

Baze de date

Sistemul de programe Visual FoxPro

Capitolul 1

Notiuni introductive privind sistemele de baze de date

I. SISTEM DE BAZE DE DATE

Un **sistem de baze de date (SBD)** este un ansamblu de componente corelate între ele, care contribuie la realizarea și exploatarea unei aplicații cu *baze de date*.

O **baza de date (BD)** conține toate informațiile necesare despre obiectele ce intervin într-o mulțime de aplicații, relațiile logice dintre aceste informații și tehnicile de prelucrare corespunzătoare.

Ea este privită ca un ansamblu *organizat*, pe niveluri de organizare a datelor în memoria externă; *coerent*, conform unor restricții de integritate; *structurat* conform unui model de date; cu o *redundanță minimă* și controlată, asigurată printr-o tehnică de proiectare; *accesibil* mai multor utilizatori în același timp.

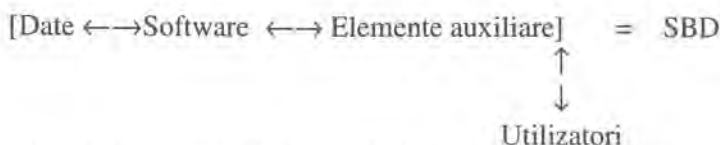
Elementele care compun un *sistem de baze de date* sunt

: *Componenta de bază* (baza de date) care privește modul de memorare a datelor în colecțiile de date și în fișierele anexe;

: *componenta software* (totalitatea programelor necesare stocării și prelucrării datelor în/din BD) (este constituită din sistemul de gestiune a BD (SGBD) precum și din programele de aplicație asociate)

: *componenta auxiliara* ce cuprinde procedurile manuale și automate utilizate (ex: culegerea de informații), legislație asociată, mijloace hardware necesare, personalul implicat.

Arhitectura unui SBD este următoarea



Prin *utilizatori ai unui SBD* înțelegem:

- programatorii de aplicații, administratorul BD (care stabilește structura inițială a bazei de date și modul de memorare a datelor la nivel fizic, modifică structura BD dacă este nevoie, stabilește condiții pentru asigurarea securității și integrității datelor);
- administratorul SBD (care stabilește bazele de date de pe un sistem de calcul, alocă spațiul de memorare și asigură drepturi de acces);
- alte categorii de utilizatori.

Datele sunt organizate în memoria externă în baze de date conform unui *model de date*. Componenta software are rolul de a utiliza/prelucra datele cu ajutorul unui SGBD (sistem de gestiune a bazelor de date) în care se dezvoltă programele de aplicație.

Elementele auxiliare intervin atât în componenta de date cât și în cea de software.

Un **SGBD** este un sistem de programe care permite construirea unor BD, introducerea și prelucrarea informațiilor în/din aceste baze de date. Un SGBD permite accesul utilizatorului la date prin intermediul unui limbaj de nivel înalt și reprezintă o interfață între utilizator și sistemul de operare (SO).

Un SGBD conține

- un limbaj de descriere a datelor (LDD) care permite descrierea structurii unei BD, a fiecărei componente a ei, a relațiilor dintre componente, a drepturilor de acces a utilizatorilor la BD etc; LDD este necesar atât pentru proiectarea bazelor de date cât și pentru redefinirea lor.
- Un limbaj de cereri (LC) sau limbaj de prelucrare a datelor (LPD) care permite efectuarea de operații asupra datelor cum ar fi : încărcarea BD, inserarea/stergerea unui element, modificarea unui element, căutarea unor elemente, realizarea de statistici asupra datelor etc.

Principalele funcții ale unui SGBD sunt:

- Funcția *de definire* a datelor, care permite descrierea structurii BD cu ajutorul LDD.
- Funcția *de manipulare* a datelor, care permite efectuarea de operații asupra datelor proprii LMD (adică operații de încărcare a BD, inserare/stergere unui element, modificarea unui element, căutare a unor elemente etc)
- Funcția *de utilizare* a datelor asigură interfețele necesare comunicării între utilizatori și BD. (Astfel SGBD-ul asigură limbaje și interfețe pentru programatorii de aplicație, instrumente specializate pentru administratorul BD și alte interfețe și instrumente specifice pentru alte categorii de utilizatori.
- Funcția *de administrare* a BD. SGBD-ul asigură, administratorului BD, instrumentele necesare proiectării, exploatării și întreținerii BD (de exp: instrumente pentru încărcarea structurii BD, reorganizarea BD, actualizarea BD, efectuarea de statistici asupra datelor din BD). De asemenea, în colaborarea cu programatorii, administratorul BD poate să-și creeze și alte interfețe care să-l ajute în procesul de gestionare a BD.

I.1 Arhitectura unui SGBD

Nucleul SGBD conține în principal limbajele sistemului LDD și LMD, prin intermediul cărora sunt îndeplinite funcțiile principale ale sistemului.

Interfețele au, în special, rolul de facilita dezvoltarea de aplicații cu baze de date. Aici putem include interfețele de legătură cu alte limbaje de programare cu ajutorul cărora se pot crea aplicații care să constituie noi instrumente de lucru asupra BD, generatoarele sau elementele de proiectare asistată.

Instrumentele sunt componente necesare administrării și întreținerii BD (De exp: depanatoare, editoare, browsere, produse de rețea-necesare administratorului de BD etc.)

II. ARHITECTURA UNEI BD

O baza de date poate fi privita din mai multe puncte de vedere cum ar fi:

Punctul de vedere al utilizatorilor, care lucreaza cu anumite parti componente ale unei BD numite *vederi*.

Punctul de vedere al administratorului BD care integreaza toate vederile referitoare la BD intr-un singur model numit *schema conceptuala* (sau nivel logic al BD).

Punctul de vedere al implementatorului BD care de obicei coincide cu administratorul BD si priveste BD ca pe o colectie de fisiere memorate pe diferite medii externe. Acesta reprezinta *nivelul fizic* al BD.

Primele doua nivele sunt descrise prin planuri ce constau in enumerarea tipurilor de entitati ce apar in BD, relatiile dintre aceste tipuri de entitati si modul de trecere de la notiunile de trecere de la un nivel la cel imediat superior. Aceste planuri se numesc *scheme externe*, *subscheme conceptuale* sau *vederi* pentru primul nivel si *scheme conceptuale* pentru al doilea nivel. Descrierile la nivel fizic sunt facute prin *scheme interne* sau scheme fizice.

II.1 Scheme externe(vederi)

Vederile sunt definite prin intermediul unui sublimbaj de definire a datelor (SLDD) adesea inclus in LDD. Reprezentarea intuitiva a vederilor presupune stabilirea entitatilor, relatiilor dintre ele, a atributelor acestora, a cheilor, a dependentelor functionale a diagramelor etc, notiuni ce vor fi detaliate in subcapitolul Modelarea logica a datelor ce va fi prezentat mai jos. Pentru utilizatorul obisnuit definirea vederilor se face prin intermediul unor comenzi cu structura data, folosind forme predefinite pe care le completeaza sau utilizand un sistem de meniuri.

II.2 Scheme conceptuale

Schema conceptuala se bazeaza pe un model teoretic. Ea combina diferitele subscheme conceptuale (vederi) ce privesc o aplicatie intr-un mod unitar. Modul de descriere si reprezentare este acelasi cu al vederilor si ca si in cazul acestora intervin notiunile de entitate, relatie, atribut, cheie, dependenta functionala, diagrama.

Proiectarea schemei conceptuale a unei BD se face in functie de modelul teoretic ale SGBD-urilor: modelul retea, arborescent si modelul relational. Aceste modele vor fi prezentate mai pe larg in paragrafele de mai jos.

II.3 Scheme interne

Schemele interne descriu diferitele fisiere utilizate pentru memorarea informatiilor bazei de date si modul de operare cu ele. De exemplu exista fisiere cu organizare secventiala, organizare cu dispersie, cu index rar si index dens. De asemenea drept probleme legate de organizarea fisierelor putem aminti pe cele legate de cautarea informatiilor dupa chei secundare sau chei incomplete si memorarea inregistrarilor de dimensiuni variabile.

III. MODELAREA LOGICA A DATELOR

III.1 Modelul entitate –relatie

Modelul utilizat frecvent pentru definirea bazelor de date este cel numit modelul **entitate –relatie**.

Entitatea reprezinta un lucru ce este unic identificabil (Entitatea nu se poate defini formal. Este o notiune primara. Totusi putem incerca o definitie: mai multe elemente de acelasi tip formeaza o entitate.). Exista entitati obisnuite si entitati speciale(care depind de existenta altor entitati).

Proprietatea sau atributul este o alta latura a entitatii si poate lua valori intr-un domeniu asociat. Proprietatile pot fi unice sau compuse, chei (adica identifica in mod unic entitatea respectiva), pot fi omise etc(proprietate necunoscuta sau neaplicabila).

Relatia –defineste o asociere intre entitati. Numarul de entitati care apar intr-o relatie se numeste gradul sau aritatea relatiei. De cele mai multe ori aritatea relatiei este $k=2$ si avem de a face cu relatii binare.

In acest caz, in functie de cate elemente dintr-o entitate corespund la cealalta entitate avem urmatoare clasificare a relatiilor: *relatie unu la unu* (fiecarui element din prima entitate ii corespunde un singur element din a doua si reciproc), *relatie unu la mai multi* (fiecarui element din prima entitate ii corespund mai multe elemente din a doua dar fiecarui element din a doua entitate ii corespunde cel mult unul din prima) si *relatie mai multi la mai multi*(fiecarui element din prima entitate ii corespunde un singur element din a doua si reciproc)

Subtipul unei entitati-este un tip de entitate ce formeaza o submultime a entitatii respective(de obicei cu proprietati suplimentare).

Modelul logic al bazelor de date este reprezentat grafic prin diagrame entitate - relatie.

De exemplu, avem entitatile Facultate de Inginerie, Studenti si Profesori care reprezinta colectii de date privind Facultatea de inginerie, profesorii si respectiv studentii acesteia. Ce proprietati pot fi stabilite pentru aceste entitati? Ce relatii se pot stabili intre ele?

Principalele modele de BD sunt urmatoarele:

1. **Modelul retea.** Datele sunt prezentate cu structura unui graf directionat oarecare. Acest model permite lucrul cu entitati si relatii binare de tip mai multi la mai multi. Nodurile corespund entitatilor iar relatiile sunt reprezentate de sageți între noduri de la tata la fiu.
2. **Modelul ierarhic.** Este un caz particular al modelului retea. Datele sunt prezentate cu structura unui arbore. Acest model permite lucrul cu entitati si relatii binare de tip unul la unu si unu la mai multi si diagrama este alcatuita dintr-o multime de arbori in care toate legaturile sunt pe directia drumului de la radacina la nodul fiu din relatie.
3. **Modelul relational** in care intervin numai relatii si operatii cu aceste relatii. Datele sunt reprezentate sub forma unor tabele. (Tabela este o submultime a produsului cartezian a unor domenii de valori. In tabela sunt coloane, atribute (caracteristici sau campuri) si linii sau *t-upluri* (sau inregistrari) iar legaturile între tabele sunt logice, realizate prin valori).

III.2 Modelul relational. Proiectarea schemei conceptuale a unei BD relationale

-Se bazeaza pe notiunea matematica de **relatie**. Relatia este definita ca reprezentand o submultime a produsului cartezian de multimi finite numite domenii. Elementele unei relatii se numesc t- upluri iar numarul domeniilor reprezinta aritatea relatiei.

Exemplu: Daca D_1, D_2, \dots, D_n sunt domenii atunci orice submultime a produsului cartezian $D_1 \times D_2 \times \dots \times D_n$ se numeste relatie iar (a_1, a_2, \dots, a_n) unde $a_i \in D_i, i \in \{1, 2, \dots, n\}$ este un t-uplu.

O relatie este reprezentata sub forma unui tabel in care fiecare linie reprezinta un t-uplu si fiecare coloana reprezinta valorile t-uplului dintr-un domeniu dat al produsului cartezian.

Coloanelor si respectiv domeniilor li se asociaza nume intitulate attribute. (De exemplu numele coloanei care ia valori in domeniul D_i este $A_i, i \in \{1, 2, \dots, n\}$)

Ansamblul format din numele relatiei si lista numele atributelor acesteia reprezinta schema relatiei (sau schema relationala). Exp: $R(A_1, A_2, \dots, A_n)$.

Cheia unei relatii este un atribut sau o multime de attribute cu ajutorul careora se poate identifica un t-uplu dintr-o relatie.

Se numeste **candidat de cheie** al unei relatii acea cheie ce contine un numar minim de attribute cu ajutorul carora se poate face o identificare unica (exista un singur t-uplu pentru o anumita valoare a cheii). Cu alte cuvinte un **candidat de cheie** al unei relatii R este coloana sau multimea coloanelor din R pentru care valorile din oricare 2 t-upluri nu coincid si nu contin strict o submultime de coloane cu aceasta proprietate.

Pentru fiecare relatie se alege un candidat de cheie care sa va numi **cheia primara**, cu observatia ca t-uplurile unei relatii nu pot avea valoarea nula in coloanele ce apartin cheii primare. Ceilalti **candidati de cheie** se vor numi **chei alternante**.

Cheia este **simpla** daca este formata dintr-un singur atribut sau este **compusa** daca este formata din mai multe attribute.

Cheia externa este o coloana sau o multime de coloane ale unei relatii R ale caror valori, daca nu sunt nule, coincid cu valori ale unei chei primare dintr-o relatie R_1 , nu neaparat distincta de R.

Proiectarea schemei conceptuale a unei BD relationale

In cazul unei BD relationale proiectarea schemei conceptuale se face utilizand tehnica normalizari, care presupune trecerea prin cel putin 3 forme normale.

Forma normala 1.(FN1) O tabela este in forma normala 1 daca toate campurile sunt la nivel elementar (indivizibile) si nu exista campuri repetitive.

De exemplu o baza de date poate contine informatii despre angajatii unei intreprinderi intr-un camp "adresa" in care se precizeaza adresa : localitate, judet, strada, nr. Se observa ca aceasta informatie poate fi divizata in informatii elementare care ar putea fi memorate in campurile "localitate", "judet", "strada", "nr" caz in care am avea satisfacuta conditia FN1.

Forma normala 2.(FN2) O tabela este în forma normala 2 dacă este în FN1 și toate atributele non-cheie (atributele care nu apar în nici o cheie) depind funcțional complet de câmpul cheie. O BDR este în FN2 dacă toate tabelele sale sunt în FN2.

Fie $R(A_1, \dots, A_n)$ o schema relatională și X, Y submultimi de atribute ale lui R . Spunem că X *determină funcțional pe Y* sau că Y *depinde funcțional de X* (scriem $X \rightarrow Y$) dacă nu există în tabelă două t-upluri care să aibă aceeași valoare pentru atributele din X și să aibă valori diferite pentru cel puțin un atribut din Y . Se spune că Y *este complet dependentă funcțional de X* dacă Y este dependentă funcțional de X și nu este dependentă funcțional de nici o submulțime proprie a lui X . De exemplu în relația Depozite(numedep, adresadep, marfa, pret) unde numele de depozite sunt unice iar marfa are același pret putem considera cheia "numedep, marfa". Avem dependentă funcțională $\text{numedep} \rightarrow \text{adresadep}$ (adresadep fiind un atribut noncheie iar numedep o submulțime strictă a cheii), deci tabela nu este în FN2.

Forma normala 3.(FN3) O tabela este în forma normala 3 dacă este în FN2 și nu există dependente tranzitive. (Altfel spus, o relație R este în FN3 dacă pentru orice dependentă $X \rightarrow A$ cu A necontinut în X , fie X conține o cheie fie A apare în cel puțin o cheie). O BDR este în FN3 dacă toate tabelele sale sunt în FN3.

Capitolul 2

SISTEMUL DE PROGRAME VISUAL FOXPRO

I. INTRODUCERE IN VISUAL FOXPRO(VFP)

VFP este un SGBD relational complet deoarece

- este un sistem software, îndeplinește funcțiile și obiectivele unui SGBD;
- este un sistem relational (implementează modelul de date relational ;
- conține un limbaj relational-SQL) și este complet relational (îndeplinește regulile lui Codd).

Visual FoxPro (VFP), spre deosebire de versiunile FoxPro 2.x tratează noțiunea de BD în mod explicit. O *baza de date* VFP reprezintă o colecție de obiecte VFP (tabele (tables), Views și legăturile dintre ele). Fisierul corespunzător BD are extensia DBC.

În VFP *tabelele* pot exista într-o BD, caz în care ea se numește tabelă atașată sau în afara BD și vorbim despre o tabelă liberă. Dacă o tabelă este atașată la o BD atunci, la definirea tabelii se pot specifica reguli de validare la nivelul tabelii, la nivelul înregistrării sau câmpului și valori inițiale pentru câmpuri. Aceste elemente fac ca tabelă să poată fi accesată automat din orice aplicație. Fisierul corespunzător unei tabeli are extensia .DBF și este compus din două zone: structură de date și zonă de date propriu-zisă.

Legăturile între tabele, în versiunile FoxPro 2.x, se puteau face doar dinamic prin proceduri de program sau cu ajutorul comenzii SET RELATION. În afara de această posibilitate, care rămâne valabilă, în VFP se pot defini legături persistente care pot fi realizate cu ajutorul componentei VFP numită Referential Integrity Builder. Aceste legături sunt folosite pentru realizarea rapoartelor, vederilor (Views) sau cererilor de regăsire.

II. TIPURI DE DATE IN FOXPRO

Un *tip de date* reprezintă o caracteristică a datelor prin care se stabilesc operațiile care se pot executa asupra lor, modul de codificare în memoria calculatorului și semnificația acestor date.

Limbajele de programare au implementat o serie de tipuri de date de bază și, de obicei, permit definirea de noi tipuri pe baza celor predefinite. În FoxPro există patru tipuri principale de date: *logic (boolean)*, *numeric*, *șir de caractere* și *data calendaristică*. Un alt tip de date este cel numit *MEMO* și prin intermediul lui sunt manipulate șirurile lungi (sau foarte lungi) de caractere precum și cele de lungime variabilă.

De asemenea *vectorii, matricile sau tablourile multidimensionale* reprezintă un alt tip de date mai complex.

Variabilele reprezintă zone de memorie folosite într-un program pentru memorarea temporară a unor date.

Incarcarea unei date in zona de memorie a unei variabile se face prin instructiunea de atribuire care are sintaxa

<nume variabila>=<expresie>

unde expresie este fie o constanta fie o combinatie valida de constante, variabile, functii si operatori.

Tipul Logic(boolean). Poate lua doar doua valori, adevarat (True) sau fals (False). Valoarea adevarat a unei expresii de tip logic este specificata prin constructia .T. (sau .Y.) iar cea de fals prin .F. (sau .N.). Operatorii logici, in ordinea prioritatii de evaluare sunt :

Operator	Semnificatie
(,)	Grupeaza expresiile
!, Not	Negatie logica
AND	si logic
OR	Sau logic

Exemplu : A=5
? NOT(a=6)
.F.

Tipul Numeric. – este folosit pentru descrierea datelor ce sunt reprezentate de numere. Spre deosebire de alte limbaje de programare care fac distinctie intre numerele intregi si cele rationale (si folosesc tipuri distincte pentru reprezentare) in FoxPro nu se face aceasta distinctie si exista doar tipul numeric pentru manipularea numerelor. In ordinea prioritatii de evaluare operatorii care actioneaza asupra datelor de tip numeric sunt :

Operator	Semnificatie
(,)	Grupeaza expresiile
**, ^	Ridica la putere
*, /, %	Inmultire, impartire, modulo(restul impartirii)
+, -	Adunare, scadere

Intre doua expresii numerice se pot aplica operatori relationali si se obtin expresii logice. Operatorii relationali sunt cei din tabelul de mai jos:

Operator	Semnificatie
<	Mai mic decat
>	Mai mare decat
=	Egal cu
<>, #, !=	Diferit de
<=	Mai mic sau egal
>=	Mai mare sau egal

Funcțiile care operează cu numere sunt :

- funcții referitoare la semnul datelor numerice (de exemplu ABS() care returnează valoarea absolută a parametrului (argumentului), SIGN() care returnează +1 dacă parametrul este pozitiv, 0, dacă este nul și -1 dacă este negativ)
- funcții de aproximare a datelor numerice (de exemplu INT() returnează partea întreagă)
- funcții matematice elementare (EXP(), LOG(), LOG10(), SQRT())
- funcții trigonometrice (SIN(), COS(), TAN(), ASIN(), ACOS(), ATAN())

Tipul Sir de caractere.

Un sir de caractere reprezintă o mulțime ordonată de caractere tratată ca un tot unitar.

Numărul de caractere din sir reprezintă lungimea sirului. Un subsir al sirului dat este o porțiune din sir care începe de la o poziție specificată, de o lungime dată. Sirurile de caractere se scriu între ghilimele sau apostrofuri. Pentru a include unul dintre cele două delimitatoare într-un sir de caractere, sirul va fi încadrat între delimitatoarele de celălalt tip.

Operatorii care acționează asupra sirurilor de caractere sunt: *operatorul de concatenare* (simplu "+" și special "-") (cel special face aceeași operație ca și cel simplu numai că blankurile de la sfârșitul primului sir sunt trecute la sfârșitul celui de al doilea) și operatorii de comparație sau relaționali.

Aceștia din urmă sunt prezentați în tabelul de mai jos:

Operator	Relație
\$	Inclus în
<	Mai mic decât
>	Mai mare decât
<>, #, !=	Diferit de
<=	Mai mic sau egal
>=	Mai mare sau egal
=	identic

Funcțiile pentru prelucrarea sirurilor de caractere se clasifică în:

funcții pentru - aflarea diferitelor informații despre un sir de caractere;

-extragerea unui subsir dintr-un sir în vederea prelucrării lui separat;

-alcatuirea de noi siruri de caractere sau adăugarea de caractere noi la cele existente;

-cautarea unor construcții în cadrul unui sir de caractere;

-aplicarea de diferite transformări asupra caracterelor din sir, cum ar fi trecerea la majuscule.

Câteva dintre **funcțiile** FoxPro care acționează asupra sirurilor de caractere sunt următoarele:

LEN() returnează lungimea unui sir de caractere

SUBSTR() este o funcție cu trei parametri : primul este sirul din care se extrage subsirul, al doilea este o valoare numerică care indică poziția primului caracter din subsir în sirul de bază și ultimul semnifică lungimea subsirului care se extrage. Funcția returnează subsirul extras.

LEFT() primește drept parametri sirul și respectiv lungimea subsirului ce se extrage din partea stângă a sirului de bază (începând cu primul caracter al acestuia).

RIGHT() primește drept parametrii sirul și respectiv lungimea subsirului ce se extrage din partea dreaptă a sirului de baza(începând de la ultimul caracter al sirului de baza spre stanga).

REPLICATE() returnează un sir de caractere obținut prin repetarea sirului primit ca argument de un număr de ori specificat de al doilea argument.

?replicate('ads',5)

adsadsadsadsads

SPACE() returnează un sir de blankuri cu lungimea egală cu numărul transmis ca parametru.

? "aa" + space(5) + "bb"

aa bb

ALLTRIM() elimină blankurile de la începutul și sfârșitul sirului de caractere transmis ca parametru.

LTRIM() elimină blankurile de la începutul sirului de caractere transmis ca parametru.

TRIM() și RTRIM() elimină blankurile de la sfârșitul sirului de caractere transmis ca parametru.

PADC(), PADL(), PADR() admit trei parametrii, primul parametru reprezintă sirul de la care se pleacă, al doilea lungimea la care trebuie să ajungă sirul final iar al treilea parametru arată ce caracter este folosit pentru umplere(dacă el lipsește se folosesc blankuri). PADR() completează sirul inițial la dreapta lui, PADL() la stanga lui iar PADC() la ambele capete (sirul inițial este plasat pe centrul sirului final).

?PADC("aa",15,"v")

wwwaawwww

LOWER() transformă toate majusculele sirului transmis drept parametru în litere mici;

UPPER() transformă toate literele mici ale sirului transmis drept parametru în majuscule;

PROPER() transformă primul caracter dintr-un cuvânt în majusculă(dacă este alfabetic) iar următoarele în litere mici;

CHRTRAN() are trei parametrii, primul reprezintă sirul ce va fi transformat, al doilea reprezintă lista caracterelor ce vor fi înlocuite conform listei de caractere din al treilea sir transmis ca parametru(Primul caracter al celui de-al doilea parametru este înlocuit cu primul din cel de al treilea parametru, al doilea al celui de-al doilea parametru este înlocuit cu al doilea al din cel de al treilea parametru etc. Se realizează un fel de codificare.);

?CHRTRAN("acum","au","xy")

xcym

AT() primește drept argumente două siruri, primul este cel căutat iar al doilea cel în care se caută(ca subsir) primul și returnează poziția la care s-a găsit subsirul în sir sau zero în caz de negăsire;

?AT("Ion","PopescuIon")

8

OCCURS() realizează același lucru ca și funcția AT() numai că returnează numărul de apariții ale subsirului în sirul respectiv.

?OCCURS("Ion","IonPopescuIon")

2

Tipul Data calendaristica

În VFP o constantă de tip data calendaristică se specifică prin luna, zi și an separate prin caracterul '/'. Ordinea de specificare a zilei lunii și anului, modul de scriere a anului (cu două sau patru cifre) cât și separatorul dintre cele trei componente ale datei sunt controlate de comenzi specifice ale limbajului. Implicit anul se specifică prin două cifre, ordinea este ll/zz/aa iar separatorul este '/'. Adică formatul american de data calendaristică. Data calendaristică vidă se specifică prin blancuri în pozițiile celor trei elemente ale ei sau un singur blank între paranteze acolade: { / / } sau { }. Datele calendaristice invalide sunt considerate în FoxPro 2.x ca date vide în VFP este semnalată eroare.

Exp: d={ 14/04/09}

?d={ }

.T.

Formatul de specificare al datelor calendaristice este controlat de comanda SET DATE urmată de unul din următorii parametri:

Tipul datei	Formatul
AMERICAN	ll/zz/aa
ANSI	aa.ll.zz
BRITISH	zz/ll/aa
FRENCH	zz/ll/aa
GERMAN	zz.ll.aa
ITALIAN	zz-ll-aa
JAPAN	aa/ll/zz
USA	ll-zz-aa
MDY	ll/zz/aa
DMY	zz/ll/aa
YMD	aa/ll/zz

Specificarea anului cu două cifre se alege folosind comanda SET CENTURY OFF iar cea cu patru cifre cu SET CENTURY ON.

Funcții privind data calendaristică

DATE() (fără parametru) returnează data curentă a sistemului de operare;

DAY() returnează ziua corespunzătoare unei anumite date calendaristice transmise drept argument (este necesară folosirea anului cu patru cifre);

a=date() && data este 10/29/2006

?day(a)

29

MONTH() și YEAR() returnează luna respectiv anul în aceleași condiții ca funcția DAY();

GOMONTH() are doi parametri, primul reprezintă data de la care se pleacă iar cel de al doilea numărul de luni cu care se avansează (sau devansează în cazul unui număr negativ); rezultatul este o data calendaristică;

De asemenea pentru a avansa/devansa cu un anumit număr de zile se pot folosi operatorii '+' și '-'.

? {02/21/06}+2

02/23/06

Tipul Tablou

“Un tablou este o structura de date ce permite memorarea mai multor valori intr-o zona de memorie continua careia i se atribuie un nume si care poseda un mecanism specific de identificare a elementelor componente”

Pentru declararea unei variabile de tip tablou (unidimensional (vector) sau multidimensional (matrice)) sunt folosite comenzile DIMENSION sau DECLARE care au sintaxa urmatoare:

DIMENSION <nume tablou>[N₁] sau
DIMENSION <nume tablou>[N₁,N₂].

In primul caz este declarat un vector cu N₁ componente numarate de la 1 la N₁, iar in al doilea caz este declarata o matrice cu N₁ linii si N₂ coloane numarate de la 1 la N₁ si respectiv de la 1 la N₂.

In cazul folosirii comenzii DECLARE sintaxa este urmatoarea iar semnificatia este aceeaasi cu cea din cazul de mai sus:

DECLARE <nume tablou>[N₁]
sau
DECLARE <nume tablou>[N₁,N₂]

Obs: In locul parantezelor drepte pot fi folosite si cele rotunde(Exp: DIMENSION b(5))
Elementele tabloului sunt identificate prin pozitia lor in cadrul acestuia: numarul liniei si numarul coloanei in cazul matricilor (<nume tablou>[i,j]) si respectiv pozitia elementului in cadrul vectorului in cazul vectorilor (<nume tablou>[i,j]).

De asemenea elementele unei matrice pot fi identificate printr-un singur numar care reprezinta pozitia in matrice a elementului respectiv, numaratoarea facandu-se astfel: intai se numara prima linie a matricei apoi urmatoarea s.a.m.d pana se ajunge la elementul dorit.

Dupa declararea unui tablou toate elementele acestuia vor fi de tip logic si vor lua valoarea .F. Tipul si valoarea unui element pot fi schimbate prin instructiunea de atribuire. O modalitate speciala de initializare a tuturor elementelor unui tablou se realizeaza tot printr-o instructiune de atribuire in care in locul variabilei de atribuit este introdus numele tabloului.

Exp: DIMENSION T1[2,4]

T1=2
?T1[2,1]
2

Marimea si dimensiunile unui tablou pot fi schimbate printr-o noua comanda DIMENSION sau DECLARE. Astfel se poate transforma un tablou unidimensional intr-unul bidimensional prin copierea elementelor tabloului unidimensional, in ordine, intai in prima linie a celui bidimensional apoi in cea de a doua s.a.m.d. si initializarea cu .F. a elementelor care lipsesc (atunci cand nr. elementelor tabloului unidimensional este mai mic decat cel al celui bidimensional) sau eliminarea din memorie a celor care sunt in plus (atunci cand nr. elementelor tabloului unidimensional este mai mare decat cel al celui

bidimensional). De asemenea se poate transforma un tablou bidimensional intr-unul unidimensional prin copierea linie cu linie a elementelor tabloului bidimensional in cel unidimensional si initializarea cu .F. a celor care lipsesc sau eliminarea din memorie a celor care sunt in plus.

Redimensionarea tablourilor unidimensionale respectiv bidimensionale se face respectand aceleasi reguli prezentate mai sus.

Spre exemplu programul urmator realizeaza redimensionarea unui tablou:

```
dimension t1(2,3)
```

```
for i=1 to 6
```

```
  t1(i)=i
```

```
endfor
```

```
for i=1 to 2
```

```
  ?
```

```
  for j=1 to 3
```

```
    ?? t1(i,j)
```

```
  endfor
```

```
endfor
```

```
dimension t1(3,4)
```

```
for i=1 to 3
```

```
  ?
```

```
  for j=1 to 4
```

```
    ?? t1(i,j)
```

```
  endfor
```

```
endfor
```

Vom obtine pe ecran rezultatele

```
1      2      3
4      5      6
1      2      3      4
5      6      .F.    .F.
.F.    .F.    .F.    .F.
```

Funcții pentru prelucrarea tablourilor:

ALen() este o functie care admite doi parametrii, primul este numele tabloului iar al doilea este o valoare numerica care primeste valoarea 0 daca se doreste ca functia sa returneze numarul de elemente al tabloului, 1 daca se cauta numarul de linii al tabloului(sau numarul de elemente pentru vectori), 2 cand se doreste sa se afle numarul de coloane ale tabloului.

In cazul tabloului din exemplul de mai sus avem:

```
?ALen(T1,2)
```

```
4
```

```
?ALen(T1,0)
```

```
12
```

```
?ALen(T1,1)
```

```
3
```

Functia AINS() realizeaza inserarea de elemente, linii sau coloane intr-un tablou.

In cazul inserarii de linii sau elemente AINS() are doi parametrii, primul este numele tabloului in care se doreste inserarea iar cel de al doilea reprezinta, in cazul vectorilor,

poziția în care va fi inserat un nou element iar în cazul matricilor numărul unei noi linii inserate în matrice. Dacă se dorește inserarea unei coloane funcția mai primește un al treilea parametru egal cu 2 și valoarea celui de al doilea parametru va reprezenta numărul coloanei ce va fi inserată în matrice. Inserarea de elemente conduce la pierderea elementelor care nu mai încap în urma inserării. Dimensiunea tabloului nu se modifică.

ADEL() este funcția opusă lui AINS() și apelul ei este asemănător cu cel al funcției AINS. Ea determină ștergerea unui element a unei linii sau a unei coloane într-un tablou. ACOPY() permite copierea elementelor dintr-un tablou Tablou1 în altul Tablou2 și returnează o valoare numerică ce reprezintă numărul de elemente copiate în tabloul destinație Tablou2.

Are sintaxa

ACOPY(<Tablou1>, <Tablou 2> [, <N1>
[, <N2> [, <N3>]]])

Dacă tabloul destinație nu există atunci FoxPro îl creează automat și acesta va avea aceeași dimensiune ca și cel sursă.

