

Cuprins

Cuprins.....	1
1. Bănci și baze de date	5
1.1. Noțiuni generale	5
1.2. Sisteme de baze de date	6
1.3. Organizarea datelor într-o bază de date.....	7
1.4. Modelarea la nivel logic a datelor într-o bază de date.....	8
1.5. Sistem de gestiune a bazelor de date	10
1.5.1. Interfețe SGBD	11
1.5.2. Funcții și servicii oferite de un SGBD	12
1.5.3. Activitățile asigurate de SGBD.....	14
1.5.4. Obiectivele unui SGBD	14
2. Etapele de realizare a unei bănci de date.....	17
2.1. Etapa de analiză.....	17
2.2. Etapa de programare.....	17
2.3. Punerea în funcțiune și exploatarea bazei de date	18
2.4. Documentația aplicației	19
3. Sisteme de gestiune a bazelor de date relaționale	20
3.1. Modelul relațional al datelor	20
3.1.1. Structura relațională a datelor	20
3.1.2. Operații în algebra relațională.....	21
3.1.3. Optimizarea cererilor de date.....	28
3.1.4. Baze de date relaționale	30
3.1.5. Tipuri de relații	32
3.1.6. Restricția de cardinalitate.....	33
3.2. Regulile lui Codd.....	35
3.3. Normalizarea datelor	37
3.4. Cerințele minimale de definire a SGBDR	37
4. Componentele funcționale ale sistemului Visual FoxPro	39
4.1. Programarea orientată pe obiecte	39
4.2. Ierarhia claselor în Visual FoxPro.....	44
4.2. Arhitectura VFP 6.0.....	45
4.3. Moduri de lucru în VFP.....	46
4.3.1. Modul de lucru meniu sistem.....	47
4.3.2. Modul de lucru prin comenzi.....	51

5. Organizarea datelor în Visual FoxPro	52
5.1. Manipularea bazei de date și a tabelor în VFP	52
5.1.1. Crearea bazei de date	52
5.1.2. Crearea tabelor	53
5.1.3. Validarea câmpurilor unei înregistrări la introducere	54
5.2. Deschiderea bazei de date/tabeli	59
5.3. Consultarea și modificarea bazei de date/tabeli	61
5.3.1. Modificarea structurii de date a tabelor în modul asistat	61
5.3.2. Deplasări în tabelă. Căutări secvențiale	62
5.4. Închiderea bazei de date/tabeli	64
6. Utilizarea asistentului Wizard în VFP	65
6.1. Table Wizard	65
6.2. Form Wizard	66
6.3. Report Wizard	67
6.4. Label Wizard	68
6.5. Query Wizard	69
6.6. Mail Merge Wizard	70
6.7. Editorul de texte în VFP 6.0	71
6.7.1. Lansarea editorului	71
6.7.2. Lucrul cu blocul de linii din fișier	71
7. Elemente ale limbajului propriu Visual FoxPro	73
7.1. Simboluri	73
7.2. Variabile de sistem	73
7.3. Comentariul	73
7.4. Tipuri de date, operații și funcții	74
7.4.1. Date de tip numeric. Funcții aritmetice și financiare	76
7.4.2. Aplicații ale funcțiilor financiare	77
7.4.3. Date și funcții de tip caracter	80
7.4.4. Date și funcții de tip calendaristic	82
7.4.5. Date de tip memo	83
7.5. Variabile și masive	83
7.5.1. Variabile	83
7.5.2. Macrosubstituția	84
7.5.3. Masive de date	85

7.6. Funcții de prelucrare a masivelor	86
8. Accesul și actualizarea datelor	91
8.1. Manipularea structurii unei tabele	91
8.2. Ordonarea datelor	92
8.3. Accesul la date.....	98
8.4. Accesul concurențial	102
8.5. Actualizarea datelor.....	103
8.5.1. Adăgarea de noi înregistrări.....	103
8.5.2. Modificarea înregistrărilor	107
8.5.3. Ștergerea înregistrărilor	110
8.5.5. Relații între tabele	113
9. Programarea procedurală	120
9.1. Programarea structurată.....	120
9.1.1. Structura liniară.....	121
9.1.2. Structura alternativă	124
9.1.3. Structura repetitivă.....	126
9.2. Modularizarea programelor	129
9.2.1. Proceduri	130
9.2.2. Funcții	131
10. Comenzi ale nucleului SQL.....	134
11. Proiectarea meniurilor și a barelor de instrumente	145
12. Aplicații	146
12.1. Evidența rezultatelor activității studenților într-o facultate.....	146
12.1.1. Formularea și analiza problemei	146
12.1.2. Crearea, actualizarea, modificarea și interogarea tabelor	148
12.1.3. Crearea structurii unui fișier	149
12.1.4. Adresarea prin macrosubstituție	150
12.1.5. Crearea unui meniu	150
12.2. Gestionarea unei magazii	152
12.2.1. Proiectarea și realizarea aplicației.....	152
12.2.2. Realizarea documentației și a aplicației în format executabil.....	220
12.2.3. Utilizarea Setup Wizard.....	224

A. Anexa.....	226
Index de termeni.....	228
Bibliografie	230

1. Bănci și baze de date

1.1. Noțiuni generale

O *bază de date* reprezintă un ansamblu de date integrat, anume structurat și dotat cu o descriere a acestei structuri. Descrierea structurii poartă numele de *dicționar de date* sau *metadate* și crează o interdependență între datele propriu-zise și programe.

Baza de date poate fi privită ca o colecție de fișiere interconectate care conțin nucleul de date necesare unui sistem informatic. Astfel, poate fi considerată drept un model al unor aspecte ale realității unei unități economice, modelată prin intermediul datelor. Diferitele obiecte din cadrul realității, ce prezintă interes, sunt denumite *clase* sau *entități*. Pentru aceste obiecte sunt achiziționate și memorate date referitoare la diferite *caracteristici (atribute)*. Baza de date se constituie ca un ansamblu intercorelat de colecții de date, prin care se realizează reprezentarea unei realități.

Datele constituie orice mesaj primit de un receptor, sub o anumită formă.

Informațiile reprezintă cantitatea de noutate adusă de un mesaj din exterior (realitate).

Un *fișier* este un ansamblu de *înregistrări fizice*, omogene din punct de vedere al conținutului și al prelucrării.

O *înregistrare fizică* este o unitate de transfer între memoria internă și cea externă a calculatorului.

O *înregistrare logică* este unitatea de prelucrare din punct de vedere al programului utilizator.

O înregistrare se compune din *câmpuri (attribute)* care descriu anumite aspecte ale realității. Câmpurile sunt înregistrări logice.

O bază de date trebuie să asigure:

- *abstractizarea datelor* (baza de date fiind un model al realității),

- *integrarea datelor* (baza de date este un ansamblu de colecții de date intercorelate, cu redundanță controlată),
- *integritatea datelor* (se referă la corectitudinea datelor încărcate și manipulate astfel încât să se respecte restricțiile de integritate),
- *securitatea datelor* (limitarea accesului la baza de date),
- *partajarea datelor* (datele pot fi accesate de mai mulți utilizatori, eventual în același timp),
- *independența datelor* (organizarea datelor să fie transparentă pentru utilizatori, modificările în baza de date să nu afecteze programele de aplicații).

1.2. Sisteme de baze de date

Sistemele de baze de date, sau *băncile de date*, reprezintă un sistem de organizare și prelucrare, respectiv teleprelucrare (prelucrare la distanță) a informației, constituit din următoarele trei elemente:

- colecție de date aflate în interdependență
- descrierea datelor și a relațiilor dintre ele
- un sistem de programe care asigură exploatarea bazei de date (actualizare, interogare).

creare baza
de date

Arhitectura sistemului de baze de date este formată din următoarele componente (Figura 1):

- baza/bazele de date – reprezintă componenta de tip date a sistemului (colecțiile de date propriu-zise, indecșii);
- sistemul de gestiune a bazei/bazelor de date – ansamblul de programe prin care se asigură gestionarea și prelucrarea complexă a datelor și care reprezintă componenta software a sistemului de baze de date (Sistem de Gestiune a Bazelor de Date – SGBD);
- alte componente – proceduri manuale sau automate, inclusiv reglementări administrative, destinate bunei funcționări a sistemului, dicționarul bazei de date (metabaza de date) care conține informații despre date, structura acestora, elemente de descriere a semanticii, statistici, documentații, mijloacele hardware utilizate, personalul implicat.

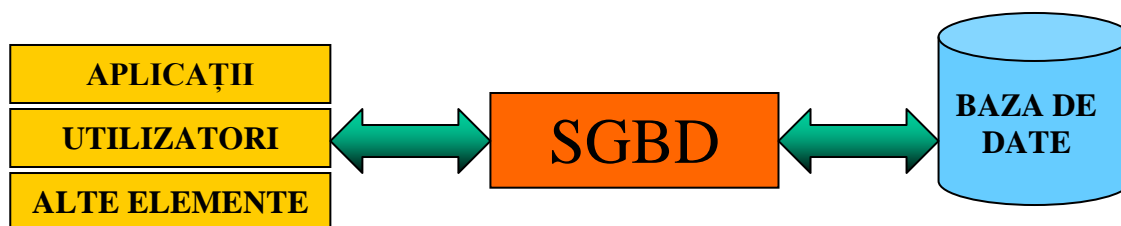


Fig. 1

1.3. Organizarea datelor într-o bază de date

Arhitectura internă a unui sistem de baze de date conform standardului ANSI/X3/SPARC (1975) conține trei niveluri funcționale. O caracteristică fundamentală a bazelor de date este aceea că produce câteva niveluri de abstractizare a datelor prin ascunderea (transparența) detaliilor legate de stocarea datelor, utilizatorilor.

Se definește *modelul datelor*, ca un set de concepte utilizat în descrierea structurii datelor. Prin *structura bazei de date* se înțelege tipul datelor, legătura dintre ele, restricțiile aplicate datelor. O structură de date asociată unei baze de date poate fi reprezentată pe trei niveluri, Figura 2, astfel:

- *Nivelul intern* – constituit din schema internă ce descrie structura de stocare fizică a datelor în baza de date, utilizând un model al datelor fizice. La acest nivel se descriu detaliile complete ale stocării și modul de acces la date.
- *Nivelul conceptual* – sau schema conceptuală, descrie structura întregii baze de date pentru o comunitate de utilizatori. La nivel conceptual se face o descriere completă a bazei de date ascunzându-se detaliile legate de stocarea fizică și detaliind descrierea entităților, tipurilor de date, relațiile dintre ele și restricțiile asociate.

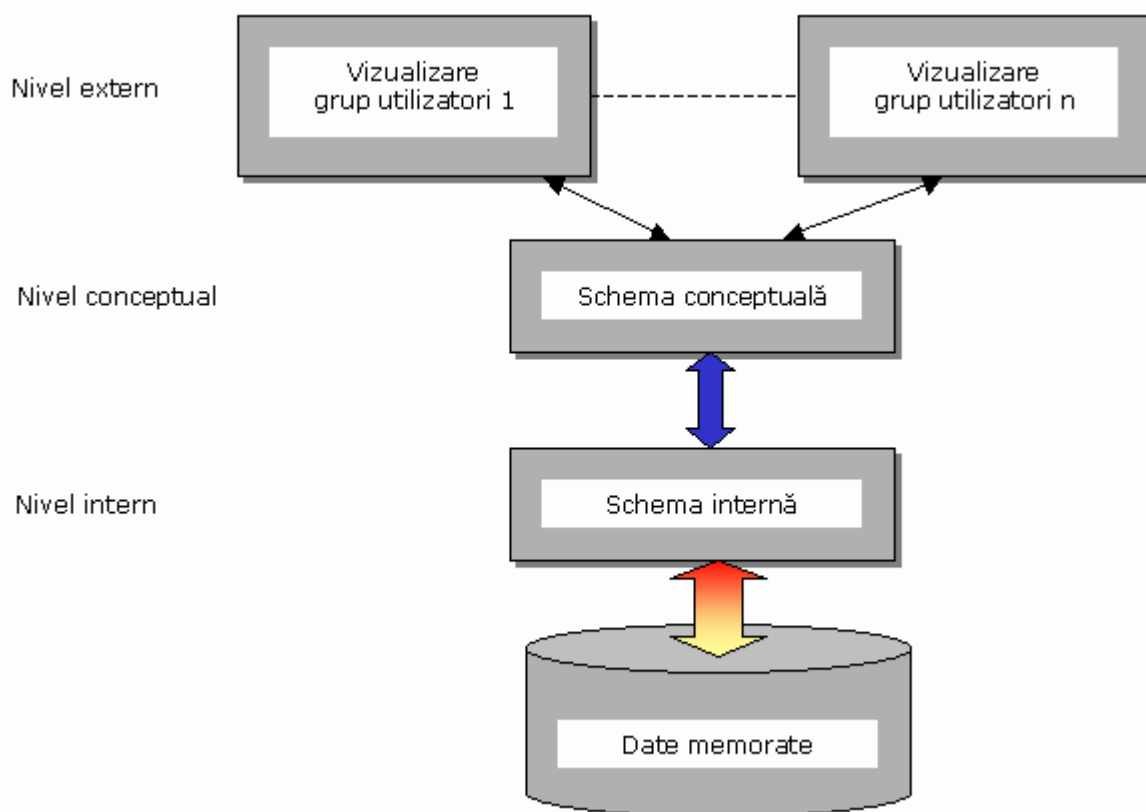


Fig. 2

- *Nivelul extern* – sau nivelul vizual (utilizator), include o colecție de scheme externe ce descriu baze de date prin prisma diferiților utilizatori.

În multe SGBD nu se poate face o distincție clară între cele trei niveluri, de multe ori nivelul conceptual este puternic dezvoltat și suplinește aparent celelalte niveluri. De asemenea, la dezvoltarea aplicațiilor se remarcă o contopire a nivelului extern cu cel conceptual.

De la modelul conceptual, cererile sunt adresate modelului intern pentru a fi procesate și aplicate datelor stocate. Procesul de transfer a cererilor și rezultatelor între nivele se numește *cartografiere* (mapping).

În funcție de categoria de personal implicată în folosirea bazei de date, datele dintr-o bază de date pot fi structurate pe trei niveluri:

- *nivelul conceptual* (global) – exprimă viziunea administratorului bazei de date asupra datelor. Acestui nivel îi corespunde structura conceptuală (schema) a bazei de date, prin care se realizează o descriere a tuturor datelor, într-un mod independent de aplicații, ce face posibilă administrarea datelor.
- *nivelul logic* – exprimă viziunea programatorului de aplicație asupra datelor. La acest nivel se realizează o descriere a datelor corespunzătoare unui anumit program de aplicație.
- *nivelul fizic* – care exprimă viziunea inginerului de sistem asupra datelor. Corespunde schemei interne a bazei de date prin care se realizează o descriere a datelor pe suport fizic de memorie.

1.4. Modelarea la nivel logic a datelor într-o bază de date

Modelul de date reprezintă ansamblul de concepte și instrumente necesare pentru a construi o schemă a bazei de date. Modelarea datelor poate viza totalitatea datelor din cadrul bazei de date (schema/arhitectura datelor) sau o parte a acestora (subscheme ale bazei de date). Schema și subschema bazei de date sunt modelele logice ale bazei de date, care au asociate principii generale pentru gestionarea/definirea (structurarea) datelor, manipularea și asigurarea integrității datelor, fără a reflecta modul de reprezentare și stocare a acestor date pe suportul de memorie (care sunt ele modelului fizic).

Se cunosc mai multe tipuri de baze de date după modul de organizare, modul de dispunere pe suport magnetic a informației și a elementelor componente:

- *modele primitive* – datele sunt organizate la nivel logic în fișiere, structura de bază este înregistrarea, mai multe înregistrări fiind grupate în structuri de tip fișier;
- *baze de date ierarhice* – legăturile dintre date sunt ordonate unic, accesul se face numai prin vârful ierarhiei, un subordonat nu poate avea decât un singur superior direct și nu se poate ajunge la el decât pe o singură cale;
- *baze de date în rețea* – datele sunt reprezentate ca într-o mulțime de ierarhii, în care un membru al ei poate avea oricâți superiori, iar la un subordonat se poate ajunge pe mai multe căi;
- *baze de date relaționale* – structura de bază a datelor este aceea de *relație – tabelă*, limbajul SQL (Structured Query Language) este specializat în comenzi de manipulare la nivel de tabelă. Termenul relațional a fost introdus de un cercetător al firmei IBM, dr. E. F. Codd, în 1969, cel care a enunțat cele 13 reguli de bază necesare pentru definirea unei baze de date relaționale. Baza de date relațională reprezintă o mulțime structurată de date, accesibile prin calculator, care pot satisface în timp minim și într-o manieră selectivă mai mulți utilizatori. Această mulțime de date modelează un sistem sau un proces din lumea reală și servește ca suport unei aplicații informatice;
- *baze de date distribuite* – sunt rezultatul integrării tehnologiei bazelor de date cu cea a rețelelor de calculatoare. Sunt baze de date logic integrate, dar fizic distribuite pe mai multe sisteme de calcul. Integrarea bazei de date distribuite se face cu ajutorul celor trei tipuri de scheme care sunt implementate:
 1. *schema globală* – definește și descrie toate informațiile din baza de date distribuită în rețea;
 2. *schema de fragmentare* – descrie legăturile dintre o colecție globală și fragmentele sale. Ea este de tipul unu la mai mulți și are forma unei ierarhii;
 3. *schema de alocare* – descrie modul de distribuire a segmentelor pe calculatoarele (nodurile) din rețea. Fiecare segment va avea o alocare fizică pe unul sau mai multe calculatoare. Schema de alocare introduce o redundanță minimă și controlată: un anumit segment se poate regăsi fizic pe mai multe calculatoare.

Utilizatorul unei asemenea baze de date o vede ca pe o bază de date unică, compactă (nivel logic), cu toate că în realitate ea este distribuită pe mai multe calculatoare legate în rețea (nivel fizic). Această organizare a dus la o creștere substanțială a vitezei de acces la o bază de date într-o rețea de calculatoare. Anumite date stocate pe un server local sunt mult mai rapid accesate decât dacă ele s-ar afla pe un server la distanță, unde baza de date ar fi fost stocată în întregime (nedistribuită);

- *modele semantice – orientate spre obiecte*. Aceste modele sunt orientate pe reprezentarea semnificației datelor. Structura de bază folosită pentru reprezentarea datelor este cea de clasă de obiecte definită prin abstractizare din entitatea fizică pe care o regăsim în lumea reală. Aici există entități simple și clase de entități care se reprezintă prin obiecte simple sau clase de obiecte, ordonate în ierarhii de clase și subclase. Acest tip de bază de date a apărut din necesitatea gestionării obiectelor complexe: texte, grafice, hărți, imagini, sunete (aplicații multimedia) și a gestionării obiectelor dinamice: programe, simulări.

1.5. Sistem de gestiune a bazelor de date

Sistemele de gestiune a bazelor de date (DBMS – DataBase Management System) sunt sisteme informatice specializate în stocarea și prelucrarea unui volum mare de date, numărul prelucrărilor fiind relativ mic.

Termenul de bază de date se va referi la datele de prelucrat, la modul de organizare a acestora pe suportul fizic de memorare, iar termenul de gestiune va semnifica totalitatea operațiilor ce se aplică asupra datelor din baza de date.

În arhitectura unui sistem de baze de date SGBD ocupă locul central.

Un SGBD este ansamblul software interpus între utilizatori și baza de date și este un interpretor de cereri de acces sau regăsire de date în baza de date, execută cererea și returnează rezultatul. SGBD este un sistem de programe care facilitează procesul definirii, construcției, organizării și manipulării datelor pentru diverse aplicații. Utilizatorul are acces la SGBD prin intermediul unei interfețe (aplicație) cu ajutorul căreia stabilesc parametrii interogării și se primește răspuns; întreg ansamblul este descris în Figura 3.

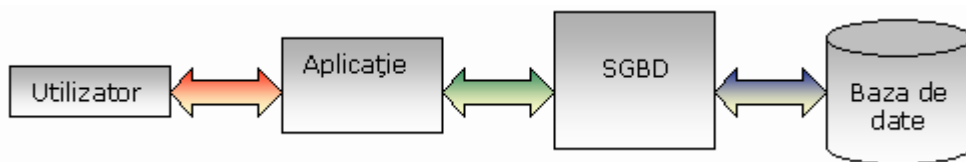


Fig. 3

Organizarea pe trei niveluri a sistemelor de baze de date este strâns legată de conceptul de independență a datelor, în sensul că sistemul bazei de date poate fi modificat la orice nivel fără a afecta nivelurile superioare. Independența datelor poate fi privită în două moduri, corespunzătoare nivelurilor conceptual (logic) și intern (fizic).

Independența logică a datelor reprezintă capacitatea modificării schemei conceptuale fără a provoca modificări în schema externă sau în programele de aplicație. Schema conceptuală se poate modifica prin mărirea bazei de date datorită adăugării de noi tipuri de înregistrări (o nouă coloană într-o tabelă) sau date (înregistrări) sau micșorarea bazei de date în cazul ștergerii unor înregistrări.

Independența fizică a datelor este dată de capacitatea de schimbare a schemei interne fără modificarea schemei conceptuale sau externe.

Funcționarea unui SGBD se realizează prin comenzi specifice limbajului SQL. Nivelele conceptual și intern nefiind distinct delimitate sunt adresate printr-un limbaj comun numit DDL – Data Definition Language, utilizat pentru administrarea și proiectarea bazei de date în definirea ambelor scheme. Dacă SGBD are o delimitare clară între nivelul conceptual și cel intern, atunci DDL se folosește pentru comenzile la nivel conceptual, iar pentru specificarea schemei interne se folosește limbajul SDL – Storage Definition Language. Pentru cel de al treilea nivel, extern, se folosește limbajul VDL – View Definition Language, destinat utilizatorilor și pentru interfața acestora cu nivelul conceptual. Pentru operațiile tipice legate de căutare, inserare, ștergere și modificarea datelor, SGBD dispune de un limbaj de manipulare numit DML - Data Manipulation Language.

1.5.1. Interfețe SGBD

Un SGBD este un ansamblu complex de programe care asigură interfața între o bază de date și utilizatorii acesteia. SGBD este componenta software a unui sistem de baze de date care interacționează cu toate celelalte componente ale acestuia asigurând legătura și interdependența între ele.

Un SGBD oferă interfețele corespunzătoare tuturor categoriilor de utilizatori pentru a facilita legătura acestora cu baza de date. Principalele tipuri de interfețe:

- *Interfețe pe bază de meniuri* – care oferă utilizatorilor o listă de opțiuni (meniuri) pentru formularea interogărilor.
- *Interfețe grafice* – afișează utilizatorului un set de diagrame, cererile sunt formulate prin manipularea acestor diagrame. De cele mai multe ori interfețele grafice sunt asociate cu meniurile.
- *Interfețe bazate pe videoformate* – se utilizează pentru introducerea de noi date, actualizarea bazei de date și căutare.

- *Interfețe în limbaj natural* – acceptă comenzi scrise în limba engleză sau alte limbi de circulație internațională. Interpretarea cererilor se face pe baza unui set de standard de cuvinte cheie ce sunt interpretate pe baza schemei interne.
- *Interfețe specializate pentru cereri repetate* (limbaj de comandă) – sunt destinate unei anumite categorii de utilizatori, de exemplu pentru angajații unei bănci se implementează un mic set de comenzi prescurtate pentru a micșora timpul necesar introducerii comenzii.
- *Interfețe pentru administrarea bazei de date* – se utilizează pentru comenzile privilegiate utilizate de administratorii bazei de date și se referă la crearea de conturi, parole, setarea parametrilor sistemului, autorizarea intrării pe un anumit cont, reorganizarea structurii de stocare a datelor din baza de date, accesul la e, înregistrări.

Exemple de SGBD: MySQL, Microsoft SQL, Microsoft Access, Visual FoxPro, Oracle.

1.5.2. Funcții și servicii oferite de un SGBD

Un SGBD trebuie să asigure funcțiile (Figura 4):

- *funcția de descriere a datelor* – se face cu ajutorul LDD, realizându-se descrierea atributelor din cadrul structurii bazei de date, legăturile dintre entitățile bazei de date, se definesc eventualele criterii de validare a datelor (§5.1.3.), metode de acces la date, integritatea datelor. Concretizarea acestei funcții este schema bazei de date.
- *funcția de manipulare* – este cea mai complexă și realizează actualizarea și regăsirea datelor.
- *funcția de utilizare* – asigură mulțimea interfețelor necesare pentru comunicare a tuturor utilizatorilor cu baza de date.

Categorii de utilizatori:

neinformaticieni – beneficiarii informației, nu trebuie să cunoască structura bazei de date, nu trebuie să programeze aplicații, ci doar să le folosească prin intermediul unei interfețe suficient de prietenoase.

informaticieni – crează structura bazei de date și realizează procedurile complexe de exploatare a bazei de date;

administratorul bazei de date – utilizator special, cu rol hotărâtor în funcționarea optimă a întregului sistem.

- *funcția de administrare* – administratorul este cel care realizează schema conceptuală a bazei de date, iar în perioada de exploatare a bazei de date autorizează accesul la date, reface baza în caz de incident.
- *funcția de protecție a bazei de date* – ansamblul de măsuri necesare pentru asigurarea integrității (semantică, acces concurrent, salvare/restaurare) și securității datelor (autorizare acces, utilizare viziuni, criptare).

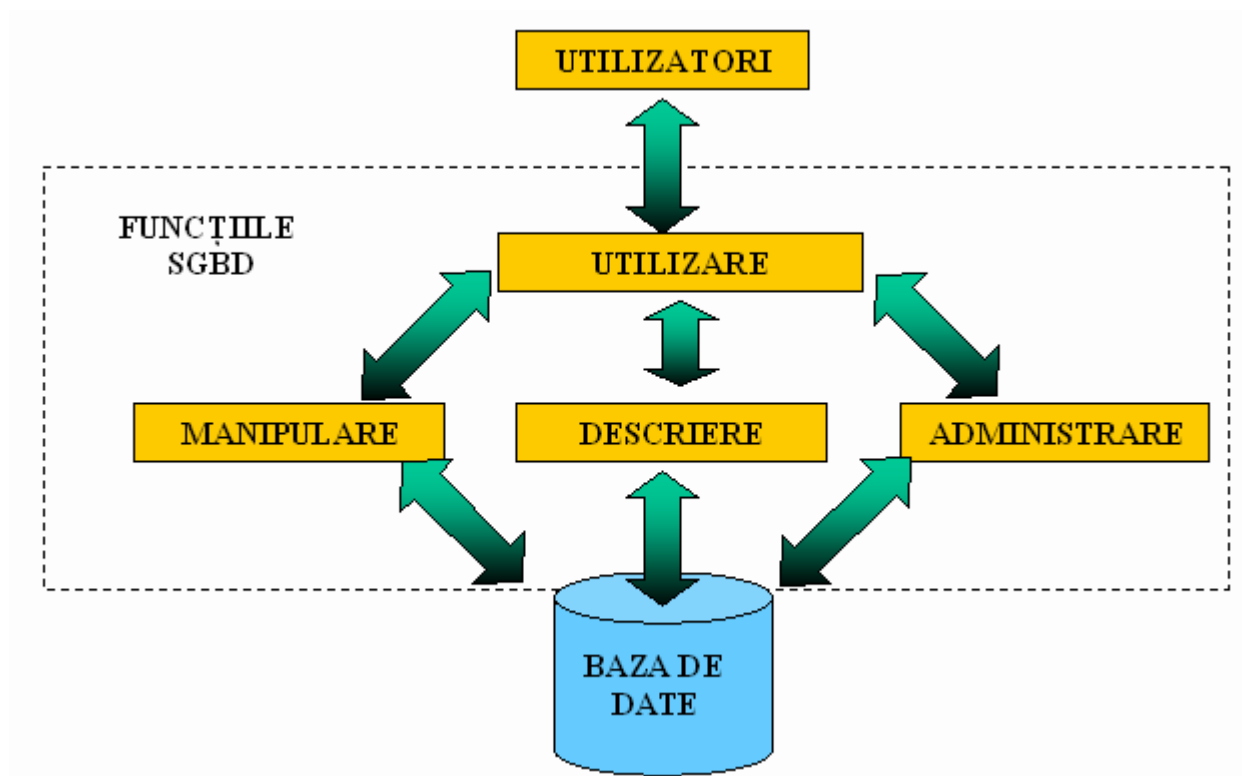


Fig. 4

Prin acestea, un SGBD trebuie să asigure:

- *definirea – crearea bazei de date;*
- *introducerea (adăugarea) datelor în baza de date;*
- *modificarea structurii sau a unor date deja existente în baza de date;*
- *ștergerea datelor din baza de date;*
- *consultarea bazei de date – interogare/extragerea datelor.*

În plus un SGBD mai asigură și alte servicii:

- *suport pentru limbaj de programare;*
- *interfață cât mai atractivă pentru comunicarea cu utilizatorul;*
- *tehnici avansate de memorare, organizare, accesare a datelor din baza de date;*
- *utilitare încorporate: sistem de gestiune a fișierelor, listelor, tabelor etc.;*

- “*help*” pentru ajutarea utilizatorului în lucrul cu baza de date.

Apariția și răspândirea rețelelor de calculatoare a dus la dezvoltarea SGBD-urilor în direcția *multiuser*: mai mulți utilizatori folosesc simultan aceeași bază de date.

Principalul avantaj al rețelelor a fost eficiența mult sporită de utilizare a resurselor sistemelor de calcul: la o bază de date aflată pe un server central au acces simultan mai mulți utilizatori, situați la distanță de server, de unde rezultă o bună utilizare a resurselor server-ului și o economie de memorie datorată memorării unice a bazei de date.

Un SGBD este dotat cu un *limbaj neprocedural de interogare a bazei de date – SQL* – care permite accesul rapid și comod la datele stocate în baza de date.

1.5.3. Activitățile asigurate de SGBD

Un SGBD trebuie să asigure următoarele activități:

- *definirea și descrierea structurii bazei de date* – se realizează printr-un limbaj propriu, limbaj de definire a datelor (LDD), conform unui anumit model de date;
- *încărcarea datelor în baza de date* – se realizează prin comenzi în limbaj propriu, limbaj de manipulare a datelor (LMD);
- *accesul la date* – se realizează prin comenzi specifice din limbajul de manipulare a datelor. Accesul la date se referă la operațiile de *interogare și actualizare*.

Interogarea este complexă și presupune vizualizarea, consultarea, editarea de situații de ieșire (rapoarte, liste, regăsiri punctuale).

Actualizarea presupune trei operațiuni: *adăugare*, *modificare* (efectuate prin respectarea restricțiilor de integritate ale bazei de date) și *ștergere*;

- *întreținerea bazei de date* – se realizează prin utilitare proprii ale SGBD;
- *reorganizarea bazei de date* – se face prin facilități privind actualizarea structurii bazei de date și modificarea strategiei de acces. Se execută de către administratorul bazei de date;
- *securitatea datelor* – se referă la asigurarea *confidențialității* datelor prin autorizarea și controlul accesului la date, *criptarea datelor*, *realizarea de copii ale programelor și fișierelor de bază*.

1.5.4. Obiectivele unui SGBD

Un SGBD are rolul de a furniza suportul software complet pentru dezvoltarea de aplicații informatice cu baze de date.

El trebuie să asigure:

- *minimizarea costului de prelucrare a datelor,*
- *reducerea timpului de răspuns,*
- *flexibilitatea aplicațiilor și*
- *protecția datelor.*

Pentru satisfacerea performanțelor enumerate, SGBD trebuie să asigure un minim de *obiective*.

1. *Asigurarea independenței datelor* – trebuie privită din două puncte de vedere:
 - *independența logică* – se referă la posibilitatea adăgării de noi tipuri de înregistrări de date sau extinderea structurii conceptuale, fără a determina rescrierea programelor de aplicație;
 - *independența fizică* – modificarea tehnicilor fizice de memorare fără a determina rescrierea programelor de aplicație.
2. *Asigurarea redundanței minime și controlate a datelor* – stocarea informațiilor în bazele de date se face astfel încât datele să nu fie multiplicat. Totuși, pentru a îmbunătăți performanțele legate de timpul de răspuns, se acceptă o anumită redundanță a datelor, controlată, pentru a asigura coerența bazei de date și eficiența utilizării resurselor hardware.
3. *Asigurarea facilităților de utilizare a datelor* – presupune ca SGBD-ul să aibă anumite componente specializate pentru:
 - *folosirea datelor de către mai mulți utilizatori în diferite aplicații* – datele de la o aplicație trebuie să poată fi utilizate și în alte aplicații.
 - *accesul cât mai simplu al utilizatorilor la date* – fără ca ei să fie nevoiți să cunoască structura întregii baze de date; această sarcină cade în seama administratorului bazei de date.
 - *existența unor limbaje performante de regăsire a datelor* – care permit exprimarea interactivă a unor cereri de regăsire a datelor.
 - *sistemul de gestiune trebuie să ofere posibilitatea unui acces multicriterial la informațiile din baza de date* – spre deosebire de sistemul clasic de prelucrare pe fișiere unde există un singur criteriu de adresare, cel care a stat la baza organizării fișierului.
4. *Asigurarea securității datelor împotriva accesului neautorizat.*

5. *Asigurarea coerenței și integrității datelor împotriva unor ștergeri intenționate sau neintenționate* – se realizează prin intermediul unor proceduri de validare, a unor protocoale de control concurrent și a unor proceduri de refacere a bazei de date.
6. *Asigurarea partajabilității datelor* – se referă pe de o parte la asigurarea accesului mai multor utilizatori la aceleași date și de asemenea la posibilitatea dezvoltării unor aplicații fără a se modifica structura bazei de date.
7. *Asigurarea legăturilor între date* – corespund asocierilor care se pot realiza între obiectele unei aplicații informatice. Orice SGBD trebuie să permită definirea și descrierea structurii de date, precum și a legăturilor dintre acestea, conform unui model de date (de exemplu, modelul relațional).
8. *Administrarea și controlul datelor* – sunt asigurate de SGBD, în sensul că datele pot fi folosite de mai mulți utilizatori în același timp, iar utilizatorii pot avea cerințe diferite și care pot fi incompatibile. SGBD trebuie să rezolve probleme legate de concurență la date, problemă care apare mai ales în lucrul în mediu de rețea de calculatoare.

2. Etapele de realizare a unei bănci de date

2.1. Etapa de analiză

Proiectarea unei baze de date constă din proiectare logică și fizică a acesteia astfel încât să răspundă cerințelor utilizatorilor pentru un anumit set de aplicații specifice. În general, proiectarea corectă a unei baze de date parcurge etapele de analiză și programare. Pentru etapa de analiză se recomandă parcurgerea pașilor:

1. *Studierea problemei de rezolvat* – constă din studiul și descrierea activităților pentru care se va organiza baza de date (cerințe și resurse);
2. *Proiectarea structurii bazei de date* – are ca principale activități alegerea modelului de SGBD, proiectarea funcțiilor bazei de date, stabilirea intrărilor și ieșirilor aplicațiilor;
3. *Stabilirea modului de memorare a datelor* – în memorie (temporare) sau pe disc în baza de date, pentru care trebuie stabilită și structura;
4. *Stabilirea algoritmului general de rezolvare a problemei* – la nivel de schemă bloc;
5. *Stabilirea structurii meniului principal al aplicației* – acesta trebuie să conțină opțiuni pentru toate funcțiunile sistemului informatic care sunt accesibile utilizatorului.
6. *Împărțirea aplicației pe programe* – fiecare program urmărește rezolvarea unei părți a problemei generale.

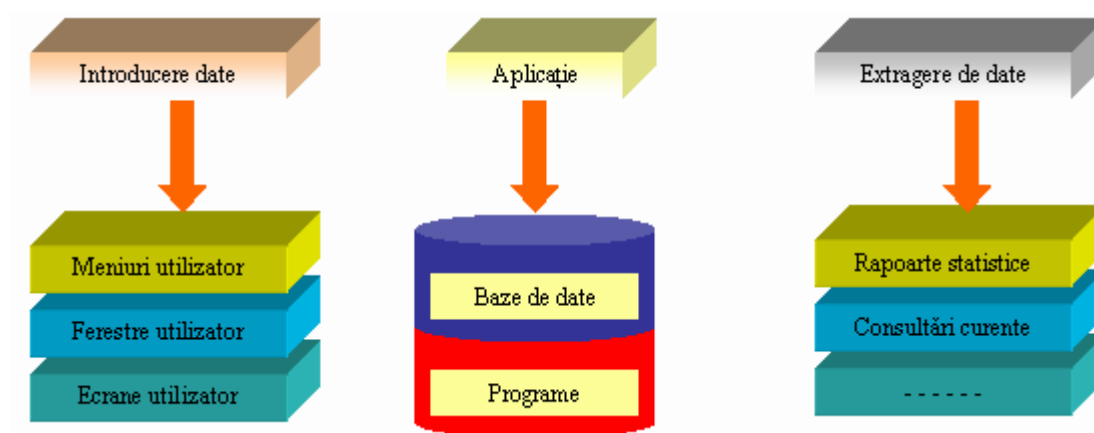
2.2. Etapa de programare

În etapa de programare se recomandă urmărirea pașilor:

1. *Elaborarea programelor*: realizarea componentelor logice – programe/aplicații – începând cu programul monitor (principal), continuând cu cele de introducere/actualizare a datelor, cu cele de prelucrare și terminând cu programele de interogare baza de date și extragere de informații.
2. *Testarea funcționării corecte a programelor individuale și a sistemului în ansamblu* – pentru cât mai multe situații posibile (ideal ar fi testarea în toate situațiile posibile, dar numărul acestora este foarte mare) incluzând în mod obligatoriu și situațiile limită.
3. *Înlăturarea erorilor depistate* – și reluarea pasului 2. Dacă nu mai sunt erori, se continuă cu următoarea etapă.

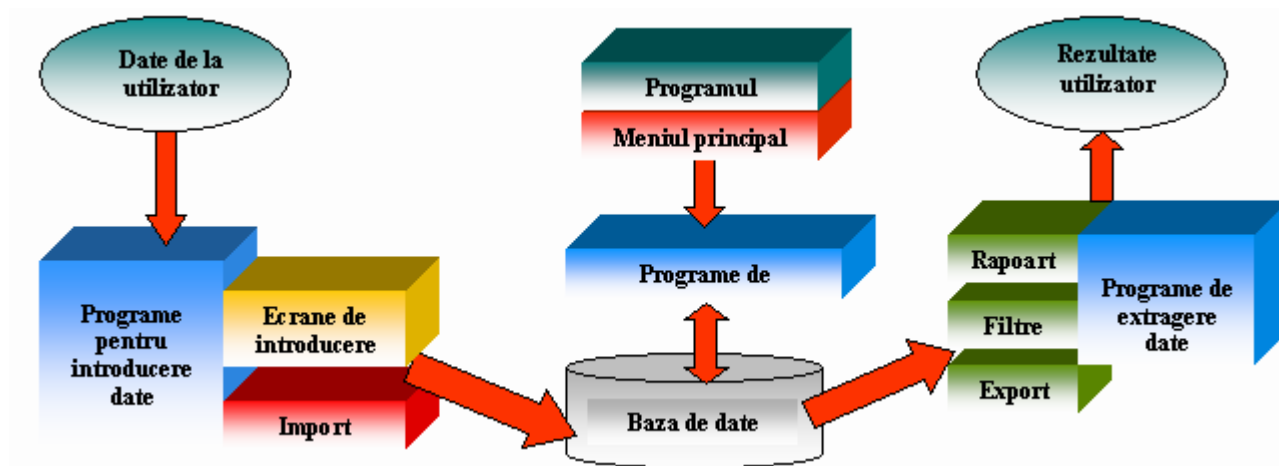
2.3. Punerea în funcțiune și exploatarea bazei de date

1. *Încărcarea și manipularea datelor* – operații de actualizare și consultare, întreținere (dezvoltare) a bazei de date.
2. *Elaborarea documentației de utilizare* – care se predă beneficiarului, cât și documentația tehnică a sistemului (de preferat ca fiecare program să aibă câte o fișă tehnică, pentru o mai ușoară depanare, întreținere).



Schema bloc a unei aplicații din punct de vedere utilizator

Fig. 5



Schema bloc a unei aplicații din punct de vedere al proiectantului

Fig. 6

Notă: Programele de extragere date pot fi elaborate de programator sau pot fi obținute cu generatoarele incluse în SGBD.

Întreaga activitate din sistem este controlată de programul monitor care conține un meniu principal definit de utilizator. Programele aplicației pentru introducerea datelor, prelucrare, actualizare și extragere de date sub formă de rapoarte se află sub controlul programului monitor.

2.4. Documentația aplicației

Deoarece etapa de analiză este proprie fiecărei aplicații în parte, nu se recomandă detalierea etapelor 1 ÷ 6. De aceea în cele ce urmează este detaliată documentarea programelor prin fișa tehnică, fișă care trebuie să conțină rubricile:

Program – nume;

Scop;

Parametri;

Variabile – de intrare, de ieșire;

Ce programe apelează;

Care sunt programele care îl apelează;

Ce fișiere utilizează;

Explicații - sub formă de comentarii în programele sursă;

Programul sursă.

În §12.2.2 se arată cum se obține în mod automat, cu ajutorul VFP, documentarea unei aplicații realizată cu acest sistem.

3. Sisteme de gestiune a bazelor de date relaționale

3.1. Modelul relațional al datelor

Un sistem de gestiune a bazelor de date relaționale se definește ca fiind un sistem de gestiune care utilizează organizarea datelor conform modelului relațional. Conceptul de bază al modelului relațional este acela de relație/tabelă (limbajul SQL specializat în comenzi de manipulare la nivel de tabelă).

3.1.1. Structura relațională a datelor

Structura relațională a datelor cuprinde următoarele componente.

Domeniul – este ansamblul de valori caracterizat printr-un nume (domeniu de valori). El poate fi precizat explicit – prin enumerarea tuturor valorilor care aparțin domeniului – sau implicit prin precizarea proprietăților pe care le au valorile din domeniu. Pentru un ansamblu de valori D_1, \dots, D_n produsul cartezian al acestora reprezintă ansamblul tuplurilor $\langle v_1, \dots, v_n \rangle$, unde v_1 este o valoare din D_1 , v_2 este o valoare din D_2 , etc., iar tuplura corespunde unei linii din tabelă.

Relația (tabela) – este un subansamblu al produsului cartezian al mai multor domenii, caracterizat prin nume și care conține tupluri cu semnificație (tabelă). Într-o relație, tuplurile trebuie să fie distincte – nu se admit duplicate. O reprezentare a relației este tabelul bidimensional (tabela de date) în care liniile reprezintă tuplurile, iar coloanele corespund domeniilor. Numărul tuplurilor dintr-o tabelă este *cardinalul tablei*, numărul valorilor dintr-un tuplu este *gradul tablei*. Pentru a diferenția coloanele care conțin valori ale aceluiași domeniu, eliminând dependența de poziție, se asociază fiecărei coloane un nume distinct – *atribut*. În timp ce tuplurile sunt unice, un domeniu poate apărea de mai multe ori în produsul cartezian pe baza căruia este definită tabela.

Tabelul 1 prezintă comparativ conceptele utilizate în organizarea datelor în fișiere, în baze de date, conceptele din teoria relațională și din SGBDR.

Tabelul 1

Fișiere	Teoria BD	Teoria relațională	SGBDR
Fișier	Colecție de date	Relație	Tabelă
Înregistrare	Familie de caracteristici	Tuplu	Linie
Câmp	Caracteristică	Atribut	Nume coloană
Valoare	Domeniu de Valori	Domenii	Domeniu coloană

3.1.2. Operații în algebra relațională

Reuniunea – operație în algebra relațională, definită pe două relații R_1 și R_2 , ambele cu aceeași schemă (structură), ce constă din construirea unei noi relații R_3 , cu schema identică cu R_1 și R_2 , având drept extensie (înregistrările) tuplurile din R_1 și R_2 luate împreună, o singură dată.

Diferența – definită pe două relații R_1 și R_2 , cu aceeași schemă, având drept extensie tupluri ale relației R_1 care nu se regăsesc în relația R_2 .

Produs cartezian – definit pe două relații R_1 și R_2 , operație care constă din construirea unei noi relații R_3 , a cărei schemă se obține din concatenarea schemelor relațiilor R_1 și R_2 , și a cărei extensie cuprinde toate combinațiile tuplurilor din R_1 cu cele din R_2 .

Proiecția – operație din algebra relațională definită asupra unei relații R , ce constă din construirea unei noi relații P , în care se regăsesc numai acele atribute din R specificate explicit în cadrul operației. Suprimarea unor atribute din R înseamnă efectuarea unor tăieturi verticale asupra lui R și pot avea ca efect apariția unor tupluri duplicate, care se cer a fi eliminate. În practică, operația de proiecție se face considerând și un câmp care conține o cheie unică, astfel încât fiecare înregistrare devine unică.

Selecția – operație în algebra relațională definită asupra unei relații R , care constă din construirea unei relații S , a cărei schemă este identică cu cea a relației R și a cărei extensie este constituită din acele tupluri din R care satisfac o condiție explicită în cadrul relației. Deoarece nu toate tuplurile din R satisfac condiția, selecția înseamnă efectuarea de tăieturi pe orizontală asupra relației R , adică eliminarea de tupluri. Condiția precizată în cadrul operației de selecție este de forma:

<Atribut> <operator de comparație> <Valoare>

Joncțiunea (join-ul) – operație în algebra relațională definită pe două relații R_1 și R_2 , care constă din construirea unei noi relații R_3 , prin concatenarea unor tupluri din R_1 cu tupluri din R_2 . Se concatenează acele tupluri din R_1 și R_2 care satisfac o anumită condiție, specificată explicit în cadrul operației. Extensia relației R_3 va conține combinațiile acelor tupluri care satisfac condiția de concatenare. Cel mai important tip de join din punct de vedere al utilizării este *equijoin-ul*, care este o joncțiune dirijată de o condiție de forma:

$$\langle \text{Atribut din } R_1 \rangle = \langle \text{Atribut din } R_2 \rangle$$

Joncțiunea naturală se referă la equijoin-ul aplicat tuturor câmpurilor comune. Operatorul pentru joncțiunea naturală este $*$.

Joncțiunea exterioară (outer join) include trei tipuri de joncțiune:

- *Joncțiunea exterioară stânga* \bowtie_{\leftarrow} care include toate tuplurile din partea stângă a relației, iar în partea dreaptă a relației include numai acele tupluri care au corespondent.
- *Joncțiunea exterioară dreapta* \bowtie_{\rightarrow} care include toate tuplurile din partea dreaptă a relației, iar în partea stângă a relației include numai acele tupluri care au corespondent.
- *Joncțiunea exterioară completă* \bowtie_{full} care include toate tuplurile din stânga și din dreapta relației.

Intersecția – operație din algebra relațională definită pe două relații R_1 și R_2 , ambele cu aceeași schemă, care constă din construirea unei noi relații R_3 , cu schema identică cu a operanzilor și cu extensia formată din tuplurile din R_1 și R_2 .

Diviziunea – operație din algebra relațională definită asupra unei relații R , care constă din construirea cu ajutorul unei relații r a relației Q . Tuplurile relației Q concatenate cu tuplurile relației r permit obținerea tuplurilor relației R .

Exemple

Pentru a înțelege mai bine aceste operații să presupunem că avem următoarele trei structuri de date:

1. $Y1$ – fișier închirieri mașini

Sofer_id	Auto_id	Ziua
22	101	10/10/2004
58	103	11/12/2004

2. $S2$ – fișier clienți

Sofer_id	Sofer_num	Permis_an
22	Ionescu Paul	7
31	Leahu Mihai	8
58	Arsene Ion	10

3. $S3$ – fișier clienți

Sofer_id	Sofer_num	Permis_an
28	Doncea Horia	5
31	Leahu Mihai	8
44	Coman Stefan	12
58	Arsene Ion	10

Semnificația structurilor de date este următoarea:

- în relația $Y1$ se stochează informații referitoare la mașinile rezervate cu attributele: $Sofer_id$ – codul șoferului, $Auto_id$ – codul mașinii închiriate, $Ziua$ – data la care a fost închiriată;
- în relațiile $S2$ și $S3$ se stochează informații referitoare la șoferi (fragmentat), cu următoarele attribute: $Sofer_id$ – codul șoferului (pentru a stabili o legătură între structura de date $Y1$ și $S2$, $S3$), $Sofer_num$ – numele șoferului, $Permis_an$ – vechimea permisului de conducere.

Principalele operații:

- *Reuniunea* $R3 = S2 \cup S3$

Sofer_id	Sofer_num	Permis_an
22	Ionescu Paul	7
31	Leahu Mihai	8
58	Arsene Ion	10
44	Coman Stefan	12
28	Doncea Horia	5

- *Intersecția* $R3 = S2 \cap S3$

Sofer_id	Sofer_nume	Permis_ani
31	Leahu Mihai	8
58	Arsene Ion	10

- *Diferența* $S2 - S3$

Sofer_id	Sofer_nume	Permis_ani
22	Ionescu Paul	7

- *Produs cartezian*, $Y1 \times S2$. Fiecare linie din relația $Y1$ se combină cu fiecare linie din relația $S2$; ca rezultat al produsului cartezian, apare un conflict deoarece ambele relații $Y1$ și $S2$ au câte un câmp cu numele *sofer_id*, care poate fi rezolvat prin operația de *redenumire*.



Sofer_id_a	Auto_id	Ziua	Sofer_id_b	Sofer_nume	Permis_ani	
22	101	10.10.04	22	Ionescu Paul	7	
22	101	10.10.04	31	Leahu Mihai	8	
22	101	10.10.04	58	Arsene Ion	10	
58	103	11.12.04	22	Ionescu Paul	7	
58	103	11.12.04	31	Leahu Mihai	8	
58	103	11.12.04	58	Arsene Ion	10	

Comanda SQL pentru produs cartezian $Y1 \times S2$ este dată în Figura 1.

Operatorul redenumire se utilizează sub forma:

$$\rho(C(1 \rightarrow \text{Sofer_id1}, 5 \rightarrow \text{Sofer_id2}), Y1 \times S2)$$

cu următoare interpretare: se redenumesc coloana 1 în *Sofer_id1* și coloana 5 în *Sofer_id2*, în relația dată de produsul cartezian obținut din relațiile $Y1$ și $S2$.

- *Proiecția*. Deoarece valorile dintr-un domeniu se pot repeta este necesară eliminarea repetărilor:

$$\pi_{\text{Sofer_nume}, \text{Permis_ani}}(S3)$$

Sofer_nume	Permis_ani
Doncea Horia	5
Leahu Mihai	8
Coman Stefan	12
Arsene Ion	10

- *Selecția* $\sigma_{\text{Permis_ani} > 8}(S3)$

Sofer_id	Sofer_nume	Permis_ani
44	Coman Stefan	12
58	Arsene Ion	10

Din relația $S3$ au fost selectate numai acele înregistrări care satisfac condiția de vechime a permisului (mai mult de 8 ani).

Operația de proiecție se poate combina cu cea de selecție pentru optimizarea cererilor de date; de exemplu dacă dorim extragerea datelor din relația $S3$ numai pentru câmpurile Sofer_nume și Permis_ani mai mare de 8 ani:

$$\pi_{\text{Sofer_nume}, \text{Permis_ani}}(\sigma_{\text{Permis_ani} > 8}(S3))$$

Sofer_nume	Permis_ani
Coman Stefan	12
Arsene Ion	10

Joncțiunea (join-ul). Operația de joncțiune condiționată a două relații este echivalentă cu selecția condiționată a produsului cartezian al celor două relații.

$$S2 \bowtie_{S2.\text{Sofer_id} < Y1.\text{Sofer_id}} Y1$$

(Sofer_id)	Sofer_nume	Permis_ani	(Sofer_id)	Auto_id	Ziua
22	Ionescu Paul	7	58	103	11/12/2004
31	Leahu Mihai	8	58	103	11/12/2004

Operația de joncțiune condiționată mai poartă uneori și numele de *theta-join*.

Pentru acest exemplu, *equijoin*-ul, caz special al joncțiunii condiționate în care condiția *c* conține doar egalități, devine

$$S2 \bowtie_{\text{Sofer_id}} Y1$$

Sofer_id	Sofer_nume	Permis_ani	Auto_id	Ziua
22	Ionescu Paul	7	101	10/10/2004
58	Arsene Ion	10	103	11/12/2004

Noua relație rezultată din *equijoin* este similară cu produsul cartezian dintre *S2* și *Y1*, dar numai o copie a câmpului pentru care este specificată egalitatea, va fi specificată în noua relație (nu mai apare duplicarea câmpului).

