Universitatea Politehnica Timisoara
Facultatea de Automatica si Calculatoare
Departamentul Calculatoare si Tehnologia Informatiei

prof.dr.ing.IONEL JIAN

BAZE DE DATE

Editura POLITEHNICA
TIMISOARA 2016

PREFAȚĂ

Sistemele de gestiune a bazelor de date (SGBD) au apărut practic în jurul anului 1970 şi au cunoscut o rapidă dezvoltare pe calculatoarele de capacitate mare, medie, mini calculatoare si apoi după 1980 pe microsisteme. Se pot delimita în general 3 generații de baze de date după modelul de structura folosit pentru bazele de date realizate:

- **Modelul ierarhic** – destinat BD cu structură arborescentă reprezentat prin **IMS**-IBM (Information Management System);
- **Modelul rețea** cu structură de graf (rețea) a informațiilor din BD cu două direcții:
  - conform recomandărilor **CODASYL-DBTG 1994**
  - sistemul **SOCRATE** bazat pe o structură virtuală a BD, dezvoltat de un colectiv condus de H. Abrial la Universitatea din Grenoble (varianta Clotilde la TGV Paris).
- **Modelul relational** - dezvoltat pe baza specificațiilor de Algebră relațională a lui **Codd** (1971), în care BD este structurată în tabele legate între ele prin referințe realizate prin chei simbolice.
  Primele SGBD-uri relaționale comerciale au apărut în jurul anului 1980.
  - **SQL** - limbaj dezvoltat de IBM - New Jersey, generalizat si standardizat de toate marile companii de calculatoare (DB2-IBM, PL/SQL ORACLE, Ms SQL) sau produsele free sub Linux PostgreSQL, MySQL si PostSQL (cu posibilități de definire clase de obiecte).
  - **dBASE** - dezvoltat de firma Ashton-Tate pe microsisteme (sisteme XBase), care are un limbaj procedural si admite limbajul SQL standard.

In cadrul acestui curs s-a ales pentru lucrările utilizate sistemul **dBASE Plus** versiunea 2.8 (2014) pentru care deținem licență pe 4 ani. Limbajul dBase are cea mai mare răspândire pe microsisteme şi admite un subset destul de complet al limbajului SQL. Creat de Cecil Wayne Ratliff, dBase este lansat de firma Ashton-Tate în 1981 sub CP/M în varianta dBASE II si devine standard pentru BD pe microsisteme. În 1982 apare varianta dBASE II pentru PC care obține în 1984 premiul PC-WORLD CLASS ca cel mai bun program al anului. În 1986 se lansează dBASE III , iar în 1988 dBASE IV. În 1991 Ashton Tate este cumpărat de Borland si lansează dBASE IV ver.1.5 si pentru Windows Visual dBase 5 si 7. În ultimele versiuni se recuperează performantele de viteza unde a fost întrecut de FoxPro 2.0 si Clipper care folosesc acelaşi limbaj si

concepte, fiind practic compatibile cu dBASE. Trebuie apreciată calitatea documentației dBASE, a HELP-urilor si siguranța în funcționare. In prezent firmele Borland si Microsoft (care a preluat firma Visual Fox) au dezvoltat versiuni mult mai puternice, care utilizează conceptele programării orientate pe obiecte la descrierea structurii BD si a procedurilor de prelucrare, permițând realizarea unor BD extinse de imagini si sunete (multi-media).Sistemul dBase foloseşte acelaşi motor BDE (Borland Database Engine) ca limbajele Delphi, C++ Builder, Java Builder fiind compatibile la nivel de fisiere.

Cursul are un pronunțat caracter aplicativ, cu mărirea treptată a gradului de complexitate. Se urmăreşte formarea deprinderilor de proiectare a unor structuri care realizează legături între fişiere mergând până la structuri ierarhice si de tip rețea. Programele anexate au fost testate şi constituie modele pentru conceperea unor aplicații de BD de mare complexitate şi performanță. Lucrările urmăresc cursul Baze de date predat la secția de Calculatoare şi Tehnologia Informatiei din Universitatea Politehnica din Timişoara.