N1 este opțional și reprezintă numărul primului element din tabloul sursă care va fi copiat în tabloul destinație. Dacă N1 nu apare copierea începe cu primul element al tabloului sursă.

N2 specifică câte din elementele tabloului sursă vor fi copiate (începând cu elementul N1). Dacă N2=-1 atunci vor fi copiate toate elementele tabloului sursă copiate (începând cu elementul N1).

N3 specifică primul element în tabloul destinație ce va fi înlocuit.

Exp.

CLOSE DATABASES

```
SELECT DISTINCT company ;
      FROM customer ;
      ORDER BY company ;
      WHERE state = 'WA';
      INTO ARRAY companies
```

= ACOPY(companies, temp_comp) && Make a copy of the companies array

ADIR() încarcă într-un tablou informațiile despre fișierele din directorul curent;

AFIELDS() admite un parametru, numele unui tablou și încarcă într-un masiv structură tabelii curente și returnează numărul de câmpuri din tabel. Se poate folosi comanda COPY STRUCTURE EXTENDED pentru a copia aceeași informație într-o tabelă în loc de un tablou.

Tabloul în care încarcă funcția AFIELDS() structura unei tabeli (dacă nu există va fi creat automat de FoxPro) are patru coloane și un număr de linii egal cu numărul de câmpuri din tabelă. Tabelul de mai jos prezintă care va fi conținutul fiecărei coloane și tipul de date ce va apărea în fiecare coloană

NR. Coloanei	Informația despre câmp	Tipul de date
1	Numele câmpului	Caracter
2	Tipul câmpului	Caracter (C, D,L, M, N, G,

		P)
3	Lungimea campului	Numeric
4	Numarul de zecimale	Numeric

ASCAN() cauta o expresie într-un tablou;

ASORT() ordoneaza elementele unui tablou dupa un anumit criteriu.

III. CONVERSII INTRE TIPURILE DE DATE DE BAZA

Aceasta operatie este necesara atunci cand într-o expresie este necesara prezenta a doua date de tipuri diferite (de exemplu in cazul indexarii, in cazul in care trebuie construta o expresie care contine doua campuri cu tipuri de date diferite ale tabelii).

Conversia intre o data calendaristica si un sir de caractere

Funcțiile

DTOC() și DTOS() au un singur argument ce reprezinta o data calendaristica și returneaza un sir de caractere, care reprezinta respectiva data calendaristica in formatul stabilit de comanda SET DATE in cazul functiei DTOC() sau in formatul aaaallzz in cazul functiei DTOS().

Conversia inversa, din sir de caractere in data calendaristica este asigurat de functia CTOD() care primeste drept argument un sir de caractere și returneaza o data calendaristica.

Conversia intre un sir de caractere si un numar

Funcția

STR(<N>, <N1>, <N2>) primeste drept parametrii pe <N>, numarul de transformat în sir, <N1> ce semnifica lungimea totala a sirului și <N2>, ce reprezinta numarul de cifre ce se vor folosi pentru partea fractionara a numarului.

VAL(<sir caractere>) primeste drept argument sirul de caractere ce va fi transformat într-un numar.

Funcții pentru diferite tipuri de date

Funcția

EVALUATE(<sir caractere>) primeste drept argument sirul de caractere ce va fi evaluat și apoi transformat corespunzator într-o data de tip sir de caractere, numeric sau data calendaristica.

Exp: ?EVALUATE('8/2+2*3')

10

? EVALUATE(replicate('a', 6)+'bb')

aaaaaabb

?DATE()

10/29/06

? EVALUATE(DATE()+5)

11/03/06

EMPTY() primește drept argument expresia ce va fi evaluată și returnează un rezultat de tip logic, adevărat dacă expresia de testat este vidă și fals în caz contrar. Semnificația termenului vidă pentru diferite tipuri de date este dată în tabelul de mai jos

Tipul datei	Semnificația termenului vidă
Sir de caractere	Contine numai blăncuri (CHR(32)), nuluri(CHR(0)), tab-uri (CHR(9)), sfărsit de linie (CHR(13), CHR(10))
Numeric	Este 0
Data calendaristică	Data vidă { / / }, { }
Logic	Este FALSE
Memo	Fără conținut

MAX(), MIN() primesc drept argumente două sau mai multe expresii de același tip care urmează a fi comparate. Prima funcție returnează valoarea maximă iar cea de-a doua valoarea minimă.

INLIST(<expr1>, <expr2>

[, <expr3>...]) Determină dacă prima expresie primită ca argument se regăsește sau nu în lista delimitată sau specificată de <expr2> [, <expr3>...]. Ea returnează o valoare logică, adevărată dacă găsește expresia în lista de expresii și fals în caz contrar.

Multimea de expresii în care este căutată prima expresie este reprezentată prin <expr2>, <expr3> etc.. Trebuie inclusă cel puțin o expresie (<expr2>), și cel mult 24. Toate expresiile din lista vor avea același tip (caracter, numeric, logic sau de tip dată calendaristică).

Exemplu

luna='Ianuarie'

? INLIST(luna,'Ianuarie','Februarie','Martie')

.T.

IV. CREAREA BAZELOR DE DATE ȘI A TABELELOR

IV.1 Crearea bazelor de date

La începerea unui proiect pentru o bază de date, prima etapă în administrarea datelor va fi întotdeauna crearea bazei de date.

Sintaxa tipică pentru instruirea CREATE DATABASE arată astfel:

CREATE DATABASE [<nume_baza_de_date>|?];

- nume_baza_de_date specifică numele bazei de date ce va fi creată. Fișierul corespunzător bazei de date are extensia .DBC.

-? Afisează o fereastră de dialog în care se poate specifica numele bazei de date ce va fi creată.

La crearea unei baze de date se face automat și deschiderea ei.

Observatie: 1. Adaugarea unei tabele intr-o baza de date se face cu comanda ADD TABLE.

2. Comenzile foxPro se pot scrie (continua) pe mai multe randuri daca aceste randuri sunt despartite intre ele prin caracterul < ; >. Numarul maxim de caractere intr-o linie de comanda este de 2048.

```
Exemplu: SELECT customer.cno, customer.contact, customer.city, customer.state, ;
          customer.zip ;
FROM customer ;
ORDER BY customer.state, ;
          customer.zip
```

Comentariile se scriu dupa caracterul &&.

IV.2 Crearea tabelor

Crearea tabelor se poate realiza utilizand comanda CREATE sau CREATE TABLE din nucleul SQL si optiunea New a submeniului File.

1. Comanda CREATE

Sintaxa:

CREATE [<numefis> | ?]

Efect:

Creaza noi tabele FoxPro.

Se poate specifica un nume (<numefis>) pentru tabela ce va fi creata. Daca acest lucru nu este specificat sau daca este inclus caracterul ?, atunci este deschisa o fereastră de dialog care permite introducerea numelui tabelii ce va fi creata. Nu sunt acceptate numele: CON, NUL, PRN, COM1 ...(cuvinte rezervate). Comanda CREATE [<numefis>] conduce la deschiderea ferestrei **Structure** unde pot fi specificate numele campurile si caracteristicile acestora respectiv tipul, lungimea si respectiv numarul de zecimale , daca este cazul, pentru fiecare camp. Pentru deplasarea in interiorul acestei ferestre se utilizeaza sagetile directionale, [tab], [ctrl] + [PgUp] sau [PgDn]. Fereastra contine urmatoarele optiuni: **Name**, **Type** (tipul), **Width** (precizia), **Dec** (scala), **Ok** si **Cancel**. Sub **Name** se tasteaza numele campului format din maximum 10 caractere, incepand cu o litera, iar sub **Type** se tasteaza tipul campului nou creat, sau se alege o optiune din meniul pop-up afisat. Putem avea urmatoarele tipuri: sir de caractere (C), numeric (N), data calendaristica (D), logic (L), memo (M), general (G) sau imagine (P) si o descrierea a lor este data in tabelul de mai jos. Unele tipuri de date necesita specificarea <preciziei> (lungimea campului sau numarul maxim de caractere pe care il pot ocupa datele memorate in campul respectiv) si <scala> (nr.de zecimale).

<Tipul>, <precizia> si <scala> pot fi:

Tip (Type)	Precizie (Width)	Scala Dec	Descriere
C	N	-	Sir de caractere de lungime n. Admite maximum 254

			caractere(litere, numere, simboluri speciale, spatii goale)
D	-	-	Data calendaristica. Admite maximum 8 caractere si implicit are formatul ll/zz/aa. Formatul poate fi schimbat cu comanda SET DATE TO
F	N	d	Real, de latime n, cu d zecimale
L	-	-	Logic. Admite maximum un caracter T sau Y , F sau N.
M	-	-	Memo admite implicit 10 caractere, dar sistemul poate stoca blocuri mari de text pentru fiecare inregistrare. Dimensiunile blocurilor text sunt limitate de spatiul liber pe disc. Tabelele care contin campuri Memo sunt stocate sub forma a doua fisieresi anume, unul cu extensia obisnuita .DBF si altul cu extensia .FPT. Cel din urma este un fisier in care se salveaza campurile memo si care se creeaza automat la salvarea tabeli.
N	N	d	Numeric, de latime n, cu d zecimale. Admite maximum 20 de cactere(numere inclusiv +/- si .)
G	-	-	General. Gazduieste elemente de tip OLE(Object linking and embedding) texte, foi de calcultabelar, imagini, benzi sonore ale altor aplicatii Windows.
P	-	-	Imagine

Introducerea datelor in campurile de tip Memo sau General este diferita de introducerea in alte campuri. Cand cursorul este pozitionat pe campul MEMO se apasa [ctrl] +[PgDn] sau [PgUp] si apare fereastra cu acelasi nume ca al campului MEMO creat in structura tabeli. Este tastat textul dorit folosind facilitatile editorului FoxPro. Textele sunt salvate cu [ctrl] + [W], fereastra este inchisa si cursorul revine in fereastra de adaugare a datelor. Selectare optiunii Ok sau tastand [Ctrl]+[enter] se salveaza structura creata iar folosirea optiunii Cancel intrerupe crearea structurii tabeli de date. Odata ce structura tabeli este creata, se pot adauga inregistrari tabeli. Daca structura este salvata apare un mesaj Input data records now? (Y/N)

Se tasteaza [Y] sau [N] dupa cum se doreste sau nu introducerea de inregistrari in tabela. Datele se salveaza si apasand tastele [Ctrl]+[W] sau {ctrl}+[End]. Cu tasta [Esc] sau [Ctrl]+[Q] se incheie introducerea, fara salvarea datelor si cursorul se pozitioneaza in fereastra de comanda.

Daca este folosita varainta CREATE ? atunci este deschisa fereastra de dialog Sava as cu ajutorul cariaa poate fi specificat discul logic, directorul si numele tabelii ce va fi creata. Alegand optiunea **Create** (tastand [Enter]) este activata fereastra **Structure** care permite definirea structurii tabelii asa cum a fost aratat mai sus.

2. Comanda CREATE TABLE din SQL

Sintaxa:

```
CREATE TABLE < numele tabelii >  
(<numecamp1> <tip camp>  
    [(<precizie> [, <scala>])  
    [, < numecamp2> ...]])  
| FROM ARRAY <tablou>
```

Efect: Comanda CREATE TABLE creaza tabele cu specificarea campurilor. Fiecare camp nou este definit prin precizarea numelui, tipului, lungimii si numarului de zecimale(daca este cazul) dupa cum am arata mai sus. Aceste definitii pot fi obtinute prin comanda sau pot fi preluate dintr-un tablou (array).

Noua tabela este deschisa in cea mai mica zona de lucru disponibila si poate fi accesata prin alias-ul sau.

CREATE TABLE <numele tabelii>

In clauza de mai sus, < numele tabelii > specifica numele tabelii ce va fi creata. < numele tabelii > poate include calea sau poate fi doar un identificator. Tabela este deschisa intr-o noua zona de lucru (daca o alta tabela este deschisa in zona de lucru curenta) si aceasta devine cea curenta.

```
(<numecamp1> <tip camp>  
    [(<precizie> [, < scala >])  
    [, < numecamp2> ...]])
```

In clauza de mai sus, <numecamp1> si <numecamp2> sunt nume de campuri in noua tabela. <numecamp1> este un identificator prin care se va realiza accesul la campul respectiv.

<tip> este tipul campului, adica tipul datelor ce vor fi memorate in campul respectiv si este indicat printr-o singura litera (vezi tabelul prezentat mai sus). Specificarea <preciziei> si <scalei> sunt ignorate pentru tipurile D, L, M, G si P. Daca nu sete precizata <scala> pentru tipurile N si F atunci ea ia implicit valoarea zero (fara locuri pentru zecimale).

FROM ARRAY <array>

În clauza de mai sus <array> este numele unui tablou existent al cărui conținut reprezintă numele, tipul, precizia și scala pentru fiecare câmp din tabel. Conținutul unui tablou (array) poate fi definit folosind funcția AFIELDs().

Exemplul de mai jos creează o tabelă și afișează structura sa.

```
CLOSE ALL
CREATE TABLE angajati ;
    (nume C(20), adresa C(30), oras C(30), codpostal C(5), salariu N(8,2), ;
    comentarii M)
DISPLAY STRUCTURE
```

3. Alte comenzi pentru crearea tabelor

O altă comandă utilă pentru crearea de tabele este următoarea

```
COPY STRUCTURE TO <fisier>
    [FIELDS <lista de câmpuri>
    | FIELDS LIKE <condiție>
    | FIELDS EXCEPT <condiție>]
    [[WITH] CDX | [WITH] PRODUCTION]
```

Efect: Copiază structura tabelii curente într-o nouă tabelă.

Comanda COPY STRUCTURE creează o tabelă goală numită <fisier> cu aceeași structură ca și tabelă curentă.

Dacă este inclusă clauza FIELDS < lista de câmpuri >, sunt copiate în nouă tabelă numai câmpurile al căror nume apare în lista < lista de câmpuri >. Dacă această clauză este omisă atunci vor fi copiate toate câmpurile în nouă tabelă.

Putem opta pentru copierea selectivă a câmpurilor în nouă tabelă folosind clauzele FIELDS LIKE < condiție > | FIELDS EXCEPT < condiție > (cate una sau amândouă). Dacă folosim LIKE < condiție >, atunci FoxPro copiază doar câmpurile care satisfac < condiție > iar dacă este folosită clauza EXCEPT < condiție >, FoxPro copiază toate câmpurile cu excepția celor care satisfac < condiție > în nouă tabelă.

Condiția suportă caractere speciale. De exemplu pentru a copia câmpurile care încep cu literele A și P în nouă tabelă folosim:

```
COPY STRUCTURE TO tabel1 FIELDS LIKE A*,P*
```

Clauza LIKE poate fi combinată cu EXCEPT :

```
COPY STRUCTURE TO tabel1 FIELDS LIKE A*,P* EXCEPT PARTEN*
```

Clauzele [WITH] CDX | [WITH] PRODUCTION permit crearea unui fisier index structural pentru noua tabela. Etichetele si expresiile index din fisierul index strutral original sunt copiate in noul fisierul index strutral. Cele doua clauze au acelasi efect.

De asemenea clauza SCATTER cu

Sintaxa:

SCATTER

```
[FIELDS <field list>
| FIELDS LIKE <conditie>
| FIELDS EXCEPT <conditie>]
[MEMO]
TO <array> | TO <array> BLANK
| MEMVAR | MEMVAR BLANK
```

are drept efect copierea datelor din inregistrarea curenta într-un tablou (array) sau într-o multime de variabile.

SCATTER creaza automat variabile de memorare sau tablouri, daca acestea nu exista deja. Clauzele [FIELDS <lista de campuri>

```
| FIELDS LIKE <conditie>
| FIELDS EXCEPT <conditie>] au acelasi efect ca si cele descrise mai sus.
```

Obs: Comanda GATHER copiaza valorile variabilelor de memorie sau elementele tablourilor ca inregistrari într-o tabela.

Exemplu de mai jos defineste un tablou cu o linie si 4 coloane si apoi creaza tabela cu numele REZERVA prin preluarea structurii acesteia din tablou.

CLOSE DATABASES

CLEAR

USE angajati

DISPLAY STRUCTURE

COPY STRUCTURE EXTENDED TO temporar

USE temporar

DIMENSION tablou[1,4]

SCATTER TO tablou

CREATE DBF angajati1 FROM ARRAY tablou

DISPLAY STRUCTURE

4. Utilizarea ferestrei New

In sfarsit asa cum am spus mai sus o tabela poate fi creata cu ajutorul ferestrei New. Este aleasa optiunea New a submeniului File. Apare fereastra de dialog New care contine urmatoarele optiuni:

() Table/DBF- selectand aceasta optiune se creeaza o tabela;

() Program- selectand aceasta optiune se poate crea un program utilizand editorul FoxPro sau un alt editor de texte;

- () File- selectand aceasta optiune se poate crea un fisier utilizand editorul FoxPro;
 - () Index- selectand aceasta optiune se lanseaza fereastra de dialog **Index** pentru tabela de date curenta;
 - () Label- selectand aceasta optiune se activeaza editorul de etichete ce permite generarea unei etichete;
 - () Report- selectand aceasta optiune se activeaza editorul de rapoarte ce permite generarea unui raport;
 - () Form- selectand aceasta optiune se activeaza generatorul de ecrane formatate;
 - () Menu- selectand aceasta optiune se activeaza generatorul de meniuri orizontale;
 - () Query- selectand aceasta optiune se deschide fereastra de dialog **RQBE** pentru generarea cererilor. In cazul in care nu este deschisa nici o tabela, va aparea intai fereastra de dialog **Open file**, apoi fereastra **RQBE**;
 - () Project- selectand aceasta optiune se afiseaza fereastra generatorului de proiectecare va contine fisiere program, meniuri, rapoarte si fisiere ecran interconectate.
 - () Class- creeaza o clasa de obiecte;
 - () Text file- creeaza un fisier text;
 - () View- creaeaza o tabela virtuala;
 - () Remote View- creaeaza o tabela virtuala ce utilizeaza ca surse date din afara bazei de date create;
 - () Connection- proiecteaza o definitie stocata in baza de date ce specifica numele sursei de date;
- Pentru crearea unei tabele este selectata optiunea () Table/DBF se apasa OK si vor fi deschise pe rand fereastra **Structure**, cu ajutorul careia se creeaza structura tabelii si apoi fereastra **Directory** unde este ales discul logic, directorul si numele tabelii ce va fi creata.

IV.3 Modificarea structurii unei tabele

Comanda MODIFY STRUCTURE

Sintaxa:

MODIFY STRUCTURE

Efect:

Permite modificarea structurii tabelii active.

Cu ajutorul comenzii de mai sus, in cazul in care nu este deschisa nici o tabela in zona de lucru curenta, se deschide o fereastra de dialog pe baza careia se stabileste tabela a carei structura dorim sa o modificam. Este afisata fereastra *Select Database*. Este aleasa tabela pe care dorim sa o modificam . Se apasa tasta [Enter] si va aparea fereastra de dialog Strcture utilizata si la definirea structurii tabelii. U ajutorul tastelor [Ins], [Del], si [Bakspace] se adauga sau se sterg campuri si se pot modifica cele existente din punct de vedere al numelui, tipului si marimii.

Pentru a salva modificarile facute se apasa tasta [Enter] si va fi afisat mesajul

Make structure changes permanent?

<<Yes>>

<<No>>

Raspunsul Yes este urmat de reintoarcerea in fereastra de comanda.

FoxPro face automat o copie backup a tabelii curente inainte de a schimba structura tabelii. Cand mofificarea este completa, datele continute in copia backup sunt incarcate inregistrările care apar si in structura noua a tabelii. De asemenea este creat si un fisier memo backup. Fisierul backup al tabelii are extensia .BAK, iar fisierul memo backup are extensia .TBK.

Obs. Atentie la tipul de date al campurilor. De exemplu, daca dorim sa convertim un camp de tip data calendaristica intr-unul numeric continutul campului nu va fi transferat din copia backup.

Daca sunt probleme cu comanda MODIFY STRUCTURE, putem sterge noile fisiere si vom renumi fisierele .BAK si .TBK, daca exista, cu extensiile originale (.DBF and .FPT). si vom obtine tabela initiala nemodificata.

IV.4 Deschiderea si inchiderea tabelilor in zone de lucru

Comanda USE

Sintaxa:

```
USE [<file> | ?]
    [IN <expN1>]
    [AGAIN]
    [INDEX <lista de fisiere index> | ?
        [ORDER [<expN2>
            | <idx index file>
            | [TAG] <tag name>
            [OF <cdx file>]
            [ASCENDING | DESCENDING]]]]
    [ALIAS <alias>]
    [EXCLUSIVE]
    [SHARED]
    [NOUPDATE]
```

Efect: Comanda USE are rolul de a deschide o tabela/BD precum si fisierele index asociate.

USE deschide o tabela/BD n zona de lucru curenta.

Daca USE este folosit fara numele tabelii/BD si tabela/BD este deschisa in zona de lucru curenta, tabela/BD este inchisa. O tabela/BD este de asemenea inchisa cand alta tabela/BD este deschisa in zona de lucru curenta.

<file> | ?

Numele tabelii/BD deschise este specificat prin <file>. Daca se alege ? in loc de numele unei tabeli/BD atunci se deschide o fereastra de dialog cu o lista de tabeli/.BD disponibile pentru a face o alegere.

IN <expN1>

Cu ajutorul clauzei IN se poate deschide o tabela/BD intr-o zona de lucru diferita de cea curenta. Specificarea zonei de lucru se face cu ajutorul numarului <expN1> care o reprezinta. Putem inchide o tabela/BD intr-o zona de lucru diferita de cea curenta folosind comanda USE fara nume de fisier dar specificand numarul zonei de lucru.

Clauza IN suporta valoarea 0 ca numar al zonei de lucru. Se foloseste 0 pentru a deschide o tabela/BD in cea mai joasa zona de lucru disponibila. De exemplu daca tabelele/BD sunt deschise in toate din primele 10 zone de lucru atunci comanda

USE client IN 0

deschide tabela/BD "client" in zona de lucru 11.

AGAIN

Pentru a deschide o tabela/BD simultan in mai multe zone de lucru, trebuie procedat in unul din urmatoarele moduri:

1. Selectati o alta zona de lucru si folositi USE cu numele tablei/BD si clauza AGAIN.
2. Folositi USE cu numele tablei/BD si clauza AGAIN si specificati o alta zona de lucru.

Cand se foloseste o tabela/BD din nou in alta zona de lucru, tabela/BD respectiva pastreaza atributele tablei/BD din zona de lucru originala. De exemplu, daca o tabela/BD este deschisa numai pentru citire si este folosita din nou intr-o alta zona de lucru, tabela/BD ea este deschisa numai pentru citire in noua zona de lucru.

Fisierele index deschise pentru tabela/BD originala sunt disponibile pentru tabela/BD deschisa din nou daca nu s-au deschis fisierele index cand s-a redeschis tabela/BD.

De asemenea se pot deschide indecsi care nu au fost deschisi cu tabela/BD originala.

Unei tabel/Bd deschise ii este atribuit aliasul implicit al zonei de lucru. Se poate include un alias de fiecare data cand se deschide o tabela/BD in mai multe zone de lucru atata timp cat aliasii sunt unici.

INDEX <lista de fisiere index> | ?

Pot fi deschise fisiere index cu tabela/BD incluzind clauza INDEX si specificind multimea de fisiere index prin <lista de fisiere index>. Daca o tabela/BD are fisiere index structurale compuse, atunci fisierele index sunt deschise automat cu tabela/BD. Fisierele index simplu pot fi deschise si folosind INDEX ?. Atunci se deschide o fereastra de dialog cu o lista de fisiere index disponibile.

<lista de fisiere index> poate contine orice combinatie de nume de fisiere index de tipul .IDX si .CDX . In general nu trebuie inclusa si extensia acestora in comanda, cu

exceptia cazului in care doua fisiere .IDX si .CDX din lista de fisiere index au acelasi nume.

Primul fisier index din the lista de fisiere index este cel activ, care controleaza care inregistrari din tabela/BD sunt accesate si afisate. Daca primul fisier index este un fisier .CDX compus, inregistrarile din tabela/BD sunt afisate si accesate in ordinea fizica de inregistrare.

ORDER [<expN2>]

Clauza ORDER specifica fisierul index sau eticheta(tag) activ(a), alta decat primul/a din <lista de fisiere index>. <expN2> specifica fisierul index sau eticheta(tag) activ(a) asa cum apare in lista de fisiere index. Fisierele .IDX sunt numarate in ordinea in care apar in lista de fisiere index. Etichetele din fisierele structurale compuse (daca exista) sunt numarate in ordinea in care au fost create(etichetele). In sfarsit, etichetele din alte fisiere compuse(nestructurale) sunt numarate in ordinea in care au fost create. De asemenea poate fi folosita clauza SET ORDER pentru a specifica fisierele index/ etichetele active..

Daca <expN2> este 0, inregistrarile din tabela/BD sunt afisate si accesate in ordinea fizica a inregistrarilor si indexii raman deschisi. Folosirea clauzei ORDER 0 nu permite updatarea fisierele index. Folosirea clauzei ORDER fara <expN> este identica cu ORDER 0.

ORDER [idx fisier index]

ORDER [[TAG <tag name>]

[OF <cdx file>]]

De asemenea se poate alege fisierul index .IDX activ prin specificarea numelui acestuia cu <idx fisier index>.

Pentru a desemna o eticheta (tag) a unui fisier index .CDX ca eticheta activa, este specificat numele ei prin <tag name>. Numele ei poate fi dintr-un fisier index structural compus sau din orice fisier index compus deschis. Daca exista nume identice pentru etichete in fisiere index compuse deschise, trebuie inclusa clauza OF <cdx file> si specificat numele fisierului index compus ce contine eticheta dorita.

ASCENDING | DESCENDING

Cuvintele cheie ASCENDING sau DESCENDING care urmeaza clauzei ORDER specifica daca inregistrarile tabelii/BD sunt accesate si afisate in ordine ascendenta sau descendenta. Includerea unuia din aceste cuvintele cheie nu schimba fisierul index/eticheta (tag) activa ci doar ordinea in care inregistrarile sunt afisate si accesate.

ALIAS <alias>

Folosim ALIAS <alias> pentru a crea un alias pentru tabela/BD. Ne putem referi la tabela/BD prin aliasul ei in comenzile si functiile care necesita sau suporta alias-uri.

Cand o tabela/BD este deschisa, I se asociaza automat un alias, care este numele tabelii/BD daca ALIAS nu este folosita. Pot fi create alias-uri diferite pentru tabela/BD folosind comanda ALIAS si un nou alias. Un alias poate contine can contain up to 10 letters, digits or underscores and must begin with a letter or an underscore.

Este asociat un alias implicit daca deschidem o singura tabela/BD simultan in mai multe zone de lucru cu ajutorul clauzei AGAIN si nu a fost specificat un alias la deschiderea tabelii/BD in fiecare zona de lucru.

De asemenea este folosit un alias implicit daca apare un conflict. De exemplu:

CLOSE DATABASES

```
USE clienti ALIAS facturi IN 1      && Alias este "facturi"
USE facturi IN 3                    && Conflict, alias este C
```

EXCLUSIVE

Clauza EXCLUSIVE este folosita pentru deschiderea unei tabele/BD pentru utilizare exclusiva intr-o retea. See SET EXCLUSIVE for more information on the exclusive use of tables/.DBFs.

SHARED

Clauza SHARED este folosita pentru deschiderea unei tabele /BD pentru utilizare partajata intr-o retea. SHARED permite deschiderea tabelii/BD pentru utilizare partajata chiar si cand este activa clauza EXCLUSIVE (este setat ON). See SET EXCLUSIVE for more information on exclusive and shared use of tables/.DBFs.

NOUPDATE

Clauza NOUPDATE previne schimbarea structurii tabelii/BD.

Example: Acest exemplu deschide 3 tabele in 3 zone de lucru diferite. Fereasta View este deschisa pentru a arata unde sunt deschise tabelele si aliasurile lor.

CLOSE DATABASES

```
ACTIVATE WINDOW View
SELECT A
USE client IN 0
USE facturi IN 0
USE vanzatori IN 0
```

Functia USED([<expresie numerica>]|<sir de caractere>]) returneaza valoarea logica .T., respectiv .F. daca tabela este deschisa, respectiv nu este deschisa in zona de lucru specificata. [<expresie numerica>]|<sir de caractere>] specifica numarul <expresie numerica> zonei de lucru, respectiv aliasul <sir de caractere> tabelii. Daca se specifica un alias functia USED() returneaza valoarea logica .T. daca tabela cu aliasul respectiv este deschisa intr-una din zonele de lucru. Daca nu este specificat nici numarul zonei de

lucru, nici aliasul tabelii functia raspunde la intrebarea : Este zona de lucru curenta ocupata?

Exemplu: USE agenti in 1

?USED(1)

.T.

Functia ALIAS([<expresie numerica>]|<sir de caractere>]) returneaza sinonimul zonei de lucru specificate. Daca nu are argumente >]) returneaza sinonimul zonei de lucru curente.

Exemplu: USE agenti in 1

?ALIAS(1)

agenti

Inchiderea unei tabele se face cu comanda CLOSE care are urmatoarea sintaxa si efectul prezentat mai jos

CLOSE ALL -inchide toate tipurile de fisiere din toate zonele de lucru si selecteaza zona de lucru 1. De asemenea ea inchide ferestrele Label, Project, Report, RQBE, screen si nu afecteaza ferestrele Command, Debug, Desk Accessories, Help, Trace.

CLOSE DATABASES – inchide toate tabelele de date deschise precum si toate fisierele index aferente si selecteaza zona de lucru 1.

CLOSE FORMAT inchide toate fisierele de tip format din zona de lucru curenta. Utilizatorul poate deschide un alt fisier format pentru aceeasi tabela de date.

CLOSE INDEX inchide fisierele index deschise pentru zona de lucru curenta. Modificarile continutului tabelii nu se vor reflecta in fisierele index. Daca se fac modificari, este necesara regenerarea indexilor.

CLOSE PROCEDURE-inchide un fisier de proceduri deschis prin comanda SET PROCEDURE TO <nume fisier>.

V. INDICATORUL DE INREGISTRARI. DOMENIUL INREGISTRARILOR. CAUTAREA DATELOR IN TABELE

Indicatorul de inregistrari

Indicatorul inregistrarii este o variabila specifica, ce indica care este inregistrarea asupra careia actioneaza o anumita comanda in cadrul fiecarei zone de lucru. Acesta este asociat unei tabele la deschiderea ei si este eliminat la inchiderea tabelii.

Comanda **GO/GOTO** determina pozitionarea indicatorului de inregistrari al zonei de lucru curente pe inregistrarea cu numarul specificat.

Are sintaxa

GO [RECORD] <expN1> [IN <expN2> | IN <expC>]

GO TOP | BOTTOM [IN <expN2> | IN <expC>]

sau

GOTO [RECORD] <expN1> [IN <expN2> | IN <expC>]

GOTO TOP | BOTTOM [IN <expN2> | IN <expC>]

Numarul inregistrarii pe care va fi pozitionat indicatorul de inregistrari este dat de valoarea numerica <expN1>.

Ea admite doua clauze importante BOTTOM si TOP care permit pozitionarea indicatorului de inregistrari pe ultima si respectiv prima inregistrare din tabela.

Obs: Daca tabela este indexata atunci prima si respectiv ultima inregistrare pot sa nu coincidă cu cele date de ordinea fizica din tabela. Deci clauzele BOTTOM si TOP tin cont de ordinea logica a inregistrarilor.

IN <expN2> | IN <expC>

Implicit, comenzile GO sau GOTO muta indicatorul de inregistrari in tabela deschisa in zona de lucru curenta. Pentru mutarea indicatorului de inregistrari in alta zona de lucru este inclus cu ajutorul clauzelor de mai sus numarul acestei zone de lucru <expN2> sau alias-ul tabelii <expC>.

Un alt tip de deplasare de-a lungul tabelii a indicatorului de inregistrari este realizat cu ajutorul comenzii SKIP <expresie numerica>. Acesta muta indicatorul de inregistrari peste un numar de inregistrari (<expresie numerica>) calculat relativ la inregistrarea curenta. De asemenea aceasta comanda tine cont de ordinea logica a inregistrarilor.

Funcții de acces la date

RECNO() returneaza numarul inregistrarii curente din tabela deschisa in zona de lucru curenta sau in cea specificata.

Sintaxa functiei este RECNO([<expN> | <expC>]) unde [<expN> | <expC>] specifica, daca apar zona de lucru lalias-ul tabelii pentru care functia va returna pozitia curenta a indicatorului de inregistrari.

RECNO() returneaza o valoare mai mare cu 1 decat numarul inregistrarilor din tabela daca indicatorul de inregistrari este pozitionat dupa ultima inregistrare din fisier si returneaza valoarea 1 daca indicatorul de inregistrari este pozitionat inaintea primei inregistrari din fisier. Dca tabela nu are inregistrari RECNO() returneaza valoarea 1 iar daca tabela nu este deschisa in zona de lucru specificata RECNO() returneaza valoarea 0.