Exemple de *joncțiuni exterioare*

- joncțiune exterioară stânga

$$\ltimes Y1 \quad Y1.\text{Sofer_id}=S2.\text{Sofer_id} \quad S2$$

Sofer_id_a	Auto_id	Ziua	Sofer_id_b	Sofer_nume	Permis_ani
22	101	10.10.04	22	Ionescu Paul	7
58	103	11.12.04	58	Arsene Ion	10

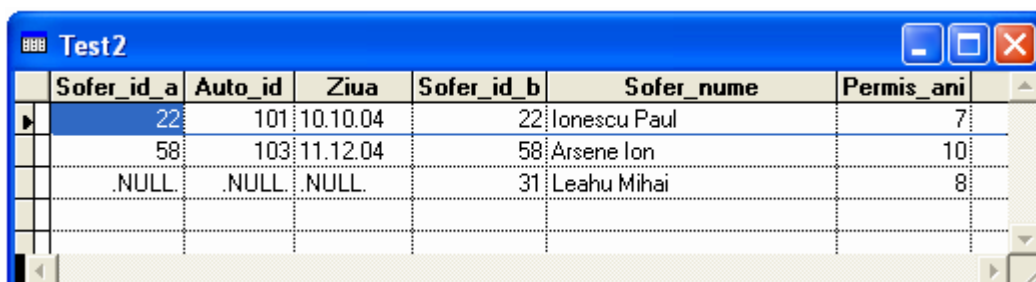
- joncțiune exterioară dreapta

$$\rtimes Y1 \quad Y1.\text{Sofer_id}=S2.\text{Sofer_id} \quad S2$$

Sofer_id_a	Auto_id	Ziua	Sofer_id_b	Sofer_nume	Permis_ani
22	101	10.10.04	22	Ionescu Paul	7
.NULL.	.NULL.	.NULL.	31	Leahu Mihai	8
58	103	11.12.04	58	Arsene Ion	10

- joncțiune exterioară completă

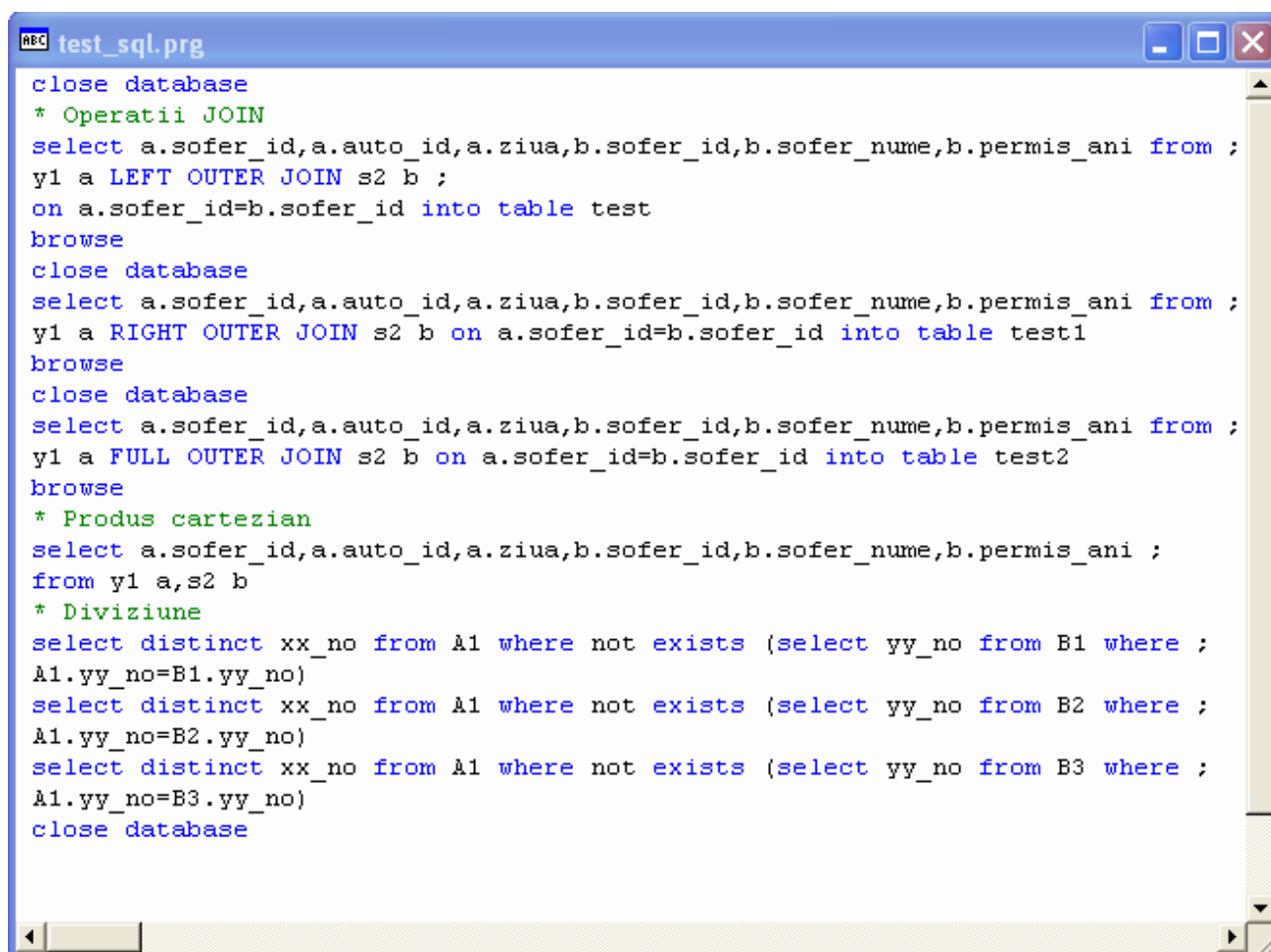
$$\ltimes \rtimes Y1 \quad Y1.\text{Sofer_id}=S2.\text{Sofer_id} \quad S2$$



Sofer_id_a	Auto_id	Ziua	Sofer_id_b	Sofer_nume	Permis_an
22	101	10.10.04	22	Ionescu Paul	7
58	103	11.12.04	58	Arsene Ion	10
NULL	NULL	NULL	31	Leahu Mihai	8

De remarcat prezența câmpurilor cu valoarea **NULL** utilizată în joncțiuni pentru a specifica lipsa de date.

Presupunând că tabele au fost create, programul sursă pentru testarea celor trei tipuri de joncțiuni exterioare este prezentat în Figura 1.



```

close database
* Operatii JOIN
select a.sofer_id,a.auto_id,a.ziua,b.sofer_id,b.sofer_nume,b.permis_an from ;
y1 a LEFT OUTER JOIN s2 b ;
on a.sofer_id=b.sofer_id into table test
browse
close database
select a.sofer_id,a.auto_id,a.ziua,b.sofer_id,b.sofer_nume,b.permis_an from ;
y1 a RIGHT OUTER JOIN s2 b on a.sofer_id=b.sofer_id into table test1
browse
close database
select a.sofer_id,a.auto_id,a.ziua,b.sofer_id,b.sofer_nume,b.permis_an from ;
y1 a FULL OUTER JOIN s2 b on a.sofer_id=b.sofer_id into table test2
browse
* Produs cartezian
select a.sofer_id,a.auto_id,a.ziua,b.sofer_id,b.sofer_nume,b.permis_an ;
from y1 a,s2 b
* Diviziune
select distinct xx_no from A1 where not exists (select yy_no from B1 where ;
A1.yy_no=B1.yy_no)
select distinct xx_no from A1 where not exists (select yy_no from B2 where ;
A1.yy_no=B2.yy_no)
select distinct xx_no from A1 where not exists (select yy_no from B3 where ;
A1.yy_no=B3.yy_no)
close database

```

Fig. 1

Diviziunea –Se utilizează la exprimarea interogărilor de tipul: “Găsește automobilul care a rezervat toate automobilele”.

$$A/B = \langle x \rangle \mid \neg \exists \langle x, y \rangle \in A, \forall \langle y \rangle \in B$$

relația obținută prin diviziune va conține toate tuplurile x care pentru fiecare tuplură y din B nu are asociată un tuplu xy în A . Pentru a înțelege mai bine operația de diviziune vom considera următorul exemplu, în care considerăm tabelul $A1$ (câmpuri xx_no , yy_no) și tablele $B1$, $B2$, $B3$ (un singur câmp yy_no , corespunzător cu cel din tabelul A) și realizăm operațiile de diviziune $A1/B1$, $A1/B2$, $A1/B3$:

Xx_no	Yy_no
X1	Y1
X1	Y2
X1	Y3
X1	Y4
X2	Y1
X2	Y2
X3	Y2
X4	Y2
X4	Y4

Yy_no
Y2

Yy_no
Y2
Y4

Yy_no
Y1
Y2
Y4

Secvența de program care realizează cele trei operații de diviziune se găsește în figura 1, ultima secțiune **Diviziune*. Rezultatul celor trei comenzi SQL este:

Xx_no
X1
X2
X4

Xx_no
X1
X2

Xx_no
X1

A1/B1

A1/B2

A1/B3

3.1.3. Optimizarea cererilor de date

Se realizează în două etape:

1. exprimarea cererilor de date sub forma unor expresii algebrice relaționale care au la bază echivalența dintre calculul relațional și algebra relațională,

2. aplicarea unor transformări algebrice asupra expresiilor obținute în etapa precedentă, în scopul obținerii unor expresii echivalente cu cele inițiale, dar care să fie executate mai eficient.

Proprietăți

Comutativitatea operațiilor de join și produs cartezian:

$$E_1 \bowtie E_2 = E_2 \bowtie E_1$$

$$E_1 \times E_2 = E_2 \times E_1$$

Asociativitatea operațiilor de join și produs cartezian:

$$(E_1 \bowtie E_2) \bowtie E_3 = E_1 \bowtie (E_2 \bowtie E_3)$$

$$(E_1 \times E_2) \times E_3 = E_1 \times (E_2 \times E_3)$$

Compunerea proiecțiilor

Compunerea selecțiilor $\sigma_{F_1}(\sigma_{F_2}(E)) = \sigma_{F_1 \cap F_2}(E)$, deoarece $F_1 \cap F_2 = F_2 \cap F_1$, selecțiile se pot comuta:

$$\sigma_{F_1} \circ \sigma_{F_2}(E) = \sigma_{F_2} \circ \sigma_{F_1}(E)$$

Comutarea selecției cu proiecția

Comutarea selecției cu produsul cartezian

Comutarea selecției cu reuniunea

Comutarea selecției cu diferența

Comutarea proiecției cu produsul cartezian

Comutarea proiecției cu reuniunea

Prin deplasarea operațiilor de selecție cât mai la stânga expresiilor algebrice se reduce numărul de tupluri care trebuie manipulate în procesul de executare a cererii.

Se pot menționa următoarele strategii de optimizare a cererilor de date:

- Deplasarea operației de selecție înaintea operației de joncțiune – joncțiunea și produsul cartezian acționează ca generatori de tupluri. Prin selecție se reduce dimensiunea relațiilor la care se aplică acești generatori de tupluri. Se ține seama că operația de joncțiune poate fi exprimată sub forma unui produs cartezian urmat de o selecție, iar în cazul joncțiunii naturale printr-un produs cartezian urmat de o selecție și de o proiecție.
- Deplasarea operațiilor de proiecție înaintea operațiilor de joncțiune – se realizează prin folosirea proprietății de comutare a selecției cu produsul cartezian
- Combinarea selecțiilor multiple – se realizează cu ajutorul relației de compunere a selecțiilor.

- Deplasarea operațiilor de selecție înaintea operațiilor de proiecție- realizată pe baza proprietății de comutare a selecției cu proiecția. Eliminarea tuplurilor duplicate obținute prin proiecție se face prin ordonarea tuplurilor. Selecția reduce numărul tuplurilor ce trebuiesc ordonate facilitând operația de proiecție.

3.1.4. Baze de date relaționale

Conceptul bazelor de date relaționale este axat pe metodologia entitate-tabelă (*E-T*).

Entitățile modelează obiectele care sunt implicate într-o organizație, de exemplu studenții, profesorii și cursurile dintr-o universitate.

Tabelele modelează legăturile dintre entități, de exemplu profesorii predau cursuri. În plus, restricțiile de integritate aplicate entităților și relațiilor formează o parte importantă a specificațiilor *E-T*, de exemplu, un profesor poate predă un singur curs la o anumită oră dintr-o anumită zi.

Entitățile similare pot fi agregate în tipuri de entități. De exemplu, Ionescu, Popescu, Albu, pot fi agregați în tipul de entitate *PERSOANĂ*, pe baza faptului că aceste entități sunt oameni. Ionescu și Albu pot aparține tipului de entitate *STUDENT*, deoarece aceste obiecte sunt studenți.

La fel ca tabelele, entitățile sunt descrise utilizând atribute. Fiecare atribut specifică o particularitate semnificativă a entității. De exemplu, atributul *Nume* al unei entități de tip *PERSOANĂ* specifică șirul de caractere care alcătuiește numele persoanei din lumea reală. De asemenea, *Vârsta* este atributul care specifică de câte ori Pământul a înconjurat Soarele față de momentul în care o anumită persoană s-a născut. Pentru fiecare atribut, se asociază un domeniu care specifică setul de valori pe care le poate lua atributul. În principiu, este posibil ca două entități diferite ale aceluiași tip să posede valori identice pentru toate atributele.

Atributelor li se asociază valori care au drept scop identificarea entității, realizându-se o înregistrare în tabela respectivă. Atribute pot fi:

- *complexe* - cele care pot fi divizate în mai multe părți cu semnificație independentă. De exemplu, atributul *Adresa* poate fi divizat în mai multe atribute: *Oraș*, *Cod poștal*, *Stradă*, *Număr*, *Bloc*, *Etaj*, *Nr_Apartament*. Una din cerințele importante referitoare la valorile din domeniu se referă la *atomicitatea datelor*. Atomicitatea datelor nu înseamnă că aceste valori nu se pot descompune, datele pot fi șiruri de caractere, ceea ce înseamnă că pot fi descompuse. Cerința de atomicitate se referă la faptul că modelul relațional nu specifică nici un mijloc pentru a privi în interiorul structurii valorilor, astfel că valorile sunt invizibile pentru operatorii relaționali;

- *cu o singură valoare* – din setul de valori ale atributului pentru fiecare entitate există numai o singură valoare, nerepetabilă la o altă entitate. De exemplu, *CNP* (Codul Numeric Personal) pentru o persoană este unic și nici o altă persoană nu mai are atribuită aceeași valoare a *CNP*;

- *cu set de valori* – atributul poate lua orice valoare din set și care se poate repeta pentru o altă entitate. Este cazul culorilor (două mărci de mașină diferite pot avea aceeași culoare) sau gradul cadrelor didactice;

- *derivate* – sunt cele ce se pot determina din alte atribute, de exemplu vârsta unei persoane se poate determina scăzând din data curentă data nașterii persoanei respective, sau determinarea valorii unei entități prin produsul dintre prețul unitar și cantitate. Prezența atributelor derivate împreună cu cele primare cresc gradul de redundanță al datelor din baza de date și trebuie evitată;

- *fără valoare (NULL)* – bazele de date acceptă și posibilitatea lipsei datelor pentru câmpurile anumitor înregistrări, de exemplu, din atributul *Adresă*, poate lipsi *Nume_blocului* sau *Nr_Scara*.

Cheile de restricție introduse în modelul relațional sunt asociate cu tipurile de entități, de exemplu, două persoane diferite nu pot avea aceeași valori pentru atributele *Nume* și *Adresa*.

Schema în modelul relațional se definește ca fiind un tip de entitate compus din numele tipului de entitate, colecția de atribute (cu domeniile asociate și indicații în cazul în care un atribut este complex sau cu valori unice) și cheile de restricție.

Reprezentarea diagramelor E-T – tipurile de entități reprezentate în diagramele *E-T* sunt sub formă de dreptunghiuri și atributele sub formă de elipse. Atributele tip set de de valori (modelul relațional permite atributelor să posede un set de valori din domeniu) sunt reprezentate în elipse duble. Sublinierea unui atribut în diagramă îl desemnează ca fiind cheie primară, adică un atribut cu valoare unică. Figura 2 descrie o diagramă *E-R* pentru entitatea *PERSOANA*:

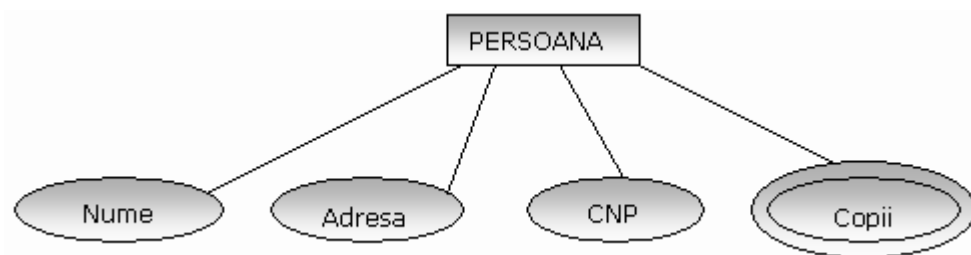


Fig. 2

3.1.5. Tipuri de relații

Relația unu-la-unu (1-1) este tipul cel mai simplu de relație, prin care unui element din tabela *R1* îi corespunde un singur element din tabela *R2* și reciproc. Relația 1-1 este mai rar folosită, în general pentru a reduce numărul de attribute dintr-o tabelă, de exemplu dacă vrem să despărțim elementele fixe care se modifică mai rar (datele personale) față de cele care se modifică mai des (situația școlară anuală).

Relația unu-la-mai mulți (1-N) este tipul de relație prin care unui element din tabela *R1* îi corespund unul sau mai multe elemente din tabela *R2*, iar unui element din tabela *R2* îi corespunde un singur element din tabela *R1*. De exemplu, presupunem că avem:

- tabela *FACULTATE* cu attributele *CodFac*, *Denumire*, *Adresa*, *NumeDecan*;

- tabela *STUDENTI* cu attributele *CodStud*, *CodFac*, *An*, *Grupa*, *Medie*, *Bursa*;

Între cele două tabele există o relație 1-N prin intermediul câmpului *CodFac*.

Relația mai mulți-la-mai mulți (M-N) prin care unui element din tabela *R1* îi corespunde unul sau mai multe elemente din tabela *R2* și reciproc. De exemplu, un student frecventează mai multe cursuri și o disciplină este frecventată de mai mulți studenți. Acest tip de relație este frecvent întâlnită, dar nu se poate implementa direct în bazele de date relaționale. Pentru modelare se folosește o relație suplimentară, de tip 1-N pentru fiecare din tabelele inițiale. Presupunând că avem două tabele *MATERII* și *STUDENTI* se introduce suplimentar tabela *NOTE* care face legătura între tabelele *MATERII* și *STUDENTI*:

- tabela *MATERII* are attributele *CodMaterie*, *Denumire*, *An*, *NumeProfesor*;

- tabela *STUDENTI* cu attributele: *CodStud*, *CodFac*, *An*, *Grupa*, *Medie*, *Bursa*;

- tabela *NOTE* cu attributele: *CodStud*, *CodMaterie*, *Nota*, *Data*.

Relația unară este acea relație care folosește o singură tabelă care este asociată cu ea însăși.

Exemplul clasic este cel al managerului unei companii, care la rândul său este tot un angajat al acestei companii.

Pentru fiecare tip de entitate participantă în cadrul relației se definește un rol care primește un nume pentru tipul relației. În diagramele *E-T*, tipul relației se reprezintă sub forma unui romb. De exemplu, tipul de relație *LUCREAZĂ* cu atributul *Data*, definit în Figura 3, are două roluri *Profesor* și *Departament*:

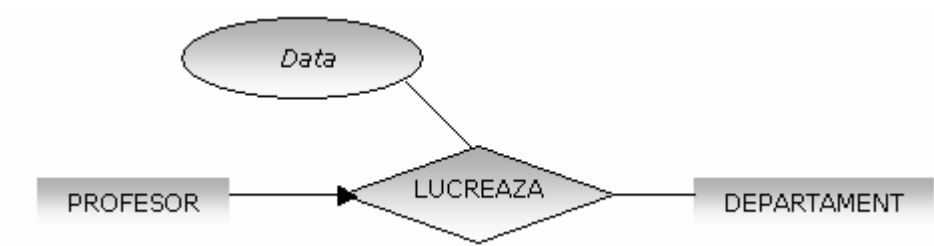


Fig. 3

Rolul *Profesor* identifică entitatea PROFESOR implicată în relația LUCREAZĂ, iar rolul *Departament* identifică entitatea DEPARTAMENT în cadrul relației.

În Figura 4 este prezentată o relație între trei entități: PROIECT (cu rolul *Client*), PARTENER (cu rolul *Produs*), FURNIZOR (cu rolul *Furnizor*), cu tipul de relație SOLD și atributele *Data* și *Pret*.

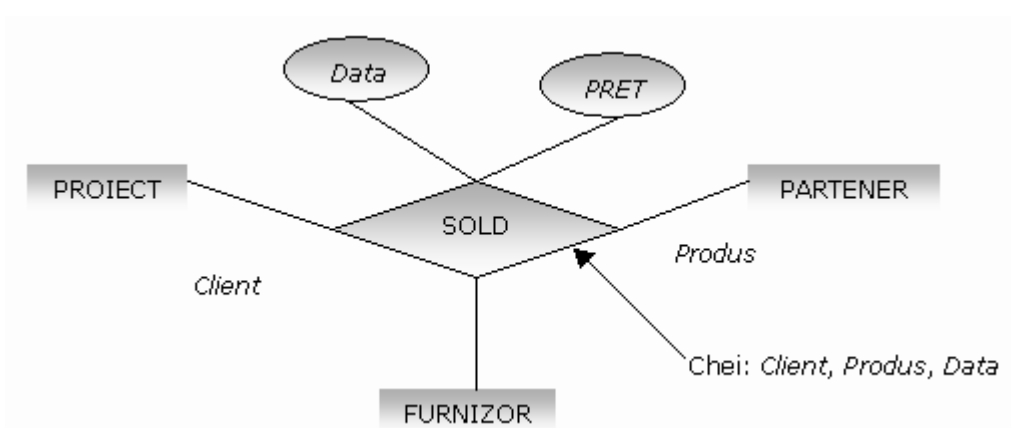


Fig. 4

3.1.6. Restricția de cardinalitate

Fie C un tip de entitate și A tipul de relație conectată la C prin intermediul rolului R . Restricția de cardinalitate a rolului R este o declarație de forma $min..max$ atașată la R , care restricționează numărul de instanțe a relațiilor de tip A în care o singură entitate de tip C poate participa în rolul R și este un număr în intervalul $min..max$, Figura 5.

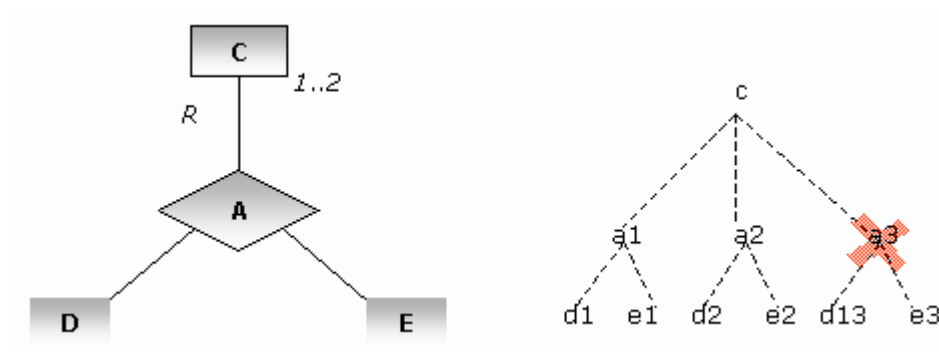


Fig. 5

Figura 5 prezintă și o diagramă cu restricția de cardinalitate pentru rolul R . În partea dreaptă se prezintă o instanță validă a diagramei, unde entitatea C participă în două relații de tip A . Relația $a3$ este tăiată deoarece violează limitele de cardinalitate $1..2$ ale rolului R .

În general, modelul $E-T$ suportă restricția de cardinalitate de forma $min..max$, unde min este un număr mai mare sau egal cu zero, max este un număr mai mare decât zero cu condiția $min \leq max$. În plus, max poate fi simbolizat și prin caracterul $*$, care reprezintă infinitul. Astfel, o restricție de forma $3..*$ asupra rolului R care conectează un tip de entitate C , cu un tip de relație A , înseamnă că fiecare entitate de tip C trebuie să participe în rolul R , cu cel puțin trei relații de tip A (fără limită superioară). Pentru restricțiile de cardinalitate de forma $N..N$ (unde $min=max$) este adesea abreviată cu N , iar caracterul $*$ se folosește pentru sintagma mai mulți.

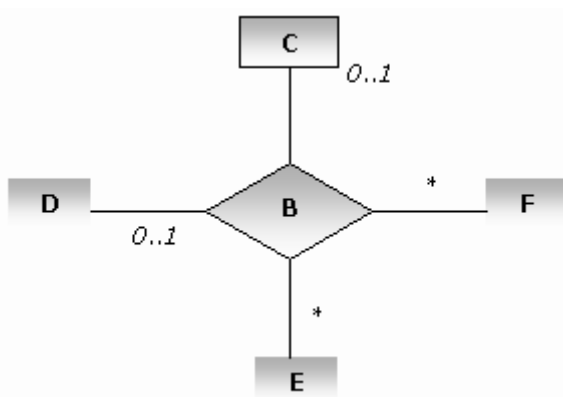


Fig. 6

În Figura 6 sunt prezentate tipuri de relații de grad superior (4). Tipul relației este determinat de restricția de cardinalitate a rolurilor unei relații. De exemplu, relația de tip B implică o

corespondență unu-la-unu între entitatea C și D . Aceasta înseamnă că o entitate de tip C poate fi asociată cu cel puțin o entitate de tip D și vice versa. În același timp, relația B implică o corespondență unu-la-mai mulți pentru entitățile E la C și D (ca și pentru F la C și D). Aceasta înseamnă că o entitate de tip E poate fi asociată cu orice număr (inclusiv zero) de entități de tip C și D , dar o entitate de tip C (sau D) poate fi asociată cu cel mult o entitate de tip E . Relația de corespondență unu-la-mai mulți nu este simetrică; inversa acestei corespondențe (C la E) este numită mai *mulți-la-unul*. Corespondența între E și F este numită mai *mulți-la-mai mulți*, adică o entitate de tipul E poate fi asociată cu orice număr de entități F și vice versa.

3.2. Regulile lui Codd

Detalierea caracteristicilor, pe care trebuie să le prezinte un SGBD pentru a fi considerat relațional, s-a făcut de E. F. Codd în 1985, sub forma a 13 reguli.

R0 – Gestionarea datelor la nivel de relație.

Toate informațiile din baza de date sunt gestionate numai prin mecanisme relaționale. Rezultă că SGBD-ul trebuie să-și îndeplinească toate funcțiile utilizând ca unitate de informație mulțimea, adică să utilizeze limbaje (SQL) care să opereze la un moment dat pe o întreagă relație.

R1 – Reprezentarea logică a datelor

Informațiile din baza de date relațională trebuie să fie reprezentate explicit la nivel logic într-un singur mod și anume ca valori în tabelele de date. Rezultă că toate datele trebuie să fie memorate și prelucrate în același mod. Informațiile privind numele de tabele, coloane, domenii, definițiile tabelelor virtuale, restricțiile de integritate trebuie să fie memorate tot în tabele de date (catalog).

R2 – Garantarea accesului la date

Accesarea informațiilor din baza de date relațională se va face prin specificarea numelui tabelului, a valorii cheii primare și numelui de coloană.

R3 - Valorile NULL

SGBD trebuie să permită declararea și manipularea valorilor null, cu semnificația unor date lipsă sau inaplicabile. Valorile **NULL**, care diferă de șirurile de caractere spațiu sau de șirurile vide de caractere sunt importante în implementarea restricțiilor de integritate (integritatea entității și integritatea referențială) din modelul relațional.

R4 - Metadatele

Informațiile despre descrierea bazei de date, metadatele, trebuie să se prezinte la nivel logic în același mod cu descrierea datelor propriu-zise, utilizatorul aplicând asupra descrierii bazei de date aceleași operații ca și la datele obișnuite. Sistemul nu trebuie să facă diferențe între descrierea datelor și a metadatelor utilizând o singură structură, cea relațională.

R5 – Facilitățile limbajelor utilizate

Un sistem relațional trebuie să facă posibilă utilizarea mai multor limbaje, în mai multe moduri. Trebuie să existe cel puțin un limbaj de nivel înalt ale cărui instrucțiuni să poată exprima oricare din următoarele operații: definirea tabelor de bază, definirea tabelor virtuale, manipularea datelor, definirea restricțiilor de integritate, autorizarea accesului, precizarea limitelor tranzacțiilor.

R6 – Actualizarea tabelor virtuale

Nu toate atributele din cadrul unei tabele virtuale, deci nu toate tabelele virtuale, sunt teoretic actualizabile.

R7 – Actualizările în baza de date

SGBD trebuie să ofere posibilitatea manipulării unei tabele (de bază sau virtuale) nu numai în cadrul operațiilor de regăsire, ci și în cazul celor de adăugare, modificare și ștergere a datelor. În cursul operațiilor prin care se schimbă conținutul bazei de date să se lucreze la un moment dat pe o întreagă relație.

R8 – Independența fizică a datelor

Programele de aplicație nu trebuie să fie afectate de schimbările efectuate în modul de reprezentare a datelor sau în metodele de acces. O schimbare a structurii fizice a datelor nu trebuie să blocheze funcționarea programelor de aplicație.

R9 – Independența logică a datelor

Programele de aplicație nu trebuie să fie afectate de schimbările efectuate asupra relațiilor bazei de date.

R10 – Restricțiile de integritate

Restricțiile de integritate trebuie să poată fi definite în limbajul utilizat de SGBD pentru definirea datelor și să fie memorate

R11 – Distribuirea geografică a datelor

În situația în care datele sunt distribuite, programele de aplicație să fie logic aceleași cu cele utilizate în cazul în care datele sunt fizic centralizate. Utilizatorul trebuie să perceapă datele ca fiind centralizate. Localizarea și recompunerea datelor distribuite cade în seama sistemului.

R12 – Prelucrarea datelor la nivel de bază

Dacă SGBD posedă un limbaj de bază de nivel scăzut orientat pe prelucrarea de înregistrări (tupluri) și nu pe prelucrarea mulțimilor (relațiilor) acest limbaj nu trebuie folosit, pentru a se evita restricțiile de integritate sau restricțiile introduse prin utilizarea limbajelor relaționale de nivel înalt

3.3. Normalizarea datelor

Este procesul prin care se elimină redundanța datelor în proiectul bazei de date și se construiește un model de bază de date care susține diverse cerințe funcționale și structuri alternative ale bazei de date. Sunt aplicate în normalizarea unei baze de date următoarele trei reguli.

N₁. Toate atributele trebuie specificate o singură dată. Aceasta este *forma întâi normală*.

N₂. Un atribut trebuie să depindă în întregime de identificatorul unic al entității pe care o descrie. Aceasta este *forma a doua normală*. Se pun atributele într-o tabelă în care depind exclusiv de o cheie principală. Nu se utilizează tabele în care atributele să nu depindă exclusiv de o singură cheie!

N₃. Pentru a fi în *forma normală a treia*, fiecare tabelă trebuie să posede o singură cheie primară, iar datele din tabelă trebuie să depindă exclusiv de cheia primară a tablei.

3.4. Cerințele minimale de definire a SGBDR

Dintre primele versiuni apărute, practic nici un sistem de gestiune a bazelor de date nu reușea să acopere în întregime regulile lui Codd. De aceea s-au formulat o serie de criterii minimale pe care trebuie să le satisfacă un sistem de gestiune a bazelor de date pentru a putea fi considerat relațional.

Pentru ca un SGBD să fie *minimal relațional*:

1. toate datele din cadrul bazei de date sunt reprezentate prin valori în tabele;

2. între tabele nu există pointeri observabili de către utilizatori – operațiile cu relații nu fac apel la pointeri, indecși, fișiere inverse etc.;
3. sistemul suportă operatori relaționali de proiecție, selecție și joncțiune natural, fără limitări impuse. Unitatea de informație în cadrul acestor operații trebuie să fie relația.

SGBD este *complet relațional* dacă este minimal relațional și satisface în plus următoarele condiții:

- sistemul suportă toate operațiile de bază ale algebrei relaționale;
- sistemul suportă două dintre restricțiile de integritate de bază ale modelului relațional și anume unicitatea cheii unei relații și restricția referențială;

SGBD este *pseudorelațional* dacă satisface condițiile 1 și 3 de mai sus;

SGBD este cu *interfață relațională* dacă satisface condițiile 1 și 3 de mai sus și condiția 3 este îndeplinită numai în raport cu operația de interogare.

4. Componentele funcționale ale sistemului Visual FoxPro

Sistemul de gestiune a bazelor de date este componenta unui sistem de baze de date, care are rolul de a permite descrierea și manipularea datelor conform unui model de date. La acest moment, în lume, cea mai mare parte a SGBD-urilor utilizate sunt bazate pe modelul relațional.

Visual FoxPro este un SGBD complet relațional, cu limbaj propriu, care suportă un nucleu extins din limbajul relațional SQL. Varianta “Visual” de FoxPro (VFP) a ajuns la versiunea 6.0 lansată în 1998 în cadrul pachetului de programe Visual Studio 6.0, care mai conține următoarele programe: Visual C++, Visual Basic 6.0 și Visual Java 6.0. VFP este un sistem rapid, modularizat, flexibil, care nu necesită resurse de calcul foarte mari, îmbină programarea procedurală (prin limbaj propriu) cu cea descriptivă, pe obiecte (programare vizuală).

Utilizatorii neinformaticieni au la dispoziție o gamă largă de generatoare pentru ecrane, meniuri, rapoarte etc.

Utilizatorii de specialitate (informaticieni) pot dezvolta programe în limbaj propriu și în SQL, aplicații (prin componentele **Designer** și **Application**).

Administratorul bazei de date are la dispoziție instrumente oferite de VFP pentru asigurarea securității și integrității datelor, pentru refacerea bazei de date etc.

4.1. Programarea orientată pe obiecte

În programarea orientată pe obiecte, implementată în Visual FoxPro, se utilizează conceptele descrise în continuare.

– *Clasa de obiecte* – reprezintă un tip abstract de date care definește structura obiectelor din acea clasă (proprietățile) și mulțimea de metode (operații) pentru obiectele respective.

O clasă este un model sau o schiță care definește caracteristicile unui obiect și în care este descris modul în care trebuie să arate și să se comporte obiectul. Se poate crea o clasă nouă utilizând comanda **CREATE CLASS** sau modulul de proiectare a clasei (**Class Designer**). Rezultă un fișier cu extensia **.vcx (Visual Class Library)** care va conține clasele definite de utilizator.

Subclass este folosit pentru definirea unei noi clase, pentru un obiect care are ca punct de plecare definirea din altă clasă (clasă părinte). Noua definire va moșteni orice modificare din clasa părinte. O subclassă poate avea toate funcționalitățile unei clase existente și în plus orice control sau funcționalitate care se dorește a fi inclusă. Dacă, de exemplu, clasa de bază este telefonul, se poate

construi o subclasă care are toate funcționalitățile telefonului original și în plus se pot adăuga caracteristici speciale. Astfel, subclasele permit reutilizarea secvenței de instrucțiuni sursă (reutilizarea codului). Utilizarea subclaselor este una din căile de reducere a mărimii codului care trebuie scris. Se începe cu definirea unui obiect care este apropiat de cel dorit și se gestionează.

– *Obiectele* – reprezintă o colecție de proprietăți care se referă la aceeași entitate. Aceste proprietăți descriu structura de date a obiectului. Un obiect are un nume, prin care este referit, un identificator unic, metode, o implementare (privată – utilizatorul nu are acces) și o interfață (care este publică). Cererile adresate unui obiect pentru a returna o valoare sau pentru a schimba o stare se numesc *mesaje*.

Obiectul este o instanță a unei clase, care combină atât date, cât și proceduri. De exemplu, un control într-un videoformat (**Form**) este un obiect.

Clasele și obiectele sunt entități apropiate și înrudite, dar nu se identifică în totalitate. O clasă conține informații despre modul în care un obiect trebuie să arate și să se comporte, dar nu este obiectul ca atare. O clasă este o schiță, o schemă a unui obiect.

Operatorii modelului orientat pe obiecte se referă la actualizarea metodelor, a proprietăților, a claselor, a instanțelor. La baza operațiilor dintr-un astfel de model stau mesajele, care ajută obiectele să comunice între ele.

Restricțiile de integritate a datelor:

- orice obiect respectă restricțiile impuse clasei din care face parte;
- identificatorul obiectului asigură integritatea referirii la acesta;
- accesul la obiect este limitat la folosirea protocolului de mesaje definit pentru clasa din care face parte.

– *Metoda* – definește operațiile permise (operatorii) asupra obiectului, adică definește comportamentul acestuia. Metoda este o acțiune pe care un obiect este capabil să o realizeze. De exemplu, obiectele de tip **List Boxes** au atașate ca metode: **AddItem**, **RemoveItem**, **Clear** pentru a gestiona conținutul listelor.

– *Proprietatea* – este un atribut al unui control, câmp sau obiect dintr-o bază de date, pe care programatorul îl stabilește pentru a defini una din caracteristicile obiectului sau un aspect al comportării acestuia. De exemplu, proprietatea **Visible** dacă este fixată pe **True**, face ca obiectul respectiv să fie vizibil la momentul rulării videoformatului. Proprietățile obiectului pot fi schimbate

în fereastra de *proprietăți*, în general, în timpul proiectării videoformatului; există situații când anumite proprietăți pot fi schimbate și în momentul execuției videoformatului, ca urmare a apariției unor evenimente.

– *Evenimentul* – reprezintă o acțiune, recunoscută de un obiect pentru care se poate scrie o secvență de cod pentru răspuns. Evenimentele pot fi generate de o acțiune a utilizatorului, cum este apăsarea butonului stâng al mouse-ului (**Click** event) sau apăsarea unei taste (**Keypress** event), prin secvențele de program sau de către sistem (*timers*).

Un obiect are anumite proprietăți, de exemplu, un telefon are o anumită culoare și mărime. Când instalăm un telefon pe birou, el are o anumită poziție. Receptorul poate fi pus sau nu în furcă. Obiectele create au de asemenea proprietăți care sunt determinate de clasa din care face parte obiectul. Aceste proprietăți pot fi stabilite în momentul proiectării sau la momentul execuției aplicației. De exemplu, anumite proprietăți pe care le poate avea un obiect de tip **Check Box** sunt enumerate mai jos:

Proprietate	Descriere
Caption	Textul descriptiv de pe lângă Check Box
Enabled	Specifică dacă un obiect Check Box poate fi adresat de utilizator.
ForeColor	Culoarea textului stabilit prin Caption.
Left	Poziția față de marginea din stânga a obiectului Check Box.
MousePointer	Forma pointer-ului când trecem pe deasupra obiectului cu mouse-ul.
Top	Poziția față de marginea de sus a obiectului Check Box.
Visible	Specifică când un obiect Check Box este vizibil.

Fiecare obiect recunoaște și răspunde la anumite acțiuni numite *evenimente*. Un eveniment este o activitate specifică și predeterminată inițiată de utilizator sau de sistem. Evenimentele, în majoritatea cazurilor, sunt generate de interacțiunea cu utilizatorul. De exemplu, la telefon, un eveniment este declanșat în momentul în care un utilizator ridică receptorul din furcă. Evenimentele sunt declanșate și atunci când utilizatorul apasă pe butoane pentru a forma un număr de apel.

În Visula FoxPro acțiunile utilizatorului care declanșează evenimente includ: apăsarea butonului stâng de la mouse (**Click**), mișcarea mouse-ului (**MouseMove**) și apăsarea unei taste (**Keypress**). Sistemul poate iniția evenimente în cazul în care întâlnește o linie de cod care cauzează eroare, cum ar fi mesaje specifice și chiar întreruperea execuției aplicației.

Metodele sunt proceduri care sunt asociate cu un obiect. Metodele sunt diferite de procedurile normale din Visual FoxPro: metodele sunt legate intrinsec de evenimente.

Evenimentele pot avea asociate metode. De exemplu, dacă se scrie un cod pentru o metodă pentru evenimentul **Click**, acel cod este executat atunci când are loc evenimentul **Click**. De asemenea, metodele pot exista independent de orice eveniment. Metodele trebuie să fie apelate explicit în cod. *Setul de evenimente este fix, nu se pot crea evenimente noi!*

Să considerăm câteva evenimente asociate cu un obiect de tip *check box*:

Eveniment	Descriere
Click	Utilizatorul face click pe check box.
GotFocus	Utilizatorul selectează check box cu mouse-ul sau apăsând tasta [TAB].
LostFocus	Utilizatorul selectează un alt obiect.

În tabelul de mai jos sunt date câteva metode asociate cu obiectul de tip *check box*:

Metodă	Descriere
Refresh	Valoarea din check box este actualizată pentru a reflecta orice modificare apărută în datele sursă.
SetFocus	Controlul este dat către check box ca și cum utilizatorul ar apăsa tasta TAB pentru a selecta check box.

Toate proprietățile, evenimentele și metode asociate unui obiect sunt specificate în definiția clasei.

Programarea orientată spre obiecte este un sistem de programare care permite abstractizarea și organizarea modulară de tip ierarhie și are caracteristici polimorfice, de moștenire și încapsulare.

Polimorfismul – este abilitatea de a avea metode cu același nume, dar cu un conținut diferit pentru clasele înrudite. Procedura de utilizare este determinată în momentul execuției de către clasa unui obiect. De exemplu, obiectele înrudite pot avea amândouă metode *Draw*. O procedură transmite un astfel de obiect ca parametru poate apela metoda *Draw* fără a mai fi nevoie să știe ce tip de obiect este parametrul transmis. Polimorfismul reprezintă posibilitatea ca diferitele obiecte să poată răspunde diferit la aceleași mesaje.

Moștenirea – este un termen din programarea orientată spre obiecte și reprezintă abilitatea unei subclase de a prelua caracteristicile clasei părinte (clasa de bază). Când caracteristicile clasei părinte se modifică, subclasa derivată moștenește noile modificări. De exemplu, dacă se adaugă o nouă proprietate *IsBold* la un control de editare, orice subclasă care are la bază clasa acestui control va avea de asemenea proprietatea *IsBold*. Asadar, moștenirea reprezintă capacitatea unui obiect de a-și deriva datele și funcționalitatea din alte obiecte.

Prin intermediul moștenirii dacă se face o modificare într-o clasă, modificarea se reflectă în toate subclasele care aparțin clasei respective. Această actualizare automată făcută prin moștenire aduce beneficii în privința timpului și a efortului de proiectare. De exemplu, dacă un producător de telefoane dorește să schimbe modul de formare a numărului prin trecerea de la sistemul bazat pe disc la sistemul de telefon cu butoane, va economisi multă muncă dacă va fi capabil să facă modificările în schema principală și dacă toate telefoanele realizate anterior pe baza acestei scheme principale vor moșteni automat modificările aduse comparativ cu situația în care ar trebui să facă modificările individual la toate telefoanele existente.

Dacă se descoperă o greșeală în clasa de bază, în loc să fie modificat codul pentru fiecare subclasă se remediază eroarea în clasa de bază și noua modificare se propagă în toate subclasele aferente.

Încapsularea – în programarea orientată spre obiecte este legată de abilitatea de a conține și ascunde informațiile despre un obiect, cum sunt cele referitoare la structura internă a datelor și codul sursă. Încapsularea izolează complexitatea internă a modului de operare a obiectului de restul aplicației. De exemplu, când se fixează proprietatea *Caption* la un buton de comandă nu este necesară cunoașterea modului de stocare a șirului de caractere. Se spune că obiectul din acest punct de vedere este transparent. Încapsularea permite ca descrierea obiectelor să se facă astfel încât să nu existe acces din afara obiectului la datele sale (“black box”).

Abstractizare – este procesul de identificare a caracteristicilor distinctive a unei clase sau obiect fără a fi nevoie de a procesa toate informațiile referitoare la clasă sau obiect. Când se crează o clasă, de exemplu, un set de butoane pentru navigare într-o tabelă, setul se poate utiliza ca o singură entitate în loc de a ține evidența componentelor individuale (butoane) și modul în care acestea interacționează.

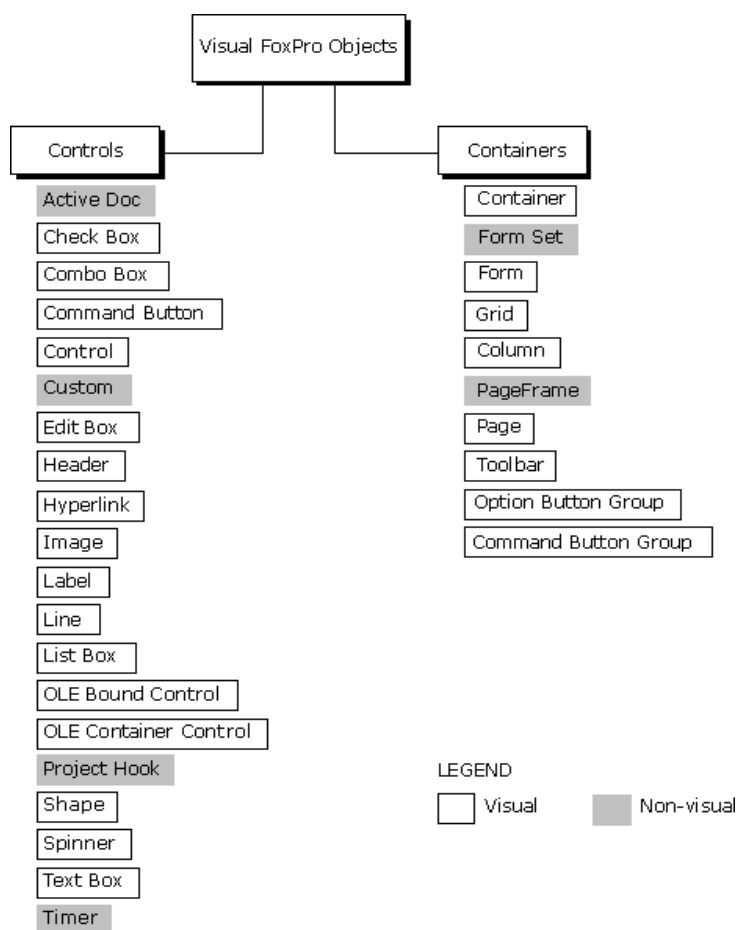
Proiectarea și programarea orientată spre obiecte reprezintă o schimbare a focalizării activității față de programarea procedurală standard. În loc de a gândi schema logică și algoritmul de la prima

linie la ultima linie de cod sursă, trebuie să ne concentrăm asupra creării de obiecte autoconținute într-o aplicație care să posede atât o funcționalitate privată cât și o funcționalitate expusă utilizatorului. Între cele două tipuri de funcționalitate există o interfață prin care utilizatorul poate schimba anumite proprietăți, dar nu are acces la codul sursă intrinsec al obiectului.

În Visula FoxPro videoformatele (**Form**) și controalele (**Control**) sunt obiecte care sunt incluse în aplicații. Aceste obiecte pot fi manipulate prin intermediul *proprietăților*, *evenimentelor* și *metodelor*.

4.2. Ierarhia claselor în Visual FoxPro

La crearea unei noi clase este de mare ajutor cunoașterea ierarhiei obiectelor din Visual FoxPro, preluată din [8]:



4.2. Arhitectura VFP 6.0

Arhitectura SGBD VFP 6.0 (Figura 1) corespunde unui model complet relațional, componentele sale fiind structurate pe trei niveluri: nucleul (*kernel*), interfețele (*interfaces*) și instrumentele (*toolkit*).

1. *Nucleul* – este componenta centrală a sistemului. Din nucleu fac parte:

- *Limbajul FoxPro* care este propriu sistemului, este de tip procedural. El conține comenzi atât pentru descrierea datelor (LDD) cât și pentru manipularea datelor (LMD). Tot aici sunt incluse comenzile pentru programarea vizuală din tehnologia orientată spre obiecte.

- *Nucleul extins SQL* – este un subset din standardul SQL. Acesta este un limbaj relațional descriptiv, care conține atât comenzi pentru descrierea datelor cât și pentru manipularea datelor.

2. *Interfețele* – sunt produse VFP pentru dezvoltarea aplicațiilor cu baze de date relaționale. Ele au următoarele componente:

- **Designer** – permite crearea de diferite obiecte VFP: tabele (**Table**), cereri de regăsire (**Query**), videoformate (**Form**), rapoarte (**Report**), etichete (**Label**), meniuri (**Menu**).

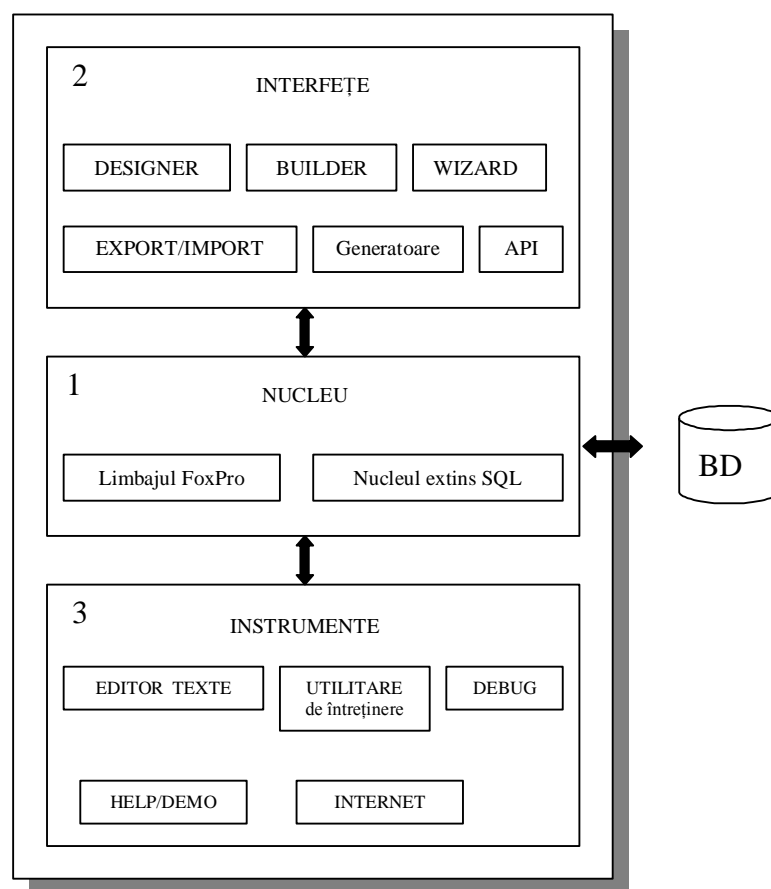


Fig. 1

- Un produs **Designer** se apelează printr-o comandă CRETE/MODIFY aferentă obiectului respectiv. Comanda poate fi generată de sistem sau scrisă de utilizator.
- **Builder** – permite adăgarea rapidă a unor noi elemente la obiectele tip VFP create deja cu produsul **Designer**.
- **Wizard** – permite realizarea completă și rapidă a obiectelor de tip VFP. Utilizarea modulului **Wizard** presupune asistarea utilizatorului de către sistem în proiectarea obiectelor. Generarea lor se face automat, pe baza opțiunilor exprimate de utilizator, sub formă de răspuns la diferite întrebări. Acest modul face parte din categoria instrumentelor de proiectare asistată de calculator.
- **Export/Import** – permite schimbul de date între VFP și alte sisteme.
- Generatoare specializate pentru realizarea proiectelor (PROJECT), a aplicațiilor (APP), a documentației (DOC).
- API (Application Programmer's Interface) – permite apelarea din aplicația VFP a unor rutine scrise în limbajul C sau limbaj de asamblare.

3. *Instrumentele* – sunt produse VFP pentru întreținerea și exploatarea bazei de date.

Instrumentele includ:

Editor de texte – permite încărcarea și editarea programelor sursă (fișiere cu extensia .PRG), precum și a fișierelor ASCII. Apelarea editorului se poate face prin comanda dată în fereastra *Command*:

MODIFY COMMAND <nume_fișier>

- UTILITARE de întreținere – permit gestiunea fișierelor, setarea unor parametri de lucru, activități desfășurate de administratorul bazei de date.
- **debug** – permite depanarea interactivă a programelor scrise în FoxPro.
- **Help/Demo** – permite instruirea interactivă a utilizatorilor.
- **Internet** – permite utilizarea unor servicii de Internet (mail, transfer de fișiere etc.).

4.3. Moduri de lucru în VFP

După intrarea în VFP utilizatorul poate lucra în două moduri: cu meniul sistem și prin comenzi. Ecranul principal VFP conține o fereastră de tip Microsoft cu următoarele elemente (Figura 2):

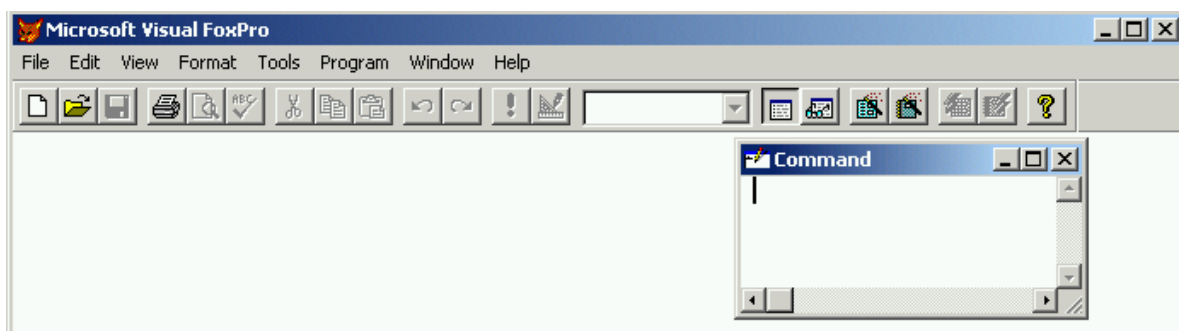


Fig. 2

- *bara de titlu* pe care este scris **Microsoft Visual FoxPro** (în stânga) și butoanele pentru minimizare/maximizare și închidere fereastră (dreapta).
- *bara meniului sistem* care conține un meniu orizontal cu opțiunile de lucru (modul de lucru meniu sistem): **File**, **Edit**, **View**, **Format**, **Tools**, **Program**, **Window**, **Help**. La selectarea unei opțiuni apare un submeniu vertical, de unde se alege mai departe subopțiunea dorită.
- *bara cu instrumente* care conține butoane (icon-uri) dispuse orizontal. Acestea pot fi active sau nu, în funcție de starea curentă de lucru. Prin aceste butoane se poate apela, sub o altă formă, o suboperațiune din meniul sistem.
- *fereastra de comandă* care conține un cursor și permite introducerea unei comenzi VFP sau apelul unui program (modul de lucru prin comenzi).
- *aria de ieșire* este formată din restul spațiului neocupat din fereastra principală unde vor fi afișate rezultatele execuției unei comenzi sau a unui program VFP.

