15'})
INSERT INTO incasari VALUES (1236, {'^2001/08/16'}, 'OP', '333', {'^2000/08/09'})
INSERT INTO incasari VALUES (1237, {'^2001/08/17'}, 'CEC', '444', {'^2000/08/10'})
INSERT INTO incasari VALUES (1238, {'^2001/08/17'}, 'OP', '555', {'^2000/08/10'})
INSERT INTO incasari VALUES (1239, {'^2001/08/18'}, 'OP', '666', {'^2000/08/11'})

```

```

INSERT INTO incasfact VALUES (1234, 1111, 5399625)
INSERT INTO incasfact VALUES (1234, 1118, 1026375)
INSERT INTO incasfact VALUES (1235, 1112, 487705)
INSERT INTO incasfact VALUES (1236, 1117, 975410)
INSERT INTO incasfact VALUES (1236, 1118, 1026375)
INSERT INTO incasfact VALUES (1236, 1120, 731557)
INSERT INTO incasfact VALUES (1237, 1117, 975410)
INSERT INTO incasfact VALUES (1238, 1113, 1160250)
INSERT INTO incasfact VALUES (1239, 1117, 369680)

```

Am pornit de la presupunerea că în directorul curent există câte un fișier grafic, cu extensia .TIF pentru fiecare produs, iar ordinea acestora coincide cu ordinea produselor din tabelă, ceea ce e destul de riscant. Astfel încât procedura din listingul 5.3 este mai indicată.

Listing 5.3. Noua secvență pentru popularea cu imagini a câmpurilor de tip General

```

SELECT produse
SCAN
  nume_ = 'figura_'+LTRIM(STR(codpr,6))+'.TIF'
  IF FILE((nume_))
    APPEND GENERAL Image FROM (nume_) LINK
  ENDIF
ENDSCAN

```

### 5.3.4. Modificarea valorilor unor atribute

Pentru a modifica valoarea unuia sau mai multor atribute pe una sau mai multe linii dintr-o tabelă se folosește comanda UPDATE cu formatul general (simplificat):

```

UPDATE tabelă ;
SET atribut1 = expresie1 [, atribut2= expresie2 ....] ;
WHERE predicat

```

Modificarea se va produce pe toate liniile tabelii care îndeplinesc condiția formulată prin predicat.

#### Exemplul 3

*Noul număr de telefon al clientului ce are codul 1001 este 032-313131. Să se opereze modificarea în baza de date.*

Se actualizează atributul Telefon din tabela CLIENTI:

```

UPDATE CLIENTI ;
SET Telefon = '032-313131' ;
WHERE CodCl = 1001

```



**Exemplul 4**

*În cadrul unei noi relaxări fiscale, se decide creșterea procentului TVA de la 19% la 22% pentru toate produsele.*

Atributul modificat este ProcTVA din tabela PRODUSE, pe toate liniile:

```
UPDATE PRODUSE ;  
SET ProcTVA = .22 ;
```

## 5.4. Interogarea bazelor de date – fraza SELECT

Baza de date prezentată pe parcursul capitolului 2 este doar puțin diferită de cea din cartea *SQL. Dialecte DB2, Oracle și Visual FoxPro* (Polirom, Iași, 2001), așa încât nu mai reluăm decât câteva exemple.

### 5.4.1. Sintaxa de bază

Comanda SELECT respectă în VFP sintaxa de bază din standardul SQL-92, cele trei clauze principale fiind SELECT, FROM și WHERE, la care se adaugă ORDER BY, GROUP BY, HAVING și alte elemente legate de subconsultări. De asemenea, VFP folosește notația SQL-92 în materie de reprezentare a joncțiunii interne (INNER JOIN) și externe (OUTER JOIN). Câteva exemple sunt grăitoare în acest sens.

**Exemplul 1**

*Presupunând că plata se face la 14 zile, care sunt facturile ce urmează a fi încasate în perioada 15-18 august 2001 ?*

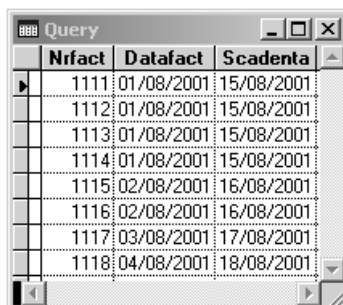
- Soluția 1.1:

```
SELECT DISTINCT NrFact, DataFact, DataFact + 14 AS Scadenta ;  
FROM FACTURI ;  
WHERE DataFact + 14 >= {^2001/08/15} AND ;  
      DataFact + 14 <= {^2001/08/18}
```

- Soluția 1.2 – folosind operatorul BETWEEN:

```
SELECT DISTINCT NrFact, DataFact, DataFact + 14 AS Scadenta ;  
FROM FACTURI ;  
WHERE DataFact + 14 BETWEEN {^2001/08/15} AND {^2001/08/18}
```

Indiferent de variantă, rezultatul este cel din figura 5.1.



	Nrfact	Datafact	Scadenta
▶	1111	01/08/2001	15/08/2001
	1112	01/08/2001	15/08/2001
	1113	01/08/2001	15/08/2001
	1114	01/08/2001	15/08/2001
	1115	02/08/2001	16/08/2001
	1116	02/08/2001	16/08/2001
	1117	03/08/2001	17/08/2001
	1118	04/08/2001	18/08/2001

Figura 5.1. Facturile ce urmează a fi încasate în perioada 15-18 august 2001

## Exemplul 2

*Care sunt clienții din orașele Iași și Pașcani ?*

- Soluția 2.1:

```
SELECT CLIENTI.* ;
FROM CLIENTI INNER JOIN LOCALITATI ;
      ON CLIENTI.CodPost = LOCALITATI.CodPost ;
WHERE Loc = „Iasi” OR Loc = „Pascani”
```

- Soluția 2.2 – folosind operatorul IN:

```
SELECT CLIENTI.* ;
FROM CLIENTI INNER JOIN LOCALITATI ;
      ON CLIENTI.CodPost = LOCALITATI.CodPost ;
WHERE Loc IN ( 'Iasi', 'Pascani' )
```

- Soluția 2.3 – vechea sintaxă din SQL-89:

```
SELECT CLIENTI.* ;
FROM CLIENTI, LOCALITATI ;
WHERE CLIENTI.CodPost = LOCALITATI.CodPost AND ;
      Loc IN ( 'Iasi', 'Pascani' )
```

- Soluția 2.4 – sintaxa SQL-89, cu aliasuri pentru tabele:

```
SELECT C.* ;
FROM CLIENTI C, LOCALITATI L ;
WHERE C.CodPost = L.CodPost AND ;
      Loc IN ( 'Iasi', 'Pascani' )
```

- Soluția 2.5 – bazată pe subconsultări:

```
SELECT * ;
FROM CLIENTI ;
WHERE CodPost IN ;
      (SELECT CodPost ;
       FROM LOCALITATI ;
       WHERE Loc IN ( 'Iasi', 'Pascani' ) ;
      )
```

### Exemplul 3

*Ce persoane-cheie de la firmele-client trebuie felicitate de Sf. Ion ?*

- Soluția 3.1 – folosind operatorul LIKE:

```
SELECT dencl, functie, p.* ;
FROM PERSOANE p INNER JOIN PERSCLIENTI pc ON p.CNP=pc.CNP ;
    INNER JOIN CLIENTI c ON pc.CodCl=c.codcl ;
WHERE UPPER(Prenume) LIKE 'ION%' OR ;
    UPPER(Prenume) LIKE 'IOAN%' OR ;
    UPPER(Prenume) LIKE '% ION%' OR ;
    UPPER(Prenume) LIKE '% IOAN%' OR ;
    UPPER(Prenume) LIKE '%-ION%' OR ;
    UPPER(Prenume) LIKE '%-IOAN%'
```

Dencl	Functie	Cnp	Nume	Prenume	Adresa	Sex	Codpost	Telacasa	Telbirou	Telmobil	Email
Client 2 SA	Director general	CNP2	Vasile	Ion	NULL	B	6600	234567	876543	094222223	ion@a.ro
Client 2 SA	Self aprovizionare	CNP3	Popovici	Ioana	V.Micle, B.I, Sc.B,Ap.2	F	5725	345678	NULL	094222224	NULL
Client 5 SRL	Self aprovizionare	CNP7	Popa	Ioanid	I.Ion, B.IH2, Sc.C, Ap.45	B	1900	789012	321098	NULL	NULL

Figura 5.2. VIP-urile de la firmele-client pe care trebuie să le salutăm de Sf. Ion

### Exemplul 4

*Ce persoane-cheie de la firmele-client nu au cont de e-mail ?*

- Soluția 4.1 – folosind operatorul IS NULL:

```
SELECT dencl, functie, p.* ;
FROM PERSOANE p INNER JOIN PERSCLIENTI pc ON p.CNP=pc.CNP ;
    INNER JOIN CLIENTI c ON pc.CodCl=c.codcl ;
WHERE email IS NULL
```

### Exemplul 5

*Care sunt facturile emise în aceeași zi ca și factura 1113 ?*

- Soluția 5.1 – bazată pe subconsultare:

```
SELECT * ;
FROM FACTURI ;
WHERE DataFact IN ;
    (SELECT DataFact ;
    FROM FACTURI ;
    WHERE NrFact=1113)
```

- Soluția 5.2 – bazată pe joncțiune:

```
SELECT F1.NrFact, F1.DataFact ;
FROM FACTURI F1 INNER JOIN FACTURI F2 ;
    ON F1.DataFact = F2.DataFact AND F2.NrFact = 1113
```

- Soluția 5.3 – folosind o interogare corelată:

```
SELECT * ;
FROM FACTURI F1 ;
WHERE EXISTS ;
    (SELECT 1 ;
     FROM FACTURI F2 ;
     WHERE F1.DataFact = F2.DataFact AND F2.NrFact=1113)
```

### Exemplul 6

*Care dintre persoanele din tabela cu același nume nu au funcții la firmele-client ?*

- Soluția 6.1 – pe bază de subconsultare:

```
SELECT p.* ;
FROM PERSOANE p ;
WHERE CNP NOT IN ;
    (SELECT CNP ;
     FROM PERSCLIENTI)
```

- Soluția 6.2 – bazată pe joncțiune externă:

```
SELECT * ;
FROM PERSOANE p LEFT OUTER JOIN PERSCLIENTI pc ;
    ON p.CNP=pc.CNP ;
WHERE NVL(pc.CodCl,0) = 0
```

## 5.4.2. Funcții-agregat, cu și fără grupare

### Exemplul 7

*La câți clienți s-au trimis facturi ?*

- Soluția 7.1 – o funcție COUNT și o subconsultare:

```
SELECT COUNT (*) ;
FROM CLIENTI ;
WHERE CodCl IN ;
    (SELECT CodCl ;
     FROM FACTURI)
```

- Soluția 7.2 – clauza DISTINCT a unei funcții COUNT:

```
SELECT COUNT (DISTINCT CodCl) ;
FROM FACTURI
```

### Exemplul 8

*Care este valoarea totală a facturii 1119, valoarea medie, maximă și minimă a produselor vândute pe această factură ?*

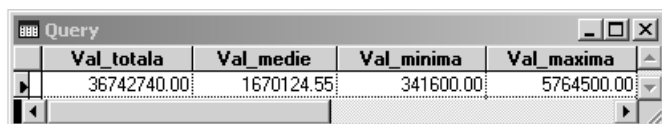
Scopul exemplului este de a ilustra modul de folosirea a funcțiilor-agregat SUM, AVG, MIN, MAX. Rezultatul este prezentat în figura 5.3.