RECCOUNT() returneaza numarul total de inregistrari dintr-o tabela.

BOF() si EOF() testeaza daca indicatorul de inregistrari este pozitionat la inceputul, respectiv sfarsitul tabelii curente. Functiile BOF() si EOF() returneaza o valoare de tip logic, anume .T. daca conditiile de mai sus sunt indeplinite si .F. in caz contrar.

Comanada SKIP

Sintaxa: SKIP[<expN1>][IN <expN2> | <expC>]

Efect: Muta indicatorul de inregistrari inainte sau inapoi in tabela deschisa in zona de lucru curenta sau in cea specificata prin clauza IN ([IN <expN2> | <expC>])

<expN1>-reprezinta numarul de inregistrari cu care se muta indicatorul de inregistrari.

Folosirea lui SKIP fara <expN1> face ca indicatorul de inregistrari sa avanseze la inregistrarea urmatoare. Daca <expN1> este un numar pozitiv atunci mutarea se face spre sfarsitul fisierului iar daca <expN1> este un numar negati mutarea se face spre inceputul fisierului

Exemplu:

```
CLOSE DATABASES
```

```
SELECT 2
```

```
USE agenti
```

```
SELECT 1
```

```
USE clienti
```

SKIP 4 IN 2
? RECNO(2)
GO BOTTOM
SKIP -5
? RECNO()

Domeniul inregistrarilor

Domeniul inregistrarilor reprezinta un mecanism de selectie a unui grup de inregistrari din tabela, inregistrari care vor face obiectul unei prelucrari.

Pentru controlul domeniului inregistrarilor sunt introduse patru clauze:

ALL pentru prelucrarea tuturor inregistrarilor din tabela

NEXT <expresie numerica> pentru cazul in care comanda se va referi la urmatoarele n=<expresie numerica> inregistrari, incepand de la inregistrarea curenta, inclusiv inregistrarea curenta

RECORD <expresie numerica> pentru prelucrarea inregistrarii cu numarul <expresie numerica>

REST pentru prelucrarea tuturor inregistrarilor incepand cu cea curenta si pana la sfarsitul tabelului.

Aceste clauze determina o selectie statica a inregistrarilor, in sensul ca selectarea lor pentru prelucrare nu se face in functie de continutul lor.

O selectie dinamica a inregistrarilor este realizata cu ajutorul clauzelor **FOR**<expresiLogica> (pentru) si **WHILE**<expresieLogica>(cat timp).

Clauza **FOR** este folosita pentru selectarea inregistrarilor care satisfac conditia logica <expresiLogica> (adica valoarea de adevar a conditiei <expresiLogica> este True). Dupa gasirea unei inregistrari care nu indeplineste conditia este continuata testarea celorlalte.

Acelasi efect il are si clauza **WHILE**, cu deosebirea ca daca este intalnita o inregistrare care nu satisface conditia <expresiLogica> cautarea este intrerupta.

Exemplu:

DISPLAY ALL FOR RECNO()>3

DISPLAY ALL WHILE agenti.numa="POPESCU"

LOCATE este o comanda de cautare care foloseste domeniul inregistrarilor.

Sintaxa: **LOCATE FOR** <expresieLogica 1>

[<domeniu de inregistrari >]

[**WHILE** <expresieLogica 2>]

Efect: Cauta secvential in tabela o inregistrare care se potriveste unei expresii date si returneaza numarul acesteia daca nu avem setarea **OFF** pentru comanda **SET TALK**(Atunci se foloseste **RECNO()** pentru a afla nr. inregistrarii). Dupa ce **LOCATE** gaseste o inregistrare care se potriveste se continua cautarea folosind **CONTINUE**.

Semnificatia clauzelor **FOR** si **WHILE** este cea data mai sus.

Mai jos prezentam cateva comenzi specifice tabelelor indexate.

Comanda SEEK are sintaxa: **SEEK** <expr> si cauta intr-o tabela indexata (a se vedea sectiunea despre indexarea tabelului) o inregistrare pentru care cheia de indexare se potriveste cu expresia <expr>.

Comanda FIND are sintaxa : FIND <expC> și caută într-o tabelă indexată o înregistrare pentru care cheia de indexare se potrivește cu expresia de tip șir de caractere <expC>.

Comanda SET FILTER determină afișarea doar a înregistrărilor care îndeplinesc condiția specificată în filtrul de selecție. Sintaxa comenzii este SET FILTER TO [<expresieLogica>]

Unde <expresieLogica> este o expresie logică după care se face căutarea.

VI. INCARCAREA SI MODIFICAREA DATELOR DIN TABELE

Încărcarea datelor în tabelă se poate face imediat după crearea structurii tabelului. Ulterior este posibilă adăugarea de noi date respectiv modificarea celor existente prin folosirea comenzilor APPEND, BROWSE EDIT, CHANGE și sau prin folosirea submeniului BROWSE al ferestrei VIEW.

Adăugarea de noi date se poate face cu ajutorul comenzilor APPEND, APPEND FROM și APPEND FROM ARRAY a căror sintaxă este prezentată mai jos.

Comanda APPEND

Sintaxa:

APPEND [BLANK][<în zona de lucru>|<alias>]

Efect: Permite adăugarea de înregistrări la sfârșitul tabelului curent sau în zona de lucru specificată prin [<în zona de lucru>|<alias>].

În cazul în care nu este deschisă nici o tabelă în zona de lucru specificată comanda APPEND determină deschiderea ferestrei **Directory** cu ajutorul căreia se alege o tabelă pentru adăugare de noi înregistrări.

APPEND deschide o fereastră de editare unde pot fi introduse noi înregistrări. Fișierele index deschise în timpul execuției comenzii APPEND sunt actualizate pe măsura ce înregistrările sunt adăugate.

APPEND BLANK adaugă o înregistrare albă la sfârșitul tabelului.

Obs. Pentru adăugarea de înregistrări este preferată comanda mai rapidă SQL, INSERT.

Comanda APPEND FROM

Sintaxa:

APPEND FROM <fișier> | ?
[FIELDS <lista de câmpuri>
| FIELDS LIKE <condiție>
| FIELDS EXCEPT <condiție>]
[FOR <expresieLogica>]
[[TYPE]
[DELIMITED [WITH TAB
| WITH <delimitator>
| WITH BLANK] | DIF | FW2 | MOD

| PDOX | RPD | SDF | SYLK | WK1
| WK3 | WKS | WR1 | WRK | XLS]]

Efect : adauga la sfarsitul tabelii o inregistrare preluata dintr-un fisier.

Fisierul <fisier>, de unde se incarca datele, este presupus a fi o tabela FoxPro cu extensia .DBF. Daca fisierul este o tabela FoxPro care nu are extensia .DBF trebuie sa ii fie specificata extensia. Daca fisierul nu este tabela FoxPro, trebuie sa ii specificati tipul. Daca nu este inclusa extensia fisierului este presupusa implicit extensia .DBF.

Daca incarcarea datelor se face dintr-o tabela FoxPro, aceasta poate fi deschisa in alta zona de lucru. De asemenea datele pot fi incarcate dintr-o tabela care nu este deschisa dar este disponibila pe disk.

Daca este inclusa clauza ?, in loc de numele tabelii, atunci este deschisa o fereasta de dialog cu ajutorul careia poate fi selectata tabela(fisierul) din care vor fi preluate datele.

FIELDS <lista de campuri>

APPEND FROM suport optiune <lista de campuri>. Sunt incarcate doar datele specificate in lista de campuri.

FIELDS LIKE <conditie> | FIELDS EXCEPT <conditie>

Se pot prelua date selectiv din anumite campuri incluzind clauzele LIKE sau EXCEPT sau pe amandoua. Daca includem LIKE <conditie>, FoxPro incarca date din campuri care satisfac conditia <conditie>. Daca este inclusa clauza EXCEPT <conditie>, FoxPro incarca date din toate campurile cu exceptia celor care satisfac conditia <conditie>..

<conditie> suporta caractere speciale. De exemplu, pentru a incarca date din toate campurile care incep cu literele A si P, scriem:

APPEND FROM FIELDS LIKE A*,P*

Clauzele LIKE si EXCEPT pot fi combinate:

APPEND FROM FIELDS LIKE A*,P* EXCEPT PARTNO*

FOR <expresieLogica>

Daca este incusa clauza FOR sunt incarcate doar inregistrarile din fisierul sursa pentru care <expresieLogica> are valoarea de adevar adevarat (.T.).

TYPE

Defineste extensia fisierului sursa daca nu este FoxPro. Incarcarea se poate face dintr-o mare varietate de tipuri de fisier, inclusiv fisiere text in care se poate specifica delimitatorul de camp (DELIMITED ASCII text fisiers). Tipurile fisierelor pot fi urmatoarele:

DIF - Data Interchange Format (utilizat de VisiCalc) (Datele din aceste fisiere sunt preluate astfel; vectorii (coloanele) devin camouri iar tuplurile (liniile) devin inregistrari. Au extensia .DIF.

FW2 - sunt create cu Framework II si au extensia .FW2.

MOD- sunt create cu Microsoft Multiplan versiunea 4.01 si au extensia .MOD.

PDOX sunt fisiere de baze de date din Paradox versiunea 3.5 si 4.0. Au extensia .DB.

RPD - sunt create cu RapidFisier version 1.2. si au extensia . RPD.

SDF (System Data Format) - sunt fisiere text ASCII in care inregistrarile au lungime fixa si se sfarsesc cu (CR)(carriage return) si (LF)(line feed). Campurile nu sunt delimitate. Extensia este presupusa a fi .TXT.

SYLK (Symbolic Link interchange format (folosit in Microsoft Multiplan)) in care coloanele devin campuri si liniile inregistrari. Fisierele SYLK nu au extensie implicita.

WK1-este creat cu Lotus 1-2-3 vers. 2.x

WK3 -este creat cu Lotus 1-2-3 vers. 3.x (Fiecare coloana din foaia de calcul devine camp si fiecare linie devine inregistrare in tabela).

WKS -este creat cu Lotus 1-2-3 vers. 1-A.

WR1-este creat cu Lotus Symphony vers. 1.1 sau 1.2.

WRK-este creat cu Lotus Symphony vers 1.0.

XLS -foie de calcul creata cu Microsoft Excel.(Fiecare coloana din foaia de calcul devine camp si fiecare linie devine inregistrare in tabela).Are extensia XLS.

DELIMITED [WITH TAB

| WITH <delimiter>

| WITH BLANK]

Un fisier delimitat este un fisier text ASCII in care fiecare inregistrare se sfarseste cu (CR) sau (LF). Continutul campurilor se presupune ca sunt separate implicit prin virgule si valorile campurilor ce contin caractere sunt delimitate suplimentar prin ghilimele. De exemplu:

"Smith", 9999999, "TELEPHONE"

Optiunea DELIMITED WITH TAB este folosita atunci cand campurile sunt separate prin tab in loc de virgule. Optiunea DELIMITED WITH <delimiter> este folosita atunci cand campurile sunt separate prin <delimiter>. Optiunea DELIMITED WITH BLANK este folosita atunci cand campurile sunt separate prin spatii. Extensia fisierelor delimitate este presupusa a fi .TXT.

Datele calendaristice pot fi importate daca sunt in formatul corespunzator: 'mm/dd/yy'. Datele in alte formate pot fi0 importate daca formatul lor se potriveste cu cel specificat de comanda SET DATE.

Exemplu

In acest exemplu, este deschisa tabela CLIENTI, este copiată structura sa in tabela numita BACKUP si este deschisa BACKUP. FoxPro incarca toate inregistrarile din

tabela CLIENTI pentru care judetul este GORJ. Aceste inregistrari sunt apoi copiate intr-un fisier delimitat numit TEMP.TXT.

```
CLOSE DATABASES
USE CLIENTI
COPY STRUCTURE TO backup
USE backup
APPEND FROM client FOR judet = 'GORJ'
COPY TO temp TYPE DELIMITED
MODIFY FISIER temp.txt
USE
DELETE FISIER backup.dbf
DELETE FISIER temp.txt
```

Comanda APPEND GENERAL

Sintaxa:

```
APPEND GENERAL <camp general > FROM <fisier>DATE CEXp[LINK][CLASS
<ole class>]
```

Efect: Importa obiecte OLE intr-un camp de tip GENERAL.

Daca exista deja un obiect OLE in campul general, el este inlocuit cu obiectul OLE din fisier.

<general camp> specifica numele campului general in care este plasat obiectul (continului fisierului <fisier>). Se poate specifica si un camp general intr-o tabela care nu este deschisa in zona de lucru curenta incluzind aliasul tablei in numele campului.

<fisier> specifica fisierul care contine obiectul OLE. (Va fi inclusa si extensia sa); poate contine si calea fisierului.

LINK are drept efect crearea unei legaturi intre obiectul OLE si fisierul care il contine. Obiectul OLE apare in campul general dar definitia obiectului ramane in fisier. Daca aceasta clauza este omisa atunci obiectul OLE este complet incorporat in campul general.

CLASS <clasa ole>- specifica in mod explicit o clasa OLE pentru un obiect OLE. Pentru a determina clasa <clasa ole> se lanseaza Regedit si se executa dublu click pe obiectul importat. Numele clasei este afisat dedesubtul referirii Identifier.

Exemplu de mai jos importa un grafic Excel intr-un camp general numit GRAFIC.

```
APPEND GENERAL GRAFIC FROM ;
"C:\EXCEL\CHART1.XLC" CLASS EXCELCHART
```

Comanda APPEND FROM ARRAY

Sintaxa:

```
APPEND FROM ARRAY <array> [FOR <expresieLogica>]
[FIELDS <camp list> | FIELDS LIKE <sablon>
| FIELDS EXCEPT <sablon >]
```

Efect: Adauga inregistrari in tabela curenta preluate dintr-un array (tablou).
Campurile MEMO si GENERAL sunt ignorate de comanda APPEND FROM ARRAY.
Daca o tabela este deschisa pentru utilizare partajata, APPEND FROM ARRAY blocheaza header-ul tabelii in timpul adaugarii de inregistrari.

<array> reprezinta numele tabloului din care sunt incarcate noile inregistrari. Noile inregistrari sunt adaugate in tabela pana cand sunt folosite toate linile din tablou.

In cazul tablourilor unu-dimensionale este adaugata o singura inregistrare in tabela(Primul element din vector reprezinta continutul primului camp din tabela, al doilea este continutul celui de al doilea camp etc.) Daca vectorul are mai multe elemente decat campurile tabelii atunci cele in plus vor fi ignorate. Daca vectorul are mai putine atunci restul campurilor sunt initializate cu valoarea nula.

In cazul tablourilor bidimensionale sunt create un numar de inregistrari egal cu cel al liniilor tabloului. Se procedeaza la fel ca in cazul tablourilor unu-dimensionale.

FOR <expresieLogica> permite incarcarea numai a inregistrarilor din tablou care satisfac <expresieLogica>. <expresieLogica> va contine si numele campului in care vor fi incarcate date.

Daca un element al tabloului nu satisface conditia corespunzatoare campului vizat atunci linia tabloului care contine elementul nu va mai fi incarcata in tabela.

FIELDS <lista de campuri> este inclusa, pentru ca numai campurile specificate in lista sa fie incarcate din array. Primul camp din lista este inlocuit cu continutul primului element al tabloului al doilea cu continutul celui de al doilea element etc.

FIELDS LIKE <sablon> | FIELDS EXCEPT < sablon > Clauzele LIKE sau EXCEPT actioneaza ca si in cazul comenzii APPEND FROM fisier.

Exemplu:

```
APPEND FROM ARRAY tab1 FIELDS LIKE A*,P*
APPEND FROM ARRAY tab1 FIELDS LIKE A*,P* EXCEPT PARTNO*
```

Daca tipul datelor nu se potriveste, incarcarea se poate totusi face daca elementul din array este compatibil cu cel din campul corespunzator. In caz de incompatibilitate completarea se face cu date vide.

Comanda CHANGE

```
CHANGE [FIELDS <camp list>][<domeniu de inregistrari>]
[FOR < expresie logica 1>][WHILE < expresie logica 2>]
[FONT <expC1> [, <expN1>]]
[STYLE <expC2>][FREEZE <camp>]
[KEY <expr1> [, <expr2>]][LAST]
[LEDIT] [REDIT][LPARTITION][NOAPPEND]
[NOCLEAR][NODELETE][NOEDIT | NOMODIFY]
[NOLINK][NOMENU][NOOPTIMIZE]
[NORMAL][NOWAIT][PARTITION <expN2>]
[PREFERENCE <expC3>][REST][SAVE][TIMEOUT <expN3>]
[TITLE <expC4>][VALID [:F] <expL3>][ERROR <expC5>]]
```

```
[WHEN <expL4>][WIDTH <expN4>][[WINDOW <window name1>]
[IN [WINDOW] <window name2> | IN SCREEN]]
[COLOR SCHEME <expN5> | COLOR <color pair list>]
```

Efect: Afiseaza tabela in fereastra **Change**, pentru editare. Salvarea modificarilor si parasirea ferestrei de editare se realizeaza tastand [Ctrl]+[W] sau [Ctrl]+[End]. Apasarea tastelor [Ctrl]+[Q] sau [Esc] are drept efect parasirea ferestrei de editare fara salvarea modificarilor. Functia lui CHANGE este identica cu cea a lui EDIT si BROWSE.

In cazul unui program, se va folosi comanda DEACTIVATE WINDOW pentru salvarea modificarilor si inchiderea ferestrei **Change**.

Obs: 1 Comada **SET SKIP** permite realizarea de relatii de tip unu la mai multi intre tabele astfel incat, pentru fiecare inregistrare din tabela parinte se stabilesc mai multe corespondente cu inregistrari din tabela copil. In acest caz comanda CHANGE permite vizualizare inregistrarilor atat din tabela parinte cat si din cea copil.

CHANGE are numeroase optiuni dintre care cele mai importante sunt descrise mai jos,

FIELDS <lista de campuri> Campurile ce vor fi modificate apar in ordinea din <lista de campuri>. Daca clauza FIELDS nu apare atunci campurile apar in ordinea din tabela.

Lista de campuri poate contine orice combinatie de campuri sau expresii rezultate din calcule cu campuri din tabele deschise in diferite zone de lucru. Sintaxa pentru lista de campuri este:

```
<camp1> [:R]
[:V = <expr1> [:F] [:E = <expC1>]]
[:B = <expr2>, <expr3> [:F]]
[:H = <expC3>]
[:W = <expL1>]
```

```
[, <camp2> [:R] ...]
```

unde [:R] este optiunea pentru **read only** si va fi asignata campului inclus cu ajutorul comenzii CHANGE; campul va fi vazut dar nu editat.

Exp: Fie tabela agenti(cod_ag, nume, prenume, sex, data nasterii, adresa,telefon, comision, newfld)

CLOSE DATABASES

USE agenti

CHANGE FIELD nume:R, prenume:R, adresa

Exemplul de mai sus afiseaza campurile nume prenume dar permite modificarea doar a campului adresa.

[:V = <expr1>] este optiunea pentru validarea (prin verificarea expresiei <expr1>) continutului campului (in caz de nevalidare cursorul va ramane pozitionat in acelasi camp si va fi afisat un mesaj de eroare). <expr1> nu este verificata pentru campurile MEMO. Optiunea [:F] forteaza validarea campului cand cursorul baleiaza campul, [:E = <expC1>]] permite afisarea mesajului <expC1> cand apare o eroare la completarea campului. <expC1> afisata numai daca SET NOTIFY este ON. Avertizarea sonora functioneaza numai daca BELL este SET ON.

:P = <expC2> permite specificarea unei optiuni PICTURE pentru fiecare camp din lista. <expC2> va controla si afisa formatul in care apar datele din fiecare fereastră Change.

Exemplu

```
CLOSE DATABASES
USE agenti
CHANGE FIELDS nume, comision :P = '999,999.99'
```

[B = <exp2>, <exp3> [F]]-permite specificarea unor limite de valori (<exp2>, <exp3>) intre care sa se incadreze continutul unui camp.

[H = <expC3>]-permite specificarea unui alt nume de camp in locul celui implicit, nesemnificativ;

Exp: In tabela agenti de mai sus campul newfld va fi renumit.

```
CLOSE DATABASES
```

```
USE agenti
```

```
CHANGE FIELD nume:R, prenume:R, newfld:H="zona_act"
```

<domeniu de inregistrari>- specifica domeniului inregistrarilor care vor fi afisate infereastră Change. Domeniul implicit pentru CHANGE este ALL.

FOR < expresie logica 1>, si respectiv WHILE < expresie logica 2> sunt clauze pentru domeniul inregistrarilor si au fost prezentate in paragraful dedicat domeniului inregistrarilor.

FONT <expC1> [, <expN1>]- specifica fontul, <expC1> este numele fontului, iar expresia numerica <expN1> is dimensiunea fontului. Exemplu:

```
FONT 'ROMAN',16
```

STYLE <expC2>-specifica stilul fontului (B, I, N, U etc)

Exemplu: CHANGE FIELDS nume FONT 'System', 15 STYLE 'NU'

FREEZE <camp> -permite modificarea la nivelul unui simplu camp si simpla afisare a celorlalte.

KEY <expr1> [,<expr2>] limiteaza inregistrările afisate; <expr1> este valoarea cheii de index sau daca folosim (<expr1>, <expr2>) limitam domeniul inregistrarilor la cele care iau valori in (<expr1>, <expr2>).

Exp: USE agenti

```
SET ORDER TO comision
```

```
CHANGE FIELD zona KEY '430','439'
```

LAST-deschide fereastră Change in aceeași configuratie salvata ultima oara in fisierul FOXUSER(fisier in care sunt memorate lista de campuri, marimea fiecarui camp, marimea ferestrei CHANGE etc) in cazul in care comanda SET RESOURCE este ON.

LEDIT|REDIT specifica faptul ca partitia stanga sau dreapta este in modul BROWSE

```
USE agenti
```

```
CHANGE PARTITION 30 LEDIT
```

LPARTITION-plaseaza cursorul in primul camp al partitiei din stanga (respectiv din dreapta daca LPARTITION lipseste)

NOAPPEND - nu permite adaugarea de inregistrari noi in tabela.

NOCLEAR- imaginea ferestrei CHANGE ramane afisata si dupa inchiderea sa.

NODELETE -nu permite ca inregistrarile sa fie marcate pentru stergere logica.(In mod normal inregistrarile pot fi marcate pentru stergere logica in fereastra CHANGE prin tastarea combinatiei [Ctrl]+[T]) sau selectand cu dublu click inregistrarea.

NOEDIT | NOMODIFY - impiedica efectuarea modificarilor in tabela.

NOLINK- nu efectueaza legaturi intre partitii(cand parcurgem o partitie, in mod normal este parcursa si cealalta)

NOMENU –inhiba accesul la linia de selectie a meniurilor Browse, utilizate in mod normal prin [Ctrl]+[B].

NOOPTIMIZE – dezactiveaza tehnologia de optimizare Rushmore.

NORMAL – deschide “normal”, ignorand caracteristicile altor ferestre deschise sau selectate pentru iesire. Cand o fereastra utilizata este selectata pentru iesire, fereastra **Change** alege culorile, marimea, pozitia si optiunile de control (GROW, FLOAT, ZOOM, etc) din acesta fereastra

NOWAIT – continua executia programului dupa ce fereastra **Change** a fost deschisa.

Optiunea este disponibila doar in programe.

PARTITION <expN2>-divide fereastra **Change** in doua ferestre, iar <expN2> specifica coloana din care va fi afisata a doua fereastra.

PREFERENCE <expC3> - salveaza atributele si optiunile ferestrei **Change** pentru utilizarea viitoare. <expC3> este numele ferestrei **Change** pentru care sunt salvate atributele.

REST-este utilizat cu CHANGE FOR pentru a impiedica pozitionarea cursorului la inceputul tabelii.

SAVE –pastreaza fereastra **Change** pe ecran dupa parasirea acesteia.

TITLE <expC4> -specifica titlul ferestrei **Change**.

VALID [:F] <expL2> [ERROR <expC5>] este executata numai daca a fost facuta o schimbare intr-un camp al inregistrarii si apoi se incearca trecerea la alte inregistrari. Clauza VALID nu se executa daca schimbarea e facuta intr-un camp MEMO. Atunci cand clauza Valid returneaza o valoare logica (.T.), se poate muta cursorul pe alta inregistrare. In caz contrar, cursorul ramane in campul curent din inregistrarea curenta si este afisat mesajul "Invalid input". Poate fi afisat propriul mesaj de eroare, <expC5>, atunci cand se foloseste optiunea ERROR. Daca VALID returneaza 0, cursorul ramane in campul curent si mesajul de eroare nu mai este afisat.

WHEN <expL4> - permite sau nu selectarea unei inregistrari. Daca <expL4> ia valoarea (.F.), sau (0) inregistrarea selectata are atributul read-only si nu poate fi modificata. The cursor moves to the next camp.

WIDTH <expN4> -limiteaza numarul de caractere afisate pentru toate campurile.

WINDOW <nume fereastra 1> IN [WINDOW] < nume fereastra 2>- deschide fereastra Change in interiorul unei ferestre definite cu comanda DEFINE WINDOW. O fereastra Change activata in interiorul ferestrei principale nu poate fi mutata in afara ei si nici nu poate fi mai mare decat aceasta.Daca se muta fereastra principala, se muata si fereastra Change.

IN WINDOW SCREEN –permite plasarea in fereastra principala FoxPro a ferestrei **Change**.

COLOR SCHEME <expN5>|COLOR <lista de culori>- stabileste lista de culori. Implicit fereastra Change isi asuma schema coloristica numarul 10.

Comanda BROWSE

Sintaxa:

```
BROWSE [FIELDS <field list>] [FONT <expC1> [, <expN1>]]  
[STYLE <expC2>][FOR <expL1>][FORMAT][FREEZE <field>]  
[KEY <expr1> [, <expr2>]][LAST | NOINIT][LEDIT]  
[REDIT][LOCK <expN2>][LPARTITION] [NOAPPEND]  
[NOCLEAR][NODELETE][NOEDIT | NOMODIFY][NOLGRID] [NORGRID]  
[NOLINK][NOMENU][NOOPTIMIZE][NOREFRESH]  
[NORMAL][NOWAIT][PARTITION <expN3>][PREFERENCE <expC3>]  
[REST][SAVE][TIMEOUT <expN4>][TITLE <expC4>]  
[VALID [:F] <expL2> [ERROR <expC5>]] [WHEN <expL3>]  
[WIDTH <expN5>][[WINDOW <window name1>]  
[IN [WINDOW] <window name2> | IN SCREEN]]  
[COLOR SCHEME <expN6>| COLOR <color pair list>]
```

Efect: Acelasi ca si in cazul comenzii CHANGE.

Toate clauzele care apar in ambele comenzi (CHANGE si BROWSE) au acelasi efect.

Comanda REPLACE

Sintaxa:

REPLACE

```
<camp1> WITH <expr1> [ADDITIVE]  
[, <camp2> WITH <expr2>  
[ADDITIVE]] ...  
[<domeniu al inregistrarilor>]  
[FOR <expL1>]  
[WHILE <expL2>]  
[NOOPTIMIZE]
```

Efect: Inlocuieste datele din campuri <camp1>,<camp2>... cu valorile din expresii, <expr1>, <expr2> Campurile din zonele de lucru neselectate vor fi prefatate cu aliasul lor. De exemplu daca ne referim la tabela agenti.dbf deschisa in zona de lucru 2 iar zona de lucru curenta este una atunci ne vom referi la campul nume al tabelii prin "agenti.nume" .

Obs.1. Nu au loc inlocuiri daca indicatorul de inregistrari curent este la sfarsitul fisierului in zona de lucru curenta si nu este specificat un camp in alta zona de lucru.

2. Deoarece comanda Replace actualizeaza si fisierele index active se recomanda sa nu fie aplicata campurilor cheie ale fisierului index in timp ce asupra lor actioneaza una din clauzele comenzii REPLACE.

<camp1> WITH <expr1>

[, <camp2> WITH <expr2> ...- valoarea din <camp1> este inlocuita cu valoarea expresiei <expr1>, valoarea din <camp2> cu cea a expresiei <expr2> etc.

Daca valoarea expresiei este mai lunga decat lungimea unui camp numeric atunci REPLACE forteaza scrierea astfel: partea zecimala este trunchiata si portiunea ramasa din zecimale este rotunjita; daca tot nu este posibila scrierea continutului campului este memorat in scriere stiintifica; daca nici asa nu este destul loc atunci continutul campului este inlocuit cu asteriscuri (*).

ADDITIVE-influenteaza inlocuirile numai in campurile MEMO, unde adaugarea informatiilor se va face la sfarsitul celor existente. Daca aceasta clauza lipseste campul MEMO este rescris(vechiul continut se pierde).

<domeniu al inregistrarilor> reprezinta domeniul inregistrarilor la care se aplica comanda REPLACE (ALL, NEXT <expN>, RECORD <expN> si REST). Domeniul implicit al inregistrarilor pentru comanda REPLACE este inregistrarea curenta (NEXT 1).

Clauzele FOR <expL1> si WHILE <expL2> au acelasi efect ca si in cazul comenzii CHANGE.

NOOPTIMIZE – dezactiveaza tehnologia de optimizare Rushmore.

Exemplu

CLOSE DATABASES

USE agenti

COPY NEXT 1 TO temp

USE TEMP

SET TALK OFF

SCATTER MEMVAR BLANK

DEFINE WINDOW menter FROM 7,10 to 17,70 PANEL

ACTIVATE WINDOW menter

@ 1,3 SAY 'cod_agent: ' GET temp.cod_agent

@ 3,3 SAY 'Adresa: ' GET temp.adresa

@ 5,3 SAY 'nume: ' GET temp.nume

@ 7,3 SAY 'prenume: ' GET temp.prenume

READ

DEACTIVATE WINDOW menter

RELEASE WINDOW menter

IF UPDATED()

APPEND BLANK

REPLACE cod_agent WITH temp.cod_agent

REPLACE adresa WITH temp.adresa

REPLACE nume WITH temp.nume

REPLACE prenume WITH temp.prenume

BROWSE

ENDIF

CLOSE DATABASES

ERASE temp.dbf

Comanda DELETE

Sintaxa:

DELETE [<domeniu de inregistrari>]

[FOR <expL1>]
[WHILE <expL2>]
[NOOPTIMIZE]

Efect: Marcheaza inregistrari pentru stergere in tabela curenta.

Inregistrarile marcate pentru stergere nu sunt sterse fizic din tabela. Acest lucru se intampla daca comanda DELETE este urmata de comanda PACK. Inregistrarile marcate pentru stergere cu comanda DELETE pot fi recuperate (nemarcate) cu comanda RECALL.

Inregistrarile pot fi marcate pentru stergere si din ferestrele Browse, Change sau Edit asa cum am aratat mai sus.

Semnificatia clauzelor comenzii DELETE sunt cele prezentate la CHANGE.

Comanda RECALL are aceeaasi sintaxa cu DELETE.

RECALL [<domeniu de inregistrari>]

[FOR <expL1>]
[WHILE <expL2>]
[NOOPTIMIZE]

Marcajul pentru stergere poate fi la randul sau sters si din ferestrele Browse, Change sau Edit, apasand tastele [Ctrl]+[T] sau selectand cu dubluclick inregistrarea.

Funcția DELETED([<expC> | <expN>]) returneaza valoarea de adevar adevarat (.T.) daca inregistrarea curenta este marcata pentru stergere si fals in caz contrar.

<expC> | <expN> - specifica daca in zona de lucru <expN> sau tabela <expC> inregistrarea curenta este marcata sau nu pentru stergere. Daca tabela <expC> respectiv nici o tabela in zona de lucru <expN> nu este deschisa atunci DELETED() returneaza fals. Daca <expC> | <expN> lipsesc atunci este returnata starea inregistrarii curente din zona curenta de lucru.

Exemplu:

CLOSE DATABASES
SET TALK OFF
CLEAR
USE agenti
GOTO 5
DELETE REST
GOTO 6
?DELETED()
RECALL ALL

Comanda PACK

Sintaxa:

PACK [MEMO] [DBF]

Efect: Sterge fizic inregistrarile marcate pentru stergere in baza de date curenta. De asemenea PACK reduce dimensiunea unui fisier memo fisier asociat unei tabele.

La executarea comenzii PACK, toate inregistrarile care nu sunt marcate pentru stergere sunt copiate intr-un fisier temporar. Cand comanda PACK este completa, FoxPro sterge

tabela originala de pe disk si renumeste tabela temporara cu numele tabelii originale. PACK poate fi intrerupta apasand tasta [Esc], tabela temporara este stearsa si tabela originala ramane neschimbata. De asemenea tabela originala este recuperata daca se depaseste spatiul de pe disc in timpul executiei comenzii PACK.

Daca tabela curenta are fisiere index acestea vor fi refacute.

Clauza MEMO permite stergerea fisierului memo atasat tabelii, fara stergerea fisierului tabelii (cele doua tabele au acelasi nume).

DBF permite stergerea inregistrarilor marcate pentru stergere in tabela dar nu permite stergerea fisierului memo atasat tabelii.