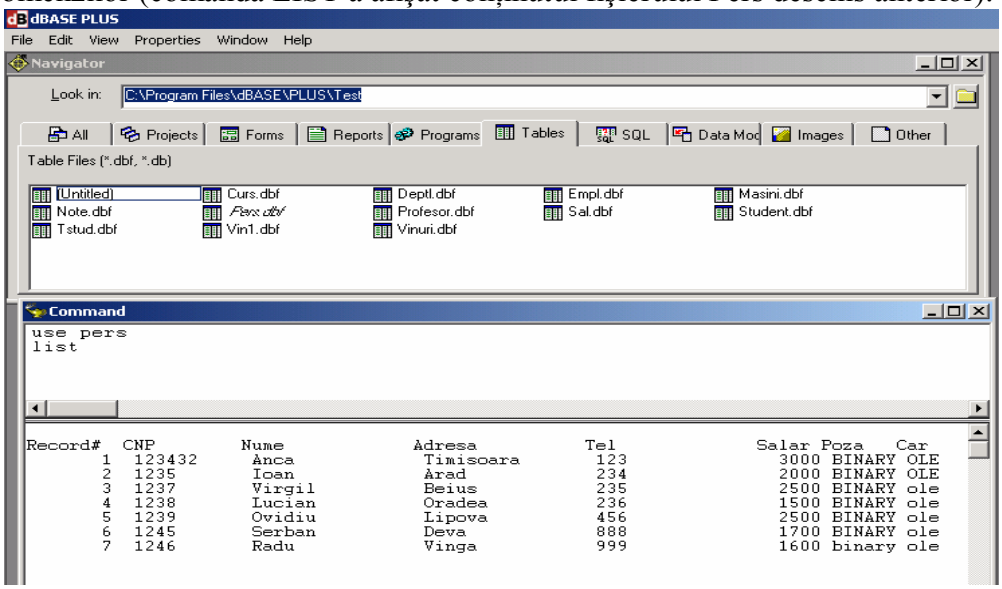
Timişoara, aprilie 2016 prof.univ.dr.ing.Ionel JIAN

CUPRINS

1. FIŞIERE de DATE şi EXPRESII

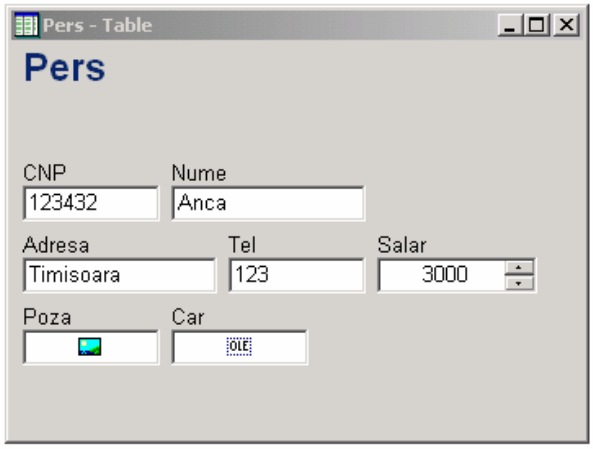
1.1. Lansare dBASE Plus şi sintaxă comenzi

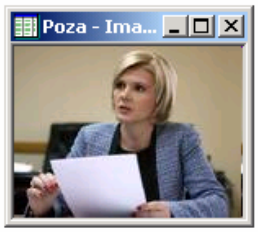
Sistemul de gestiune a bazelor de date dBase are o evoluție de peste 25 ani. Initial a fost lansat sub CP/M si apoi sub DOS unde a avut ultima versiune 4. Sistemul Visual dBASE 5 păstrează toate comenzile din DOS, adaugă programarea orientata pe obiecte si interfața grafica Windows. În Visual dBASE 7 şi dBase Plus se elimina comenzile specifice din DOS şi se acceptă numai programarea orientata pe obiecte.
Lansarea dBase Plus se face din Start/Programs/dBasePlus sau utilizând *icon-ul* de pe desktop.
dBASE Plus are o Interfața grafica prezentată mai jos care cuprinde:
- linie de meniu principal cu lista de funcții asociată si o bara de selecție rapida
linie de selecție a directorului curent printr-un click, în partea dreaptă (se va selecta un director din discul D şi directorul Student ,unde accesul este permis pentru lucru)
- fereastra Navigator care permite deschiderea fişierelor din directorul curent grupate pe clase(tabele, programe, form-uri, rapoarte, imagini, sunete, ..)
fereastră de comandă unde se scriu comenzile executate imediat (după ENTER);
- fereastră de date unde se afişează rezultatele obținute prin execuția comenzilor (comanda LIST a afişat conținutul fişierului Pers deschis anterior).