```
SELECT SUM(Cantitate * PretUnit * ( 1 + ProcTVA) ;
        AS Val_Totala, ;
        AVG(Cantitate * PretUnit + Cantitate * PretUnit * ProcTVA);
```

```

    AS Val_Medie, ;
    MIN(Cantitate * PretUnit + Cantitate * PretUnit * ProctVA);
    AS Val_Minima, ;
    MAX(Cantitate * PretUnit + Cantitate * PretUnit * ProctVA);
    AS Val_Maxima ;
FROM PRODUSE P INNER JOIN LINIIFACT LF ON P.CodPr = LF.CodPr

```



Val_totala	Val_medie	Val_minima	Val_maxima
36742740.00	1670124.55	341600.00	5764500.00

Figura 5.3. Valorile: totală, medie, minimă și maximă ale facturii 1119

### Exemplul 9

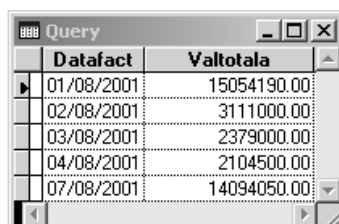
*Care este valoarea totală a vânzărilor pentru fiecare zi în care s-au emis facturi ?*

Pentru rezolvarea problemei este obligatorie folosirea grupării (figura 5.4).

```

SELECT DataFact, SUM(Cantitate * PretUnit * (1+ProctVA));
    AS ValTotala ;
FROM LINIIFACT LF, PRODUSE P, FACTURI F ;
WHERE LF.CodPr = P.CodPr AND LF.NrFact = F.NrFact ;
GROUP BY DataFact

```



Datafact	Valtotala
01/08/2001	15054190.00
02/08/2001	3111000.00
03/08/2001	2379000.00
04/08/2001	2104500.00
07/08/2001	14094050.00

Figura 5.4. Folosirea clauzei GROUP BY

### Exemplul 10

*Care sunt vânzările, cantitativ și valoric, pentru fiecare produs ?*

```

SELECT DenPr, UM, SUM(Cantitate) AS Cantitativ, ;
SUM(Cantitate * PretUnit * (1+ProctVA)) AS Valoric ;
FROM LINIIFACT LF, PRODUSE P, FACTURI F ;
WHERE P.CodPr = LF.CodPr AND LF.NrFact = F.NrFact ;
GROUP BY DenPr, UM

```

### Exemplul 11

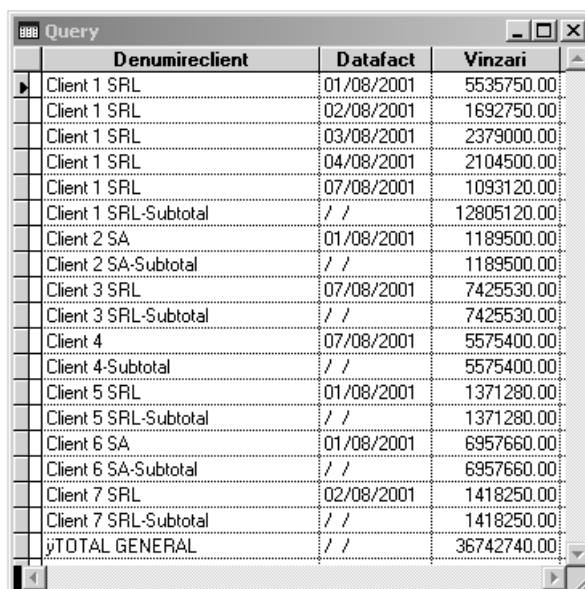
*Să se obțină situația vânzărilor pe clienți și zile, afișându-se câte un subtotal la nivel de client și un total general.*

Varianta propusă reunește mulțimea facturilor propriu-zise, obținută din prima frază SELECT, cu mulțimea subtotalurilor la nivel de client (al doilea SELECT) și cu o linie ce reprezintă totalul general.

```

SELECT PADR(ALLT(DenCl),40) AS DenumireClient, ;
      DataFact, ;
      SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari ;
FROM LINIIFACT LF, PRODUSE P, FACTURI F, CLIENTI C ;
WHERE P.CodPr = LF.CodPr AND LF.NrFact = F.NrFact ;
      AND F.CodCl=C.CodCl ;
GROUP BY DenCl, DataFact ;
UNION ;
      SELECT PADR(ALLT(DenCl)+'-Subtotal',40), ;
      { / / }, ;
      SUM(Cantitate * PretUnit * (1+ProcTVA)) ;
FROM LINIIFACT LF, PRODUSE P, FACTURI F, CLIENTI C ;
WHERE P.CodPr = LF.CodPr AND LF.NrFact = F.NrFact ;
      ND F.CodCl=C.CodCl ;
GROUP BY DenCl ;
UNION ;
SELECT CHR(255)+'TOTAL GENERAL' , ;
      { / / }, ;
      SUM(Cantitate * PretUnit * (1+ProcTVA)) ;
FROM LINIIFACT LF, PRODUSE P ;
WHERE P.CodPr = LF.CodPr

```



Denumireclient	Datafact	Vinzari
Client 1 SRL	01/08/2001	5535750.00
Client 1 SRL	02/08/2001	1692750.00
Client 1 SRL	03/08/2001	2379000.00
Client 1 SRL	04/08/2001	2104500.00
Client 1 SRL	07/08/2001	1093120.00
Client 1 SRL-Subtotal	/ /	12805120.00
Client 2 SA	01/08/2001	1189500.00
Client 2 SA-Subtotal	/ /	1189500.00
Client 3 SRL	07/08/2001	7425530.00
Client 3 SRL-Subtotal	/ /	7425530.00
Client 4	07/08/2001	5575400.00
Client 4-Subtotal	/ /	5575400.00
Client 5 SRL	01/08/2001	1371280.00
Client 5 SRL-Subtotal	/ /	1371280.00
Client 6 SA	01/08/2001	6957660.00
Client 6 SA-Subtotal	/ /	6957660.00
Client 7 SRL	02/08/2001	1418250.00
Client 7 SRL-Subtotal	/ /	1418250.00
TOTAL GENERAL	/ /	36742740.00

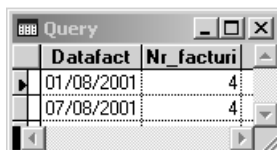
Figura 5.5. Mini-OLAP in SQL-VFP

### Exemplul 12

*Care sunt zilele în care s-au întocmit cel puțin trei facturi ?*

```
SELECT DataFact, COUNT(*) AS Nr_Facturi ;
FROM FACTURI ;
GROUP BY DataFact ;
HAVING COUNT(*) >= 3
```

Este o primă situație în care se recurge la opțiunea HAVING.



Datafact	Nr_facturi
01/08/2001	4
07/08/2001	4

Figura 5.6. Zilele în care s-au întocmit minimum 3 facturi

### Exemplul 13

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

Deși are un enunț banal, rezolvarea acestei probleme oferă oportunitatea unor interogări dintre cele mai spumoase. Vă prezentăm numai cinci dintre acestea.

- Soluția 13.1 – joncționarea unei instanțe obținută prin joncțiunea PRODUSE-LINIIFACT-FACTURI (în care DenPr = 'Produs 1') cu o altă instanță a aceleiași combinații (în care DenPr = 'Produs 2'):

```
SELECT DISTINCT F1.DataFact ;
FROM PRODUSE P1 ;
    INNER JOIN LINIIFACT LF1 ON P1.CodPr = LF1.CodPr ;
    INNER JOIN FACTURI F1 ON LF1.NrFact = F1.NrFact ;
    INNER JOIN FACTURI F2 ON F1.DataFact=F2.DataFact ;
    INNER JOIN LINIIFACT LF2 ON LF2.NrFact = F2.NrFact ;
    INNER JOIN PRODUSE P2 ON LF2.CodPr = P2.CodPr ;
WHERE P1.DenPr = 'Produs 1' AND P2.DenPr = 'Produs 2'
```

- Soluția 13.2 – folosind clauza HAVING și funcția COUNT:

```
SELECT DISTINCT DataFact ;
FROM PRODUSE INNER JOIN LINIIFACT ;
    ON PRODUSE.CodPr = LINIIFACT.CodPr ;
    INNER JOIN FACTURI ON LINIIFACT.Nrfact =FACTURI.NrFact ;
WHERE DenPr IN ('Produs 1', 'Produs 2') ;
GROUP BY DataFact ;
HAVING COUNT(DISTINCT LINIIFACT.CodPr) = 2
```

- Soluția 13.3 – simulând intersecția prin subconsultări:

```
SELECT DISTINCT DataFact ;
FROM PRODUSE INNER JOIN LINIIFACT ;
    ON PRODUSE.CodPr = LINIIFACT.CodPr ;
    INNER JOIN FACTURI ON LINIIFACT.NrFact = FACTURI.NrFact ;
WHERE DenPr = 'Produs 1' AND DataFact IN ;
    (SELECT DISTINCT DataFact ;
    FROM PRODUSE ;
        INNER JOIN LINIIFACT ON ;
            PRODUSE.CodPr = LINIIFACT.CodPr ;
        INNER JOIN FACTURI ON ;
            LINIIFACT.NrFact = FACTURI.NrFact ;
    WHERE DenPr = 'Produs 2')
```

- Soluția 13.4 – prin corelarea a două instanțe ale joncțiunii PRODUSE-LINIIFACT-FACTURI; prima conține liniile legate de *Produs 1*, iar a doua de *Produs 2*:

```
SELECT DISTINCT DataFact ;
FROM PRODUSE P1, LINIIFACT LF1, FACTURI F1 ;
WHERE P1.CodPr = LF1.CodPr AND ;
    LF1.NrFact = F1.NrFact AND DenPr = 'Produs 1' ;
AND EXISTS ;
    (SELECT 1 ;
    FROM PRODUSE P2, LINIIFACT LF2, FACTURI F2 ;
    WHERE P2.CodPr = LF2.CodPr AND LF2.NrFact = F2.NrFact ;
        AND P2.DenPr = 'Produs 2' AND F2.DataFact=F1.DataFact)
```

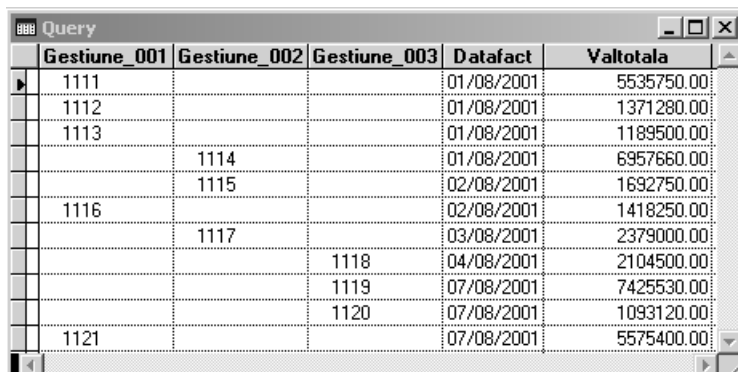
- Soluția 13.5:

```
SELECT DISTINCT DataFact ;
FROM FACTURI ;
    INNER JOIN LINIIFACT ON FACTURI.NrFact=LINIIFACT.NrFact ;
    INNER JOIN PRODUSE ON LINIIFACT.CodPr=PRODUSE.CodPr AND ;
        DenPr IN ('Produs 1','Produs 2') ;
WHERE DTOC(DataFact)+DTOC({//}) NOT IN ;
    (SELECT DISTINCT ;
        DTOC(F1.DataFact)+DTOC(NVL(F2.DataFact,{//})) ;
    FROM FACTURI F1 INNER JOIN PRODUSE P1 ;
        ON P1.DenPr IN ('Produs 1','Produs 2') ;
        LEFT OUTER JOIN (LINIIFACT LF2 INNER JOIN FACTURI F2 ;
            ON LF2.NrFact=F2.NrFact) ;
        ON F1.DataFact=F2.DataFact AND P1.CodPr=LF2.CodPr)
```



### Exemplul 14

Să se afișeze o situație a facturilor emise, pe gestiuni, de forma celei din figura 5.7.



Gestiune_001	Gestiune_002	Gestiune_003	Datafact	Valttotala
1111			01/08/2001	5535750.00
1112			01/08/2001	1371280.00
1113			01/08/2001	1189500.00
	1114		01/08/2001	6957660.00
	1115		02/08/2001	1692750.00
1116			02/08/2001	1418250.00
	1117		03/08/2001	2379000.00
		1118	04/08/2001	2104500.00
		1119	07/08/2001	7425530.00
		1120	07/08/2001	1093120.00
1121			07/08/2001	5575400.00

Figura 5.7. Folosirea IIF-urilor în interogări

Afișarea pe coloane separate, pentru cele trei gestiuni, a facturilor necesită folosirea unei secvențe de IF-uri imediate (IIF-uri):

```
SELECT IIF(gestiune = '001', STR(F.NrFact,8), space(8)) ;
      AS Gestiune_001, ;
      IIF(gestiune = '002', STR(F.NrFact,8), space(8)) ;
      AS Gestiune_002, ;
      IIF(gestiune = '003', STR(F.NrFact,8), space(8)) ;
      AS Gestiune_003, ;
      DataFact, SUM(Cantitate * PretUnit * (1+ProcTVA)) ;
      AS ValTotala ;
FROM FACTURI F ;
  INNER JOIN LINIIFACT LF ON F.NrFact = LF.NrFact ;
  INNER JOIN PRODUSE P ON LF.CodPr = P.CodPr ;
GROUP BY F.NrFact
```

### Exemplul 15

Scadența fiecărei facturi emise este de 20 de zile. Dacă însă data-limită cade într-o sâmbătă sau duminică, atunci scadența se mută în luna următoare. Care sunt zilele scadente în aceste condiții ?

Visual FoxPro prezintă funcțiile CDOW (Character Day Of the Week) și DOW (Day Of the Week), rezultatul din figura 5.8 fiind obținut după cum urmează:

```
SELECT NrFact AS Factura, DataFact, ;
      DataFact + 20 AS Scadental, ;
      CDOW(DataFact+20) AS NumeZil, ;
      IIF(DOW(DataFact+20)=6, ;
      DataFact+22, ;
      IIF(DOW(DataFact+20)=1, ;
      DataFact+21, ;
      DataFact+20) ) AS Scadenta, ;
```

```

CDOV(IIF(DOW(DataFact+20)=6, ;
DataFact+22, ;
IIF(DOW(DataFact+20)=1, ;
DataFact+21, ;
DataFact+20) ) ) AS Zi_Scadenta ;
FROM FACTURI

```

Factura	Datafact	Scadenta1	Numezi1	Scadenta	Zi_scadenta
1111	01/08/2001	21/08/2001	Tuesday	21/08/2001	Tuesday
1112	01/08/2001	21/08/2001	Tuesday	21/08/2001	Tuesday
1113	01/08/2001	21/08/2001	Tuesday	21/08/2001	Tuesday
1114	01/08/2001	21/08/2001	Tuesday	21/08/2001	Tuesday
1115	02/08/2001	22/08/2001	Wednesd	22/08/2001	Wednesd
1116	02/08/2001	22/08/2001	Wednesd	22/08/2001	Wednesd
1117	03/08/2001	23/08/2001	Thursda	23/08/2001	Thursda
1118	04/08/2001	24/08/2001	Friday	26/08/2001	Sunday
1119	07/08/2001	27/08/2001	Monday	27/08/2001	Monday
1120	07/08/2001	27/08/2001	Monday	27/08/2001	Monday
1121	07/08/2001	27/08/2001	Monday	27/08/2001	Monday
1122	07/08/2001	27/08/2001	Monday	27/08/2001	Monday

Figura 5.8. Determinarea scadenței de încasare

### Exemplul 16

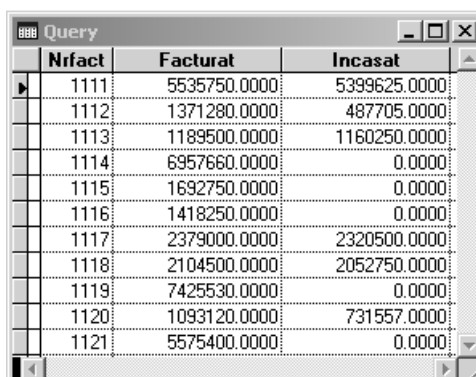
*Care sunt valorile facturate și încasate ale fiecărei facturi ?*

Soluția de mai jos, ce obține liniile din figura 5.9, reunește facturile care au măcar o tranșă de încasare cu cele neîncasate deloc:

```

SELECT F. NrFact, ;
SUM(Cantitate * PretUnit * (1+ProcTVA)) / ;
COUNT(DISTINCT I.CodInc) AS Facturat, ;
SUM(Transa) / MAX(LF.Linie) AS Incasat ;
FROM LINIIFACT LF, PRODUSE P, FACTURI F, INCASFACT I ;
WHERE LF.CodPr = P.CodPr AND LF.NrFact = F.NrFact ;
AND F.NrFact=I.NrFact ;
GROUP BY F.NrFact ;
UNION ;
SELECT F. NrFact, ;
SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Facturat, ;
0 AS Incasat ;
FROM LINIIFACT LF, PRODUSE P, FACTURI F ;
WHERE LF.CodPr = P.CodPr AND LF.NrFact = F.NrFact ;
AND F.NrFact NOT IN ;
(SELECT NrFact ;
FROM INCASFACT) ;
GROUP BY F.NrFact

```



	NrFact	Facturat	Incasat
	1111	5535750.0000	5399625.0000
	1112	1371280.0000	487705.0000
	1113	1189500.0000	1160250.0000
	1114	6957660.0000	0.0000
	1115	1692750.0000	0.0000
	1116	1418250.0000	0.0000
	1117	2379000.0000	2320500.0000
	1118	2104500.0000	2052750.0000
	1119	7425530.0000	0.0000
	1120	1093120.0000	731557.0000
	1121	5575400.0000	0.0000

Figura 5.9. Valorile facturate și încasate ale fiecărei facturi

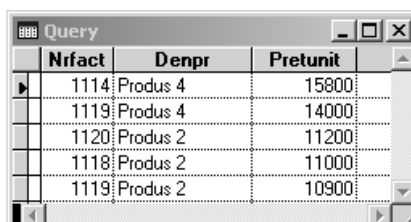
### 5.4.3. Clauza TOP. Operatorii ALL și ANY (SOME)

#### Exemplul 17

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

În alte SGBD-uri, această problemă ar crea dureri de cap plene. Clauza TOP din VFP realizează ordonarea și extragerea celor cinci valori – vezi figura 5.10.

```
SELECT TOP 5 NrFact, DenPr, PretUnit ;
FROM LINIIFACT INNER JOIN PRODUSE ;
ON LINIIFACT.CodPr=PRODUSE.CodPr ;
ORDER BY PretUnit DESC
```



	NrFact	Denpr	Pretunit
	1114	Produs 4	15800
	1119	Produs 4	14000
	1120	Produs 2	11200
	1118	Produs 2	11000
	1119	Produs 2	10900

Figura 5.10. Folosirea clauzei TOP

#### Exemplul 18

*Care sunt produsele vândute la prețuri unitare superioare oricărui preț unitar la care a fost vândut 'Produs 1' ?*

Unul dintre obiectivele acestui exemplu este de a prezenta modul de conexiune a unei consultări cu subconsultarea sa prin operatorul ALL.