Comanda ZAP

Sintaxa: ZAP

Efect: Sterge (fizic) toate inregistrările din tabela curenta, pastrand doar structura tabelii. Folosirea comenzii ZAP este echivalenta cu cea a comenzii DELETE ALL urmata de PACK, dar ZAP este mai rapida.

Daca SET SAFETY este ON, FoxPro intreaba daca se doreste inlaturarea inregistrarilor din tabela curenta.

Obs. Inregistrările sterse cu comanda ZAP nu pot fi recuperate.

VII. CREAREA VIDEO FORMATELOR

Accesul la datele unei tabele pentru adaugare, modificare, selectare sau stergere poate fi realizata si prin intermediul video-formatelor. Crearea video formatelor se poate realiza cu ajutorul comenzii CREATE FORM care are drept efect deschiderea ferestrei **Form designer** sau folosind asistentul **Form Wizard**. Aceasta fereastra permite crearea si modificarea video-formatelor sau a multimilor de forme (form sets=una sau mai multe forme ce pot fi manipulate intr-un mod unitar).

In cele ce urmeaza vom expune doar modalitatea de creare a video formatelor prin utilizarea asistentilor **Form Wizard** si **One-To-Many Form Wizard**.

Acestea creeaza videoformate pentru datele dintr-o singura tabela (**Form Wizard**) respectiv doua tabele relationate (**One-To-Many Form Wizard**). Dimensiunea videoformatului este determinata de valorile setate pentru **Maximum design area** in cadrul suboptiunii **Forms** a optiunii **Options** a meniului **Tools** a ferestrei principale Visual FoxPro (**Tools→Options→Forms**).

Pentru a fi lansati in executie asistentii **Form Wizard** sau **One-To-Many Form Wizard** este selectata comanda **Forms** din suboptiunea **Wizards** a meniului **Tools** (**Tools→Wizards→Forms**) iar din cutia de dialog **Wizard Selection** este aleasa optiunea **Form Wizard**, respectiv **One-To-Many Form Wizard**.

Pentru realizarea unui video format cu ajutorul asistentului **Form Wizard** sunt parcursi urmatoarii pasi:

Pasul 1 – Se selecteaza tabela pentru care se creeaza video formatul si apoi sunt selectate campurile care vor apare in video format. **Select Fields**.

Pasul 2 – Se alege stilul video formatului. La acest pas se pot alege diferite stiluri afisate in cutia Style, iar asistentul afiseaza o imagine care sa exemplifice stilul respectiv in coltul din stanga sus al ferestrei. In caseta **Button type** vor fi alese tipurile de butoane de navigare care vor apare in videoformat. Butoanele de navigare care sunt create de asistent sunt urmatoarele

Buton Descriere

Top Muta indicatorul de inregistrari pe prima inregistrare.

Prev Muta indicatorul de inregistrari inapoi cu o inregistrare (pe inregistrarea precedenta).

Next Muta indicatorul de inregistrari inainte cu o inregistrarea (pe inregistrarea urmatoare).

Bottom Muta indicatorul de inregistrari pe ultima inregistrare.

Find Afiseaza cutia de dialog **Search** (Cauta).

Print Tipareste un raport.

Add Adauga o noua inregistrare la sfarsitul tabeli.

Edit Permite utilizatorului schimbarea valorilor inregistrarii curente.

Delete Sterge inregistrarea curenta.

Exit Inchide videoformatul.

Observatie. La inchiderea videoformatului acesta va fi salvat intr-un fisier cu extensia .scx . Videoformatul poate fi modificat cu ajutorul ferestrei **Form Designer**, care poate fi deschisa utilizand comanda MODIFY FORM [<numef>|?].

La un videoformat salvat putem adauga campuri care sa foloseasca acelasi stil folosind comanda **Quick Form** a meniului **Form** care este disponibil atunci cand este deschisa fereastra **Form Designer**. La selectarea comenzii **Quick Form** este deschisa fereastra **Form Builder**, care permite adugarea de noi campuri care sa pastreze sau nu acelasi stil cu cel folosit pentru realizarea videoformatului initial. Videoformatul poate fi salvat sub aceasta noua forma. Odata realizat un videoformat acesta poate fi lansat in executie cu ajutorul comenzii DO FORM <[cale]numeforma> sau selectand comanda **Run Form** a meniului **Form**.

Pasul 3 – La acest pas sunt sortate inregistrarile. Sunt selectate campurile in ordinea in care se doreste sortarea inregistrarilor in interiorul fiecarui grup. De asemenea se poate alege o eticheta index.

Pasul 4 – Este pasul final. Aici se poate preciza daca forma va fi executata dupa ce va fi salvata sau nu, daca va fi salvata si modificata imediat cu ajutorul ferestrei **Form Designer**. De asemenea daca a fost ales un numar mare de campuri la Pasul 1, pentru a fi siguri ca acestea vor avea loc in videoformat se poate selecta optiunea **Add pages for fields that do not fit**, (pentru a aduga pagini suplimentare pentru campurile care nu vor incapa pe pagina initiala a videoformatului). Selectarea butonului **Preview** va permite vizualizarea formei videoformatului inainte de incheierea operatiunii de creare a lui prin intermediul asistentului **Form Wizard**.

Asistentul **One-To-Many Form Wizard** creaza un videoformat pentru date ce sunt stocate in doua tabele relationate.

Pentru realizarea unui video format cu ajutorul asistentului **One-To-Many Form Wizard** sunt parcursi urmatoorii pasi:

Pasul 1 – Selectarea campurilor tabelii parinte pe care dorim sa le includem in videoformat.

Pasul 2 – Selectarea campurilor tabelii copil ce se afla in relatie cu tabela parinte selectata la Pasul 1.

Pasul 3 – Stabilirea(selectarea) campurilor ce definesc relatia intre cele doua tabele.

Pasul 4 – Alegerea stilului videoformatului (vezi Pasul 2 de la **Form Wizard**).

Pasul 5 – Selectarea campurilor pentru stabilirea ordinii de sortare a inregistrarilor in tabela parinte (Se poate alege si o eticheta index).

Pasul 6 – Se termina operatia de creare a videoformei. Vezi pasul similar de la **Form Wizard**).

VIII. SELECTAREA SI AFISAREA DATELOR DIN TABELE

Selectarea dupa anumite criterii si apoi afisarea datelor din tabele reprezinta unul din scopurile celorlalte operatii executate asupra tabelilor. Comenzile LIST si DISPLAY prezentate mai jos realizeaza aceasta operatie de selectare si afisare a datelor.

Comanda LIST

Sintaxa:

```
LIST [FIELDS <lista de campuri> | FIELDS LIKE <sablon>
| FIELDS EXCEPT < sablon >]
[<domeniu de inregistrari>][FOR <expL1>][WHILE<expL1>]
[OFF]
[NOCONSOLE]
[NOOPTIMIZE]
[TO PRINTER [PROMPT]
| TO FILE <nume fisier>]
```

Efect: Afiseaza continutul inregistrarilor tabelii care satisfac conditiile impuse

Comanda LIST este identica cu DISPLAY cu exceptia urmatoarelor diferente:

- a) In cazul comenzii LIST domeniul inregistrarilor este implicit ALL.
- b) List nu face pauza in momentul in care ecranul sau fereastra se completeaza.
- c) Nu se afiseaza inregistrarile marcate pentru stergere atunci cand comanda SET DELETED is ON.

FIELDS < lista de campuri > specifica campurile care vor fi afisate. Daca nu este inclusa < lista de campuri > atunci vor fi afisate toate campurile tabelii. Continutul campurilor memo nu este afisat decat daca numele campului este inclus explicit in lista. Lungimea de afisare a campurilor memo este determinata de SET MEMOWIDTH.

FIELDS LIKE <sablon> | FIELDS EXCEPT < sablon > permit realizarea unor filtre de selectie la nivelul campurilor pentru al caror nume corespunde unui model (LIKE) sau nu (EXCEPT).

Exemplu:

```
LIST FIELDS LIKE A*,P*
DISPLAY FIELDS LIKE A*,P* EXCEPT PA*
```

<domeniu de inregistrari>(se cunoaste vezi sectiunea dedicata lui).

FOR <expL1>| WHILE <expL2> au efectul prezentat in sectiune domeniul inregistrarilor.

OFF inhiba afisarea numarului inregistrarii.

TO PRINTER [PROMPT] - directioneaza iesirea la imprimanta. Clauza PROMPT activeaza inainte de imprimare fereastra de dialog pentru optiunile imprimantei.

TO FILE <fisier> directioneaza iesirea intr-un fisier cu numele <fisier>.

NOCONSOLE dezactiveaza iesirea spre ecran sau spre fereastra activa.

NOOPTIMIZE dezactiveaza tehnica Rushmore.

Comanda LIST FILES

Sintaxa: LIST FILES [ON <disc\director>] [LIKE <sablon>]
[TO PRINTER [PROMPT]] TO FILE <nume fisier>]

Efect: afiseaza informatii despre tabelele din locatia specificata prin [ON <disc\director>] sau din directorul curent, daca aceasta lipseste.

Clauza LIKE <sablon> afiseaza numai fisierele specificate de utilizator prin identificatorii * si ?.

[TO PRINTER [PROMPT]] TO FILE <file>] directioneaza iesirea la imprimanta sau intr-un fisier.

Comanda LIST STRUCTURE

Sintaxa: LIST STRUCTURE[NOCONSOLE][TO PRINTER [PROMPT]] TO FILE <file>]

Efect: afiseaza structura tabeli de date curente la imprimanta sau intr-un fisier.

Comanda DISPLAY

Sintaxa:

DISPLAY

[[FIELDS] <field list> | FIELDS LIKE <skel>
| FIELDS EXCEPT <skel>]
[<scope>]
[FOR <expL1>]
[WHILE <expL2>]
[OFF]
[TO PRINTER [PROMPT]]
| TO FILE <file>]
[NOCONSOLE]
[NOOPTIMIZE]

Efect:

Afiseaza continutul inregistrarilor tabeli curente.

Clauzele comenzii DISPLAY au acelasi efect ca si cele ale comenzii LIST.
Facem observatia ca domeniul inregistrarii implicit pentru comanda DISPLAY este inregistrarea curenta (NEXT 1).

DISPLAY poate de asemenea sa afiseze rezultatelor expresiilor de tip caracter, a variabilelor de tip tablou sau a variabilelor de memorie.

Comanda display admite si versiunile DISPLAY FILES si DISPLAY STRUCTURE care au aceleasi clauze si acelasi efect ca si in cazul comenzilor corespunzatoare LIST.

IX. ORDONAREA DATELOR

Cautarea unei inregistrari intr-o tabela este mult usurata prin ordonarea acesteia dupa un criteriu adecvat. Acest lucru poate fi constat si in practica, de exemplu, cautarea unei carti intr-o biblioteca cu cateva mii de exemplare ar necesita o munca imensa daca aceste carti nu ar fi ordonate dupa unul sau mai multe criterii. De asemenea gasirea numarului de telefon al unei persoane folosind cartea de telefon este un bun exemplu ca ordonarea datelor dupa un anumit criteriu scurteaza timpul de cautare.

FoxPro dispune de doua tipuri de ordonare a datelor

Ordonarea fizica – care presupune schimbarea intre ele a inregistrarii, conform unui algoritm pana cand aceste apar in ordinea dorita. Se obtine o noua tabela care contine aceleasi inregistrari ca si cea de la inceput dar in care ordinea este cea dorita.

Ordonarea logica – presupune crearea unui fisier special, numit fisier index, care contine informatii cu privire la ordinea inregistrarii tabelii. Tabela initiala ramane neschimbata, dar este creat un fisier nou, fisierul index, care nu contine inregistrari tabelii, dar contine ordinea acestora (conform criteriului stabilit).

Ordonarea unei tabele se face dupa un anumit criteriu, caruia de obicei ii este asociata o expresie. Compararea a doua inregistrari din tabela se reduce de fapt la compararea valorilor acestei expresii.

Aceasta expresie se numeste **criteriu de ordonare**. O tabela este ordonata crescator (ascendent) sau descrescator (descendent) daca fiecare inregistrare din aceasta (adica fiecare valoare a criteriului de ordonare corespunzator inregistrarii) este mai mare, respectiv mai mica decat cea calculata pentru inregistrarea anterioara.

Sortarea

Ordonarea fizica (sau **sortarea**) a unei tabele se realizeaza cu ajutorul comenzii SORT

Comanda SORT

Sintaxa:

```
SORT TO <fisier> ON <camp1> [/A | /D] [/C]
    [, <camp2> [/A | /D] [/C] ...]
    [ASCENDING | DESCENDING]
    [<domeniu de inregistrari>]
    [FOR <expL1>]
    [WHILE <expL2>]
```

```
[CAMPS <lista de campuri>  
| CAMPS LIKE <sablon>  
| CAMPS EXCEPT <sablon>]  
[NOOPTIMIZE]
```

Efect:

Sorteaza inregistrarile din tabela curenta si creaza o noua tabela in care inregistrarile sunt in ordinea dorita. Dupa sortare, in zona de lucru curenta ramane deschisa tabela initiala.

Ordinea inregistrarilor din tabela ordonata va fi determinata de unul sau mai multe campuri din tabela curenta.

Obs. Sa se verifice daca este suficient spatiu pe disc pentru a efectua operatia de sortare. Se stie ca pentru a efectua sortarea este necesar un spatiu de trei ori mai mare decat cel ocupat de tabela care va fi sortata

Campurile de tip sir de caractere care contin numere si spatii nu vor fi sortate in ordinea la care ne asteptam. De exemplu daca doua inregistrari ale tabelii contin intr-un camp de tip sir de caractere valorile 156 si 16 si facem o ordonare crescatoare atunci 156 apare inaintea lui 16. Aceasta deoarece FoxPro citește caracterele(caracter cu caracter) de la stanga la dreapta.

<fisier> reprezinta numele noii tabele ce va fi creata si va contine tabela initiala sortata. (Fisierului i se asociaza implicit extensia .dbf)

ON <camp1> - In aceasta clauza <camp1> reprezinta un camp al tabelii curente(ce va fi sortata) care va determina ordinea pentru inregistrari in tabela sortata. Implicit sortarea se face in ordine crescatoare. Nu se poate face sortare dupa campuri memo sau campuri generale. De asemenea se pot include si alte campuri (<camp2>, <camp3>) in criteriul de sortare. Primul camp <camp1> reprezinta principalul camp dupa care se face sortarea, al doilea camp este campul secundar pentru sortare etc. Se efectueaza o prima sortare dupa <camp1> si daca pentru acesta exista valori egale (repetate) atunci se face o sortare dupa <camp2>, daca si pentru acesta exista inregistrari cu valori egale atunci se face sortarea dupa <camp3> etc.

Exemplu: CLOSE DATABASES
 CLEAR
 USE agenti
 LIST CAMPS nume, prenume NEXT 10
 SORT TO temp ON nume
 USE temp
 LIST CAMPS nume, prenume NEXT 10

[/A | /D] [/C] – Optiunea /A optiunea specifica ordinea ascendenta a campului iar optiunea /D ordinea descendenta. (/A si /D pot fi incluse pentru orice tip de camp).

Deoarece, implicit sortarea campurilor este “case sensitive”(se face distinctie intre literele mari si mici) este inclusa optiunea /C dupa numele campului de tip sir de caractere pentru a nu se mai face deosebirea intre literele mari si mici.(obs. Se poate folosi sintaxa /AC sau /DC pentru a folosi optiunea /C cu /A sau /D.

ASCENDING | DESCENDING – Aceste clauze permit specificarea ordinii de sortare pentru toate campurile care nu sunt urmate de optiunile /A sau /D. (Ordinea va fi

crescatoare, daca folosim clauza ASCENDING (sau daca nici una dintre clauze nu este inclusa) si descrescatoare, daca este folosita clauza DESCENDING).

<domeniu de inregistrari> -Clauzele specifice domeniului inregistrarii sunt: ALL, NEXT <expN>, RECORD <expN>, and REST. Cu ajutorul lui se poate specifica un domeniu al inregistrarii ce va fi sortat. (Numai inregistrariile din domeniul inregistrarii vor fi sortate). Domeniul implicit este ALL.

Actiunea clauzelor FOR <expL1> si WHILE <expL2> a fost explicata in paragraful dedicat domeniului inregistrarii.

CAMPS <lista de campuri>-noua tabela creata de comanda SORT poate contine o submultime de campuri ale tabelii originale. Daca clauza CAMPS nu este inclusa, in noua tabela vor apare toate campurile vechii tabelii.

CAMPS LIKE < sablon > | CAMPS EXCEPT < sablon > -se poat specifica campurile care vor apare in noua tabela cu ajutorul clauzelor LIKE sau EXCEPT sau cu amandoua.

Exp:

SORT TO agenti_sort ON nume CAMPS LIKE A*,P*

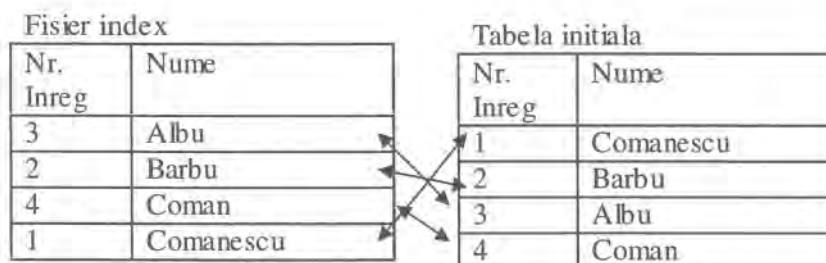
SORT TO agenti_sort ON nume CAMPS LIKE A*,P* EXCEPT POP*

NOOPTIMIZE -dezactivazeaza tehnologia Rushmore

Indexarea

Sortarea nu este cel mai indicat mod de ordonare a inregistrarii unei tabelii deoarece dureaza mult si ocupa, asa cum am precizat mai sus, mult spatiu pe disc. De asemenea in cazul unei actualizarii a tabelii este necesara o noua sortare.

Atunci s-a recurs la ordonarea logica a tabelii adica indexarea acestora. Principiul pe baza caruia functioneaza indexarea este asemanator cu indexul unei carti (contine informatii despre localizarea inregistrarii dintr-o tabela). De exemplu in cazul indexarii tabelii agenti dupa campul nume obtinem un fisier index care memoreaza valorile dupa care s-a facut ordonarea (crescator) si numarul inregistrarii care corespunde respectivei valori.



Obs. 1. Intr-un fisier index pot fi incorporate mai multe criterii de ordonare. Fiecare dintre aceste criterii este identificat printr-o **eticheta index**.

2. Modul de lucru cu o tabela indexata este urmatorul

- a) este indexata tabela, adica se stabilec criteriile de ordonare dorite si se creeaza fisierul index asociat tablei;
- b) in momentul in care se doreste folosirea tablei indexate, acesta se deschide si impreuna cu ea sunt deschise si fisierele index asociate, fie automat de catre SGBD, fie manual de catre utilizator,
- c) se realizeaza operatiile dorite asupra tablei(adaugare, modificare, stergere, listare) inregistrările fiind vazute in ordinea data de fisierul index activ sau de eticheta index activa. Fisierele index deschise sunt actualizate automat (de asemenea este refacuta ordinea) conform modificarilor facute in tabela.
- d) dupa icheierea operatiilor efectuate asupra tablei acesata se inchide, si odata cu ea trebuie inchise si fisierele index.

Sunt doua modalitati de a realiza indexarea tabelor.

Prima utilizeaza un fisier index compus (pe baza unei chei multiple) cu extensia .CDX (compound index file) . FoxPro permite deschiderea simultana a mai multor fisiere .CDX (pentru o anumita tabela) dar unul singur, cel activ (principal) sau o singura eticheta (cea activa) va controla ordinea inregistrarilor. Unele comenzi (cum ar fi SEEK) folosesc fisierul index principal sau eticheta principala pentru cautarea inregistrarilor. Fisierul index principal sau eticheta index activa pot fi specificate cu clauza INDEX a comenzii USE sau cu comanda SET INDEX si SET ORDER

Optiunea TAG a unei comenzi INDEX inmagazineza intr-un fisier .CDX informatiile privind indexarea. Fisierul .CDX creat in acest caz se numeste fisier index structural.(Fisierele structurale sunt deschise si asociate automat tablei odata cu deschiderea acesteia prin comanda USE).

A doua metoda este aceea de a crea un fisier individual (.IDX) pentru fiecare indexare realizata(contin un singur criteriu de ordonare). De exemplu indexarea tablei agenti dupa valorile campurilor *nume* sau *prenume* va conduce la crearea a doua fisiere index (.IDX) separate.

Comanda INDEX

Sintaxa:

```
INDEX ON <expr> TO < fisier idx >          | TAG < nume eticheta>
      [OF < fisier cdx >][FOR <expresieLogica>][COMPACT]
      [ASCENDING | DESCENDING]
      [UNIQUE]
      [ADDITIVE]
```

Efect: Creeaza un fisier index sau o eticheta pentru afisarea inregistrarilor tablei curente in ordine logical.

Inregistrările dintr-o tabela indexata sunt afisate si accesate in ordinea specificata de expresia de indexare.(Ordinea fizica nu este schimbata de un fisier index).

Daca TALK este SET ON, FoxPro da informatii despre numarul inregistrarilor care au fost indexate in timpul procesului de indexare. Comanda DISPLAY STATUS da inforamtii despre fisierele index deschise.

Numarul de fisiere index (.IDX or .CDX) care pot fi deschise simultan este limitata de memoria disponibila si resursele sistemului de operare.

Obs: Expresiile de indexare pot contine functii definite de utilizator (User-Defined Functions (UDF)).

De exemplu poate fi folosita o functie definita de utilizator pentru a extrage numele strazii dintr-un singur camp care contine informatii despre adresa. UDF returneaza portiunea din sirul de caractere reprezentand numele strazii, dupa care inregistrarile din tabela pot fi accesate in ordinea data de numele strazii

<expr>- este expresia de indexare si poate contine numele unui camp sau campuri din tabela curenta.

Obs. Desi nu este recomandat, <expr> poate fi numele unei variabile de memorie, un element al unui tablou sau un camp sau o expresie formata cu campuri din alta tabela deschisa in alta zona de lucru

TO < fisier idx > - cu ajutorul acestei clauze se poate crea un fisier .IDX. The index fisier is given the default extension .IDX, which can be overridden by explicitly including a different extension or by changing the default index extension in the FoxPro configuration fisier. Standard MS-DOS rules for naming fisiers must be observed when creating index fisiers.

TAG < nume eticheta > [OF < fisier cdx >] - permite crearea unui fisier index compus cu extensia .CDX. Un fisier index compus este un fisier index care poate contine un numar de intrari(etichete=tag) index. Fiecare eticheta este identificata prin numele sau (unic). Numele, ca si in cazul variabilelor de memorie poate incepe cu o litera sau o liniuta de subliniere si consta intr-o combinatie de pana la 10 litere cifre sau liniute de subliniere. Numarul de etichete dintr-un fisier index compus este limitat doar de spatiul liber pe disc sau de memoria disponibila. Pot fi create doua tipuri fisiere index compuse.

Primul tip, cel al **fisierelelor index compuse structurale** este creat atunci cand este inclusa optiunea TAG <nume eticheta> fara includerea clauzei OF < fisier cdx >. Un fisier compus structural are acelasi nume cu tabela si este deschis automat la deschiderea tablei. Daca un fisier index compus structural nu poate fi localizat sau a fost sters sau redenumit este afisata o casuta de dialog atunci cand se incearca deschiderea tablei. Alegerea optiunii Cancel anuleaza actiunea de deschidere a tablei, alegere optiunii Ignore deschide tabela si inlatura marcajul din header-ul tablei care indica prezenta unui fisier index compus structural.

Obs. Pentru a reasocia un fisier index compus structural unei table se foloseste comanda
USE <nume fisier> INDEX <nume fisier>.

Al doilea tip, cel al **fisierelelor index compuse nestructurale** poate fi creat cu ajutorul clauzei OF < fisier cdx > inclusa dupa TAG <nume eticheta>. Spre deosebire de fisierele index compuse structurale, cele nestructurale trebuie deschise explicit cu comanda SET INDEX sau cu clauza INDEX a comenzii USE.

Daca un fisier compus a fost deja creat si deschis atunci folosirea comenzii INDEX cu TAG <nume eticheta > adauga o eticheta la fisierul index. Daca un astfel de fisier nu a fost creat deja atunci este creat automat unul.

FOR <expresieLogica>-permite crearea unui fisier index care actioneaza ca un filtru pentru tabela. Numai inregistrarile care indeplinesc conditia <expresieLogica> vor fi disponibile pentru accesare si afisare; sunt create chei index in fisierul index numai pentru acele inregistrari care indeplinesc conditia de filtrare

COMPACT- creaza un fisier.IDX compact.

ASCENDING | DESCENDING – permit crearea fisierelor .CDX fisiers in ordine crescatoare (ASCENDING) sau descrescatoare (DESCENDING).

In cazul fisierelor .IDX nu poate fi folodsita clauza DESCENDING. Acest lucru poate fi totusi realizat cu comenzile SET INDEX si SET ORDER.

UNIQUE - specifica faptul ca numai prima inregistrare care se potriveste unei valori a cheii de indexare este inclusa intr-un fisier .IDX sau unei etichete .CDX. (UNIQUE este folosita pentru a preveni afisarea sau accesul la inregistrari duplicate)

ADDITIVE- La crearea cu ajutorul comenzii INDEX a unui fisier index pentru o tabela, celelalte fisiere index (cu exceptia celui compus structural) sunt inchise. Includerea clauzei ADDITIVE face ca aceste sa ramana deschise.

Exemplu

CLOSE DATABASES

USE agenti

INDEX ON nume TO lista_agent

CLOSE DATABASES

USE agenti

INDEX ON nume+prenume TO lista_ag

In exemplu urmator este creat un filtru.

CLOSE DATABASES

USE agenti

INDEX ON nume+prenume FOR nume > "POP" TO sub_agenti

Crearea unei etichete

CLOSE DATABASES

USE agenti

INDEX ON zona TAG zona1

INDEX ON nume+prenume TAG np OF ag.cdx

Comanda SET INDEX

Sintaxa: SET INDEX TO

[<lista fisiere index> | ?]

[ORDER <expN>

| <idx index file>

| [TAG] <tag name>

[OF <cdx file>]
[ASCENDING | DESCENDING]]
[ADDITIVE]

Efect: deschide unul sau mai multe fisiere index asociate tabele active.
(pot fi deschise atat fisierele .IDX cat si cele .CDX). Numai un fisier index .IDX sau o eticheta dintr-un fisier .CDX controleaza la un moment dat ordinea inregistrarilor dintr-o tabela.

Folosirea comenzii SET INDEX TO fara alte argumente inchide toate fisierele index deschise (cu exceptia celor structurale compuse) in zona de lucru curenta.

<lista fisiere index> | ?- deschide fisierul index al carui nume este specificat prin <lista fisiere index>. Nu trebuie indicata extensia fisierului decat daca exist fisiere .IDX si .CDX cu acelasi nume. Poate fi deschise si alte fisiere index daca introducem numele lor separate prin virgula in lista de fisiere index. Primul fisier din lista este cel principal (care controleaza care inregistrari vor fi accesate si afisate). Modificarea ordinului de prioritate se face cu ajutorul comenzii SET ORDER TO;

De asemenea poate fi deschis un singur fisier .IDX cu comanda SET INDEX TO ?. Va fi initiat un dialog pentru alegerea fisierului index (Este deschisa fereastra **Open file**).

ORDER <expN>- expresia numerica <expN> specifica numarul fisierului index principal, respectiv etichetei principale din lista de fisiere index. Numarul corespunde ordinii in care au fost scrise fisierele in lista. Etichetele din fisierele structurale compuse(daca exista) sunt considerate in ordinea in care au fost create. Daca <expN> = 0, atunci inregistrările din tabela sunt afisate si accesate in ordinea fizica, dar fisierele index raman deschise.

ORDER <idx index file>-stabileste fisierul index .IDX principal.

ORDER [TAG] <tag name> [OF <cdx file>]- stabileste eticheta principala din fisierul index .CDX specificat.

ASCENDING | DESCENDING-determina ordinea inregistrarilor tabelei active. Fisierele index sau cheile nu se schimba, ci numai ordinea in care se face accesul la inregistrari.

ADDITIVE-permite deschiderea unui fisier index pentru o tabela in conditiile in care fisierele index deschise anterior raman deschise.

Comanda SET ORDER

Sintaxa:

SET ORDER TO

[<expN1> | <idx index file>
| [TAG] <tag name>
[OF <cdx file>]
[IN <expN2> | <expC>]
[ASCENDING | DESCENDING]]

Efect: Cand se lucreaza cu chei multiple in cadrul unor fisiere index compuse comanda permite utilizatorului activarea oricarei chei.

Exemplu

INDEX AGENTI ON nume TAG nume

Set ORDER TO TAG nume

[<expN1> | <idx index file> indica numele (printr-un numar <expN1> sau direct <idx index file>) fisierului index principal sau etichetie index active. Numarul <expN1> , daca este folosit reprezinta numele fisierului nu numarul <expN1> din lista folosita intr-o comanda USE sau SET INDEX. Facem observatia ca intai sunt numarate fisierele index simple.

IN <expN2> | <expC> indica zona de lucru in care este deschisa tabela pentru care se deschid fisierele index. Actiunea restului clauzelor este asemanatoare cu cea de la comanda precedenta.

Exemplu de numarare a fisierelor sau etichetelor. Presupunem ca a fost creata tabela exemplu.dbf care are asociate fisierele nume.idx, salariu.cdx, functie.idx.

USE exemplu INDEX nume.idx, salariu.cdx, functie.idx IN 1

Presupunem ca tabela exemplu.dbf are un fisier index structural (exemplu.cdx) cu doua etihete, s_2006 si s_2005. Fisierul va fi deschis odata cu deschiderea tabelii exemplu.dbf. Deoarece fisierele index .IDX sunt numarate primele folosim SET ORDER TO 1 pentru ca nume.idx sa devina activ si SET ORDER TO 2 pentru ca functie.idx sa devina activ:

Apoi sunt numarate etichetele fisierelor compuse, incepand cu cele ale fisierelor index structurale (in cazul nostru ale fisierului exemplu.cdx):

Comanda SET ORDER TO 3 face ca eticheta s_2006 sa fie cea activa iar

Comanda SET ORDER TO 4 face ca eticheta s_2005 sa fie cea activa.

In continuare vor fi numarate etichetele fisierului compus salariu.CDX:

Comanda SET ORDER TO 5 face ca prima eticheta a fisierului compus salariu.cdx sa devina activa etc

Comanda REINDEX

Sintaxa: REINDEX [COMPACT]

Efect: Reface fisierele index deschise. (Up-dateaza fisierele index).

Refacerea fisierelor index este necesara atunci cand tabela a fost deschisa fara deschiderea fisierelor index asociate si au fost facute in tabela modificari care afecteaza valorile cheilor de indexare.

Exemplu de folosire cu comadna USE:

USE <table> INDEX <outdated index or indexes>

REINDEX

Clauza COMPACT converteste fisierele.IDX obisnuite la fisiere compacte.

Comanda CLOSE INDEX – inchide toate fisierele index deschise la un moment dat in zona de lucru curenta cu exceptia fisierelor index structurale.

X. RELATII INTRE TABELE

Pentru a justifica necesitatea relationarii tabelelor vom considera exemplul urmator in care presupunem ca dorim sa organizam datele privitoare la intrarea materialelor dintr-un depozit. Trebuie memorate informatii despre fiecare material (cod_mat,denumire, unit_mas), despre fiecare furnizor (cod_furn, denumire, adresa, cod_fiscal) si despre cantitatile care intra in depozit, pretul cu care este obtinut de la furnizori materialul, data la care a intrat in depozit etc. Pentru a elimina erorile datorate valorilor repetate si pentru a reduce spatiul de memorare informatiile de mai sus se vor regasi in 3 tabele MATERIAL (cod_mat,denumire, unit_mas), FURNIZOR (cod_furn, denumire, adresa, cod_fiscal) si respectiv CONTRACT(cod_mat, cod_furn, cantitate, pret_u, data_i). Tabela INTRARI va fi legata prin intermediul campurilor cod_mat, respectiv cod_furn de tabelele MATERIAL si respectiv FURNIZOR. Selectarea unei inregistrari din tabela INTRARI va conduce la selectarea automata a cate unei inregistrari din tabelele MATERIAL si respectiv FURNIZOR. Relatia care se stabileste intre tabelele unei BD relationale este o relatie de subordonare in care una din tabele va fi conducatoare (tabela parinte) iar cealalta (celelalte) va fi tabela condusa (tabela copil). Mutarea indicatorului de inregistrari in tabela parinte pe o anumita inregistrare va determina mutarea indicatorului de inregistrari din tabela copil pe inregistrarea corespunzatoare. Relatia (legatura) dintre tabela parinte si tabela copil se realizeaza prin intermediul unei expresii, cheia de indexare. Aceasta cheie de indexare poate fi un camp comun al tabelii parinte si al tabelii copil, dar poate fi si o expresie numerica, caz in care indicatorul de inregistrari al tabelii copil va fi mutat pe inregistrarea al carui numar este egal cu valoarea expresiei de indexare.