La lansarea programului din meniul View se activeaza ferestrele Navigator si Comand. Afişarea unui fişier de date se poate face şi din Navigator dacă se dă DblClick pe fişier.

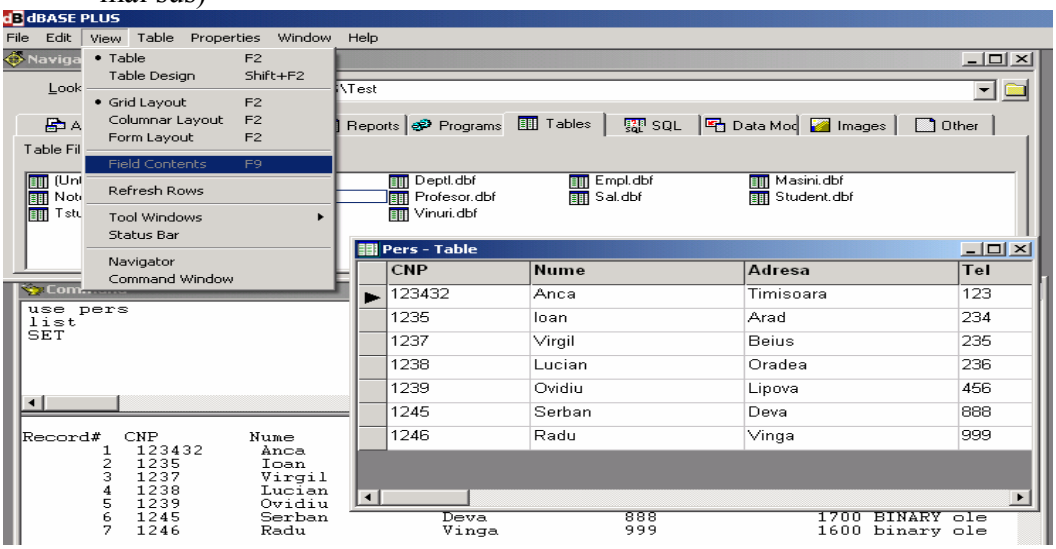
Dacă se dă DblClick pe câmpul poza, se va afişa poza persoanei.
Dacă este selectat form-ul Pers prin PgDown se trecela următoarea persoană, iar prin PgUp la precedenta, actualizând si poza.





Forma de afişare a înregistrărilor dintr-un fişier de date (.dbf) se modifică prin ***meniul view***:

- Grid Layout - afişare înregistrări din fişer ca tabel
- Column Layout - afişare înregistrari separat cu fiecare câmp pe rând nou
- Form Layout - afişare fiecare înregistrare separat într-o fereastră (ca mai sus)