4.3.1. Modul de lucru meniu sistem

Este modul de lucru care permite apelarea tuturor instrumentelor și interfețelor sistemului VFP. Soluția este adoptată de utilizatorii care preferă dezvoltarea aplicațiilor cu ajutorul generatoarelor. Efortul depus este redus și nu se programează în cod sursă. Opțiunile din meniul principal ca și cele din submeniuri pot fi apelate prin mouse, sau cu o combinație de taste. De exemplu, apelarea meniului **File** (Figura 3) se face tastând secvența [ALT] + [F] (litera subliniată din componența numelui meniului).

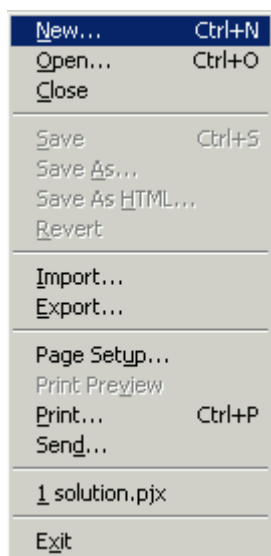


Fig. 3

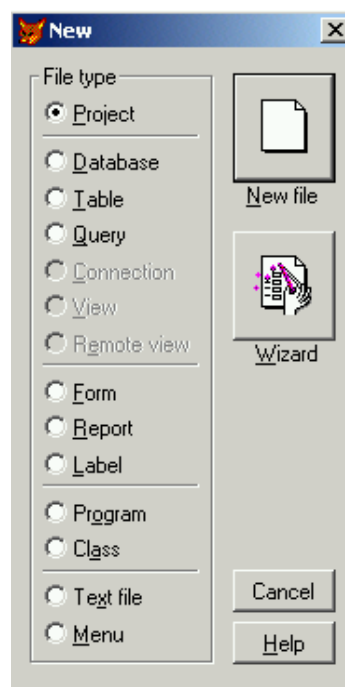


Fig. 4

Funcțiile opțiunilor din meniul principal al sistemului sunt:

– **File** – apelează instrumente pentru întreținerea fișierelor aferente unei baze de date VFP (Figura 3):

New – crează un obiect nou. Apare o listă din care putem selecta: **Project** (crearea unui proiect), **Database** (crearea de bază de date), **Table** (crearea unei tabele), **Query** (crearea unei cereri), **Connection** (realizează oconexiune), **View** (crearea unei viziune locală), **Remote View** (crearea unei viziune la distanță), **Form** (crearea unui videoformat), **Report** (crearea unui raport), **Label** (crearea etichetelor), **Program** (pentru crearea unui program sursă), **Class** (crearea unei clase de obiecte), **Text file** (crearea de fișier de tip text - ASCII), **Menu** (crearea unui meniu utilizator). În partea dreaptă există două butoane, pentru crearea unui fișier nou (**New**) și pentru utilizarea asistentului (**Wizard**) (Figura 4).

Open (deschide) – apare o fereastră din care se alege tipul fișierului, directorul și numele, cu opțiunile **New**, **Open**, **Cancel**.

Close – închide fișierul deschis.

Save/Save As – salvare, respectiv salvare cu redenumire.

Import/Export – permite importul respectiv exportul de date cu alte sisteme de gestiune a bazelor de date.

Print Preview – vizualizare înainte de ieșire la imprimantă.

Print – ieșire la imprimantă.

Send – trimite prin e-mail.

Exit – ieșire din program.

Edit – oferă facilități de lucru obișnuite într-o fereastră de editare de texte (Figura 5):

Undo/Redo – renuță la ultima modificare în text / repetă ultima acțiune în text.

Cut/Copy/Paste – tăiere, memorare în memoria tampon, copiere la o nouă locație în fișier a unui text.

Clear – ștergere text.

Select all – selectează întreg textul.

Find/Find again/Replace – caută/caută în continuare un șir de caractere sau înlocuiește șirul găsit cu un altul. Căutarea se poate face cu activarea/dezactivarea opțiunii de diferențiere între litere mari/litere mici.

Undo	Ctrl+Z
Redo	Ctrl+R
Cut	Ctrl+X
Copy	Ctrl+C
Paste	Ctrl+V
Paste Special...	
Clear	
Select All	Ctrl+A
Find...	Ctrl+F
Find Again	Ctrl+G
Replace...	Ctrl+L
Go to Line...	
Insert Object...	
Object...	
Links...	
Properties...	

Fig. 5

– **View** – permite setarea unor parametri privind bara curentă de instrumente de lucru.

– **Format** – permite setarea unor parametri privind literele (Font), spațierea etc.

– **Tools** – permite apelul interfețelor și instrumentelor VFP.

– **Program** – permite lansarea sau oprirea unuia din pașii ce se parcurg la execuția unui program VFP (Figura 6).

Do...	Ctrl+D
Cancel	
Resume	Ctrl+M
Suspend	
Compile...	

Fig. 6

Do – lansează în execuție un program.

Cancel – anulează execuția programului.

Resume – reia execuția programului.

Suspend – suspendă execuția programului.


Compile – compilează programul.

Window – permite setarea parametrilor ferestrei curente de lucru (inclusiv fereastra de comandă).

Help – apelează instrumentele pentru autodocumentare.

Datele pot fi introduse în două moduri, utilizând fie tabele independente (**File/New/Table**) care nu aparțin unei baze de date, fie creând o bază de date (**File/New/Database**), în care se pot introduce tabele existente sau se pot crea noi tabele.

Relațiile între tabele se pot stabili în ambele cazuri. Există trei tipuri de relații între datele unor tabele: $1 \leftrightarrow 1$, $1 \leftrightarrow n$, $m \leftrightarrow n$.

Deschiderea unei tabele individuale, cu secvența **File/Open/Table/nume_tabelă** nu duce automat la afișarea conținutului acesteia în spațiul de lucru. Pentru aceasta trebuie activat icon-ul  care corespunde opțiunii **Data Session**, care va afișa într-o fereastră o serie de opțiuni legate de tabelă (Figura 7):

Properties – pentru modificarea/consultarea structurii tablei (nume, tip, poziție câmp).

Browse – pentru afișare date/nume câmpuri.

Open – pentru deschidere altor tabele.

Close – închidere fișier selectat.

Relation – stabilirea relațiilor între tabele independente, deschise în sesiunea curentă.

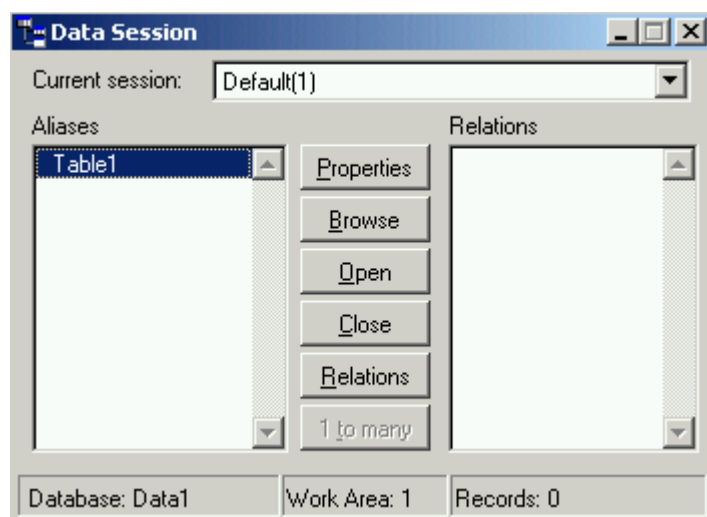


Fig. 7



Fig. 8

La alegerea opțiunii **Browse** (răsfoire), fie din meniul **View** fie din fereastra **Data Session**, va apare meniul **Table** în bara de meniuri cu următoarele subopțiuni (Figura 8):

Go to Record – regăsirea unei înregistrări după diferiți parametri: număr înregistrare, deplasare la începutul/sfârșitul tablei, localizare înregistrare după valoarea unui câmp etc.

Append New Record – adăgarea unei noi înregistrări.

Append Records – adăugarea de noi înregistrări dintr-un alt fișier

Delete Records – marcarea pentru ștergere (la nivel logic).

Recall Records – anularea marcării pentru ștergere.

Remove Deleted Records – ștergerea definitivă din tabelă a înregistrărilor marcate (la nivel fizic).

Replace Field – actualizarea câmpului unei înregistrări (schimbarea valorii câmpului).

Size Field – modificarea lăţimii de afişare a câmpului în browser.

Move Field – schimbarea poziţiei de afişare a câmpului în browser.

În cazul în care se construieşte o aplicaţie în care va fi folosită o gamă largă de obiecte din VFP, (baze de date, tabele independente, cereri, videoformate, rapoarte, etichete, programe sursă, clase de obiecte, meniuri utilizator, iconu-uri etc.) se va construi un proiect (**Project**), în secţiunile căruia se pot declara aceste obiecte (Figura 9). Întreg proiectul se va finaliza într-un program executabil care va conţine toate obiectele declarate ca fiind utilizate în aplicaţie. Pentru a realiza proiectul în formă executabilă, în prealabil se foloseşte opţiunea **Build**, pentru a realiza compilarea şi link-editarea (§12.2.2.).

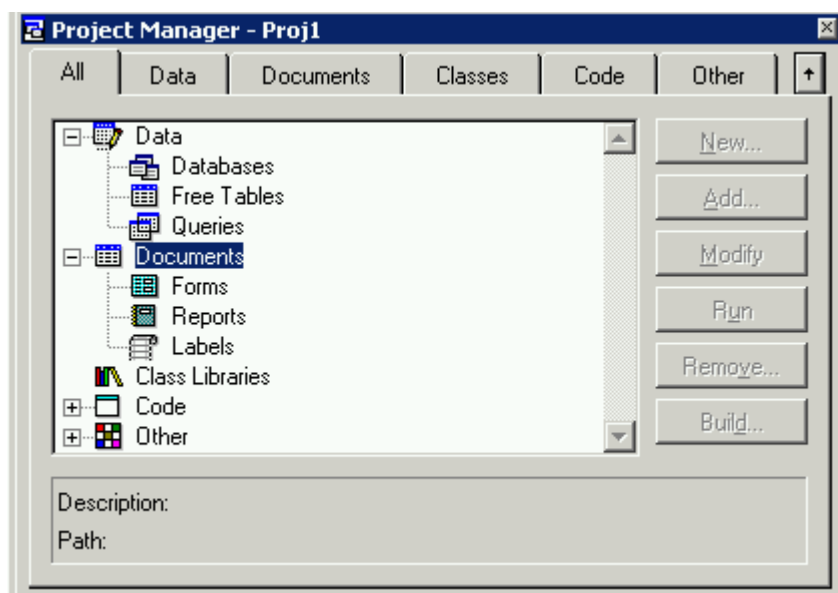


Fig. 9

4.3.2. Modul de lucru prin comenzi

Activitatea în acest mod de lucru se desfăşoară în fereastra de comandă din ecranul principal al VFP (**Command**). Aici se găseşte poziţionat cursorul şi utilizatorul poate lucra în stil interpretor sau compilator.

5. Organizarea datelor în Visual FoxPro

5.1. Manipularea bazei de date și a tabelelor în VFP

Manipularea presupune următoarele operații care pot fi executate fie în modul asistat, fie în modul comandă, fie în modul procedural:

- Crearea bazei de date/tabelei
- Deschiderea bazei de date/tabelei. Deschiderea unei tabele presupune rezervarea în memoria internă a calculatorului a unei zone, numită *zonă de lucru*, în care VFP va memora toate informațiile necesare utilizării acestei tabele (structură, număr de înregistrări etc).

Într-o zonă de lucru, la un moment dat, este deschisă o singură tabelă, iar simultan pot fi deschise 32767 tabele. Identificarea zonelor de lucru de face printr-un număr natural cuprins între 1 și 32767, iar primele 10 zone și prin litere, A-J. La lansarea sistemului VFP, automat este selectată zona de lucru A (sau 1).

Cu comanda

SELECT *litera/cifra*

se poate selecta o altă zonă de lucru.

- Efectuarea operațiilor de consultare, actualizare (adăgare, modificare, ștergere) asupra bazei de date/tabelei
- Închiderea bazei de date/tabelei

5.1.1. Crearea bazei de date

Sistemul VFP permite crearea unei baze de date și introducerea tabelor care vor face parte din baza de date:

- *definirea structurii* acestora;
- *crearea fișierului* (fișierelor) în care se pot stoca datele;
- *stabilirea relațiilor* între tabelele bazei de date,

Crearea se face o singură dată, înainte de utilizare.

Notă Secvențele din linia de comandă care apar între paranteze drepte au semnificația de element opțional.

Crearea unei baze de date se face cu comanda:

CREATE DATABASE [*<nume_bază_de_date>*];

nume_bază_de_date – specifică numele bazei de date care va fi creată.

? – afișează o fereastră de dialog pentru specificarea numelui bazei de date.

Fișierul corespunzător bazei de date are extensia .DBC. La crearea bazei de date se face automat și deschiderea ei. Adăugarea unei tabele în baza de date se face cu comanda ADD TABLE sau folosind bara de instrumente DATABASE DESIGNER.

5.1.2. Crearea tabelor

Crearea tabelor se realizează în funcție de modul de lucru, astfel:

1. *modul asistat* – se utilizează secvența de adresare a meniului sistem: **File/New** se alege opțiunea **Table** din fereastra de dialog și se apasă butonul **New**. Această secvență are ca efect apariția unei ferestre de dialog **Create**, în care se specifică discul logic, directorul și numele tabelii care va fi creată. Deplasarea până la poziția dorită în fereastră, se face utilizând tasta [TAB] sau mouse-ul. După introducerea numelui tabelii se apasă butonul **Save** (fișierul va fi salvat cu extensia .DBF) și se afișează o nouă fereastră **Table Designer**. În această fereastră, se selectează opțiunea **Fields** afișată sus, utilizatorul trebuie să introducă structura tabelii. Fereastra conține următoarele câmpuri de preluare a informațiilor structurale: **Name**, **Type**, **Width**, **Dec**, **Index**, **NULL** și patru butoane pentru opțiunile: **OK**, **Cancel**, **Insert**, **Delete**. Sub **Name** se tastează numele câmpului, format din maxim 10 caractere, începând cu o literă, iar sub **Type** se declară tipul câmpului nou creat sau se alege o opțiune din meniul pop-up afișat.

Câmpul poate fi de tip:

- *character* – admite un șir de maxim 254 caractere (caractere alfanumerice și spații goale);
- *numeric* – admite maxim 20 de caractere, numere, caracterele +/-, punctul zecimal;
- *logic* – admite maxim un caracter (**T** sau **Y**, **F** sau **N**);
- *dată* – admite maxim 8 caractere și are formatul implicit *ll/zz/aa*. Formatul poate fi schimbat cu comanda SET DATE TO (de exemplu **BRITISH** și are ca efect schimbarea în formatul *zz/ll/aa*);
- *memo* – admite implicit 10 caractere, însă sistemul poate stoca blocuri mari de text pentru fiecare înregistrare. Dimensiunea blocurilor de text este limitată de spațiul pe hard disk. Tabelele care au declarate în structură câmpuri de tip memo sunt stocate sub forma a două fișiere, unul cu extensia .DBF și celălalt cu extensia .FTP în care se salvează câmpurile memo ale tabelii.
- *general* – se folosește pentru memorarea elementelor de tip OLE (Object Linking and Embedding), texte, foi de calcul tabelar (Excel), imagini, sunete etc.

Notă: Introducerea datelor în câmpurile memo și general este diferită de introducerea datelor în alte câmpuri. Se poziționează cursorul pe câmpul memo, se apasă simultan tastele **[Ctrl]+[PgDown]** și apare o fereastră cu același nume ca al câmpului memo în care poate fi introdus textul. Textele introduse se salvează apăsând tastele **[Ctrl]+[W]**, fereastra memo se închide și cursorul revine în fereastra de adăugare a datelor. Cu **[Ctrl]+[Esc]** se iese din fereastra memo fără salvarea datelor introduse în fișierul memo.

Width – opțiune care specifică numărul maxim de poziții ale câmpului.

Dec – specifică mărimea părții zecimale, în cazul câmpurilor numerice.

Index – specifică dacă atributul (câmpul) respectiv este index (ascendent/descendent) sau nu (§8.2.).

NULL – specifică dacă atributul respectiv admite sau nu valori de tip **NULL** (cu semnificația unor date lipsă sau inaplicabile).

Dacă se dorește schimbarea poziției unui câmp în structură, se apasă tasta **[Tab]** până când cursorul se poziționează pe câmpul respectiv și se apasă simultan tastele **[Ctrl] + [PgDown]**, **[Ctrl] + [PageUp]**. Aceasta va avea ca efect deplasarea câmpului selectat cu o poziție în jos, sau în sus.

Operația se poate efectua și cu mouse-ul: se selectează rândul care conține câmpul a cărei poziție se modifică, cursorul de poziție se mută pe acest rând; ținând apăsat butonul stâng de la mouse pe cursor și deplasând mouse-ul sus/jos se va muta câmpul selectat, în cadrul structurii generale.

Butonul cu opțiunea **Insert** se folosește pentru adăugarea unui nou câmp.

Butonul cu opțiunea **Delete** se folosește pentru ștergerea unui câmp (cel marcat de cursor).

Butonul **Cancel** anulează modificările făcute, iar butonul **OK** validează modificările (sau folosind combinația de taste **[Ctrl] + [Enter]**).

După ce structura este salvată, fișierul fiind nou creat, apare un mesaj:

Input data records now? (Y/N)_

Dacă se răspunde cu **Y** este afișat un ecran în care se introduc datele în ordinea stabilită în structură.

5.1.3. Validarea câmpurilor unei înregistrări la introducere

În exemplul care urmează se crează o bază nouă de date GESTIUNE.DBC (dbc - data base container), Figura 10, care va conține tabelele: FURNIZOR.DBF și MATERIALE.DBF cu următoarele câmpuri:

FURNIZOR.DBF	
Cod_furn	N(5)
Denumire	C(20)
Adresa	Memo
Cod_mat	N(5)
Cant	N(7)
Data_livr	D

MATERIALE.DBF	
Cod_mat	N(5)
Denumire	C(20)
Data_exp	D
UM	C(5)
PU	N(5)

După definirea structurii înregistrărilor în tabela FURNIZOR.DBF se selectează câmpul *cod_furn* și se completează cele trei câmpuri **Rule**, **Message**, **Default value** (Figura 11), astfel:

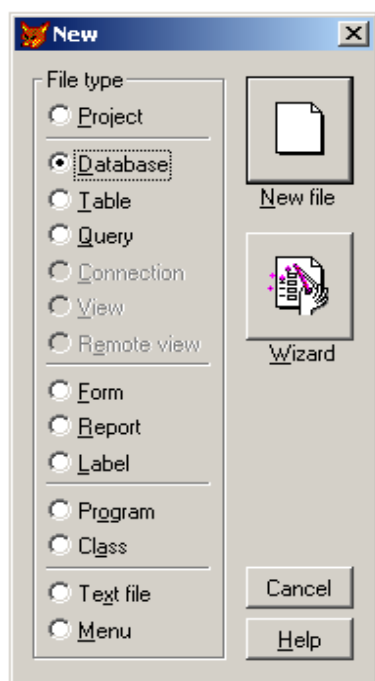


Fig. 10

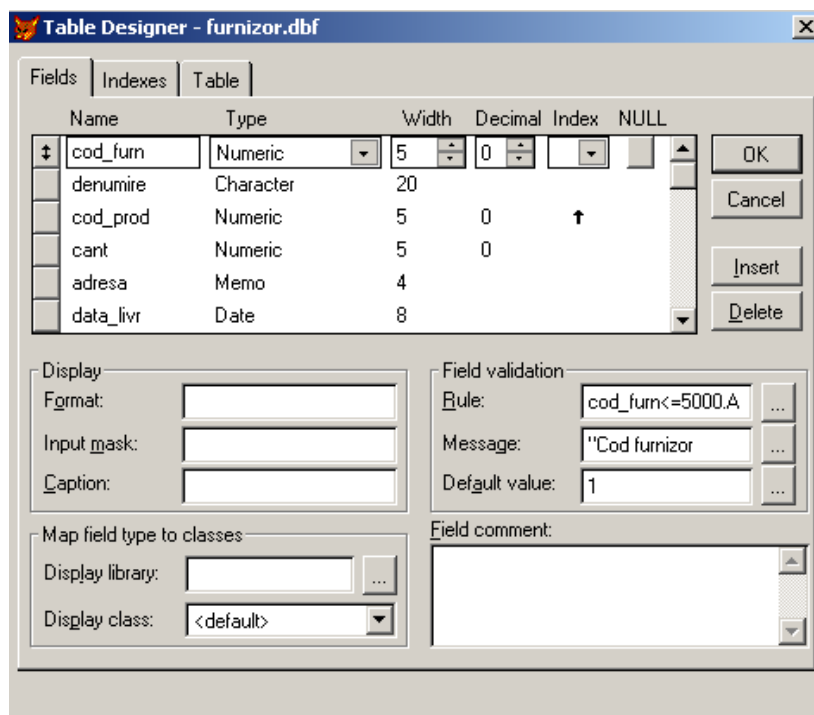


Fig. 11

- la opțiunea **Rule** se introduce condiția logică de filtrare pentru acceptarea valorii înscrise în câmp. În acest caz se consideră că pentru câmpul *cod_furn* valorile acceptate sunt între 1 și 5000. Condiția este:

$$cod_furn \leq 5000 \text{ and } cod_furn \geq 1$$

- la opțiunea **Message** se introduce între apostrof mesajul:

'Cod furnizor eronat !'

- la opțiune **Default Value** se introduce valoarea 1. Dacă se omite declararea unei valori implicite, la introducerea datelor pentru câmpul *cod_furn*, se va afișa mesajul de eroare la orice încercare de introducere a datelor.

După definirea structurii înregistrărilor în tabela MATERIALE.DBF se selectază câmpul *um* și se completează cele trei câmpuri **Rule**, **Message**, **Default Value** (Figura 12), astfel:

- la opțiunea **Rule** se introduce condiția logică de filtrare pentru acceptarea valorii înscrise în câmp. În acest caz se consideră că pentru câmpul *um* valorile acceptate sunt doar trei: *kg*, *ml*, *buc*. Condiția este:

um='kg' or um='ml' or um='buc'

- la opțiunea **Message** se introduce între apostrof mesajul:
'Eroare unitate de masura !!'
- la opțiune **Default Value** se introduce între apostrof valoarea '*buc*'. Dacă nu se declară nimic, practic, nu se pot introduce datele la fel ca mai sus.

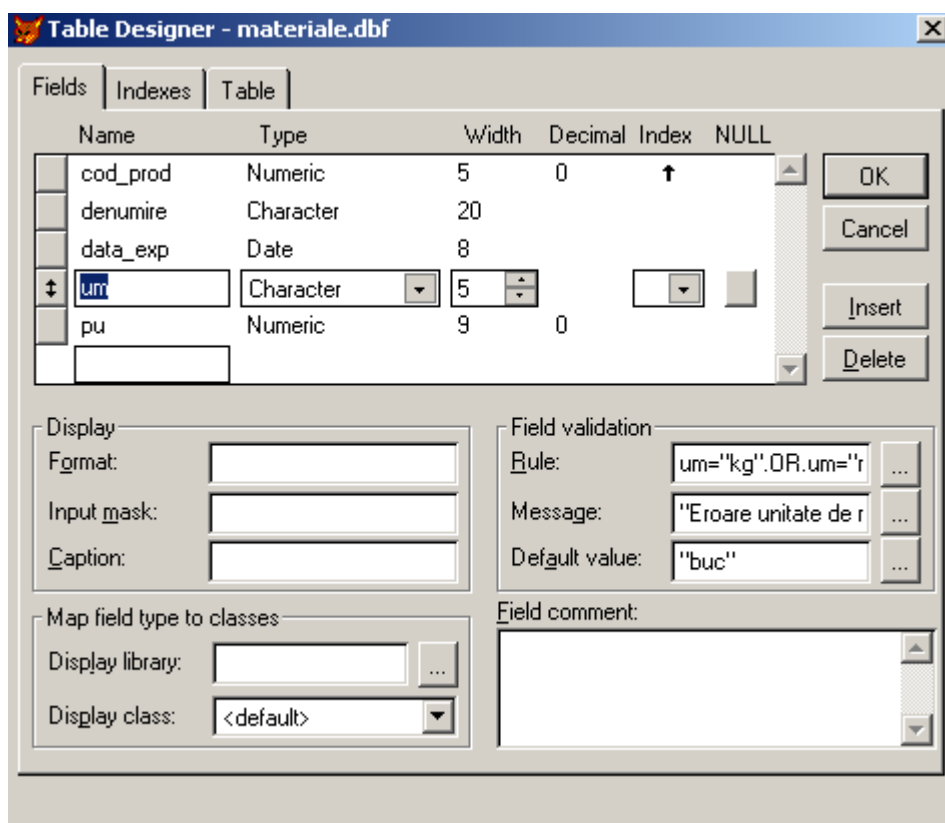


Fig. 12

2. **modul comandă**: implică utilizarea comenzii CREATE TABLE din nucleul SQL:

CREATE TABLE *nume_tabelă* [FREE]

(*nume_câmp1 tip_câmp1* [(*lungime_câmp* [,*precizie*])]

[,nume_câmp2...]

.....

)

unde:

- *nume_tabelă* – numele tabelii create;
- FREE – specifică că tabela nu va fi adăugat la baza de date deschisă;
- *nume_câmp1 tip_câmp1 (lungime_câmp ,precizie)* – specifică numele câmpului 1, tipul câmpului 1, - lungimea maximă și numărul de zecimale pentru câmpurile numerice. O tabelă poate conține până la 255 de câmpuri. Dacă unul sau mai multe câmpuri acceptă și valoarea **NULL** atunci numărul de câmpuri se reduce cu 1 (254).

Tipul câmpului se declară cu una din următoarele litere:

Tip_câmp	Lungime_câmp	Precizie	Descriere
C	<i>n</i>	–	câmp tip caracter de lungime <i>n</i>
D	–	–	câmp de tip dată calendaristică
T	–	–	câmp de tip timp (<i>hh/mm/ss</i>)
N	<i>n</i>	<i>d</i>	câmp numeric de lungime <i>n</i> cu <i>d</i> zecimale
F	<i>n</i>	<i>d</i>	câmp numeric în virgulă mobilă de lungime <i>n</i> cu <i>d</i> zecimale
I	–	–	câmp numeric întreg
B	–	<i>d</i>	câmp numeric dublă precizie
Y	–	–	moneda
L	–	–	câmp de tip logic
M	–	–	câmp de tip memo
G	–	–	câmp de tip general

Exemplu. Folosind modul comandă se crează o tabelă cu evidența studenților, cu numele ESTUD, cu următoarele câmpuri:

- număr matricol: MATR – de tip numeric, lungime maximă 5
- nume și prenume: NUME – de tip caracter, lungime maximă 40

- adresă: ADRESA – de tip caracter, lungime maximă 30
- anul de studii: ANS – de tip numeric, lungime maximă 1
- anul intrării în facultate: ANIF – de tip numeric, lungime maximă 4
- data nașterii: DATAN – de tip dată.

Comanda:

```
CREATE TABLE ESTUD (MATR N(5), NUME C(40), ADRESA C(30), ANS N(1), ANIF N(4), DATAN D)
```

Alte comenzi utilizate în crearea tabelelor

Pentru crearea unei tabele se pot utiliza și comenzile COPY STRUCTURE EXTENDED TO sau CREATE ... FROM ...cu sintaxa

```
COPY STRUCTURE EXTENDED TO <nume_tabelă> [FIELDS<listă_câmpuri>] [DATABASES <nume_bază_de_date> [NAME] <b_nume_tabelă>]
```

unde:

- *nume_tabelă* – numele tablei nou create;
- FIELDS <listă_câmpuri> – specifică câmpurile care urmează să apară în noua tabelă. Dacă nu se specifică nimic, implicit sunt luate toate câmpurile din tabelă sursă;
- DATABASE <nume_bază_de_date> – numele unei bazei de date existente la care va fi adăgată noua tabelă;
- NAME <b_nume_tabelă> – specifică numele tablei așa cum apare în baza de date.

Fișierul de structură creat are următoarele câmpuri:

FIELD_NAME	câmpul nume (10 caractere)
FIELD_TYPE	câmpul tip (maxim 1 caracter): C, N, F, D, L, M, G
FIELD_LEN	lungimea câmpului (maxim 3 caractere)
FIELD_DEC	număr de zecimale admis (maxim 3 caractere)

Comanda CREATE... FROM...crează o tabelă dintr-un fișier de structură (realizat cu comanda anterioară COPY STRUCTURE EXTENDED TO...). Sintaxa comenzii:

```
CREATE [<nume_tabela_1>] FROM <nume_tabela_2>
```

nume_tabela_1 – numele tablei care va fi creată;

nume_tabela_2 – numele fișierului care conține structura care va fi utilizată la creare.

Exemplu. Folosind tabela creată anterior (ESTUD), să se realizeze o nouă tabelă (ESTUD1), care să conțină suplimentar un câmp numeric (lungimea maximă 5 caractere, din care 2 la partea zecimală) cu media de la bacalaureat (MBAC). Secvența de comenzi:

SELECT 2	&& selectează zona de lucru 2
USE ESTUD	&& se deschide tabela ESTUD în zona de lucru 2
COPY STRUCTURE EXTENDED TO TEMP	&& copiază structura tabeli ESTUD în fișierul TEMP
USE TEMP	&& se deschide tabela TEMP
APPEND BLANK	&& se adaugă o înregistrare goală
REPLACE FIELD_NAME WITH "MBAC"	&& se adaugă un nou câmp MBAC
REPLACE FIELD_TYPE WITH "N"	&& tip numeric
REPLACE FIELD_LEN WITH 5	&& de lungime maxim 5 poziții
REPLACE FIELD_DEC WITH 2	&& 2 poziții pentru partea zecimală && la fiecare REPLACE se afișează mesajul "1 replacements"
CREATE ESTUD1 FROM TEMP	&& se crează tabela ESTUD1 cu structura dată de fișierul TEMP
USE ESTUD1	&& se deschide tabela ESTUD1
APPEND FROM ESTUD	&& se adaugă înregistrările din tabela inițială ESTUD
BROWSE	&& afișarea conținutului tabeli ESTUD1 în browser
CLOSE DATABASES	&& se închid toate tabelele
DELETE FILE TEMP.DBF	&& șterge fișierul TEMP.DBF

5.2. Deschiderea bazei de date/tabelei

Deschiderea bazei de date se face cu comanda:

```
OPEN DATABASE [<nume_bd>] [EXCLUSIVE | SHARED] [NOUPDATE]
```

unde:

nume_bd – numele bazei de date care trebuie deschisă;

EXCLUSIVE – alți utilizatori nu pot să acceseze această bază de date;

SHARED – deschide baza de date în modul partajat, ceilalți utilizatori au acces;

NOUPDATE – specifică faptul că nu se pot face schimbări în baza de date (deschis doar pentru citire).

Comanda pentru deschiderea unei tabele într-o zonă de lucru:

```
USE <nume_fișier> IN <nr_zonă_de_lucru> INDEX <listă_fișiere_index> ALIAS <alias_tabelă>  
[EXCLUSIVE | SHARED] [NOUPDATE]
```

unde:

nume_fișier – numele tablei pe care vrem să o deschidem.

nr_zonă_de_lucru – numărul zonei de lucru asociate la deschiderea tablei (numere 0, 1, 2.. sau litere A, B...).

INDEX <listă_fișiere_index> - specifică un set de indecși care se deschid odată cu tabela. Dacă tabela are un fișier de index structural compus, fișierul de index este deschis automat odată cu fișierul. Lista de fișiere index conține orice fel de nume de fișier de index, simplu (.IDX) sau nestructural compus (.CDX), separate prin virgulă, fără specificarea extensiei. Primul fișier de index din listă este cel care controlează accesarea și afișarea datelor din tabelă (§8.2.).

ALIAS <alias_tabelă> - crează un nou nume pentru tabelă, diferit de numele extern. Referirea la fișier se poate face prin intermediul alias-ului, în funcții sau comenzi care cer sau suportă alias-ul. Specificarea zonei de lucru poate fi făcută și separat asociind comanda SELECT la comanda USE.

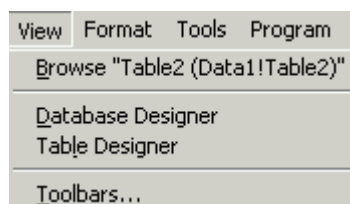
```
SELECT <nr_zonă_de_lucru> | <nume_alias>
```

```
USE <nume_fișier>
```

Notă Semnul | în sintaxa unei comenzi marchează alternanța unor clauze care se exclud.


În modul asistat, deschiderea se face utilizând meniul **File/Open**, se selectează **Database** sau **Table** din meniul pop-up afișat și se specifică discul logic, subdirectorul și numele bazei de date sau tablei, precum și modul de deschidere (se bifează sau nu, opțiunea **Exclusive**). Când toate aceste informații sunt specificate se apasă butonul **OK**.

După operația de deschidere în meniul **View** avem subopțiunile **Browse** (vizualizare conținut), **Database Designer** (proiectantul pentru baza de date), **Table Designer** (proiectantul pentru tabele). Se pot face consultări cu **Browse**, modificări ale structurii cu **Database Designer** (adăugare, eliminare de tabele din bază, stabilirea/anularea relațiilor între tabele) sau **Table Designer** (actualizare câmpuri).



5.3. Consultarea și modificarea bazei de date/tabelei

5.3.1. Modificarea structurii de date a tabelor în modul asistat

Modificarea structurii logice a unei tabele se poate face fie cu ajutorul comenzii **MODIFY STRUCTURE** din fereastra de comandă, fie activând butonul **Data Session** din bara de instrumente .

În primul caz, după lansarea comenzii **MODIFY STRUCTURE**, se va afișa fereastra de dialog **Open**. Cu ajutorul săgeților direcționale se poziționează cursorul pe tabela dorită a fi modificată. Se apasă tasta **[Enter]** și va apărea fereastra de dialog **Structure**, utilizată și la definirea structurii tabelor.

În cel de al doilea caz, din fereastra **Data Session** se alege opțiunea **Open**, care deschide fereastra de selectare a discului logic, a subdirectorului și a numelui de fișier. Se selectează fișierul și se apasă butonul **OK**. În fereastra **Data Session** va fi afișat numele tablei deschise. Pot fi deschise mai multe tabele simultan, numele lor fiind afișat în fereastra **Data Session**. Se alege opțiunea **Properties** care va activa fereastra de dialog **Structure**.

În ambele cazuri operația **Open** se va face cu opțiunea **Exclusive** pentru a putea face modificări.

În fereastra **Structure**, poziționarea cursorului se face apăsând tasta **[Tab]** pentru deplasare înainte sau **[Shift] + [Tab]** pentru deplasare înapoi, în zona ce cuprinde descrierea structurii, unde vrem să inserăm un nou câmp. În partea dreaptă a ferestrei vor fi active opțiunile **Insert**, **Delete** (adăgare/ștergere). Se apasă tasta **[Insert]** sau butonul cu opțiunea **Insert** și va fi afișată următoarea linie:

New field	character	10
-----------	-----------	----

Câmpurile ce urmează vor fi împinse cu un rând în jos. Se introduce numele câmpului în prima căsuță, se alege tipul și lungimea maximă din meniurile pop-up. Pentru ștergere se poziționează cursorul pe câmpul respectiv și se apasă tasta **[Delete]** sau se folosește butonul cu opțiunea **Delete**.

Salvarea modificărilor se face selectând opțiunea **OK** (buton), sistemul afișând următorul mesaj:

Make structure changes permanent ?

YES	NO
-----	----

Opțiunea **YES** salvează noua structură a tabelii, fereastra de dialog **Structure** se închide și controlul (cursorul) trece în fereastra de comandă.

Notă Modificarea structurii tabelilor se poate face și cu comanda ALTER TABLE din nucleul SQL.

La alegerea opțiunii **Browse** (răsfoire), fie din meniul **View** fie din fereastra **Data Session**, va apare opțiunea **Table** în bara de meniuri cu următoarele subopțiuni:

Go to Record – regăsirea unei înregistrări după diferiți parametri: număr articol, deplasare la începutul/sfârșitul tabelii, localizare înregistrare după valoarea unui câmp etc.

Append New Record – adăgarea unei noi înregistrări.

Append Records – adăugarea de noi înregistrări dintr-un alt fișier

Delete Records – marcare pentru ștergere (la nivel logic).

Recall Records – anularea marcării pentru ștergere.

Remove Deleted Records – ștergerea definitivă din tabelă a înregistrărilor marcate (la nivel fizic).

Replace Field – actualizare câmp înregistrare (schimbarea valorii câmpului).

Size Field – modificarea lățimii de afișare a câmpului în browser.

Move Field – schimbarea poziției de afișare a câmpului în browser.

Vizualizarea datelor cu opțiunea **Browse**, se face într-o fereastră de tip **Windows** care cuprinde o tabelă, coloanele reprezentând câmpurile, iar liniile reprezentând înregistrările. Trecerea de la un câmp la altul se face apăsând tasta **[Tab]** (înainte), **[Shift] + [Tab]** (înapoi); pe verticală deplasarea se face cu săgețile **[Sageata sus]**, **[Sageata jos]**. Închiderea ferestrei **Browse** se face apăsând simultan tastele **[Ctrl] + [End]**.

5.3.2. Deplasări în tabelă. Căutări secvențiale

Odată cu deschiderea tabelii, acesteia i se asociază o locație de memorie în care este stocat numărul înregistrării curente (pointer-ul de înregistrare).

Există 2 tipuri de deplasări:

- criteriul utilizat este numărul înregistrării, care se realizează utilizând din submeniul **Go to Record** subopțiunile: **Top**, **Bottom**, **Next**, **Previous**, **Record n** (prima înregistrare, ultima înregistrare, următoarea înregistrare, înregistrarea anterioară, la înregistrarea n).
- criteriul utilizat se obține pe baza specificării unei condiții de tip **For** sau **While** aplicată unui câmp. Se utilizează opțiunea **Locate** din submeniul **Go to Record**.

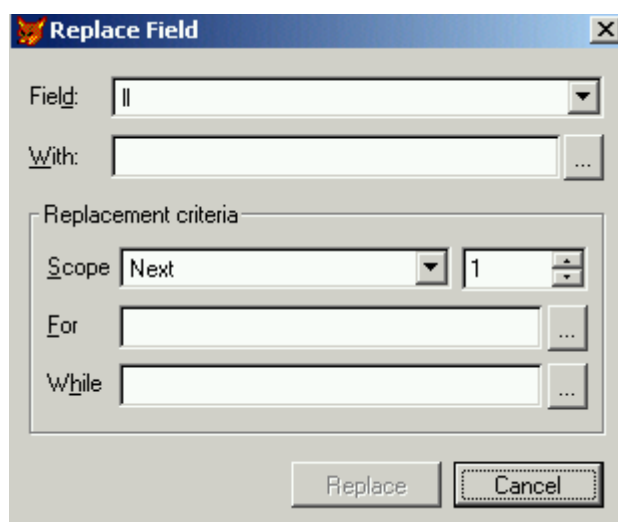
Pentru a modifica o înregistrare din tabelă se alege opțiunea **Browse** și apoi din meniul **Table**, subopțiunea **Replace Field** din meniul sistem. Apare o fereastră de dialog (**Replace Field**) cu următoarele specificații:

Field - numele câmpului ale cărui valori vrem să le modificăm;

With – noua valoare.

Scope – domeniul (**Next n** , **All**, **Record n** , **Rest**): toate înregistrările, următoarele n înregistrări, numai pentru înregistrarea n , restul înregistrărilor inclusiv cea curentă pe care este poziționat cursorul.

For / **While** – vor fi modificate doar înregistrările care satisfac condiția. **For** este o condiție construită cu **Expression Builder** (constructorul de expresii) și vor fi modificate acele înregistrări care au valoarea **True** pentru expresia dată. Clauza **While** este de asemenea o expresie logică și înregistrările selectate trebuie să verifice condiția pentru a se realiza modificarea.



Constructorul de expresii are o regiune de editare în care vor fi introduse condițiile, manual sau asistat de calculator.

Prin apăsarea butonului **Replace** are loc operația de modificare.

5.4. Închiderea bazei de date/tabelei

Cu comanda CLOSE se închid diferite tipuri de fișiere:

CLOSE DATABASES [ALL] – închide baza de date curentă și tabelele sale. Dacă nu este deschisă nici o bază de date, vor fi închise toate tabelele libere, fișierele de index, din toate zonele de lucru și se va selecta zona de lucru 1. Clauza ALL specifică că vor fi închise toate bazele de date și tabelele lor, toate tabelele libere și toate fișierele de index.

CLOSE INDEX – închide toate fișierele de index pentru zona de lucru curentă (fișiere .IDX și nestructurale .CDX). Un fișier de index structural (.CDX, compus – care se deschide automat cu tabela) nu va fi închis.

CLOSE ALL – închide toate bazele de date și tabelele lor, toate tabelele libere, toate fișierele de index din toate zonele de lucru și se selectează zona de lucru 1. Comanda nu va închide fereastra de comenzi, fereastra **Help**, fereastra **Debug**.

CLOSE TABLES [ALL] – închide toate tabelele din baza de date curentă, sau toate tabelele libere din toate zonele de lucru dacă nu există o bază de date deschisă. Includerea clauzei ALL închide toate tabelele din toate bazele de date și tabelele libere, dar bazele de date rămân deschise.

6. Utilizarea asistentului Wizard în VFP





VFP este însoțit de peste 20 de programe de tip **Wizard**, care asigură funcția de proiectare asistată de calculator. Acestea sunt programe interactive care ajută la crearea de videoformate, interogări/cereri (**Query**), importarea și redimensionarea datelor, grafice, rapoarte cu date din diferite tabele, baze de date, tabele, etichete, viziuni, kit-uri de instalare, documentație, aplicații, documente pentru pagini de WEB. Programul **Wizard** are forma unei serii de ecrane, în care utilizatorul răspunde la întrebări sau alege o serie de opțiuni, în urma cărora se va realiza o operație sau se va genera un fișier. Lansarea programului **Wizard** se poate face:

Când: se alege opțiunea **New** din meniul sistem **File** sau selectând submeniului **Wizards** din meniul sistem **Tools**.

6.1. Table Wizard

Crează tabele bazate pe structura tipică a tabelelor. Asistentul pentru tabele pune la dispoziția utilizatorului modele de tabele, acesta alegând pe cel care răspunde necesităților sale. Se poate modifica structura tabelelor și câmpurilor, fie în derularea etapelor din **Wizard** sau mai târziu după ce asistentul a salvat fișierul. Dacă sunt deschise una sau mai multe baze de date, asistentul adaugă automat noua tabelă la baza curentă. Dacă nu este deschisă nici o bază de date asistentul crează tabele libere (*free tables*).

Lansarea asistentului se face din submeniul **Wizards** din meniul sistem **Tools**, opțiunea **Table**: în următorii pași

pasul 1 – selectarea câmpurilor: din fereastra **Sample tables** se pot alege una sau mai multe tabele prin opțiunea **Add**. Pentru fiecare tabelă sunt afișate câmpurile disponibile (**Available Fields**) care pot fi adăugate în selecția câmpurilor utilizatorului (**Selected Fields**) folosind butoanele:  - adaugă câmpul selectat,  - adaugă toate câmpurile tabelii,  - elimină câmpul selectat  - elimină toate câmpurile selectate.

pasul 2 – selectarea unei baze de date: tabela creată poate fi de sine stătătoare sau poate fi adăugată la o bază de date. Dacă se crează tabela într-o bază de date asistentul va furniza opțiunile de formatare în pasul următor.

pasul 3 – modificarea setărilor câmpurilor: se pot schimba setările câmpurilor pentru tabela creată. De exemplu, se poate modifica lungimea maximă a unui câmp de tip caracter.

pasul 4 – indexarea tabelului : se selectează câmpul care va fi cheie primară de indexare în noua tabelă. Se pot desemna de asemenea alte câmpuri drept chei secundare.

pasul 5 – setarea relațiilor: dacă tabela aparține unei baze de date, se pot stabili relațiile între câmpuri, între tabela nou creată și tabelele existente în bază. Pentru definirea relației se face click cu mouse-ul pe butonul radio care identifică tipul relației, se selectează câmpul care se leagă în baza de date (dacă se selectează **New Field**, se va tipări numele) și operația se termină cu click pe butonul **OK**.

pasul 6 – terminare (butonul **Finish**): noua tabelă va fi afișată într-o fereastră de tip **Browse**. Se poate alege opțiunea **Append New Record** din meniul sistem **Table**, pentru a adăuga înregistrări noi în tabelă. După salvarea fișierului, acesta poate fi deschis mai târziu pentru a modifica structura cu aplicația **Table Designer**.

6.2. Form Wizard

Form Wizard – crează un videoformat pentru date extrase dintr-o singură tabelă. Lansarea asistentului se face utilizând meniul sistem **Tools**, submeniul **Wizard**, opțiunea **Form**, în următorii pași:

pasul 1 – selectarea câmpurilor: se face pe baza opțiunilor utilizatorului exprimate în trei ferestre: **Databases and Tables**, **Available Fields**, **Selected Fields**, la fel ca în cazul **Wizard Table**.

pasul 2 – selectarea stilului videoformatului – specifică stilul controalelor din videoformat. În fereastra **Style** este afișată o listă cu tipurile de videoformate: **Standard**, **Chiseled**, **Shadowed**, **Boxed**, **Embossed**, **Fancy** etc. Se poate alege din panoul cu butoane radio (**Button Type**) tipul butoanelor utilizate: **Text Buttons**, **Picture Buttons**, **No Buttons**, **Custom Bbuttons**. Tipul de butoane ales va genera în videoformat, butoane pentru navigare în fișier (**Top**, **Previous**, **Next**, **Bottom**), **Find** (afișează cutia de dialog **Search** - caută), **Print** (tipărește un raport), **Add** (adaugă o nouă înregistrare), **Delete** (șterge înregistrarea curentă), **Exit** (închide videoformatul).

pasul 3 – sortarea înregistrărilor: se vor selecta acele câmpuri care dorim să fie sortate. Se poate alege de asemenea o etichetă (intrare) pentru un index (TAG).

pasul 4 – terminare (butonul **Finish**): dacă s-a ales un număr mare de câmpuri la pasul 1 și pentru a fi siguri că toate câmpurile vor încăpea în videoformat se poate alege opțiunea **Add pages for fields that do not fit**. În caz contrar, dacă numărul câmpurilor depășește dimensiunea

videoformatului, VFP va afișa formatul cu bare de deplasare (**Scroll Bars**). Videoformatul poate fi vizualizat înainte de salvare cu opțiunea **Preview**. După salvarea videoformatului, acesta poate fi deschis mai târziu și modificat cu aplicația **Form Designer**.

6.3. Report Wizard

Crează rapoarte utilizând o singură tabelă liberă sau o viziune într-o bază de date. Asistentul pune întrebări într-o succesiune de pași, prin care se specifică sursa și câmpurile ce vor fi utilizate pentru a crea controalele raportului. Pentru lansarea asistentului, se alege din meniul sistem **Tools**, submeniul **Wizard**, opțiunea **Report**. Din caseta de dialog **Wizard Selection** se alege **Report Wizard** și se urmează pașii:

pasul 1 – selectarea câmpurilor: se introduc opțiunile în cele trei ferestre de dialog: **Databases and Tables**, **Available Fields**, **Selected Fields**. Selecția poate fi făcută dintr-o tabelă sau o viziune.

pasul 2 – gruparea înregistrărilor: se poate folosi gruparea datelor pentru a împărți pe categorii și a sorta înregistrările, pentru a fi mai ușor de citit (de exemplu situația încasării impozitelor pe familie, pe sectoare și centralizat pe municipiu). După alegerea unui câmp în căsuța **Group by**, se pot alege opțiunile **Grouping Options** și **Summary Option** pentru a îmbunătăți selecția. Alegând **Grouping Options** se va deschide caseta de dialog **Grouping Intervals** care permite selectarea nivelurilor de filtrare relevante pentru tipul de date conținute în câmpurile selectate pentru grupare. Alegând **Summary Options** se deschide o cutie de dialog în care utilizatorul poate specifica tipul de calcul care se va aplica unui câmp numeric:

Tipuri de calcul	Rezultat
SUM	Suma valorilor din câmpul numeric specificat
AVG	Media valorilor din câmpul numeric specificat
COUNT	Numărarea înregistrărilor care conțin valori non-NULL din câmpul specificat
MIN	Valoarea minimă în câmpul numeric specificat
MAX	Valoarea maximă în câmpul numeric specificat

Se pot alege de asemenea opțiunile **Summary only** sau **No Totals** pentru raport. Câmpurile care sunt selectate pentru grupare nu vor fi disponibile la pasul 3.

pasul 3 – alegerea stilului pentru raport : sunt disponibile următoarele stiluri: **Executive, Ledger, Presentation, Banded, Casual**, asistentul prezentând într-un grafic, sub o lupă, un exemplu al stilului ales.

pasul 4 – definirea formatului raportului: când se specifică un număr de coloane sau se selectează una din opțiunile pentru format, asistentul prezintă într-un grafic, sub o lupă, un exemplu. Dacă s-a făcut opțiunea de grupare la pasul 2, opțiunile **Columns** și **Field Layout** nu mai sunt disponibile.

pasul 5 – sortarea înregistrărilor: se pot selecta unul sau mai multe câmpuri sau o etichetă de index (TAG) în ordinea în care se dorește să se sorteze înregistrările (ascendent, descendent).

pasul 6 – terminare (butonul **Finish**): dacă numărul câmpurilor selectate nu încap pe o singură linie pe lățimea paginii de raport, acest câmp va fi înglobat pe linia următoare. Dacă nu vrem să fie înglobat câmpul pe linia următoare, se deselectează opțiunea **Wrap fields that do not fit**. Cu butonul **Preview** se poate vizualiza raportul fără a părăsi asistentul. După salvarea raportului, acesta poate fi deschis ulterior și modificat cu aplicația **Report Designer**.

6.4. Label Wizard

Acesta este asistentul pentru generarea etichetelor cu date preluate dintr-o tabelă. Pentru lansarea asistentului se alege din meniul sistem **Tools**, submeniul **Wizard**, opțiunea **Label** și se urmăresc pașii:

pasul 1 – selectarea tabelelor: se selectează o tabelă sau o viziune.

pasul 2 – alegerea tipului de etichetă: asistentul oferă o listă cu tipurile de etichete standard care sunt instalate odată cu VFP. Asistentul afișează de asemenea lista cu tipurile de etichete care au fost create de utilizator, prin folosirea aplicației **AddLabel**.

pasul 3 - definirea formei etichetei: se adaugă câmpurile în ordinea în care trebuie să apară pe etichetă. Se pot folosi semne de punctuație, spații, pentru a formata eticheta. Se folosește căsuța **Text** pentru a adăga textul. Pe măsură ce se adaugă elemente în etichetă, asistentul va arăta modificările făcute. Utilizatorul poate verifica imaginea etichetei pentru a vedea dacă câmpurile introduse se încadrează în dimensiuni. De asemenea, se poate folosi butonul **Font** pentru a schimba stilul și mărimea literelor folosite în etichetă.

pasul 4 – sortarea înregistrărilor: se selectează câmpurile sau etichetele de index (TAG) în ordinea în care dorește să sorteze înregistrările.

pasul 5 – terminare (butonul **Finish**): utilizatorul poate activa opțiunea **Preview** pentru a verifica restricțiile impuse, înainte de terminare. După salvarea etichetei, aceasta poate fi deschisă și modificată cu aplicația **Label Designer**.