Modul de lucru cu tabelele intre care se stabilesc relatii este urmatorul:

1. Se deschid tabelele componente ale bazei de date relationale, fiecare in cate o zona de lucru distincta;
2. Tabelele copil vor fi indexate cu o cheie de indexare care coincide cu cea a relatiei sau este inclusa in aceasta.
3. Sunt stabilite relatiile intre tabelele deschise anterior;
4. Se executa operatii asupra bazei de date relationala, adica asupra tabelilor componente(se adauga date, se modifica cele existente)
5. Se inlatura relatiile stabilite intre tabele si se inchid tabelele.

Legarea tabelor este realizata in FoxPro cu ajutorul comenzii **SET RELATION**. Aceasta comanda stabileste o relatie intre tabela parinte, activa si o alta tabela (considerata copil) indicata prin intermediul alias-ului sau introdus in clauza INTO>.

Sintaxa:

```
SET RELATION TO <expr> INTO <expN1|expC1>[<expr2> INTO
<alias>...][ADDITIVE]
```

Unde

<expr> este de obicei expresia de indexare activa in tabela copil cu care se face relationare(poate fi un camp comun pentru ambele tabele sau o expresie numerica).

<expN1|expC1> este numele tabelii copil (tabela cu care se face legatura) indicat prin expN1, numarul zonei de lucru in care este deschisa tabela sau numele sau.

Clauza ADDITIVE este optionala si este utilizata atunci cand se realizeaza o relationare de mai multe tabele. Aceasta clauza face ca relatiile existente pentru tabela activa sa nu fie sterse si la acestea sa se adauge noua relatie definita. Absenta clauzei face ca noua relatie sa inlocuiasca vechile relatii ale tabelului active. Comanda SET RELATION TO fara alte clauze inlatura toate relatiile tabelului active.

Limitarea datelor selectate din tabelul supuse legaturii se realizeaza cu ajutorul comenzilor SET FILTER si INDEX ON.

Sintaxa comenzii SET FILTER este SET FILTER TO <Expresie>, unde <Expresie> reprezinta conditiile pe care inregistrarile trebuie sa le satisfaca.

In exemplul de mai jos sunt create tabelele MATERIAL, FURNIZOR si CONTRACT ,descrise mai sus, intre care vor fi stabilite legaturi astfel: intre CONTRACT si MATERIAL prin intermediul campului COD_MAT, care reprezinta si cheia de indexare pentru tabela copil MATERIAL, iar intre CONTRACT si FURNIZOR prin intermediul campului COD_FURN (COD_FURN este cheia de indexare pentru FURNIZOR). Ca iesire este obtinuta lista tuturor contractelor, sub forma, denumire material, denumire furnizor, cantitate, data intrarii, lista contractelor pentru care cantitatea intrata este mai mare de 20 si respectiv 40.

Exemplu:

SELE 1

```
CREATE TABLE MATERIAL(COD_MAT N(3),DENUMIRE C(20), UNIT_MAS
C(3), PRET_U N(4))
```

APPEND

```
INDEX ON COD_MAT TO IMATERIAL
```

```
SET INDEX TO IMATERIAL
```

SELE 2

```
CREATE TABLE FURNIZOR (COD_FURN N(3), DENUMIRE C(20), ADRESA M,
COD_FISCAL C(10))
```

APPEND

```
INDEX ON COD_FURN TO IFURNIZOR
```

```
SET INDEX TO IFURNIZOR
```

SELE 3

```
CREATE TABLE CONTRACT(COD_MAT N(3), COD_FURN N(3),CANTITATE
N(4), DATA_I D)
```

APPEND

```
INDEX ON COD_MAT TAG em
```

```
SET ORDER TO em
```

```
SET RELATION TO COD_MAT INTO 1
```

```
SET RELATION TO COD_FURN INTO 2
```

```
LIST MATERIAL.DENUMIRE, FURNIZOR.DENUMIRE, CANTITATE,
MATERIAL.PRET_U, DATA_I
```

Rezultat:

Cafea	SC PROD	30	8	06/02/06
Orez	SC PROD	16	2,2	07/03/06
Faina	SA Banat	40	1,4	07/04/06

```
SET FILTER TO CANTITATE>20
```

```
LIST MATERIAL.DENUMIRE, FURNIZOR.DENUMIRE, CANTITATE, DATA_I
```

Rezultat:

Cafea	SC PROD	30	8	06/02/06
Faina	SA Banat	40	1,4	07/04/06

SET FILTER TO CANTITATE>35

LIST MATERIAL.DENUMIRE, FURNIZOR.DENUMIRE, CANTITATE

Rezultat:

Faina	SA Banat	40	1,4	07/04/06
-------	----------	----	-----	----------

Comanda SET RELATION OFF are sintaxa SET RELATION OFF INTO <exprC> si efectul de inlaturare a relatiei intre tabela curenta si tabela copil specificata de <exprC>.

Exemplu: Sa se inlature legatura intre tabelele CONTRACT si MATERIAL

SELE 3

SET RELATION OFF INTO 1

Funcția TARGET are sintaxa

TARGET(<expN1> [, <expN2> | <expC>])

Si returneaza un sir de caractere ce reprezinta numele tabelii copil (tabelii secundare) cu care s-a facut relationarea.

<expN1>-specifica un numar de la 1 la numarul de legaturi care au fost stabilite pentru tabela curenta. Daca <expN1> este mai mare decat numarul de legaturi ale tabelii curente atunci functia va returna sirul nul.

<expN2> | <expC>-specifica numarul zonei de lucru sau alias-ul tabelii directoare deschise într-o alta zona de lucru decat cea curenta pentru care va fi returnat numele tabelii secundare.

De exemplu datca tabela CONTRACT din exemplele anterioare este deschisa in zona de lucru 3 atunci secventa de comenzi de mai jos va conduce la afisarea tabelii secundare(intr-o ordine aleatoare) cu care aceasta a fost relationata.

Exp:

SELE 5

?TARGET(1,3)

Relatii de tipul unu la unu

In cazul in care intre tabela conducatoare si tabela secundara este stabilita o relatie de tipul unu la unu, adica fiecarei inregistrari din tabela parinte i se asociaza o unica inregistrare din tabela copil atunci relationarea celor doua tabele prin comanda SET RELATION este simpla (pentru fiecare inregistrare din tabela parinte se gaseste o singura inregistrare in tabela copil). Cheia de relationare poate fi un singur camp sau o concatenare de mai multe campuri astfel incat legatura stabilita sa fie de tipul precizat mai sus. De exemplu putem avea situatia in care comanda SET RELATION se scrie SET RELATION TO (<nume camp1 > +<nume camp2 > +...+<nume campn >) INTO <nume tabela>.

Relatii de tipul unu la mai multi

Daca valoarea chei de relationare asociaza unei inregistrari din tabela parinte mai multe inregistrari din tabela copil atunci avem de a face cu o relatie de tipul unu la mai multi, relatia dintre tabele se realizeaza tot prin comanda SET RELATION dar este necesara folosirea comenzii SET SKIP pentru a se continua cautarea inregistrarii urmatoare din tabela secundara pentru care valoarea cheii de indexare se potriveste cu cea din tabela parinte. Astfel o relatie dintre tabele (de tip unu la unu) este transformata intr-o relatie de tip unu la mai multi

Sintaxa:

SET SKIP TO [TableAlias1 [, TableAlias2] ...]

TableAlias1 [, TableAlias2] ... –specifica alias-uri ale mai multor tabele copil (care vor fi separate prin virgula). In comenzile care permit folosirea domeniului inregistrarii (DISPLAY, LIST, etc), inregistrările din tabela parinte vor fi repetate pentru fiecare inregistrare corespunzatoare tablei copil.

Folosirea comenzii SET SKIP TO fara argumente conduce la transformarea relatiei de tip unu la mai multi intr-o relatie de tip unu la unu.

Exemplu: Daca in tabela copil MATERIAL (din exemplele de mai sus) sunt introduse doua inregistrari cu acelasi cod (lucru care se poate intampla daca un material este achizitionat de la acelasi furnizor cu preturi diferite; de exemplu produsul *cafea* are codul 1 si este achizitionat de la acelasi furnizor la preturi diferite) atunci efectul comenzilor

SET SKIP TO MATERIAL

LIST COD_MAT, MATERIAL.DENUMIRE, FURNIZOR.DENUMIRE, CANTITATE, DATA_I

este:

1	Cafea	SC PROD	30	8	06/02/06
1	Cafea	SC PROD	38	8	06/02/06
2	Orez	SC PROD	16	2,2	07/03/06
3	Faina	SA Banat	40	1,4	07/04/06

Indicatorul de inregistrari ramane pozitionat pe valoarea 1 a cheii de relationare (COD_MAT) in tabela CONTRACT pana ce vor fi selectate toate inregistrările din tabela copil MATERIAL care se potrivesc acestei valori. Daca in continuare se scriu comenzile

SET SKIP TO MATERIAL

LIST COD_MAT, MATERIAL.DENUMIRE, FURNIZOR.DENUMIRE, CANTITATE, DATA_I

Efectul este:

1	Cafea	SC PROD	30	8	06/02/06
2	Orez	SC PROD	16	2,2	07/03/06
3	Faina	SA Banat	40	1,4	07/04/06

Gasirea unei inregistrari in tabela copil care se potriveste cu cheia de relationare din tabela parinte determina mutarea indicatorului de inregistrari din tabela parinte pe urmatoarea inregistrare.

Fereastra DATA SESSION

Poate fi folosta pentru realizarea de legaturi temporare intre tabele si pentru afisarea tabelelor sau vederilor sau setarea proprietatilor zonei de lucru. Pentru deschiderea ferestrei **Data session** se alege optiunea cu acelasi nume a meniului **Window** al ferestrei principale Visual FoxPro.

In fereastra **Data session**

caseta **Current Session** – sesiunii de lucru curente.

Aliases – afiseaza numele vederilor sau tabelelor fara extensie.

Relations- Indica relatiile temporare stabilite intre tabelele sau vederile dincutia **Aliases**.

Properties – afiseaza cutia de dialog **Work Area Properties**, in care se poate modifica structura tablei, se pot selecta fisierele index si campurile sau se pot defini filtre pentru date. Daca in cutia **Aliases** nu este listat nici un nume atunci este deschisa o cutie de dialog **Open** care permite deschiderea tabelor sau vederilor dorite.

Browse -afiseaza tabela sau vederea selectata in cutia **Aliases** intr-o fereastra **Browse** unde pot fi examinate sau editate date.

Open - permite deschiderea tabelor sau vederilor dorite.

Close – inlatura tabelele sau vederile selectate si toate fisierele asociate din cutia **Aliases**.

Relations - Defineste relatii intre tabele sau vederi folosin **Expression Builder**. Cutia de dialog **Set Index Order** apare inainte de deschiderea cutiei de dialog **Expression Builder** daca nu a fost stabilita nici o ordine de indexare intre tabele ce vor fi relationate.

1-To-Many –afiseaza cutia de dialog **Create One-To-Many Relationships** care permite stabilirea de relatii temporare de tip unu la mai multi intre tabela copil si tabela parinte. Butonul este activ atunci cand cheia de relationare face ca relatiile intre tabele sa fie de tipul unu la mai multi.

Cutia de dialog Create One-To-Many Relationships

permite stabilirea unei relatii de unu la mai multi si are un efect similar comenzii SET SKIP.

Fereastra de dialog care se deschide are urmatoarele optiuni:

Child Aliases – este lista tabelor copil cu care tabela parinte specificata(in partea superioara a cutiei de dialo) are relatii.

Selected Aliases – reprezinta lista tabelor implicate in sesiunea de lucru curenta in relatii de tip unul la mai multi cu tabela parinte.

Move -copiaza alias-urile tabelor copil selectate in cutia **Child Aliases** in cutia **Selected Aliases**. Butonul este activ daca s-a realizat o selectie in cutia **Child Aliases**.

All - copiaza toate alias-urile tabelor copil din **Child Aliases** in **Selected Aliases**.

Remove –inlatura alias-urile selectate din cutia **Selected Aliases**. Pentru ca butonul sa fie activ trebuie selectate alias-uri in cutia **Selected Aliases**.

Remove All - inlatura toate alias-urile din **Selected Aliases**.

XI. PROGRAMAREA PROCEDURALA

Un **program** reprezinta o succesiune de instructiuni scrisa conform regilor proprii limbajului de programare si care respectand un anumit algoritm rezolva o anumita problema [ManI].

Realizarea programelor.

Limbajul de programare VFP contine comenzi(ce pot fi numite si instructiuni) si functii cu ajutorul carora se scriu datele si se efectueaza, conform unor algoritmi, prelucrarea acestora in vederea obtinerii anumitor rezultate. De asemenea un program VFP admite folosirea comenzilor din nucleul SQL.

Programul sursa VFP este un text care poate fi creat cu ajutorul comenzii MODIFY COMMAND <nume fis>. Aceasta comanda apeleaza editorul de texte propriu sistemului VFP in care se introduce textul sursa al programului. Va apare o fereasta de editare in care textul programului poate fi introdus sau modificat. Iesirea din fereasta se face tastand [Ctrl + W] pentru salvare sau [Esc] pentru abandon. Fisierul salvat va avea numele introdus in comanda si extensia .prg.

Apelul unui program sursa se face prin comanda Do care preia fisierul sursa, il compileaza si rezulta fisierul obiect (.FXP), apoi il link-editeaza si rezulta fisierul executabil (.EXE) pe care il lanseaza in executie. Fisierul compilat ramane pe disc si dupa executia programului in timp ce fisierul executabil nu. Pentru a obtine fisierul executabil trebuie folosita componenta VFP de RUNTIME-BUID EXE.

Structuri de control fundamentale

Ordinea secventiala de executie a instructiunilor care formeaza un program s-a dovedit a fi ineficienta in multe situatii cum ar fi acelea in care un grup de instructiuni trebuie executat de mai multe ori sau daca executia unui anumit grup de instructiuni este conditionata de indeplinirea anumitor conditii.

Rezolvarea acestor neajunsuri a fost realizata odata cu introducerea modelului programarii structurate. Exista o teorema in teoria programarii structurate care demonstreaza ca pentru realizarea oricarui program sunt necesare trei dintre structurile fundamentale; structura secventiala, structura alternativa si cea repetitiva cu test initial sau final. Aceste structuri se regasesc in totalitate in sistemul VFP.

XI.1 Structura secventiala

O structura secventiala sau liniara poate fi realizata din oricare grup de comenzi (instructiuni) care pot fi folosite si in fereasta de comanda.

Mai precis (comenzile) instructiunile care pot intra intr-o structura secvantiala pot fi a)-Comenzile pentru atribuire

-Sunt comenzi care realizeaza initializarea unor variabile sau atribute cu o anumita valoare care poate proveni din evaluarea unei expresii sau este o constanta. Prin initializare se face de fapt si declararea tipului variabilelor. Exista doua moduri de atribuire: prin atribuirea propriu-zisa "=" sau prin comanda STORE.

Sintaxa: STORE <expr> TO <lista var mem>|<nume tablou>

Efect: permite stocarea datelor în variabile de memorie sau în tablouri.

<expr>-reprezintă valoarea ce va fi stocată;

<lista var mem>-este o listă de variabile sau elementele unui tablou în care sunt stocate valorile variabilei <expr>.

<nume tablou> -este numele unui tablou în care sunt stocate valorile variabilei <expr>.

b)-Comenzile pentru operațiile de intrare/ieșire. (Ieșirea standard este ecranul, intrarea standard este tastatura)(Comenzile SET PRINTER, SET CONSOLE)

b1) comenzile pentru afișare pe ecran/la imprimantă. Comenzile ?, ??, ???, \, \\ etc.

b2) comenzi pentru afișare/citire pe/de pe perifericul standard, Comenzile @...SAY, @...GET, READ

XI.2 Structura alternativă

În VFP sunt folosite două tipuri de structuri alternative, cea cu două ramuri (comanda IF) și structura alternativă cu mai multe ramuri (comanda CASE)

Comanda IF

Sintaxă:

```
IF <expL>
    <secvență-de-comenzi 1>
[ELSE
    <secvență-de-comenzi 2>]
ENDIF
```

Efect: Este evaluată expresia <expL> și dacă aceasta ia valoarea adevărată (.T.) este executată <secvență-de-comenzi 1>. Dacă <expL> este falsă și comanda IF conține cuvântul cheie ELSE atunci este executată <secvență-de-comenzi 2>. Dacă <expL> este falsă și nu apare ELSE atunci toate comenzile dintre IF și ENDIF sunt ignorate. În acest caz, execuția programului continuă cu prima comandă ce urmează după ENDIF.

Poate fi folosit un bloc IF ... ENDIF în interiorul altui bloc IF ... ENDIF.

Pot fi plasate comentarii pe aceeași linie după IF, ELSE și ENDIF. Ele vor fi ignorate în timpul compilării și execuției programului.

Exemplu: Să se scrie un program care să afișeze informațiile stocate în tabela agentii referitoare la prima persoană care este înregistrată cu numele MARCU

```
CLOSE ALL
USE AGENTI
IF UPPER(nume)="MARCUS"
    DISPLAY
ELSE
    ? 'numele albu nu a fost găsit'
ENDIF
```

Comanda CASE

Sintaxa:

```
DO CASE
    CASE <expL1>
        < secventa-de-comenzi 1>
    [CASE <expL2>
        < secventa-de-comenzi 2>
    ...
    CASE <expLN>
        < secventa-de-comenzi N>]
    [OTHERWISE
        < secventa-de-comenzi N+1>]
ENDCASE
```

Efect: Comanada DO CASE este folosita pentru a executa o multime de comenzi FoxPro bazate pe rezultatul evaluarii unei conditii logice. Sunt evaluate o succesiune de conditii logice pe baza evaluarii carora este determinat setul de comenzi ce va fi executat.

Cand o prima expresie logica (<expLi>, $i \in \{1, \dots, N\}$) este adevarata atunci se executa comenzile ce urmeaza pana la urmatorul CASE sau pana la ENDCASE. Executia programului continua cu comanda care urmeaza lui ENDCASE. Daca o expresie logica este falsa atunci comenzile care urmeaza pana la urmatorul CASE sunt ignorate.

O singura clauza CASE este executata – prima pentru care expresia logica este adevarata. Restul clauzelor CASE sunt ignorate.

Daca toate clauzele CASE sunt evaluate ca false (.F.) atunci OTHERWISE determina executarea comenzilor < secventa-de-comenzi N+1>] si executia continua cu prima instructiune (comanda) care urmeaza lui ENDCASE.

Daca OTHERWISE nu este inclusa (si toate clauzele CASE sunt evaluate ca false (.F.)) atunci executia continua cu prima instructiune (comanda) care urmeaza lui ENDCASE.

Exemple: Sa se afiseze trimestrul in care ne aflam.

```
STORE CMONTH(DATE()) TO luna && Luna curenta
DO CASE
    CASE INLIST(luna,'January','February','March')
        ? 'primul trimestru'

    CASE INLIST(luna,'April','May','June')
        ? 'al doilea trimestru'

    CASE INLIST(luna,'July','August','September')
        ? 'al treilea trimestru'

    OTHERWISE
        ? 'al patrulea trimestru'
ENDCASE
```

XI.3 Structura repetitiva

In VFP sunt implementate

- Structura repetitiva conditionata anterior(DO WHILE)

- Structura repetitiva conditionata anterior cu numarator (FOR)
- Structura specifica VFP, ce parcurge o tabela de date automat de la inceput pana la sfarsit, in mod conditionat (SCAN)

Structura repetitiva conditionata anterior DO WHILE

Sintaxa:

```
DO WHILE <expL>
    < secventa-de-comenzi >
    [LOOP]
    [EXIT]
ENDDO
```

Efect: Este executat setul de instructiuni cuprins intre DO WHILE si ENDDO cat timp expresia logica <expL> ramane adevarata (.T.). Orice comanda DO WHILE se termina cu ENDDO.

Comentariile pot fi plasate dupa DO WHILE si ENDDO pe aceeasi linie.

<expL>-este o expresie logica. Comenzile < secventa-de-comenzi > dintre DO WHILE si ENDDO sunt executate cat timp <expL> ia valoarea de adevar true (.T.).

LOOP-poate fi plasata oriunde intre DO WHILE si ENDDO si determina reluarea comenzii DO WHILE.

EXIT –transfera controlul programului la prima comanda care urmeaza lui ENDDO. EXIT poate fi plasata oriunde intre DO WHILE si ENDDO si reprezinta o iesire fortata din ciclul DO WHILE...ENDDO.

Example: In acest exemplu este obtinuta cantitatea totala de lemn intrata intr-un depozit. Se foloseste tabela DEPOZIT(denmat C(20), Cantitmat N(5), dataint D)

```
CLOSE DATABASES
SET TALK OFF
USE DEPOZIT
totlemn = 0
DO WHILE .T.
    IF EOF()
        EXIT
    ENDIF
    IF UPPER(denmat) = 'LEMN'
        totlemn = totlemn + Cantitmat
    SKIP
    LOOP
    ENDIF
    SKIP
ENDDO
?totlemn
```


Structura repetitiva conditionata anterior cu numarator FOR

Sintaxa:

```
FOR <var> = <expN1> TO <expN2>
    [STEP <expN3>]
    < secventa-de-comenzi >
    [EXIT]
    [LOOP]
ENDFOR | NEXT
```

Efect: Executa comenzile dintre FOR si ENDFOR de un numar specificat de ori <var> -este o variabila de memorie(sau un element de tablou) cu rol de contor. Ea determina de cate ori se executa ciclul de la FOR la ENDFOR|NEXT sau la EXIT. Contorul <var> pleaca de la o valoare initiala <expN1> si este incrementat cu valoarea <expN3>. In cazul in care clauza STEP lioseste contotul este incrementat cu 1. Contorul este comparat cu valoarea finala <expN2> si daca este mai mic sau egal cu valoarea finala <expN2> instructiunile care urmeaza clauzei FOR sunt executate din nou. In caz contrar (contorul este mai mare decat <expN2>) executia programului continua cu instructiunea care urmeaza dupa ENDFOR sau NEXT.

Observatie. Modificarea contorului in interiorul buclei poate afecta numarul de cicluri executate.

Daca valoarea <expN3> este negativa si valoare initiala <expN1> este mai mare decat valoarea finala <expN2>, atunci contorul este micșorat la fiecare executare a ciclului.

<expN1>, <expN2>, <expN3> -sunt expresii numerice, variabile de memorie sau constante a caror valoare poate fi modificata in interiorul ciclului fara ca numarul de cicluri executate sa se modifice(aceasta deoarece valorile lor sunt citite o singura data la intrarea in primul ciclu FOR)

EXIT-determina iesirea fortata din ciclul FOR

LOOP-determina reluarea comenzii FOR

Exemplu: Sa se scrie suma numerelor pare de la 1 la 2100

CLEAR

S=0

FOR i = 2 TO 2100

STEP 2

S=S+i

ENDFOR

?S

Structura repetitiva SCAN ... ENDSCAN

Sintaxa:

```
SCAN [NOOPTIMIZE]
    [<domeniul inregistrarilor>]
    [FOR <expL1>]
    [WHILE <expL2>]
```

[< secventa-de-comenzi >]
[LOOP]
[EXIT]
ENDSCAN

Efect: Muta indicatorul de inregistrari in interiorul tabelii curente si executa un bloc de comenzi pentru fiecare inregistrare care indeplineste conditiile specificate.

NOOPTIMIZE-specifica executia comenzii SCAN intr-un mod neoptimizat.

Daca exista <domeniul inregistrarilor> atunci se vor lua in considerare numai acele inregistrari care sunt specificate prin <domeniul inregistrarilor> (ALL, NEXT n, RECORD n, REST). Domeniul inregistrarilor implicit pentru SCAN este ALL.

FOR <expL1> WHILE <expL2> -sunt clauze care specifica faptul ca doar acele inregistrari din <domeniul inregistrarilor> care indeplinesc conditiile <expL1> sau <expL2> sunt considerate.

< secventa-de-comenzi >-reprezinta setu de comenzi ce va fi executat.

LOOP-poate fi plasat oriunde in interiorul buclei SCAN si determina reluarea comenzii SCAN.

EXIT – detrimina iesirea fortata din SCAN ... ENDSCAN.

XI.4 Modularizarea programelor

“Un modul reprezinta o unitate de program (de preferat independenta) construita dupa anumite reguli sintactice specifice limbajului, care realizeaza o anumita sarcina.[MID]”

“Modularizarea programelor este acea proprietate a limbajului care permite impartirea unui program complex in module, pentru a se realiza un control mai bun, o mai usoara depanare si o mai buna organizare.”

In general, in teoria limbajelor s-au dezvoltat doua tipuri de module: procedurile si functiile.

In general modulele comunica cu programul apelant doar prin intermediul parametrilor special destinati acestui scop si nu prin variabile de mediu sau alte variabile care fac modulul utilizabil doar in anumite conditii. Domeniul de valabilitate al unei variabile definite intr-un program incepe de la instructiunea de definire si pana la eliminarea sa din memorie (prin comanda RELEASE) sau pana la sfarsitul fisierului.

Daca un program apeleaza un modul, atunci variabilele programului sunt vazute si in interiorul modulului. Pentru modul ele se numesc variabile **globale**. O variabila definita in interiorul unui modul nu este disponibila si pentru programul apelant al modulului si variabila este numita **locala**.

In interiorul unui modul pot fi declarate si variabile globale dar atunci trebuie folosita o instructiune speciala.

Variabilele globale sau publice sunt **definite si create** prin comanda PUBLIC <lista de variabile>.

Variabilele globale sau publice pot fi accesate si modificate in orice modul in curs de executie de pe orice nivel (dar nu superior modulului in care s-a definit variabila).

Variabilele locale sau private sunt declarate cu comanda **PRIVATE** <lista de variabile>, dar spre deosebire de comanda **PUBLIC** aceasta nu le si creeaza (doar le declara).

Comunicarea intre module si programul apelant se face cu ajutorul unor variabile speciale numite **parametrii**. Transferul parametrilor intre module se face in general prin trei metode

- prin valoare (Modulului îi este transmisă o copie a variabilei, copie pe care el o poate modifica, dar variabila propriu zisă rămâne neschimbată.)
- prin referință (În zona de memorie comună este introdusă chiar zona de memorie a variabilei globale, astfel încât orice modificarea a sa efectuată de către modul este simțită și în program.)
- prin adresă (Programul apelant plasează în zona de memorie comună adresa unei variabile, iar programul apelant folosește variabila respectivă printr-o adresare indirectă. Modificările aduse variabilei în modul sunt văzute și de programul apelant. Modulul referă variabila primită ca parametru prin adresă sa (variabila de la adresă..))

Transmisia parametrilor la un subprogram se face în felul următor

- se stabilesc variabilele care se vor transmite subprogramului ca parametri într-o ordine stabilită de programator
- se stabilește un set de variabile locale subprogramului, care vor prelua valorile variabilelor transmise ca parametri de programul apelant.
- Corespondența între variabilele transmise de programul apelant și cele locale ale subprogramului se face prin poziția în două liste: lista cu parametri de apel al subprogramului și lista variabilelor locale specificată de comanda **PARAMETERS**.
- În modulul apelat se lucrează cu variabilele locale respective.
- Dacă transmisia este prin referință, la sfârșitul executării subprogramului conținutul variabilelor este trecut în variabilele din programul apelant transmise ca parametri;
- Dacă este o transmisie prin valoare atunci această copie nu are loc și variabilele de apelare nu vor mai fi actualizate cu noile valori ale variabilelor locale corespunzătoare.

Lista variabilelor transmise ca parametri este stabilită fie cu clauza **WITH** a comenzii **DO**, în cazul apelului unui subprogram sau al unei proceduri fie prin lista care urmează numelui funcției în cazul în care avem un apel de funcție.

Procedura – reprezintă o secvență logică de instrucțiuni, apelabilă printr-un nume dintr-o altă procedură externă care îndeplinește o anumită sarcină în program.

O procedură definită de utilizator nu poate intra în alcătuirea unei expresii ca operand fiind analoagă comenzilor standard **VFP**.

O procedură este alcătuită din :

- partea de început cu sintaxa **PROCEDURE** <nume procedură>
- lista de parametri cu sintaxa **PARAMETERS** <lista de parametri>
- corpul procedurii <secvență de instrucțiuni>
- sfârșitul procedurii cu sintaxa **RETURN** (menționăm aici că prezenta comenzii **RETURN** nu este obligatorie sfârșitul unei proceduri fiind marcat de întâlnirea unei comenzi **FUNCTION** sau **PROCEDURE** sau a sfârșitului de fișier).

Comanda **PROCEDURE** <nume procedură>- marchează începutul fizic al procedurii și stabilește numele acesteia <nume procedură>. Această procedură va putea fi apelată

ulterior prin acest nume. Prezenta numelui este obligatorie, lungimea lui fiind de maxim 8 caractere. Numele incepe cu o litera sau caracterul '_' si in scrierea lui nu se admit caractere speciale ASCII. In acelasi timp aceasta comanda constituie punctul de intrare, adica de receptionare a controlului executiei dintr-un program sau procedura apelanta.

PARAMETERS <lista de parametrii>- este o comanda cu rolul de a atribui parametrilor transmisi dintr-un program apelant, numele unor variabile de memorie locale, ce pot fi folosite in corpul procedurii pentru realizarea diferitelor operatii. Prezenta acestei parti intr-o procedura nu este obligatorie, dar daca apare sintaxa este cea precizata mai sus.

In mod implicit **transferul** parametrilor intre **programul apelant si procedura** se face prin **adresa**. Daca se doreste schimbarea acestui mod de transfer in **transfer prin valoare** atunci este necesara introducerea parametrilor, despartiti prin virgule intre paranteze.

Lista parametrilor poate fi constituita din variabile si constante si are rolul de a asigura comunicatia directa cu modulul apelant. Daca lista de parametrii contine un numar mai mare de variabile(sau tablouri) decat numarul transferat de programul apelant atunci parametrii care sunt in plus vor fi initializati cu valoarea logica False(.F.).

Daca valoarea unui parametru este modificata in cadrul procedurii, el va fi returnat programului apelant cu noua valoare.

Parametrii din lista pot indeplini rolul de parametri de **intrare**(datele de intrare care vor fi prelucrate) si de **iesire** (rezultatul prelucrarii datelor de intrare).

Utilizatorul poate afla numarul de parametri transferati catre ultima rutina apelata prin functia **PARAMETERS()**.

Corpul procedurii este alcatuit dintr-o succesiune de comenzi care realizeaza una sau mai multe sarcini.

Partea de sfarsit a unei proceduri este definita prin comanda **RETURN**. Marcheaza sfarsitu fizic al unei proceduri si preda controlul executiei procedurii programului apelant. Daca **RETURN** incheie executia unui program atunci controlul executiei este returnat sistemului VFP.

Apelul unei proceduri se face cu comanda **DO**.

Sintaxa:

DO <nume-fis> [**WITH** <lista parametri>][**IN** <nume-fis>]

Efect: transfera controlul executiei dintre procedura programul apelant (unde se afla comanda **DO**) catre procedura apelata, impreuna cu parametrii din lista(daca s-a folosit clauza **WITH**).

[**WITH** <lista parametri>]- permite specificarea unei liste de expresii, variabile de memorie, nume de tabele, nume de campuri din tabele, constante etc care sunt transmisi procedurii cu numele <nume-fis> prin adresa. Daca se doreste transmiterea prin valoare elementele listei sunt incluse intre paranteze. Numarul maxim al parametrilor transmisi este 24.

[**IN** <nume-fis>]- permite executarea procedurii intr-un program specificat.

Funcțiile

Cu ajutorul lor se execută anumite calcule, rezultând o anumită valoare care va fi returnată de modul. Apelul unei funcții poate fi inclus direct într-o expresie, la evaluare el fiind înlocuit cu valoarea returnată de funcție.

Ca și în cazul procedurilor corpul unei funcții conține

- partea de început cu sintaxa `FUNCTION <nume funcție>`
- lista de parametri cu sintaxa `PARAMETERS <lista de parametri>`
- corpul funcției <secvența de instrucțiuni>
- sfârșitul funcției cu sintaxa `RETURN <expresie>`.

Descrierea părților componente ale unei funcții este aceeași cu cea a procedurilor. Spre deosebire de proceduri transferul implicit al parametrilor în cazul unei funcții se face prin valoare. Absența clauzei `RETURN` în cazul unei funcții are drept efect returnarea valorii .T. de către funcție.

`RETURN <expresie>` specifică faptul că valoarea returnată de funcție este egală cu cea obținută prin evaluarea expresiei <expresie>. Pentru a schimba modul de transmitere implicită a parametrilor se pot folosi comenzile `SET UPDFPARMS` care va fi urmată de `TO VALUE` sau `TO REFERENCE`, în funcție de transferul dorit, prin valoare sau prin referință. Pentru a forța transferul unui parametru prin valoare la apelul unei funcții acesta se include între paranteze rotunde iar pentru forțarea transmiterii prin referință parametrul va fi precedat de simbolul '@'.