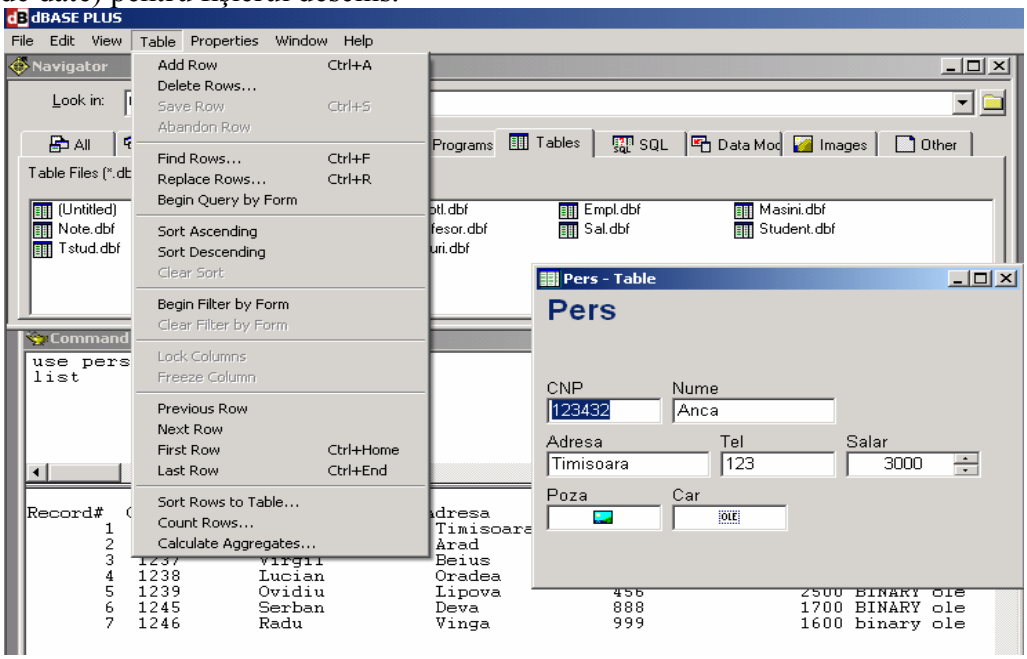


Exista 3 moduri de lucru:

- regim comanda - când comenzile introduse se executa imediat;

- regim program - când comenzile se plasează într-un program editat în prealabil cu orice editor text (editorul propriu NotePad) şi se lansează în execuţie folosind comanda DO;

- regim asistat - utilizând meniurile de pe ecran (pentru începători) permite execuţia unor funcţii si comenzi printr-un dialog lansat din meniul TABLE (adaugă, caută, şterge, editează si înlocuieşte înregistrări din fişierele de date) pentru fişierul deschis.



Comenzile limbajului Xbase (dBASE, Foxpro) pot apare în programe, când se executa automat o înlănţuire specificată, sau pot fi date în fereastra de comenzi, caz în care se executa imediat.

Formatul liniei de comanda este:

C-da arg1,arg2,... // comentariu.

- Comanda (verbul) poate fi precedată de spaţii, pentru ca programul sa fie mai inteligibil şi trebuie urmată de cel puţin un spaţiu. Argumentele se separa prin virgulă şi unele pot lipsi;
- Argumentele pot fi cuvinte cheie (clauze) şi atribute a căror sens semantic este specificat la fiecare comanda(Exemplu: DISPLAY FOR Nume =’Ion’).
- Orice linie de comanda se termina cu ENTER (cr).
- Comenzile pot conţine la sfârşit comentarii precedate de " //"sau “&&”;

- Liniile de comentariu încep cu "*" pe prima poziție (sau NOTE). Se admit si linii vide;
- Comenzile si clauzele sunt corecte dacă se dau cel puțin primele 4 caractere si se pot folosi litere mari sau mici.
- linie de comandă poate avea maxim 256 caractere şi poate fi scrisă pe mai multe rânduri terminate cu "**;**"

Ex: **DISP FOR** bursa>0 // se afişează studenţii bursieri.

Apăsarea tastei ESC întrerupe execuția oricărei comenzi sau program.

Un program întrerupt cu ESC poate fi reluat prin comanda RESUME, începând cu comanda următoare celei întrerupte.

1.2. Documentarea interactivă (HELP)

Sistemul dispune de o documentație succintă generală si câte o pagină pentru fiecare comandă sau funcție, care poate fi consultată prin comanda HELP:

HELP - afişează grupele de comenzi selectabile prin mouse sau sageti;