6.5. Query Wizard

Asistentul de interogare, care pe baza tabelelor sau viziunilor selectate pentru a furniza informații, extrage înregistrările care îndeplinesc criteriile de filtrare și direcționează rezultatele către tipul de ieșire specificat: fereastră **Browse**, raport, tabelă, etichetă etc. Pentru lansarea asistentului se alege din meniul sistem **Tools**, submeniul **Wizard**, opțiunea **Query Wizard** și se urmăresc pașii:

pasul 1 – selectarea câmpurilor: se pot selecta câmpuri din diferite tabele sau viziuni. Mai întâi se selectează câmpurile dintr-o tabelă sau viziune, se mută în căsuța **Selected fields**, utilizând butoanele de adăugare; apoi se selectează câmpurile din altă tabelă sau viziune etc.

pasul 2 – stabilirea relațiilor între tabele: utilizatorul poate selecta câmpurile comune mai multor fișiere din lista afișată, pentru a stabili relațiile între tabele sau viziuni.

pasul 3 – includerea înregistrărilor: se poate limita interogarea (**Query**) alegând din două tabele, numai liniile care coincid, sau toate liniile dintr-unul din tabele. Implicit numai înregistrările care coincid sunt incluse. Dacă se dorește crearea unei joncțiuni interne se selectează opțiunea **Only matching row** (numai înregistrările care coincid). Dacă se dorește crearea unei joncțiuni externe, se selectează toate înregistrările din unul din cele două tabele.

pasul 4 – filtrarea înregistrărilor: se poate reduce numărul înregistrărilor selectate în interogare, prin crearea de expresii logice care să filtreze înregistrările din tabelele sau viziunile alese. Se pot crea două expresii logice, legate cu operatorul **And**, care va permite selectarea numai a acelor înregistrări care satisfac ambele criterii. Utilizarea operatorului **Or** face ca în selecția înregistrărilor, să intre acelea care satisfac doar unul din criterii. Opțiunea **Preview** permite vizualizarea înregistrărilor care vor fi incluse în interogare (**Query**), pe baza aplicării criteriilor de filtrare.

pasul 5 – sortarea înregistrărilor: se pot selecta până la 3 câmpuri (chei de sortare) sau o etichetă de index (TAG) pentru a stabili ordinea în care rezultatele interogării vor fi sortate (ascendent sau descendent).

pasul 6 – limitarea înregistrărilor: se poate limita suplimentar numărul de înregistrări în interogare, fie indicând un procent de vizualizare din numărul de înregistrări cu valorile cele mai mari (sortare descendentă) / mai mici (sortare ascendentă), fie indicând numărul de înregistrări cu valorile cele mai mari/mai mici, care vor fi afișate. De exemplu, pentru a vedea doar primele 10

înregistrări cu valoarea cea mai mare din interogare (sortare descendentă), se introduce la opțiunea **Number of records** valoarea 10 în căsuța **Portion value**. Cu opțiunea **Preview** se pot vizualiza rezultatele interogării pe baza restricțiilor impuse.

pasul 7 – terminare (butonul **Finish**): după ce interogarea a fost salvată, ea poate fi deschisă ulterior, lansând aplicația **Query Designer**.

6.6. Mail Merge Wizard

Acesta este asistentul pentru documente interclasate (fuziune / unire) de tip Word sau fișiere de tip text care sunt compatibile cu orice procesor de text. Pentru a executa aplicația, trebuie să fie instalată o versiune Microsoft Word curentă precum și protocolul standard pentru servere baze de date, ODBC (Open Database Connectivity). ODBC se instalează pentru diferite tipuri de baze de date și va permite programului VFP 6.0 să se conecteze la baza de date și să acceseze datele. Pentru lansarea asistentului se alege din meniul sistem **Tools**, submeniul **Wizard**, opțiunea **Mail Merge** și se continuă cu pașii:

pasul 1 – selectarea câmpurilor: se pot selecta câmpuri doar dintr-o singură tabelă sau viziune.

pasul 2 – alegerea procesorului de text: trebuie să fie instalată o versiune curentă de Microsoft Word în cazul în care se alege opțiunea Microsoft Word. Dacă se selectează opțiunea **Comma-delimited text file**, asistentul va sări peste pașii 3 și 4 și va merge direct în ecranul de terminare.

pasul 3 – selectarea tipului de document: se crează un nou document sau se adaugă date la un document existent.

pasul 4 – selectarea stilului documentului: se selectează unul din tipurile de fișiere pe care le recunoaște editorul Word. Dacă se crează un document nou, se va specifica tipul documentului **Form Letter, Label, Envelope, Catalog**.


pasul 5 – terminare (butonul **Finish**): dacă s-a selectat crearea unui document nou în Word, asistentul va deschide documentul în Microsoft Word. Dacă s-a selectat opțiunea de creare a unui fișier de tip text, asistentul îl va salva.

Introducerea unui câmp al bazei de date în documentul Word se face astfel:

- se poziționează cursorul în locul din document unde se dorește inserarea câmpului;



pe bara de instrumente pentru interclasare din opțiunea pop-up **Insert Merge Field** se alege numele câmpului ce trebuie inserat;

- activând opțiunea  - **View Merged Data** (vizualizarea datelor care au fuzionat), se poate vedea efectiv valoarea câmpului selectat pentru inserare. Butoanele pentru deplasare la începutul / sfârșitul fișierului sau secvențial înainte / înapoi, vor muta pointerul de fișier pe înregistrarea corespunzătoare, afișând noua valoare a câmpului, în documentul Word.

6.7. Editorul de texte în VFP 6.0

Mediul VFP 6.0 pune la dispoziția utilizatorului un editor de texte, pentru crearea fișierelor de tip text, aplicație care se deschide într-o fereastră de tip Windows.

6.7.1. Lansarea editorului

Lansarea editorului se face fie cu una din comenzile:

MODIFY FILE <nume_fișier> && modificare fișier de tip text;

MODIFY COMMAND <nume_program> && modificare program sursă.

introduse în fereastra de comenzi, fie din meniul sistem se alege meniul **File**, submeniul **New**, opțiunea **Program / Text File**. În editor se pot folosi pentru deplasare: tastele cu săgeți direcționale, **[Page Up]** - o pagină ecran în sus, **[Page Down]** - o pagină ecran în jos, **[Home]** - la începutul rândului, **[End]** - la sfârșitul rândului curent.

6.7.2. Lucrul cu blocul de linii din fișier

Asupra unei linii sau unui bloc de linii de text se pot executa următoarele operații:

marcarea blocului – se ține tasta **[Shift]** apăsată și cu una din săgețile direcționale, sau tastele **[Home]**, **[End]**, **[Page Up]**, **[Page Down]** se marchează liniile de text (care vor apare în video-invers).

copierea blocului de linii marcat în Clipboard – se face:

folosind opțiunea **Cut** din meniul **Edit** sau apăsând simultan tastele **[Ctrl] + [X]**, textul selectat va fi șters din fișier;

folosind combinația de taste **[Ctrl]+[C]**, caz în care textul rămâne în fișier.

inserarea la o nouă poziție în fișier a textului din Clipboard – se face utilizând opțiunea **Paste** din fereastra **Edit** sau combinația de taste **[Ctrl] + [V]**.

deselectarea unui bloc de linii – se face fie apăsând una din săgețile direcționale sau butonul stâng de la mouse.

Programele în VFP au ca regulă generală, scrierea instrucțiunilor pe o singură linie (255 caractere). Dacă totuși o instrucțiune trebuie să fie scrisă pe mai multe rânduri, la sfârșitul fiecărei linii se pune un caracter de continuare ; .

7. Elemente ale limbajului propriu Visual FoxPro

7.1. Simboluri

Pentru scrierea comenzilor (instrucțiunilor) se folosesc:

- literele alfabetului latin, litere mari / litere mici, compilatorul nefăcând diferență între cele două tipuri;
- cifrele 0..9;
- caractere speciale: +, -, *, /, (,), <, >, =, &, @, !, ' , " , #, \$, %, ^, \, |, _ , spațiu, ;, virgula, ?.

7.2. Variabile de sistem

Sunt variabile proprii sistemului, predefinite, la care utilizatorul are acces. Folosirea lor în program nu implică operații de inițializare, incrementare. De exemplu, variabila `_PAGENO=n` conține numărul paginii *n* dintr-un raport, cu ajutorul ei putându-se controla afișarea paginii *n* la monitor sau la imprimantă.

7.3. Comentariul

Se folosește pentru a da lămuriri asupra a ceea ce face programul sursă, explicații asupra acțiunilor programului, numelor variabilelor etc, pentru a indica începutul unei linii neexecutabile în program. Sunt admise pentru comentariere:

caracterul * plasat la începutul liniei va comentaria întreagul rând (fără spațiu de demarcare);

caracterele && plasate la începutul / sfârșitul liniei, cu cel puțin un spațiu de demarcare față de textul propriu-zis;

cuvântul rezervat NOTE, pentru a comentaria mai multe rânduri de text, se introduce în continuare prima linie de comentariu (la distanță de minim un spațiu), se marchează continuarea comentariului pe rândul următor cu caracterul punct și virgulă (;), închiderea comentariului se face cu caracterul punct.

7.4. Tipuri de date, operații și funcții

Datele (variabile sau constante) utilizate în programare pot fi simple sau sub formă de masiv de date (vectori, matrici, care presupun o ordonare a datelor după un criteriu de poziție). La rândul lor aceste două categorii pot fi alcătuite din date de tip:

- *logic*, care nu pot lua decât două valori: **.T.** | **.Y.** (adevărat) și **.F.** | **.N.** (fals);
- *numeric*, numere întregi sau fracționare;
- *șir de caractere*, reprezintă o mulțime ordonată de caractere, fiecare caracter având un număr de ordine în șir, numerotarea începând cu 1. Șirurile de caractere (constante) se scriu între apostrof sau ghilimele;
- *calendaristic*, de exemplu 01/12/2007.

Se pot construi expresii, combinând datele (operanzi) cu operații specifice (operatori). Operatorii la rândul lor sunt de tip:

- *logic*, lucrează cu toate tipurile de date și returnează o valoare logică (**.T.** / **.F.**)

Operator	Acțiune	Cod
()	gruparea expresiilor logice	<i>Cvar1</i> AND (<i>Cvar2</i> AND <i>Cvar3</i>)
NOT, !	negare logică	IF NOT <i>CvarA</i> = <i>CvarB</i> sau IF ! <i>Nvar1</i> = <i>Nvar2</i>
AND	ȘI logic	<i>LvarX</i> AND <i>LvarY</i>
OR	SAU logic	<i>LvarX</i> OR <i>LvarY</i>

unde *Cvar* – variabilă de tip caracter, *Nvar* – variabilă de tip numeric, *Lvar* – variabilă de tip logic.

- *relațional*, lucrează cu toate tipurile de date, expresia este evaluată și se returnează o valoare logică **.T.** / **.F.** (adevărat / fals).

Operator	Acțiune	Cod
<	mai mic decât	? 23 < 54
>	mai mare decât	? 1 > 2 sau ? .t. > .f.
=	egal cu	? <i>cVar1</i> = <i>cVar</i>

<>, #, !=	diferit de	? .T. <> .F.
<=	mai mic egal cu	? {^1998/02/16} <= {^1998/02/16}
>=	mai mare egal cu	? 32 >= n_ani
==	comparare șiruri de caractere	? status == "Open"

În cazul variabilelor de tip caracter operatorii relaționali funcționează prin compararea codurilor ASCII corespunzătoare caracterelor din cele două șiruri, comparația făcându-se caracter cu caracter, de la stânga la dreapta.

Dacă șirurile care se compară nu au aceeași lungime, implicit se completează șirul mai scurt cu codul ASCII pentru caracterul 0.

Operația de egalitate a două șiruri de caractere este controlată de comanda:

SET EXACT ON|OFF

prin clauza OFF, implicită, se consideră că cele două șiruri sunt egale, în cazul când caracterele coincid pe lungimea celui mai scurt.

prin clauza ON, egalitatea se verifică pe lungimea șirului mai lung, spațiile de la sfârșitul șirurilor sunt ignorate.

- *numeric*, lucrează cu toate tipurile de date numerice.
-

Operator	Acțiune	Cod
()	gruparea subexpresiilor	(4-3) * (12/Nvar2)
**, ^	exponent (putere)	? 3 ** 2 sau ? 3 ^ 2
*, /	înmulțire, împărțire	? 2 * 7 sau ? 14 / 7
%	restul împărțirii (modulo)	? 15 % 4
+, -	adunare, scădere	? 4 + 15

Ordinea operatorilor din tabelă este cea folosită în matematică.

Operația modulo (%) se mai poate executa apelând funcția $\text{MOD}(n1,n2)$ care va calcula restul împărțirii numărului $n1$ la numărul $n2$.

- *caracter*, care permite concatenarea, compararea datelor de tip șir de caractere, utilizând următorii operatori:

Operator	Acțiune	Cod
+	<i>Concatenare</i> . Se unesc două șiruri de caractere, un șir și un câmp dintr-o înregistrare sau un șir de caractere și o variabilă	? 'Bună ' + 'ziua'
-	<i>Concatenare</i> . Se îndepărtează spațiile goale de la sfârșitul elementului care precede operatorul, apoi se unesc cele două elemente.	? 'Bună ' + 'seara'
\$	<i>Comparare</i> . Se caută un șir de caractere în interiorul altui șir de caractere.	? 'punct' \$ 'contrapunct' ? 'principal' \$ clienti.adresa

7.4.1. Date de tip numeric. Funcții aritmetice și financiare

Expresiile de tip numeric pot fi:

- câmpuri de tip numeric (N), întreg (I), virgulă mobilă (F) dintr-o tabelă;
- funcții care returnează o variabilă (constantă) numerică;
- variabile și constante de tip numeric.

Scop / reprezentare matematică	Funcție VFP
restul împărțirii exacte a lui n_1 la n_2	$\text{mod}(n_1, n_2)$
$ x $	$\text{abs}(x)$
partea întreagă a lui x	$\text{int}(x)$
următorul întreg mai mare sau egal cu x (plafon)	$\text{ceiling}(x)$
următorul întreg care este mai mic sau egal cu x (podea)	$\text{floor}(x)$
rotunjirea unei expresii numerice la un număr specificat de zecimale	$\text{round}(nExpr, nr_zecimale)$
e^x	$\text{exp}(x)$
$\ln(x)$	$\text{log}(x)$

$\log_{10} x$	<code>log10(x)</code>
\sqrt{x}	<code>sqrt(x)</code>
$\sin x$	<code>sin(x)</code>
$\cos x$	<code>cos(x)</code>
$\operatorname{tg} x$	<code>tan(x)</code>
$\arcsin x$	<code>asin(x)</code>
$\arccos x$	<code>acos(x)</code>
$\operatorname{arctg} x$	<code>atan(x)</code>
trecerea din grade în radiani	<code>dtor(x)</code>
trecerea din radiani în grade	<code>rtod(x)</code>
alegerea numărului de zecimale n	<code>set decimals to n</code>
valoarea constantei π	<code>PI()</code>

Funcții financiare

Scop	Funcție VFP
returnează valoarea ratei necesare pentru achitarea unui împrumut s , cu dobânda d^* , pe perioada nr^{**} (annual sau lunar)	<code>PAYMENT(s,d,nr)</code>
returnează suma ce trebuie depusă în cont pentru a plăti o rată s , pe o perioadă nr (ani sau luni), dacă dobânda acordată este d	<code>PV(s,d,nr)</code> (Present Value)
returnează suma ce se poate strânge în cont, după o perioadă nr (ani sau luni), dacă dobânda este d iar rata depunerii este s	<code>FV(s,d,nr)</code> (Future Value)

*Argumentul d (dobânda) este utilizat în formă zecimală ($d=0.06$ | 6%). Dacă dobânda se preia sub formă procentuală se va împărți la 100. Implicit este dobânda anuală.

**Argumentul nr (perioada) este implicit exprimat în ani. Dacă calculul se efectuează pentru o perioadă exprimată în luni, va trebui modificată corespunzător și dobânda (anuală | lunară).

7.4.2. Aplicații ale funcțiilor financiare

A. Funcția `PAYMENT` returnează valoarea fiecărei plăți, dintr-o serie periodică de plăți (rată), a unui împrumut s , cu dobânda fixă d , pe o perioadă nr .

Sintaxa funcției: `PAYMENT(s,d,nr)`

Exemplul 1. Firma DAEWOO vinde autoturisme marca MATIZ care pot fi achitate în rate lunare, pe o perioadă de trei ani. Împrumutând de la BRD suma de 6000 \$ cu dobânda anuală de 5.3%, să se stabilească care este rata lunară pe care trebuie să o achite un client.

În fereastra de comenzi introducem comanda pentru crearea fișierul sursă IMPR:

MODI COMM IMPR

se lansează editorul de programe VFP și se introduc liniile de comenzi (instrucțiuni):

SET TALK OFF && anulează afișarea rezultatelor comenzilor

CLEAR && curăță spațiul de afișare

SET CURRENCY TO '\$' && definirea simbolului monedei

STORE 0 TO S,NR && inițializarea variabilelor S, NR cu 0

D=0.00 && inițializarea variabilei D

@5,20 SAY 'STABILIREA RATEI LUNARE PENTRU UN IMPRUMUT'

@7,15 SAY 'SUMA IMPRUMUTATA ?:' GET S

NOTE afișarea la coordonate fixate de carcaterul @ (linia 7, coloana 15) a mesajului dintre apostrof și editarea variabilei S.

@9,15 SAY 'DOBANDA ANUALA (%) ?:' GET D

@11,15 SAY 'IN CATE LUNI TREBUIE ACHITAT IMPRUMUTUL ?:' GET NR

READ && citirea întregului ecran – variabilele editate cu @.. GET

D=D/12/100 && transformare dobândă anuală procentuală, în dobândă lunară, exprimată zecimal

@15,20 SAY 'RATA LUNARA ESTE DE '

SET CURRENCY RIGHT && setarea afișării simbolului monedei la dreapta

@15,45 SAY PAYMENT(S,D,NR) FUNCTION '\$9999.99' && afișarea rezultatului funcției și a monedei

WAIT WINDOW 'APASATI ENTER' && păstrează ecranul cu rezultate până se apasă Enter

Fereastra de editare se închide cu [CTRL] + [W] sau din butonul de închidere și se salvează fișierul. Lansarea în execuție a programului se face fie din meniul sistem **Program/Do**, fie din bara de instrumente se activează icon-ul cu semnul exclamării, sau în fereastra de comenzi se introduce comanda

DO IMPR

Rata lunară este 180.63 \$.

B. Funcția PV returnează valoarea sumei care trebuie să existe în cont, pentru a putea plăti o rată s , pe o perioadă nr , pentru o dobândă anuală d acordată de bancă la depozit.

Sintaxa funcției: $PV(s,d,nr)$

Exemplul 2. O firmă trebuie să achite o chirie lunară de 300 \$ pentru sediu, pe o perioadă de 5 ani. Câți bani trebuie depuși în cont la banca BCR, dacă dobânda la acest tip de depozit este de 4% pe an.

În fereastra de comenzi se introduce comanda pentru crearea fișierului CONT:

MODI COMM CONT

se lansează editorul de programe VFP și se introduc liniile de comenzi:

SET CLOCK ON && afișarea cesul cu ora dată de sistemul de operare

CLEAR

STORE 0 TO S,NR

D=0.00

@5,25 SAY 'STABILIREA SUMEI NECESARE IN CONT PENTRU PLATA UNEI RATE LUNARE'

@7,20 SAY 'RATA LUNARA ?:' GET S

@9,20 SAY 'DOBANDA ANUALA (%) ?:' GET D

@11,20 SAY 'PERIOADA (LUNI) DE ACHITARE A RATEI ?:' GET NR

READ

D=D/12/100

@13,25 SAY 'SUMA INITIALA DIN CONT TREBUIE SA FIE DE ' ;

+ALLTRIM(STR(PV(S,D,NR),10,2))+ ' \$'

NOTE funcția STR() transformă o valoare numerică într-un șir de caractere;

funcția ALLTRIM() elimină spațiile de la începutul și sfârșitul șirului.

WAIT WINDOW 'APASATI ENTER'

Se salvează programul și se lansează în execuție cu comanda:

DO CONT

Suma necesară este 16289.72\$.

C. Funcția FV returnează valoarea sumei ce se strânge într-un cont, în cazul în care se depune o rată s , banca oferă o dobândă anuală d , pe o perioadă nr .

Sintaxa funcției: $FV(s,d,nr)$

Exemplu. Ce sumă va strânge în cont un copil, care primește pensie alimentară timp de 10 ani, câte 50 \$ pe lună, la o bancă ce oferă o dobândă de 4% pe an ?

În fereastra de comenzi se introduce comanda pentru crearea fișierului SUMA:

MODI COMM SUMA

se lansează editorul de programe VFP și se introduc liniile de comenzi:

CLEAR

STORE 0 TO S,NR

D=0.00

@5,25 SAY 'STABILIREA SUMEI STRANSE IN CONT DUPA O PERIOADA DE TIMP'

@7,20 SAY 'RATA LUNARA ?:' GET S

@9,20 SAY 'DOBANDA ANUALA ?:' GET D

@11,20 SAY 'PERIOADA (LUNI) DE DEPUNERE A RATEI ?:' GET NR

READ

D=D/12/100

@13,25 SAY 'SUMA STRANSA IN CONT ESTE '+ALLTRIM(STR(FV(S,D,NR),10,2))+ ' \$'

WAIT WINDOW 'APASATI ENTER'

Se salvează programul și se lansează în execuție cu comanda:

DO SUMA

Suma strânsă în cont este 7462.49\$.

7.4.3. Date și funcții de tip caracter

Operanzii de tip caracter pot fi:

- câmpurile de tip CHARACTER ale unei tabele;
- funcțiile care returnează un șir de caractere;
- variabile și constante de tip șir de caractere.

Dacă șirul de caractere conține în componența sa caracterul ‘ (‘) atunci pentru definirea șirului se folosesc caracterele “ (”).

Scop	Funcție VFP
Returnează caracterul corespunzător codului ASCII n ($I=1...127$)	CHR(n)
Returnează codul ASCII pentru un caracter c	ASC('c')
Extragerea unui subșir de de caractere, de lungime n_2 , din șirul s , începând cu poziția n_1	SUBSTR(s, n_1, n_2)
Extragerea unui subșir de n caractere, începând din stânga șirului s	LEFT(s, n)
Extragerea unui subșir de n caractere, începând din dreapta șirului s	RIGHT(s, n)
Returnarea unui șir de caractere s , în mod repetat de n ori	REPLICATE(s, n)
Obținerea unui șir de n spații goale	SPACE(n)
Eliminarea spațiilor într-un șir de caractere:	
de la începutul și sfârșitul șirului s	ALLTRIM(s)
de la începutul șirului s	LTRIM(s)
de la sfârșitul șirului s	RTRIM(s)
Adăugarea de spații, sau a unui caracter c_pad , într-un șir s pentru a ajunge la lungimea n :	
la ambele capete	PADC($s, n[, c_pad]$)
la stânga	PADL($s, n[, c_pad]$)
la dreapta	PADR($s, n[, c_pad]$)
Returnează lungimea unui șir de caractere s	LEN(s)
Returnează poziția de început, la a n -a apariție, a unui subșir de caractere ss într-un șir de caractere s	AT(s, ss, n)
Returnează poziția de început, la a n -a apariție, a unui subșir de caractere ss într-un șir de caractere s , fără a se ține seama de litere mari/mici	ATC(s, ss, n)
Returnează poziția numerică, la a n -a apariție, a unui subșir de caractere ss într-un șir de caractere s , începând căutarea de la dreapta	RAT(s, ss, n)
Transformarea caracterelor unui șir s în litere mari	UPPER(s)
Transformarea caracterelor unui șir s în litere mici	LOWER(s)
Transformarea primului caracter al unui șir s , dacă este o literă, în majusculă	PROPER(s)

7.4.4. Date și funcții de tip calendaristic

Expresiile de tip dată calendaristică pot fi:

- câmpuri de tip dată calendaristică (DATE), dintr-o tabelă;
- funcții care returnează data calendaristică;
- constante de tip dată calendaristică.

O dată de tip dată calendaristică precizează ziua, luna, anul, ordinea acestor 3 elemente poate fi aleasă din 11 moduri, cu ajutorul comenzii:

SET DATE TO <tip_dată>

unde:

tip_dată	Format
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/dd

Cele 3 elemente pot fi separate cu '/', '-', '.'. Implicit se consideră formatul: ll/zz/aa.

Cu comanda:

SET CENTURY ON|OFF

se poate preciza dacă anul este afișat cu 2 cifre (ON) sau 4 cifre (OFF).

Comanda:

SET MARK TO <character>

precizează ce caracter se va folosi ca separator între cele 3 elemente ale datei.

Funcții pentru datele de tip dată calendaristică sunt date în tabelul următor:

Scop	Funcție VFP
Returnează data din sistem(calculator)	DATE()
Numele zilei dintr-o expresie de tip dată <i>d</i>	CDOW(<i>d</i>)
A câta zi din săptămână dintr-o expresie de tip dată <i>d</i>	DOW(<i>d</i>)
Numele lunii dintr-o expresie de tip dată <i>d</i>	CMONTH(<i>d</i>)
A câta lună din an dintr-o expresie de tip dată <i>d</i>	MONTH(<i>d</i>)
Izolarea anului dintr-o expresie de tip dată <i>d</i>	YEAR(<i>d</i>)
Returnează ora din sistem (calculator)	TIME()
Returnarea sub formă de șir de caractere, în format <i>aaaallzz</i> a unei expresii de tip dată calendaristică <i>d</i>	DTOS(<i>d</i>)

Folosirea într-un program a unei constante de tip dată calendaristică, se face între acolade ({^31/01/01}).

7.4.5. Date de tip memo

Acest tip de date este asemănător tipului șir de caractere. Este indicată folosirea datelor de tip memo în cazul în care un câmp al unei înregistrări dintr-o tabelă, nu are o lungime cunoscută (sau care nu poate fi aproximată). Utilizarea unui câmp memo într-o tabelă, are ca efect asocierea la tabelă a unui fișier suplimentar în care se depun datele câmpului memo. În tabelă, în câmpul memo sunt stocate informații referitoare la tabela suplimentară. Lungimea unui câmp memo este 10 octeți.

7.5. Variabile și masive

7.5.1. Variabile

O variabilă are asociate următoarele elemente:

- numele
- conținutul (valoarea)
- tipul variabilei
- zonă de memorie

Din punct de vedere al utilizării lor, variabilele pot fi:

- *locale*, acționează într-o funcție sau procedură. Declararea utilizării lor se face cu comanda:

LOCAL <listă_variabile>

<listă_variabile> - variabilele se declară prin nume și sunt separate în listă prin virgula.

- *globale*, acționează la nivelul întregului program, inclusiv în funcții și proceduri. Declararea utilizării lor se face cu comanda:

PUBLIC <listă_variabile>

Atribuirea de valori unei variabile se face cu sintaxa:

<nume_var>=<expresie>

Efectul comenzii: se evaluează expresia din dreapta, se caută dacă variabila a fost definită și i se atribuie valoarea și tipul expresiei.

Atribuirea se mai poate face utilizând comanda:

STORE <expresie> TO <listă_variabile>

Efectul comenzii: se evaluează expresia și se atribuie variabilelor din listă, valoarea și tipul ei.

Eliberarea zonelor de memorie ocupate de variabile care nu mai sunt necesare în program se face cu una din comenzile:

RELEASE [ALL] <listă_variabile>

CLEAR [ALL] <listă_variabile>

Afișarea conținutului variabilelor existente în memorie se face cu comenda:

DISPLAY MEMORY [TO PRINTER[PROMPT] | TO FILE *nume_fișier*] [NOCONSOLE]

unde:

TO PRINTER[PROMPT] – ieșirea la imprimantă. Clauza PROMPT se folosește pentru a confirma imprimarea, într-o fereastră de dialog;

TO FILE *nume_fișier* – ieșirea direcționată către un fișier;

[NOCONSOLE] – împiedică afișarea în fereastra principală VFP, a rezultatelor comenzii.

7.5.2. Macrosubstituția

Macrosubstituția tratează conținutul unei variabile ca un șir de caractere în sens literal (câmpul unei înregistrări dintr-o tabelă, nume de fișier etc.), ca și cum în locul variabilei respective ar fi pus șirul de caractere fără apostrofuri.

Sintaxa comenzii:

& *nume_variabilă*

Exemplu. Presupunem că există 2 fișiere ESTUD1.DBF și ESTUD2.DBF. Se dorește să li se vizualizeze conținutul într-un ciclu FOR.

```

PUBLIC FIS C(20)
FOR I=1 TO 2
    FIS='ESTUD'+ALLTRIM(STR(I))+'.DBF'
    USE &FIS
    BROWSE
ENDFOR
CLOSE DATABASES

```

7.5.3. Masive de date

Masivele de date (vectori și matrice) pot fi privite ca variabile sau constante, care au o structură compusă, identificarea în cadrul structurii făcându-se pe baza unui criteriu de poziție. Pentru vector criteriul de poziție este numărul de ordine al elementului, pentru matrice, criteriul de poziție este fie numărul liniei și al coloanei, fie liniarizând matricea, se poate adresa un element prin poziția care o ocupă în masiv (de exemplu, într-o matrice A de dimensiune 3×3 , elementul $A(3,2)$ este al 8-lea element din matrice și se poate adresa cu $A(8)$). Prin liniarizare o matrice poate fi adresată ca un vector. Calculul poziției în matricea liniarizată (vector), pe baza coordonatelor linie/coloană se poate face cu formula:

$$\text{poziție} = (\text{linie} - 1) * \text{Nr_linii} + \text{coloană}$$

Utilizarea masivelor de date comportă următoarele aspecte:

- atribuirea unui nume pentru identificare;
- declararea tipului de masiv: vector sau matrice (rezervare zonă de memorie);

Masivele de date pot fi declarate în două moduri:

- fie la începutul programului sau al subprogramului, utilizând comanda PUBLIC (*variabilă globală*) sau LOCAL (*variabilă locală*):

```

PUBLIC [ARRAY] nume_masiv1(n_linie[,n_col])[ , nume_masiv2(n_linie[,n_col])...
LOCAL [ARRAY] nume_masiv1(n_linie[,n_col])[ , nume_masiv2(n_linie[,n_col])...

```

Variabilele create cu PUBLIC / LOCAL sunt inițializate cu valoarea fals (**.F.**). Variabilele locale sunt legate de funcția sau procedura în care au fost declarate, dar pot fi transmise prin referință;

- fie în interiorul programului utilizând una din comenzile:

DIMENSION [ARRAY] *nume_masiv1*(*n_linie*[,*n_col*])[, *nume_masiv2*(*n_linie*[,*n_col*)...

DECLARE [ARRAY] *nume_masiv1*(*n_linie*[,*n_col*])[, *nume_masiv2*(*n_linie*[,*n_col*)...

Elementele masivului sunt inițializate cu valoarea fals ca și în primul caz.

După declarare, masivul poate fi inițializat global (toate elementele capătă aceeași valoare) și cu această operație se stabilește tipul de date al elementelor masivului:

STORE <val_init> TO <nume_masiv>

Exemplu

DIMENSION MAT(5,7)

STORE 0 TO MAT

Tipul și dimensiunile unui masiv de date pot fi modificate în cadrul aceluiași program, prin noi declarații DIMENSION, DECLARE.

7.6. Funcții de prelucrare a masivelor

Funcția ALEN (nume_masiv,n)

Returnează următoarele informații:

- numărul de elemente, dacă $n=0$;
- numărul de linii, dacă $n=1$;
- numărul de coloane, dacă $n=2$.

Funcția AINS(nume_masiv,n[,2])

Are ca efect:

- inserarea unui element nou într-un vector, înaintea elementului de pe poziția n , dacă argumentul 2 lipsește;
- inserarea unei linii într-o matrice, înaintea liniei cu numărul n , dacă argumentul 2 lipsește;
- inserarea unei coloane într-o matrice, înaintea coloanei cu numărul n , dacă argumentul 2 apare.

Funcția returnează valoarea 1 dacă inserarea s-a efectuat cu succes.

Observație. Inserarea unui element, a unei linii sau coloane într-un masiv nu modifică dimensiunea masivului, ci duce la pierderea elementelor de pe poziția n .

Funcția ADEL(*nume_masiv*,*n*,[2])

Are ca efect operația inversă comenzii AINS, deci ștergerea elementului, liniei, coloanei de pe poziția n . Argumentul 2 trebuie să apară în cazul în care ștergem o coloană dintr-o matrice.

Prin cele două funcții, elementele unui masiv sunt translatate cu o poziție la dreapta (AINS) sau la stânga (ADEL). Funcția va returna valoarea 1 dacă ștergerea liniei/coloanei s-a efectuat cu succes.

Funcția ACOPY(*masiv_sursă*,*masiv_destinație*[,*nr_încep_sursă*[,*nr_elem_copiate*[,*nr_încep_destin*]]])

Este comanda de copiere a elementelor unui masiv (sursă) în elementele altui masiv (destinație), unde:

- *masiv_sursă*, *masiv_destinație* – numele masivelor sursă/destinație;
- *nr_încep_sursă* – numărul elementului din sursă de la care începe copierea;
- *nr_elem_copiate* – numărul de elemente ce vor fi copiate din masivul sursă, dacă are valoarea -1 toate elementele începând cu poziția *nr_încep_sursă* vor fi copiate;
- *nr_încep_destin* – numărul elementului din masivul destinație de la care începe inserarea.

Funcția returnează numărul elementelor copiate în masivul destinație.

Funcția ASORT(*nume_masiv*[,*nr_încep*[,*nr_elem_sortate*[,*nr_ordine_sortare*]]])

Comanda are ca efect sortarea elementelor unui masiv, putând preciza primul element de la care începe sortarea (*nr_încep*), câte elemente vor fi sortate (*nr_elem_sortate*). Ordinea de sortare (*nr_ordine_sortare*) va fi ascendentă dacă argumentul este 0 sau omis, sau descendentă dacă argumentul este 1 sau orice valoare diferită de 0.

Funcția va returna valoarea 1 dacă sortare s-a efectuat cu succes, sau -1 în caz contrar.

Exemplul 1. Secvența următoare de comenzi va crea o matrice cu 3 linii și 2 coloane, elementele masivului se identifică prin numărul de ordine în matricea liniarizată:

```
DIMENSION LIT(3,2)
```

```
LIT(1) = 'G'
```

```
LIT(2) = 'A'
```

```
LIT(3) = 'C'
```

LIT(4) = 'Z'

LIT(5) = 'B'

LIT(6) = 'N'

Elementele din matrice vor fi distribuite astfel:

Linia	Coloana 1	Coloana 2
1	G	A
2	C	Z
3	B	N

Se sortează masivul cu comanda ASORT(LIT,1)

Sortarea începe cu primul element al masivului (1,1). Elementele din prima coloană a masivului sunt plasate în ordine ascendentă prin rearanjarea liniilor masivului.

Linia	Coloana 1	Coloana 2
1	B	N
2	C	Z
3	G	A

Apoi masivul este sortat începând cu elementul 4 (2,2) din masiv cu comanda ASORT(LIT,4)

Elementele din coloana a 2-a sunt plasate în ordine ascendentă prin rearanjarea liniilor masivului, începând cu elementul 4 din masiv.

Linia	Coloana 1	Coloana 2
1	B	N
2	G	A
3	C	Z

Dacă sortarea se efectuează cu succes funcția returnează valoarea 1, în caz contrar valoarea -1.

Pentru a putea fi sortate, elementele masivului trebuie să fie de același tip.

Exemplul 2.

```
PUBLIC DIMENSION LIT(3,2)
CLEAR
LIT(1) = 'G'
LIT(2) = 'A'
LIT(3) = 'C'
LIT(4) = 'Z'
LIT(5) = 'B'
LIT(6) = 'N'
@1,1 SAY "MATRICEA INITIALA"
K=1
FOR I=1 TO 3
    @I+K,1 SAY LIT(I,1)
    @I+K,2 SAY ""
    @I+K,3 SAY LIT(I,2)
ENDFOR
@6,1 SAY "MATRICEA SORTATA INCEPAND CU PRIMUL ELEMENT"
K=6
ASORT(LIT,1)
FOR I=1 TO 3
    @I+K,1 SAY LIT(I,1)
    @I+K,2 SAY ""
    @I+K,3 SAY LIT(I,2)
ENDFOR
@11,1 SAY "MATRICEA SORTATA INCEPAND CU ELEMENTUL 4"
K=11
ASORT(LIT,4)
FOR I=1 TO 3
    @I+K,1 SAY LIT(I,1)
    @I+K,2 SAY ""
    @I+K,3 SAY LIT(I,2)
ENDFOR
@16,1 SAY "INSERAREA UNEI LINII INAINTE DE LINIA 3"
```

```

AINS(LIT,3)
LIT(3,1)="2"
LIT(3,2)="5"
K=16
FOR I=1 TO 3
    @I+K,1 SAY LIT(I,1)
    @I+K,2 SAY ""
    @I+K,3 SAY LIT(I,2)
ENDFOR
@21,1 SAY "STERGEREA LINIEI 3"
K=21
ADEL(LIT,3)
FOR I=1 TO 3
    @I+K,1 SAY LIT(I,1)
    @I+K,2 SAY ""
    @I+K,3 SAY LIT(I,2)
ENDFOR

```

În urma execuției programului de mai sus, se obțin rezultatele din Figura 1.

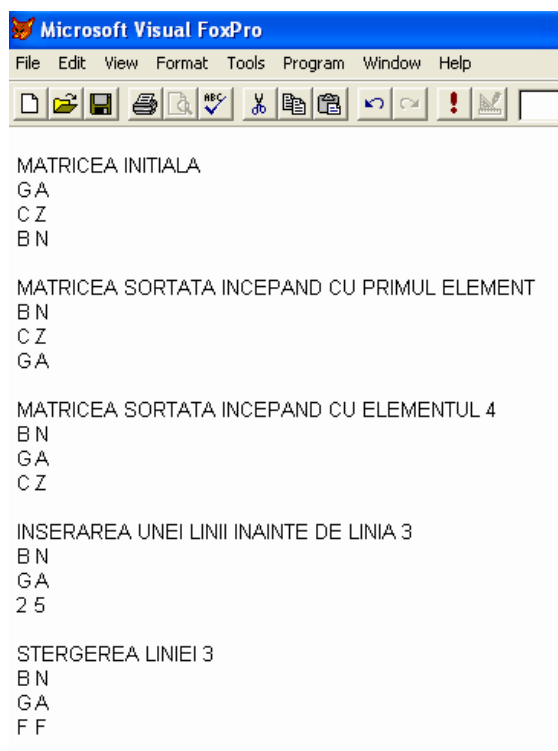


Fig. 1

8. Accesul și actualizarea datelor

Lucrul cu baze de date și tabele în cadrul limbajului propriu VFP, presupune trei activități, care segmentează construcția unei aplicații:

- deschiderea/crearea bazei de date, a tabelelor, a fișierelor asociate (index);
- exploatarea bazei de date și a tabelelor, ceea ce implică operații de actualizare și consultare, la nivel de structură sau înregistrări;
- închiderea/ștergerea bazei de date, a tabelelor sau a fișierelor asociate (index).

Comenzile pentru operațiile de deschidere, creare, închidere a unei baze de date (OPEN DATABASES..., CREATE DATABASES..., CLOSE DATABASES), a unei tabele de date (USE..., CREATE TABLE..., CLOSE TABLES, USE), a unor fișiere de index (USE...INDEX..., INDEX ON..., CLOSE INDEX) în modul de lucru comandă (interpretor), au aceeași sintaxă și în cadrul programelor scrise în limbajul propriu VFP.

8.1. Manipularea structurii unei tabele

Modificarea structurii unei tabele se face cu comanda

MODIFY STRUCTURE

Dacă nu este deschisă nici o tabelă în zona de lucru curentă, se deschide caseta de dialog **Open** care permite utilizatorului selectarea (deschiderea) unei tabele. Comanda are ca efect lansarea aplicației **Table Designer** (proiectantul de tabele) cu ajutorul căruia se fac modificările de structură (adăugare și ștergere de câmpuri, modificare nume câmp, mărime, tip, adăugarea / ștergerea / modificarea etichetelor de index, specificarea valorilor de tip **NULL**).

Vizualizarea structurii unei tabele se face cu comanda

```
DISPLAY STRUCTURE [IN <nr_zonă_de_lucru> | <alias_tabelă>] [TO PRINTER [PROMPT]
[TO FILE <nume_fișier>] [NOCONSOLE] IN <nr_zonă_de_lucru> | <alias_tabelă>
```

Are ca efect afișarea structurii tabelei dintr-o zonă de lucru, alta decât cea curentă, sau a unei tabele adresată prin alias. Semnificația clauzelor este următoarea:

- TO PRINTER [PROMPT] – direcționează informațiile la imprimantă.
- Clauza PROMPT determină afișarea unei casete de dialog înainte de ieșirea la imprimantă (se pot face setări legate de imprimantă).
- TO FILE <nume_fișier> - direcționează ieșirea informațiilor într-un fișier.
- NOCONSOLE – suprimă afișarea datelor în fereastra principală a VFP.

Se mai poate folosi și comanda LIST STRUCTURE care are aceleași clauze.

Copierea structurii unei table

Crearea unei noi table care va conține structura tablei active se face cu comanda

COPY STRUCTURE EXTENDED TO <nume_fișier> [FIELDS <listă_câmpuri>]

Numai câmpurile din lista de câmpuri vor fi incluse în înregistrarea din noul fișier. Dacă clauza FIELDS este omisă, vor fi incluse toate câmpurile tablei sursă.

8.2. Ordonarea datelor

Datele dintr-o tabelă se pot ordona după un singur criteriu sau după mai multe (ordonare multicriterială). Ordonarea poate fi ascendentă, dacă fiecare câmp după care se face ordonarea are o valoare mai mare decât câmpul corespunzător din înregistrarea anterioară, sau descendentă. Dacă ordonarea se face multicriterial, se definește o cheie primară, care este principalul criteriu de ordonare și chei secundare. În cazul în care două sau mai multe înregistrări au aceeași valoare pentru cheia principală, se utilizează cheile secundare pentru departajare, în ordinea în care au fost declarate. În VFP ordonarea unui fișier se poate face în două moduri:

- folosind comanda SORT, prin care se crează un nou fișier cu înregistrările sortate după criteriul ales.
- folosind metoda indexării, care constă în crearea unui fișier de index, în care se stochează ordinea înregistrărilor din tabela sursă.

Comanda SORT are sintaxa:

```
SORT TO <nume_tabela>ON <nume_câmp_1> [/A | /D] [/C] [, <nume_câmp_2> [/A | /D] [/C] ...]  
[ASCENDING | DESCENDING] [Scope] [FOR <expresie_1>] [WHILE <expresie_2>] [FIELDS  
    <lista_câmpurilor>]
```

unde:

- *nume_tabela* – numele fișierului sortat.
- *nume_câmp_1* – specifică câmpul din tabela nesortată asupra căreia se aplică procedura. Implicit sortarea este ascendentă. Nu se pot sorta câmpuri memo sau general. Este cheia primară după care se face sortarea.
- */A, /D, /C* – specifică ordinea: *A*: ascendentă, *D*: descendentă, *C*: fără a se ține cont de litere mari/mici.
- *nume_câmp_2* – specifică un câmp cu rol de cheie secundară.
- *ASCENDING* – specifică o sortare ascendentă pentru câmpurile care nu sunt urmate de opțiunea */D*.
- *DESCENDING* – specifică o sortare descendentă pentru câmpurile care nu sunt urmate de opțiunea */A*.
- *Scope* – specifică domeniul de sortare (tot fișierul sau doar o parte din înregistrări) **All**, **Next n**, **Record n**, **Rest**.
- *FOR expresie_1* – precizează care înregistrări participă la sortare, respectiv acelea pentru care *expresie_1* are valoarea **True**
- *WHILE expresie_2* – precizează condiția pentru care atâta vreme cât au valoarea **True**, înregistrările vor fi sortate.
- *FIELDS lista_câmpurilor* – precizează câmpurile din tabela originală care vor fi incluse în noul fișier sortat. Dacă se omite clauza, toate câmpurile din tabela originală vor fi incluse în tabela sortată.

Exemplu. Să se sorteze fișierul ESTUD1, într-un nou fișier SESTUD1, după câmpul NUME, ascedent, fără să se țină seama de litere mici/mari, luând doar studenții cu media peste 8.00 și câmpurile MATR, NUME, MBAC.

```
USE ESTUD1
SORT TO SESTUD1 ON NUME/A/C FOR MBAC>8.00 FIELDS MATR,NUME,MBAC
USE SESTUD1
BROWSE
CLOSE DATABASES
DELETE FILE SESTUD1.DBF
```

Sortarea este un procedeu care, în cazul tabelelor mari, duce la mărirea timpului de execuție și a spațiului solicitat pe hard disk. Se consideră că este mai eficientă folosirea metodei indexării.

Prin metoda indexării se crează un fișier de index, care va conține cel puțin un câmp al tabelii de date, sortat după un criteriu alfabetic, numeric sau cronologic. Fișierul de index este o sortare virtuală a unei tabele, înregistrările din tabela de date rămânând nesortate. Fișierul de index conține informații de localizare a înregistrărilor dintr-o tabelă.

Fișierele index sunt de 2 tipuri:

- *simple* – au extensia IDX, conțin o singură cheie de indexare și deci un singur criteriu de ordonare.
- *compuse* – au extensia CDX (Compound Index File), în care se stochează mai multe criterii de ordonare, fiecare având asociat un nume (TAG - eticheta de index), corespunzător mai multor chei de indexare.

Crearea unui fișier de index se face cu comanda:

```
INDEX ON <expr> TO <.IDX file>|TAG<tag_name>[OF<.CDX file>] [FOR <exprL>]
[ASCENDENT | DESCENDENT] [UNIQUE] [ADDITIVE]
```

unde:

- <expr> - expresia indexului care include câmpuri din tabela ce va fi indexată. Tipurile de expresii admise sunt: C, D, N.
- TO <.IDX file> - specifică numele fișierului de index care va fi creat.
- TAG<tag_name> OF<.CDX file> - permite crearea unui fișier cu mai multe intrări (etichete de index – TAG-uri), numărul lor fiind limitat doar de spațiul pe hard disk.

Fișierele cu extensia .CDX pot fi:

- *structurale* – se deschid automat odată cu deschiderea tabelii de date și au același nume.
- *nestructurale* – deschiderea se face indicând un nume la clauza OF<.CDX file>, diferit de cel al tabelii de date.

- FOR <exprL> - permite ca indexul creat să acționeze ca un filtru asupra tabelii.
- ASCENDENT|DESCENDENT – specifică ordinea de sortare în fișierul de index.
- UNIQUE – pentru valori repetate ale cheii, specifică includerea în fișierul de index numai la prima apariție.
- ADDITIVE – permite crearea unui fișier de index pentru o tabelă, chiar dacă mai sunt deschise alte fișiere de index.

Indexarea după mai multe câmpuri este similară sortării multiple. Nu se pot face indexări multiple pe câmpuri cu tipuri diferite, deci trebuie aduse la o formă comună (șir de caractere).

Comanda

```
INDEX ON <câmp_car> + STR(câmp_numeric,nc) + DTOS(câmp_dată) TO <fișier_index>
```

permite indexarea după 3 câmpuri diferite.

Semnificația parametrilor și a clauzelor:

- STR(câmp_numeric,nc) – transformă un număr într-un șir de caractere,
nc – numărul maxim de caractere pe care se reprezintă câmpul numeric.
- DTOS(câmp_data) – transformă o variabilă de tip dată în șir de caractere.

Pentru deschiderea simultană a tabelii de date și a fișierelor de index asociate se folosește comanda:

```
USE <nume_tabelă> INDEX <listă_fișiere_index>
```

Exemplul 1. Presupunem că este creată o tabelă (STUD) cu următoarea structură:

matricula	N(5)
nume	C(40)
data_n	D
adresa	C(40)

Să se indexeze după câmpul matricola. Tabela are următoarele înregistrări:

nr_înreg	matricola	nume	data_n	adresa
1	215	Aldea I. Dan	22/03/79	București
2	110	Barbu A. Vasile	10/05/80	Craiova
3	300	Alexandru F. Ion	13/07/80	Iași
4	200	Dinu A. Adrian	25/08/79	București
5	210	Cornea C. Ilie	16/07/80	Ploești
6	250	Ene V. Dan	24/03/80	București

Secvența de comenzi

CLEAR

USE STUD

BROWSE

INDEX ON MATRICOLA TO IMATR.IDX

* Obținerea numelui fișierului IDX deschis pentru baza de date curentă sau cea specificată

?? NDX(1)

* Afășează primul fișier index activ

BROWSE

* Înregistrările din fișierul de date apar sortate după câmpul matricola.

CLOSE DATABASES

CLEAR

crează fișierul index IMATR.IDX. Acesta are structura:

matricola	nr_înreg
110	2
200	4
210	5
215	1
250	6
300	3

Exemplul 2. Să se indexeze tabela STUD după câmpul NUME și să se creeze un fișier structural STUD.CDX.

```
USE STUD
INDEX ON UPPER(NUME) TAG NUME
BROWSE
CLOSE DATABASES
```

Exemplul 3. Să se indexeze tabela ESTUD1 după câmpul NUME, să se creeze un fișier nestructural INUM.CDX, ordonat ascendent după valorile cheii:

```
USE ESTUD1
INDEX ON UPPER(NUME) TAG NUME OF INUM.CDX ASCENDING
BROWSE
CLOSE DATABASES
```

Exemplul 4. Utilizând tabela ESTUD1 să se indexeze tabela după o cheie compusă (NUME, MBAC), fără a permite existența dublurilor, într-un fișier de index INM.IDX. Structura tablei este: MATR N(5), NUME C(40), ADRESA C(30), ANS N(1), ANIF N(4), DATAN N(4), MBAC N(5,2).

```
USE ESTUD1
INDEX ON UPPER(NUME)+STR(MBAC,5) TO INM.IDX UNIQUE
BROWSE
CLOSE DATABASES
```

Exemplul 5. Să se indexeze tabela ESTUD1 după câmpul numeric MATR, într-un fișier de index IMATR.IDX, numai pentru înregistrările care au câmpul MBAC (medie bacalaureat) >8.00:

```
USE ESTUD1
INDEX ON MATR TO IMATR.IDX FOR MBAC>8.0
BROWSE
CLOSE DATABASES
```

8.3. Accesul la date

În funcție de tipul accesului la date, comenzile sunt:

- pentru *acces secvențial* : GO, SKIP, LOCATE, CONTINUE.
- pentru *acces direct*: SEEK, caz în care tabela trebuie indexată.

Comanda GO

GO [RECORD] <nr_înreg> [IN <nr_zonă_de_lucru> | IN <alias_tabelă>]

sau

GO TOP | BOTTOM [IN <nr_zonă_de_lucru> | IN <alias_tabelă>]

Comanda are ca efect mutarea pointer-ului de pe înregistrarea curentă pe înregistrarea *nr_înreg*.

- IN *nr_zonă_de_lucru* – specifică zona de lucru în care este mutat pointer-ul.
- IN *alias_tabelă* – specifică alias-ul tabelii în care este mutat pointer-ul.
- TOP – poziționează pointer-ul pe prima înregistrare din fișier.
- BOTTOM – poziționează pointer-ul pe ultima înregistrare.

Funcția RECNO()

Returnează numărul curent al înregistrării pe care este poziționat pointer-ul în fișierul curent sau specificat. Apelul funcției

RECNO(<nr_zonă_de_lucru> | <alias_tabelă>)

unde: *nr_zonă_de_lucru* – reprezintă zona de lucru în care este activă tabela. Dacă s-a lansat o comandă SEEK pentru a căuta o înregistrare și aceasta nu a fost găsită, se poate utiliza comanda GO RECNO(0) pentru a returna numărul înregistrării cu cea mai apropiată valoare căutată. Funcția RECNO(0) va returna valoarea 0 dacă nu s-a găsit nici o valoare apropiată.

Funcția RECCOUNT()

Returnează numărul de înregistrări din tabela curentă sau specificată. Apelul funcției

RECCOUNT(<nr_zonă_de_lucru> | <alias_tabelă>)

Funcția EOF()

Este funcția logică care returnează valoarea adevărat (.T.), dacă pointer-ul de fișier este poziționat pe ultima înregistrare, în fișierul curent. În caz contrar, valoarea returnată este fals (.F.).

Apelul funcției

EOF ([<nr_zonă_de_lucru> | <alias_tabelă>])

Funcția BOF()

Este o funcția logică și returnează valoarea adevărat (.T.), dacă pointer-ul de fișier este poziționat pe prima înregistrare, în fișierul curent. În caz contrar, valoarea returnată este fals (.F.).

Apelul funcției

BOF ([<nr_zonă_de_lucru> | <alias_tabelă>])

Comanda SKIP

SKIP [<nr_înreg>] [IN <nr_zonă_de_lucru> | IN <alias_tabelă>]

unde: *nr_înreg* – specifică numărul de înregistrări peste care se mută pointer-ul în fișier. Dacă nu se specifică, pointer-ul se va muta la următoarea înregistrare. Dacă valoarea lui *nr_înreg* este pozitivă, pointer-ul se mută către sfârșitul fișierului. Dacă valoarea este negativă, pointer-ul se mută către începutul fișierului.

Comanda SET FILTER TO

Precizează condiția pe care trebuie să o îndeplinească înregistrările din tabela curentă pentru a putea fi accesate.

SET FILTER TO [<expr_L>]

unde: *expr_L* – condiția de accesare a înregistrărilor. Numai înregistrările care satisfac expresia logică *expr_L* sunt disponibile și toate comenzile care vor accesa tabela, vor respecta condiția declarată în comanda SET FILTER TO. Dacă este omisă expresia logică, toate înregistrările sunt accesibile.

Comanda LOCATE

Realizează căutarea secvențială într-o tabelă, a primei înregistrări care îndeplinește condițiile specificate în expresia logică de căutare, *expr_L*.