În continuare vom da două exemple, primul exemplifică forțarea transmiterii parametrilor unei funcții și cel de al doilea definește o funcție fără parametri.

```
CLEAR
STORE 2 TO a,b,c,d
R=test((a), @b,c,@d)
? a,b,c,d
FUNCTION test
PARAMETERS a1, a2, a3, a4
STORE 3 TO a1, a2, a3, a4
RETURN .T.
```

```
CLEAR
a=5
? var()
function var
return "este bine"
```

XII. AFISAREA SI CITIREA DATELOR

Afisarea si citirea datelor se realizeaza cu ajutorul comenzilor SAY, GET si READ.

XII.1 Afisarea datelor

Comanda SAY

Sintaxa: @ <linie, coloana> SAY <expr>
[FUNCTION <expC1>][PICTURE <expC2>]
[SIZE <expN1>, <expN2>]
[FONT <expC3> [, <expN3>]
[STYLE <expC4>][COLOR SCHEME <expN4>| COLOR <color pair list>]

Efect: Afiseaza iesirea pe ecran la linia si coloana specificata.

Comenzile @ ... SAY si @ ... GET pot fi combinate intr-o singura comanda. In acest caz este specificat un singur set de <linie, coloana> iar comanda se scrie

@ ... <linie, coloana> SAY <expr>
GET <var>.

In acest caz este inserat automat un spatiu intre iesirea comenzii @ ... SAY si zona in care se face citirea valorii variabilei comenzii GET.

Daca este folosita comanda SET DEVICE TO SCREEN, iesirea comenzii @ ... SAY apare in fereastra principala VFP sau in fereastra definita de utilizator activa. Folosirea comenzii SET DEVICE TO PRINTER, face ca iesirea sa fie directionata spre imprimanta.

<linie, coloana>-sunt expresii numerice cu valori 0 (prima linie/coloana de pe ecran de sub meniul principal VFP are numarul 0) sau mai mari ce specifica numarul liniei si respectiv coloanei de la care este afisata iesirea comenzii @ ... SAY. (In general, pentru afisarea in fereastra principala VFP, *linie* accepta valori cuprinse intre 0 si 24 iar *coloana* valori din intervalul 0-79)

Liniile sunt numarate de sus in jos. In cazul imprimantei prima linie este 1 iar numarul fizic de linii pe pagina determina numarul maxim de linii admise in comanda. Coloanele sunt numarate de la stanga la dreapta. In cazul imprimantei prima coloana are numarul 1 si numarul maxim de coloane este determinat de marimea fizica a paginii.

Daca iesirea @ ... SAY este directionata spre o fereastra definita de utilizator atunci numarul liniei si respectiv coloanei se calculeaza relativ la fereastra definita de utilizator.

De asemenea pozitia in fereastra principala VFP sau in fereastra definita de utilizator depinde de fontul selectat pentru acestea.

<expr>-reprezinta expresia care este evaluata si afisata incepand din punctul ale carui coordonate sunt (linie, coloana). Poate fi si o functie definita de utilizator.

FUNCTION <expC1> | PICTURE <expC2>-sunt clauze care controleaza modul in care <expr> este afisata sau tiparita. Principala deosebire intre cele doua cluze este ca fiecare caracter din clauza PICTURE controleaza un singur caracter din textul de iesire pe cand fiecare caracter din clauza FUNCTION se refera la cate o caracteristica a intregului text de iesire

Codurile FUNCTION pot fi incluse in clauza PICTURE. In acest caz clauza PICTURE trebuie sa inceapa cu @. De asemenea o clauza PICTURE poate contine coduri FUNCTION , PICTURE sau pe amandoua. Deoarece clauza FUNCTION afecteaza intreaga expresie ea nu poate contine decat coduri FUNCTION.

Semnificatia principalelor coduri FUNCTION este data in tabelul de mai jos

Coduri FUNCTION pentru afisare

Coduri FUNCTION	Semnificatie
B	Aliniaza la stanga date numerice in interiorul regiunii de afisare.
C	CR(Carriage Return) este afisat dupa un numar pozitiv pentru a indica un credit. Poate fi folosi numai cu date numerice.
D	Foloseste formatul de data calendaristica curent, stabilit prin comanda SET DATE
E	Editeaza date calendaristice cu formatul BRITISH.
T	Elimina blancurile de la inceputul si sfarsitul <expr>.
X	Este afisat DB dupa un numar negativ pentru a indica un debit. Poate fi folosi numai cu date numerice.
Z	<expr> sunt afisate numai blancuri daca valoarea campului este 0. Se foloseste numai cu date de tip numeric.
(Pune numerele negative intre paranteze. Se foloseste numai cu date de tip numeric.
!	Converteste caracterele alfabetice la litere majuscule. Se foloseste numai cu date de tip caracter.
^	Afiseaza datele numerice folosind scrierea stiintifica. Se foloseste numai cu date de tip numeric.
\$	Afiseaza datele in format currency (tip curs valutar) Simbolul stabilit poate apare inainte sau dupa expresia numerica asa cum a este setat prin comanda SET CURRENCY. Se foloseste numai cu date de tip numeric.

O expresie PICTURE poate include orice caractere dar doar cele specificate mai jos participa activ la afisarea sau tiparirea datelor.

Coduri PICTURE pentru afisare

Coduri	Semnificatie
--------	--------------

PICTURE	
A	Permite introducerea numai a caracterelor alfabetice.
W	Permite doar date de tip logic
N	Permite doar litere si cifre
X	Permite orice caractere
Y	Permite doar caracterele Y, y, N si n. Converteste y si n in Y si N.
.	Specifica pozitia punctului zecimal.
,	Separa cifrele la stanga punctului zecimal
*	Este afisat asteriscul in fata unei valori numerice. Se foloseste cu simbolul \$.

Exemplu de mai jos combina doua coduri FUNCTION pentru a formata o valoare numerica.

```
CLEAR
@ 2, 2 SAY 1.15 FUNCTION '^C'
0.6666E+0 CR
```

SIZE <expN1>, <expN2> -este o clauza care controleaza lungimea si inaltimea regiunii de afisare a iesirii comenzii @ ... SAY.

Implicit FoxPro creaza o regiune cu inaltimea de o linie. Incluzand clauza SIZE inaltimea regiunii in linii este specificata de <expN1>, iar lungimea in coloane este data de <expN2>. Fontul este o alta caracteristica care determina marimea regiunii de editare.

FONT <expC3> [, <expN3>]- specifica numele(tipul) fontului prin expresia tip sir de caractere <expC3> si dimensiunea fontului prin expresia numerica <expN3>. De exemplu, comanda de mai jos un text folosind fontul Times NewRoman si dimensiunea 16:

```
@ 2, 12 SAY 'Exemplu de folosire a clauzei FONT' FONT 'Times NewRoman ', 16
```

Exemplu de folosire a clauzei FONT

Omiterea dimensiunii fontului conduce la folosirea dimensiunii implicite de 10 pt. Daca fontul specificat nu este disponibil, Windows inlocuieste fontul cu unul ce are caracteristici asemanatoare.

STYLE <expC4>- specifica stilul prin<expC4>. Daca aceasta clauza nu apare atunci este folosit stilul standard de font.

Codificarea stilurilor este urmatoarea

Character	Stilul Fontului
B	Bold(ingrosat)
I	Italic(inclinat)
N	Normal
O	Outline
Q	Opac

S	Shadow(cu umbra)
-	Strikeout(cu o linie orizontala trasa)
T	Transparent
U	Subliniat

Poate fi inclus mai mult de un caracter pentru a specifica combinatia de stiluri dorite pentru font. De exemplu, comanda de mai jos specifica Opac Cu o linie orizontala trasa:

@ 2, 2 SAY 'Font clause example' STYLE 'Q-'

COLOR SCHEME <expN4>| COLOR <color pair list>- specifica schema de culori care va fi folosita pentru afisare. In lipsa unei clauze COLOR este folosita schema de culori a ferestrei VFP sau a celei definite de utilizator.

Numai prima pereche de culori din schema de culori sau din lista de culori determina modul de afisare al iesirii comenzii @ ... SAY.

Culoare folosita pentru afisarea iesirii comenzii @ ... SAY poate fi specificata incluzand numarul unei scheme de culori existente in clauza COLOR SCHEME sau a unei multimi de perechi de culori din clauza COLOR.

O schema de culori este o multime de 10 perechi predefinite de culori. Perechile de culori din schema de culori pot fi schimbate cu comanda SET COLOR OF SCHEME.

O pereche de culori este reprezentata printr-o pereche de doua litere separate prin "/". Prima culoare specifica culoarea care apare peste fond iar cea de a doua culoarea fondului. Exemplu : R/W

O lista de culori si codurile corespunzatoare este data mai jos.

Culoare	Cod
Negru	N
Spatiu(Blank)	X
Albastru	B
Maro	GR
Cyan	BG
Verde	G
Inverse	I
Magenta	RB
Rosu	R
Alb	W
Galben	GR+
Subliniat	U
(Underlined)	

Example

@ 2, 2 SAY 'Acesta este verde peste galben' COLOR G/ GR+

@ 4, 2 SAY 'Aceasta este Schema de culori 16' COLOR SCHEME 16

@ 6, 2 SAY 'Acesta este alb peste rosu' COLOR W/R

XII.2 Citirea datelor

Comanda @ ... GET

Sintaxa: @ <linie, coloana> GET <memvar> | <field>
[FUNCTION <expC1>][PICTURE <expC2>]
[FONT <expC3> [, <expN1>]]
[STYLE <expC4>][DEFAULT <expr1>]
[ENABLE | DISABLE][MESSAGE <expC5>]
[[OPEN] WINDOW <nume-fer>]
[RANGE [<expr2>] [, <expr3>]]
[SIZE <expN2>, <expN3>][VALID <expL1> | <expN4>]
[ERROR <expC6>]] [WHEN <expL2>]
[COLOR SCHEME <expN5>| COLOR <color pair list>]

Efect: Creaza o regiune de editare pentru continutul unei variabile, unui tablou sau al unui camp.

Se foloseste comanda READ sau READ CYCLE pentru activarea regiunilor de editare a comenzii @ ... GET.

<linie, coloana> - are aceeasi semnificatie ca si in cazul comenzii SAY prezentata mai sus.

<memvar> | <field>-comanda @ ... GET creaza regiuni de editare pentru variabile sau elementele unui tablou specificate in <memvar> sau campul unei tabele specificat in <field>.

@ ... GET poate fi folosita si pentru crearea unei regiuni de editare pentru campuri memo.Editarea unui camp Memo a fost precizata la prezentarea acestui tip de data.

Observatie: O metoda mai buna de editare a campurilor Memo presupune folosirea comenzii @ ... EDIT in loc de @ ... GET. @ ... EDIT creaza o fereastră de editare a textului cu bara derulanta si este afisat continutul campului Memo atunci cand este folosita comanda @ ... EDIT.

FUNCTION <expC1> | PICTURE <expC2> -au aceeasi semnificatie ca si in cazul comenzii SAY prezentata mai sus.

Coduri FUNCTION pentru editare

Coduri FUNCTION	Semnificatie
A	Permite citirea numai a caracterelor alfabetice (fara spatii sau simboluri)
B	Aliniaza la stanga date numerice in interiorul regiunii de editare.
C	CR(Carriage Return) este afisat dupa un numar pozitiv pentru a indica un credit. Poate fi folosi numai cu date numerice.
D	Foloseste formatul de data calendaristica curent, stabilit prin comanda SET DATE

E	Editeaza date calendaristice cu formatul BRITISH.
I	Centreaza textul in campul de editare.
J	Aliniaza textul la dreapta in camp.
K	Selecteaza intregul camp de editare cand cursorul se muta in acest camp.
L	Afiseaza zerouri in fata punctului zecimal(in locul spatiilor) in iesirile numerice. Se foloseste numai pentru date numerice.
M <lista>	Permite selectarea unui element al listei ca valoare pentru variabila citita cu comanda GET. Elementele listei sunt separate prin virgula si pot fi parcurse (in timpul citirii campului respectiv) cu tasta Space sau tastandu-se prima litera a elementului de selectat. Terminarea editarii campului este semnalata prin apasarea tastei Enter. Se foloseste doar cu comanda GET.
S(n)	Limiteaza latimea de afisare la n caractere, chiar daca latimea campului de editat este mai mare. In acest caz prin deplasarea cursorului in campul de editat se pot vedea si portiuni ascunse ale acestuia.
T	Elimina blancurile de la inceputul si sfarsitul <expr>.
Z	<expr> sunt afisate numai blancuri daca valoarea campului este 0. Se foloseste numai cu date de tip numeric.
,	Afiseaza date numerice folosind scrierea stiintifica. Se foloseste numai cu date de tip numeric.
\$	Afiseaza datele in format moneda(si in functie de forma comenzii SET CURRENCY.) Simbolul stabilit poate apare inainte sau dupa expresia numerica asa cum a este setat prin comanda SET CURRENCY. Se foloseste numai cu date de tip numeric.

Coduri PICTURE pentru editare

Coduri PICTURE	Semnificatie
A	Permite introducerea numai a caracterelor alfabetice.
L	Permite doar date de tip logic
N	Permite doar litere si cifre
X	Permite orice caractere
Y	Permite doar caracterele Y, y, N si n. Converteste y si n in Y si N.
9	Cu date de tip sir de caractere permite numai cifre iar cu date numerice permite folosirea doar a cifrelor si semnelor.
#	Permite cifre, blancuri si semne.

!	Toate caracterele alfabetice sunt trecute la majuscule.
.	Specifica pozitia punctului zecimal.
,	Separa cifrele la stanga punctului zecimal
*	Este afisat asteriscul in fata unei valori numerice. Se foloseste cu simbolul \$.

FONT <expC3> [, <expN1>], STYLE <expC4> - au aceeasi semnificatie ca si in cazul comenzii SAY.

DEFAULT <expr1>-daca nu este specificata o variabila de memorie pentru regiunea de editare a comenzii @ ... GET atunci o astfel de variabila este creata si initializata automat daca este inclusa clauza DEFAULT. (Acest lucru nu se intampla si in cazul in care in regiunea de editare se doreste citirea elementului unui tablou)

Observatie:

Daca nu este inclusa clauza DEFAULT si variabila de memorie nu exista, este afisat mesajul de eroare "Variable not found",

Expresia <expr1> determina tipul de variabila creata si valoarea sa initiala.

ENABLE | DISABLE-este clauza prin care se controleaza posibilitatea de a modifica sau nu un camp GET.Implicit un camp de editare @ ... GET se poate modifica.

MESSAGE <expC5>-permite afisarea <expC5> cand este selectata regiunea de editare @ ... GET. Afisarea se face pe ultima linie a ecranului, iar cand este specificata clauza SET MESSAGE TO <expN>, mesajul este afisat pe linia al carei numar este cel rezultata in urma evaluarii <expN>.

[OPEN] WINDOW < nume-fer>- permite editarea unui camp Memo intr-o ferestra definita de utilizator anterior. Daca este inclusa si clauza OPEN atunci fereastra definite de utilizator este deschisa implicit la executarea comenzilor READ sau READ CYCLE.

RANGE [<expr2>] [, <expr3>] -valideaza <memvar> astfel incat valorile introduse sa se incadreze in limitele specificate de <expr2>, <expr3>. <expr2> specifica limita inferioara iar <expr3> limita superioara. Poate lipsi oricare dintre ele, dar nu amndoua. Daca una lipseste se face doar o verificare (nu doua ca in cazul prezentei ambelor limite).

Observatie:

Domeniul de valori nu este verificat daca este apasata tasta Enter fara schimbarea valorii variabilei, elementului de tablou sau a campului.

SIZE <expN2>, <expN3>- are aceeasi semnificatie ca si in cazul comenzii SAY

VALID <expL1> | <expN4> -permite validarea valorii introduse conform <expL1> sau <expN4>. Daca <expL1> ia valoarea de adevar true (.T.), intrarea este considerata corecta si se paraseste regiunea de editare. Daca ia valoarea de adevar false (.F.), valoarea introdusa este considerata incorecta si este afisat un mesaj care iti indica faptul ca apasand tasta Spacebar se poate incerca introducerea din nou a datelor. Prin

intermediul <expN4> clauza VALID specifica ce obiecte sunt activate la iesirea din regiunea de editare GET .

ERROR <expC6>-afiseaza un mesaj de eroare daca la clauza VALID conditia este falsa. In lipsa acestui mesaj apare un mesaj standard al VFP.

WHEN <expL2> -stabileste o conditie (un filtru de selectie). Daca <expL2> este evaluata adevarat (.T.) atunci accesul la campul de sditare GET este permis. Altfel accesul la campul de editare nu este permis si se trece la campul urmator.

COLOR SCHEME <expN5>| COLOR <color pair list>- au aceeasi semnificatie ca si in cazul comenzii SAY.

Exemplul de mai jos citeste valorile campurilor cod_mat, denumire si unit_mas ale tabelii MATERIAL, folosita in sectiunea dedicata relationarii tabelelor, si le introduce la sfarsitul tabelii:

```
Close all
use material
clear
cod_mat1=1
denumire1="      "
unit_mas1="      "

@ 3, 13 SAY 'Codmaterial: ' GET cod_mat1 PICTURE '99999' COLOR gr+/b, r/w
@ 5, 13 SAY 'Denumire: '   GET denumire1 COLOR gr+/b, r/w
@ 7, 13 SAY 'Unitate de masura: ' GET unit_mas1 FUNCTION '!' COLOR gr+/b, r/w
READ
Append blank
REPLACE cod_mat WITH cod_mat1;
        denumire WITH denumire1 ;
        unit_mas WITH unit_mas1
GOTO BOTTOM
DISPLAY
Acelasi efect se obtine daca folosim secventa de instructiuni
Close all
use material
clear
APPEND BLANK
@ 3, 13 SAY 'Codmaterial: ' GET cod_mat1 PICTURE '99999' COLOR gr+/b, r/w
@ 5, 13 SAY 'Denumire: '   GET denumire1 COLOR gr+/b, r/w
@ 7, 13 SAY 'Unitate de masura: ' GET unit_mas1 FUNCTION '!' COLOR gr+/b, r/w
READ
GOTO BOTTOM
DISPLAY
```

Pentru modificarea inregistrării “n” a tabelii putem folosi programul

Clear

```
@ 3, 13 SAY 'Se modifica inregistrarea : ' GET n DEFAULT 1;
```

```
PICTURE '99999' COLOR gr+/b, r/w
```

```
READ
```

```
use material
```

```
GOTO n
```

```
@ 5, 13 SAY 'Codmaterial: ' GET cod_mat PICTURE '99999' COLOR gr+/b, r/w
```

```
@ 7, 13 SAY 'Denumire: ' GET denumire COLOR gr+/b, r/w
```

```
@ 9, 13 SAY 'Unitate de masura: ' GET unit_mas FUNCTION '!' COLOR gr+/b, r/w
```

```
READ
```

```
DISPLAY
```

Comanda READ

Sintaxa: READ[CYCLE][ACTIVATE <expL1>][DEACTIVATE <expL2>]
[MODAL][WITH < lista de titluri de ferestre>]
[SHOW <expL3>][VALID <expL4 | expN1>]
[WHEN <expL5>][OBJECT <expN2>]
[TIMEOUT <expN3>][SAVE][NOMOUSE][LOCK | NOLOCK]
[COLOR <color pair list>| COLOR SCHEME <expN4>]

Efect: Activeaza obiectele create cu comenzile @ ... GET sau @ ... EDIT. Sunt activate toate obiectele create dupa ultima comanda READ sau CLEAR GETS. Pe tot parcursul introducerii datelor in campurile de editare rulara programului stationeaza in derptul comenzii READ. Din aceasta se iese

-cand au fost completate toate campurile (deplasand cursorul dupa ultimul obiect creat prin GET)

-deplasand cursorul inaintea primului obiect creat cu GET(daca nu este folosita comanda READ CYCLE)

-fortat apasand tasta [Esc] sau [Ctrl+W] sau

selectand o optiune definita special pentru parasirea unei comenzi READ.

Un READ pentru mai multe ferestre- pot fi plasate obiecte in diferite ferestre and pot fi activate toate cu un singur READ. Obiectele sunt selectate in ordinea in care au fost create(oferestra este activata daca obiectul din interiorul ei devine curent). Intr-o fereastră, apasarea tastelor [Tab], [Enter] sau [↓] atunci cand cursorul este pozitionat pe ultimul obiect din fereastră face ca acesta sa se pozitioneze pe primul obiect din fereastră urmatoare. Daca cursorul este pozitionat pe primul obiect dintr-o fereastră atunci apasarea tastelor [Shift+Tab] sau [↑] face ca acesta sa se pozitioneze pe ultimul obiect din fereastră precedenta.

Comenzi READ incluse. Pot fi realizate utilizand un set de comenzi GET si un READ, intr-o procedura apelata atunci cand se executa un alt READ. Comenzile READ pot fi incluse una in interiorul alteia pana la maxim 5 niveluri.

CYCLE-Includerea acestei clauze face ca comanda READ sa nu se termine daca cursorul este mutat inaintea primului obiect sau dupa ultimul obiect definit de comanda GET. In aceste situatii cursorul se muta de fapt pe ultimul obiect, respectiv primul obiect definit de comanda GET. Iesirea dintr-o comanda READ in care apare clauza CYCLE se face fie apasand tastele [Escape] sau [Ctrl+W], fie folosind comanda CLEAR READ sau incluzand clauza TIMEOUT in comanda READ.

ACTIVATE <expL1> -Expresia logica <expL1> este in mod obisnuit o functie definita de utilizator. Functia WOUTPUT() poate fi folosita in functia definita de utilizator pentru a determina care fereastră este activata. Aceasta functie definita de utilizator poate dezactiva obiecte @ ... GET din alte ferestre, ascunde ferestre, afisa mesaje etc.

DEACTIVATE <expL2>-este o clauza care se executa atunci cand se doreste trecerea la alte ferestre.. DEACTIVATE poate fi privita ca o clauza VALID la nivel de fereastră. <expL2> este o functie definita de utilizator ce poate fi folosita pentru a valida continutul campurilor dintr-o fereastră inainte de a se activa o alta fereastră. Daca functia definita de utilizator returneaza valoarea fals (.F.), nu se termina executia comenzii READ, altfel executia se incheie.

MODAL

MODAL –este un cuvânt cheie care in timpul executiei comenzii READ previne activare tuturor ferestrelor cu exceptia celor implicate in executia comenzii READ.
[WITH < lista de titluri de ferestre>]- permite ca ferestre care nu sunt implicate in executarea comenzii READ sa devina active in timpul derularii comenzii.

SHOW <expL3> este executata dor in prezenta comenzii SHOW GETS . Valoarea returnata de o procedura SHOW este ignorata, procedura fiind utila numai pentru reexecutarea comenzii @ ... SAYs sau pentru activarea/dezactivarea obiectelor GET.

VALID <expL4> | <expN1> Realizeaza validarea valorilor introduse in momentul in care se doreste parasirea comenzii READ. Comanda READ se termina daca <expL4> ia valoarea de adevar true (.T.) iar daca <expL4> ia valoarea false (.F.), cursorul ramane pozitionat pe acelasi obiect GET. Daca clauza VALID returneaza un numar atunci cursorul se muta pe obiectul cu numarul respectiv. Daca obiectul nu exista atunci comanda READ se termina. Returnarea unei valori pentru expresia numerica care nu este de tip numeric are acelasi efect ca in cazul returnarii de catre expresia logica a valorii .T.

WHEN <expL5> - este o clauza ce permite executia comenzii READ daca <expL5> este adevarata (.T.). Daca este falsa comanda READ este ignorata iar programul continua cu executia instructiunii urmatoare.

OBJECT <expN2>-specifica ce obiect este selectat initial, atunci cand se executa comanda READ. <expN2> determina care obiect este selectat initial. Numarul obiectului este dat de ordinea in care acestea au fost create cu comanda GET.

In exemplul de mai jos sunt create un camp si patru butoane radio. Pentru ca primul selectat sa fie primul buton am inclus clauza OBJECT 2.

```

STORE 1 TO calif
STORE SPACE(10) TO numele
CLEAR
@ 2,25 SAY 'Introduceti numele: ' GET numele
@ 4,25 GET calif PICTURE '@*R Foarte Bine;Bine;Satisfacator;Nesatisfacator'
READ CYCLE OBJECT 2

```

TIMEOUT <expN3>-este o clauza prin care se stabileste perioada de timp in care comanda READ asteapta sa fie executata. <expN3> reprezinta numarul de secunde pe care sistemul le asteapta pentru completarea comenzilor GET. Dupa aceasta perioada este lansata automat in executie comanda READ. Daca comanda READ este terminata datorita clauzei TIMEOUT, atunci functia READKEY() returneaza 20 daca nu au fost efectuate modificari asupra nici unui obiect. Daca au avut loc schimbari atunci ea returneaza valoarea 276.

SAVE-permite ca dupa incheierea unui READ valorile introduse sa nu fie sterse de pe ecran.

NOMOUSE-nu permite ca in timpul executarii comenzii READ obiectele sa fie selectate cu mouse-ul ci numai cu tastatura.

LOCK | NOLOCK -specifica daca inregistrările utilizate in comenzi Get sunt in mod automat blocate in varianta pentru retea a sistemului FoxPro.

COLOR <color pair list> si COLOR SCHEME <expN4>-au semnificatia asemanatoare celei prezentate la comanda @ ... SAY.

XII.3 OBIECTE DE CONTROL

Obiectele de control reprezinta elemente ale interfeței cu utilizatorul a unui program(sistem informatic). Astfel de obiecte de control sunt campurile de editare cu butoane de incrementare/decrementare, listele de diferite tipuri, butoanele cu selectie multipla(radio), casutele de selectie(comutatoarele) etc.

In cele ce urmeaza vom prezenta cateva dintre cele mai folosite obiecte de control in FoxPro. Toate aceste obiecte sunt definite cu ajutorul comenzii GET si citite cu comanda READ. Tipul obiectului de control este specificat cu ajutorul codurilor FUNCTION sau PICTURE corespunzatoare.

1. Campuri de editare

Campurile de editare simple sunt cele definite cu ajutorul functiei GET fara vreo clauza sau cod FUNCTION /PICTURE specific.

Pot fi create si campuri de editare mai complexe care sunt insotite in partea dreapta de doua butoane (sageți directionale [↑], [↓]) cu ajutorul carora se poate realiza incrementarea (se apasa sageata orientata in sus) sau decrementarea (se apasa sageata

orientata in jos) valorii existente in camp. Valoarea cu care se modifica (incrementeaza, respectiv decrementeaza) valoarea variabilei citite este prestabilita.

Comanda GET prin care sunt implementate butoanele de incrementare/decrementare contine clauza SPINNER care este urmata de trei valori ce reprezinta : incrementul, limita inferioara si cea superioara permisa pentru valoarea citita.

Exemplu:

```
@ 12, 23 SAY 'Introduceti un numar multiplu de 3';  
GET a DEFAULT 3 PICTURE '99999';  
SPINNER 3, 3, 99999
```

READ

Introduceti un numar multiplu de 3

Sintaxa generala a comenzii este:

```
@ <linie, coloana> GET <var> | <camp>  
SPINNER <expN1> [, <expN2> [, <expN3>]]  
    [FUNCTION <expC1>][PICTURE <expC2>]  
    [FONT <expC3> [, <expN4>]][STYLE <expC4>]  
    [DEFAULT <expN5>][SIZE <expN6>, <expN7>]  
    [ENABLE | DISABLE][MESSAGE <expC5>]  
    [RANGE [<expN8>] [, <expN9>]][VALID <expL1> | <expN10>  
    [ERROR <expC6>]][WHEN <expL2>][COLOR SCHEME <expN11>  
    | COLOR <color pair list>]
```

Vom descrie mai jos numai clauzele care nu apar in comanda GET care a fost prezentata in sectiunile precedente.

SPINNER <expN1> [, <expN2> [, <expN3>]]- Valoarea din caseta text este incrementata sau decrementata cu o valoare egala <expN1>. Valoarea minima acceptata este <expN2>, iar cea maxima <expN3>. Aceste valori intra sub incidenta clauzelor VALID si RANGE.

FUNCTION <expC1> | PICTURE <expC2>-aceste clauze controleaza editarea sau afisarea campului de editare. Codurile FUNCTION valabile in acest caz sunt B, I, J, K, L, ^, \$ iar codurile PICTURE admise sunt 9, #, \$, *, . si , (virgula). Semnificatia acestor coduri este data in tabelele prezentate pentru comenzile GET si SAY.

2. Butoanele simple sau declansatoarele

-Sunt foarte des intalnite si sunt folosite atunci cand este necesara declansarea unei anumite operatii la comanda utilizatorului. Pentru a preciza ca o comanda GET defineste

un buton, in clauza FUNCTION a sa este introdus caracterul '*' (sau codul PICTURE echivalent '@*').

Variabila in care se realizeaza citirea trebuie sa fie *numerica* sau sir de *caractere*. In primul caz este returnat (citit) *numarul de ordine al butonului*, iar in al doilea caz *textul care este asociat butonului* (care este specificat in clauzele FUNCTION sau PICTURE dupa codul * si care este cunoscut sub numele de **prompt**).

Plasarea in fata numelui butonului a caracterelor '?' echivaleaza tastei [Esc] si este permis pentru un READ.

Se pot crea mai multe (un grup de) butoane cu aceeasi comanda GET incluzandu-se in clauza FUNCTION (sau PICTURE) prompt-urile lor separate prin ";". Dimensiunea butonului se specifica cu ajutorul clauzei SIZE a comenzii GET.

Exemplu:

```
@ 12, 23 GET a DEFAULT 3 PICTURE '@* Salvare; Iesire; Stergere; \?ESC'
READ
```



3. Comutatoarele(check boxes)

-sunt folosite pentru citirea datelor care nu pot lua decat doua valori: adevarat (.T.) sau fals (.F.); Da sau Nu; ON sau OFF etc.

Crearea lor se realizeaza tot cu comanda GET in care codul FUNCTION este '*C' (sau codul '@*C' in cazul clauzei PICTURE. Comutatorul are asociat un text explicativ care prezinta intr-un mod sugestiv semnificatia sa. Variabila in care se realizeaza citirea trebuie sa fie de tip numeric (0-neactivat si diferit de 0 activat) sau logic (fals pentru neactivat si adevarat pentru activat).

Dimensiunea unui comutator este precizata cu ajutorul clauzei SIZE a comenzii GET.

Exemplu:

```
@ 12, 23 GET a DEFAULT 0 PICTURE '@*C DA'
READ
```



4. Listele

-ofera utilizatorului posibilitatea alegerii unui element dintr-o multime finita de elemente de acelasi tip afisate pe ecran total sau partial. Listele care sunt afisate total pe ecran sunt **listele simple** in timp ce listele afisate partial se mai numesc si **liste ascunse**.

Listele ascunse afiseaza pe ecran un singur element, cel selectat curent. Cand utilizatorul doreste sa selecteze un alt element lista este desfasurata si utilizatorul are acces la toate elementele sale.

Codul FUNCTION pentru lista simpla este '^' iar pentru lista ascunsa este '^'. Elementele listei pot fi preluate dintr-un tablou (daca se foloseste clauza FROM <tablou>), dintr-un submeniu(caz in care se foloseste clauza POPUP <submeniu>) sau pur si simplu pot fi specificate direct in sirul clauzei FUNCTION folosind separatorul '^'.
Sintaxa:

```
@ <linie, coloana> GET <var> | <camp>
    FROM <tablou>[RANGE <expN1> [, <expN2>]]| POPUP <nume submeniu>
    [FUNCTION <expC1>] | [PICTURE <expC2>]
    [FONT <expC3> [, <expN3>]][STYLE <expC4>]
    [DEFAULT <expr>][SIZE <expN4>, <expN5> ]
    [ENABLE | DISABLE][MESSAGE <expC5>][VALID <expL1> | <expN6>]
    [WHEN <expL2>][COLOR SCHEME <expN7>| COLOR <color pair list>]
```

FROM <tablou> -lista se creeaza din tabloul <tablou>. Daca tablou este unu dimensional elementele din tablou vor apare in lista in ordinea din tablou. Daca tabloul este bidimensional sunt folosite elementele primei coloane a tabloului pentru a obtine lista.

RANGE <expN1> [, <expN2>]- De obicei lista incepe cu primul element din tablou. Daca dorim sa desemnam un element diferit ca prim element din lista vom include clauza RANGE <expN1>. Atunci elementul cu numarul <expN1> va fi primul element din lista. Celelalte elemente sunt cele urmatoare acestuia in cazul unu dimensional, sau restul elementelor din coloana corespunzatoare primului element in cazul tablourilor bidimensionale.De exemplu in cazul tabloului de mai jos Clauza RANGE 5

```
a b c
d e f
g h i
```

are drept efect crearea unei liste care contine elementele e si h.(a se vedea sectiunea rezervata tablourilor pentru numararea elementelor unui tablou).