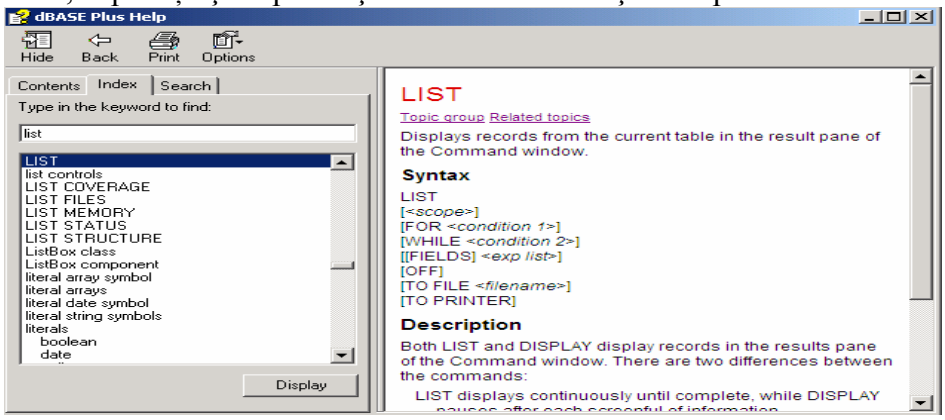
HELP C-DA - afişează sintaxa comenzii, explicaţii şi exemple de utilizare

Ex: HELP DISP -- afişează sintaxa comenzii Display

- Apăsarea tastei F1 lanseaza HELP-ul în **orice moment** al lucrului.

HELP-ul poate fi lansat si prin click pe Help din meniu, sau „**?**” în meniul rapid.

Documentatia este foarte completă şi sistematică, încât se poate lucra fără altă documentaţie de către un utilizator, care ştie conceptele principale şi elemente de principiu ale limbajului dBASE (exista peste 600 de comenzi si funcţii). Documentaţia este structurată ca în figură. Prin Index se poate căuta Help-ul pentru o comandă dată prin primele caractere. Se afişează sintaxa comenzii, explicaţii şi se pot afişa comenzi înrudite şi exemple de utilizare.



1.3. Crearea şi actualizarea fişierelor de date

O baza de date (BD) este o colecție de informații structurate păstrate în fişiere (tabele), intre care se stabilesc relații prin chei simbolice. În dBASE sunt mai multe tipuri de fişiere recunoscute, care sunt identificate prin numele pe care îl dă utilizatorul si extensia (completată automat), care specifica tipul. Se recomanda ca toate fişierele unei BD sa fie în acelaşi director.

Fişierele de date sunt cele mai importante si au extensia **.DBF** - **D**ata**B**ase **F**ile. Ele conțin informațiile de descriere a structurii înregistrărilor în primul articol. Toate înregistrările unui fişier .DBF au aceeaşi lungime, structura şi natură.Ele se referă la aceleaşi **entități** (persoane, studenți, cărți, maşini, materiale, clasament sportiv).

Pentru a defini structura unui fişier de studenți (STUD) vom specifica numele câmpurilor, tipul si lungimea lor, folosind exemplul de tabel.

STUD.DBF

CODS	NUME	ADRESA	DATA_N	BURSA	CAS	- nume câmpuri
5 car	20 car	15 car	8 car	7.2	1 car	- lățime câmpuri
char	char	char	date	numeric	logic	- tip câmpuri

CODS – este codul studentului, care cuprinde pe câte un caracter facultatea, secția, anul, grupa si numărul în grupa: MT437, CC429, HS215.
CAS – este de tip logic si indica daca este casătorit sau nu

în dBASE se admit urmatoarele tipuri de date:
C - CHARACTER - sir de maxim 254 caractere ASCII;
N - NUMERIC -numar cu maxim 19 cifre (l5 intregi si 9 zecimale)
F - FLOAT - numar în virgula flotanta(exponential-1.34E+2);
D - Date - de tip data în forma ll/zz/aa;
L - LOGICAL - tip logic cu valori permise Y, N, T, F
M - MEMO - contine texte de lungime variabila (rezumat,observatii,fisa,etc.)
BIN – pentru imagini şi sunete
OLE – pentru documente Word, Exel, Html, Paint, CorelDraw (de tip OLE)
T- Timestamp conține data si ora