LOCATE FOR <expr_L_1> [Scope] [WHILE <expr_L_2>]

unde:

- FOR *expr_L_1* este condiția pentru care are loc căutarea secvențială.
- *Scope* – specifică domeniul înregistrărilor în care are loc căutarea. Numai înregistrările din interiorul domeniului vor fi evaluate. Clauza poate lua valorile **ALL**, **NEXT n**, **RECORD n**, **REST**. Clauza implicită este **ALL** (toate înregistrările).
- WHILE *expr_L_2* – specifică condiția pentru căutarea înregistrărilor, atâta vreme cât *expr_L_2* este adevărată.

Comanda CONTINUE

Se utilizează după o comandă LOCATE, pentru a continua căutarea, după găsirea primei înregistrări care îndeplinește condițiile specificate. Comanda CONTINUE mută pointer-ul de fișier la următoarea înregistrare pentru care expresia logică din comanda LOCATE este adevărată.

Exemplu. Presupunem că există o tabelă CLIENTI și trebuie să se afle numărul clienților care sunt din Franța.

```
USE CLIENTI
STORE 0 TO TOTAL
LOCATE FOR ALLTRIM(UPPER(TARA))='FRANTA'
DO WHILE FOUND()
    TOTAL=TOTAL+1
    CONTINUE
ENDDO
?'TOTAL CLIENTI DIN FRANTA: "+LTRIM(STR(TOTAL))
CLOSE DATABASES
```

Comanda SEEK

Caută într-o tabelă o înregistrare la prima apariție, a cărei cheie de index se potrivește cu expresia generală și mută pointer-ul pe înregistrarea găsită.

SEEK <expr_gen> [ORDER nr_index | nume_fişier_IDX | [TAG] nume_etichetă_index [OF nume_fişier_CDX] [ASCENDING | DESCENDING]] [IN <nr_zonă_de_lucru> | IN <alias_tabelă>]

unde:

- *expr_gen* – specifică cheia de index pentru care se caută cu comanda SEEK;
- ORDER *nr_index* – specifică numărul fişierului index care conţine cheia de indexare. Dacă s-au deschis odată cu tabela şi fişierele de index, este numărul care indică poziţia din lista de fişiere de index;
- ORDER *nume_fişier_IDX* - specifică numele fişierului de index;
- ORDER [TAG] *nume_etichetă_index* [OF *nume_fişier_CDX*] – specifică o etichetă de index dintr-un fişier .CDX care conţine cheia de indexare. Numele etichetei de index (TAG) poate fi dintr-un fişier .CDX structural sau dintr-un fişier .CDX independent (nestructural);
- [ASCENDING | DESCENDING] – specifică dacă înregistrările sunt căutate în ordine ascendentă sau descendentă.

Exemplu. Să se indexeze fişierul ESTUD2 după câmpul NUME şi să se caute înregistrările care încep cu literele BAR. Structura tabeli ESTUD2 este: MATR N(5), NUME C(40), ADRESA C(30), ANS N(1), DATAN D, MBAC N(5,2)

```
USE ESTUD2
INDEX ON UPPER(NUM) TAG NUME
STORE 'BAR' TO CAUTL
SEEK CAUTL
IF FOUND( )
    BROWSE FIELDS NUME, ADRESA FOR UPPER(NUM)=CAUTL
ELSE
    WAIT WINDOW 'NU EXISTA '
ENDIF
CLOSE DATABASES
```

Funcţia FOUND()

Returnează valoarea logică adevărat (.T.), dacă comenzile CONTINUE, LOCATE sau SEEK s-au executat cu succes.

FOUND(<zonă_de_lucru> | <alias_tabelă>)

Dacă argumentele *zonă_de_lucru*, *alias_tabelă* sunt omise, funcția va returna rezultatul pentru fișierul deschis în zona de lucru curentă.

Vizualizarea conținutului unei tabele

Comanda afișează informații (nume câmpuri, date) despre tabela activă în fereastra principală VFP sau într-o fereastră definită de utilizator. Sintaxa este

```
DISPLAY[[FIELDS] <listă_câmpuri>] [Scope] [FOR <expr_L_1>] [WHILE <expr_L_2>]
      [NOCONSOLE] [TO PRINTER [PROMPT] | TO FILE <nume_fișier>]
```

unde:

- [FIELDS] <listă_câmpuri> – specifică câmpurile care vor fi afișate. Dacă se omite clauza se vor afișa toate câmpurile.
- Scope – domeniul de adresare a înregistrărilor (**ALL**, **NEXT n**, **RECORD n**, **REST**);
- FOR <expr_L_1> - vor fi afișate doar înregistrările care satisfac condiția logică;
- WHILE <expr_L_2> - atâta vreme cât condiția este îndeplinită, înregistrările vor fi afișate.

Se mai poate utiliza și comanda LIST care are aceleași clauze.

8.4. Accesul concurențial

Este controlat în VFP cu comanda SET EXCLUSIVE, care specifică dacă VFP deschide tabelele pentru utilizare exclusivă sau partajată în rețea.

Sintaxa comenzii este:

```
SET EXCLUSIVE ON|OFF
```

Semnificația argumentelor comenzii SET EXCLUSIVE :

- ON - este valoarea implicită pentru datele din sesiunea globală. Are ca efect limitarea accesibilității la tabela deschisă prin rețea pentru un utilizator, fiind neaccesibilă și pentru ceilalți utilizatori din rețea. Comanda SET EXCLUSIVE ON determină pentru toți utilizatorii un acces de tip *read-only* (numai pentru citire). Un fișier poate fi deschis prin rețea pentru utilizare exclusivă prin includerea clauzei EXCLUSIVE în comanda USE. Tabele deschise cu clauza EXCLUSIVE nu mai necesită utilizarea ulterioară a comenzilor pentru blocare tabelă (FLOCK() – *File Lock*) sau înregistrare (RLOCK() – *Record Lock*). Deschiderea tabelii în mod exclusiv previne modificările pe care le-ar putea face în tabelă, ceilalți utilizatori din rețea.

- OFF - este valoarea implicită pentru o sesiune de lucru privată. Permite ca o tabelă să fie deschisă în rețea în mod partajat și poate fi modificată de către orice utilizator autorizat al rețelei.

Crearea unei aplicații care să ruleze pe diferite calculatoare legate în rețea, sau dacă câteva instanțe ale unui videoformat accesează aceleași date, presupune realizarea unui program pentru accesul partajat.

Accesul partajat furnizează mijloace eficiente pentru utilizarea comună a datelor printre utilizatori dar și restricții de acces atunci când este cazul.

Visual FoxPro furnizează suport pentru partajare sau acces exclusiv al datelor, opțiuni de blocare, sesiuni de date, sisteme de buffering (utilizarea memoriei tampon) pentru înregistrări, tabele și sesiuni de tranzacții. Toate aceste caracteristici sunt folosite într-un mediu partajabil cum este rețeaua de calculatoare, dar pot fi folosite și în cazul mediilor cu un singur utilizator.

8.5. Actualizarea datelor

Operația de actualizare constă în adăugarea de noi înregistrări, modificarea înregistrărilor existente, ștergerea logică și/sau fizică a înregistrărilor. Operațiile fac parte din limbajul de manipulare a datelor (LMD).

8.5.1. Adăugarea de noi înregistrări

Adăugarea se poate face utilizând comenzile de tip APPEND, prin preluarea datelor din alte tabele, masive de date și variabile de memorie.

```
APPEND [BLANK] [IN nr_zonă_de_lucru | alias_tabelă]
```

Comanda are ca efect adăugarea unei înregistrări vide la sfârșitul unei tabele.

```
APPEND FROM <nume_fișier> [FIELDS <listă_câmpuri>] [FOR <expr_L>] [TYPE]  
[DELIMITED [WITH <delimiter> | WITH BLANK | WITH TAB]]
```

Comanda are ca efect adăugarea unei înregistrări preluată din altă tabelă, sau dintr-un fișier de tip text, în tabela curentă, unde:

- *nume_fișier* – specifică numele fișierului din care se face adăugarea;

- **FIELDS** <listă_câmpuri> - specifică lista câmpurilor în care vor fi adăugate datele;
- **FOR** <expr_L> - se va adăuga câte o nouă înregistrare, din înregistrările tabelii selectate pentru adăugare, cele care îndeplinesc condiția dată de expresia logică *expr_L*;
- **TYPE** – specifică tipul fișierului sursă din care se adaugă. Se folosește în cazul în care nu este o tabelă VFP;
- **DELIMITED** – fișierul sursă din care se adaugă datele în tabela VFP este un fișier de tip ASCII în care fiecare înregistrare se termină cu <CR> și <LF> (**Enter** și **Line Feed** - salt la linie nouă). Câmpurile conținute sunt implicit separate prin virgulă, iar cele de tip caracter sunt declarate între apostrof;
- **DELIMITED WITH** <delimiter> - câmpurile de tip caracter, sunt separate cu un caracter diferit de apostrof;
- **DELIMITED WITH BLANK** - câmpurile sunt separate cu spații în loc de virgulă;
- **DELIMITED WITH TAB** - câmpurile sunt separate cu cu TAB-uri în loc de virgulă.

APPEND FROM ARRAY <nume_masiv> [FOR <expr_logică>] [FIELDS <listă_câmpuri> |
FIELDS LIKE <șablon> | FIELDS EXCEPT <șablon>]

Comanda are ca efect adăugarea unei înregistrări în tabela curentă, pentru fiecare linie de masiv, datele scrise în fiecare câmp corespund coloanelor din masiv. Semnificația parametrilor și a clauzelor este următoarea:

- *nume_masiv* – specifică numele masivului care conține datele ce vor fi copiate ca înregistrări noi;
- **FOR** <expr_logică> - specifică condiția pentru adăugarea înregistrărilor în tabelă. Expresia logică trebuie să conțină și o condiție pentru un câmp al înregistrării. Înainte de a fi adăugată o linie din masiv în tabelă sub formă de înregistrare, în expresia logică se verifică dacă elementele liniei de masiv respectă condiția;
- **FIELDS** <listă_câmpuri>- specifică faptul că numai anumite câmpuri (cele din listă) din noua înregistrare vor fi actualizate cu date din linia masivului;
- **FIELDS LIKE** <șablon> - specifică faptul că doar câmpurile care se potrivesc șablonului vor fi actualizate.
- **FIELDS EXCEPT** <șablon>- specifică faptul că toate câmpurile vor fi actualizate, cu excepția celor care se încadrează în șablon.

Dacă masivul este unidimensional, comanda APPEND FROM ARRAY adaugă o singură înregistrare în tabelă, conținutul primului element trece în primul câmp al înregistrării, conținutul celui de al doilea element trece în cel de al doilea câmp etc., câmpurile MEMO și GENERAL sunt ignorate.

Exemplu comanda:

```
APPEND FROM ARRAY NUME FIELDS LIKE M*, A* EXCEPT NUM*
```

are ca efect adăugarea din masivul NUME, a câmpurilor care încep cu litera M și A mai puțin cele care au primele trei litere NUM. Caracterul asterisc (**Wildcard**) are semnificația de orice combinație de caractere.

Notă Utilizarea clauzelor FIELDS LIKE și EXCEPT trebuie să se facă într-o tabelă ale cărei înregistrări au câmpuri care acceptă valori de tip **NULL** (pentru a putea fi exceptate de la scriere).

Există și comanda inversă, cu care putem adăuga datele din înregistrarea curentă a unei tablele într-un masiv:

```
SCATTER [FIELDS <listă_câmpuri>] | [FIELDS LIKE <șablon>] | [FIELDS EXCEPT <șablon>]  
[MEMO] TO ARRAY <nume_masiv> [BLANK]
```

Semnificația clauzelor este aceeași ca și la comanda APPEND FROM ARRAY ..., clauza BLANK determină crearea masivului cu elemente vide, care sunt de aceeași mărime și tip cu câmpurile din tabelă. Clauza MEMO specifică existența unui câmp de tip MEMO în listă; implicit câmpurile MEMO sunt ignorate.

Exemplu. Să se creeze un masiv cu numele MASIV1, să se încarce cu date vide, de același tip și mărime cu cele ale câmpurilor tablei ESTUD1 (MATR N(5), NUME C(40), ADRESA C(30), ANS N(1), DATAN D, MBAC N(5,2)). Să se introducă datele câmpurilor în masiv și să se adauge sub formă de înregistrare nouă în tabela ESTUD1.

```
USE ESTUD1
```

* se deschide tabela ESTUD1

SCATTER TO MASIV1 BLANK

* se crează vectorul MASIV1, cu elemente vide și de același tip și mărime

* cu câmpurile din tabela ESTUD1

FOR I=1 TO ALEN(MASIV1)

@I,1 SAY FIELD(I) GET MASIV1(I)

* se afișează numele câmpului și se editează elementul I din masiv

ENDFOR

READ

* se citesc valorile introduse cu GET

IF READKEY() != 12

* dacă nu s-a apăsat tasta <ESC>

APPEND FROM ARRAY MASIV1

* se adaugă din elementele masivului o nouă înregistrare în tabela

ENDIF

BROWSE

* vizualizare tabela ESTUD1

CLOSE TABLE

* închidere tabela ESTUD1

CLEAR

APPEND GENERAL <nume_câmp_general> FROM <nume_fișier> DATA <expr_c> [LINK]
[CLASS <nume_clasă>]

Comanda realizează importul unui obiect OLE dintr-un fișier și îl plasează într-un câmp de tip general. Semnificația parametrilor și a clauzelor este următoarea:

- *nume_câmp_general* – specifică numele câmpului general în care obiectul OLE va fi plasat.
- FROM <nume_fișier> - specifică numele fișierului care conține obiectul OLE;
- DATA <expr_c> - expresie de tip caracter care este evaluată și trecută sub formă de șir de caractere către obiectul OLE din câmpul general. Obiectul OLE trebuie să fie capabil să primească și să proceseze șirul de caractere. De exemplu, nu poate fi trimis un șir de caractere către un obiect grafic creat cu Paint Brush;

- **LINK** - crează o legătură între obiectul OLE și fișierul care conține obiectul. Obiectul OLE apare în câmpul general, dar definirea obiectului rămâne în fișierul care-l conține. Dacă se omite clauza, obiectul este înglobat în câmpul general;
- **CLASS** <nume_clasă> - specifică o clasă pentru un obiect de tip OLE, alta decât clasa implicită.

8.5.2. Modificarea înregistrărilor

Comanda **CHANGE** afișează câmpurile unei tabele pentru editare (modificare).

CHANGE [FIELDS <listă_câmpuri>][Scope] [FOR <expresie_L_1>] [WHILE <expresie_L_2>]
[FONT <nume_literă> [,<mărime_literă>]] [FREEZE <nume_câmp>] [NOAPPEND]
[NODELETE] [NOEDIT]

Semnificația clauzelor este următoarea:

- **FIELDS** <listă_câmpuri> - specifică câmpurile care vor apare în fereastra de editare;
- **Scope** – specifică domeniul de afișare a înregistrărilor (**ALL**, **NEXT** *n*, **RECORD** *n*, **REST**);
- **FOR** <expresie_L_1> - specifică faptul că doar înregistrările care satisfac condiția logică dată de *expresie_L_1*, vor fi afișate în fereastra de editare;
- **WHILE** <expresie_L_2> - atâta vreme cât *expresie_L_2* este adevărată vor fi afișate înregistrările în fereastra de editare pentru modificare;
- **FONT** <nume_literă> [,<mărime_literă>] – specifică pentru fereastra de editare, tipul de literă și mărimea, cu care vor fi afișate datele;
- **FREEZE** <nume_câmp> - permite ca modificările să fie făcute doar în câmpul specificat cu *nume_câmp*. Celelalte câmpuri vor fi afișate dar nu pot fi editate (modificate);
- **NOAPPEND** – împiedică utilizatorul de a adăuga noi înregistrări (se blochează **Append Mode** din meniul sistem **View** sau combinația de taste **[Ctrl] + [Y]**);
- **NODELETE** – împiedică marcarea înregistrărilor pentru ștergere din interiorul ferestrei de editare. Includerea clauzei nu inhibă comanda de marcarea pentru ștergere din interiorul unei proceduri;
- **NOEDIT** – împiedică un utilizator să modifice o tabelă. Includerea clauzei permite căutarea sau răsfoirea tabelii dar fără a-l putea modifica (edita).

Se poate folosi și comanda **EDIT** cu aceiași parametri ca și **CHANGE**.

Comanda **BROWSE** este una dintre cele mai utilizate comenzi pentru afișarea și editarea înregistrărilor dintr-o tabelă. Cu ajutorul tastelor **[TAB]** sau **[SHIFT] + [TAB]** se poate face deplasarea în câmpul următor sau anterior. Cu tastele **[Page Up]**, **[Page Down]** se pot face deplasări pe verticală în tabelă.

Apăsarea simultană a tastelor: **[Ctrl] + [End]** sau **[Ctrl] + [W]** - ieșire cu salvare din fereastra de **BROWSE**; **[Ctrl] + [Q]** sau **[Esc]** ieșire fără salvarea modificărilor.

```
BROWSE [FIELDS <listă_câmpuri>] [FONT <nume_font>[,<mărime_font>]]
[STYLE <stil_font>] [FOR <expr_L_1>] [FREEZE <nume_câmp>] [LOCK <nr_câmpuri>]
[NOAPPEND] [NODELETE] [NOEDIT | NOMODIFY] [VALID <expr_L_2>] [ERROR <mesaj>]
[WHEN <expr_L_3>] [WIDTH <lățime_afișare_câmp>] [IN [WINDOW] <nume_fer>]
```

unde:

- **FIELDS <listă_câmpuri>** - specifică care câmpuri vor apărea în fereastra **BROWSE**. Câmpurile vor fi afișate în ordinea specificată în listă;
- **FONT <nume_font>[,<mărime_font>]** – specifică tipul literei și mărimea utilizate în fereastra **Browse** (de exemplu **Font** ‘Courier’, 16);
- **STYLE <stil_font>** - specifică stilul literelor folosite în fereastra **Browse** (**B**- îngroșat, **I**- italic, **U**- subliniat);
- **FOR <expr_L_1>** - vor fi afișate numai acele înregistrări care îndeplinesc condiția dată de expresia logică *expr_L_1*;
- **FREEZE <nume_câmp>** - permite modificări doar într-un singur câmp, specificat prin *nume_câmp*;
- **LOCK <nr_câmpuri>** - specifică numărul câmpurilor care pot fi afișate în partea stângă, în fereastra **Browse**, fără a fi necesară deplasarea cu **[TAB]**-uri sau cursorul ferestrei (**ScrollBar**);
- **NOAPPEND** – împiedică utilizatorul să adauge noi înregistrări (se inhibă acțiunea tastelor **[Ctrl] + [Y]** și opțiunea **Append new record** din meniul **Table**);
- **NODELETE** – împiedică marcarea înregistrărilor pentru ștergere în interiorul ferestrei **BROWSE**;
- **NOEDIT | NOMODIFY** – împiedică utilizatorul să facă modificări în fereastra **Browse**.
- **VALID <expr_L_2> [ERROR <mesaj>]** – realizează o validare la nivel de înregistrare. Clauza se execută doar dacă se fac modificări și se încearcă trecerea la o altă înregistrare.

Dacă *expr_L_2* returnează valoarea **.T.** (adevărat) utilizatorul se poate deplasa la următoarea înregistrare. În caz contrar, se generează un mesaj de eroare;

- **WHEN** *<expr_L_3>* - se evaluează condiția dată de expresia logică *expr_L_3* atunci când utilizatorul mută cursorul la o altă înregistrare. Dacă expresia returnează valoarea **.F.** (fals) înregistrarea nu mai poate fi modificată (devine **read-only**);
- **WIDTH** *<lățime_afișare_câmp>* - limitează numărul de caractere afișate pentru toate câmpurile din fereastră;
- **IN** [**WINDOW**] *<nume_fer>* - specifică fereastra părinte în interiorul căreia va fi deschisă fereastra **Browse**. Dacă fereastra părinte este mutată, se va muta și fereastra **Browse**.

Comanda **REPLACE** realizează actualizarea câmpurilor dintr-o înregistrare.

REPLACE *<nume_câmp_1>* **WITH** *<expr_1>* [**ADDITIVE**] [,*<nume_câmp_2>* **WITH** *<expr_2>* [**ADDITIVE**]] ... [*Scope*] [**FOR** *<expr_L_1>*] [**WHILE** *<expr_L_2>*] [**IN** *nr_zonă_de_lucru* | *alias_tabelă*]

unde:

- *<nume_câmp_1>* **WITH** *<expr_1>* [,*<nume_câmp_2>* **WITH** *<expr_2>*] precizează că datele din *nume_câmp_1* vor fi înlocuite cu valorile expresiei *expr_1*, datele din *nume_câmp_2* cu valorile expresiei *expr_2* etc.
- [**ADDITIVE**] – pentru câmpurile de tip memo, adaugă valoarea expresiei la sfârșitul câmpului memo (în continuare). Dacă clauza lipsește, atunci câmpul memo va fi rescris cu valoarea dată de expresie;
- *Scope* – specifică domeniul înregistrărilor care vor fi înlocuite (**NEXT** *n*, **ALL**, **REST**, **RECORD** *n*);
- **FOR** *<expr_L_1>* - specifică faptul că vor fi înlocuite câmpurile desemnate, doar din înregistrările pentru care evaluarea expresiei logice *expr_L_1* are valoarea **.T.** (adevărat);
- **WHILE** *<expr_L_2>* - precizează condiția pentru care atâta vreme cât expresia logică are valoarea **.T.**, câmpurile desemnate vor fi modificate;
- **IN** *nr_zonă_de_lucru* | *alias_tabelă* – specifică zona de lucru sau alias-ul tabelii, în care înregistrările vor fi actualizate.

Exemplu. În fișierul ESTUD2, să se modifice a treia înregistrare, câmpul ADRESA, cu o nouă valoare “Cluj”.

USE ESTUD2

REPLACE ADRESA WITH "Cluj" FOR RECNO()=3

BROWSE

CLOSE DATABASE

Comanda SET SAFETY ON | OFF determină afișarea de către sistemul VFP a unei casete de dialog înainte de a suprascrie un fișier.

SET SAFETY ON | OFF

- Clauza ON este implicită și determină afișarea unei casete de dialog, pentru confirmarea de către utilizator a suprascrierii fișierului.
- Clauza OFF împiedică afișarea casetei de confirmare a suprascrierii.

8.5.3. Ștergerea înregistrărilor

Ștergerea înregistrărilor se poate face la nivel logic sau fizic.

Comanda DELETE realizează ștergerea la nivel logic. Înregistrările continuă să existe în fișier dar sunt marcate ca fiind șterse.

DELETE [*Scope*] [FOR <*expr_L_1*>] [WHILE <*expr_L_2*>] [IN <*nr_zonă_de_lucru*> |
<*alias_tabelă*>]

unde:

- *Scope* – specifică domeniul în care se face marcarea pentru ștergere a înregistrărilor;
- FOR <*expr_L_1*> - specifică condiția pentru care vor fi marcate pentru ștergere acele înregistrări pentru care *expr_L_1* este adevărată;
- WHILE <*expr_L_2*> - vor fi marcate pentru ștergere înregistrări, atâta vreme cât condiția *expr_L_2* este adevărată.

Funcția DELETED() returnează o valoare logică care indică dacă înregistrarea curentă a fost marcată pentru ștergere.

DELETED ([*nr_zonă_de_lucru* | *alias_tabelă*])

Comanda RECALL deselectează înregistrările marcate pentru ștergere.

RECALL [*Scope*] [FOR <*expr_L_1*>] [WHILE <*expr_L_2*>]

unde:

- *Scope* – precizează domeniul în care se aplică deselectarea;
- FOR <*expr_L_1*> - doar înregistrările pentru care *expr_L_1* este adevărată (.T.) vor fi deselectate;
- WHILE <*expr_L_2*> - atâta vreme cât *expr_L_2* este adevărată, înregistrările vor fi deselectate.

Comanda PACK face parte din comenzile de ștergere la nivel fizic; va șterge toate înregistrările marcate pentru ștergere din tabela curentă.

PACK [MEMO] [DBF]

unde:

- MEMO – permite ștergerea fișierului MEMO atașat tabelii curente, dar nu și înregistrările marcate pentru ștergere (fișierul MEMO are același nume cu tabela, dar extensia este .FPT);
- DBF – se vor șterge înregistrările marcate pentru ștergere din tabela curentă, dar nu afectează fișierul MEMO.

Comanda ZAP șterge fizic toate înregistrările din tabelă, rămânând doar structura tabelii. Este echivalentă cu comanda DELETE ALL, urmată de comanda PACK, dar este mult mai rapidă.

ZAP [*nr_zonă_de_lucru* | *alias_tabelă*]

Comanda DELETE FILE șterge un fișier de pe disc.

DELETE FILE [<*nume_fișier*>] [RECYCLE]

unde:

- *nume_fișier* – specifică numele fișierului și extensia. Numele fișierului poate conține și caracterul asterisc: DELETE FILE *.BAK va avea ca rezultat ștergerea tuturor fișierelor cu extensia .BAK de pe disc. Comanda se execută pentru fișierele care nu sunt deschise (active);

- RECYCLE – fișierul nu va fi șters imediat de pe disc ci mutat în directorul RECYCLE BIN din sistemul WINDOWS 9x. Se mai poate folosi și comanda ERASE care are aceleași clauze.

Comanda SET DELETED specifică dacă sistemul VFP va procesa (șterge) înregistrările marcate pentru ștergere sau dacă acestea sunt disponibile pentru a fi utilizate în alte comenzi.

SET DELETED ON | OFF

Semnificația clauzelor:

- Clauza OFF este implicită. Înregistrările marcate pentru ștergere pot fi accesate de alte comenzi care operează cu înregistrări;
- Clauza ON are ca efect ignorarea înregistrărilor marcate pentru ștergere, de comenzile care operează cu înregistrări.

Comanda COPY TO crează un nou fișier cu același conținut ca și cel al tabelii curente.

COPY TO <nume_fișier> [FIELDS <listă_câmpuri> | FIELDS LIKE <șablon> | FIELDS EXCEPT <șablon>][[Scope] [FOR <exprL_1>] [WHILE <exprL_2>] [[WITH] CDX] | [[WITH] PRODUCTION]

unde:

- *nume_fișier* – specifică numele noului fișier care se crează cu comanda COPY TO;
- FIELDS <listă_câmpuri> - specifică în listă câmpurile ce vor fi copiate în noul fișier. Dacă clauza se omite, atunci toate câmpurile din tabela activă vor fi copiate. Prin declararea clauzei FIELDS se realizează operația de proiecție (se selectează numai anumite câmpuri - coloane - din tabelă);
- FIELDS LIKE <șablon> - specifică câmpurile din tabela sursă care se potrivesc unui șablon și vor fi copiate în noul fișier. Se poate folosi și caracterul asterisc (de exemplu comanda COPY TO TABELA1 FIELDS LIKE P*, A* va avea ca efect copierea în noul fișier a tuturor câmpurilor care încep cu litera P și A);
- FIELDS EXCEPT <șablon> - vor fi copiate toate câmpurile, mai puțin cele care se încadrează în șablonul specificat;
- *Scope* – specifică domeniul din care vor fi selectate înregistrările care vor fi copiate.

- FOR <exprL_1> - vor fi copiate doar acele înregistrări pentru care evaluarea expresiei logice exprL_1 are valoarea adevărat (.T.)
- WHILE <exprL_2> - specifică condiția pentru care atâta vreme cât expresia logică exprL_2 este adevărată, înregistrările vor fi copiate în noul fișier.

Notă Dacă se utilizează clauzele [Scope], FOR sau WHILE, fără clauza FIELDS, se va implementa operatorul relațional de selecție (se aleg doar o parte din înregistrări, cu toate câmpurile). Dacă se folosește clauza FIELDS în combinație cu clauzele [Scope], FOR sau WHILE, va rezulta o operație compusă: proiecție și selecție.

- [WITH] CDX] | [[WITH PRODUCTION] – la copiere se va crea și un fișier de index structural pentru noua tabelă, care este identic cu cel al tabelii sursă. Etichetele de index (TAG-urile) vor fi copiate în noul fișier de index structural. Cele două clauze sunt identice. Clauzele nu se folosesc dacă se crează un fișier care nu este tabelă VFP (extensia .DBF).

Comanda SET ALTERNATE direcționează la ecran sau la imprimantă, ieșirea rezultată din folosirea comenzilor DISPLAY sau LIST.

SET ALTERNATE ON | OFF

sau

SET ALTERNATE TO [<nume_fișier> [ADDITIVE]]

unde:

- ON – direcționează ieșirea către un fișier de tip text;
- OFF – clauza implicită, dezactivează ieșirea către un fișier de tip text;
- TO *nume_fișier* – crează un fișier de tip text, cu extensia implicită .TXT ;
- ADDITIVE – clauză prin care ieșirea este adăugată la sfârșitul fișierului specificat cu *nume_fișier*. Dacă se omite clauza, conținutul fișierului de ieșire este suprascris.

Comanda CLOSE ALTERNATE închide un fișier deschis cu comanda SET ALTERNATE.

Sintaxa comenzii este

CLOSE ALTERNATE

8.5.5. Relații între tabele

Comanda SET RELATION TO stabilește o legătură între două tabele deschise.

```
SET RELATION TO [<expresie1> INTO <nr_zonă_de_lucru> | <alias_tabelă> [, <expresie2>  
INTO <nr_zonă_de_lucru> | <alias_tabelă>...] [IN <nr_zonă_de_lucru> | <alias_tabelă>]  
[ADDITIVE]]
```

unde:

- *expresie1* – specifică expresia relațională care stabilește legătura între tabela-părinte și tabela-fiu. Expresia relațională este de obicei câmpul indexat din tabela-fiu. Indexul pentru tabela-fiu poate proveni dintr-un fișier simplu de index (.IDX) sau un fișier de index compus (.CDX);
- INTO *nr_zonă_de_lucru* | *alias_tabelă* – specifică numărul zonei de lucru sau alias-ul tabelii-fiu;
- IN *nr_zonă_de_lucru* | *alias_tabelă* – specifică zona de lucru sau alias-ul tabelii-părinte;
- ADDITIVE – clauza păstrează toate celelalte relații stabilite anterior în zona de lucru specificată.

Pentru a stabili o relație între două tabele trebuie îndeplinite următoarele condiții:

- cele două tabele între care se stabilește relația tata-fiu trebuie să fie indexate după aceeași cheie (câmp), care face legătura între ele;
- cele două tabele trebuie deschise în zone de lucru diferite.

Exemplul 1.

Tabelele legate printr-o relație au în general un câmp comun. De exemplu presupunem că avem o tabelă STUDENT, care conține informații despre studenți și are câmpuri pentru *nume*, *adresă*, și un cod unic al studentului (*matr*). O a doua tabelă, NOTE, conține informații despre notele studentului (*matr*, *cod_disc*, *nota*), care va avea de asemenea un câmp pentru codul unic al studentului (*matr*). Se pot uni astfel informațiile despre studenți cu cele despre note. Comanda SET RELATION leagă cele două tabele prin câmpul comun *matr*. Pentru a stabili relația, tabela-fiu (NOTE) trebuie să fie indexat după câmpul comun. După stabilirea relației, de câte ori mutăm pointer-ul pe o înregistrare în fișierul STUDENT, pointer-ul din fișierul NOTE (fiu) se va muta pe înregistrarea care are același cod *matr* corespunzător în fișierul STUDENT.

Fișier discipline.dbf		Fișier student.dbf		Fișier note.dbf	
Cod_disc	N(3)	Matr	N(5)	Matr	N(5)
Denumire	C(60)	Nume	C(50)	Cod_disc	N(3)
Exam	L	Adresa	m	Nota	N(2)
Coloc	L	An_stud	C(2)		
Tip_curs	C(1)	Grupa	N(2)		
		Media	N(5,2)		
		Integr	L		

Programul pentru calculul mediilor generale

PUBLIC MEDG, INTEGRAL

SELECT 2

USE STUDENT

SELECT 3

USE NOTE

FOR I=1 TO RECCOUNT(2)

 GOTO I

 MATRICOLA=MATR

 INTEGRAL=.T.

 MEDG=0

 K=0

 SELECT 3

 LOCATE FOR MATR=MATRICOLA

 DO WHILE FOUND(3)

 K=K+1

 MEDG=MEDG+NOTA

 IF NOTA<5 THEN

 INTEGRAL=.F.

 ENDIF

 CONTINUE

ENDDO

MEDG=MEDG/K

SELECT 2

REPLACE MEDIA WITH MEDG, INTEGR WITH INTEGRAL

```

ENDFOR
SELECT 2
SORT TO TEMP1 ON NUME/A
CLOSE DATABASE
DELETE FILE 'STUDENT.DBF'
RENAME 'TEMP1.DBF' TO 'STUDENT.DBF'
SELECT 2
USE STUDENT
REPORT FORM MEDIIG PREVIEW
CLOSE DATABASES

```

The screenshot shows the 'Report Designer - mediig.frx' window. The report layout is as follows:

- Title:** A section with the text 'MEDII GENERALE' in green.
- Page Header:** A section containing three fields: 'Matricola', 'Nume si prenume', and 'Media'.
- Detail:** A section containing three input fields: 'MATR_', 'NUME_', and 'MEDIA_'.
- Page Footer:** A section containing the text '"Page " + ALLTRIM(STR(PA'.

Programul pentru stabilirea relațiilor

```

SELECT 2
USE STUDENT
SELECT 3
USE DISCIPLINE
SELECT 4
USE NOTE
SELECT 2
INDEX ON MATR TAG MATR ADDITIVE
&& se indexează după câmpul matr
SELECT 3
INDEX ON COD_DISC TAG COD_DISC
&& se indexează după câmpul cod_disc

```

```
SELECT 4
SET ORDER TO TAG MATR OF STUDENT.CDX IN STUDENT
&& se stabilește indexul principal
SET RELATION TO MATR INTO STUDENT ADDITIVE
&& SE stabilește relația între student și note
SET ORDER TO TAG COD_DISC OF DISCIPLINE.CDX IN DISCIPLINE
&& se stabilește indexul principal
SET RELATION TO COD_DISC INTO DISCIPLINE ADDITIVE
&& se stabilește relația între discipline, note
REPORT FORM REPORT1 PREVIEW
SET RELATION OFF INTO STUDENT
SET RELATION OFF INTO DISCIPLINE
CLOSE DATABASE
```

Pentru crearea raportului REPORT1.FRX se deschid tabelele: STUDENT, NOTE, DISCIPLINE, se accesează meniul **File/New** opțiunea **Report**.

Pagina de raport este structurată în trei secțiuni: **Page Header** (antet), **Detail** (conținut), **Page Footer** (subsol pagină). Datele pot fi grupate după una sau mai multe variabile, introducându-se în **GroupHeader**; **Detail** sau **Group Footer** (Figura 1).

În raport se includ următoarele câmpuri de date (Figura 1):

- *recno()* – numărul înregistrării din tabelă,
- *student.nume*,
- *student.media*,
- *note.nota*,
- *discipline.denumire*.

Raportul poate conține și elemente grafice: linii / chenare utilizate pentru construcția de tabele sau imagini.

CATALOG STUDENTI

▲ Page Header	
REC	nume_____ MEDIA MEDIA__
NOTA	DISCIPLINA
▲ Group Header 1:nume	
nota__	denumire_____
▲ Detail	
▲ Group Footer 1:nume	
▲ Page Footer	

Fig. 1

Exemplul 2.

Presupunem că avem o tabelă **CLIENTI**, care conține informații despre clienți, câmpuri pentru *nume*, *adresă*, și un cod unic al clientului (*cod_client*). O a doua tabelă, **FACTURI**, conține informații despre facturi (*număr*, *cantitate*, *produse* etc.), care va avea de asemenea un câmp pentru codul clientului. Se pot uni astfel informațiile despre clienți cu cele despre facturi. Comanda **SET RELATION** leagă cele două tabele prin câmpul comun *cod_client*. Pentru a stabili relația, tabela-fiu (**FACTURI**) trebuie să fie indexat după câmpul comun. După stabilirea relației, de câte ori se mută pointer-ul în fișierul **CLIENTI** pe o înregistrare, pointer-ul din fișierul **FACTURI** (fiu) se va muta pe înregistrarea care are același *cod_client* în fișierul **CLIENTI**.

```
SELECT 2
```

```
USE CLIENTI
```

```
SELECT 3
```

```
USE FACTURI
```

```
INDEX ON COD_CLIENT TAG COD_CLIENT ADDITIVE
```

&& se indexează câmpul *cod_client*

```
SELECT 2
```

```
SET ORDER TO TAG COD_CLIENT OF FACTURI.CDX IN FACTURI
```

&& se stabilește indexul principal

```
SET RELATION TO COD_CLIENT INTO FACTURI ADDITIVE
```

&& se stabilește relația între clienți, facturi

Comanda RENAME redenumeste un fișier.

RENAME <nume_fișier_vechi> TO <nume_fișier_nou>

unde:

- *nume_fișier_vechi* TO *nume_fișier_nou2* – specifică numele fișierului sursă și noul nume. Se va include extensia pentru fiecare fișier, dacă nu, implicit se va atribui extensia .DBF. Comanda RENAME nu se folosește pentru o tabelă care aparține unei baze de date. Pentru a schimba numele unei tabele dintr-o bază de date se folosește comanda RENAME TABLE. Numele fișierului sursă și cel al fișierului nou creat poate cuprinde și caracterul asterisc (RENAME *.PRG TO *.BAK).

Comanda DISPLAY FILES

Afișează informații despre un fișier.

DISPLAY FILES [ON <drive>] [LIKE <șablon>] [TO PRINTER [PROMPT]] | TO FILE
<nume_fișier>]

Unde:

- ON <drive> - specifică calea către fișiere;
- LIKE <șablon> - specifică condiția pentru care vor fi afișate informații, doar despre acele fișiere care se încadrează în șablon;
- TO PRINTER [PROMPT] | TO FILE <nume_fișier> - direcționează informațiile la imprimantă (cu afișarea unei casete de dialog, clauza PROMPT) sau într-un fișier.

Informații identice se pot obține și cu comanda DIR care are aceleași clauze.

9. Programarea procedurală

Mediul VFP oferă pe lângă modul de lucru în fereastra de comandă (stil interpretor de comenzi) și posibilitatea de a dezvolta programe sursă în limbajul propriu sistemului (stil compilator), prin comenzi și funcții care descriu datele și pot efectua prelucrările necesare dezvoltării aplicației.

Programul sursă folosește atât tehnica de programare procedurală (structurată, modulară) cât și cea de programare pe obiecte. Într-un program sursă pot fi utilizate de asemenea și comenzile din nucleul SQL, care permit o programare descriptivă și de manipulare a datelor la nivel de tabelă.

9.1. Programarea structurată

Maniera de programare care folosește numai cele trei structuri de bază: structura liniară, structura alternativă și structura repetitivă, este cunoscută sub numele de programare structurată. Se poate demonstra ca orice program poate fi realizat utilizând numai cele trei structuri de bază

Limbajul VFP are comenzi specifice pentru implementarea celor trei structuri de program fundamentale: *secvențială*, *alternativă* (IF și CASE) și *repetitivă* (WHILE, FOR și SCAN), dar nu are comenzi pentru salt necondiționat.

Tehnica *programării modulare* se utilizează la rezolvarea unor probleme complexe și implică descompunerea problemei în părți componente (module), conform unei metode (top-down, bottom-up) și analiza lor, care stabilește:

- caracteristicile principale ale fiecărui modul;
- legăturile dintre module;
- alegerea unui modul principal (program, meniu sau videoformat);
- ierarhizarea modulelor față de ansamblu.

Aceste subprobleme sunt apoi programate utilizând tehnica programării structurate.

Limbajul VFP oferă facilitatea de creare a unui proiect (§12.2. Figura 1), în care aceste module pot fi declarate în una din categoriile implementate:

- **Data** – se pot declara baze de date (**Databases**), tabele libere (**Free Tables**), interogări (**Query**);
- **Class Libraries** – librării de clase;
- **Documents** - se pot declara videoformate (**Forms**), rapoarte (**Reports**), etichete (**Labels**);
- **Code** – programe (**Programs**), aplicații (**Applications**);
- **Other** – alte componente: meniuri (**Menus**), fișiere de tip text (**Text Files**);

- **Other Files** – alte fișiere utilizate: icon-uri, fișiere .BMP etc.

Toate aceste componente sunt înglobate în proiect, care constituie baza pentru crearea aplicației în format executabil.

9.1.1. Structura liniară

O structură liniară este constituită dintr-o succesiune de comenzi, dintre cele admise și în modul de lucru interpretor (în fereastra de comenzi). O astfel de structură este o secvență de comenzi care nu conține structuri alternative sau repetitive, ci doar comenzi, funcții și atribuiri de valori unor variabile.

Comenzi de afișare/citire la monitor

Comanda utilizată pentru afișarea la monitor, a unei variabile/constante de tip caracter sau numeric este:

@<linie>,<coloana> SAY <expr> [PICTURE <exp_c1>] | [FUNCTION <exp_c2>] [FONT <exp_c3>[,<exp_n>]] [STYLE <exp_c4>]

unde:

- @<linie>,<coloana> - caracterul @ fixează coordonatele de afișare, date sub formă de linie și coloană.
- SAY <expr> - comanda de afișare pe ecran, la coordonatele fixate, a unei expresii sau a unei funcții.
- PICTURE <exp_c1> - formatul de afișare (se utilizează codurile din tabelă indicându-se formatul între apostrof: '9999.99', pentru date de tip numeric).
- FUNCTION<exp_c2> - oferă o alternativă la stabilirea formatului.

exp_c2	Semnificație
A	Caractere alfabetice
L	Date de tip logic (.T., .F., .Y., .N.)
N	Litere și cifre
X	Orice caractere
9	Cu date de tip caracter permite numai numere. Cu date de tip numeric permite numere și semne algebrice
#	Permite cifre, semne algebrice și spații
\$	Afișează semnul curent al monedei (definit cu SET CURRENCY)

- FONT <exp_c3>[,<exp_n>] – specifică numele tipului de literă (font) utilizat și mărimea acesteia.
- [STYLE <exp_c4> - definirea stilului de afișare (B – bold, I – italic, N – normal, U - subliniat).

Exemplu. Următoarea secvență de comenzi:

```
@1,1 SAY 'TEST' FUNCTION 'A' FONT 'ARIAL',14 STYLE 'BIU'
```

```
S=10
```

```
@3,1 SAY S PICTURE '99.99'
```

```
WAIT WINDOW && afișarea unei ferestre de continuare
```

```
CLEAR
```

are ca efect afișarea în spațiul de lucru, în linia 1, coloana 1, a cuvântului TEST (îngroșat, înclinat, subliniat), iar în linia 3, coloana 1, valoarea variabilei *S* cu formatul ales.

Se pot afișa în spațiul de lucru sau tipări obiecte de tip BMP (imagini) cu comanda

```
@<linie>,<coloana> SAY <fișier_imag>BITMAP|<câmp_GEN> [STYLE <exprC>] [CENTER]
[ISOMETRIC] | [STRETCH] [SIZE <expN1>,<expN2>]
```

unde:

- <fișier_imag>BITMAP|<câmp_GEN> - clauza care face referire la fișierul de tip BMP (Bitmap Picture) sau imaginea conținută într-un câmp general dintr-un fișier. <fișier_imag> este o expresie de tip caracter, delimitată de apostrof sau ghilimele;
- STYLE <exprC> – asignează atributul de opac – **Q** sau transparent – **T**, pentru obiectele inserate;
- CENTER – clauza pentru plasarea obiectului BMP în centrul unei arii determinate prin clauza SIZE sau delimitate de coordonatele linie, coloană;
- ISOMETRIC | STRETCH – clauze pentru cazul în care zona de afișare a obiectului BMP este mai mică decât dimensiunea reală a acestuia. ISOMETRIC redimensionează obiectul la dimensiunea ariei, cu păstrarea proporțiilor. STRECH crează un raport separat pentru fiecare coordonată *Ox*, *Oy*, cu care obiectul se poate distorsiona pe verticală sau orizontală.
- SIZE <expN1>,<expN2>] – stabilește coordonatele colțului din dreapta jos al zonei de afișare.

Exemplu.

```
@1,1 SAY "1.JPG" BITMAP CENTER ISOMETRIC
WAIT WINDOW
CLEAR
```

Comanda

`@<linie>,<coloana> GET <var> [PICTURE <expr_c>]`

permite editarea variabilei de memorie *var* (pentru a introduce o nouă valoare), unde:

- PICTURE <expr_c> - reprezintă formatul de editare.

Se poate combina operația de afișare cu cea de editare a unei variabile, caz în care vom utiliza comanda:

`@<linie>,<coloana> SAY <mesaj> GET <var>`

Întotdeauna o comandă (sau mai multe comenzi) @...GET va fi însoțită de o comandă READ. Practic se poate construi o machetă cu câmpuri de afișare @...SAY și câmpuri de citire @...GET, cursorul se deplasează de la o linie la alta apăsând tasta [ENTER] sau [TAB] și înapoi cu [SHIFT] + [TAB] (permite reactualizarea unor câmpuri înainte de citirea efectivă). La apăsarea tastei [ENTER] la ultima linie GET se va executa comanda READ și se va citi întreg ecranul.

Comanda READ realizează citirea tuturor câmpurilor editate cu comanda GET.

`READ [CYCLE] [MODAL] [VALID <expr_L>|<expr_N>] [WHEN <expr_L1>]`

unde:

- CYCLE – clauză care împiedică încheierea citirii variabilelor editate, atunci când cursorul depășește primul sau ultimul obiect creat cu GET. Dacă utilizatorul se plasează pe ultimul obiect GET și apasă [ENTER] sau [TAB], cursorul se va deplasa pe primul obiect GET. Pentru a încheia o comandă READ ciclic, se apasă tasta [Esc] sau tastele [CTRL] + [W] ;
- MODAL – este clauza care previne activarea ferestrelor care nu sunt implicate în execuția comenzii READ;
- VALID *expr_L* | *expr_N* – clauză care validează valorile introduse în câmpul GET, la terminarea comenzii READ. Dacă expresia logică returnează valoarea adevărat (.T.) comanda READ se încheie;

- **WHEN** *expr_L1* – clauza condiționează execuția comenzii **READ** de evaluarea expresiei logice. Dacă expresia returnează valoarea fals comanda **READ** este ignorată.

9.1.2. Structura alternativă

Structura alternativă este implementată în cele două forme:

- cu două ramuri (comenzile **IF**, **IIF**);
- cu mai multe ramuri (comanda **DO CASE**).

Comanda IF...ENDIF execută condiționat un set de comenzi, în funcție de valoarea logică returnată de o expresie evaluată. Expresia evaluată nu furnizează decât două alternative (ramuri) corespunzătoare valorilor adevărat (**THEN**) sau fals (**ELSE**).

IF <*exprL*> [THEN] <*comenzi*> [ELSE <*comenzi*>] ENDIF

unde:

- *exprL* – specifică expresia logică ce este evaluată. Dacă cele două clauze **THEN** și **ELSE** sunt prezente și expresia este adevărată vor fi executate comenzile de pe ramura **THEN**, dacă expresia este falsă, vor fi executate comenzile de pe ramura **ELSE**;
- Dacă expresia logică este falsă și ramura **ELSE** nu este inclusă, va fi executată prima comandă care urmează clauzei **ENDIF**;
- Clauza **ENDIF** trebuie inclusă la orice declarare a unui **IF**. O comandă **IF** poate include mai multe blocuri **IF...ENDIF** pe ramurile sale (îmbricare), cu condiția ca închiderea lor (cu **ENDIF**) să se facă în aceeași ordine ca și deschiderea (nu se admite intersecția blocurilor).

Exemplu. În fișierul **PRODUSE**, să se caute un anumit produs; dacă este găsit să se afișeze câmpurile **PRODUS**, **CANT**, **PU**, dacă nu, să se afișeze un mesaj.

```
CLEAR
USE PRODUSE
GET EXPR 'INTRODUCETI CONDITIA DE LOCALIZARE' TO
GTEMP;
TYPE 'L' DEFAULT 'PRODUS = ""'
LOCATE FOR &gtemp
&& caută produsul cu comanda LOCATE
```

```

IF FOUND( )
&& dacă este găsit
DISPLAY FIELDS PRODUS,CANT,PU&& afișează câmpurile PRODUS, CANT, PU
ELSE
&& dacă nu
    ? 'CONDITIA ' + GTEMP + ' NU A FOST GASITA '
&& afișează mesajul
ENDIF
CLOSE TABLE

```

Funcția IIF returnează pe baza evaluării unei expresii logice, una din cele două valori ale unei expresii de tip numeric, caracter, monedă, dată calendaristică sau timp.

$$\text{IIF}(\text{expr}_L, \text{expr}_1, \text{expr}_2)$$

unde: *expr_L* – specifică expresia logică pe care funcția o evaluează.

- Dacă expresia logică este adevărată, va returnată valoarea expresiei *expr₁*.
- Dacă expresia logică este falsă va fi returnată valoarea dată de *expr₂*.

Această funcție este cunoscută și sub numele de IF scurt (imediat). Este folosită mai ales la expresiile care condiționează afișarea conținutului unui câmp, în rapoarte sau etichete. Execuția sa este mai rapidă decât echivalentul său IF...ENDIF.

Comanda DO CASE...ENDCASE

Execută primul set de comenzi ale căror expresie condițională este adevărată. Comanda este echivalentul unor instrucțiuni IF în cascadă.

```

DO CASE
    CASE <expr_L_1>
        <comenzi1>
    CASE <expr_L_2>
        <comenzi2>
    ....
    CASE <expr_L_n>
        <comenzi_n>

```

[OTHERWISE

<comenzi>]

ENDCASE

Acțiunea comenzii:

- CASE <expr_L_1> <comenzi1>... - când prima expresie logică CASE întâlnită, are valoarea adevărat, blocul de comenzi va fi executat până la apariția unei noi comenzi CASE sau a clauzei ENDCASE. Execuția programului continuă cu prima instrucțiune executabilă care urmează lui ENDCASE;
- Dacă expresia din prima clauza CASE este falsă, blocul de comenzi aferent este ignorat și se trece la evaluarea expresiei logice a următoarei clauze CASE;
- După găsirea primei expresii logice adevărată, orice altă clauză CASE pentru care expresia logică este adevărată, va fi ignorată;
- OTHERWISE <comenzi> - dacă toate expresiile logice CASE au returnat valoarea fals, clauza oferă o alternativă pentru introducerea unui bloc de comenzi.

Exemplu

@1,1 SAY "DATI N=" GET N

READ

DO CASE

CASE N = 0

? 'ZERO'

CASE N > 0

? 'POZITIV'

OTHERWISE

? 'NEGATIV'

ENDCASE

WAIT WINDOW "Apasati Enter"

9.1.3. Structura repetitivă

Pentru descrierea structurilor repetitive limbajul pune la dispoziție următoarele tipuri de comenzi:

- ciclul de repetiție cu număr finit de pași - comanda FOR...ENDFOR;

- ciclu cu număr nedefinit de pași, condiționat anterior - comanda DO WHILE...ENDDO;
- o comandă de parcurgere a înregistrărilor unei tabele de la început la sfârșitul fișierului – comanda SCAN...ENDSCAN.

Comanda FOR...ENDFOR execută un bloc de comenzi de un număr de ori precizat.

FOR <var>=<val_inițială> TO <val_finală> [STEP <val_increment>] <comenzi> [EXIT]
[LOOP] ENDFOR | NEXT

unde:

- *var* – specifică o variabilă sau un element de masiv, de tip întreg, care acționează ca un contor (variabilă de ciclare);
- *val_inițială*, *val_finală* – valoarea inițială, respectiv finală a contorului;
- STEP <val_increment> - domeniul dat de valoarea inițială / finală poate fi parcurs cu incrementul 1 (implicit) sau cu un pas (increment) precizat prin *val_increment*. Dacă *val_increment* este negativ contorul este decrementat;
- comenzi – specifică blocul de comenzi care vor fi executate în interiorul ciclului.
- EXIT – permite ieșirea forțată din ciclu (înainte de atingerea valorii finale a variabilei de ciclare). În program va fi executată prima comandă care urmează clauzei ENDFOR;
- LOOP – returnează controlul comenzii FOR, fără a se mai executa comenzile dintre LOOP și ENDFOR. Contorul este incrementat sau decrementat ca și cum s-ar ajunge la clauza ENDFOR.

Exemplu. Din fișierul CLIENTI, se vor afișa la monitor, din primele 10 înregistrări, din doi în doi, clienții.

USE CLIENTI

STORE 2 TO VI,K && valoarea inițială și pasul contorului

STORE 10 TO VF && valoarea finală a contorului

FOR I=VI TO VF STEP K && I variabila contor

 GOTO I && poziționare pe înregistrarea I

 DISPLAY FIRMA && se afișează numele firmei de pe poziția I din fișierul CLIENTI

ENDFOR && sfârșitul ciclului FOR

CLOSE TABLE

Comanda DO WHILE ... ENDDO execută un bloc de comenzi în interiorul unui ciclu condiționat anterior.

DO WHILE <expr_L> <comenzi> [LOOP] [EXIT] ENDDO

unde:

- *expr_L* – specifică o expresie logică a cărei valoare de adevăr determină dacă blocul de comenzi cuprins între DO WHILE și ENDDO va fi executat. Atâta vreme cât condiția logică este adevărată, comenzile vor fi executate;
- LOOP – returnează controlul comenzii DO WHILE. Comenzile între LOOP și ENDDO nu vor fi executate;
- EXIT – transferă controlul, din interiorul ciclului DO WHILE, către prima comandă care urmează clauzei ENDDO. Clauza este echivalentă cu o ieșire forțată din ciclu;
- Fiecare comandă DO WHILE trebuie să aibă un corespondent (să se închidă) ENDDO.