```

SELECT DISTINCT DenPr, PretUnit ;
FROM PRODUSE P, LINIIFACT LF ;
WHERE P.CodPr=LF.CodPr AND PretUnit > ALL ;
      (SELECT DISTINCT PretUnit ;
       FROM PRODUSE P, LINIIFACT LF ;
       WHERE P.CodPr=LF.CodPr AND DenPr ='Produs 1')

```

Denpr	Pretunit
Produs 2	10300
Produs 2	10500
Produs 2	10700
Produs 2	10900
Produs 2	11000
Produs 2	11200
Produs 4	14000
Produs 4	15800

Figura 5.11. Operatorul ALL

### Exemplul 19

Care sunt produsele vândute la prețuri unitare superioare **măcar** unui preț unitar al 'Produsului 1' ?

Este genul de situații în care se folosește SOME sau ANY (au aceeași funcțiune).

```

SELECT DISTINCT DenPr, PretUnit ;
FROM PRODUSE P INNER JOIN LINIIFACT LF ;
ON P.CodPr=LF.CodPr ;
WHERE PretUnit > ANY ;
      (SELECT DISTINCT PretUnit ;
       FROM PRODUSE P INNER JOIN LINIIFACT LF ;
       ON P.CodPr=LF.CodPr ;
       WHERE DenPr ='Produs 1')

```

Denpr	Pretunit
Produs 1	9500
Produs 1	10000
Produs 2	9750
Produs 2	10000
Produs 2	10300
Produs 2	10500
Produs 2	10700
Produs 2	10900
Produs 2	11000
Produs 2	11200
Produs 4	14000
Produs 4	15800

Figura 5.12. Operatorul ANY

**Exemplul 20**

*Care este ultima factură întocmită (factura cea mai recentă) și data în care a fost emisă ?*

Și aceasta este o problemă cu multe variante de rezolvare:

- Soluția 20.1:

```
SELECT DataFact, NrFact AS UltimaFactura ;  
FROM FACTURI ;  
WHERE NrFact IN ;  
      (SELECT MAX(NrFact) ;  
       FROM FACTURI)
```

- Soluția 20.2 – în locul operatorului IN, operatorul de conexiune este semnul egal:

```
SELECT DataFact, NrFact AS UltimaFactura ;  
FROM FACTURI ;  
WHERE NrFact = ;  
      (SELECT MAX(NrFact) ;  
       FROM FACTURI)
```

- Soluția 20.3 – folosind operatorul ANY:

```
SELECT DataFact, NrFact AS UltimaFactura ;  
FROM FACTURI ;  
WHERE NrFact =ANY ;  
      (SELECT MAX(NrFact) ;  
       FROM FACTURI)
```

- Soluția 20.4 – operatorul ALL și, în subconsultare, funcția MAX:

```
SELECT DataFact, NrFact AS UltimaFactura ;  
FROM FACTURI ;  
WHERE NrFact =ALL ;  
      (SELECT MAX(NrFact) ;  
       FROM FACTURI)
```

- Soluția 20.5 – conexiune prin >= ALL:

```
SELECT DataFact, NrFact AS UltimaFactura ;  
FROM FACTURI ;  
WHERE NrFact >= ALL ;  
      (SELECT NrFact ;  
       FROM FACTURI) ;
```

- Soluția 20.6 – soluție „tipică” VFP – ce folosește clauza TOP aplicată grupurilor:

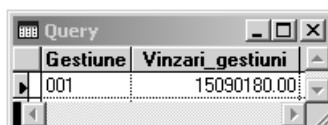
```
SELECT TOP 1 DataFact, COUNT(*) AS Nr ;  
FROM FACTURI ;  
GROUP BY DataFact ;  
ORDER BY Nr DESC
```

### Exemplul 21

*Care este gestiunea cu vânzările cele mai bune ?*

Dacă în alte SGBD-uri sunt necesare subconsultări în clauza FROM, în VFP, aceeași clauză TOP este salvatoare – figura 5.13:

```
SELECT TOP 1 Gestiune, SUM(Cantitate * PretUnit * ;
    (1+ProcTVA)) AS Vinzari_Gestiuni ;
FROM FACTURI F ;
    INNER JOIN LINIIFACT LF ON F.NrFact=LF.NrFact ;
    INNER JOIN PRODUSE P ON LF.CodPr=P.CodPr ;
GROUP BY Gestiune ;
ORDER BY Vinzari_Gestiuni DESC
```



Gestiune	Vinzari_gestiuni
001	15090180.00

Figura 5.13. Gestiunea pentru care s-au înregistrat cele mai mari vânzări

## 5.5. Actualizarea datelor folosind (sub)consultări

Visual FoxPro prezintă, de mai bine de șase ani, câteva atuuri ce constituiau altădată apanajul lumii bune a SGBD-urilor: procedurile stocate și, implicit, declanșatoarele. Devine astfel posibilă introducerea unor atribute redundante, dar care măresc sensibil viteza de lucru a aplicației. Procesul se numește denormalizare și frământă lumea proiectanților de baze de date (nu pe noi, autorii de față). Aceste atribute pot fi gestionate automat și ferite de accidente voite sau întâmplătoare.

Exagerând poate (un pic) cu denormalizarea, introducem o serie de atribute noi, după cum urmează:

- în tabela LINIIFACT – atributul ValCuTVA, pentru a elimina, la obținerea rapoartelor/informațiilor în rapoarte, joncțiunea cu tabela PRODUSE (din care se preia procentul de TVA);
- în tabela FACTURI adăugăm nu mai puțin de patru atribute:
  - ValTotală – reprezintă valoarea totală (inclusiv TVA) a facturii;
  - ValIncasata – este valoarea încasată (până în prezent) din contravaloarea facturii;
  - Reduceri – de obicei pentru plata rapidă a facturii;
  - Penalități – este opusul atributului anterior.

Comenzile SQL prin care ne punem în operă intențiile sunt:

```
ALTER TABLE LINIIFACT ADD ValCuTVA NUMERIC(16)
ALTER TABLE FACTURI ADD ValTotala NUMERIC(16)
ALTER TABLE FACTURI ADD ValIncasata NUMERIC(16)
```

```
ALTER TABLE FACTURI ADD Reduceri NUMERIC (15)
ALTER TABLE FACTURI ADD Penalizari NUMERIC (15)
```

Deoarece aceste modificări nu au fost luate în calcul în momentul creării bazei, atributele trebuie „umplute” din mers. Motiv de prezentare a comenzilor de actualizare într-o variantă mai de finețe – folosind subconsultări. Lucrul acesta nu e permis încă de comanda INSERT, în care valorile nu pot fi specificate printr-un SELECT (spre deosebire de SQL-92).

În schimb, comanda DELETE este destul de tolerantă în acest sens. Spre exemplu, se dorește *ștergerea tuturor liniilor din tabela FACTURI care nu au copii în LINIIFACT* (altfel spus, facturile fără nici un produs/serviciu). O soluție elegantă este următoarea:

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

Firește, subconsultarea poate fi și corelată:

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

*Să se elimine din tabela FACTURI toate liniile care se referă la facturi cu valoare sub 5 milioane lei.*

Rezolvarea problemei presupune folosirea unei subconsultări cu grupare.

```
DELETE FROM FACTURI ;
WHERE NrFact NOT IN ;
  (SELECT NrFact ;
   FROM LINIIFACT LF INNER JOIN PRODUSE P ;
   ON LF.CodPr=P.CodPr ;
   GROUP BY NrFact ;
   HAVING SUM(Cantitate*PretUnit*(1+ProcTVA)) >= 5000000 ;
  )
```

Pentru actualizarea atributelor introduse în acest paragraf, ne interesează, în primul rând, comanda UPDATE. Dacă am fi siguri că toate produsele/serviciile ar avea procentul de TVA unic – și anume 19% –, actualizarea atributului ValCuTVA din LINIIFACT ar fi ușoară:

```
UPDATE LINIIFACT ;
SET ValCuTVA = Cantitate * PretUnit * 1.19
```

Schema bazei este una pe termen lung însă. În funcție de legislația în vigoare, la un moment dat, produsele pot avea procente diferite de TVA, în funcție de categoria din care fac parte (de strictă necesitate, de lux etc.). De aceea, la calculul valorii cu TVA este obligatorie preluarea procentului TVA din tabela PRODUSE. Din păcate, varianta care ne-ar trebui:

```

UPDATE LINIIFACT ;
SET ValCuTVA = ;
  (SELECT Cantitate * PretUnit * (1+ProctTVA) ;
   FROM LINIIFACT LF INNER JOIN PRODUSE P ;
    ON LF.CodPr=P.CodPr ;
   WHERE LF.NrFact = LINIIFACT.NrFact AND ;
    P.CodPr = LINIIFACT.CodPr ;
  )

```

nu are de gând să funcționeze în nici una dintre versiunile VFP. Drept pentru care suntem nevoiți să folosim o soluție procedurală, cu sau fără cursoare (vezi paragraful următor). Singurul loc în care UPDATE-ul poate prezenta o subconsultare este clauza WHERE. De exemplu, dacă se dorește reducerea penalizărilor pentru toți clienții din Pașcani, se poate folosi comanda:

```

UPDATE FACTURI ;
SET Penalizari = Penalizari * .5 ;
WHERE CodCl IN ;
  (SELECT CodCl ;
   FROM CLIENTI INNER JOIN LOCALITATI ;
    ON CLIENTI.CodPost = LOCALITATI.CodPost ;
   WHERE Loc = 'Pascani')

```

## 5.6. Includerea frazelor SELECT în programe

Multe dintre problemele complexe de interogare necesită în SQL folosirea mai multor niveluri de subconsultare sau prezența subconsultărilor în clauzele FROM sau HAVING, facilități pe care nucleul SQL din Visual FoxPro nu le are încă implementate. În schimb, pentru aceste cazuri se pot crea programe în care rezultatele intermediare ale unor interogări sunt stocate în cursoare sau tabele virtuale care, la rândul lor, pot constitui argumente ale unor noi consultări.

### 5.6.1. Salvarea rezultatului unei fraze SELECT în cursor

Cursorul este, în Visual FoxPro, o tabelă temporară, a cărei viață se întinde de la momentul unui SELECT ce prezintă clauza INTO CURSOR sau CREATE CURSOR până la închiderea sa, implicită sau explicită.

Revenim la **exemplul 13**:

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

La seria interogărilor care au fost prezentate adăugăm și o variantă simplă de program – listingul 5.4 –, care se bazează pe folosirea a două cursoare, astfel:



Listing 5.4. Program VFP ce salvează rezultatele intermediare ale interogărilor în cursoare