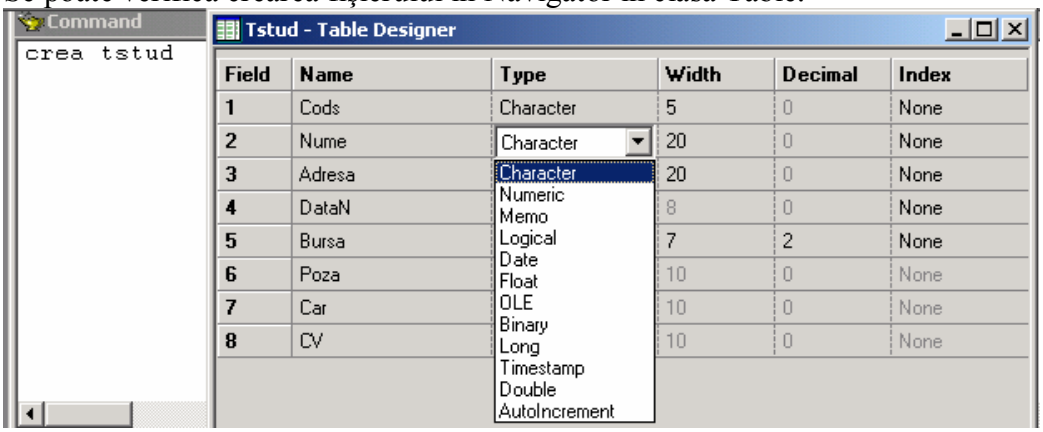
Crearea fişierului STUD.DBF crează înregistrare de descriere a structurii prin comanda:

CREATE STUD - va afişa un tabel într-o fereastră pe care îl completăm:

Name - Numele câmpului - din maxim 10 litere sau cifre si _
Type - Tipul câmpului - se tastează prima litera sau se selectează din listă
Width - Lăţimea câmpului - implicita pentru tipul LOGIC (1), Date (8), Memo (10)
Decimal - numărul de zecimale pentru tipul Numeric
Index - daca se crează sau nu un fişier index pentru acel câmp

Indicaţii de completare pentru fiecare coloana se dau în partea de jos a ecranului. Terminarea unei coloane se face cu ENTER sau TAB, iar revenirea la coloana precedenta cu SH-TAB sau click de mouse.

Terminarea descrierii se face prin închiderea ferestrei sau tastând ENTER în coloana goala a unui nou nume de câmp, CTRL/END sau CTRL/W. Apare atunci mesajul INPUT DATA RECORD NOW (Y/N), la care daca se răspunde Y. Se afişează macheta înregistrării pentru introducerea de date. Se accepta numai informaţii de tipul specificat, iar în caz contrar se sesizează eroare prin semnal sonor. Se introduc astfel înregistrări până când se tastează ENTER pe prima poziţie intr-o noua înregistrare, se da CTRL/END sau CTRL/W, sau se închide fereastra.
Se poate verifica crearea fişierului în Navigator în clasa Table.



Afişarea înregistrărilor din fişier în dBASE se face cu:

USE stud - deschide fişierul
LIST - afişează toate înregistrările si toate câmpurile.

LIST NUME, ADRESA, DATA_N - afişează câmpurile specificate

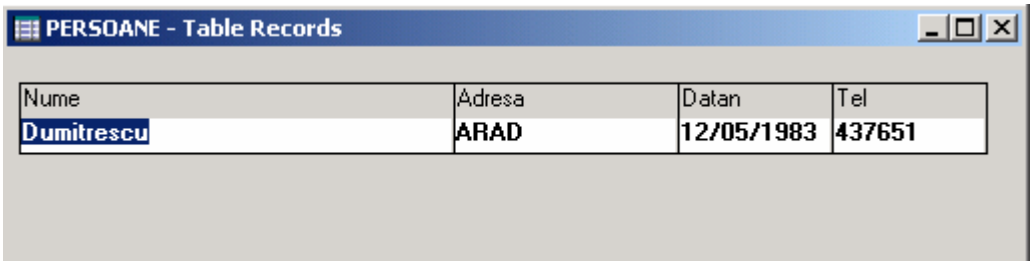
Record#	NUME	ADRESA	DATA_N
1	Dumitrescu Petre	Timisoara,23	01/12/1978
2	Ionescu ovidiu	Timisoara	23/09/1988
3	Popa Vasile	Timisoara	05/08/1976
4	Popescu Aurelian	Oradea	15/02/1989
5	Ionas Valentin Gh	Timisoara,23	29/05/1988
7	Cosma Liviu	Tr.Severin	02/11/1985
8	Alexandru Dan	Hunedoara	05/07/1983
10	Vacaru Cornel	Brasov	17/02/1950
11	Furdui Ion	Beius	23/05/1985
12	BALAN GHEORGHE	Sacalaz	23/05/1980
13	Albinaru Grigore	Baia Mare	12/02/1957