Exemplu. Din fișierul PRODUSE să se afișeze totalul produselor în stoc, care au prețul mai mare de 30\$.

CLEAR

USE PRODUSE

STOC=0

DO WHILE !EOF()

IF PU<30

SKIP

ENDIF

STOC=STOC+CANT

? PRODUS,CANT

SKIP

ENDDO

CLOSE TABLES

?STOC

Comanda SCAN...ENDSCAN realizează mutarea pointer-ului prin toate înregistrările din tabela curentă și execută un bloc de comenzi pentru înregistrările care corespund condițiilor specificate.

SCAN [Scope] [FOR <expr_L_1>] [WHILE <expr_L_2>] [comenzi] [LOOP] [EXIT]
[ENDSCAN]

unde:

- *Scope* – specifică domeniul înregistrărilor care vor fi scanate (clauze: **ALL**, **NEXT** *n*, **RECORD** *n*, **REST**). Clauza implicită este **ALL**;
- **FOR** *expr_L_1* – execută comenzile doar pentru înregistrările pentru care expresia logică *expr_L_1* este adevărată. Includerea clauzei permite filtrarea înregistrărilor, eliminând pe cele pe care nu trebuie să fie scanate;
- **WHILE** *expr_L_2* – specifică o condiție pentru care blocul de comenzi va fi executat atâta vreme cât valoarea expresiei logice *expr_L_2* este adevărată;
- **LOOP** – returnează controlul înapoi la comanda **SCAN** (se reia execuția comenzii);
- **EXIT** – transferă programului controlul din interiorul buclei **SCAN...ENDSCAN**, la prima comandă care urmează clauzei **ENDSCAN**;
- **ENDSCAN** – indică sfârșitul procedurii de scanare.

Exemplu. Utilizând comanda **SCAN**, să se afișeze din fișierul **CLIENTI**, numele firmei, orașul și persoana de contact, cu care o firmă are contracte în **FRANȚA**.

```
USE CLIENTI
SCAN FOR UPPER(TARA)='FRANTA'
    ? FIRMA,ORAS,CONTACT
ENDSCAN
CLOSE TABLES
```

9.2. Modularizarea programelor

Modularizarea programelor, în cazul unor aplicații complexe, se poate realiza prin proceduri și funcții definite de utilizator.

Definirea unei funcții utilizator, reprezintă un bloc de comenzi independent, care primește un set de parametri de la programul apelant și returnează acestuia o valoare ca rezultat al prelucrării parametrilor de intrare.

Definirea unei proceduri de către utilizator, este de asemenea un bloc de comenzi, care prelucrează parametri transmiși din programul apelant. După prelucrare controlul este redat programului apelant.

Din punct de vedere al locației, funcțiile se pot defini fie în programul apelant, fie într-o librărie sau pot fi salvate în fișiere externe, de tip **.PRG**.

9.2.1. Proceduri

Asocierea procedurilor cu programul apelant se face cu comanda:

SET PROCEDURE TO [<fișier_1> [,<fișier_2>,...]] [ADDITIVE]

unde:

- *fișier_1, fișier_2* – specifică succesiunea în care vor fi deschise fișierele. Se pot declara mai multe nume de fișiere care conțin proceduri. Această opțiune permite crearea unor librării de sine stătătoare de funcții care pot fi specificate separat.
- **ADDITIVE** – se deschid fișierele cu proceduri din lista specificată, fără a fi închise fișierele cu proceduri, deja deschise.

Cu comanda **RELEASE PROCEDURE** [<fișier_1> [,<fișier_2>,...]] se închid fișierele individuale.

Variabilele definite în interiorul procedurilor și funcțiilor utilizator, sunt la nivel local (nu se văd din afara procedurilor sau funcțiilor).

Structural o procedură, cuprinde următoarele elemente:

PROCEDURE <i>nume_procedură</i>	→ specifică numele procedurii create
PARAMETERS <i>listă_parametri</i>	→ declararea parametrilor formali de intrare
<i>Bloc de comenzi</i>	→ bloc de comenzi, corpul procedurii
RETURN [<i>expresie</i>]	→ ieșirea din procedură, opțional cu returnarea unei expresii calculate.

unde:

- **PROCEDURE** – este o declarație în interiorul unui program, care specifică începutul unei proceduri și definește numele procedurii. Numele procedurii poate începe cu o literă sau caracterul “_” (underscore) și poate conține orice combinație de litere, numere și caractere underscore. Lungimea maximă a numelui este de 254 caractere;
- **PARAMETERS** – asigură trecerea parametrilor din programul apelant în procedură. Se pot transmite maxim 27 de parametri în procedură. Lista parametrilor poate cuprinde: expresii, variabile de memorie, nume de tabele, nume de câmpuri din tabele de date, constante, care sunt transmise prin referință;
- **RETURN** [*expresie*] – returnează controlul și opțional o expresie, în programul apelant;

- Parametrii pot fi transmiși în procedură, prin includerea clauzei PARAMETERS în procedură, sau prin plasarea unei liste de parametri imediat după PROCEDURE *nume_procedură*. Lista se închide între paranteze, iar parametrii sunt separați cu virgulă.

Apelul unei proceduri sau program se face cu comanda

DO <nume_procedură> | <nume_program> WITH <listă_parametri>

Observații

- Metoda folosită la transmiterea parametrilor la proceduri sau funcții este implicit prin valoare, iar la programe prin referință;
- Variabilele transmise către procedură cu comanda DO...WITH sunt transmise prin referință.

Sistemul VFP are anumite *restricții legate de utilizarea comenzii DO*:

- pot fi imbricate maxim 32 de apeluri DO (*program principal* | *procedura1* | *procedura2* |...);
- nu se poate face apel dintr-o procedură la ea însăși (nu admite recursivitatea directă);
- din interiorul unei proceduri nu poate fi apelat programul/procedura apelantă (nu admite recursivitatea indirectă).

9.2.2. Funcții

Elementele unei funcții definite de utilizator (UDF) sunt:

FUNCTION <i>nume_funcție</i>	→ specifică numele funcției create
<i>Bloc de comenzi</i>	→ bloc de comenzi, corpul funcției
RETURN [<i>expresie</i>]	→ ieșirea din funcție, opțional cu returnarea unei expresii calculate.

Implicit parametrii sunt transmiși în funcție prin valoare. Numărul maxim de parametri care poate fi transmis, este 27. Parametrii pot fi transmiși în funcție, prin includerea clauzei PARAMETERS în funcție, sau prin plasarea unei liste de parametri imediat după FUNCTION *nume_funcție*. Lista se închide între paranteze, parametrii sunt separați cu virgulă.

Comanda

SET UDFPARAMS TO VALUE | REFERENCE

specifică dacă parametrii sunt transmiși către o funcție definită de utilizator (UDF) prin valoare sau prin referință.

Când o variabilă este transmisă prin valoare, valoarea variabilei poate fi modificată în funcția definită de utilizator, dar valoarea originală a variabilei în programul apelant nu se modifică.

Când o variabilă este transmisă prin referință și funcția definită de utilizator modifică valoarea variabilei transmise, valoarea originală a variabilei în programul apelant de asemenea se modifică.

Implicit parametrii sunt transmiși prin valoare. Se poate forța transmiterea parametrilor către o funcție utilizator, în funcție de opțiunea aleasă în comanda SET UDFPARAMS. Variabilele se închid între paranteze pentru a forța transmiterea prin valoare. Forțarea transmiterii unui parametru prin referință se face punând în fața parametrului caracterul @.

Exemplul 1.

*** Transmiterea Variabilelor Prin Valoare ***

CLEAR

SET TALK OFF

WAIT 'APASATI O TASTA PENTRU A TRANSMITE VARIABILA PRIN VALOARE'

WINDOW

SET UDFPARMS TO VALUE

STORE 1 TO GNX

*** Valoarea Lui Gnx Este Neschimbata ***

@ 2,2 SAY 'VALOARE UDF: ' + STR(PLUSUNU(GNX))

@ 4,2 SAY 'VALOARE GNX: ' + STR(GNX)

*** TRANSMITEREA VARIABILEI PRIN REFERINTA ***

WAIT ' APASATI O TASTA PENTRU A TRANSMITE VARIABILA PRIN REFERINTA'

WINDOW

CLEAR

SET UDFPARMS TO REFERENCE

STORE 1 TO GNX

*** Valoarea Lui Gnx Se Schimba ***

@ 2,2 SAY 'VALOARE UDF: ' + STR(PLUSUNU(GNX))

@ 4,2 SAY 'VALUE OF GNX: ' + STR(GNX)

SET UDFPARMS TO VALUE

*** Functie Utilizator (Udf) Care Aduna Unu La Un Numar ***

FUNCTION PLUSUNU

PARAMETER GNZ

GNZ = GNZ + 1

RETURN GNZ

Următorul exemplu arată cum sunt transmise variabilele prin valoare și prin referință, utilizând parantezele și caracterul @.

Exemplul 2.

*** Transmiterea Variabilei Prin Valoare ***

CLEAR

SET TALK OFF

WAIT ' APASATI O TASTA PENTRU A TRANSMITE VARIABILA PRIN VALOARE '

WINDOW

STORE 1 TO GNX

@ 2,2 SAY 'VALOARE UDF: ' + STR(PLUSUNU((GNX)))

@ 4,2 SAY 'VALOAREA GNX: ' + STR(GNX)

*** Transmiterea Variabilei Prin Referinta ***

WAIT ' APASATI O TASTA PENTRU A TRANSMITE VARIABILA PRIN REFERINTA '

WINDOW

CLEAR

STORE 1 TO GNX

@ 2,2 SAY 'VALOARE UDF: ' + STR(PLUSUNU(@GNX))

@ 4,2 SAY 'VALOARE GNX: ' + STR(GNX)

FUNCTION PLUSUNU(GNZ)

GNZ = GNZ + 1

RETURN GNZ

10. Comenzi ale nucleului SQL

Sistemul VFP suportă comenzile SQL care operează la nivel de tabelă (relație) și o singură comandă SQL poate fi folosită pentru a înlocui o întreagă secvență de comenzi.

Sistemul VFP poate executa următoarele comenzi SQL:

1. Comanda SELECT – SQL

Comanda specifică criteriile pe baza cărora se bazează și lansează interogarea (**Query**). Sistemul VFP interpretează interogarea și extrage datele cerute din tabelă (tabele). Comanda SELECT pentru interogare poate fi lansată din următoarele zone:

- în fereastra de comenzi (modul de lucru interpretor);
- din interiorul unui program VFP;
- utilizând asistentul de proiectare pentru interogare (**Query Designer**).

Sintaxa comenzii:

```
SELECT [ALL | DISTINCT] [TOP expr_num [PERCENT]] alias.]art_selectat [AS nume_coloană]
[[alias.]art_selectat [AS nume_coloană]...] ROM [FORCE] nume_bază_de_date!]tabelă [[AS]
alias_local] [INNER | LEFT [OUTER] | RIGHT [OUTER] | FULL [OUTER] JOIN
nume_bază_de_date!]tabelă [[AS] alias_local] [ON condiție_join [[INTO destinație] | [TO FILE
nume_fișier [ADDITIVE] | TO PRINTER [PROMPT] | TO SCREEN ]] [NOCONSOLE] [PLAIN]
[WHERE condiție_join [AND condiție_join [AND | OR condiție_filtrare [AND | OR
condiție_filtrare]] [GROUP BY coloană_grup [,coloană_grup] [HAVING condiție_filtrare]
[UNION [ALL] comandă_selectare] [ORDER BY articol_ordonat [ASC | DESC]
[,articol_ordonat [ASC | DESC]...]]
```

Semnificația clauzelor:

- SELECT – precizează câmpurile, constantele și expresiile care vor fi afișate în urma interogării;
- ALL – implicit se vor afișa toate înregistrările din tabelă care îndeplinesc condițiile de selecție în rezultatul interogării;
- DISTINCT – se exclud duplicatele din înregistrările care îndeplinesc condițiile de selecție. Argumentul se poate folosi o singură dată în cadrul unei clauze SELECT;

- **TOP *expr_num* [PERCENT]** – rezultatul interogării va conține un număr specificat de înregistrări sau un anumit procent din înregistrări. Dacă se folosește clauza TOP trebuie inclusă și clauza ORDER BY, care specifică coloana din care clauza TOP, va determina numărul de înregistrări (linii) care vor fi incluse în rezultatul interogării. Se pot specifica înregistrările în domeniul 1..32767. Înregistrările care au valori identice pentru coloanele specificate cu clauza ORDER BY sunt incluse în rezultatul interogării. De exemplu, dacă se specifică valoarea 10 pentru *expr_num*, rezultatul interogării poate conține mai mult de 10 înregistrări dacă există în tabelă mai mult de 10 linii care au valori identice în coloanele specificate cu clauza ORDER BY;
- **alias.** – se folosește pentru identificarea numelui câmpurilor (coloanelor) prin calificare. Fiecare câmp specificat cu *art_selectat* va genera o coloană în rezultatul interogării. Dacă două sau mai multe câmpuri au același nume, se va include alias-ul tabelii și caracterul punct (.) în fața numelui câmpului, pentru a preveni duplicarea coloanei (calificare cu punct);
- **art_selectat** – specifică câmpul care trebuie inclus în rezultatul interogării. Acesta poate fi:
 - numele unui câmp din tabela care apare în clauza FROM;
 - constantă care specifică o valoare, care va apare în fiecare înregistrare din rezultatul interogării;
 - expresie care poate fi numele unei funcții utilizator;
- **AS *nume_coloană*** – specifică numele coloanei care va apare la ieșire, în rezultatul interogării. Această opțiune este utilă atunci când *art_selectat* este o expresie sau conține un câmp calculat și vrem să dăm un nume sugestiv coloanei respective. Nu sunt admise decât caracterele care se folosesc la stabilirea numelui câmpurilor la proiectarea tabelilor (de exemplu nu se folosesc spații);
- **FROM** – se declară tabelele care conțin datele pe care vrem să le extragem în urma interogării. Dacă nu este deschis nici o tabelă, VFP va deschide caseta de dialog **Open** pentru a specifica locația fișierului care conține datele;
- **FORCE** – clauza prin care specifică că tabele sunt unite (join) în ordinea în care ele apar în clauza FROM;
- **nume_bază_de_date![tabelă]** – specifică numele unei baze de date, care nu este deschisă, care conține tabela cu date. Clauza se folosește în cazul în care tabela cu date nu face parte din baza de date curentă (deschisă). Semnul ! se folosește pentru adresarea tabelii prin calificare;

- [AS] *alias_local* – specifică un nume temporar pentru tabela menționată în argumentul tabelă;
- INNER JOIN – rezultatul interogării va conține numai înregistrările din tabelă care coincid cu una sau mai multe înregistrări, din altă tabelă (joncțiune internă);
- LEFT [OUTER] JOIN – rezultatul interogării va conține toate înregistrările din tabelă în stânga cuvântului JOIN și înregistrările care coincid vor fi afișate la dreapta cuvântului JOIN. Clauza OUTER (exterior) este opțională, se include pentru a sublinia faptul că se crează o joncțiune exterioară;
- RIGHT [OUTER] JOIN – rezultatul interogării va conține toate înregistrările din tabelă la dreapta cuvântului JOIN și înregistrările care coincid în stânga;
- FULL [OUTER] JOIN – rezultatul interogării va conține atât înregistrările care coincid, cât și cele care nu coincid, din cele două tabele;
- [*nume_bază_de_date*!] *tabelă* [[AS] *alias_local*] – specifică numele tabelii (baza de date) cu care se realizează joncțiunea (eventual definit ca alias local);
- ON *condiție_join* – specifică condiția pentru care tabelele sunt unite (fac joncțiune);
- INTO *destinație* .- specifică locul unde vor fi stocate rezultatele interogării. Destinația poate fi una din următoarele clauze:
 - ARRAY *nume_tablou* – va stoca rezultatele într-o variabilă de memorie de tip matrice. Variabila nu se crează dacă rezultatul interogării furnizează 0 înregistrări;
 - CURSOR *nume_cursor* – stochează rezultatele interogării într-un cursor (fișier temporar). După executarea comenzii SELECT, cursorul temporar rămâne deschis (read-only) și este activ. După închiderea cursorului temporar, fișierul este șters;
 - DBF | TABLE *nume_tabelă* – stochează rezultatele interogării într-o tabelă;
 - TO FILE *nume_fișier* – clauză prin care rezultatele interogării sunt direcționate către un fișier de tip ASCII (text);
 - ADDITIVE – adaugă rezultatele interogării la conținutul existent al fișierului de tip text specificat la clauza TO FILE;
 - TO PRINTER [PROMPT] – direcționează ieșirea către imprimantă;
 - TO SCREEN – direcționează ieșirea în fereastra principală a sistemului VFP sau în fereastra activă definită de utilizator;
 - NOCONSOLE – împiedică afișarea rezultatelor interogării trimise către un fișier, imprimantă sau fereastra principală;
- PLAIN – împiedică afișarea capului coloanei (numele coloanei) la ieșirea din interogare;

- WHERE - specifică includerea doar a anumitor înregistrări în rezultatele interogării;
- *condiție_join* [AND *condiție_join...*] – precizează câmpurile care leagă tabele din clauza FROM. Se include operatorul AND pentru a lega condiții multiple de unire (joncțiune);
- AND | OR *condiție_filtrare* [AND | OR *condiție_filtrare...*] – specifică criteriile pe care trebuie să le îndeplinească înregistrările pentru a fi incluse în rezultatul interogării. Se pot declara mai multe condiții de filtrare, legate prin operatorii AND și/sau OR. Se poate folosi de asemenea și operatorul NOT pentru a inversa valoarea expresiei logice;
- GROUP BY *coloană_grup* [,*coloană_grup...*] – grupează înregistrările pe baza valorilor din una sau mai multe coloane; *coloană_grup* poate fi numele unui câmp dintr-o tabela obișnuită, sau un câmp care include o funcție SQL, sau o expresie numerică care indică locația coloanei în tabela rezultată;
- HAVING *condiție_filtrare* – specifică o condiție de filtrare pe care grupul trebuie să o îndeplinească pentru a putea fi inclus în rezultatele interogării. HAVING trebuie utilizat împreună cu clauza GROUP BY. Clauza HAVING fără clauza GROUP BY acționează ca și clauza WHERE;
- UNION [ALL] *comandă_selectare* - combină rezultatele finale ale unei clauze SELECT cu rezultatele finale ale altei clauze SELECT. Implicit UNION verifică rezultatele combinate și elimină înregistrările duplicat. Clauza ALL împiedică acțiunea clauzei UNION de a elimina înregistrările duplicat.
- ORDER BY *articol_ordonat* ASC | DESC – sortează rezultatele interogării pe baza datelor din una sau mai multe coloane. Fiecare *articol_ordonat* trebuie să corespundă unei coloane din rezultatele interogării. ASC respectiv DESC specifică ordinea sortării (ascendent, descendent).

Exemplu. Din tabelele CLIENȚI și COMENZI să se selecteze firma (CLIENȚI), data comenzii și mijlocul de transport (COMENZI), sortate ascendent după dată. Rezultatul se va scrie în fișierul TRANSPORT.DBF.

OPEN DATABASES ('TEST')

SELECT A.FIRMA, B.DATA_CDA, B.MIJLOC FROM CLIENTI A, COMENZI B ;

WHERE A.COD_FIRMA=B.COD_FIRMA ;

ORDER BY B.DATA_CDA ASC INTO TABLE TRANSPORT

BROWSE

CLOSE DATABASES

2. Comanda ALTER TABLE – SQL modifică o tabelă existentă.

Sintaxa comenzii :

```
ALTER TABLE nume_tabelă_1 ADD      | ALTER [COLUMN] nume_câmp1, tip_câmp  
[(mărime_câmp [,precizie])] [NULL | NOT NULL] [PRIMARY KEY | UNIQUE]  
[REFERENCES nume_tabelă_2 [TAG nume_etichetă]]
```

unde:

- *nume_tabelă_1* – specifică numele tabelii a cărei structură se modifică;
- ADD [COLUMN] *nume_câmp1* – specifică numele câmpului care se adaugă;
- ALTER [COLUMN] *nume_câmp1* – specifică numele unui câmp existent care se modifică;
- *tip_câmp*[(*mărime_câmp* [,*precizie*])] – specifică tipul câmpului, mărimea și precizia pentru un câmp nou sau pentru modificarea unui câmp existent;
- NULL | NOT NULL – permite declararea unui câmp care acceptă sau nu valori de tip **NUL**;
- PRIMARY KEY – crează o etichetă primară de index. Eticheta de index are același nume cu cel al câmpului;
- UNIQUE – crează o etichetă de index candidat, cu același nume cu cel al câmpului;
- REFERENCES *nume_tabelă_2* TAG *nume_etichetă* – specifică tabela părinte către care se stabilește o relație persistentă. TAG *nume_etichetă* specifică eticheta de index din tabela părinte pe baza căreia se stabilește relația.

3. Comanda UPDATE – SQL actualizează înregistrările dintr-o tabelă. Înregistrările pot fi actualizate pe baza rezultatelor unei declarații SELECT – SQL.

Sintaxa comenzii:

```
UPDATE [nume_BD!] nume_tabelă SET nume_coloană=exprL_1 [, nume_coloană=exprL_2...]  
WHERE condiție_filtrare_1 [AND | OR condiție_filtrare_2...]]
```

unde:

- *[nume_BD!]nume_tabelă* – specifică tabela în care vor fi actualizate înregistrările, cu noile valori;
- *SET nume_coloană=exprL_1[, nume_coloană=exprL_2...]* – specifică coloanele care sunt actualizate și noile valori. Dacă se omite clauza, fiecare înregistrare din coloană va fi actualizată cu aceeași valoare;
- *WHERE condiție_filtrare_1[AND | OR condiție_filtrare_2...]* – specifică înregistrările care vor fi actualizate cu noile valori; *condiție_filtrare_1* specifică criteriul pe care trebuie să-l îndeplinească înregistrările pentru a fi actualizate. Se pot include mai multe condiții de filtrare legate prin operatorii logici AND și/sau OR. Se poate folosi de asemenea operatorul NOT pentru a inversa valoarea expresiei logice.

Exemplu. În fișierul *CLIENTI* din baza de date *TEST*, să se modifice câmpul cantitate maximă (*CANT_MAX*) la valoarea 25.

```
OPEN DATABASES ('TEST')
USE CLIENTI
UPDATE CLIENTI SET CANT_MAX=25
BROWSE FIELDS FIRMA,CANT_MAX
CLOSE DATABASES
```

4. *Comanda INSERT* – SQL adaugă o nouă înregistrare la sfârșitul unei tabele existente. Noua înregistrare conține date descrise în comanda *INSERT* sau pot fi preluate dintr-un masiv.

Sintaxa comenzii:

```
INSERT INTO tabelă [(nume_câmp_1[, nume_câmp_2,...])] VALUES (valoare_1, valoare_2,...)
sau
INSERT INTO tabelă FROM ARRAY nume_masiv | FROM MEMVAR
```

unde:

- *tabelă* – specifică numele tablei în care se adaugă o nouă înregistrare;
- *(nume_câmp_1[, nume_câmp_2,...])* – specifică numele câmpurilor din noua înregistrare în care vor fi inserate valori;
- *(valoare_1, valoare_2,...)* – specifică valorile câmpurilor care vor fi inserate în noua înregistrare;

- *nume_masiv* – specifică numele masivului din care vor fi inserate datele în noua înregistrare;
- FROM MEMVAR – conținutul variabilelor va fi inserat în câmpurile care au același nume cu variabilele.

Exemplu. În tabela CLIEȚI din baza de date TEST conținutul înregistrării curente va fi transmis în memorie ca variabile și structura tabelului va fi copiată într-un noua tabelă CLIEȚI2.

CLOSE DATABASES

CLEAR

OPEN DATABASES ('TEST')

USE CLIENTI

* Se transmite înregistrarea curentă în memorie ca variabile

SCATTER MEMVAR

* Se copiază structura tabelului curent în tabela CLIENTI2

COPY STRUCTURE EXTENDED TO CLIENTI2

* Se inserează înregistrarea memorată în variabile

INSERT INTO CLIENTI2 FROM MEMVAR

SELECT CLIENTI2

BROWSE

USE

DELETE FILE CLIENTI2.DBF

5. *Comanda* CREATE CURSOR – SQL crează o tabelă temporară. Fiecare câmp din tabela temporară este definit cu nume, tip, precizie, număr zecimale, valoare **NULL** și reguli de integritate referențială. Aceste definiții pot fi obținute din comandă sau dintr-un masiv.

Sintaxa comenzii:

```
CREATE CURSOR nume_alias(nume_câmp_1 tip_câmp [precizie[,nr_zecimale])  
[NULL | NOT NULL] [CHECK expr_L [ERROR mesaj_eroare]] [DEFAULT expresie]  
[UNIQUE] [NOCPTRANS]] [, nume_câmp_2...]) | FROM ARRAY nume_masiv
```

unde:

- *nume_alias* – specifică numele tabeli temporare creată, care poate fi și numele unei expresii;
- *nume_câmp_1* – specifică numele unui câmp din fișierul temporar;
- *tip_câmp* – prin intermediul unei singure litere, specifică tipul fiecărui câmp;
- *precizie[,nr_zecimale]* – specifică mărimea și dacă este cazul, numărul de zecimale pentru câmpuri numerice;
- **NULL** | **NOT NULL** – alocă sau nu valori de tip **NULL** câmpului;
- **CHECK** *expr_L* – specifică regula de validare pentru valorile care se vor înscrie în câmp. Expresia logică *expr_L* poate fi și o funcție definită de utilizator;
- **ERROR** *mesaj_eroare* – specifică mesajul de eroare pe care sistemul VFP îl va afișa în cazul în care validarea datelor generează eroare;
- **DEFAULT** *expresie* – specifică valoarea implicită pentru câmp. Tipul de dată dat de expresie trebuie să fie de același fel cu tipul câmpului.
- **UNIQUE** – crează un index candidat pentru câmp. Eticheta de index candidat are același nume cu cel al câmpului. Valorile **NULL** și înregistrările duplicat nu sunt permise într-un câmp utilizat ca index candidat;
- **NOCPTRANS** – previne trecerea la o altă pagină de cod pentru caractere și câmpuri de tip memo. Dacă cursorul este convertit la o altă pagină de cod, câmpurile pentru care a fost specificată clauza **NOCPTRANS** nu vor fi translate;
- **FROM ARRAY** *nume_masiv* – specifică numele unui masiv existent care conține numele, tipul, precizia, numărul de zecimale pentru fiecare câmp din tabela temporară.

Exemplu. Se crează un cursor cu nume alias **ANGAJAȚI** cu următoarea structură:

ID N(5) – identificator angajat;

NUME C(25);

ADRESA C(30);

ORAS C(20);

TELEFON C(8) – care acceptă și valori de tip **Null**;

SPEC M – specialitatea, câmp de tip memo.

Se va adăuga o înregistrare goală, după care se vor înscrie valori în câmpurile tabeli.

CLOSE DATABASES

CLEAR

```

CREATE CURSOR ANGAJATI ;
(ID N(5), NUME C(25), ADRESA C(30), ORAS C(20), TELEFON C(8) NULL, SPEC M)
DISPLAY STRUCTURE
WAIT WINDOW 'Apasati o tasta pentru a introduce o înregistrare'
INSERT INTO ANGAJATI (ID, NUME, ADRESA, ORAS, MARCA, SPECIALITATE);
VALUES (1004,"DR. ION GARCEA","B-DUL LACUL TEI 124, SECTOR 2", "BUCURESTI",;
"88567902","PAZA SI PROTOCOL")
BROWSE
* În acest punct se poate copia înregistrarea într-o tabelă permanentă.
CLOSE ALL && odată ce cursorul s-a închis, toate datele sunt golite din memorie.
CLEAR

```

6. *Comanda CREATE TABLE* – SQL determină crearea unei tabele. La fiecare tabelă nou creată, i se specifică numele câmpurilor din înregistrare și caracteristicile lor: tip, mărime, zecimală (pentru tipurile numerice), valori de tip **NULL** și regulile de integritate referențiale. Definirea câmpurilor se poate obține fie prin descriere în comandă, fie dintr-un masiv.

Sintaxa comenzii:

```

CREATE TABLE | DBF nume_tabelă_1 [NAME nume_lung] [FREE] (nume_câmp_1 tip_câmp
[mărime[(mărime_câmp[,precizie])] [NULL | NOT NULL] [CHECK expr_L_1 [ERROR
mesaj_eroare_1]] [DEFAULT expr_1] [PRIMARY KEY | UNIQUE] [REFERENCES
nume_tabelă_2 [TAG etich_1_index]] [NOCPTRANS] [nume_câmp_2 ...]
[PRIMARY KEY expr_2 TAG etich_2_index], UNIQUE expr_3 TAG etich_3_index]
[, FOREIGN KEY expr_4 TAG etich_4_index [NODUP] REFERENCES nume_tabelă_3 [TAG
etich_5_index]]
[,CHECK expr_L_2[ERROR mesaj_eroare_2]]) | FROM ARRAY nume_masiv

```

unde:

- *nume_tabelă_1* – specifică numele tablei care va fi creată. Opțiunile TABLE și DBF sunt identice;
- NAME *nume_lung* – specifică un nume lung pentru tabelă. Acest nume poate fi specificat numai dacă este deschisă o bază de date, el fiind stocat (memorat) în cadrul bazei de date. Numele lung poate conține până la 128 caractere și poate fi folosit numai în cadrul bazei de date;

- **FREE** – noua tabelă nu va fi adăugată la baza de date curentă (deschisă);
- **(nume_câmp_1 tip_câmp [mărime[(,mărime_câmp[,precizie])])** – specifică numele câmpului, tipul, mărimea și precizia (numărul de poziții zecimale). O tabelă poate conține până la 255 câmpuri. Dacă unul sau mai multe câmpuri acceptă valori de tip **NULL**, limita se reduce la 254;
- **NULL | NOT NULL** – permite / împiedică introducerea de valori de tip **NULL** în câmp;
- **CHECK expr_L_1** – specifică o regulă de validare pentru câmp. Expresia logică *expr_L_1* poate fi o funcție definită de utilizator. La adăugarea unei înregistrări vide, se verifică regula de validare. Dacă regula de validare nu prevede acceptarea de valori vide în câmp, se generează eroare;
- **ERROR mesaj_eroare_1** – specifică mesajul de eroare pe care sistemul VFP îl afișează la apariția unei erori, generate de clauza **CHECK**, care verifică regula de validare;
- **DEFAULT expr_1** – specifică valoarea implicită pentru câmp. Expresia *expr_1* trebuie să fie de același tip cu tipul câmpului;
- **[PRIMARY KEY** – crează un index primar pentru câmp. Eticheta de index primar are același nume cu cel al câmpului;
- **UNIQUE** – crează un index candidat pentru câmp. Numele etichetei de index candidat este același cu cel al câmpului.
- **REFERENCES nume_tabelă_2 [TAG etich_1_index]** – specifică numele tabelii părinte, la stabilirea unei relații persistente. Dacă se omite clauza *TAG etich_1_index*, relația se stabilește utilizând cheia primară de index a tabelii părinte. Dacă tabela părinte nu are un index de cheie primară, sistemul va genera eroare. Se include clauza *TAG etich_1_index* pentru a stabili o relație bazată pe existența etichetei de index pentru tabela părinte;
- **NOCPTRANS** – previne trecerea la o altă pagină de cod pentru caractere și câmpuri memo. Dacă tabela este convertită la o altă pagină de cod, câmpul pentru care a fost specificată clauza **NOCPTRANS**, nu va fi translatat;
- **[nume_câmp_2 ...]** – următorul câmp din structura tabelii. Are aceleași caracteristici de descriere ca și primul câmp;
- **PRIMARY KEY expr_2 TAG etich_2_index** – specifică indexul primar care va fi creat. Expresia *expr_2* poate specifica orice câmp sau combinație de câmpuri din tabelă. *TAG etich_2_index* specifică numele etichetei de index primar. Deoarece o tabelă nu poate avea decât o singură cheie primară de indexare, nu se poate include această clauză dacă deja există definită o cheie primară;

- **UNIQUE** *expr_3* **TAG** *etich_3_index* – crează un index candidat. Expresia *expr_3* specifică orice câmp sau combinație de câmpuri din tabelă. Clauza **TAG** *etich_3_index* specifică numele etichetei de index pentru eticheta de index candidat, care va fi creat;
- **FOREIGN KEY** *expr_4* **TAG** *etich_4_index* [**NODUP**] – crează un index extern (non-primar) și stabilește o relație cu tabela părinte. *expr_4* specifică expresia indexului extern. Clauza **TAG** *etich_4_index* specifică numele etichetei de index externe. Clauza **NODUP** se include pentru a crea un index candidat extern;
- **REFERENCES** *nume_tabelă_3* [**TAG** *etich_5_index*] – specifică tabela părinte către care se stabilește o relație persistentă. Includerea clauzei **TAG** *etich_5_index* determină stabilirea relației pe baza unei etichete de index a tabelii părinte. Dacă este omisă clauza, relația se stabilește utilizând implicit cheia primară de indexare din tabela părinte;
- **CHECK** *expr_L_2*[**ERROR** *mesaj_eroare_2*] – specifică regula de validare pentru tabelă;
- **FROM ARRAY** *nume_masiv* – specifică numele unui masiv existent care conține numele, tipul, mărimea și precizia pentru fiecare câmp al tabelii. Conținutul masivului poate fi definit cu funcția **AFIELDS()**.

7. Comnda **DELETE** – SQL realizează ștergerea la nivel logic (marcarea pentru ștergere) a înregistrărilor dintr-o tabelă.

Sintaxa comenzii:

```
DELETE FROM [nume_BD!]nume_tabelă [WHERE condiție_filtrare_1 [AND | OR
condiție_filtrare_2...]]
```

unde:

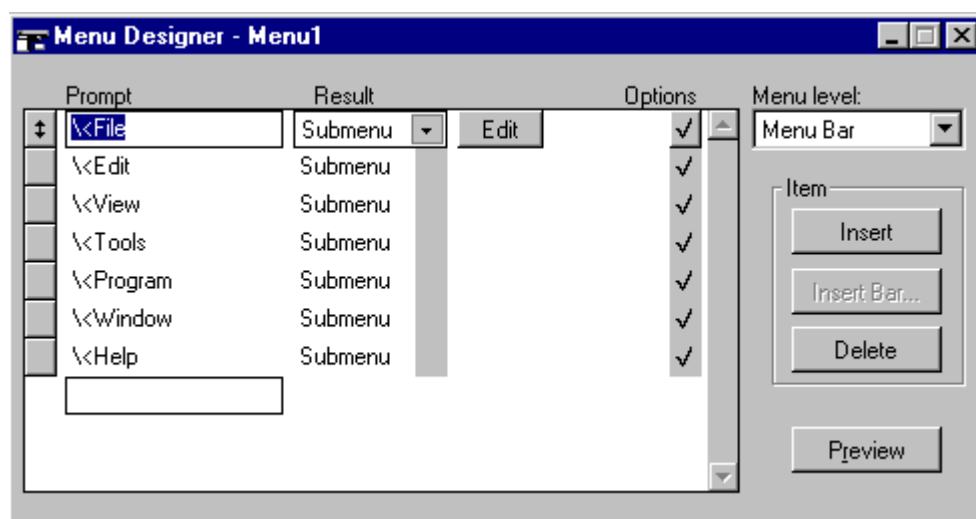
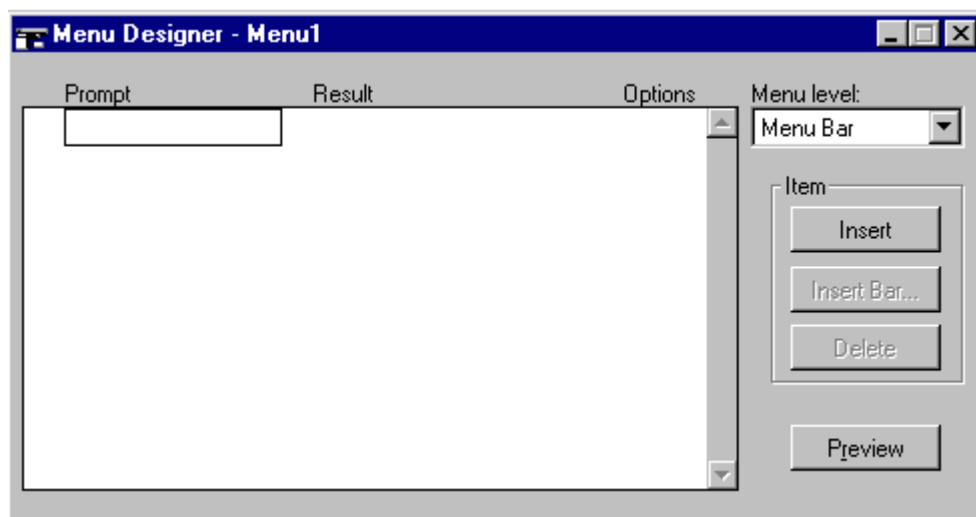
- **FROM** [*nume_BD!*]*nume_tabelă* – specifică numele tabelii în care înregistrările sunt marcate pentru ștergere la nivel logic;
- **WHERE** *condiție_filtrare_1* [AND | OR *condiție_filtrare_2...*] – vor fi marcate pentru ștergere numai anumite înregistrări, cele care îndeplinesc condițiile de filtrare.

Ștergerea fiind la nivel logic, înregistrările vor fi șterse fizic din tabelă doar după utilizarea comenzii **PACK**.

11. Proiectarea meniurilor și a barelor de instrumente

Meniurile (**Menus**) și barele de instrumente (**Toolbars**) furnizează o cale structurată și accesibilă pentru mânăuirea comenzilor aplicației. Prin planificarea și proiectarea judicioasă a meniurilor și barelor de instrumente, se poate crește calitatea aplicațiilor.

O bună parte din activitatea de creare a unui meniu sistem este realizată cu ajutorul proiectantului de meniuri (**Meniu Designer**), în care se crează meniul principal, submeniurile și opțiunile din meniu.



12. Aplicații

12.1. Evidența rezultatelor activității studenților într-o facultate

12.1.1. Formularea și analiza problemei

Enunțul problemei

Tema abordată este *Evidența rezultatelor activității studenților într-o facultate*.

Datele de intrare

Fișa de înscriere;
Cataloagele de note;
Planul de învățământ;
Programele analitice.

Datele de ieșire

Componenta grupelor;
Situția după o sesiune normală;
Situția statistică asupra repartiției notelor la o disciplină;
Situția statistică asupra repartiției mediilor.

Actualizarea datelor

Adăugarea de noi studenți;
Modificarea datelor despre studenți;
Ștergerea unui student;
Adăugarea de note;
Modificarea notelor;
Ștergerea studentului din fișierul de note;
Adugarea de noi discipline;
Modificarea numelui unei discipline;
Ștergerea unei discipline.

Coduri adoptate

Codul județului;

Codurile disciplinelor;

An studii 1-6, an calendaristic 2006-2007.

Interfața cu utilizatorul

Stabilirea meniurilor și a videoformatelor pentru programul monitor și a programelor subordonate.

Proiectarea intrărilor

Cererea de înscriere;

Cataloage;

Formularele cu codurile și denumirile disciplinelor.

Proiectarea ieșirilor

Lista studenților pe grupe;

Situația după sesiune;

Statistica la o disciplină;

Statistica medii;

Listă bursieri;

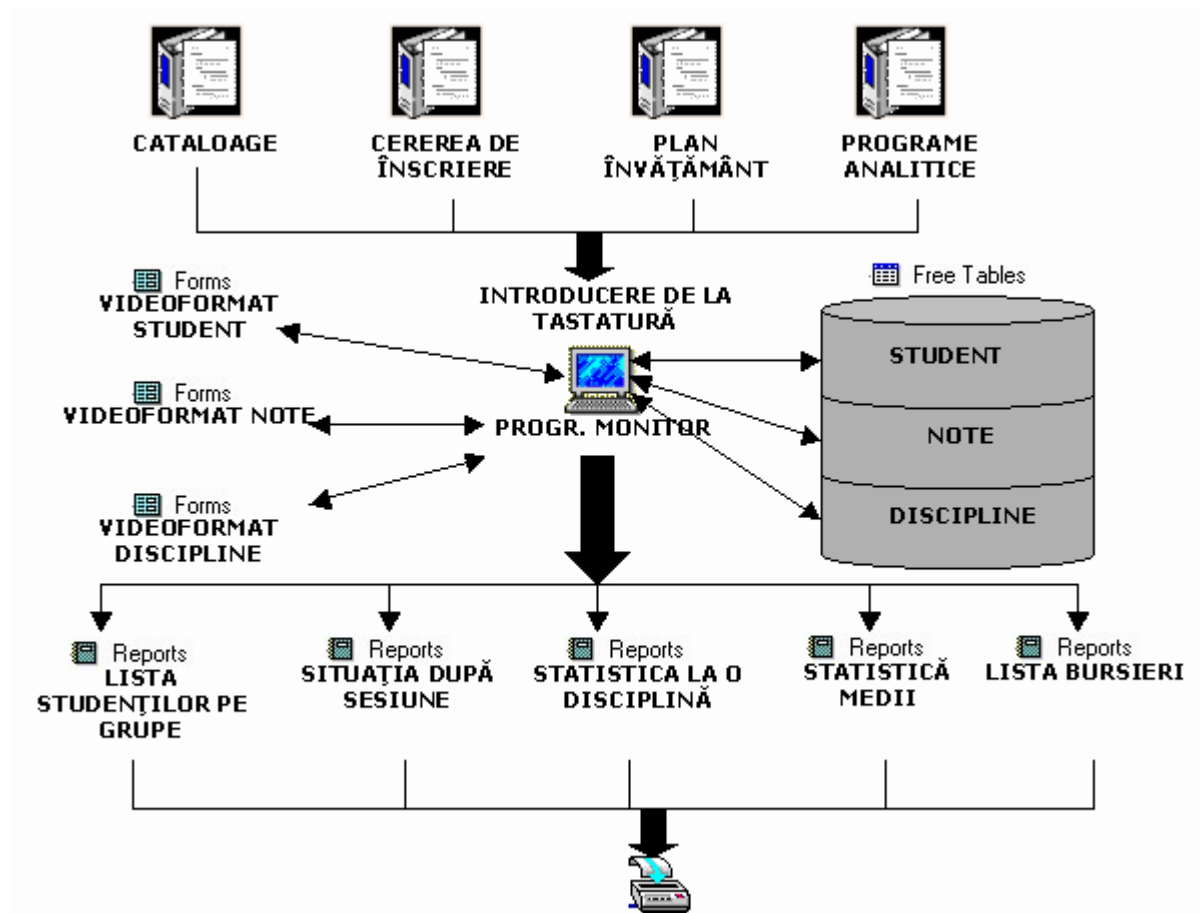
Listă integraliști.

Stabilirea cadrului pentru:

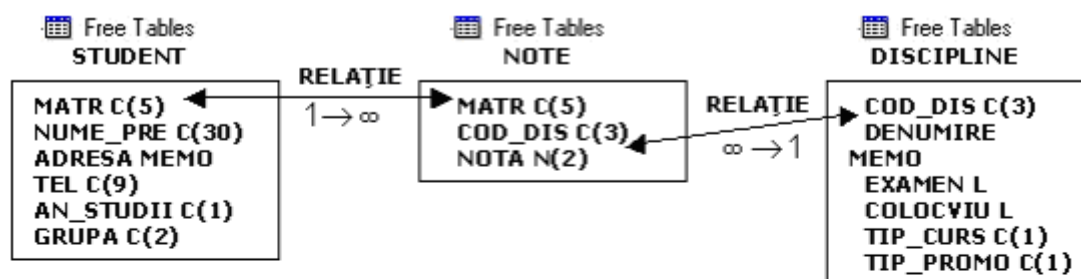
Operațiile de actualizare

Obținerea rapoartelor finale.

Schema generală propusă



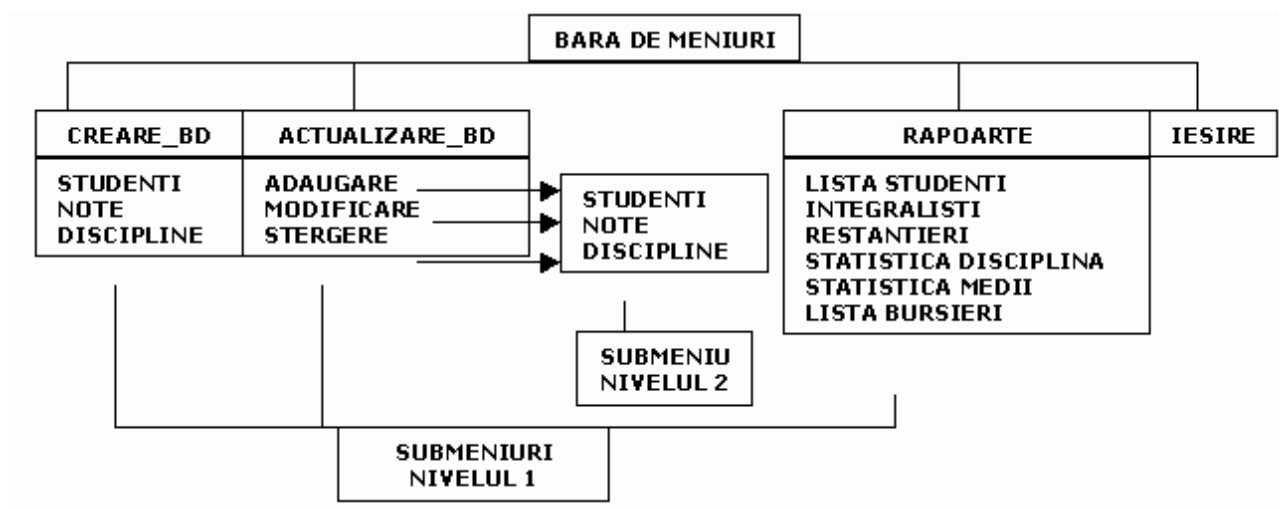
12.1.2. Crearea, actualizarea, modificarea și interogarea tabelelor



CODIFICARI

TIP_CURS { O - OBLIGATORIU
 F - FACULTATIV
 P - OPTIONAL
TIP_PROMO { O - OBLIGATORIU
 N - Nu este obligatoriu pentru promovare

Meniul programului ar putea fi de forma



12.1.3. Crearea structurii unui fișier

În fereastra de comenzi se introduc comenzile de creare a unui fișier și vizualizare:

```
CREATE TABLE STUD1 (MATR C(3), NUME C(30), AN C(1), GRUPA C(2))
```

```
BROWSE
```

Se lansează comenzile **[CTRL] + [Y]** și se populează cu 3-4 înregistrări. Se salvează cu **[CTRL] + [W]**.

Se deschide fereastra **Data Session** și se închide fișierul STUD1 de la butonul **Close**.

Se va crea programul de modificare a structurii fișierului anterior pentru adăugarea unui nou câmp cu numele *mbac* (medie bacalaureat): **File/New/Program/New File** și introduc următoarele linii:

```
SELECT 2
USE STUD1
COPY STRUCTURE EXTENDED TO TEMP
USE TEMP
BROWSE
APPEND BLANK
REPLACE FIELD_NAME WITH 'MBAC'
REPLACE FIELD_TYPE WITH 'N'
REPLACE FIELD_LEN WITH 5
REPLACE FIELD_DEC WITH 2
BROWSE
```

```
USE TEMP
CREATE STUD2 FROM TEMP
USE STUD2
APPEND FROM STUD1
BROWSE
CLOSE DATABASES
```

Se salvează și se rulează programul.

12.1.4. Adresarea prin macrosubstituție

Se va crea un program care va folosi macrosubstituția pentru numele de fișiere STUD1 și STUD2. Se vor introduce următoarele linii de program:

```
PUBLIC FIS C(20)
FOR I=1 TO 2 DO
    GOTO I
    FIS='STUD'+ALLTRIM(STR(I))+'.DBF'
    USE &FIS
    BROWSE
ENDFOR
CLOSE DATABASES
```

Se salvează și se rulează programul.

12.1.5. Crearea unui meniu

Menu Bar va avea următoarele componente:

Creare BD – **Results** – **Submenu** – **Create** – Studenti/Note – **Results** - **Procedure**
Actualizare - **Results** – **Submenu** – **Create** – Studenti/Note – **Results** - **Procedure**
Media – **Mesults**- **Procedure**
Iesire– **Results**- **Procedure**

a) Procedurile pentru crearea BD

Pentru fișierul *Studenti*

```
CREATE TABLE STUD (MATR C(3), NUME C(30), AN C(1), GRUPA C(2))  
CLOSE DATABASES
```

Pentru fișierul *Note*

```
CREATE TABLE NOTE (NOTA1 N(2), NOTA2 N(2), MEDIA N(5,2))  
CLOSE DATABASES
```

b) Procedurile de actualizare

Pentru fișierul *Studenti*

```
USE STUD  
BROWSE  
CLOSE DATABASES
```

Pentru fișierul *Note*

```
USE NOTE  
BROWSE  
CLOSE DATABASES
```

În fiecare fereastră **Browse** cu [CTRL] + [Y] se populează fișierele cu 3-4 înregistrări. La fișierul *Note* nu se completează câmpul *media*.

c) Procedura de calcul a mediei

```
USE NOTE  
FOR I=1 TO RECCOUNT()  
    GOTO I  
    REPLACE MEDIA WITH (NOTA1+NOTA2)/2  
ENDFOR  
BROWSE
```

CLOSE DATABASES

d) Procedura de ieșire

CLEAR EVENTS

CLOSE DATABASES

SET SYSMENU TO DEFAULT

12.2. Gestionarea unei magazii

Să se realizeze o aplicație în Visual Fox Pro 6 prin care să se gestioneze intrările de materiale, furnizorii și materialele într-o magazie a unei secții de producție.

12.2.1. Proiectarea și realizarea aplicației

Aplicația va cuprinde un modul pentru definirea bazei de date, un modul pentru actualizare (adăugare, modificare, ștergere) și un modul de exploatare a bazei de date prin rapoarte.

Sunt prevăzute trei tabele independente care au următoarea structură:

Fișierul <i>materiale.dbf</i>		Fișierul <i>furnizori.dbf</i>		Fișierul <i>intrari.dbf</i>	
Denumire	Tip	Denumire	Tip	Denumire	Tip
Denumire	C(40)	Denumire	C(40)	Cod_mat	N(5)
Cod_mat	N(5)	Cod_furn	N(4)	Cod_furn	N(4)
Um	C(6)	Adresa	M	Cant	N(6)
		Tel	C(10)	Pu	N(8,2)
				Data_in	D

Existența câmpurilor comune din fișierul *intrari.dbf* cu cele ale celorlalte fișiere oferă posibilitatea ca în etapa de proiectare a rapoartelor să se poată stabili relații între tabele pentru a extrage date corelate din toate sursele.

Se vor realiza două rapoarte:

- pentru obținerea unei liste de intrări de materiale sortată ascendent după data calendaristică cu câteva comenzi SQL de manipulare a tabelor;

- pentru obținerea unei liste cu furnizori și materiale folosind facilitatea de grupare a modului de rapoarte. Se vor utiliza date extrase din toate cele trei fișiere (stabilind relații între acestea), calculându-se valoarea plătită fiecărui furnizor.

Aplicația conține proceduri pentru întreținerea și exploatarea bazei de date fiind construită cu o interfață accesibilă și logică. Structura proiectului este prezentată în Figura 1.

Se va utiliza programare vizuală, orientată spre obiecte și procedurală în realizarea aplicației care rezolvă problema formulată mai sus.

În programul principal *init_var* sunt declarate și inițializate variabilele globale, se deschid sau se crează fișierele utilizate și se lansează meniul aplicației.