```

SELECT DISTINCT DataFact ;
INTO CURSOR Zile_P1 ;
FROM PRODUSE INNER JOIN LINIIFACT ON PRODUSE.CodPr = LINIIFACT.CodPr ;
    INNER JOIN FACTURI ON LINIIFACT.Nrfact = FACTURI.NrFact ;
WHERE DenPr = 'Produs 1'

SELECT DISTINCT DataFact ;
INTO CURSOR Zile_P2 ;
FROM PRODUSE INNER JOIN LINIIFACT ON PRODUSE.CodPr = LINIIFACT.CodPr ;
    INNER JOIN FACTURI ON LINIIFACT.Nrfact = FACTURI.NrFact ;
WHERE DenPr = 'Produs 2'

SELECT * ;
FROM Zile_P1 ;
WHERE DataFact IN ;
    (SELECT DataFact ;
     FROM Zile_P2)

```

Prima interogare creează cursorul Zile\_P1, ce conține numai zilele în care s-a vândut primul produs; al doilea cursor este populat cu zilele în care s-a vândut al doilea produs; ultima interogare realizează intersecția celor două cursoare – figura 5.14.

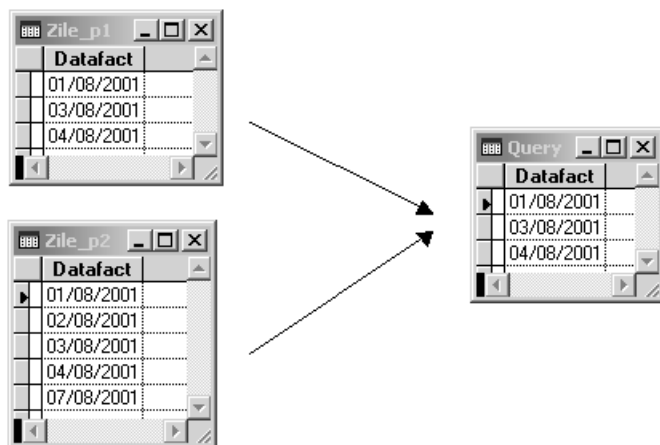


Figura 5.14. Conținutul celor două cursoare și rezultatul interogării finale

## Exemplul 22

*Care este județul cu vânzări imediat superioare județului Neamț ?*

Soluția din listingul 5.5 are un dram suplimentar de interes, deoarece la ultima sa interogare, în afara clauzei WHERE, care conține două subconsultări, se efectuează theta-joncțiunea dintre cele două cursoare, cVinzari\_Neamt și cVinzari\_Judete.

Listing 5.5. Județul cu vânzări imediat peste cele ale județului Neamț

```

* Vinzarile jud. Neamt
SELECT SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari ;
FROM JUDETE J ;
    INNER JOIN LOCALITATI L ON J.Jud=L.Jud ;
    INNER JOIN CLIENTI C ON L.CodPost=C.CodPost ;
    INNER JOIN FACTURI F ON C.CodCl=F.CodCl ;
    INNER JOIN LINIIFACT LF ON F.NrFact=LF.NrFact ;
    INNER JOIN PRODUSE P ON LF.CodPr=P.CodPr ;
INTO CURSOR cVinzari_Neamt ;
WHERE Judet='Neamt'

* Vinzarile fiecarui judet
SELECT Judet, SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari ;
FROM JUDETE J ;
    INNER JOIN LOCALITATI L ON J.Jud=L.Jud ;
    INNER JOIN CLIENTI C ON L.CodPost=C.CodPost ;
    INNER JOIN FACTURI F ON C.CodCl=F.CodCl ;
    INNER JOIN LINIIFACT LF ON F.NrFact=LF.NrFact ;
    INNER JOIN PRODUSE P ON LF.CodPr=P.CodPr ;
INTO CURSOR cVinzari_Judete ;
GROUP BY Judet

SELECT Judet, Vinzari ;
FROM cVinzari_Judete ;
WHERE Vinzari <= ALL ;
    (SELECT cVinzari_Judete.Vinzari ;
    FROM cVinzari_Judete, cVinzari_Neamt ;
    WHERE cVinzari_Judete.Vinzari > cVinzari_Neamt.Vinzari) ;
AND Vinzari > ALL ;
    (SELECT Vinzari ;
    FROM cVinzari_Neamt)

```

Conținutul cursoroarelor și al rezultatului final constituie obiectul figurii 5.15.

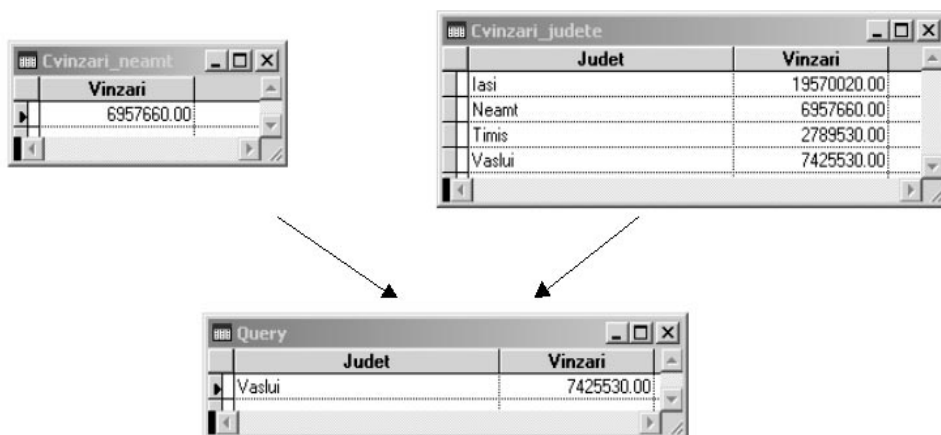


Figura 5.15. Determinarea județului cu vânzări imediat peste cele ale județului Neamț

## Exemplul 22

*Care este clientul cel mai mare datornic?*

Listingul 5.6 conține programul ce răspunde la această problemă.

Listing 5.6. Clientul cu cel mai mare rest de plată

```
* Valorile facturate, pe clienti
SELECT CodCl, SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Facturat ;
FROM FACTURI F ;
    INNER JOIN LINIIFACT LF ON F.NrFact=LF.NrFact ;
    INNER JOIN PRODUSE P ON LF.CodPr=P.CodPr ;
INTO CURSOR cFACTURAT ;
GROUP BY CodCl

* Valorile incasate, pe clienti
SELECT CodCl, SUM(Transa) AS Incasat ;
FROM FACTURI F ;
    INNER JOIN INCASFACT I ON F.NrFact=I.NrFact ;
INTO CURSOR cINCASAT ;
GROUP BY CodCl

* Restul de plata, pe clienti
SELECT cFACTURAT.CodCl, Facturat, NVL(Incasat,0) AS Incasat ,
    Facturat - NVL(Incasat,0) AS De_Incasat ;
FROM cFACTURAT LEFT OUTER JOIN cINCASAT ;
    ON cFACTURAT.CodCl=cINCASAT.CodCl ;
INTO CURSOR cDe_INCASAT ;

* Finalul apoteotic
SELECT DenCl, cDe_INCASAT.* ;
FROM cDe_INCASAT INNER JOIN CLIENTI ;
    ON cDe_INCASAT.CodCl=CLIENTI.CodCl ;
WHERE De_Incasat >= ALL ;
    (SELECT De_Incasat ;
    FROM cDe_INCASAT)
```

### 5.6.2. Utilizarea cursoroanelor pentru actualizări

În paragraful anterior au fost introduse patru câmpuri calculate, dar nici unul nu a fost adus „la zi”, din cauza limitărilor comenzii UPDATE în VFP. Modalitățile în care se poate redacta programul de actualizare a fiecăruia dintre atribute sunt destul de eterogene, de la soluții „curat xBase-iste” la soluții „preponderent SQL-istice” (scuzați terminologia !). Iată în listingul 5.7 o variantă simplă în care valoarea cu TVA a fiecărei linii se caculează și se salvează în cursorul cVal, cursor care se parcurge secvențial, pentru fiecare linie a sa operându-se modificarea în înregistrarea din LINIIFACT corespondentă.

Listing 5.7. Prima variantă de actualizare a atributului ValCuTVA

```
*
SELECT NrFact, LF.CodPr, Cantitate * PretUnit * (1+ProcTVA) AS ValCuTVA ;
INTO CURSOR cVAL ;
FROM LINIIFACT LF INNER JOIN PRODUSE P ON LF.CodPr=P.CodPr
```

```

SELECT cVal
SCAN
    UPDATE LINIIFACT ;
    SET ValCuTVA = cVal.ValcuTVA ;
    WHERE NrFact = cVal.NrFact AND CodPr = cVal.CodPr
ENDSCAN

```

Pentru comparație, în listingul 5.8 este prezentată o soluție ce combină folosirea cursorului cu indexul primar al tabeli LINIIFACT:

Listing 5.8. A doua variantă de actualizare a atributului ValCuTVA

```

*
SELECT STR(NrFact,8)+STR(Linie,2) AS Cheie, Cantitate * PretUnit * (1+ProcTVA) AS ValCuTVA ;
INTO CURSOR cVAL ;
FROM LINIIFACT LF INNER JOIN PRODUSE P ON LF.CodPr=P.CodPr

SELECT cVal
SCAN
    SELECT LINIIFACT
    SEEK cVal.Cheie ORDER TAG primaru
    REPLACE ValCuTVA WITH cVal.ValCuTVA
    SELECT cVal
ENDSCAN

```

iar în listingul 5.9, o variantă strict xBase:

Listing 5.9. A treia variantă de actualizare a atributului ValCuTVA

```

*
IF !USED('LINIIFACT')
    USE LINIIFACT in 0
ENDIF

IF !USED('PRODUSE')
    USE PRODUSE in 0
ENDIF

SELECT LINIIFACT
SCAN
    SELECT PRODUSE
    SEEK LINIIFACT.CodPr ORDER TAG CodPr
    SELECT LINIIFACT
    REPLACE ValCuTVA WITH Cantitate * PretUnit * (1+PRODUSE.ProcTVA)
ENDSCAN

```

Programul din listingul 5.10 este cel care realizează calculul celor trei atribute, LINIIFACT.ValCuTVA, FACTURI.ValTotala și FACTURI.ValIncasată. Pentru a mai crește factorul interes, valoarea încasată se actualizează prin intermediul unui tablou (masiv) – vIncasat.

Listing 5.10. Totalul celor trei noi atribute