LIST OFF - afişează fără numărul înregistrării
SET HEAD OFF - nu mai afişează numele câmpurilor
CLEAR - şterge fereastra de date
DISPLAY ALL - afişează toate înregistrările
DISP STRUCTURE - afişează structura fişierului
GO 2 - poziţionează pe înregistrarea 2
DISP - afişează înregistrarea curenta (2)
Contorul de înregistrare rămâne poziţionat pe înregistrarea curenta si se modifica prin GO n sau prin unele comenzi ca LIST.
LIST - afişează toate înregistrările şi rămâne poziţionat la sfârşitul fişierului
DISP - nu afişează nimic fiindcă contorul de înregistrări a rămas poziţionat la sfârşit
GO 3 - readuce contorul pe înregistrarea 3
DISP NEXT 5 - afişează următoarele 5 inreg. si modifica contorul la 7 (ultima afişată)

La terminarea lucrului cu un fişier se închide cu
USE - închiderea fişierului deschis în zona activa;
USE STUD - redeschide fişierul STUD

Toate comenzile se refera la ultimul fişier deschis
GO 2 - poziţionează pe înregistrarea 2
DISP REST - afişează înregistrările până la sfârşit
DISP RECORD 4 - afişează înregistrarea 4

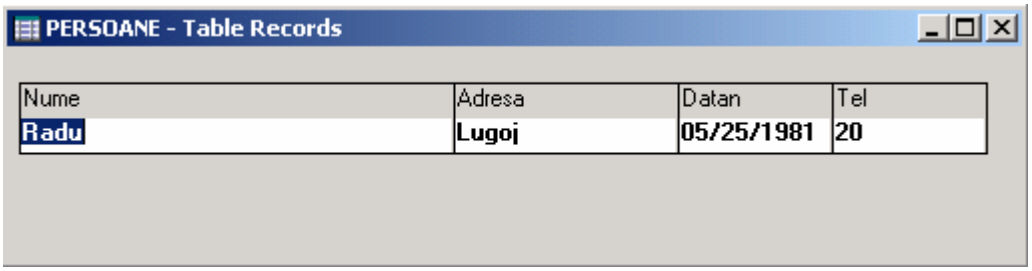
Adăugarea unor noi înregistrări la sfârşitul fişierului se face prin:
APPEND - adăugare înregistrare utilizând macheta din fereastra deschisă
Terminarea adăugărilor se face prin închiderea ferestrei.
Se pot accesa şi alte înregistrări folosind săgeţile verticale.



Pentru adăugare de înregistrări se recomandă utilizarea comenzii APPEND, care adaugă înregistrarea la sfârşit fără a afecta celelalte înregistrări din fişier. Comanda INSERT a fost dezactivată pentru ca mută înainte toate înregistrările care urmează după înregistrarea curentă consumând timp.

În tabelele bazei de date înregistrările nu trebuie să fie sortate într-o ordine dată. Parcurgerea ordonată a unei tabele se va face prin crearea de fişiere index după anumite câmpuri.

Modificarea în mod ecran a informaţiilor dintr-o înregistrare se face cu:
EDIT - modifică începând cu înregistrarea curenta (editare înregistrare)
EDIT Record 3 - permite modificarea înregistrării 3



Trecerea la alte înregistrări se face cu PGUP, PGDown.
Terminarea editării se face după ce s-a editat ultimul câmp.