POPUP <nume submeniu>-creeaza lista preluand elementele submeniuului <nume submeniu>

Exemplu1:

```
CLEAR
SET TALK OFF
STORE 1 TO m
```

```
DEFINE POPUP subm FROM 0, 0 PROMPT FILES LIKE *.DBF ;
    MARGIN SCROLL
```

```
@ 2,2 GET m POPUP subm SIZE 8, 20
```

```
READ && Activeaza lista.
```

Exemplu2:

```
CLEAR
```

```

prnume=1
@ 2,22 GET prnume FUNCTION '^ eu; tu; el; ea; noi; voi; ei'
READ

```

5. Butoanele radio

Prin acesta comanda este creat un **buton radio**. Aceste sunt folosite atunci se doreste punerea la dispozitia utilizatorului a unui anumit numar de optiuni, care sa incapa pe ecran si din care el sa aleaga una prin selectare. Optiunile butoanelor radio se exclud reciproc.

Butoanele radio se definesc in grupuri prin utilizarea in clauza FUNCTION a codului '*R'. Acest cod este urmat de textele asociate butoanelor respective separate prin ';'. Ca si in cazul butoanelor simple, daca variabila in care se realizeaza citirea este *numar* atunci in urma selectiei ie l se atribuie numarul de ordine al butonului, iar daca variabila este sir de *caractere* atunci ea returneaza *textul care este asociat butonului*.

Exemplu:

```

CLEAR
a=' '
@ 2,22 GET a FUNCTION '*R Foarte Bine;Bine;Satisfacator;Nesatisfacator'
READ
?a
☒ Foarte Bine
☐ Bine
☐ Satisfacator
☐ Nesatisfacator

```

6. Regiuni pentru editarea textelor

-Sunt zone de pe ecran unde in care utilizatorul poate introduce un text de dimensiuni mari. Pentru definirea regiunilor de editare a textelor este folosita comanda EDIT a carei sintaxa generala este data mai jos. Clauzele acestei comenzi nu vor mai fi detaliate ele fiind asemanatoare celor de la comanda GET.

Sintaxa:

```

@ <row, column> EDIT <memvar> | <field>
    SIZE <expN1>, <expN2> [, <expN3>]
    [FUNCTION <expC1>][FONT <expC2> [, <expN4>]]
    [STYLE <expC3>][DEFAULT <expr>]
    [ENABLE | DISABLE][MESSAGE <expC4>]
    [VALID <expL1> | <expN5> [ERROR <expC5>]]
    [WHEN <expL2>][NOMODIFY][SCROLL][TAB]
    [COLOR SCHEME <expN6>| COLOR <color pair list>]

```

Clauzele SCROLL si TAB nu apr in comanda GET. Ele au semnificatia de mai jos

SCROLL- va afisa pe latura dreapta a zonei de editare bara verticala de defilare(scroll bar).

TAB – Permite inserarea unui **Tab** la apasarea tastei [Tab]. De asemenea, combinatia [Ctrl+Tab] va permite salvarea modificarilor facute si inchiderea zonei de editare.

XIII. AFISAREA SECVENTIALA A DATELOR

Comanda ?|??

Sintaxa:

```
?|?? <expr1>  
    [PICTURE <expC1>] | [FUNCTION <expC2>] | [V<expN1>]  
    [AT <coloana>]  
    [FONT <expC3> [,<expN2>] [STYLE <expC4> ] [,<exp2>]  
    [,<exp3>] ...
```

Efect: Evalueaza expresii si afiseaza rezultatul.

?<expr1>-<expr1> este expresia ce va fi evaluata. Aceasta va fi afisata si este inserat un CR (carriage return) si LF (linefeed)(adica este mutat cursorul la inceputul liniei urmatoare) inainte de rezultat. Rezultatele sunt afisate pe urmatoareal linie a ferestrei principale VFP sau in fereastra definita de utilizator activa si tiparite incepand de la marginea stanga a paginii daca nu este specificat altceva.

In lipsa <expr1> ester afisata sau tiparita o linie goala. Este inclus un spatiu intre valorile expresiilor multiple.

?? <expr1>-are acelasi efect ca si ? <expr1> cu exceptia faptului ca rezultatul este afisat pe linia curenta si la pozitia curenta in ferestrei principale VFP sau in fereastra definita de utilizator activa si nu mai este inserat CR si LF inainte de rezultat.

Cele doua comenzi permit fie afisarea datelor pe ecran fie la imprimanta. Alegerea dispozitivului destinatie se face cu ajutorul comenzilor SET CONSOLE si SET PRINTER. Daca SET CONSOLE este OFF si SET PRINTER ON iesirea este directionata numai spre imprimanta. Daca SET PRINTER OFF atunci iesirea este directionata numai spre ecran. Daca SET PRINTER este ON atunci iesirea este directionata si spre imprimanta si spre ecran.

[PICTURE <expC1>] | [FUNCTION <expC2>] - specifica formatul in care vor fi afisate rezultatele, asa cum a fost prezentat la descrierea comenzii SAY. <expC1> este precedat de @ urmat de un spatiu si apoi de coduri format.

V<expN> - Specifica un cod special functie care permite unei expresii cu valoarea sir de caractere sa fie srisa într-o regiune verticala formata dintr-un anumit numar de coloane (<expN1>).

```
? 'Acesta este un exemplu care arata cum functioneaza codul function V.' ;  
    FUNCTION 'V16'
```

AT <coloana>- Specifica numarul coloanei de la care este afisat rezultatul expresiei.

FONT <expC3> [,<expN2>] , STYLE <expC4> - Specifica fontul si stilul folosite pentru afisare (vezi comanda @...SAY)

Exemplu:

? DATE() FONT 'Courier',16

Comanda ???

Sintaxa: ???<expC1>

Efect: Evalueaza expresia <expC1> si tipareste la imprimanta rezultatul.

Comanda \|

Sintaxa:

\<expC1>

sau

\\<expC1>

Efect: este afisat textul <expC1>. Textul afisat nu trebui pus intre ghilimele, comenzile de mai sus afisand tot ceea ce apare dupa ele. Cele doua comenzi difera intre ele prin faptul ca "\<" realizeaza un salt la linie noua inainte de afisare a textului in timp ce comanda "\\<" nu realizeaza acest lucru.

Daca in textul afisat se doreste introducerea unor expresii aceste trebuie introduse intre delimitatorii "<<" si ">>". Locul de afisare a textului, ecran sau fisier si decizia daca evaluarea expresiilor are loc sau nu, sunt controlate de comanda SET TEXTMERGE. Pentru ca textul marcat de delimitatorii "<<" si ">>" sa fie evaluat ca o expresie trebuie introdusa comanda SET TEXTMERGE ON(aceasta fiind si starea implicita). Daca este introdusa comanda SET TEXTMERGE OFF atunci textul cuprins intre delimitatori nu va mai fi evalua, el fiind afisat asa cum apare. Optiunea To a comenzii SET TEXTMERGE permite specificare fisierului unde vor fi afisate textele cu comenzile "\<"si "\\<".

XIV. Ferestre

Ferestrele reprezinta zone ale ecranului virtual al unui sistem tratate ca un tot unitar. Ferestrele au anumite caracteristici ce determina operatiile ce se pot executa asupra lor. Iesirile unui SGBD pot fi directionate la un moment dat fie spre ecran (fundalul) fie spre o fereastră definita de utilizator. Aceasta fereastră va fi numita fereastră activa. Ferestrele sunt definite cu comanda DEFINE WINDOW si activate cu comanda ACTIVATE WINDOW. Afisarea ferestrelor (fara activarea lor) se realizeaza cu comanda SHOW WINDOW iar eliminarea lor de pe ecran (fara eliminarea lor din memorie) este realizata cu comanda HIDE WINDOW. Dezactivarea unei ferestre se face cu comanda DEACTIVATE WINDOW iar stergerea lor din memorie cu RELEASE WINDOWS. Comanda DEFINE WINDOW defineste o fereastră in VFP, adica atribuie ferestrei o zona de memorie folosita la gestiunea ferestrei si un nume si stabileste attributele acesteia. Aceasta comanda este prezentata mai jos.

Comanda DEFINE WINDOW

Sintaxa:

```

DEFINE WINDOW <nume-fer1> FROM <linie1, coloana1> TO <linie2, coloana2>
  | AT <linie3, coloana3> SIZE <linie4, coloana4>
  | IN [WINDOW] <nume-fer2> | IN SCREEN | IN DESKTOP]
  | [FONT <expC1> [, <expN1>]][STYLE <expC2>][FOOTER <expC3>]
  | [TITLE <expC4>][HALFHEIGHT][DOUBLE | PANEL | NONE
  | SYSTEM | <caractere de bordura>][CLOSE | NOCLOSE][FLOAT | NOFLOAT]
  | [GROW | NOGROW][MDI | NOMDI][MINIMIZE | NOMINIMIZE]
  | [ICON FILE <numefis1>][FILL <expC5> | FILL FILE <numefis2>]
  | [COLOR SCHEME <expN2> | COLOR <lista palete coloristice>]

```

Efect: Creeaza o fereastră definită de utilizator și specifică atributele ei. După ce a fost creată aceasta poate fi afișată în fereastră principală Visual FoxPro cu comenzile **ACTIVATE WINDOW** sau **SHOW WINDOW**. Numărul de ferestre care poate fi creat este limitat doar de memoria disponibilă și de resursele sistemului.

Numele unei ferestre este afișat la sfârșitul listei de opțiuni a submeniului **Window** în ordinea în care au fost definite.

În momentul activării unei ferestre toate ieșirile sunt direcționate în interiorul acesteia și nu pe ecran așa cum se întâmplă de obicei. Numele ferestrei spre care sunt direcționate ieșirile la un moment dat este marcat în submeniul **Windows** prin simbolul ♦.

Fereăstrele rămân activate pe ecran până la executarea unei comenzi **DEACTIVATE WINDOW** sau **HIDE WINDOW**. Aceste comenzi înlătură fereăstrele de pe ecran dar nu și din memorie. O fereastră poate fi reactivată pe ecran cu comenzile **ACTIVATE WINDOW** sau **SHOW WINDOW**.

Pentru a elimina ferestrele din memorie (și de pe ecran) se folosesc comenzile **CLEAR WINDOWS** sau **RELEASE WINDOWS**. Ferestrele care au fost înlăturate din memorie trebuie redefinite și reactivate pentru a fi reafisate..

< nume-fer1> -reprezintă numele ferestrei ce va fi creată. Numele ferestrei nu poate să depășească 10 caractere lungime și trebuie să înceapă cu o literă sau caracterul '_'.

FROM <linie1, coloana1> TO <linie2, coloana2>-reprezintă coordonatele colțului stâng superior și respectiv drept inferior. Aceste două puncte sunt suficiente pentru a determina zona dreptunghiulară pe care o reprezintă fereastră. Coordonatele ferestrei pot depăși limitele ecranului. O fereastră definită de utilizator poate fi plasată în interiorul altei ferestre caz în care coordonatele ferestrei se referă la numărul de linii și coloane ale ferestrei părinte și nu la cele ale ecranului.

AT <linie3, coloana3>

SIZE <linie4, coloana4> - sunt clauze care determină dimensiunea unei ferestre ținând cont de dimensiunea fontului utilizat. Fontul și stilul ferestrei pot fi specificate incluzând clauzele **FONT** și **STYLE**. Dacă nu este specificat un font pentru fereastră și este inclusă clauza **SIZE**, dimensiunea ei este determinată relativ la fontul implicit al sistemului - FoxFont 10 pt.

The location of the upper-left corner of the fereastră you create is specified with Clauza AT <linie3, coloana3> specifică coordonatele colțului stanga sus al ferestrei.

Locatia lui este determinatade contul curent al fereastrei "parinte". Deoarece clauza AT este identica cu clauza FROM cele doua clauze sunt interschimbabile.

Clauza SIZE poate fi folosita in locul clauzei TO. Dimensiunea ferestrei definite de utilizator in linie si coloane este specificata prin <linie4, coloana4>.

IN [WINDOW] <nume-fer2> - este o clauza folosita pentru a plasa ferestra definita de utilizator in fereastra parinte cu numele <nume-fer2>.

Exemplu:

```
CLEAR
DEFINE WINDOW Fer1 ;
    FROM 12, 12 TO 33, 34 ;
    TITLE "Fer1" ;
    CLOSE FLOAT ZOOM MINIMIZE                && Fereastra
parinte.
ACTIVATE WINDOW Fer1
DEFINE WINDOW Fer2 ;
    FROM 1, 2 TO 18, 20 ;
    TITLE "Fer2"
    IN WINDOW Fer1;
    CLOSE FLOAT ZOOM MINIMIZE && Fereastra copil.

ACTIVATE WINDOW Fer2
RELEASE WINDOW Fer1, Fer2
CLEAR
```

IN SCREEN- este o clauza care plaseaza explicit ferestra pe ecran, nu in interiorul altei ferestre.

IN DESKTOP-este folosita pentru a plasa fereastra pe desktop-ul Windows-ului adica in afara ferestrei principale VFP.

FONT <expC1> [, <expN1>], STYLE <expC2> - precizeaza fontul si stilul pentru iesirile directionate spre ferestra.

FOOTER <expC3>- este o clauza care in FoxPro sub MS-DOS, pune sirul de caractere <expC3> pe latura inferioara a conturului ferestrei. Aceasta clauza este ignorata in VFP.

TITLE <expC4>- Specifica titlul <expC4> al ferestrei.

HALFHEIGHT- creeaza ferestre a caror bara superioara este 3/2 din inaltimea unei ferestre normale. A fost introdus pentru a compatibiliza ferestre create cu versiuni mai vechi de FoxPro cu ferestrele VFP, care au aceasta proprietate implicit.

DOUBLE | PANEL | NONE | SYSTEM | < caractere de bordura> - permite inluderea unor caractere pentru bordura (implicit bordura este o singura linie). Optiunea DOUBLE produce o linie dubla pentru bordura, optiunea PANEL produce un contur cu bara luminoasa iar optiunea NONE elimina conturul in intregime.

Includerea optiunii SYSTEM permite imprumutarea tuturor atributelor si caracteristicilor conturului unei ferestre VFP. Daca sunt incluse si CLOSE, FLOAT, ZOOM sau MINIMIZE atunci caracterele de control apar si ele pe conturul ferestrei. Poate fi creat si propriul model de bordura incluzand < caractere de bordura >. Pentru amanunte privind sintaxa sirului de bordura se poate consulta help-ul comenzii SET BORDER.

CLOSE | NOCLOSE- controleaza posibilitatea de a deschide sau inchide fereastra definta de utilizator. Inchiderea ferestrei duce la inlaturarea ei de pe ecran (sau din fereastra parinte) si din memorie. In VFP o fereatra poate fi inchisa apasand [Ctrl+F4] sau alegand optiunea Close a meniului File.

FLOAT | NOFLOAT -O fereastră poate fi mutata daca este inclusa optiunea FLOAT. In VFP o fereastră poate fi mutata alegand optiunea Move din meniul Control al ferestrei sau apasand Ctrl+F7.

[GROW | NOGROW]-controleaza posibilitatea de a controla dimensiunea ferestrei definite.

MDI | NOMDI- specifica sau nu posibilitatea de a crea in VFP o feresatra MDI (Multiple Document Interface)

MINIMIZE | NOMINIMIZE- controleaza posibilitatea de a reduce o fereastră la dimensiunea ei minima si de a o plasa in coltul drept (in partea de jos) al ecranului. Se include clauza NOMINIMIZE pentru a preveni minimizarea unei ferestre.

ZOOM | NOZOOM-- controleaza posibilitatea de a reduce o fereastră la dimensiunea maxima pe ecran sau in interiorul ferestrei parinte.

ICON FILE <numefis1> - se refera la fisierul "icon" cu numele <numefis1> ce contine o pictograma ce va fi afisata cand fereastra este minimizata. Poate fi specificat doar un fisier .ICO nu si unul bitmap (.BMP).

FILL <expC6>| FILL FILE <numefis2>

FILL <expC5> - permite (in versiunile FoxPro pentru MS-DOS) umplerea fondului unei ferestre cu un caracter <expC6>. Daca <expC6> contine mai mult de un caracter, numai primul va fi folosit pentru umplere. De asemenea poate fi specificat un caracter folosind CHR(). In VFP clauza FILL este ignorata.

FILL FILE <numefis2> creeaza in fereastra curenta un fundal (wallpaper sau background) pe baza unei imagini continute de fisierul <numefis2> (fisierul este .BMP). Clauza este valabila doar in FoxPro pentru Windows.

COLOR SCHEME <expN>| COLOR <lista palete coloristice> -permite specificarea unei palete de culori.

Comanda **ACTIVATE WINDOW**

Sintaxa:

```
ACTIVATE WINDOW [<nume-fer1>  
    [, <nume-fer2> ...]]  
    | ACTIVATE WINDOW ALL  
    [IN [WINDOW] <nume-fer3>  
    | SCREEN]  
    [BOTTOM | TOP | SAME]  
    [NOSHOW]
```

Efect: Afiseaza si activeaza una sau mai multe ferestre definite de utilizator sau ferestre ale sistemului FoxPro.

Activarea acestei ferestre face ca fereastra la care se refera comanda sa devina activa si toate iesirile sa fie redirectionate spre ea. Numele ferestrei active apare marcat in partea inferioara a meniului Windows.

Observatie: pentru a fi siguri ca este activa fereastra dorita la dezactivarea ferestrei curente se foloseste comanda **ACTIVATE WINDOW**.

<nume-fer1>

[, <nume-fer2> ...] - listeaza numele ferestrelor ce vor fi activate.

ALL –specifica faptul ca toate ferestrele vor fi activate. Ultima fereastra activata este fereastra activa pentru iesiri.

IN [WINDOW] <nume-fer3> - fereastra(ce se va numi copil) este plasata si activata in fereastra parinte specificata prin <nume-fer3>. O fereastra parinte poate avea mai multe ferestre copil. O fereastra copil activata in interiorul unei ferestre parinte nu poate fi mutata in afara ferestrei parinte.

IN [WINDOW] SCREEN-fereastra este activata pe desktop sau in fereastra principala FoxPro.

BOTTOM | TOP | SAME –pozitioneaza ferestrele activate, in raport cu alte ferestre deschise anterior, in zona inferioara, folosind **BOTTOM**, in zona superioara folosind **TOP** si la aceleasi coordonate cu aceasta daca este folosita clauza **SAME**.

NOSHOW-redirectioneaza iesirile spre fereastra specificata fara ca aceasta sa devina vizibila.

Comanda **HIDE WINDOW**

Sintaxa:

```
HIDE WINDOW <nume-fer1> [, <nume-fer2>] ... | ALL  
    [IN [WINDOW] <nume-ferN> | IN [WINDOW] SCREEN]  
    [BOTTOM | TOP | SAME]
```

Efect: Ascunde o fereastră sau o multime de ferestre de pe ecran sau dintr-o fereastră definită de utilizator. Ascunderea unei ferestre nu este același lucru cu închiderea ei. Ea rămâne în memorie și este poate fi afișată cu comenzile ACTIVATE WINDOW sau SHOW WINDOW.

<nume-fer1> [, <nume-fer2>...] -este lista ferestrelor ce vor fi ascunse

ALL-toate ferestrele vor fi ascunse. Restul clauzelor sunt asemănătoare celor descrise mai sus.

DEFINE WINDOW fer FROM 6,1 TO 19,75 TITLE 'FEREASTRA1';

CLOSE FLOAT GLINE SHADOW ZOOM

ACTIVATE WINDOW fer

WAIT WINDOW 'Apasa o tasta pentru a ascunde aceasta fereastră'

HIDE WINDOW fer

WAIT WINDOW 'Apasa o tasta pentru a vedea aceasta fereastră din nou'

SHOW WINDOW fer

WAIT WINDOW 'Apasa o tasta pentru a înlătura fereastră'

DEACTIVATE WINDOW fer

RELEASE WINDOW fer

Comanda DEACTIVATE WINDOW

Sintaxa:

DEACTIVATE WINDOW <nume-fer1> [, <nume-fer2>] ... | ALL

Efect: dezactivează și renunță la afișarea unei ferestre sau unei mulțimi de ferestre pe ecran, nu și din memorie.

Comanda RELEASE WINDOWS

Sintaxa:

RELEASE WINDOWS <nume-fer1> [, <nume-fer2>] ... | ALL

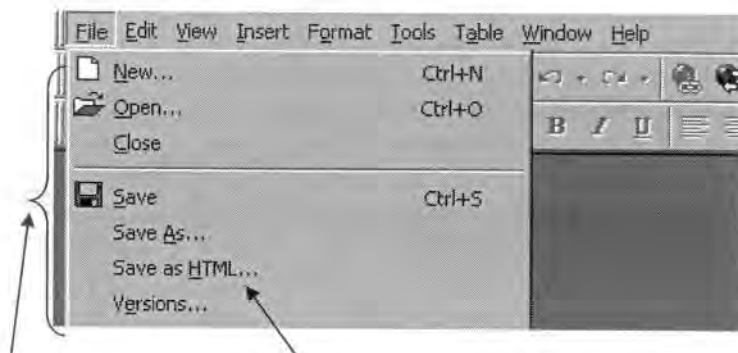
Efect: Înlătura din memorie o fereastră sau o multime de ferestre.

XV. MENIURI

Meniurile reprezinta elemente ale interfetei unui sistem informatic cu ajutorul carora utilizatorul initiaza efectuarea unor operatii cum ar fi pornirea diferitelor prelucrari de date, deschiderea unor ferestre de dialog etc.

Meniu bara

Optiune (bara) a meniului



Submeniu

Optiune de submeniu

O clasificare sumara a meniurilor ar fi **meniuri bara** (sau **meniuri orizontale**) care contin mai multe **optiuni de meniu**, fiecareia dintre acestea atasandu-i-se un **submeniu** (**meniu vertical**). Fiecare submeniu are la randul sau **optiuni**(de submeniu).

XV.1 Meniuri orizontale. Definitia meniurilor si a optiunilor de meniu

Definirea barei de meniu se realizeaza cu comanda DEFINE MENU.

Sintaxa: DEFINE MENU <nume-meniu>[BAR [AT LINE <expN1>]]
[IN [WINDOW] <nume-fer> | IN SCREEN][KEY <eticheta-taste >]
[MESSAGE <expC2>][NOMARGIN]
[COLOR SCHEME <expN2>| COLOR <lista- perechi-culori>]

Efect: Creeaza un meniu orizontal cu submeniuuri pentru fiecare optiune a sa. Optiunile sale de meniu (optiuni bara) vor fi create cu comanda DEFINE PAD si se va specifica ce submeniu se afiseaza pentru fiecare optiune cu setul de comenzi ON PAD ... ACTIVATE POPUP. Apoi se vor crea submeniuuri (popup sau meniuri verticale), pentru fiecare optiune bara, folosind comanda DEFINE POPUP. Intregul sistem este activat cu comanda ACTIVATE MENU si dezactivat cu comanda DEACTIVATE MENU.

<nume-meniu>-este numele meniului

BAR [AT LINE <expN1>]-creeaza un meniu orizontal asemanator meniului sistem al VFP-ului. Acest meniu are urmatoarele caracteristici: daca s-a facut o selectie, bara de meniu nu mai este activa, plasarea submeniurilor asociate optiunilor de meniu este realizata automat. Daca optiunile de meniu orizontal depasesc dimensiunile ecranului atunci acestea vor defila orizontal. Clauza AT LINE specifica linia (prin numarul ei <expN1>) pe care va fi plasat meniul orizontal.

IN [WINDOW] <nume-fer>

| IN SCREEN-Implicit meniul orizontal este plasat pe ecran daca nu este activa o fereastră definită de utilizator. Dacă o astfel de fereastră (al cărui nume este <nume-fer>) este activă atunci meniul va fi plasat în ea cu ajutorul clauzei IN WINDOW. Clauza IN SCREEN este inclusă pentru a plasa explicit meniul orizontal pe ecran (fereastră principala VFP).

KEY < eticheta-taste >-O bara de meniu orizontal poate fi activată prin apăsarea unei taste sau a unei combinații de taste specificată prin < eticheta-taste >. Folosirea clauzei KEY este echivalentă cu comanda:

ON KEY LABEL < eticheta-taste> ACTIVATE MENU <nume-meniu>

Observație. Dacă combinația de taste aleasă pentru a activa meniul a fost deja asociată unei acțiuni atunci meniul considerat nu va putea fi activat cu combinația respectivă.

MESSAGE <expC2>-permite afișarea unui mesaj <expC2> atunci când este selectată respectiva opțiune de meniu. Acest mesaj va apărea centrat la coordonatele specificate de comanda SET MESSAGE.

NOMARGIN- înlătură spațiul (inserat implicit) între marginea stângă și cea dreaptă a fiecărei opțiuni de meniu.

COLOR SCHEME <expN2>| COLOR <lista- perechi-culori>-specifica schema coloristică folosită pentru realizarea barei de meniu.

Exemplu de realizarea a unui meniu 'Agenti' cu două opțiuni de meniu 'Angajati' și 'Servicii' care la rândul lor au alte opțiuni de submeniu.

CLEAR

SET SYSMENU SAVE

SET SYSMENU TO

ON KEY LABEL ESC KEYBOARD CHR(13)

DEFINE MENU Agenti BAR AT LINE 1

DEFINE PAD personal OF agenti PROMPT "\<PERSONAL' COLOR SCHEME 3 ;
KEY ALT+C, "

DEFINE PAD serv OF agenti PROMPT "\<SERVICII' COLOR SCHEME 3 ;
KEY ALT+I, "

ON PAD personal OF agenti ACTIVATE POPUP pers

ON PAD serv OF agenti ACTIVATE POPUP servicii

```

DEFINE POPUP pers MARGIN RELATIVE SHADOW COLOR SCHEME 4
DEFINE BAR 1 OF pers PROMPT 'Personal de \<baza ' ;
    KEY CTRL+b, '^b'
DEFINE BAR 2 OF pers PROMPT 'Personal \<auxiliar' ;
    KEY CTRL+a, '^a'
DEFINE BAR 3 OF pers PROMPT 'Personal a\<sociat' ;
    KEY CTRL+s, '^s'

```

```

DEFINE POPUP servicii MARGIN RELATIVE SHADOW COLOR SCHEME 4
DEFINE BAR 1 OF servicii PROMPT '\<Vanzare' ;
    KEY ALT+V, "
DEFINE BAR 2 OF servicii PROMPT 'View \<Comparare' ;
    KEY ALT+C, "
DEFINE BAR 3 OF servicii PROMPT '\<Transport' ;
    KEY ALT+T, "

```

```

ON SELECTION POPUP pers;
DO proc1
ON SELECTION POPUP servicii;
DO proc2

```

```

ACTIVATE MENU agenti
DEACTIVATE MENU agenti
RELEASE MENU agenti EXTENDED
SET SYSMENU TO DEFAULT
ON KEY LABEL ESC

```

```

PROCEDURE proc1
@ 20,30 SAY "Ati ales o optiune din prima bara de meniū"
WAIT " TIMEOUT 1
PROCEDURE proc2
@ 20,30 SAY "Ati ales o optiune din a doua bara de meniū"
WAIT " TIMEOUT 1

```

Optiunile meniurilor bara (orizontale) sunt definite cu comanda DEFINE PAD.

```

DEFINE PAD <nume optiune> OF <nume-meniu> PROMPT <expC1>
    [AT <linie, coloana>]
    [BEFORE <nume pad> | AFTER <nume pad>]
    [KEY <eticheta-taste > [, <expC2>]]
    [SKIP [FOR <expL>]]
    [MESSAGE <expC4>]
    [COLOR SCHEME <expN> | COLOR <lista- perechi-culori>]

```

Efect: Creaza o optiune de meniu intr-o bara de meniu definita de utilizator sau proprie sistemului FoxPro. Este folosita impreuna cu comanda DEFINE MENU pentru a crea un meniu. Optiunea de meniu creata cu comanda DEFINE PAD poate fi selectata si aleasa

caz in care poate fi afisat un submeniu (popup) care la randul lui poate avea mai multe optiuni carora l-i se pot asocia diferite actiuni.

Fiecare optiune din bara de meniu trebuie definita cu comanda DEFINE PAD. Meniul bara principal (orizontal) trebuie definit inainte de a defini optiunile sale de meniu. Exceptie face meniul sistem FoxPro (_MSYSMENU), a carui definire nu mai este necesara.

<nume optiune> - este numele optiunii de meniu pe care il creem.

OF <nume-meniu> - specifica carui meniu bara i se asociaza optiunea de meniu.

PROMPT <expC1> - specifica textul <expC1> care va fi afisat pentru optiunea de meniu ce va fi definita. Se poate evidenta o litera in <expC1> prin includerea caracterelor '<' in fata ei. Aceasta litera poate fi folosita pentru selectarea optiunii de meniu(este o asa numita *hot key*).

Exemplu:

```
DEFINE MENU aparat
```

```
DEFINE PAD ap_el OF aparat PROMPT "Aparate \<Electrocasnice"
```

```
DEFINE PAD ap_med OF aparat PROMPT "Aparatura \<Medicala"
```

```
ACTIVATE MENU aparat
```

Prima optiune poate fi selectata apasand E iar a doua apasand M.

AT <linie, coloana>- specifica locul unde este afisata optiunea de meniu (<linie, coloana> sunt coordonatele coltului stanga sus al optiunii pe ecran sau in fereastra definita de utilizator). Daca este omisa clauza AT atunci coltul stanga sus a primei optiuni de meniu este plasat pe linia 0 a ecranului(ferestrei definite de utilizator), cea de – doua optiune este plasata in dreapta sa pe aceeasi linie etc..)

(Obs. Nu se foloseste clauza AT pentru a specifica a locatia optiunilor create cu clauza BAR a comenzii DEFINE MENU.)

BEFORE <nume pad> | AFTER <nume pad> - Optiunile de meniu sunt activate in ordinea crearii. Incluzand clauzele BEFORE si AFTER se specifica locatia si ordinea de activare a optiunilor din bara de meniu in raport cu alte optiuni <nume pad>(inainte sau dupa <nume pad>).

```
DEFINE MENU inainte_si_inapoi
```

```
DEFINE PAD unu OF inainte_si_inapoi PROMPT '1111'
```

```
DEFINE PAD doi OF inainte_si_inapoi PROMPT '2222'
```

```
DEFINE PAD trei OF inainte_si_inapoi PROMPT '3333'
```

```
DEFINE PAD patru OF inainte_si_inapoi PROMPT '4444' BEFORE doi
```

```
ACTIVATE MENU inainte_si_inapoi
```

KEY <eticheta-taste> [, <expC2>]- specifica o combinatie de taste pentru selectarea optiunii de meniu (vezi comanda DEFINE MENU).

SKIP [FOR <expL>]- accesul la o optiune de meniu pote controlat de o conditie logica. Includand clauza SKIP FOR <expL>, este evaluata expresia logica <expL> si pe baza rezultatului evaluarii (true sau false), optiunea de meniu este accesibila sau nu (adica poate fi selectata si aleasa sau nu) .

MESSAGE <expC4>-este afisat mesajul <expC4> la selectarea optiunii de meniu.
COLOR SCHEME <expN>| COLOR <lista- perechi-culori>>-specifica schema coloristica folosita pentru realizarea optiunii de meniu.

XV.2 Meniuri verticale. Definitia meniurilor verticale si a optiunilor de submeniu

Definirea submeniurilor se realizeaza cu comanda DEFINE POPUP.

```
DEFINE POPUP <nume meniu vert>[FROM <linie1, coloana1>]
    [TO <linie2, coloana2>][IN [WINDOW] <nume-fer>
    | IN SCREEN] [KEY <eticheta-taste>]
    [MARGIN][MESSAGE <expC3>]
    [MOVER][MULTISELECT][PROMPT FIELD <expr>
    | PROMPT FILES [LIKE <sablon>]| PROMPT STRUCTURE]
    [RELATIVE][SCROLL][SHADOW][TITLE <expC4>]
    [COLOR SCHEME <expN>| COLOR <lista- perechi-culori>]
```

Efect: Creaza un meniu vertical (submeniu) care poate contine o lista de optiuni definite de utilizator.

< nume meniu vert > - este numele submeniului.

FROM <linie1, coloana1> TO <linie2, coloana2>- specifica coordonatele coltului stanga sus (<linie1, coloana1>) si respectiv dreapta jos.

IN [WINDOW] <nume-fer> | IN SCREEN – vezi comanda DEFINE MENU

KEY <eticheta-taste>— vezi comanda DEFINE MENU

MARGIN – insereaza spatii la stanga si la dreapta meniului.

MESSAGE <expC3>— vezi comanda DEFINE MENU

MOVER - permite ca utilizatorul meniului vertical sa modifice ordinea in care sunt afisate optiunile submeniului. In acest caz optiunile vor fi marcate in partea stanga de caracterul '⇄'. Dupa sectarea unei optiuni aceasta se poate pdeplasa pe alta pozitie prin apasarea tastelor [Ctrl + ↑] sau [Ctrl + ↓]. Nu pot fi rearanjate optiunile unui meniu creat cu clauza PROMPT.