```

UPDATE facturi SET ValTotala = 0, ValIncasata = 0, Penalizari = 0, Reduceri = 0

UPDATE liniifact SET ValCuTva = 0

IF !USED('PRODUSE')
    USE PRODUSE IN 0
ENDIF

SELECT LINIIFACT
SCAN
    * se preia procentul TVA
    SELECT PRODUSE
    SEEK LINIIFACT.CodPr ORDER TAG CodPr

    * calculul si inlocuirea valorii cu TVA
    SELECT LINIIFACT
    REPLACE ValCuTVA WITH Cantitate * PretUnit * (1+PRODUSE.ProcTVA)

    * adaugarea valorii in tabela facturi
    SELECT FACTURI
    SEEK LINIIFACT.NrFact ORDER TAG NrFact
    REPLACE ValTotala WITH ValTotala + LINIIFACT.ValCuTVA

    SELECT LINIIFACT
ENDSCAN

* calculul valorii incasate pentru fiecare factura si salvarea intr-un masiv
SELECT FACTURI.NrFact, SUM(NVL(Transa,0)) AS ValIncasata ;
INTO ARRAY vIncasat ;
FROM FACTURI LEFT OUTER JOIN INCASFACT ON FACTURI.NrFact=INCASFACT.NrFact ;
GROUP BY FACTURI.NrFact

* actualizarea valorii incasate in tabela FACTURI
SELECT FACTURI
SCAN
    pozitie = ASCAN (vIncasat, FACTURI.NrFact)
    REPLACE ValIncasata WITH vIncasat (pozitie + 1)
ENDSCAN

```

### 5.6.3. Alte considerații privind cursoarele

Un cursor poate fi creat și prin comanda `CREATE CURSOR`, caz în care tabela temporară este actualizabilă. Formatul comenzii prezintă clauze similare `CREATE TABLE`:

```

CREATE CURSOR alias_name
    (fname1 type [(precision [, scale])]
[NULL | NOT NULL]
[CHECK lExpression [ERROR cMessageText]]
[DEFAULT eExpression]
[UNIQUE]
[NOCPTRANS])
[, fname2 ...]
| FROM ARRAY ArrayName

```

O dată creat și declarate restricțiile, un cursor poate fi actualizat ca orice tabelă obișnuită, prin comenzile SQL: INSERT, UPDATE și DELETE, precum și orice altă comandă de editare specifică VFP. De asemenea, orice încălcare a vreunei restricții definite va fi semnalizată. După cum vom vedea în capitolul 9, și cursoarelor, similar tabelelor virtuale, li se pot vizualiza și configura proprietățile prin funcțiile CURSORGETPROP() și CURSORSETPROP().

Conținutul unui cursor poate fi adăugat la înregistrările curente ale unei tabeli prin comanda APPEND FROM DBF(' <NumeCursor> '). Astfel, putem redacta o variantă mai simplă dar mai primitivă de calcul al valorilor celor trei atribute. Prin fraze SELECT construim cursoare cu aceleași structuri ca ale tabelelor FACTURI și LINIIFACT, însă cu atributele ValCuTVA, ValTotala și ValIncasata calculate. Apoi ștergem toate liniile celor două tabele și adăugăm conținutul cursoarelor. Programul este cel din listingul 5.11.

Listing 5.11. O altă variantă pentru calculul valorilor celor trei noi atribute

```
* folosirea APPEND FROM DBF(<cursor>)

* se construiește un cursor cu structura identica LINIIFACT
* si cu toate atributele completate/calculate corect
SELECT NrFact, Linie, LF.CodPr, Cantitate, ;
  PretUnit, Cantitate * PretUnit * (1+ProcTVA) AS ValCuTVA ;
INTO CURSOR cLINIIFACT ;
FROM LINIIFACT LF INNER JOIN PRODUSE P ON LF.CodPr=P.CodPr

* se goleste tabela LINIIFACT si apoi se preia continutul cursorului
DELETE FROM LINIIFACT
SELECT LINIIFACT
PACK
APPEND FROM DBF('cLINIIFACT')

* analog pentru tabela FACTURI

SELECT FACTURI.NrFact, DataFact, Gestione, CodCl, Obs, ;
  SUM(NVL(ValCuTVA,0)) AS ValTotala, ;
  Reduceri, Penalizari, ;
  SUM(NVL(Transa,0)) AS ValIncasata ;
INTO CURSOR cFacturi ;
FROM FACTURI ;
  LEFT OUTER JOIN LINIIFACT ON FACTURI.NrFact = LINIIFACT.NrFact ;
  LEFT OUTER JOIN INCASFACT ON FACTURI.NrFact=INCASFACT.NrFact ;
GROUP BY FACTURI.NrFact

DELETE FROM FACTURI
SELECT FACTURI
PACK
APPEND FROM DBF('cFACTURI')
```

Cursoarele pot fi, în unele situații, un bun substitut al vectorilor. Să luăm un asemenea caz, cel al calculării reducerilor pentru plata rapidă a facturilor. Să presupunem că, pentru clienții buni platnici, se acordă reduceri la plată:

- pentru tranșele de facturi plătite (INCASARI.DataInc) în termen de până la 10 zile, acordăm o reducere de 10%;

- între 11 și 12 zile procentul este de 9%;
- între 13 și 15 zile – 8%;
- peste 16 zile – nu se acordă reduceri (eventual se calculează penalizări, dar asta-i altă poveste).

Tranșele de acordare a reducerilor pot fi stocate în cursoare și, astfel, folosite în interogări. În listingul 5.12 este prezentat programul care realizează calculul reducerilor și actualizarea atributului FACTURI.Reduceri.

Listing 5.12. Calculul reducerilor acordate pentru încasări

```
* se creaza si populeaza cursorul cu transele de reduceri
CREATE CURSOR transe_red ( ;
    zile_min INTEGER, ;
    zile_max INTEGER, ;
    procent INTEGER ;
)

INSERT INTO transe_red VALUES ( 0, 10, 10)
INSERT INTO transe_red VALUES (11, 12, 9)
INSERT INTO transe_red VALUES (13, 15, 8)
INSERT INTO transe_red VALUES (16, 99999, 0)

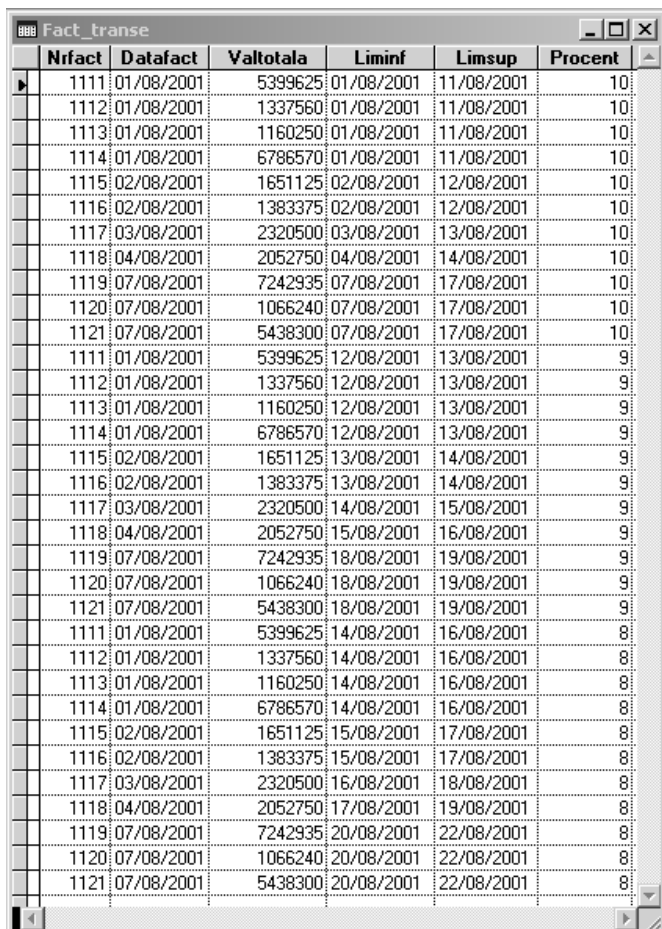
* pentru fiecare factura cu valoare peste zero se insereaza
* in cursul FACT_TRANSE cite o linie pentru fiecare transa
* posibila de reduceri
SELECT NrFact, DataFact, ValTotala, DataFact + zile_min AS LimInf, ;
    DataFact + zile_max AS LimSup, procent ;
FROM FACTURI, TRANSE_RED ;
INTO CURSOR FACT_TRANSE ;
WHERE procent > 0 AND ValTotala > 0

* se calculeaza reducerile pe fiecare transa
SELECT FACT_TRANSE.*, DataInc, Transa, INT(Procent * Transa / 100) AS Reducere ;
INTO CURSOR reduceri_transe ;
FROM INCASFACT INNER JOIN INCASARI ON INCASFACT.CodInc=INCASARI.CodInc ;
    INNER JOIN FACT_TRANSE ;
    ON INCASFACT.NrFact = FACT_TRANSE.NrFact ;
WHERE INCASARI.DataInc >= FACT_TRANSE.LimInf AND ;
INCASARI.DataInc <= FACT_TRANSE.LimSup