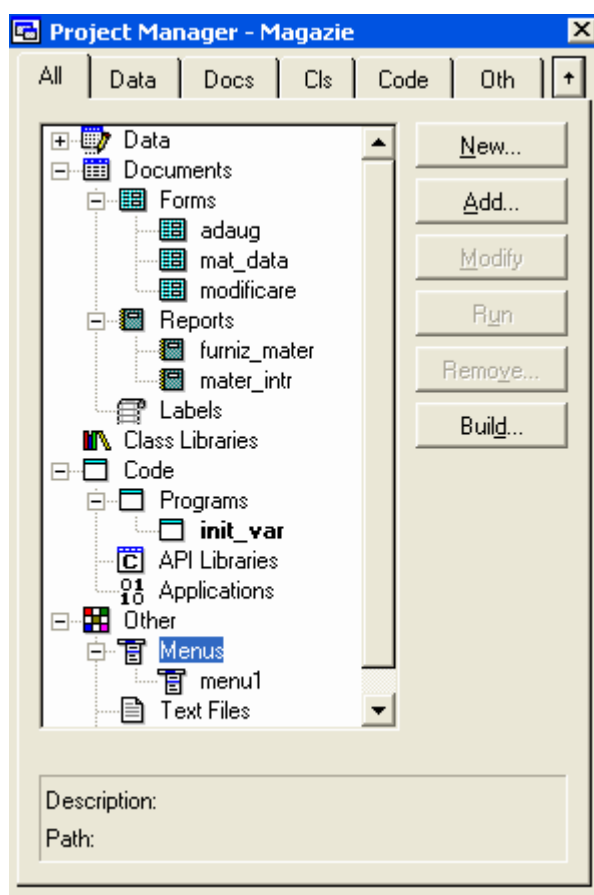


Fig. 1

Programul principal – *Init_var* – din secțiunea **Programs**

```
PUBLIC CODF, DEN, ADR, TL, CODM, DENM, PUN, DATAIN, CANTIT
SET DATE TO BRITISHCODF=0
```

* INITIALIZARI

```
DEN=' '
ADR=' '
TL=' '
CODM=0
DENM=' '
PUN=0
DATAIN={ / / }
CANTIT=0
CLOSE DATABASES
IF EMPTY(SYS(2000,'MATERIALE.DBF'))
    CREATE TABLE MATERIALE (COD_MAT N(5),DENUMIRE C(40),UM C(6))
ELSE
    USE MATERIALE
ENDIF

INDEX ON COD_MAT TAG COD_MAT ADDITIVE
IF EMPTY(SYS(2000,'INTRARI.DBF'))
    CREATE TABLE INTRARI (COD_MAT N(5), COD_FURN N(4), CANT N(6), PU N(8,2),
DATA_IN D)
ELSE
    USE INTRARI
ENDIF

IF EMPTY(SYS(2000,'FURNIZORI.DBF'))
    CREATE TABLE FURNIZORI (COD_FURN N(4), DENUMIRE C(40), ADRESA M, TEL
C(10))
ELSE
    USE FURNIZORI
ENDIF
INDEX ON COD_FURN TAG COD_FURN ADDITIVE

CLOSE DATABASE
SELECT 2
USE MATERIALE
```

```

SELECT 3
USE FURNIZORI
SELECT 4
USE INTRARI
DO MENU1.MPR
READ EVENTS

```

În continuare se definește bara de meniuri. În Figura 2 sunt prezentate *Meniul principal* și submeniul *Vizualizare BD*.

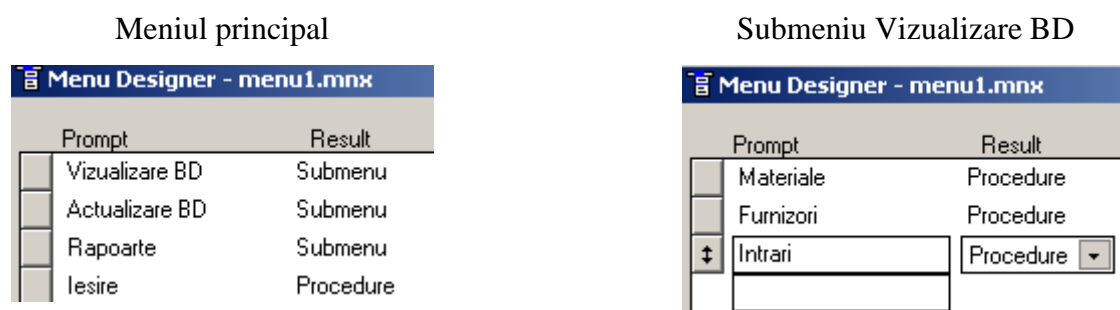


Fig. 2

Crearea submeniului Vizualizare BD

*** Vizualizarea tabelii *Materiale***

```

SELECT 2
BROWSE FIELDS COD_MAT, DENUMIRE, PU, DATA_IN NOEDIT

```

*** Vizualizarea tabelii *Furnizori***

```

SELECT 3
BROWSE FIELDS COD_FURN, DENUMIRE NOEDIT

```

*** Vizualizarea tabelii *Intrari***

```

SELECT 4
BROWSE NOEDIT

```

Crearea submeniu *Actualizare BD*

Submeniul este prezentat în Figura 3.

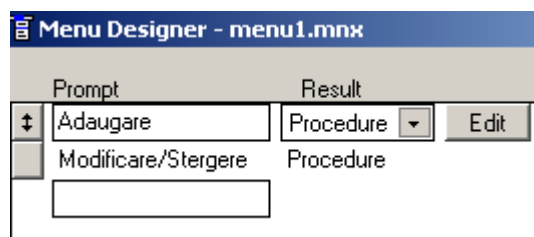


Fig. 3

Opțiunea *Adaugare*

DO FORM ADAUG

Opțiunea *Modificare/Stergere*

DO FORM MODIFICARE

* Crearea submeniu *Rapoarte*

Submeniul este prezentat în Figura 4:

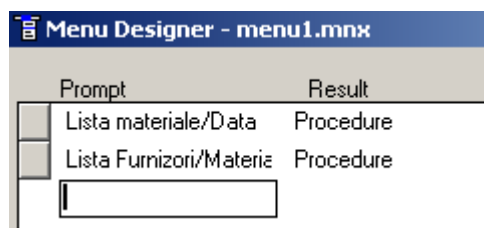


Fig. 4

Opțiunea *Lista materiale/Data*

DO FORM MAT_DATA

Opțiunea *Lista Furnizori/Materiale*

Se declară următoarea procedură:

SELECT 4

```
SET ORDER TO TAG COD_MAT OF MATERIALE.CDX IN MATERIALE
SET RELATION TO COD_MAT INTO MATERIALE ADDITIVE
SET ORDER TO TAG COD_FURN OF FURNIZORI.CDX IN FURNIZORI
SET RELATION TO COD_FURN INTO FURNIZORI ADDITIVE
REPORT FORM FURNIZ_MATER PREVIEW
SET RELATION OFF INTO MATERIALE
SET RELATION OFF INTO FURNIZORI
```

care stabilește o relație între înregistrările fișierului *materiale.dbf* și *intrari.dbf* prin intermediul câmpului comun *cod_mat* și pe de altă parte o relație între înregistrările fișierului *furnizori.dbf* și *intrari.dbf* prin intermediul câmpului comun *cod_furn*.

Pentru stabilirea relațiilor între fișiere trebuie ca în prealabil fișierele *copil* să fie indexate după câmpul de legătură (*cod_mat* în fișierul *materiale*, *cod_furn* în fișierul *furnizori*), etapă realizată în programul principal *init_var*.

După vizualizarea raportului *furniz_mater* cele două relații sunt anulate.

Opțiunea *Iesire*

CLEAR EVENTS

CLOSE DATABASE

SET SYSMENU TO DEFAULT

* **Videoformat** *Adaug*

Aceast videoformat are numele intern **Form1**, numele extern *Adaug.scx*, iar pentru proprietatea **Caption** se alege textul *Adaugare*, Figurile 5a și 5b.

Butonul *Iesire* este primul strat și se aplica deasupra obiectului **PageFrame1** (de tip container) având proprietatea **PageCount** =3 (pagini). Pe butonul *Iesire* la evenimentul **Click** se asociaza comanda *THISFORM.RELEASE*.

Se face click cu butonul din dreapta al mouse-ului și se alege opțiunea **Edit** pentru a accesarea elementelor containerului, respectiv cele trei pagini: *Furnizori*, *Materiale*, *Intrari*. Numele extern și grosimea de caracter a paginilor se stabilește utilizând proprietățile **Caption** și **FontBold= True**.

Adresarea paginilor se realizează făcând click cu butonul stâng al mouse-ului pe fiecare pagină.

Numele intern poate fi schimbat, dar vor fi păstrate cele implicite (**Page1**, **Page2**, **Page3**), respectiv adresarea paginii *Furnizori* în procedurile atașate va fi una ierarhică, prin calificare:

THISFORM.PAGEFRAME1.PAGE1.*proprietate*

Tabela *Furnizori*

The screenshot shows a form titled 'Adaugare' with three tabs: 'Furnizori', 'Materiale', and 'Intrari'. The 'Furnizori' tab is active. It contains four text input fields: 'Cod furnizor' (Text1), 'Denumire' (Text2), 'Adresa' (Text3), and 'Telefon' (Text4). There are two buttons at the bottom right: 'Validare' and 'Iesire'.

Fig. 5a

The screenshot shows the 'Properties - adaug.scx' window. The 'Form1' object is selected. The 'Caption' property is highlighted and set to 'Adaugare'. Other properties like BaseClass, BorderStyle, Box, BufferMode, Circle, Class, and ClassLibrary are also visible.

Fig.5b

Urmează stabilirea obiectelor pentru pagina **Page1** (cu proprietatea **caption**=*'Furnizori'*). Sunt patru regiuni de tip text, notate de la **Text1** la **Text4**, un buton de comandă (**Command1**) și patru etichete, de la **Label1** la **Label4**. Ordinea lor se stabilește din meniul **View**, submeniul **TabOrder**. Dacă se dorește trecerea de la vizualizarea interactivă a ordinii obiectelor la vizualizarea de tip listă, se selectează meniul **Tools**, submeniul **Options**, submeniul **Forms**, opțiunea **Tab Ordering** și se alege **By list**.

În continuare se stabilesc câteva proprietăți referitoare la obiectele alese:

- pentru obiectul **Page1** (*Furnizori*) se accesează codul procedurilor atașate diferitelor evenimente utilizând butonul dreapta al mouse-ului și alegând din meniu opțiunea **Code**. Se selectează evenimentul **Activate** și se adăugă următoarele linii de cod în editorul deschis:

* SE SELECTEZĂ ZONA DE LUCRU 3 (*FURNIZORI.DBF*)

SELECT 3

* SE TRIMITE FOCUSUL (PROMPTERUL) PE *TEXT1*

THISFORM.PAGEFRAME1.PAGE1.TEXT1.SETFOCUS

CODF=0

DEN ="

ADR=' '

TL="

* ACTUALIZAREA REGIUNILOR DE EDITARE CU NOILE VALORI

THISFORM.PAGEFRAME1.PAGE1.REFRESH

- pentru obiectele de tip text la proprietăți, secțiunea **Data**, se declară variabilele în **ControlSource**, care vor prelua datele ce se vor scrie în fișier: **Text1-** *codf*, **Text2-** *den*, **Text3-** *adr*, **Text4-** *tl*.
- Etichetele vor avea trecute la proprietatea **Caption** denumirile explicite a variabilelor utilizate: *Cod furnizor* (**Label1**), *Denumire* (**Label2**), *Adresa* (**Label3**), *Telefon* (**Label4**). Proprietatea **FontBold** se setează **True**.
- Butonul de validare (nume intern **Command1**, proprietatea **Caption=Validare**, **FontBold=True**) va avea atașate următoare linii de cod pentru evenimentul **Click**:

LOCATE FOR CODF=COD_FURN

IF FOUND(3) THEN

 WAIT WINDOW 'COD FURNIZOR DUPLICAT !'

ELSE

 LOCATE FOR UPPER(ALLTRIM(DEN))=UPPER(ALLTRIM(DENUMIRE))

 IF FOUND(3) THEN

 WAIT WINDOW 'DENUMIRE DUPLICAT !'

 ELSE

 APPEND BLANK

 REPLACE COD_FURN WITH CODF, DENUMIRE WITH DEN, ADRESA WITH

ADR, TEL WITH TL

 ENDIF

ENDIF

CODF=0

DEN=' '

ADR=' '

TL=' '

THISFORM.PAGEFRAME1.PAGE1.REFRESH

THISFORM.PAGEFRAME1.PAGE1.TEXT1.SETFOCUS

Secvența de cod realizează două validări, una pentru codul de furnizor și una pentru denumirea firmei, verificând dacă mai există o înregistrare similară. Verificarea se face pentru a păstra caracterul de unicitate al codului și denumirii. În caz ca se găsește un duplicat se afișează un mesaj specific. Dacă codul și denumirea sunt valide din punct de vedere al unicității, se trece la scrierea în fișierul de *furnizor.dbf* a noilor informații. Se reinițializează variabilele, se actualizează câmpurile afișate și se trimite focusul (prompter-ul) pe obiectul **Text1**.

Tabela Materiale

Fig. 6

Se stabilesc obiectele pentru pagina **Page2** (proprietatea **Caption**=*'Materiale'*): **Text1**, **Text2**, **Text3**, **List1** (listă), un buton de comandă (**Command1**) și trei etichete **Label1**, **Label2**, **Label3**, (Figura 6). Ordinea obiectelor se stabilește din meniul **View**, submeniul **TabOrder**.

În continuare se stabilesc câteva proprietăți referitoare la obiectele alese:

- pentru obiectul **Page2** (*Materiale*) se accesează codul procedurilor atașate diferitelor evenimente utilizând butonul dreapta al mouse-ului și alegând din meniu opțiunea **Code**. Se selectează evenimentul **Activate** și se adăugă următoarele linii de cod în editorul deschis:

```
SELECT 2
CODM=0
DENM=' '
THISFORM.PAGEFRAME1.PAGE2.REFRESH
THISFORM.PAGEFRAME1.PAGE2.TEXT1.SETFOCUS
```


- pentru obiectul **Page2** (*Materiale*) se selectează procedura **Init** la care se atașează următoarele linii de cod (se inițializează și introduc valorile într-un vector care va fi utilizat ulterior pentru obiectul **list1** referitor la unitatea de măsură)

```
PUBLIC UMAS(6)
```

```
UMAS(1)='BUC'
```

```
UMAS(2)='KG'
```

```
UMAS(3)='TO'
```

```
UMAS(4)='MC'
```

```
UMAS(5)='MP'
```

```
UMAS(6)='ML'
```

- pentru obiectul **Page2** se selectează procedura **Activate** la care se atașează următorul cod

```
SELECT 2
```

```
CODM=0
```

```
DENM=' '
```

```
THISFORM.PAGEFRAME1.PAGE2.REFRESH
```

```
THISFORM.PAGEFRAME1.PAGE2.TEXT1.SETFOCUS
```

- pentru obiectul **text1** se specifică în fereastra de proprietăți, secțiunea **Data**, proprietatea **ControlSource**, variabila *codm*.
- pentru obiectul **Text2** se specifică la proprietatea **ControlSource** variabila *denm*.
- pentru obiectul **text3** se specifică în fereastra de proprietăți: **ReadOnly=.T.** și **FontBold=.T.** care va fi utilizat pentru verificarea alegerii unității de măsură din obiectul *list1*.
- pentru obiectul **List1** se alege din fereastra de proprietăți **RowSource**, unde se declară variabila *umas*, iar la **RowSourceType** se alege **Array** și se atașă următorul cod pe evenimentul **Click**:

```
WITH THISFORM.PAGEFRAME1.PAGE2
```

```
TEXT3.VALUE=.LIST1.VALUE
```

```
TEXT3.REFRESH
```

```
COMMAND1.SETFOCUS
```

ENDWITH

prin care se preia valoarea selectată în **List1** și se afișează în **Text3**. S-a folosit combinația **With...EndWith** pentru a adresa proprietățile obiectelor **List1**, **Text3** și **Command1** din **Page2** într-o formă mai scurtă (calificare globală).

- pentru obiectul **command1** din lista de proprietăți se stabilesc valorile pentru **Caption=validare** și **Fontbold= True**. Din lista de evenimente se selectează **Click Event** pentru care se atașează următoarele linii de cod:

```
LOCATE FOR CODM=COD_MAT
```

```
IF FOUND(2) THEN
```

```
    WAIT WINDOW 'COD MATERIAL DUPLICAT !'
```

```
ELSE
```

```
    LOCATE FOR UPPER(ALLTRIM(DENM))=UPPER(ALLTRIM(DENUMIRE))
```

```
    IF FOUND(2) THEN
```

```
        WAIT WINDOW 'DENUMIRE DUPLICAT !'
```

```
    ELSE
```

```
        APPEND BLANK
```

```
        REPLACE COD_MAT WITH CODM,DENUMIRE WITH DENM,;
```

```
        UM WITH THISFORM.PAGEFRAME1.PAGE2.LIST1.VALUE
```

```
    ENDIF
```

```
ENDIF
```

```
CODM=0
```

```
DENM=''
```

```
THISFORM.PAGEFRAME1.PAGE2.REFRESH
```

```
THISFORM.PAGEFRAME1.PAGE2.TEXT1.SETFOCUS
```

Secvența de program realizează adăugarea în fișierul de materiale a valorilor introduse, care se face în urma validării codului și denumirii materialului. Se verifică existența duplicatelor, dacă acestea există vor fi semnalate prin mesaje specifice și adăugarea nu se efectuează.

Tabela *Intrari*

Stabilirea obiectelor pentru ultima pagină **Page3** (proprietatea **Caption='Intrari'**, **FontBold=True**) din setul de pagini **PageFrame1**. Se vor insera următoarele obiecte: **List1**, **List2**, **Text1**, **Text2**, **Text3**, **Text4**, **Text5**, **Command1** și șapte etichete, de la **Label1** la **Label7**, Figura 7.

Fig. 7

Se stabilesc proprietățile pentru obiectele declarate în videoformat:

- pentru obiectul **Page3** (*Intrari*) se accesează codul procedurilor atașate diferitelor evenimente utilizând butonul dreapta al mouse-ului și alegând din meniu opțiunea **Code**. Se selectează procedura **Activate** și se adaugă următoarele linii de cod în editorul deschis

```

SELECT 2
GO TOP
SELECT 3
GO TOP
SELECT 4
PUN=0
CANTIT=0
CODM=MATERIALE.COD_MAT
CODF=FURNIZORI.COD_FURN
DATAIN={ / / }
THISFORM.PAGEFRAME1.PAGE3.LIST1.SETFOCUS

```

THISFORM.PAGEFRAME1.PAGE3.REFRESH

Procedura selectează tabelele care vor fi folosite, inițializează variabilele pentru cantitate, preț unitar, cod material, cod furnizor și data intrării. Se trimite focusul pe obiectul **List1** și se actualizează pagina cu noile valori.

- pentru obiectul **List1** se alege din fereastra de proprietăți **RowSource**, unde se declară *materiale*, **RowSourceType**, se alege **fields**, **ControlSource**, se declară *denumire* (câmpul cu același nume ca în fișierul *materiale.dbf*). Pentru evenimentul **Click** se asociază următoarele linii de cod

SELECT 2

CODM=COD_MAT

THISFORM.PAGEFRAME1.PAGE3.TEXT1.REFRESH

- pentru obiectul **List2** se alege din fereastra de proprietăți **RowSource**, aici se declară *furnizori*, **RowSourceType**, se alege **Fields**, **ControlSource** se declară *denumire* (câmpul cu același nume al fișierului *furnizori.dbf*). Evenimentul **Click** are atașate următoarele linii de cod

SELECT 3

CODF=COD_FURN

THISFORM.PAGEFRAME1.PAGE3.TEXT2.REFRESH

THISFORM.PAGEFRAME1.PAGE3.TEXT3.SETFOCUS

- pentru obiectul *text1* se introduce în fereastra de proprietăți la **ControlSource** valoarea *codm* (variabila în care se preia din lista **List1** codul materialului), **FontBold= True** și **ReadOnly True**.
- pentru obiectul **Text2** se introduce în fereastra de proprietăți la **ControlSource** valoarea *codf* (variabila în care se preia din lista **List2** codul furnizorului), **FontBold= True** și **ReadOnly True**.
- pentru obiectul **Text3** se introduce în fereastra de proprietăți la **ControlSource** valoarea *cantit* (variabila în care se preia cantitatea);

- pentru obiectul *text4* se introduce în fereastra de proprietăți la **ControlSource** valoarea *pun* (variabila în care se preia prețul unitar);
- pentru obiectul **Text5** se introduce în fereastra de proprietăți la **ControlSource** valoarea *datain* (variabila în care se preia data intrării);
- pentru obiectul **Command1** la proprietatea **Caption** se introduce *validare*, **FontBold=True**, iar la evenimentul **Click** se introduce secvența de cod

SELECT 4

APPEND BLANK

REPLACE COD_FURN WITH CODF, COD_MAT WITH CODM, CANT WITH CANTIT, PU
WITH PUN,;

DATA_IN WITH DATAIN

PUN=0

CANTIT=0

SELECT 2

GO TOP

CODM=MATERIALE.COD_MAT

SELECT 3

GO TOP

CODF=FURNIZORI.COD_FURN

DATAIN={ / / }

THISFORM.PAGEFRAME1.PAGE3.REFRESH

THISFORM.PAGEFRAME1.PAGE3.LIST1.SETFOCUS

Secvența de program realizează selectarea fișierului *intrari.dbf* și adăugarea conținutului variabilelor atașate obiectelor din videoformat la câmpurile fișierului. Se reinițializează variabilele, se actualizează câmpurile cu noile valori și se trimite focus-ul pe obiectul **List1**.

Etichetele vor avea fixate valorile pentru următoarele proprietăți:

Obiect	Caption	FontBold	ForeColor
<i>Label1</i>	Cod material	true	0,0,0
<i>Label2</i>	Cod furnizor	true	0,0,0
<i>Label3</i>	Cantitate	true	0,0,0
<i>Label4</i>	Pret unitar	true	0,0,0
<i>Label5</i>	Data intrarii	true	0,0,0
<i>Label6</i>	Denumire material	true	0,64,0
<i>Label7</i>	Denumire furnizor	true	255,0,0

Videoformatul *Modificare* pentru tabela *Furnizori*

Fig. 8

Acest videoformat are numele intern **Form1**, numele extern este *modificare.scx*, iar pentru proprietatea **Caption** se alege textul *Modificare*, Figura 8.

Butonul *Iesire* este primul strat și se aplica deasupra obiectului **PageFrame1** (de tip container având proprietatea **PageCount** =3); pe butonul *Iesire* la evenimentul **Click** se asociază comanda **THISFORM.RELEASE**.

Se face click cu butonul din dreapta al mouse-ului și se alege opțiunea *Edit* pentru a accesa elementele containerului, respectiv cele trei pagini: *Furnizori*, *Materiale*, *Intrari*. Numele extern și grosimea de caracter a paginilor se stabilește utilizând proprietățile **Caption** și **FontBold= True**.

Adresarea paginilor se realizează făcând click cu butonul stâng al mouse-ului pe fiecare pagină. La fel ca și la videoformatul precedent numele intern al paginii poate fi schimbat, dar vor fi păstrate cele implicite (**Page1**, **Page2**, **Page3**).

Pentru stabilirea obiectelor pentru prima pagină **Page1** (proprietatea **caption**= '*Furnizori* ') din setul de pagini **PageFrame1** se procedează astfel: se inserează următoarele obiecte **Text1**, **Text2**, **Text3**, **Text4**, **Text5**, cinci butoane de comandă (**Command1..Command5**) și cinci etichete (**Label1..Label5**).

Stabilirea proprietăților pentru obiectele alese se realizează astfel:

- pentru obiectul **Page1** (*Furnizori*) se accesează codul procedurilor atașate diferitelor evenimente utilizând butonul dreapta al mouse-ului și alegând din meniu opțiunea **Code**. Se selectează evenimentul **Activate** și se adăugă următoarele linii de cod în editorul deschis:

```
SELECT 3
GO TOP
DENF=' '
THISFORM.PAGEFRAME1.PAGE1.REFRESH
THISFORM.PAGEFRAME1.PAGE1.TEXT1.SETFOCUS
```

- pentru obiectele de tip text la proprietăți, secțiunea **Data**, se aleg pentru **ControlSource** numele câmpurilor din fișierul *furnizori.dbf*: **Text1**- *cod_furn*, **Text2**- *denumire*, **Text3**- *adresa*, **Text4**- *tel*, iar pentru **Text5**- *denf*, care este variabila de preluare a șirului de caractere utilizat pentru căutarea în fișier a denumirii furnizorului.
- Obiectul **Text5** va avea atașat pe procedura **Valid** următoarele linii de cod pentru căutarea șirului de caractere preluat în variabila *codf*

```
LOCATE FOR UPPER(LEFT(DENF, LEN(ALLTRIM(DENF))))=UPPER(LEFT(DENUMIRE,
LEN(ALLTRIM(DENF))))
IF FOUND(3) THEN
    THISFORM.PAGEFRAME1.PAGE1.REFRESH
ELSE
    MESSAGEBOX('NU EXISTA !',64)
ENDIF
```

- Etichetele vor avea trecute la proprietatea **Caption** denumirile explicite ale variabilelor utilizate: *Cod furnizor* (**Label1**), *Denumire* (**Label2**), *Adresa* (**Label3**), *Telefon* (**Label4**) și *Cauta furnizor* (**Label5**). Proprietatea **FontBold** se setează pe valoarea **True**.
- Butonul de ștergere (nume intern **Command1**, proprietatea **Caption**=*'Stergere'*, **FontBold**=**True**, **ForeColor**=255,0,0) va avea atașate următoare linii de cod pentru evenimentul **Click**

* VARIABILA CODF CAPĂTA VALOAREA CURENTĂ A CODULUI FURNIZORULUI

```
CODF=COD_FURN
* STERGE LOGIC ÎNREGISTRAREA CURENTĂ
DELETE
* STERGE FIZIC ÎNREGISTRAREA CURENTĂ
PACK
* SELECȚIA FIȘIERULUI DIN ZONA 4(INTRARI.DBF)
SELECT 4
* CAUTĂ ÎNREGISTRĂRILE CU CODF=COD_FURN
LOCATE FOR CODF=COD_FURN
* ATÂTA VREME CÂT GASEȘTI CODF
DO WHILE FOUND(4)
* ȘTERGE LOGIC ÎNREGISTRAREA
    DELETE
    CONTINUE
* SFÂRȘIT CICLU
ENDDO
* ȘTERGE FIZIC ÎNREGISTRAREA
PACK
* SELECȚIE FIȘIER DIN ZONA 3 (FURNIZORI.DBF)
SELECT 3
* ACTUALIZARE CÂMPURI PAGINA 1
THISFORM.PAGEFRAME1.PAGE1.REFRESH
* FOCUSUL(CONTROLUL) ESTE TRIMIS LA TEXT1.
THISFORM.PAGEFRAME1.PAGE1.TEXT1.SETFOCUS
```

- urmează grupul de patru butoane (**Command2..Command5**), utilizate pentru navigarea în cadrul fișierului *furnizori.dbf*, cu următoarele valori pentru declararea numelui icon-lui care va fi atașat (proprietatea **Picture**)

Obiect	Picture
<i>Command2</i>	c:\proiect_ex\top.bmp
<i>Command3</i>	c:\proiect_ex\prev.bmp
<i>Command4</i>	c:\proiect_ex\next.bmp
<i>Command5</i>	c:\proiect_ex\end.bmp

Fiecare buton are atașată câte o procedură pe evenimentul **Click** care execută deplasarea.

- Butonul **Command2** – deplasare pe prima înregistrare

GO TOP

THISFORM.PAGEFRAME1.PAGE1.REFRESH

- Butonul **Command3** – deplasare o înregistrare înapoi

SKIP -1

IF BOF(3)

GO TOP

MESSAGEBOX('INCEPUT FISIER !',64)

ENDIF

THISFORM.PAGEFRAME1.PAGE1.REFRESH

- Butonul **Command4** – deplasare o înregistrare înainte

SKIP 1

IF EOF(3)

GO BOTTOM

MESSAGEBOX('SFÂRSIT FISIER !',64)

ENDIF

THISFORM.PAGEFRAME1.PAGE1.REFRESH

Butonul **Command5** – deplasare la ultima înregistrare

GO BOTTOM

THISFORM.PAGEFRAME1.PAGE1.REFRESH

Videoformatul *Modificare* pentru tabela *Materiale*

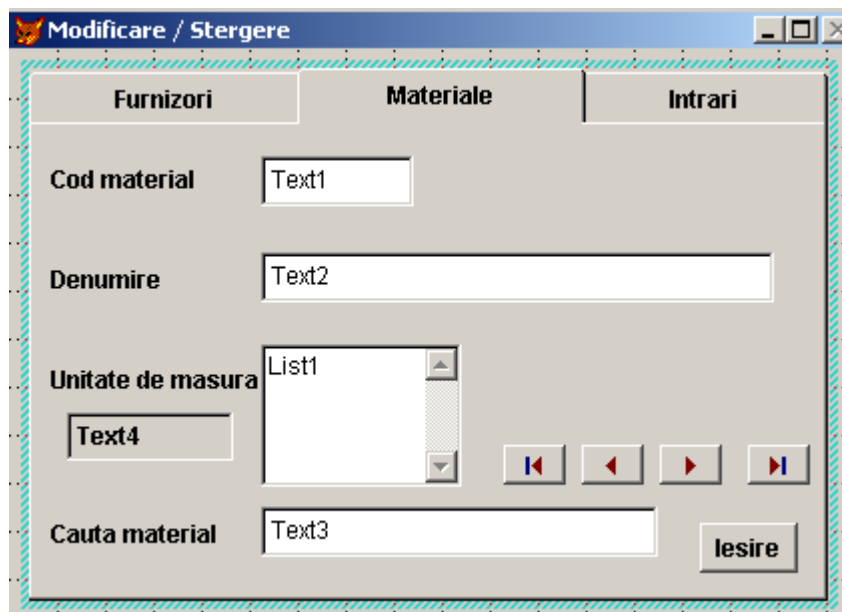


Fig. 9

Stabilirea obiectelor pentru pagina **Page2** (proprietatea **Caption**=*'Materiale'*, **FontBold**=**True**): **Text1**, **Text2**, **Text3**, **Text4**, **List1** (listă), patru butoane de comandă (**Command2..Command5**) și patru etichete **Label1**, **Label2**, **Label3**, **Label4**, Figura 9. Acțiunile atașate se realizează după cum urmează.

- pentru obiectul **Page2** (*Materiale*) se accesează codul procedurilor atașate diferitelor evenimente utilizând butonul dreapta al mouse-ului și alegând din meniu opțiunea **Code**. Se selectează procedura **Init** și se adaugă următoarele linii de cod în editorul deschis

```
PUBLIC UMAS(6)
UMAS(1)='BUC'
UMAS(2)='KG'
UMAS(3)='TO'
UMAS(4)='MC'
UMAS(5)='MP'
UMAS(6)='ML'
```

- pentru obiectul **Page2** se selectează procedura **Activate** la care se atașează următorul cod

```
SELECT 2
GO TOP
```

DENM=' '

THISFORM.PAGEFRAME1.PAGE2.REFRESH

THISFORM.PAGEFRAME1.PAGE2.TEXT1.SETFOCUS

- pentru obiectele de tip text se stabilesc următoarele valori pentru proprietățile **ControlSource** și **ReadOnly**:

Obiect	ControlSource	ReadOnly
Text1	Cod_mat	.F.
Text2	Denumire	.F.
Text3	denm	.F.
Text4	Um	.T.

Proprietatea **ReadOnly=.F.** este implicită.

- Obiectul **Text3** va avea atașat următorul cod la procedura **Valid**

```
LOCATE FOR UPPER(LEFT(DENM,LEN(ALLTRIM(DENM))))=;
UPPER(LEFT(DENUMIRE,LEN(ALLTRIM(DENM))))
IF FOUND(2) THEN
    THISFORM.PAGEFRAME1.PAGE2.LIST1.VALUE=UM
    THISFORM.PAGEFRAME1.PAGE2.REFRESH
ELSE
    MESSAGEBOX('NU EXISTA !',64)
ENDIF
```

- Obiectul **Text4** se utilizează pentru a afișa unitatea de măsură care este declarată în câmpul *um* al fișierului *materiale.dbf* și pentru a afișa noua modificare în cazul operării în **List1**.

Secvența de cod realizează căutarea materialului după șirul de caractere preluat în variabila *denm*, dacă găsește o înregistrare cu valoarea câmpului *denumire=denm* se afișează datele, dacă nu, apare un mesaj specific.

- pentru obiectul **List1** vom alege din fereastra de proprietăți **RowSource** unde declarăm variabila *umas* iar la **RowSourceType** alegem **Aarray** și atașăm următorul cod pe evenimentul **Click**:

WITH THISFORM.PAGEFRAME1.PAGE2

TEXT4.VALUE=.LIST1.VALUE

UM=.TEXT4.VALUE

TEXT4.REFRESH

TEXT3.SETFOCUS

ENDWITH

Secvența realizează actualizarea valorii stocate în obiectului **Text4** cu valoarea selectată din **List1** în cazul unei modificări, noua valoare se scrie în câmpul *um* din fișierul *intrari.dbf* și trimiterea controlului la următorul obiect **Text3** (căutare material).

- pentru cele patru etichete se stabilesc valorile pentru proprietățile **Caption** și **FontBold**

Obiect	Caption	FontBold
Label1	Cod material	true
Label2	Denumire	true
Label3	Unitate de masura	true
Label4	Cauta material	true

- cele patru butoane de comandă (**Command2..Command5**) se utilizează pentru navigarea în fișierul *materiale.dbf* au următoarele valori pentru declararea numelui icon-lui care va fi atașat:

Obiect	Picture
<i>Command2</i>	c:\proiect_ex\top.bmp
<i>Command3</i>	c:\proiect_ex\prev.bmp
<i>Command4</i>	c:\proiect_ex\next.bmp
<i>Command5</i>	c:\proiect_ex\end.bmp

Se atașează următoarele linii de cod la evenimentul **Click**

Command2 – deplasare la începutul fișierului:

GO TOP

THISFORM.PAGEFRAME1.PAGE2.REFRESH

Command3 – deplasare o înregistrare înapoi:

SKIP -1

IF BOF(2)

GO TOP

MESSAGEBOX('INCEPUT FISIER !',64)

ENDIF

THISFORM.PAGEFRAME1.PAGE2.REFRESH

Command4 – deplasare o înregistrare înainte:

SKIP 1

IF EOF(2)

GO BOTTOM

MESSAGEBOX('SFÂRSIT FISIER !',64)

ENDIF

THISFORM.PAGEFRAME1.PAGE2.REFRESH

Command5 – deplasare la sfârșitul fișierului:

GO BOTTOM

THISFORM.PAGEFRAME1.PAGE2.REFRESH

Videoformatul *Modificare* pentru tabela *Intrari*

Fig. 10

În continuare se stabilesc obiectelor pentru ultima pagină **Page3** (proprietatea **Caption='Intrari'**, **FontBold=True**) din setul de pagini **PageFrame1**. Următoarele obiecte vor fi inserate în acest videoformat : **List1**, **List2**, **Text1**, **Text2**, **Text3**, **Text4**, **Text5**, cinci butoane de comandă (**Command1..Command5**) și șapte etichete **Label1..Label7**, Figura 10.

Pentru stabilirea proprietăților pentru obiectele declarate în videoformat se procedează astfel:

- pentru obiectul **Page3** (**Intrari**) se accesează codul procedurilor atașate diferitelor evenimente Se selectează procedura **Activate** și se adăugă următoarele linii de cod în editorul deschis

```
SELECT 2
```

```
GO TOP
```

```
SELECT 3
```

```
GO TOP
```

```
SELECT 4
```

```
GO TOP
```

```
THISFORM.PAGEFRAME1.PAGE3.REFRESH
```

- pentru obiectul **List1** - se alege din fereastra de proprietăți **RowSource**, unde se declară *materiale*, **RowSourceType**, se alege **Fields**, **ControlSource**, se declară *denumire* (câmpul cu același nume ca în fișierului *materiale.dbf*) și se atașează la evenimentul **Click** codul:

SELECT 2

CODM=COD_MAT

THISFORM.PAGEFRAME1.PAGE3.TEXT1.REFRESH

- pentru obiectul **List2** se alege din fereastra de proprietăți **RowSource**, unde se declară *furnizori*, **RowSourceType**, se alege **Fields**, **ControlSource**, se declară *denumire* (câmpul cu același nume ca în fișierul *furnizori.dbf*) și se atașează la evenimentul **Click** codul:

SELECT 3

CODF=COD_FURN

THISFORM.PAGEFRAME1.PAGE3.TEXT2.REFRESH

THISFORM.PAGEFRAME1.PAGE3.TEXT3.SETFOCUS

- pentru obiectul **Text1** se introduce în fereastra de proprietăți la **ControlSource** valoarea *cod_mat* (câmpul din fișier corespunzător codului de material), **FontBold= True** și **ReadOnly=True**.
- Pentru obiectul **Text2** se introduce în fereastra de proprietăți la **ControlSource** valoarea *cod_furn* (câmpul din fișier corespunzător codului de furnizor), **FontBold=True** și **ReadOnly= True**.
- pentru obiectul **Text3** se introduce în fereastra de proprietăți la **ControlSource** valoarea *cant* (câmpul în care este stocată cantitatea);
- pentru obiectul **Text4** se introduce în fereastra de proprietăți la **ControlSource** valoarea *pu* (câmpul în care este stocat prețul unitar);
- pentru obiectul **Text5** se introduce în fereastra de proprietăți la **ControlSource** valoarea *data_in* (câmpul în care este stocată data intrării);
- pentru butonul **Command1** se stabilesc următoarele proprietăți: **Caption='Stergere'**, **FontBold= True**, **ForeColor=255,0,0** și se atașează următorul cod la evenimentul **Click**:

BTNVALUE=0

BTNVALUE=MESSAGEBOX('STERGERE ?',4+32+256)

IF BTNVALUE=6 THEN

DELETE

PACK

ENDIF

THISFORM.PAGEFRAME1.PAGE3.REFRESH

Secvența de cod realizează ștergerea înregistrării curente în urma unei confirmări suplimentare solicitată prin cutia de dialog **MessageBox**; valoarea returnată de cutia de dialog preluată în variabila *btnvalue* este testată și dacă s-a ales butonul **OK** se realizează ștergerea. Parametrii cutiei de dialog **MessageBox** sunt:

MESSAGEBOX(*cMessageText* [, *nDialogBoxType* [, *cTitleBarText*]])

cMessageText -textul care va fi afișat;

nDialogBoxType – specifică tipurile de butoane ,butonul implicit și icon-ul afișat în antet;

cTitleBarText – titlul cutiei de dialog (antet) .

Valorile pentru *nDialogBoxType* sunt date în tabelul de mai jos:

Valoare	Tipuri butoane
0	Numai butonul OK
1	OK și Cancel
2	Abort, Retry, and Ignore
3	Yes, No și Cancel
4	Yes și No
5	Retry și Cancel

Valoare	Icon
16	Stop
32	Semnul întrebării
48	Semnul exclamării
64	Semnul Informație (I)

Valoare	Buton implicit
0	Primul buton
256	Al doilea buton
512	Al treilea buton

(4- butoanele **YES** și **NO**, 32- tipul de icon afișat- semnul întrebării și 256- butonul doi, **NO** este implicit).

- cele patru butoane (**Command2..Command5**) se utilizează pentru parcurgerea înregistrărilor din fișierul *intrari.dbf* având proprietatea **Picture** cu următoarele valori:

Obiect	Picture
Command2	c:\proiect_ex\top.bmp
Command3	c:\proiect_ex\prev.bmp
Command4	c:\proiect_ex\next.bmp
Command5	c:\proiect_ex\end.bmp

la care se atașează următorul cod:

Command2 -deplasare la începutul fișierului:

```
SELECT 4
GO TOP
THISFORM.PAGEFRAME1.PAGE3.REFRESH
```

Command3 – deplasare o înregistrare înapoi:

```
SELECT 4
SKIP -1
IF BOF(4)
    GO TOP
    MESSAGEBOX('INCEPUT FISIER !',64)
ENDIF
SELECT 4
THISFORM.PAGEFRAME1.PAGE3.REFRESH
```

Command4 – deplasare o înregistrare înainte:

```
SELECT 4
SKIP 1
IF EOF(4)
    GO BOTTOM
```

```

MESSAGEBOX('SFÂRSIT FISIER !',64)
ENDIF
SELECT 4
THISFORM.PAGEFRAME1.PAGE3.REFRESH

```

Command5 – deplasare la sfârșitul fișierului:

```

SELECT 4
GO BOTTOM
THISFORM.PAGEFRAME1.PAGE3.REFRESH

```

- cele șapte etichete au stabilite următoarele proprietăți:

Obiect	Caption	FontBold	ForeColor
<i>Label1</i>	Cod material	true	0,0,0
<i>Label2</i>	Cod furnizor	true	0,0,0
<i>Label3</i>	Cantitate	true	0,0,0
<i>Label4</i>	Pret unitar	true	0,0,0
<i>Label5</i>	Data intrarii	true	0,0,0
<i>Label6</i>	Denumire material	true	0,64,0
<i>Label7</i>	Denumire furnizor	true	255,0,0

Videoformatul *mat_data.scx* pentru lista materiale/Data

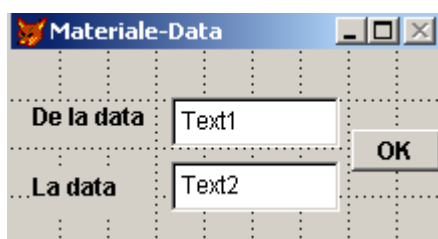


Fig. 11

Acest videoformat are 5 obiecte: două obiecte **TextBox** (**Text1**, **Text2**), două etichete (**Label1**, **Label2**) și un buton de comandă (**Command1**), Figura 11.

Proprietățile pentru aceste obiecte sunt fixate astfel:

- procedura atașată evenimentului **Init** este:

```
PUBLIC DATAIN1,DATAIN2
```

```
DATAIN1={ / / }
```

```
DATAIN2={ / / }
```

Se utilizează două variabile pentru a prelua intervalul calendaristic pentru care se face selecția înregistrărilor din fișierul *intrari.dbf*.

- procedura atașată evenimentului **Activate** este:

```
DATAIN1={ / / }
```

```
DATAIN2={ / / }
```

```
THISFORM.TEXT1.SETFOCUS
```

```
THISFORM.REFRESH
```

Se inițializează cele două variabile declarate anterior și se trimite controlul către obiectul **Text1**.

- cele două etichete **Label1** și **Label2** vor avea stabilite următoarele proprietăți:

Obiect	Caption	FontBold	ForeColor
Label1	De la data	true	0,0,0
Label1	La data	true	0,0,0

Text1 și **Text2** vor avea declarat pentru **ControlSource** *datain1* și *datain2*.

Command1 va avea atașat pe evenimentul **Click** următoarea secvență de cod

```
IF DATAIN1>DATAIN2 THEN
```

```
    MESSAGEBOX('A DOUA DATĂ CALENDARISTICĂ TREBUIE SĂ FIE MAI MARE  
DECÂT PRIMA !',48)
```

```
    DATAIN2={ / / }
```

```
    THISFORM.TEXT2.SETFOCUS
```

```
THISFORM.REFRESH
ELSE
    SELECT * FROM INTRARI INTO TABLE TEMP1 HAVING
BETWEEN(DATA_IN,DATAIN1,DATAIN2) ORDER BY DATA_IN ;  ASC
    ALTER TABLE TEMP1 DROP COLUMN COD_FURN
    ALTER TABLE TEMP1 ADD COLUMN DENUMIRE C(40)
    USE
    SELECT 5
    USE TEMP1
    FOR I=1 TO RECCOUNT(5)
        GOTO I
        CODM=COD_MAT
        SELECT 2
        LOCATE FOR CODM=COD_MAT
        IF FOUND(2) THEN
            DENM=DENUMIRE
        ENDIF
        SELECT 5
        REPLACE DENUMIRE WITH DENM
    ENDFOR
    SELECT 5
    REPORT FORM MATER_INTR PREVIEW
    USE
    DELETE FILE 'TEMP1.DBF'
    THISFORM.RELEASE
ENDIF
```

Secvența de cod realizează validarea intervalului de timp comparând *datain1* cu *datain2*. în cazul în care *datain1* > *datain2* intervalul nu este valid, se afișează un mesaj explicativ, se reinițializează variabila *datain2* și controlul este trimis la obiectul **Text2**.

În cazul în care perioada este validată se realizează o selecție din fișierul *intrari.dbf* în fișierul *temp1.dbf* a tuturor înregistrărilor care respectă condiția ca data calendaristică stocată în câmpul

data_in să fie cuprinsă în perioada care este preluată în variabilele *datain1* și *datain2*, ordonate ascendent după câmpul *data_in*.

Se șterge coloana *cod_furn* din fișierul *temp1.dbf* și se adaugă o nouă coloană cu numele *denumire* de tip caracter (se utilizează comanda SQL ALTER TABLE).

Se selectează zona de lucru 5 și se deschide tabela *temp.dbf* și într-un ciclu FOR se parcurg secvențial înregistrările din acest fișier concomitent cu căutarea denumirii materialului în fișierul *materiale.dbf* deschis în zona de lucru 2 (pe baza câmpului comun *cod_mat*). Se transferă conținutul câmpului *denumire* din fișierul *materiale.dbf* în câmpul *denumire* din fișierul *temp1.dbf*.

La ieșirea din ciclu se lansează execuția raportului *mater_intr.frx* cu datele fișierului *temp1.dbf*. După terminarea execuției raportului, se șterge fișierul *temp1.dbf* și se dă controlul meniului *menu1.mpr*.

Raportul în **Design View** are următoarea structură, Figura 12.

Report Designer - mater_intr.frx

LISTA MATERIALELOR INTRATE

Data intrării	Denumire	Cantitate	Pret unitar	Valoare
data_in	denumire	cant	pu	cant*pu
				TOTAL
				valoare
				Lei

Summary

Fig. 12

În meniul **Report** se alege opțiunea **Title/Summary** și în fereastra apărută, Figura 13, se selectează **Summary band** care va asigura tipărirea câmpurilor imediat ce se termină raportul; în caz contrar afișarea se va face în josul paginii (**Page Footer**).

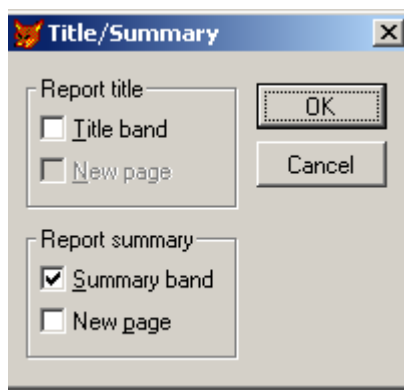


Fig. 13

În meniul **Report** la opțiunea **Variables**, se definește variabila *valoare* în care se va stoca suma produselor $pu * cant$, obținute de la fiecare înregistrare pentru a afișa o valoare totală a materialelor achiziționate într-o anumită perioadă de timp, Figura 14.

În secțiunea **Page Header** sunt șase etichete corespunzătoare titlului raportului și capului de tabel pentru care se afișează datele din fișierul *temp1.dbf*.

În secțiunea **Detail** sunt patru câmpuri de editare corespunzătoare datelor din fișier (*data_in*, *denumire*, *cant*, *pu*) și un câmp utilizat pentru a afișa valoarea mărfii ($pu * cant$) pentru fiecare înregistrare.

În secțiunea **Page Footer** nu se trece nimic.

În secțiunea **Summary** se declară câmpul pentru afișarea variabilei *valoare* care sumează produsele parțiale $pu * cant$.

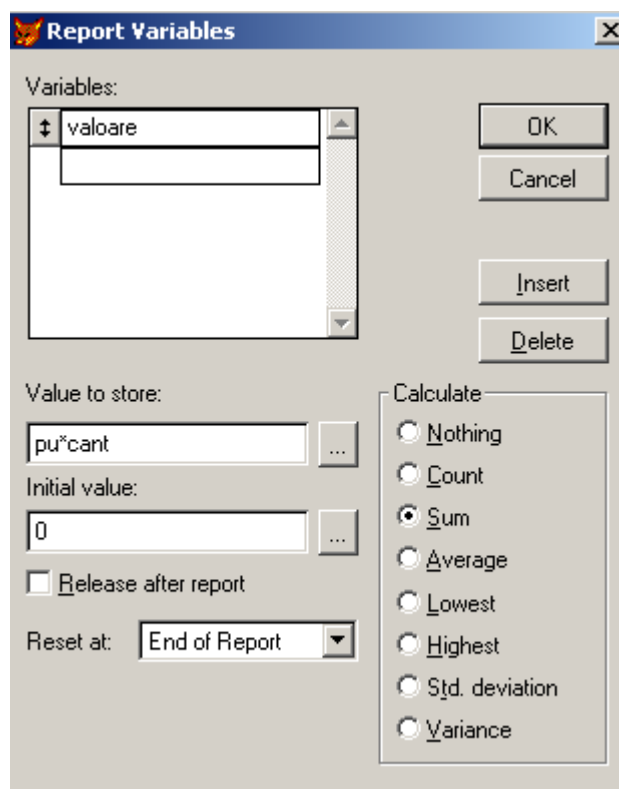


Fig. 14

Opțiunea *Lista Furnizori/Materiale* apelează în secvența de cod atașată raportul *furniz_mater.frx* care are următoarea structură în modul **Design View**, Figura 15:

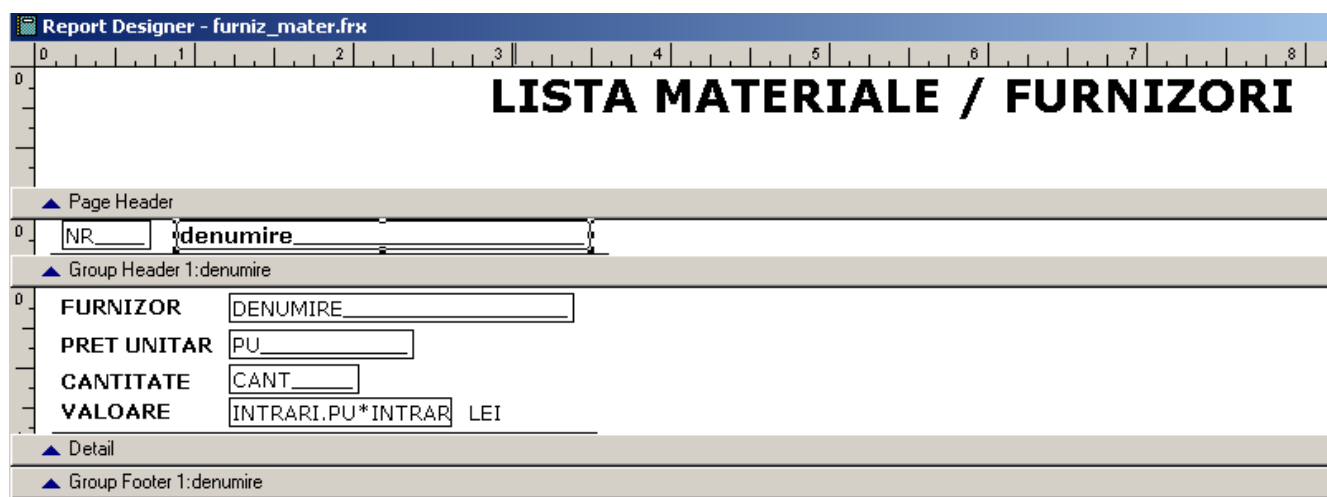


Fig. 15

În meniul **Report**, opțiunea Data Grouping, se declară câmpul *denumire* din fișierul *materiale.dbf*, după care se va face gruparea datelor, utilizându-se adresarea prin calificare

(*nume_fișier.câmp*, respectiv *materiale.denumire*) Figura 16. Opțiunea se folosește pentru a parcurge secvențial înregistrările din fișier la care se vor asocia și tipări, pe baza relațiilor definite în celelalte două fișiere, datele corespunzătoare.

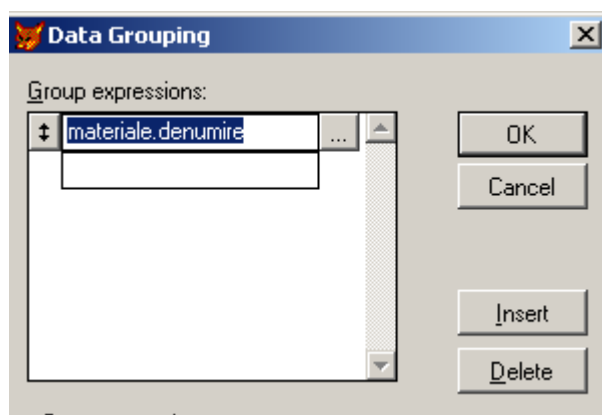


Fig. 16

Din meniul **Report** se selectează opțiunea **Variables** și se definește variabila *NR*, iar la opțiunea **Calculations**, se selectează **Count**, Figura 17. Variabila se utilizează pentru numărarea înregistrărilor tipărite obținându-se astfel numărul curent.