MULTISELECT - permite selectarea multipla a optiunilor de meniu. Selectarea optiunilor in acest caz se face tinand apasata tasta [Shift] si apoi selectand optiunile prin tasta [Enter] sau [Spacebar]. Selectare consecutiva a optiunilor se face apasand tasta [Shift] si apoi apasand [↑] sau [↓].

Facem observatia ca functiile CNTBAR(<expC>), MRKBAR(<expC>, <expN>), PRMBAR(<expC>, <expN>) actioneaza astfel:

CNTBAR(<expC>) returneaza numarul de optiuni ale submeniului <expC>,
MRKBAR() returneaza valoarea .T. daca optiunea cu numarul <expN> a submeniului <expC> a fost selectata si .F. in caz contrar iar PRMBAR(<expC>, <expN>) afiseaza textul asociat optiunii cu numarul <expN> a submeniului <expC>.

```
CLEAR
DEFINE POPUP fructe FROM 5,5 ;
    MULTISELECT MARGIN && se creeaza o selectie multipla
DEFINE BAR 1 OF fructe ;
    PROMPT '\<Mere' MARK CHR(3)
DEFINE BAR 2 OF fructe ;
    PROMPT '\<Banane' MARK CHR(4)
DEFINE BAR 3 OF fructe ;
    PROMPT '\<Portocale' MARK CHR(5)
DEFINE BAR 4 OF fructe ;
    PROMPT '\<Lamai' MARK CHR(6)

@ 12,5 SAY 'Alegerea d-voastra:'

ON SELECTION POPUP fructe DO a
ACTIVATE POPUP fructe

PROCEDURE a
@ 13,5 CLEAR
FOR i = 1 TO CNTBAR('fructe') && numara optiunile
    IF MRKBAR('fructe', i) = .T. && Daca optiunea este marcata,
        ? PRMBAR('fructe', i) AT 5 && afiseaza textul asociat optiunii
    ENDIF
NEXT
```

PROMPT FIELD <expr> - permite specificarea unui camp dintr-o tabela deschisa ale carei inregistrari vor deveni optiunile unui popup. Cand submeniul este activat, atunci este selectata zona de lucru in care este deschisa tabela ce contine campul.

<expr> poate contine mai multe nume de campuri si expresii concatenate cu operatorul (+) sau poate fi un nume de camp.

PROMPT FILES [LIKE <sablon>] permite crearea unui submeniu prin care sunt afisate toate fisierele de pe discul si directorul curent. Se vor include in meniu numai fisierele al caror nume satisface conditia <sablon>. De exemplu, pentru a crea un meniu care sa afiseze numai numele de tabele din directorul curent se foloseste comanda:

```
PROMPT FILES LIKE *.DBF
```

Iar pentru a afisa ca optiuni numele fisierele dintr-un alt director scriem comanda:

PROMPT FILES LIKE E:\PROGRAMS*.PRG

PROMPT STRUCTURE- creaza un submeniu care are drept optiuni numele campurilor din tabela curenta(conform structurii tabelii). La activarea submeniului este selectata zona de lucru ce contine tabela.

RELATIVE- controleaza ordinea in care sunt introduse optiunile in submeniu. Daca un meniu este creat fara aceasta clauza atunci optiunile vor fi pozitionate in meniu in ordinea in care sunt indicate prin comanda DEFINE BAR. Pot fi rezervate spatii in submeniu pentru optiuni care nu au fost inca definite. De exemplu daca prima si a treia optiune au fost definite, in momentul activarii submeniului va fi rezervata o linie pentru optiunea a doua. In cazul in care apare clauza RELATIVE optiunile sunt afisate in ordinea in care sunt definite. Nu se rezerva spatii pentru optiunile care nu au fost definite.Sunt acceptate clauzele AFTER si BEFORE in comanda DEFINE BAR.

Exemplu de folosire a clauzei RELATIVE

```
DEFINE POPUP rel RELATIVE FROM 1,1
```

```
DEFINE BAR 4 OF rel PROMPT '4444'
```

```
DEFINE BAR 3 OF rel PROMPT '3333'
```

```
DEFINE BAR 2 OF rel PROMPT '2222'
```

```
DEFINE BAR 1 OF rel PROMPT '1111'
```

```
DEFINE BAR 6 OF rel PROMPT '6666' BEFORE 4
```

```
ACTIVATE POPUP rel
```

Exemplu de ne-folosire a clauzei RELATIVE

```
DEFINE POPUP norel FROM 1,1
```

```
DEFINE BAR 4 OF norel PROMPT '4444'
```

```
DEFINE BAR 3 OF norel PROMPT '3333'
```

```
DEFINE BAR 2 OF norel PROMPT '2222'
```

```
DEFINE BAR 1 OF norel PROMPT '1111'
```

```
DEFINE BAR 6 OF norel PROMPT '6666'
```

```
ACTIVATE POPUP norel
```

SCROLL – plaseaza o bara de defilare in partea dreapta a submeniului.Aceasta este afisata doar daca sunt optiuni care nu incep in interiorul submeniului definit.

SHADOW- va afisa meniul cu umbra.

TITLE <expC4> -permite afisarea titlului <expC4> . Acesta va apare centrat pe latura superioara a chenarului.

COLOR SCHEME <expN>| COLOR <lista- perechi-culori> - specifica schema coloristica folosita (vezi comanda USE pentru detalii)

Opțiunile meniurilor verticale (submeniurilor) sunt definite cu ajutorul comenzii DEFINE BAR, care este prezentată mai jos.

Sintaxa:

```
DEFINE BAR <expN1> | <nume-opt-sist> OF <nume meniu-vert> PROMPT <expC1>  
    [BEFORE <expN2> | AFTER <expN3>][KEY <eticheta-taste> [, <expC2>]]  
    [MESSAGE <expC4>][SKIP [FOR <expL>]]  
    [ COLOR SCHEME <expN4>| COLOR <lista- perechi-culori>]
```

Efect: Creează opțiuni de meniu pentru un submeniu definit cu comanda DEFINE POPUP sau pentru un submeniu al meniului sistem VFP.

<expN1>-este numărul asignat opțiunii de meniu corespunzător poziției ocupate.

<nume-opt-sist>- va plasa în interiorul meniului opțiuni ale meniului sistem. Lista numelor meniurilor sistem este dată de funcția SYS(2013).

OF <nume meniu-vert > - specifică numele submeniului popup (<nume meniu-vert >) căruia i se asociază opțiunile.

PROMPT <expC1> -definește textul care apare în meniul popup ca opțiune (vezi și comanda DEFINE PAD)

```
DEFINE BAR 4 OF <nume popup> PROMPT '\'
```

Restul clauzelor au același efect ca și cele din comanda DEFINE PAD)

XV.3 Atribuirea de operații la alegerea opțiunilor

După ce au fost definite meniurile orizontale și verticale și opțiunile acestora urmează atribuirea de operații la alegerea fiecărei opțiuni. Trebuie făcută diferența între două tipuri de activare a unei acțiuni:

- una care se realizează prin deplasarea pe opțiune (fără acționarea tastei [Enter]), operație numită **selectare**;
- o alta care se realizează prin selectarea opțiunii și apăsarea tastei [Enter], operație numită **alegere**.

1. Selectarea opțiunilor de meniulsubmeniu

a) Selectarea opțiunilor de meniu orizontal se realizează cu comanda ON PAD a cărei sintaxă și acțiune este descrisă în cele ce urmează.

Sintaxa: ON PAD <nume pad> OF <nume-meniu1>

```
[ACTIVATE POPUP <nume popup>| ACTIVATE MENU <nume-meniu2>]
```

Efect: Specifică ce submeniulmeniu va fi activat când este aleasă opțiunea de meniu specificată.

Se folosesc comenzile ON SELECTION PAD sau ON SELECTION MENU pentru a executa o comanda cand este aleasa o optiune de submeniu.

<nume pad> OF <nume-meniu1> -specifica optiunea <nume pad> din meniul parinte <nume-meniu1> la alegerea careia va fi activat submeniul <nume popup> sau meniul orizontal <nume-meniu2>.

ACTIVATE POPUP <nume popup>

| ACTIVATE MENU <nume-meniu2>-specifica numele submeniului <nume popup> sau al meniului (<nume-meniu2>) care va fi activat la alegerea optiunii de meniu <nume pad> . Se foloseste comanda ON PAD <nume pad> OF <nume-meniu1> fara clauzele ACTIVATE POPUP sau ACTIVATE MENU pentru a inlatura (sterge) asocierea unui submeniu sau meniu la alegerea optiunii de meniu <nume pad>.

Ca exemplu se poate considera exemplul dat la prezentarea comenzii DEFINE MENU.

b) In cazul optiunilor de meniu vertical (submeniu) selectarea unei optiuni se face cu comanda ON BAR.

Sintaxa: ON BAR <expN> OF <nume-submeniu1>

[ACTIVATE POPUP <nume submeniu2>| ACTIVATE MENU <nume-meniu-oriz>]

Efect: Specifica ce submeniul/meniu orizontal va fi activat cand este aleasa optiunea de submeniu specificata.

Se foloseste comanda ON SELECTION BAR pentru ca la alegerea unei optiuni de submeniu sa fie executata o anumita comanda.

<expN> OF <nume-submeniu1> -specifica numarul <expN> al optiunii din submeniul parinte <nume-submeniu1> la alegerea careia va fi activat submeniul <nume submeniu2> sau <nume-meniu-oriz>.

ACTIVATE POPUP <nume submeniu2>

| ACTIVATE MENU <nume-meniu-oriz>-specifica numele submeniului <nume submeniu2> sau al meniului orizontal (<nume-meniu-oriz>) care va fi activat la alegerea optiunii de submeniu <expN> .

2. Alegerea optiunilor de meniulsubmeniu

In cazul operatiei de alegere a unei optiuni de meniulsubmeniu se va observa ca poate fi executata orice instructiune, nu numai aceea de activare a unui meniu respectiv submeniu ca in cazul operatiei de selectare.

a) Alegerea optiunilor de meniu orizontal se realizeaza cu ajutorul comenzii ON SELECTION PAD.

Sintaxa:

ON SELECTION PAD <nume pad> OF <nume-meniu> [<comanda>]

Efect: Specifica ce comanda este executata cand este aleasa optiunea de meniu precizata. De obicei ON SELECTION PAD foloseste comanda DO pentru a executa o procedura

sau un program atunci cand este aleasa o optiune de meniu. Comanda ON SELECTION PAD este plasata intre comenzile DEFINE MENU si ACTIVATE MENU cand se activeaza meniul orizontal.

Pentru activarea optiunii de meniu <nume pad> este folosita in prealabil comnda ON PAD.

<nume pad> OF <nume-meniu> [<comanda>]-specifica comanda <command> ce va fi executata la alegerea optiunii <nume pad> a meniului orizontal <nume-meniu>.

Comanda ON SELECTION POPUP

ON SELECTION POPUP <nume popup> [ALL[<comanda>]]

Efect: utilizatorul poate asigna o rutina globala (comanda, procedura, program) optiunilor ce formeaza un meniu popup sau tuturor meniurilor popup active. Deci selectarea oricarei optiuni de meniu este urmata de executarea aceleiasi rutine.

<nume popup>-este numele submeniului popup care va avea signata o rutina.

ALL este clauza prin care rutina este asignata tuturor meniurilor verticale active.

[<comanda>]-este numele unei comenzi, rutine sau program.

b) Alegerea optiunilor de meniu vertical se realizeaza cu ajutorul comenzii ON SELECTION BAR

ON SELECTION BAR

Sintaxa:

ON SELECTION BAR <expN>

OF <nume popup>[<comanda>]

Efect: Specifica ce comanda este executata cand este aleasa optiunea de submeniu precizata.(vezi si ON SELECTION PAD, ON SELECTION POPUP)

Pentru activarea optiunii de submeniu se foloseste comanda ON BAR...

<expN> OF < nume popup > [<comanda>]-specifica numarul optiunii (<expN>) al submeniului <popup name> si comanda ce va fi executata la alegerea optiunii de submeniu.

Se foloseste comanda ON SELECTION BAR <expN> OF <nume popup> pentru a sterge asocierea comenzii <comanda> la optiunea cu numarul <expN>.

XV.4 Activarea meniurilor/submeniurilor

Definirea unei bare de meniu si a unui submeniu nu este suficienta pentru a se putea lucra cu aceste elemente. Pentru afisarea si activarea lor sunt folosite comenzile ce vor fi descrise in cele ce urmeaza. Comanda ACTIVATE MENU activeaza meniurile orizontale in timp ce comanda ACTIVATE POPUP activeaza meniurile verticale.

Comanda ACTIVATE MENU

Sintaxa:

ACTIVATE MENU <nume-meniu>[NOWAIT][PAD <nume pad>]

Efect: Afiseaza si activeaza un meniu orizontal. Aceasta comanda actioneaza impreuna cu comenzile DEFINE MENU si DEFINE PAD.

Observatie. Daca este inclus meniul sistem VFP (_MSYSMENU) intr-o aplicatie, nu este necesara activarea acestuia (se foloseste in loc comanda SET SYSMENU AUTOMATIC).

<nume-meniu>-specifica numele meniului ce va fi activat.

NOWAIT-determina continuarea executarii programului dupa activarea meniului. (In mod implicit la activarea meniului executia programului este oprita pana este aleasa o optiune de meniu sau este apasata tasta [Escape].

PAD <nume pad>- are ca efect pozitionarea barei luminoase pe optiunea <nume pad>(in mod normal aceasta este pozitionata pe prima optiune).

Comanda ACTIVATE POPUP

Sintaxa:

ACTIVATE POPUP <popup name> [AT <linie, coloana>]
[BAR <expN>][NOWAIT][REST]

Efect: Afiseaza si activeaza pe ecran, sau intr-o fereastră un meniu vertical(submeniu) definit anterior. Aceasta comanda activeaza si are sens numai impreuna cu comanda DEFINE POPUP.

Intr-un popup se navigheaza folosind sagetile sus si jos.

<popup name>- specifica numele popup-ului ce va fi activat.

AT <linie, coloana> - specifica coordonatele ecranului la care se va afisa meniul. Aceste coordonate sunt luate in seama chiar daca au fost specificate alte coordonate in comanda DEFINE POPUP.

BAR <expN> -la afisarea submeniului este selectata optiunea cu numarul <expN>.

NOWAIT- executia programului continua si dupa afisarea si activarea meniului vertical(popup).

REST-va determina pozitionarea barei de selectie pe optiunea al carei numar coincide cu cel al inregistrarii curente din tabela curent selectata. (Este folosita atunci cand in comanda DEFINE POPUP este folosita clauza PROMPT FIELD.)

XV.5 Afisarea meniurilor/submeniurilor fara activarea lor

Afisarea meniurilor/submeniurilor pe ecran sau in fereastră activa fara activarea lor se realizeaza cu comenzile SHOW MENU si respectiv SHOW POPUP.

Comanda SHOW MENU

Sintaxa: SHOW MENU <nume-meniu1>[, <nume-meniu2> ...] | ALL
[PAD <nume optiune bara>][SAVE]

Efect: Afiseaza meniuri orizontale fara a le activa.

<nume-meniu1>[,<nume-meniu2> ...] – specifica numele unuia sau a mai multor meniuri orizontale ce vor fi afisate.

ALL – specifica faptul ca toate meniurile orizontale definite vor fi afisate

PAD <nume optiune bara > - specifica numele (<nume optiune bara >) optiunii bara ce va fi selectata.

SAVE – este salvata o imagine a meniului orizontal fara activarea lui. Imaginea lui poate fi stearsa cu comanda CLEAR.

Exemplu:

CLEAR

DEFINE MENU exemplu BAR AT LINE 2

DEFINE PAD studenti OF exemplu PROMPT '\<Studenti' COLOR SCHEME 3 ;
KEY ALT+S, "

DEFINE PAD profesori OF exemplu PROMPT '\<Profesori' COLOR SCHEME 3 ;
KEY ALT+P, "

SHOW MENU example

Comanda SHOW POPUP

Sintaxa:

SHOW POPUP <nume submeniu1>
[, < nume submeniu2> ...] | ALL
[SAVE]

Efect: Afiseaza meniuri verticale fara a le activa.

<popup name1> [,<popup name2> ...] – specifica numele unuia sau a mai multor meniuri verticale ce vor fi afisate.

Clauzele acestei comenzi au efect asemanator celor descrise la comanda SHOW MENU.

XV.6 Eliminarea a meniurilor/submeniurilor fara inlaturarea lor din memorie

Eliminarea de pe ecran sau din fereastra activa a meniurilor/submeniurilor fara inlaturarea lor din memorie se realizeaza cu comenzile HIDE MENU si respectiv HIDE POPUP

Comanda HIDE MENU

Sintaxa:

HIDE MENU <nume-meniu1>
[, <nume-meniu2> ...] | ALL
[SAVE]

Efect: Eliminarea de pe ecran sau din fereastra activa unul sau mai multe meniuri fara inlaturarea lor din memorie. Un meniu bara ascuns poate fi afisat din nou pe ecran sau in fereastra activa cu ajutorul comenzilor ACTIVATE MENU sau SHOW MENU.

Comanda HIDE

Sintaxa:

HIDE POPUP <popup name1>
[, <popup name2> ...] | ALL
[SAVE]

Efect: Eliminarea de pe ecran sau din fereastra activa unul sau mai multe submeniuri fara inlaturarea lor din memorie. Un submeniu ascuns poate fi afisat din nou pe ecran sau in fereastra activa cu ajutorul comenzilor ACTIVATE POPUP sau SHOW POPUP.

Clauzele acestor doua comenzi au efect asemanator celor descrise in paragraful precedent.

XV.7 Dezactivarea meniurilor

Comanda DEACTIVATE MENU

Sintaxa:

DEACTIVATE MENU <nume-meniu1>
[, <nume-meniu2> ...] | ALL

Efect: Dezactivarea unui meniu sau mai multor meniuri definite de utilizator de pe ecran, fara a inlatura definitia acestora din memorie. Meniurile pot fi reactivate cu comanda ACTIVATE MENU.

<nume-meniu1> [, <nume-meniu2> ...]-specifica numele meniurilor care vor fi dezactivate.

ALL – specifica faptul ca toate meniurile active vor fi dezactivate

Comanda DEACTIVATE POPUP

Sintaxa:

DEACTIVATE POPUP <nume submeniu1>[, <nume submeniu2> ...] | ALL

Efect: Dezactivarea unui meniu vertical sau mai multor meniuri verticale definite de utilizator de pe ecran, fara a inlatura definitia acestora din memorie. Meniurile verticale pot fi reactivate cu comanda ACTIVATE POPUP.

<nume submeniu1> [, <nume submeniu2> ...]- specifica numele submeniurilor ce vor fi dezactivate.

ALL –specifica faptul ca toate submeniurile vor fi dezactivate.

XV.8 Stergerea meniurilor

Comanda RELEASE

Sintaxa:

RELEASE MENUS [<lisat de meniuri> [EXTENDED]]

Efect: Inlatura (sterge) meniurile orizontale specificate din memorie. Meniurile active for fi mai intai dezactivate cu comanda DEACTIVATE MENU. Daca comanda RELEASE MENUS este folosita fara lista de meniuri atunci toate meniurile definite de utilizator sunt inlaturate din memorie. Includerea clauzei EXTENDED permite stergerea odata cu meniul a tuturor optiunilor, submeniurilor sale sau a tuturor actiunilor asociate prin comenzile ON SELECTION BAR, ON SELECTION MENU, ON SELECTION PAD si ON SELECTION POPUP.

Sintaxa: RELEASE PAD <nume optiune> OF <nume-meniu> | ALL OF <nume-meniu>

Efect: Sterge optiunile de meniu orizontal specificate (sau pe toate) dintr-o bara de meniu. De exemplu comanda RELEASE PAD _MEDIT OF _MSYSMENU inlatura optiunea Edit si submeniul sau din bara de meniu sistem.

Daca este inclusa clauza ALL sunt inlaturate toate optiunile de meniu. Comanda ALL nu poate fi folosta pentru a sterge optiunile meniului sistem.

Sintaxa: RELEASE POPUPS

[<lista meniuri-vert> [EXTENDED]]

Efect: Inlatura (sterge) meniurile verticale specificate din memorie. Meniurile active for fi mai intai dezactivate cu comanda DEACTIVATE POPUP. Daca comanda RELEASE POPUPS este folosita fara lista de meniuri atunci toate submeniurile definite de utilizator sunt inlaturate din memorie. Includerea clauzei EXTENDED permite stergerea odata cu meniul a tuturor optiunilor, submeniurilor sale sau a tuturor actiunilor asociate prin comenzile ON SELECTION BAR si ON SELECTION POPUP

Sintaxa: RELEASE BAR <expN>

OF <nume meniu-vert>

| ALL OF <nume meniu-vert>

Efect: Inlatura (sterge) optiunile de submeniu specificate (sau pe toate) dintr-un submeniu. Pentru inlaturarea unei optiuni a meniului sistem (Edit, File, Database etc), este inclus numele optiunii sistem in <expN>. Inlaturarea tuturor optiunilor se face incuzand clauza ALL. (Clauza ALL nu poate fi folosita pentru a sterge toate optiunile unui submeniu sistem).

Bibliografie

- [1] O.Basca, "Baze de date", Editura Didactica si Pedagogica, Bucuresti 1994.
- [2] G.Dima,M.Dima: Foxpro pentru DOS, Editura Teora, 1997.
- [3] D.Maier: The Theory of Relational Databases, Academic Press, 1992.
- [4] V.Felea: Baze de date relationale. Dependente, Ed. Univ. Iasi, 1996.
- [5] J.D.Ullman: Principles of Databases Systems, Comp. Sc. Press, 1990.
- [6] M. Velicanu, I. Lungu, M. Muntean, Dezvoltarea aplicatiilor cu baze de date in Visual FoxPro, Ed. Bic ALL, 2001.
- [7] Microsoft Visual FoxPro Help, versiunea 6.0.

Creates a new file from the contents of the currently selected table.

```
COPY TO FileName [DATABASE DatabaseName [NAME LongTableName]]  
  [FIELDS FieldList | FIELDS LIKE Skeleton | FIELDS EXCEPT Skeleton]  
  [Scope] [FOR lExpression1] [WHILE lExpression2]  
  [ [WITH] CDX ] | [ [WITH] PRODUCTION ] [NOOPTIMIZE]  
  [ [TYPE] [ FOXPLUS | FOX2X | DIF | MOD | SDF | SYLK | WK1 | WKS | WR1  
  | WRK | CSV | XLS | XL5 | DELIMITED [ WITH Delimiter | WITH BLANK  
  | WITH TAB | WITH CHARACTER Delimiter ] ] ] [AS nCodePage]
```

Parameters

FileName

Specifies the name of the new file to create. If you do not include a file extension, **COPY TO** assigns the default extension for the specified file type. If you do not specify a file type, **COPY TO** creates a new Visual FoxPro table and assigns the default .dbf extension to the table file name.

DATABASE *DatabaseName*

Specifies a database to which the new table is added.

NAME *LongTableName*

Specifies a long name for the new table. Long names can contain up to 128 characters and can be used instead of short file names in the database.

FIELDS *FieldList*

Specifies which fields are copied to the new file. If you omit **FIELDS***FieldList*, all fields are copied to the file. If the file you are creating is not a table, memo fields are not copied to the new file, even if memo field names are included in the field list.

FIELDS LIKE *Skeleton*

Specifies that fields from the original table that match the field skeleton *Skeleton* are included in the new file that **COPY TO** creates.

FIELDS EXCEPT *Skeleton*

Specifies that all fields except those that match the field skeleton *Skeleton* are included in the new file that **COPY TO** creates.

Scope

Specifies a range of records to copy to a file. Only the records within the range are copied. The scope clauses are: **ALL**, **NEXT***nRecords*, **RECORD***nRecordNumber*, and **REST**. For more information on scope clauses, see [Scope Clauses](#).

FOR *lExpression1*

Specifies that **COPY TO** copies only the records for which the logical condition *lExpression1* evaluates to True (.T.) to the file. To conditionally copy records, include the **FOR***lExpression1* clause to filter out undesired records.

WHILE *lExpression2*

Specifies a condition where records are copied while the logical expression *lExpression2* evaluates to True (.T.).

[WITH] CDX | [WITH] PRODUCTION

Creates a structural index file for the new table that is identical to the structural index file of the existing table. The tags and index expressions from the original structural index file are copied to the new structural index file. The **CDX** and **PRODUCTION** clauses are identical. However, do not include **CDX** or **PRODUCTION** if you are copying to a file other than a new Visual FoxPro table.

NOOPTIMIZE

Disables Rushmore Query Optimization for **COPY TO**. For more information, see [SET OPTIMIZE](#) and [Using Rushmore Query Optimization to Speed Data Access](#).

TYPE

Specifies the file type if the file you create is not a Visual FoxPro table. Although you must specify a file type, you do not need to include the **TYPE** keyword.

FOXPLUS

Visual FoxPro memo files have a different structure than Microsoft FoxBASE+™ memo files. If your source Visual FoxPro table contains a memo field, include the **FOXPLUS** clause to create a table that can be used in FoxBASE+. The Visual FoxPro memo field cannot contain binary data because FoxBASE+ does not support binary data in memo fields.

FOX2X

Creates a new table that can be opened in earlier versions of FoxPro (versions 2.0, 2.5, and 2.6). For Numeric, Float, Integer, Double, and Currency type fields, null values in the source table are converted to zero in the new table. For other field types, null values in the source table are converted to blanks in the new table. For further information about blank values, see [ISBLANK\(\)](#). The following table lists the Visual FoxPro field types that are converted to different field types in the new table when the FOX2X argument is included.

Visual FoxPro field type FoxPro 2.x field type

Blob	Memo
Currency	Float
DateTime	Date
Double	Float
Integer	Numeric
Varbinary	Memo
Varchar	Memo

DIF

Creates a VisiCalc® .dif (Data Interchange Format) file. Fields from the Visual FoxPro table become vectors (columns) and records become tuples (rows). The new file name is assigned a .dif extension if you do not include an extension in *FileName*.

MOD

Creates a Microsoft Multiplan® version 4.01 file. The new Microsoft Multiplan file name is assigned a .mod extension if an extension is not included.

SDF

Creates an SDF (System Data Format) file. An SDF file is an ASCII text file in which records have a fixed length and end with a carriage return and line feed. Fields are not delimited. The SDF file name is assigned a .txt file extension if you do not include an extension. The **SET CENTURY** setting is ignored when creating SDF files with **COPY TO**. If SDF files include date data, it should be in YYYYMMDD format to allow effective reconversion into Visual FoxPro tables. If date information is stored in ambiguous formats, you should make sure that the dates are in YYYYMMDD format before you perform the **COPY TO** operation.

SYLK

Creates a SYLK (Symbolic Link) interchange file. SYLK files are used in Microsoft MultiPlan. Each field from the currently selected table becomes a column in the spreadsheet, and each record becomes a row. SYLK file names have no extension.

WK1

Creates a Lotus® 1-2-3® version 2.x spreadsheet file. Each field from the currently selected table becomes a column in the spreadsheet and each record becomes a row. A .wk1 file name extension is assigned to the new spreadsheet.

WKS

Creates a Lotus 1-2-3 version 1a spreadsheet file. Each field from the currently selected table becomes a column in the spreadsheet and each record becomes a row. A .wks file name extension is assigned to the new spreadsheet.

WR1

Creates a Lotus Symphony® version 1.1 or 1.2 spreadsheet file. Each field from the currently selected table becomes a column in the spreadsheet and each record becomes a row. A .wr1 file name extension is assigned to the new spreadsheet.

WRK

Creates a Lotus Symphony version 1.0 spreadsheet file. Each field from the currently selected table becomes a column in the spreadsheet and each record becomes a row. A .wr1 file name extension is assigned to the new spreadsheet.

CSV

Creates a comma separated value file. A CSV file has the field names as the first line in the file, and the field values in the remainder of the file are separated with commas.

XLS

Creates a Microsoft Excel version 2.0 worksheet file. Each field from the currently selected table becomes a column in the spreadsheet, and each record becomes a row. If you do not include a file extension, an .xls extension is assigned to the new worksheet.

Note:

Though you can export a maximum of 65,535 rows, which includes one row reserved for the field header, versions of Excel earlier than 8.0 (Excel 97) display only the first 16,384 rows and cannot import files containing more than 32,767 rows.

XL5

Creates a Microsoft Excel version 5.0 workbook file. Each field from the currently selected table becomes a column in the spreadsheet, and each record becomes a row. If you do not include a file extension, an .xls extension is assigned to the new workbook.

Note:

Though you can export a maximum of 65,535 rows, which includes one row reserved for the field header, versions of Excel earlier than 8.0 (Excel 97) display only the first 16,384 rows and cannot import files containing more than 32,767 rows.

DELIMITED

Creates a delimited file. A delimited file is an ASCII text file in which each record ends with a carriage return and line feed. The default field separator is a comma. Because character data can include commas, character fields are additionally delimited with double quotation marks. Unless you specify otherwise, a .txt extension is assigned to all newly created **DELIMITED** files.

DELIMITED WITH *Delimiter*

Creates a delimited file with character fields delimited by a character other than a quotation mark. The character that delimits character fields is specified with *Delimiter*.

DELIMITED WITH BLANK

Creates a delimited file with fields separated by spaces instead of commas.

DELIMITED WITH TAB

Creates a delimited file with fields separated by tabs instead of commas.

DELIMITED WITH CHARACTER *Delimiter*

Creates a delimited file with all fields enclosed by the character specified with *Delimiter*. If *Delimiter* is a semicolon (;), used in Visual FoxPro to indicate command line continuation, enclose the semicolon in quotation marks. You can also specify the **BLANK** and **TAB** keywords for *Delimiter*. You can combine the **WITH *Delimiter*** clause with the **WITH CHARACTER** clause. For example, the following command creates a text file with character fields enclosed by underscores and all fields delimited from each other with semicolons:

Copy Code

```
COPY TO mytxt.txt DELIMITED WITH _ WITH CHARACTER ';' ;
```

AS *nCodePage*

Specifies the code page for the table or file that **COPY TO** creates. Visual FoxPro copies the contents of the currently selected table, and, as it copies the data, automatically converts the data to the code page you specify for the new table or file. If possible, Visual FoxPro marks the newly created table or file with the code page you specify. If you omit **AS *nCodePage***, the newly created table or file is converted to the current Visual FoxPro code page.

Remarks

The field skeleton *Skeleton* supports wildcards. For example, to specify that all fields that begin with the letters A and P are included in the new file, use the following:

Copy Code

```
COPY TO mytable FIELDS LIKE A*,P*
```

The **LIKE** clause can be combined with the **EXCEPT** clause:

Copy Code

```
COPY TO mytable FIELDS LIKE A*,P* EXCEPT PARTNO*
```

Rushmore Query Optimization optimizes **COPY TO** with a **FOR *lExpression1*** clause if *lExpression1* is an optimizable expression. For the best performance, use an optimizable expression in the **FOR *lExpression1*** clause. For information on Rushmore optimizable expressions, see [SET OPTIMIZE](#) and [Using Rushmore Query Optimization to Speed Data Access](#).

If an index order is set, records are copied in master index order.

If you specify a value for *nCodePage* that is not supported, Visual FoxPro generates an error message. You can use the **GETCP()** function for *nCodePage* to display the Code Page dialog box, allowing you to specify a code page for the table or file Visual FoxPro creates.

The **COPY TO** command preserves autoincrementing settings except under the following conditions:

- When the field list specified using **COPY TO...FIELDS** does not include the field that uses autoincrementing.
- When using the **TYPE** clause in the **COPY TO** command, regardless of the type specified by the clause.

The target table begins autoincrementing starting with the *NextValue* of the source table. For example, suppose the *NextValue* in the source table is 1000 with a *Step* value of 1. The first row of the target table then has an autoincrementing field value of 1001; the second row has a value of 1002, and so on.

When using the **COPY TO** command, be aware that the [SET VARCHARMAPPING Command](#) will impact calculated fields (character expressions of variable length) set by the [SET FIELDS Command](#).

Example

In the following example, the Customer table is opened and the next three records are copied to a new **DELIMITED** data file called Temp.txt.

Copy Code

```
CLOSE DATABASES
OPEN DATABASE (HOME(2) + 'Data\testdata')
USE Customer  && Opens Customer table.

COPY NEXT 3 TO Temp TYPE DELIMITED
WAIT WINDOW 'This is the delimited text file' NOWAIT
MODIFY FILE Temp.txt
DELETE FILE Temp.txt
```

IMPORTANT

Comanda COPY TO o folosim în special pentru exportul tablei într-o altă tabelă sau într-un fișier excel.

Cele mai utilizate comenzi sunt:

COPY TO *FileName* [FIELDS *FieldList*] - pentru exportul într-o altă tabelă

COPY TO *FileName* [FIELDS *FieldList*] TYPE XLS - pentru exportul într-un fișier excel.