* se actualizeaza atributul FACTURI.Reduceri
DIME suma_ (1,1)
SELECT facturi
SCAN
    suma_ = 0
    SELECT SUM(Reducere) ;
        INTO ARRAY suma_ ;
        FROM REDUCERI_TRANSE ;
        WHERE NrFact = facturi.Nrfact
    SELECT facturi
    REPLACE Reduceri WITH suma_(1,1)
ENDSCAN
```

Soluția e destul de scumpă, dar operațională. Tranșele de reduceri constituie obiectul cursorului actualizabil `TRANSE_RED`, actualizabil automat deoarece a fost creat prin comanda `CREATE CURSOR` și nu printr-un `SELECT`.

Prima frază `SELECT` constituie unul dintre puținele exemple de folosire a produsului cartezian. Cursorul (Read-Only) `FACT_TRANSE` conține, pentru fiecare factură cu valoare mai mare decât zero, toate intervalele calendaristice pentru care se pot acorda reduceri. Ținând cont de liniile tabelor din baza de date, iată în figura 5.16 conținutul cursorului.

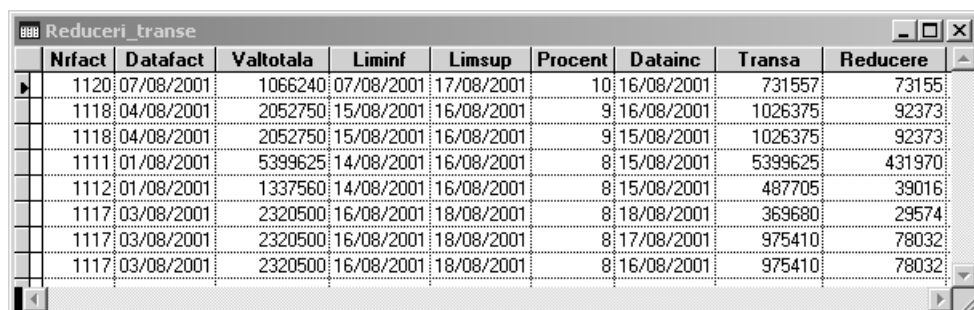


Nrfact	Datafact	Valtotala	Liminf	Limsup	Procent
1111	01/08/2001	5399625	01/08/2001	11/08/2001	10
1112	01/08/2001	1337560	01/08/2001	11/08/2001	10
1113	01/08/2001	1160250	01/08/2001	11/08/2001	10
1114	01/08/2001	6786570	01/08/2001	11/08/2001	10
1115	02/08/2001	1651125	02/08/2001	12/08/2001	10
1116	02/08/2001	1383375	02/08/2001	12/08/2001	10
1117	03/08/2001	2320500	03/08/2001	13/08/2001	10
1118	04/08/2001	2052750	04/08/2001	14/08/2001	10
1119	07/08/2001	7242935	07/08/2001	17/08/2001	10
1120	07/08/2001	1066240	07/08/2001	17/08/2001	10
1121	07/08/2001	5438300	07/08/2001	17/08/2001	10
1111	01/08/2001	5399625	12/08/2001	13/08/2001	9
1112	01/08/2001	1337560	12/08/2001	13/08/2001	9
1113	01/08/2001	1160250	12/08/2001	13/08/2001	9
1114	01/08/2001	6786570	12/08/2001	13/08/2001	9
1115	02/08/2001	1651125	13/08/2001	14/08/2001	9
1116	02/08/2001	1383375	13/08/2001	14/08/2001	9
1117	03/08/2001	2320500	14/08/2001	15/08/2001	9
1118	04/08/2001	2052750	15/08/2001	16/08/2001	9
1119	07/08/2001	7242935	18/08/2001	19/08/2001	9
1120	07/08/2001	1066240	18/08/2001	19/08/2001	9
1121	07/08/2001	5438300	18/08/2001	19/08/2001	9
1111	01/08/2001	5399625	14/08/2001	16/08/2001	8
1112	01/08/2001	1337560	14/08/2001	16/08/2001	8
1113	01/08/2001	1160250	14/08/2001	16/08/2001	8
1114	01/08/2001	6786570	14/08/2001	16/08/2001	8
1115	02/08/2001	1651125	15/08/2001	17/08/2001	8
1116	02/08/2001	1383375	15/08/2001	17/08/2001	8
1117	03/08/2001	2320500	16/08/2001	18/08/2001	8
1118	04/08/2001	2052750	17/08/2001	19/08/2001	8
1119	07/08/2001	7242935	20/08/2001	22/08/2001	8
1120	07/08/2001	1066240	20/08/2001	22/08/2001	8
1121	07/08/2001	5438300	20/08/2001	22/08/2001	8

Figura 5.16. Cursorul `FACT_TRANSE`

A doua frază `SELECT` determină reducerile efective, pe tranșe, conținutul cursorului creat – `REDUCERI_TRANSE` – fiind prezentat în figura 5.17.





	Nrfact	Datafact	Valtotala	Liminf	Limsup	Procent	Datainc	Transa	Reducere
	1120	07/08/2001	1066240	07/08/2001	17/08/2001	10	16/08/2001	731557	73155
	1118	04/08/2001	2052750	15/08/2001	16/08/2001	9	16/08/2001	1026375	92373
	1118	04/08/2001	2052750	15/08/2001	16/08/2001	9	15/08/2001	1026375	92373
	1111	01/08/2001	5399625	14/08/2001	16/08/2001	8	15/08/2001	5399625	431970
	1112	01/08/2001	1337560	14/08/2001	16/08/2001	8	15/08/2001	487705	39016
	1117	03/08/2001	2320500	16/08/2001	18/08/2001	8	18/08/2001	369680	29574
	1117	03/08/2001	2320500	16/08/2001	18/08/2001	8	17/08/2001	975410	78032
	1117	03/08/2001	2320500	16/08/2001	18/08/2001	8	16/08/2001	975410	78032

Figura 5.17. Cursorul REDUCERI\_TRANSE

În final, se însumează toate reducerile pe facturi și se actualizează atributul `FACTURI.Reduceri`.

Varianta prezentată este una exemplificatoare (se observă că, pentru câteva facturi, valoarea încasată plus reducerile sunt peste valoarea facturii), dar poate fi adaptată la practica românească.

## 5.7. Variabile utilizate în consultări.

### Macrosubstituție și SQL

Utilizarea variabilelor în interogările SQL este utilă mai ales atunci când rapoartele sau informațiile trebuie filtrate calendaristic, geografic etc.

#### 5.7.1. Variabile pentru filtrarea interogărilor

Să începem cu un exemplu simplu. Într-o aplicație avem de listat facturile emise către un client pentru un anumit interval de timp definit printr-o dată inițială și o dată finală. Pentru aceasta se poate redacta un program `l5_13.prg`, care preia cei trei parametri (codul clientului, datele inițială și finală ale intervalului calendaristic) – listing 5.13.

Listing 5.13. Programul L5\_13.PRG

```
PARAMETER codcl_, datai_, dataf_

SELECT DenCl AS Client, Nrfact, DataFact, Gestione, ValTotala AS Valoare ;
FROM CLIENTI C LEFT OUTER JOIN FACTURI F ON C.CodCl = F.CodCl ;
WHERE F.CodCl = codcl_ AND Datafact BETWEEN datai_ AND dataf_
```

Pentru a obține lista facturilor emise clientului 1 (cod 1001) în perioada 2-7 august 2001, se apelează programul de mai sus prin comanda:

```
DO l5_13 WITH 1001, {^2001/08/02}, {^2001/08/07}
```

Cu varianta din listingul 5.14 se poate ameliora afișarea pe ecran.

Listing 5.14. O afișare un pic mai plăcută

```

PARAMETER codcl_, datai_, dataf_

SELECT DenCI AS Client, Nrfact, DataFact, Gestione, ValTotala AS Valoare ;
INTO CURSOR c1 ;
FROM CLIENTI C LEFT OUTER JOIN FACTURI F ON C.CodCI = F.CodCI ;
WHERE F.CodCI = codcl_ AND Datafact BETWEEN datai_ AND dataf_

titlu_ = 'Lista facturilor pentru clientul '+ALLTRIM(c1.Client)+' in perioada '+;
        DTOC(datai_)+ ' - '+DTOC(dataf_)

SELECT c1
BROWSE TITLE titlu_
USE

```

La execuția comenzii

```
DO 15_14 WITH 1001, {^2001/08/02}, {^2001/08/07}
```

se va afișa o fereastră BROWSE de forma celei din figura 5.18.

Client	Nrfact	Datafact	Gestione	Valoare
Client 1 SRL	1115	02/08/2001	002	1651125
Client 1 SRL	1117	03/08/2001	002	6961500
Client 1 SRL	1118	04/08/2001	003	4105500
Client 1 SRL	1120	07/08/2001	003	1066240

Figura 5.18. Interogare parametrizată și titlu dinamic pentru fereastra BROWSE

Se mai cuvine de adăugat că, în afara filtrării și denumirii „dinamice” a ferestrei de vizualizare a rezultatelor interogării, ultimul program „lasă curat” în urma sa, închizând cursorul la ieșirea din BROWSE.

### 5.7.2. Macrosubstituția și fraze SELECT

Să presupunem că o aceeași listă – a facturilor ce conține următoarele date:

- numărul facturii,
- data emiterii,
- numele clientului,
- localitatea în care-și are sediul clientul,
- valoarea încasată până în momentul curent –

trebuie obținută pentru un anumit interval calendaristic, în orice ordine declarată prin toate combinațiile posibile ale celor cinci câmpuri. Listingul 5.15 conține o primă variantă de program.

Listing 5.15. Prima varinată de ordonare dinamică

```

PARAMETER datai_, dataf_, cimp1_, cimp2_, cimp3_, cimp4_

SELECT DenCI AS Client, Loc AS Localitate, Nrfact, DataFact, Gestione, ValTotala AS Valoare ;
FROM CLIENTI C LEFT OUTER JOIN FACTURI F ON C.CodCI = F.CodCI ;

```

```

INNER JOIN LOCALITATI L ON C.CodPost = L.CodPost ;
WHERE Datafact BETWEEN datai_ AND dataf_ ;
ORDER BY &cimp1_, &cimp2_, &cimp3_, &cimp4_

```

Clauza ORDER BY se construiește dinamic cu ajutorul macrosubstituției. Invocarea acestui program se face cu o comandă DO de genul:

```

DO 15_15 WITH {^2001/08/02}, {^2001/08/07}, 'Loc',
'DataFact', 'DenCl', 'ValTotala'

```

În continuare vă propunem o variantă mai elegantă. Ce-ar fi ca, în afara celor două date calendaristice, de început și de sfârșit, ordonarea să fie precizată printr-o listă care să conțină unul, două, trei... sau chiar nici un atribut? Programul din listingul 5.16 preia un parametru de tip șir de caractere care este, de fapt, lista atributelor de ordonare, attribute separate prin virgulă. Așa încât programul analizează virgulele din șir și extrage attributele pe baza cărora se va construi, dinamic, clauza WHERE.

Listing 5.16. Se preia numai data inițială și finală, plus ordinea de prezentare, sub formă de listă de attribute separate prin virgulă

```

PARAMETER datai_, dataf_, ordinea_
i = 1
* se determina prima pozitie a virgulei (asta inseamna cu sunt macar
* doua attribute de ordonare)
poz_ = AT(',', ordinea_, i)

IF poz_ > 0                && exista macar o virgula, deci sunt mai multe attribute de ordonare

* se localizeaza toate virgulele si se extrag attributele cuprinse intre virgule
poz_preced = 0
DO WHILE poz_ # 0
    ii = str(i,1)
    cimp&ii = SUBSTR(ordinea_, poz_preced + 1, poz_ - poz_preced - 1)
    poz_preced = poz_
    i = i + 1
    poz_ = AT(',', ordinea_, i)
ENDDO