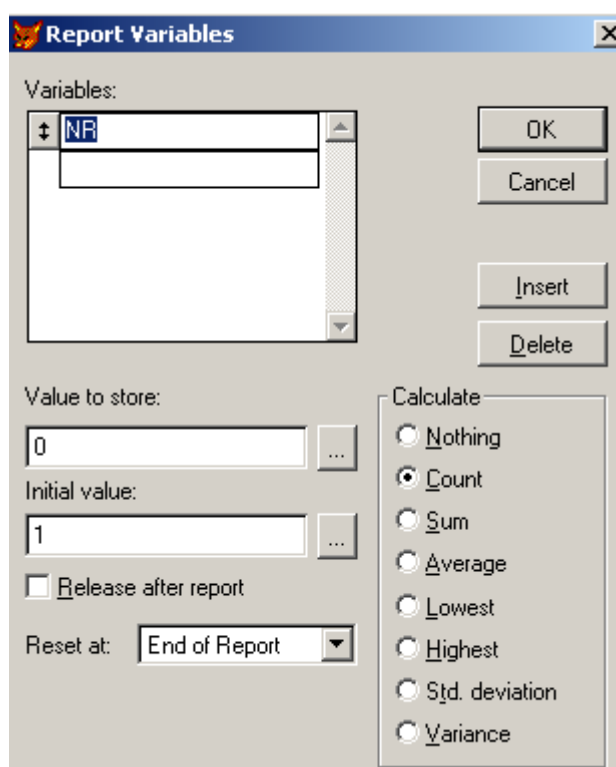


Fig. 17


```

*: Documented using Visual FoxPro Formatting wizard version .05
*:*****
*:   INIT_VAR
PUBLIC codf,denf,adr,tl,codm,denm,pun,datain,cantit
SET date to british
codf=0
denf=''
adr=' '
tl=''
codm=0
denm=''
pun=0
datain={  /  /  }
cantit=0
CLOSE databases
IF empty(sys(2000,'materiale.dbf'))
    CREATE table materiale (cod_mat n(5),denumire c(40),um c(6))
ELSE
    USE materiale
ENDIF
INDEX on cod_mat tag cod_mat additive

IF empty(sys(2000,'intrari.dbf'))
    CREATE table intrari (cod_mat n(5),cod_furn n(4),cant n(6), pu
n(8,2),data_in d)
ELSE
    USE intrari
ENDIF

IF empty(sys(2000,'furnizori.dbf'))
    CREATE table furnizori (cod_furn n(4),denumire c(40),adresa m,tel c(10))
ELSE
    USE furnizori
ENDIF
INDEX on cod_furn tag cod_furn additive

CLOSE database
SELECT 2
USE materiale
SELECT 3
USE furnizori

```

```
SELECT 4
USE intrari
DO MENU1.MPR
READ EVENTS
***** * C:\DOCUMENTS AND SETTINGS\MIHAI\MY
DOCUMENTS\CURS_BD\APLICATII_CURS_BD\PROIECT_EX\ADAUG.SCX
Name = "Dataenvironment"
Top = -1
Left = -1
Height = 272
Width = 408
DoCreate = .T.
Caption = "Adaugare"
Name = "Form1"
ErasePage = .T.
PageCount = 3
Top = 3
Left = 4
Width = 400
Height = 266
TabIndex = 1
Name = "Pageframe1"
Page1.FontBold = .T.
Page1.Caption = "Furnizori"
Page1.Name = "Page1"
Page2.FontBold = .T.
Page2.Caption = "Materiale"
Page2.Name = "Page2"
Page3.FontBold = .T.
Page3.Caption = "Intrari"
Page3.Name = "Page3"
PROCEDURE Page1.Activate
    SELECT 3
    codf=0
    den=''
    adr=' '
    tl=''
    thisform.Pageframe1.Page1.refresh
    thisform.Pageframe1.Page1.text1.setfocus
ENDPROC
PROCEDURE Page2.Activate
```

```
SELECT 2
codm=0
denm=''
thisform.Pageframe1.Page2.refresh
thisform.Pageframe1.Page2.text1.setfocus
ENDPROC

PROCEDURE Page2.Init
PUBLIC umas(6)
umas(1)='BUC'
umas(2)='KG'
umas(3)='TO'
umas(4)='MC'
umas(5)='MP'
umas(6)='ML'

ENDPROC

PROCEDURE Page3.Activate
SELECT 2
GO top
SELECT 3
GO top
SELECT 4
pun=0
cantit=0
codm=materiale.cod_mat
codf=furnizori.cod_furn
datain={ / / }
thisform.Pageframe1.Page3.list1.setfocus
thisform.Pageframe1.Page3.refresh
ENDPROC

ControlSource = "codf"
Height = 21
Left = 104
TabIndex = 1
Top = 22
Width = 81
Name = "Text1"
ControlSource = "den"
Height = 23
Left = 103
```

```

TabIndex = 2
Top = 55
Width = 250
Name = "Text2"
ControlSource = "adr"
Height = 38
Left = 106
TabIndex = 3
Top = 96
Width = 245
Name = "Text3"
ControlSource = "tl"
Height = 23
Left = 106
TabIndex = 4
Top = 144
Width = 113
Name = "Text4"
Top = 157
Left = 337
Height = 26
Width = 53
FontBold = .T.
Caption = "Validare"
TabIndex = 5
Name = "Command1"
PROCEDURE Click
    LOCATE for codf=cod_furn
    IF found(3) then
        WAIT window 'Cod furnizor duplicat !'
    ELSE
        LOCATE for upper(alltrim(den))=upper(alltrim(denumire))
        IF found(3) then
            WAIT window 'Denumire duplicat !'
        ELSE
            APPEND blank
            REPLACE cod_furn with codf,denumire with den,adresa with adr,tel
with tl
            ENDIF
        ENDIF
        codf=0

```

```
den=''
adr=' '
tl=''
thisform.Pageframe1.Page1.refresh
thisform.Pageframe1.Page1.text1.setfocus
ENDPROC

FontBold = .T.
Caption = "Cod furnizor"
Height = 21
Left = 19
Top = 22
Width = 72
TabIndex = 6
Name = "Label1"
FontBold = .T.
Caption = "Denumire"
Height = 21
Left = 18
Top = 54
Width = 80
TabIndex = 7
Name = "Label2"
FontBold = .T.
Caption = "Adresa"
Height = 22
Left = 15
Top = 94
Width = 85
TabIndex = 8
Name = "Label3"
FontBold = .T.
Caption = "Telefon"
Height = 20
Left = 14
Top = 146
Width = 74
TabIndex = 9
Name = "Label4"
ControlSource = "codm"
Height = 25
Left = 115
```

```
TabIndex = 1
Top = 16
Width = 76
Name = "Text1"
ControlSource = "denm"
Height = 27
Left = 115
TabIndex = 2
Top = 64
Width = 256
Name = "Text2"
PROCEDURE Valid
    thisform.Pageframe1.Page2.list1.setfocus
ENDPROC
Top = 165
Left = 330
Height = 29
Width = 53
FontBold = .T.
Caption = "Validare"
TabIndex = 4
Name = "Command1"
PROCEDURE Click
    LOCATE for codm=cod_mat
    IF found(2) then
        WAIT window 'Cod material duplicat !'
    ELSE
        LOCATE for upper(alltrim(denm))=upper(alltrim(denumire))
        IF found(2) then
            WAIT window 'Denumire duplicat !'
        ELSE
            APPEND blank
            REPLACE cod_mat with codm,denumire with denm,;
                um with thisform.Pageframe1.Page2.list1.value
        ENDIF
    ENDIF
    codm=0
    denm=''
    thisform.Pageframe1.Page2.refresh
    thisform.Pageframe1.Page2.text1.setfocus
ENDPROC
```

```
FontBold = .T.
Caption = "Cod material"
Height = 24
Left = 10
Top = 20
Width = 82
TabIndex = 5
Name = "Label1"
FontBold = .T.
Caption = "Denumire"
Height = 21
Left = 10
Top = 70
Width = 81
TabIndex = 6
Name = "Label2"
RowSourceType = 5
RowSource = "umas"
Height = 108
Left = 125
TabIndex = 3
Top = 110
Width = 100
Name = "List1"
PROCEDURE Click
    WITH thisform.Pageframe1.Page2
        .Text3.value=.list1.value
        .Text3.refresh
        .Command1.setfocus
    ENDWITH

ENDPROC
FontBold = .T.
Caption = "Unitate de masura"
Height = 22
Left = 10
Top = 120
Width = 105
Name = "Label3"
FontBold = .T.
Height = 25
```



```
Left = 11
ReadOnly = .T.
Top = 142
Width = 102
Name = "Text3"
FontBold = .T.
ControlSource = "codm"
Height = 25
Left = 86
ReadOnly = .T.
TabIndex = 1
Top = 85
Width = 84
Name = "Text1"
FontBold = .T.
ControlSource = "codf"
Height = 25
Left = 85
ReadOnly = .T.
TabIndex = 3
Top = 204
Width = 82
Name = "Text2"
ControlSource = "cantit"
Height = 25
Left = 276
TabIndex = 5
Top = 30
Width = 84
Name = "Text3"
ControlSource = "pun"
Height = 25
Left = 276
TabIndex = 6
Top = 67
Width = 107
Name = "Text4"
ControlSource = "datain"
Height = 25
Left = 276
TabIndex = 7
```

```
Top = 102
Width = 75
Name = "Text5"
BoundColumn = 1
RowSourceType = 6
RowSource = "materiale"
ControlSource = "denumire"
Height = 51
Left = 5
Sorted = .F.
TabIndex = 2
Top = 28
Width = 164
BoundTo = .T.
Name = "List1"
PROCEDURE Click
    SELECT 2
    codm=cod_mat
    thisform.Pageframe1.Page3.text1.refresh
ENDPROC
RowSourceType = 6
RowSource = "furnizori"
ControlSource = "denumire"
Height = 52
Left = 4
Sorted = .F.
TabIndex = 4
Top = 142
Width = 163
Name = "List2"
PROCEDURE Click
    SELECT 3
    codf=cod_furn
    thisform.Pageframe1.Page3.Text2.refresh
    thisform.Pageframe1.Page3.Text3.setfocus
ENDPROC
Top = 156
Left = 280
Height = 27
Width = 77
FontBold = .T.
```

```
Caption = "Adaugare"
TabIndex = 8
Name = "Command1"
PROCEDURE Click
    SELECT 4
    APPEND blank
    REPLACE cod_furn with codf, cod_mat with codm, cant with cantit,pu with
pun,;
        data_in with datain
    pun=0
    cantit=0
    SELECT 2
    GO top
    codm=materiale.cod_mat
    SELECT 3
    GO top
    codf=furnizori.cod_furn
    datain={ / / }
    thisform.Pageframe1.Page3.refresh
    thisform.Pageframe1.Page3.list1.setfocus
ENDPROC
FontBold = .T.
Caption = "Cod material"
Height = 23
Left = 7
Top = 87
Width = 81
TabIndex = 9
Name = "Label1"
FontBold = .T.
Caption = "Cod furnizor"
Height = 24
Left = 8
Top = 207
Width = 69
TabIndex = 10
Name = "Label2"
FontBold = .T.
Caption = "Cantitate"
Height = 28
Left = 204
```

```
Top = 34
Width = 57
TabIndex = 11
Name = "Label3"
FontBold = .T.
Caption = "Pret unitar"
Height = 28
Left = 204
Top = 72
Width = 66
TabIndex = 12
Name = "Label4"
FontBold = .T.
Caption = "Data intrarii"
Height = 26
Left = 204
Top = 105
Width = 69
TabIndex = 13
Name = "Label5"
FontBold = .T.
Caption = "Denumire material"
Height = 16
Left = 14
Top = 7
Width = 125
TabIndex = 14
ForeColor = 0,64,0
Name = "Label6"
FontBold = .T.
Caption = "Denumire furnizor"
Height = 22
Left = 7
Top = 122
Width = 142
TabIndex = 15
ForeColor = 255,0,0
Name = "Label7"
Top = 235
Left = 348
Height = 25
```

```
Width = 49
FontBold = .T.
Caption = "Iesire"
TabIndex = 2
Name = "Command1"
PROCEDURE Click
    thisform.release
ENDPROC
***** * C:\DOCUMENTS AND SETTINGS\MIHAI\MY
DOCUMENTS\CURS_BD\APLICATII_CURS_BD\PROIECT_EX\END.BMP
***** * C:\DOCUMENTS AND SETTINGS\MIHAI\MY
DOCUMENTS\CURS_BD\APLICATII_CURS_BD\PROIECT_EX\FURNIZ_MATER.FRX
***** * C:\DOCUMENTS AND SETTINGS\MIHAI\MY
DOCUMENTS\CURS_BD\APLICATII_CURS_BD\PROIECT_EX\MAT_DATA.SCX
Name = "Dataenvironment"
Top = 1
Left = 67
Height = 94
Width = 214
DoCreate = .T.
Caption = "Materiale-Data"
Name = "Form1"
PROCEDURE Activate
    datain1={ / / }
    datain2={ / / }
    thisform.text1.setfocus
    thisform.refresh
ENDPROC
PROCEDURE Init
    PUBLIC datain1,datain2
    datain1={ / / }
    datain2={ / / }
ENDPROC
Top = 39
Left = 170
Height = 21
Width = 45
FontBold = .T.
Caption = "OK"
TabIndex = 3
Name = "Command1"
```

```
PROCEDURE Click
    IF datain1>datain2 then
        MESSAGEBOX('A doua data calendaristica trebuie sa fie mai mare decât
prima !',48)
        datain2={  /  /  }
        thisform.Text2.setfocus
        thisform.refresh
    ELSE
        SELECT * from intrari into table temp1 having
between(data_in,datain1,datain2) order by data_in asc
        ALTER table temp1 drop column cod_furn
        ALTER table temp1 add column denumire c(40)
        USE
        SELECT 5
        USE temp1
        FOR i=1 to reccount(5)
            GOTO i
            codm=cod_mat
            SELECT 2
            LOCATE for codm=cod_mat
            IF found(2) then
                denm=denumire
            ENDIF
            SELECT 5
            REPLACE denumire with denm
        ENDFOR

        SELECT 5
        REPORT form mater_intr preview
        USE
        Delete file 'temp1.dbf'
        thisform.release
    ENDIF
ENDPROC

ControlSource = "datain1"
Height = 25
Left = 80
TabIndex = 1
Top = 23
Width = 84
Name = "Text1"
```

```
ControlSource = "datain2"
```

```
Height = 25
```

```
Left = 80
```

```
TabIndex = 2
```

```
Top = 55
```

```
Width = 84
```

```
Name = "Text2"
```

```
FontBold = .T.
```

```
Caption = "De la data"
```

```
Height = 22
```

```
Left = 11
```

```
Top = 25
```

```
Width = 61
```

```
TabIndex = 4
```

```
Name = "Label1"
```

```
FontBold = .T.
```

```
Caption = "La data"
```

```
Height = 22
```

```
Left = 11
```

```
Top = 60
```

```
Width = 61
```

```
TabIndex = 5
```

```
Name = "Label2"
```

```
***** * C:\DOCUMENTS AND SETTINGS\MIHAI\MY
```

```
DOCUMENTS\CURS_BD\APLICATII_CURS_BD\PROIECT_EX\MATER_INTR.FRX
```

```
***** * C:\DOCUMENTS AND SETTINGS\MIHAI\MY
```

```
DOCUMENTS\CURS_BD\APLICATII_CURS_BD\PROIECT_EX\MENU1.MNX
```

```
***** * C:\DOCUMENTS AND SETTINGS\MIHAI\MY
```

```
DOCUMENTS\CURS_BD\APLICATII_CURS_BD\PROIECT_EX\MENU1.MPR
```

```
*:*****
```

```
*:
```

```
*: Procedure File C:\DOCUMENTS AND SETTINGS\MIHAI\MY
```

```
DOCUMENTS\CURS_BD\APLICATII_CURS_BD\PROIECT_EX\DOC\MENU1.MPR
```

```
*:
```

```
*:
```

```
*:
```

```
*:
```

```
*:
```

```
*:
```

```
*:
```

```
*:
```

```
*:
*:
*:
*:
*:
*:
*:
*: Documented using Visual FoxPro Formatting wizard version .05
*: *****
*: MENU1
*: _1j10vft5m
*: _1j10vft5n
*: _1j10vft5o
*: _1j10vft5p
*: _1j10vft5q
*: _1j10vft5r
*: _1j10vft5s
*
* *****
*
* * 26/05/05 MENU1.MPR 14:40:05
*
* *****
*
* * Author's Name
*
* * Copyright (C) 2005 Company Name
*
* * Address
*
* * City, Zip
*
*
* * Description:
*
* * This PROGRAM was automatically generated BY GENMENU.
*
* *****
*
* *****
*
*
* Menu Definition
*
* *****
```



```

SET SYSMENU TO
SET SYSMENU AUTOMATIC

DEFINE PAD _1j10vft5i OF _MSYSMENU PROMPT "Vizualizare BD" COLOR SCHEME 3 ;
    KEY ALT+V, ""
DEFINE PAD _1j10vft5j OF _MSYSMENU PROMPT "Actualizare BD" COLOR SCHEME 3 ;
    KEY ALT+A, ""
DEFINE PAD _1j10vft5k OF _MSYSMENU PROMPT "Rapoarte" COLOR SCHEME 3 ;
    KEY ALT+R, ""
DEFINE PAD _1j10vft5l OF _MSYSMENU PROMPT "Iesire" COLOR SCHEME 3 ;
    KEY ALT+i, ""
ON PAD _1j10vft5i OF _MSYSMENU ACTIVATE POPUP vizualizar
ON PAD _1j10vft5j OF _MSYSMENU ACTIVATE POPUP actualizar
ON PAD _1j10vft5k OF _MSYSMENU ACTIVATE POPUP rapoarte
ON SELECTION PAD _1j10vft5l OF _MSYSMENU ;
    DO _1j10vft5m ;
    IN LOCFILE("PROIECT_EX\MENU1" , "MPX;MPR|FXP;PRG" , "WHERE is MENU1?")

DEFINE POPUP vizualizar MARGIN RELATIVE SHADOW COLOR SCHEME 4
DEFINE BAR 1 OF vizualizar PROMPT "Materiale"
DEFINE BAR 2 OF vizualizar PROMPT "Furnizori"
DEFINE BAR 3 OF vizualizar PROMPT "Intrari"
ON SELECTION BAR 1 OF vizualizar ;
    DO _1j10vft5n ;
    IN LOCFILE("PROIECT_EX\MENU1" , "MPX;MPR|FXP;PRG" , "WHERE is MENU1?")
ON SELECTION BAR 2 OF vizualizar ;
    DO _1j10vft5o ;
    IN LOCFILE("PROIECT_EX\MENU1" , "MPX;MPR|FXP;PRG" , "WHERE is MENU1?")
ON SELECTION BAR 3 OF vizualizar ;
    DO _1j10vft5p ;
    IN LOCFILE("PROIECT_EX\MENU1" , "MPX;MPR|FXP;PRG" , "WHERE is MENU1?")

DEFINE POPUP actualizar MARGIN RELATIVE SHADOW COLOR SCHEME 4
DEFINE BAR 1 OF actualizar PROMPT "Adaugare"
DEFINE BAR 2 OF actualizar PROMPT "Modificare/Stergere"
ON SELECTION BAR 1 OF actualizar ;
    DO _1j10vft5q ;
    IN LOCFILE("PROIECT_EX\MENU1" , "MPX;MPR|FXP;PRG" , "WHERE is MENU1?")
ON SELECTION BAR 2 OF actualizar ;
    DO _1j10vft5r ;
    IN LOCFILE("PROIECT_EX\MENU1" , "MPX;MPR|FXP;PRG" , "WHERE is MENU1?")

```

```

DEFINE POPUP rapoarte MARGIN RELATIVE SHADOW COLOR SCHEME 4
DEFINE BAR 1 OF rapoarte PROMPT "Lista materiale/Data"
DEFINE BAR 2 OF rapoarte PROMPT "Lista Furnizori/Materiale"
ON SELECTION BAR 1 OF rapoarte do form mat_data
ON SELECTION BAR 2 OF rapoarte ;
    DO _1j10vft5s ;
    IN LOCFILE("PROIECT_EX\MENU1" ,"MPX;MPR|FXP;PRG" ,"WHERE is MENU1?")

```

```

*          *****
*
*  * _1J10VFT5M  ON SELECTION PAD
*
*  * Procedure Origin:
*
*  * From Menu:  MENU1.MPR,          Record:   16
*  * Called By:  ON SELECTION PAD
*  * Prompt:     Iesire
*  * Snippet:    1
*
*          *****

```

```

PROCEDURE _1j10vft5m
    Clear events
    CLOSE database
    SET sysmenu to default

```

```

*          *****
*
*  * _1J10VFT5N  ON SELECTION BAR 1 OF POPUP vizualizar
*
*  * Procedure Origin:
*
*  * From Menu:  MENU1.MPR,          Record:    5
*  * Called By:  ON SELECTION BAR 1 OF POPUP vizualizar
*  * Prompt:     Materiale
*  * Snippet:    2
*
*          *****

```

```

PROCEDURE _1j10vft5n

```

```
SELECT 2
BROWSE fields cod_mat, denumire noedit
```

```

*          *****
*          *
*          * _1J10VFT5O  ON SELECTION BAR 2 OF POPUP vizualizar
*          *
*          * Procedure Origin:
*          *
*          * From Menu:  MENU1.MPR,          Record:    6
*          * Called By:  ON SELECTION BAR 2 OF POPUP vizualizar
*          * Prompt:     Furnizori
*          * Snippet:    3
*          *
*          *****

```

```
PROCEDURE _1j10vft5o
```

```
SELECT 3
BROWSE fields cod_furn, denumire noedit
```

```

*          *****
*          *
*          * _1J10VFT5P  ON SELECTION BAR 3 OF POPUP vizualizar
*          *
*          * Procedure Origin:
*          *
*          * From Menu:  MENU1.MPR,          Record:    7
*          * Called By:  ON SELECTION BAR 3 OF POPUP vizualizar
*          * Prompt:     Intrari
*          * Snippet:    4
*          *
*          *****

```

```
PROCEDURE _1j10vft5p
```

```
SELECT 4
BROWSE noedit
```

```

*          *****
*          *
*          * _1J10VFT5Q  ON SELECTION BAR 1 OF POPUP actualizar

```

```

*      *
*      * Procedure Origin:
*      *
*      * From Menu:  MENU1.MPR,          Record:  10
*      * Called By:  ON SELECTION BAR 1 OF POPUP actualizar
*      * Prompt:     Adaugare
*      * Snippet:    5
*      *
*      *****
PROCEDURE _1j10vft5q
DO form adaug

*      *****
*      *
*      * _1J10VFT5R  ON SELECTION BAR 2 OF POPUP actualizar
*      *
*      * Procedure Origin:
*      *
*      * From Menu:  MENU1.MPR,          Record:  11
*      * Called By:  ON SELECTION BAR 2 OF POPUP actualizar
*      * Prompt:     Modificare/Stergere
*      * Snippet:    6
*      *
*      *****
PROCEDURE _1j10vft5r
DO form modificare

*      *****
*      *
*      * _1J10VFT5S  ON SELECTION BAR 2 OF POPUP rapoarte
*      *
*      * Procedure Origin:
*      *
*      * From Menu:  MENU1.MPR,          Record:  15
*      * Called By:  ON SELECTION BAR 2 OF POPUP rapoarte
*      * Prompt:     Lista Furnizori/Materiale
*      * Snippet:    7
*      *
*      *****

```

```
PROCEDURE _1j10vft5s
    SELECT 4
    SET order to tag cod_mat of materiale.cdx in materiale
    SET relation to cod_mat into materiale additive
    SET order to tag cod_furn of furnizori.cdx in furnizori
    SET relation to cod_furn into furnizori additive
    REPORT form furniz_mater preview
    SET relation off into materiale
    SET relation off into furnizori
***** * C:\DOCUMENTS AND SETTINGS\MIHAI\MY
DOCUMENTS\CURS_BD\APLICATII_CURS_BD\PROIECT_EX\MODIFICARE.SCX
Name = "Dataenvironment"
Top = 0
Left = -1
Height = 294
Width = 424
DoCreate = .T.
Caption = "Modificare / Stergere"
Name = "Form1"
ErasePage = .T.
PageCount = 3
Top = 9
Left = 10
Width = 400
Height = 266
TabIndex = 1
Name = "Pageframe1"
Page1.FontBold = .T.
Page1.Caption = "Furnizori"
Page1.Name = "Page1"
Page2.FontBold = .T.
Page2.Caption = "Materiale"
Page2.Name = "Page2"
Page3.FontBold = .T.
Page3.Caption = "Intrari"
Page3.Name = "Page3"
PROCEDURE Page1.Activate
    SELECT 3
    GO top
    denf=''
    thisform.Pageframe1.Page1.refresh
```

```
        thisform.Pageframe1.Page1.text1.setfocus
```

```
ENDPROC
```

```
PROCEDURE Page2.Init
```

```
    PUBLIC umas(6)
```

```
    umas(1)='BUC'
```

```
    umas(2)='KG'
```

```
    umas(3)='TO'
```

```
    umas(4)='MC'
```

```
    umas(5)='MP'
```

```
    umas(6)='ML'
```

```
ENDPROC
```

```
PROCEDURE Page2.Activate
```

```
    SELECT 2
```

```
    GO top
```

```
    denm=''
```

```
    thisform.Pageframe1.Page2.refresh
```

```
    thisform.Pageframe1.Page2.text1.setfocus
```

```
ENDPROC
```

```
PROCEDURE Page3.Activate
```

```
    SELECT 2
```

```
    GO top
```

```
    SELECT 3
```

```
    GO top
```

```
    SELECT 4
```

```
    GO top
```

```
    thisform.Pageframe1.Page3.refresh
```

```
ENDPROC
```

```
ControlSource = "cod_furn"
```

```
Height = 21
```

```
Left = 92
```

```
TabIndex = 1
```

```
Top = 11
```

```
Width = 81
```

```
Name = "Text1"
```

```
ControlSource = "denumire"
```

```
Height = 23
```

```
Left = 92
```

```
TabIndex = 2
```

```
Top = 44
```

```
Width = 250
Name = "Text2"
ControlSource = "adresa"
Height = 38
Left = 92
TabIndex = 3
Top = 85
Width = 245
Name = "Text3"
ControlSource = "tel"
Height = 23
Left = 92
TabIndex = 4
Top = 133
Width = 113
Name = "Text4"
Top = 174
Left = 246
Height = 21
Width = 62
FontBold = .T.
Caption = "Stergere"
TabIndex = 5
ForeColor = 255,0,0
Name = "Command1"
PROCEDURE Click
    codf=cod_furn
    Delete
    PACK
    SELECT 4
    LOCATE for codf=cod_furn
    DO while found(4)
        Delete
        CONTINUE
    ENDDO
    PACK
    SELECT 3
    thisform.Pageframe1.Page1.refresh
    thisform.Pageframe1.Page1.text1.setfocus
ENDPROC
FontBold = .T.
```

```
Caption = "Cod furnizor"
Height = 21
Left = 3
Top = 11
Width = 72
TabIndex = 6
Name = "Label1"
FontBold = .T.
Caption = "Denumire"
Height = 21
Left = 3
Top = 43
Width = 80
TabIndex = 7
Name = "Label2"
FontBold = .T.
Caption = "Adresa"
Height = 22
Left = 3
Top = 83
Width = 85
TabIndex = 8
Name = "Label3"
FontBold = .T.
Caption = "Telefon"
Height = 20
Left = 3
Top = 135
Width = 74
TabIndex = 9
Name = "Label4"
Top = 135
Left = 239
Height = 20
Width = 31
Picture = top.bmp
Caption = ""
Name = "Command2"
PROCEDURE Click
    GO top
    thisform.Pageframe1.Page1.refresh
```



```
ENDPROC
Top = 135
Left = 317
Height = 20
Width = 31
Picture = next.bmp
Caption = ""
Name = "Command3"
PROCEDURE Click
    Skip 1
    IF eof(3)
        GO bottom
        MESSAGEBOX('Sfârsit fisier !',64)
    ENDIF
    thisform.Pageframe1.Page1.refresh
ENDPROC
Top = 135
Left = 361
Height = 20
Width = 31
Picture = end.bmp
Caption = ""
Name = "Command4"
PROCEDURE Click
    GO bottom
    thisform.Pageframe1.Page1.refresh

ENDPROC
Top = 135
Left = 278
Height = 20
Width = 31
Picture = prev.bmp
Caption = ""
Name = "Command5"
PROCEDURE Click
    Skip -1
    IF bof(3)
        GO top
        MESSAGEBOX('Inceput fisier !',64)
    ENDIF
```

```
thisform.Pageframe1.Page1.refresh
ENDPROC
ControlSource = "denf"
Height = 24
Left = 92
Top = 173
Width = 145
Name = "Text5"
PROCEDURE KeyPress
    LPARAMETERS nKeyCode, nShiftAltCtrl

ENDPROC
PROCEDURE Valid
    LOCATE for upper(left(denf,len(alltrim(denf))))=;
        upper(left(denumire,len(alltrim(denf))))
    IF found(3) then
        thisform.Pageframe1.Page1.refresh
    ELSE
        MESSAGEBOX('Nu exista !',64)
    ENDIF
ENDPROC
FontBold = .T.
Caption = "Cauta furnizor"
Height = 22
Left = 3
Top = 176
Width = 78
Name = "Label5"
ControlSource = "cod_mat"
Height = 25
Left = 115
TabIndex = 1
Top = 16
Width = 76
Name = "Text1"
ControlSource = "denumire"
Height = 25
Left = 115
TabIndex = 2
Top = 64
```

```
Width = 256
Name = "Text2"
FontBold = .T.
Caption = "Cod material"
Height = 24
Left = 10
Top = 20
Width = 82
TabIndex = 5
Name = "Label1"
FontBold = .T.
Caption = "Denumire"
Height = 21
Left = 10
Top = 70
Width = 81
TabIndex = 6
Name = "Label2"
RowSourceType = 5
RowSource = "umas"
Enabled = .T.
Height = 71
Left = 115
TabIndex = 3
Top = 110
Width = 100
Name = "List1"
PROCEDURE Click
    WITH thisform.Pageframe1.Page2
        .Text4.value=.list1.value
        um=.Text4.value
        .Text4.refresh
        .Text3.setfocus
    ENDWITH

ENDPROC
FontBold = .T.
Caption = "Unitate de masura"
Height = 22
Left = 10
Top = 120
```

```
Width = 105
Name = "Label3"
ControlSource = "denm"
Height = 25
Left = 115
Top = 191
Width = 198
Name = "Text3"
PROCEDURE Valid
    LOCATE for upper(left(denm,len(alltrim(denm))))=;
        upper(left(denumire,len(alltrim(denm))))
    IF found(2) then
        thisform.Pageframe1.Page2.list1.value=um
        thisform.Pageframe1.Page2.refresh
    ELSE
        MESSAGEBOX('Nu exista !',64)
    ENDIF
ENDPROC
FontBold = .T.
Caption = "Cauta material"
Height = 24
Left = 10
Top = 197
Width = 102
Name = "Label4"
Top = 160
Left = 236
Height = 20
Width = 31
Picture = top.bmp
Caption = ""
Name = "Command2"
PROCEDURE Click
    GO top
    thisform.Pageframe1.Page2.refresh
ENDPROC
Top = 160
Left = 314
Height = 20
Width = 31
Picture = next.bmp
```

```
Caption = ""
Name = "Command3"
PROCEDURE Click
    Skip 1
    IF eof(2)
        GO bottom
        MESSAGEBOX('Sfârsit fisier !',64)
    ENDIF
    thisform.Pageframe1.Page2.refresh
ENDPROC
Top = 160
Left = 358
Height = 20
Width = 31
Picture = end.bmp
Caption = ""
Name = "Command4"
PROCEDURE Click
    GO bottom
    thisform.Pageframe1.Page2.refresh

ENDPROC
Top = 160
Left = 275
Height = 20
Width = 31
Picture = prev.bmp
Caption = ""
Name = "Command5"
PROCEDURE Click
    Skip -1
    IF bof(2)
        GO top
        MESSAGEBOX('Inceput fisier !',64)
    ENDIF
    thisform.Pageframe1.Page2.refresh
ENDPROC
FontBold = .T.
ControlSource = "um"
Enabled = .T.
Height = 24
```

```
Left = 18
ReadOnly = .T.
Top = 144
Width = 83
Name = "Text4"
FontBold = .T.
ControlSource = "cod_mat"
Height = 25
Left = 86
ReadOnly = .T.
TabIndex = 1
Top = 85
Width = 84
Name = "Text1"
FontBold = .T.
ControlSource = "cod_furn"
Height = 25
Left = 85
ReadOnly = .T.
TabIndex = 3
Top = 204
Width = 82
Name = "Text2"
ControlSource = "cant"
Height = 25
Left = 276
TabIndex = 5
Top = 30
Width = 84
Name = "Text3"
ControlSource = "pu"
Height = 25
Left = 276
TabIndex = 6
Top = 67
Width = 107
Name = "Text4"
ControlSource = "data_in"
Height = 25
Left = 276
TabIndex = 7
```

```
Top = 102
Width = 75
Name = "Text5"
BoundColumn = 1
RowSourceType = 6
RowSource = "materiale"
ControlSource = "denumire"
Height = 51
Left = 5
Sorted = .F.
TabIndex = 2
Top = 28
Width = 164
BoundTo = .T.
Name = "List1"
PROCEDURE Click
    SELECT 2
    codm=cod_mat
    thisform.Pageframe1.Page3.text1.refresh
ENDPROC
RowSourceType = 6
RowSource = "furnizori"
ControlSource = "denumire"
Height = 52
Left = 4
Sorted = .F.
TabIndex = 4
Top = 142
Width = 163
Name = "List2"
PROCEDURE Click
    SELECT 3
    codf=cod_furn
    thisform.Pageframe1.Page3.Text2.refresh
    thisform.Pageframe1.Page3.Text3.setfocus
ENDPROC
FontBold = .T.
Caption = "Cod material"
Height = 23
Left = 7
Top = 87
```

```
Width = 81
TabIndex = 9
Name = "Label1"
FontBold = .T.
Caption = "Cod furnizor"
Height = 24
Left = 8
Top = 207
Width = 69
TabIndex = 10
Name = "Label2"
FontBold = .T.
Caption = "Cantitate"
Height = 28
Left = 204
Top = 34
Width = 57
TabIndex = 11
Name = "Label3"
FontBold = .T.
Caption = "Pret unitar"
Height = 28
Left = 204
Top = 72
Width = 66
TabIndex = 12
Name = "Label4"
FontBold = .T.
Caption = "Data intrarii"
Height = 26
Left = 204
Top = 105
Width = 69
TabIndex = 13
Name = "Label5"
FontBold = .T.
Caption = "Denumire material"
Height = 16
Left = 14
Top = 7
Width = 125
```



```
TabIndex = 14
ForeColor = 0,64,0
Name = "Label6"
FontBold = .T.
Caption = "Denumire furnizor"
Height = 22
Left = 7
Top = 122
Width = 142
TabIndex = 15
ForeColor = 255,0,0
Name = "Label7"
Top = 156
Left = 213
Height = 20
Width = 31
Picture = top.bmp
Caption = ""
Name = "Command2"
PROCEDURE Click
    SELECT 4
    GO top
    thisform.Pageframe1.Page3.refresh
ENDPROC
Top = 156
Left = 291
Height = 20
Width = 31
Picture = next.bmp
Caption = ""
Name = "Command3"
PROCEDURE Click
    SELECT 4
    Skip 1
    IF eof(4)
        GO bottom
        MESSAGEBOX('Sfârsit fisier !',64)
    ENDIF
    thisform.Pageframe1.Page3.refresh
    SELECT 2
    LOCATE for intrari.cod_mat=materiale.cod_mat
```

```
IF found(2)
    thisform.Pageframe1.Page3.list1.value=denumire
ENDIF
SELECT 4
thisform.Pageframe1.Page3.refresh
```

ENDPROC

Top = 156

Left = 335

Height = 20

Width = 31

Picture = end.bmp

Caption = ""

Name = "Command4"

PROCEDURE Click

SELECT 4

GO bottom

thisform.Pageframe1.Page3.refresh

ENDPROC

Top = 156

Left = 252

Height = 20

Width = 31

Picture = prev.bmp

Caption = ""

Name = "Command5"

PROCEDURE Click

SELECT 4

Skip -1

IF bof(4)

GO top

MESSAGEBOX('Inceput fisier !',64)

ENDIF

thisform.Pageframe1.Page3.refresh

SELECT 3

LOCATE for intrari.cod_furn=furnizori.cod_furn

IF found(3)

thisform.Pageframe1.Page3.List2.value=denumire

ENDIF

SELECT 4

```
thisform.Pageframe1.Page3.refresh
ENDPROC
Top = 195
Left = 222
Height = 25
Width = 63
FontBold = .T.
Caption = "Stergere"
ForeColor = 255,0,0
Name = "Command1"
PROCEDURE Click
    btnvalue=0
    btnvalue=messagebox('Stergere ?',4+32+256)
    IF btnvalue=6 then
        Delete
        PACK
    ENDIF
    thisform.Pageframe1.Page3.refresh
ENDPROC
Top = 236
Left = 345
Height = 25
Width = 49
FontBold = .T.
Caption = "Iesire"
Name = "Command2"
PROCEDURE Click
    thisform.release
ENDPROC
***** * C:\DOCUMENTS AND SETTINGS\MIHAI\MY
DOCUMENTS\CURS_BD\APLICATII_CURS_BD\PROIECT_EX\NEXT.BMP
***** * C:\DOCUMENTS AND SETTINGS\MIHAI\MY
DOCUMENTS\CURS_BD\APLICATII_CURS_BD\PROIECT_EX\PREV.BMP
***** * C:\DOCUMENTS AND SETTINGS\MIHAI\MY
DOCUMENTS\CURS_BD\APLICATII_CURS_BD\PROIECT_EX\TOP.BMP
```

12.2.2. Realizarea documentației și a aplicației în format executabil

Pentru obținerea documentației referitoare la sursele proiectului se parcurg următorii pași:

- a) Se închide aplicația **Project Manager**;
- b) Din meniul **Tools** se alege opțiunea **Wizards / Documenting**. Aplicația **Documenting Wizard** parcurge 6 pași pentru elaborarea documentației proiectului:
 1. Se alege fișierul sursă (proiectul *magazie.pjx*) inclusiv calea către acesta. Se apasă butonul **Next**;
 2. Se setează caractere mari pentru cuvinte cheie (**Keywords**) și simboluri;
 3. Se setează indentare text (**Tab** sau **Space**);
 4. Se adăugă antet;
 5. Se selectează tipuri de rapoarte;
 6. Se încheie cu specificarea subdirectorului în care vor fi generate rapoartele (existent sau nou creat).

Fișierul MAGAZIE.LST va conține codul sursă (fișier de tip text care poate fi ulterior trecut într-un fișier Word) pentru toate programele și procedurile atașate obiectelor componentelor folosite în proiect.

Se generează un tabel care va conține toate fișierele proiectului, generate, primare și implicite (videoformate, rapoarte, meniuri), fișierele de tip *BMP* etc.

De exemplu, videoformatele (**Forms**) sunt salvate într-un fișier cu extensia *.scx*. La crearea videoformatului, tabelul cu extensia *.scx* conține o înregistrare pentru videoformat, o înregistrare pentru datele de mediu și două înregistrări pentru uz intern. Pentru fiecare obiect adăugat în videoformat se adaugă o nouă înregistrare.

Anumite componente ale Visual FoxPro constau din fișiere multiple: un fișier primar și unul sau mai multe fișiere implicite.

De exemplu, când se crează un videoformat, Visual FoxPro crează un fișier *.scx* (fișier primar) și un fișier *.sct* (fișier implicit)

Următoarele componente au fișiere multiple (Tabelul 1)

Tabelul 1

Componentă	Fișiere primare	Fișiere implicite
Form	.scx	.sct
Report	.frx	.frt
Label	.lbx	.lbt
Class Library	.vcx	.vct
Menu	.mnx	.mnt
Table	.dbf	.fpt, .cdx, .idx
Database	.dbc	.dct, .dcx

Tabelul 2 prezintă extensiile și tipurile de fișiere asociate care sunt utilizate în Visual FoxPro

Tabel 2

Extensie	Tip fișier
.act	Documenting Wizard action diagram
.app	Generated application or Active Document
.cdx	Compound index
.chm	Compiled HTML Help
.dbc	Database
.dbf	Table
.dbg	Configurație debugger
.dct	memo database
.dcx	index database
.dep	Fișier dependență (creat de Setup Wizard)
.dll	Windows Dynamic Link Library
.err	Eroare de compilare
.esl	Visual FoxPro suport bibliotecă
.exe	Program executabil
.fky	Macro
.fil	Visual FoxPro Dynamic Link Library

.fmt	Format Fișier
.fpt	Table memo
.frt	Report memo
.frx	Report
.fxp	Program compilat
.h	Antet fișier (pentru includere în Visual FoxPro or C/C++ program)
.hlp	WinHelp
.htm	HTML
.idx	Index, compact index
.lbt	Label memo
.lbx	Label
.log	Coverage log
.lst	Documenting Wizard list
.mem	Salvare variabile
.mnt	Menu memo
.mnx	Menu
.mpr	Program menu generat
.mpx	Program menu compilat
.ocx	ActiveX control
.pjt	Project memo
.pjx	Project
.prg	Program
.qpr	Program query generat
.qpx	Program query compilat
.sct	Form memo
.scx	Form
.spr	Screen Program generat (numai pentru versiunile anterioare FoxPro)
.spx	Screen Program compilat (numai pentru versiunile anterioare FoxPro)
.tbk	Memo backup

.txt	Text
.vct	Visual class library memo
.vcx	Clasă de bibliotecă Visual
.vue	FoxPro 2.x view
.win	Fișier Window

Pentru a genera programul executabil al aplicației, se deschide **Project Manager** pentru aplicația *magazie.pjx* și se apasă butonul **Build** și se specifică opțiunile pentru construire, Figura 1.

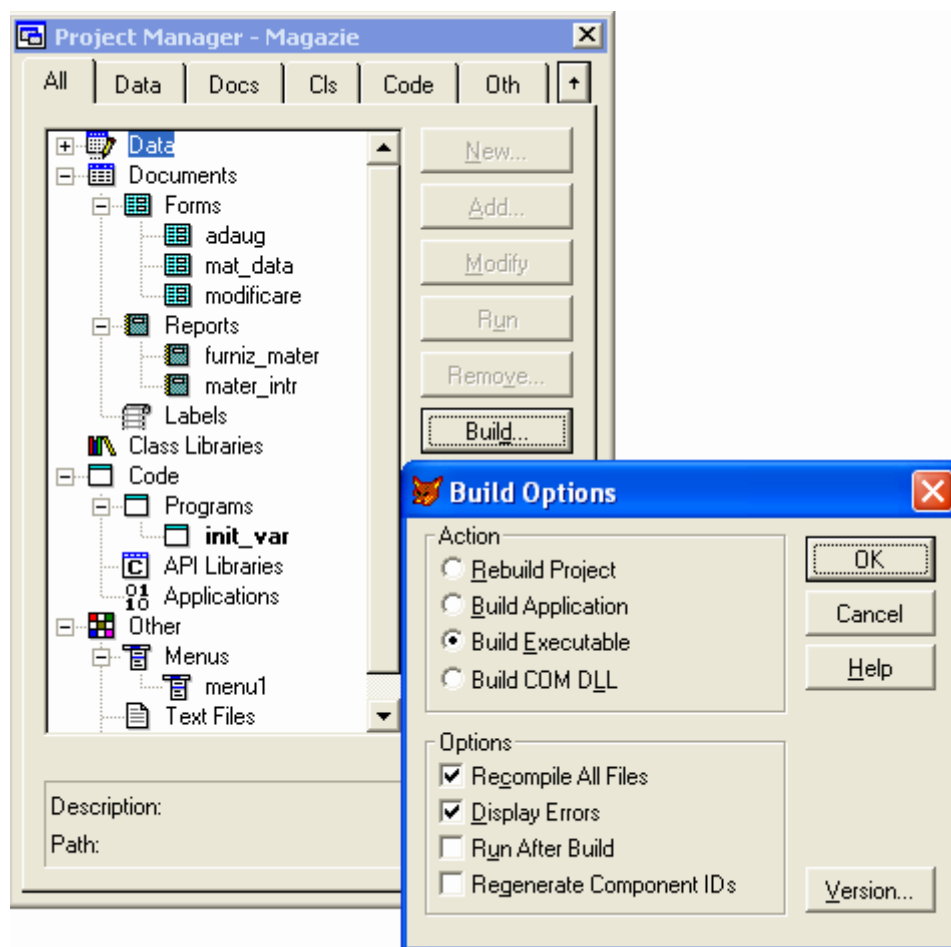


Fig. 1

Opțiunile de construire se referă la următoarele acțiuni:

- reconstruirea proiectului (**Rebuild Project**);
- construirea aplicației (**Build Application**);
- construirea programului executabil (**Build Executable**);
- construirea componentelor COM (**Component Object Model** pentru includerea controalelor **ActivX** sau dacă s-a creat un **Automation server**) sub formă de DLL (librărie de legătură dinamică).

Acțiune de construire a executabilului are un subset de opțiuni:

- recompilarea tuturor fișierelor;
- afișarea erorilor;
- lansare în execuție după construire;
- regenerarea identificării obiectelor.

Pentru celelalte acțiuni numărul opțiunilor din subset poate varia în funcție de acțiune. De exemplu, pentru construirea elementelor COM vor fi active doar trei opțiuni, fără cea de rulare după construire (un DLL nu poate fi lansat în execuție fiind o librărie dinamică).

Programul executabil generat poate fi rulat pe același calculator sau pe un altul, cu condiția ca acesta să aibă Visual FoxPro instalat.

12.2.3. Utilizarea Setup Wizard

Pentru a mări portabilitatea aplicației se folosește **Setup Wizard**, în vederea realizării de subrutine de instalare a aplicației pe un alt calculator.

Din meniul **Tools** se alege submeniul **Wizard**, apoi opțiunea **Setup**. Se parcurg 7 pași:

1. Localizarea fișierelor – se specifică subdirectorul care conține fișierele și subdirectoarele ce trebuie distribuite. Nu se poate folosi **Distrib** ca nume de subdirector în care se află fișierele și subdirectoarele ce se vor distribui;
2. Specificarea componentelor care vor fi incluse în pachetul de distribuție, și anume:
 - **Visual FoxPro 6.0 Runtime** – librăriile Visual FoxPro;
 - **Microsoft 8.0 Graph Runtime**;
 - **ODBC Drivers**;
 - **COM Components**;
 - **ActiveX Controls**;
 - **HTML help engine**;
3. Crearea subdirectorului pentru **Disk Image**:
 - **1.44 MB 3.5** – wizard-ul crează imagini pentru floppy disk;
 - **Websetup** (compressed) – wizard-ul crează un program de instalare dens compresat proiectat pentru descărcare rapidă de pe un web site;
 - **Netsetup** – wizard-ul crează un singur director care va conține toate fișierele.
4. Specificarea opțiunilor de instalare – referitoare la:

- **Setup dialog box caption** – titlul ferestrei de **Setup**, opțional;
 - **Copyright information** – obligatoriu;
 - **Post-setup executable** – specificarea unei acțiuni post instalare, opțional;
5. Identificarea destinației implicite pentru fișier – rutina setup va plasa aplicația în directorul care se specifică în **Default directory** . Opțiunea **Program group** va crea un grup de programe pentru aplicația instalată care se va regăsi în meniul **Start** al calculatorului utilizatorului. Opțiunea **User can modify**, oferă posibilitatea modificării subdirectorului implicit de instalare;
 6. Schimbarea setărilor fișierelor – se afișează fișierele și opțiunile făcute pentru ele cu posibilitatea de a se opera modificări legate de nume, destinație sau alte modificări;
 7. Terminarea procesului **Setup Wizard** – la acest pas se poate crea un fișier de dependență (.dep) care permite folosirea altor utilități la instalarea aplicației. După apăsarea butonului de **Finish, Setup Wizard** realizează următoare acțiuni:
 - înregistrează configurația pentru următoarea utilizare a distribuției din aceeași cale;
 - lansează procesul de creare a imaginilor disc.

Imaginile disc pot fi copiate pe orice mediu de stocare și poate fi distribuit utilizatorilor aplicației.

A. Anexa

Limite ale sistemului VFP 6.0

Caracteristică	Valoare
Nr. maxim de înregistrări într-o tabelă	1 miliard
Mărimea maximă a unei tabele	2 Gb
Nr. maxim de caractere pe înregistrare	65.500
Nr. maxim de câmpuri pe înregistrare	255
Nr. maxim de tabele deschise simultan	255
Nr. maxim de caractere pentru un câmp	254
Nr. maxim de bytes pentru cheia de index compus	100
Nr. maxim de bytes pentru cheia de index simplu	240
Nr. maxim de fișiere de index deschise pentru o tabelă	nelimitat
Nr. maxim de fișiere de index deschise pentru toate zonele de lucru	nelimitat
Nr. maxim de legături	nelimitat
Lungimea maximă a unei expresii relaționale	nelimitat
Valoarea maximă pentru întregi	2.147.483.647
Precizia în calculul numeric	16
Mărimea maximă pentru un câmp de tip caracter	254
Mărimea maximă pentru un câmp de tip numeric	20
Nr. maxim de caractere pentru un câmp dintr-o tabelă liberă	10
Nr. maxim de caractere pentru un câmp dintr-o tabelă care aparține unei BD	128
Nr. maxim de variabile	65.000
Nr. maxim de masive	65.000
Nr. maxim de elemente într-un masiv	65.000
Nr. maxim de linii sursă într-un program	nelimitat
Mărimea maximă a modulului de program compilat	64 Kb
Nr. maxim de proceduri (subprograme)	nelimitat
Nr. maxim de apeluri DO imbricate	128
Nr. maxim de comenzi de ciclare	384
Nr. maxim de parametri transmiși	27
Nr. maxim de tranzacții	5
Nr. maxim de obiecte la definirea unui raport	nelimitat

Nr. maxim de niveluri de grupare într-un raport	128
Nr. maxim de ferestre deschise	nelimitat
Nr. maxim de ferestre BROWSE deschise	255
Nr. maxim de caractere într-un șir de caractere	16.777.184
Nr. maxim de caractere într-o linie de comandă	8.192
Nr. maxim de fișiere deschise	limită sistem
Nr. maxim de câmpuri selectate prin comanda SELECT- SQL	255

Index de termeni

- accesul la date
 - direct, 97
 - secvențial, 97
- actualizarea bazei de date, 13
- algebră relațională
 - diferență, 20
 - diviziunea, 21
 - intersecția, 21
 - joncțiune, 21
 - joncțiune exterioară, 21
 - joncțiune exterioară completă, 21
 - joncțiune exterioară dreaptă, 21
 - joncțiune exterioară stânga, 21
 - joncțiune naturală, 21
 - produs cartezian, 20
 - proiecția, 20
 - reuniunea, 20
 - selecția, 20
- anularea marcării pentru ștergere fizică, 49
- apelul unui (sub)program, 130
- arhitectura sistemului de baze de date, 6
- atribut, 4
 - complex, 29
 - cu o singură valoare, 29
 - cu set de valori, 29
 - derivat, 29
 - fără valoare, 29
- bănci de date, 5
- bază de date, 4
- bază de date relațională, 28
- câmp, 4
- crearea etichetelor, 69
- crearea interogărilor, 70
- crearea rapoartelor, 68
- crearea structurii unei tabele, 149
- crearea tabelelor, 66
- crearea videoformatelor, 67
- de date
 - relațional, 8
- deschiderea bazei de date, 52, 60
- deschiderea unei tabele, 49
- dictionar de date, 4
- domeniu, 19
- entitate, 4
- fișier index, 94
 - compus, 94
 - simplu, 94
- funcție utilizator, 129
- închiderea bazei de date/tabelei, 64
- Independența datelor
 - fizică, 10
 - logică, 10
- indexarea tabelei, 92
- înregistrare, 4
 - fizică, 4
 - logică, 4
- interclasarea documentelor, 71
- interogarea bazei de date, 13
- legătura între tabele, 113
- matrice, 85
- metadata, 4
- model de date, 6, 7
 - distribuit, 8
 - ierarhic, 8
 - în rețea, 8
 - primitiv, 8
 - semantic (orientat obiect), 9
- modificarea înregistrărilor, 106
- modificarea structurii, 91
 - bazei de date, 62
 - tabelei, 62
- procedură utilizator, 129
- programare
 - modulară, 119
 - orientată pe obiecte, 37
 - procedurală, 119
 - structurată, 119
- programarea orientată pe obiecte
 - abstractizarea, 42
 - clasa, 37
 - clasa parinte, 38
 - evenimentul, 39
 - încapsularea, 41
 - mesaj, 38
 - metoda, 38
 - moștenirea, 41
 - obiect, 38
 - polimorfism, 41
 - proprietatea, 39
 - restricții de integritate, 38
 - subclasa, 37
- redenumire fișier, 117

- redundanță minimă, 14
- relație, 19
- reorganizarea bazei de date, 14
- Schema în modelul relațional, 29
- securitatea datelor, 14
- sistem de gestiune a bazelor de date, 5, 9
 - complet relațional, 36
 - minimal relațional, 36
 - relaționale, 19
- sisteme de baze de date, 5
- sortarea tabeli, 92
- ștergerea fișierelor, 110
- ștergerea înregistrărilor, 109
 - fizică, 50, 63, 110, 144
 - logică, 49, 63, 109, 144
- structura bazei de date, 6
- structura unei funcții, 130
- structura unei funcții, 130
- structura unei proceduri, 129
- subprograme imbricate, 130
- tabelă, 19
- transmiterea parametrilor, 130
 - prin referință, 131
 - prin valoare, 131
- validarea câmpurilor, 55
- variabilă
 - globală, 85
 - locală, 85
- vectori, 85
- zonă de lucru, 52

Bibliografie

1. **Dima, G., Dima M.** *FoxPro*, Editura Teora, București, 1993.
2. **Dima, G., Dima M.** *FoxPro Meniuri*, Editura Teora, București, 1994.
3. **Kifer, M., Bernstein, A., Lewis, Ph.** *Database Systems An Application-Oriented Approach*, State University of New York, Stony Brook, 2005.
4. **Lungu, I. și al.** *Sistemul FoxPro Prezentare și Aplicații*, Editura All, București, 1993.
5. **M. Velicanu, și al.** *Sisteme de Gestiune a Bazelor de Date*, Editura Petrion, București, 2000;
6. **Velicanu, M., Lungu, I., Muntean, M.** *Dezvoltarea Aplicațiilor cu Baze de Date în Visual FoxPro*, Editura All, București, 2001.
7. *** Visual FoxPro 6.0 Manualele Microsoft
8. *** MSDN Library, April 2003.