* a mai ramas atributul dintre ultima virgula si sfirsitul sirului de ordonare
ii = str(i,1)
cimp&ii = SUBSTR(ordinea_, poz_preced + 1, LEN(ordinea_) - poz_preced)

* se construiesc clauza ORDER BY
order_by_ = ''
FOR j = 1 TO i
    jj = STR(j,1)
    IF j > 1
        order_by_ = order_by_ + ', '
    ENDIF
    order_by_ = order_by_ + ALLTRIM(cimp&jj)
ENDFOR

ELSE && exista maximum un atribut de ordonare

```

```

IF LEN(ALLT(ordinea_)) = 0          && de fapt, nu exista nici un atribut de ordonare
    order_by_ = '1' && ordonarea se face dupa prima coloana
ELSE
    order_by_ = ALLTRIM(ordinea_) && exista un singur atribut de ordonare
ENDIF

ENDIF

* si acum, interogarea !
SELECT DenCI AS Client, Loc AS Localitate, Nrfact, DataFact, Gestiuone, ValTotala AS Valoare ;
FROM CLIENTI C LEFT OUTER JOIN FACTURI F ON C.CodCI = F.CodCI ;
    INNER JOIN LOCALITATI L ON C.CodPost = L.CodPost ;
WHERE Datafact BETWEEN datai_ AND dataf_ ;
ORDER BY &order_by_

```

Iată și câteva moduri de apel a programului:

```

DO 15_16 WITH {^2000/08/02}, {^2001/08/20}, 'valtotala,
    denc1, nrfact'

DO 15_16 WITH {^2000/08/02}, {^2001/08/20}, 'nrfact'

DO 15_16 WITH {^2000/08/02}, {^2001/08/20}, 'valtotala,
    nrfact'

DO 15_16 WITH {^2000/08/02}, {^2001/08/20}, ''

```

### 5.7.3. Calculul automat al valorilor unor attribute agregate

Practica dezvoltării aplicațiilor de lucru cu baze de date a consacrat sintagma *denormalizare*. Frecvent, în tabele sunt incluse attribute redundante, calculate prin însumarea valorilor unor attribute din tabelele-copil. Forțând un pic lucrurile, ne propunem să construim o rutină generalizată căreia să-i trimitem ca parametri numele tabelului-părinte și copil, numele atributului din părinte ce trebuie actualizat, numele atributului din copil pe baza căruia se face actualizarea, precum și atributul-cheie al tabelului-părinte.

Listing 5.17. Generalizarea actualizării valorilor unui câmp calculat

```

*
PARAMETER parinte_, cheie_parinte_, cimp_parinte_, copil_, cheie_copil_, cimp_copil_

* se zerorizeaza cimpul ce urmeaza a fi calculat
UPDATE (parinte_) SET &cimp_parinte_ = 0

* se declara masivul in care va fi preluata valoarea calculata
DIME suma_ (1,1)

SELECT (parinte_)
SCAN
    suma_ = 0
    * valoarea cheii pimate a liniei curente din tabela parinte
    val_cheie_parinte_ = &cheie_parinte_

```

```

* insumarea liniilor-copil
SELECT SUM(&cimp_copil_) INTO ARRAY suma_ FROM (copil_) ;
WHERE &cheie_copil_ = val_cheie_parinte_

* inlocuirea valorii calculate in linia curenta a tablei parinte
SELECT (parinte_)
REPLACE &cimp_parinte_ WITH suma_(1,1)
ENDSCAN

```

Pentru calcularea valorilor atributului `FACTURI.ValTotală` pe baza liniilor-copil din `LINIIFACT`, atributul-copil fiind `LINIIFACT.ValCuTVA`, se invocă programul de mai sus sub forma:

```
do 15_17 with 'facturi', 'nrfact', 'valtotala', 'liniifact',
'nrfact', 'valcutva'
```

Însumarea tranșelor de încasare (`INCASFACT.Transă`) pentru fiecare factură și, astfel, calcularea valorilor atributului `FACTURI.ValIncasata` se va face astfel:

```
do 15_17 with 'facturi', 'nrfact', 'valincasata',
'incasfact', 'nrfact', 'transa'
```

#### 5.7.4. Macrosubstituție și iterații SQL

Tot cu ajutorul macrosubstituției se pot rezolva situații dintre cele mai dificile de obținere a unor rapoarte pretențioase. Spre exemplificare, dorim obținerea unui raport matriceal în care, pe verticală, să fie prezente toate produsele comercializate de firmă, iar „ordonată” să fie constituită din primele patru zile ale lunii august 2001 – ca în figura 5.19.

Denpr	Codpr	Zi_01_08_2001	Zi_02_08_2001	Zi_03_08_2001	Zi_04_08_2001
Produs 1	1	595000	0	1130500	1660050
Produs 2	2	3969245	3034500	1190000	392700
Produs 3	3	357000	0	0	0
Produs 4	4	564060	0	0	0
Produs 5	5	9198700	0	0	0

Figura 5.19. O situație matriceală

Pentru facilitarea prezentării, ne-am limitat la patru zile, dar programul din listingul 5.18 funcționează pentru oricâte zile, singura problemă fiind popularea cursorului `ZILE`. În programul nostru, aceasta se face prin `INSERT-uri` directe, însă în aplicații datele inițiale și finale ale intervalului se pot prelua printr-un formular și, astfel, popularea cursorului se automatizează.

Listing 5.18. Situație matriceală a vânzării produselor pe primele patru zile ale lunii august 2001

```

SET MARK TO '_'                                && separatorul ptr. data calendaristica

CREATE CURSOR Zile (Zi DATE)
INSERT INTO Zile VALUES ({^2001/08/01})
INSERT INTO Zile VALUES ({^2001/08/02})

```

```

INSERT INTO Zile VALUES ({^2001/08/03})
INSERT INTO Zile VALUES ({^2001/08/04})

SELECT DataFact, DenPr, LF.CodPr, SUM(INT(Cantitate * PretUnit * (1+ProcTVA))) AS Valoare ;
INTO CURSOR Zile_Produse ;
FROM FACTURI F INNER JOIN LINIIFACT LF ON F.NrFact = LF.NrFact ;
INNER JOIN PRODUSE P ON LF.CodPr=P.CodPr ;
WHERE DataFact IN ;
  (SELECT Zi ;
   FROM Zile) ;
GROUP BY DataFact, DenPr, LF.CodPr

SELECT DenPr, CodPr ;
INTO CURSOR c1 ;
FROM PRODUSE ;
ORDER BY DenPr

i = 1
SELECT Zile
SCAN

  ii = ALLT(STR(i,2))
  ii_plus_1 = ALLT(STR(i+1,2))
  ziua_ = '_' + DTOC(Zile.Zi)

  SELECT c&ii.*, NVL(ZILE_PRODUSE.Valoare,0) AS Zi&ziua_ ;
  FROM c&ii LEFT OUTER JOIN ZILE_PRODUSE ;
    ON c&ii..CodPr = ZILE_PRODUSE.CodPr AND DataFact = Zile.Zi ;
  INTO CURSOR c&ii_plus_1
  i = i + 1
  SELECT Zile

ENDSCAN

SELECT c&ii_plus_1
BROWSE TITLE 'Vinzari - Produse/Zile'

SELECT Zile
USE
SELECT Zile_Produse
USE

FOR j = 1 TO i
  ii = ALLT(STR(j,2))
  SELECT c&ii
  USE
ENDFOR

```

Câteva explicații n-ar strica în acest moment. Cursorul ZILE conține toate datele calendaristice care trebuie să apară drept coloane ale raportului. Valorile vânzărilor pe produse și zile sunt calculate și stocate în cursorul ZILE\_PRODUSE. Liniile raportului sunt alcătuite din toate produsele firmei, astfel încât cursorul „de start” – c1 – este obținut din tabela PRODUSE.

Numărul iterațiilor depinde de numărul înregistrărilor din cursorul ZILE – numărul zilelor pentru care intersează raportul. Corpul buclei conține joncțiunea cursorului „curent” cu ziua corespunzătoare înregistrării curente din cursorul ZILE. Astfel, dacă c1 nu are

nevoie de explicații, la prima parcurgere a corpului buclei se obține cursorul c2, care, pe lângă cele două coloane din c1, conține un atribut corespunzător primei date din ZILE (1 august 2001) – vezi figura 5.20.

	Denpr	Codpr	Zi_01_08_2001
▶	Produs 1	1	595000
	Produs 2	2	3969245
	Produs 3	3	357000
	Produs 4	4	564060
	Produs 5	5	9198700

Figura 5.20. Cursorul c2

A doua trecere prin buclă se soldează cu un nou cursor – c3 –, vizualizat în figura 5.21.

	Denpr	Codpr	Zi_01_08_2001	Zi_02_08_2001
▶	Produs 1	1	595000	0
	Produs 2	2	3969245	3034500
	Produs 3	3	357000	0
	Produs 4	4	564060	0
	Produs 5	5	9198700	0

Figura 5.21. Cursorul c3

Ultimul cursor este identic celui din figura 5.19. După BROWSE-ul vizualizator, se face curățenie în zonele de lucru, închizându-se cursoarele deschise și care, la un număr mare de zile, ocupă un volum important de memorie.

Este drept că programul poate fi simplificat folosind facilitatea salvării rezultatelor consultării în același cursor din care se extrag datele – vezi listingul 5.19.

Listing 5.19. Versiunea 2 a situației matriceale a vânzării produselor pe primele patru zile ale lunii august 2001

```
*
* ... identic cu l5.18
*

SELECT DenPr, CodPr ;
INTO CURSOR c1 ;
FROM PRODUSE ;
ORDER BY DenPr

SELECT Zile
SCAN
    ziua_ = '_' + DTOC(Zile.Zi)

SELECT c1.*, NVL(ZILE_PRODUSE.Valoare,0) AS Zi&ziua_ ;
FROM c1 LEFT OUTER JOIN ZILE_PRODUSE ;
    ON c1.CodPr = ZILE_PRODUSE.CodPr AND DataFact = Zile.Zi ;
INTO CURSOR c1
```

```
SELECT Zile  
  
ENDSCAN  
  
SELECT c1  
BROWSE LAST title 'Vinzari - Produse/Zile'  
  
SELECT Zile  
USE  
SELECT Zile_Produse  
USE
```