Afişarea si modificarea în mod ecran a câmpurilor mai multor înregistrări începând cu cea curenta se face cu comanda:

BROWSE - afişează înregistrările ca un tabel

PERSOANE - Table Records

Rec	NUME	ADRESA	DATAN	TEL
1	Ion	Arad	07/07/1980	
2	dumitriu	bucuresti 45	10/14/2003	567
3	Radu	Lugoj	05/25/1981	20
4	Dumitru	Timisoara	06/15/1975	70
5	Vasile	Resita	08/16/1985	90
6	Dumitrescu	ARAD	12/05/1983	437651

Ştergerea înregistrărilor dintr-un fişier nu se face efectiv. Prin comanda DELETE se marchează pentru ştergere înregistrarea curentă. Înregistrările marcate pentru ştergere nu se vor afişa la o listare normală şi nici nu vor fi selectate la alte comenzi(Replace, Disp, Edit,..):

DELETE - se marchează cu * pe prima poziţie

DELETE FOR dataN <{01/01/1980}- marchează pentru ştergere înregistrările

LIST - înregistrările marcate nu se afişează

Se poate cere ca înregistrările marcate să nu mai poată fi selectate si prelucrate.

SET DELETE ON - invalidare selecţie înregistrări marcate

BROWSE - nu se mai afişează înregistrările marcate.

STUD - Table Records

Rec	CODS	NUME	ADRESA	DATA_N	BURSA	CAS	NRM
1	ac422	Dumitrescu Petre	Timisoara,23	01/12/1978	508.00	N	ar1
2	ac431	Ionescu ovidiu	Timisoara	23/09/1988	300.00	N	tm1
3	et312	Popa Vasile	Timisoara	05/08/1976	200.00	N	tm3
4	cc234	Popescu Aurelian	Oradea	15/02/1989	140.00	N	tm2
7	ac123	Cosma Liviu	Tr.Severin	02/11/1985	230.00	N	ar2
8	ac512	Alexandru Dan	Hunedoara	05/07/1983	154.00	N	ar3
10	et234	Vacaru Cornel	Brasov	17/02/1950	.	N	
11	ms342	Furdui Ion	Beius	23/05/1985	100.00	N	
12	ct339	BALAN GHEORGHE	Sacalaz	23/05/1980	85.00	N	

SET DELETE OFF - permite selectarea articolelor marcate

BROWSE - se afişează si articolele marcate

STUD - Table Records

Rec	Del	CODS	NUME	ADRESA	DATA_N	BURSA	CAS	NRM
1	☐	ac422	Dumitrescu Petre	Timisoara,23	01/12/1978	508.00	N	ar1
2	☐	ac431	Ionescu ovidiu	Timisoara	23/09/1988	300.00	N	tm1
3	☐	et312	Popa Vasile	Timisoara	05/08/1976	200.00	N	tm3
4	☐	cc234	Popescu Aurelian	Oradea	15/02/1989	140.00	N	tm2
5	☒	ac434	Ionas Valentin Gh	Timisoara,23	29/05/1988	.	N	tm5
6	☒		de sters		/ /	.	N	
7	☐	ac123	Cosma Liviu	Tr.Severin	02/11/1985	230.00	N	ar2
8	☐	ac512	Alexandru Dan	Hunedoara	05/07/1983	154.00	N	ar3
9	☒		PESCARU AUREL		/ /	.	N	
10	☐	et234	Vacaru Cornel	Brasov	17/02/1950	.	N	

Anumite înregistrări marcate pot fi revalidate cu:

RECALL 5 - revalidează înregistrarea 5
RECALL REST - revalidează toate înregistrările marcate până la sfârşit.
LIST

Eliminarea din fişier a înregistrărilor marcate se va face periodic (lunar), prin recopierea înregistrărilor valide în alt fişier utilizând comanda :

PACK - elimina înregistrările marcate din fişier
LIST - afişează noul fişier.
ZAP - şterge toate înregistrările (zero and pack) pastrand doar structura

Clauzele **ALL, REST, NEXT, RECORD n** pot fi utilizate şi pentru comenzile DELETE, RECALL si LIST. Ele definesc **noţiunea de Domeniu.**

Poziţionarea în fişier relativ la înregistrarea curenta se face cu **SKIP ± n**

GO 1
SKIP 3 - poziţionează pe înregistrarea 4
DISP - afişează înregistrarea curenta(4)
GO TOP - poziţionează pe primul articol
GO BOTTOM - poziţionează pe ultimul articol
DISP - afişează ultima înregistrare
SKIP -2 - poziţionează pe a 3-a înregistrare de la sfârşit.
DISP

Comenzile s-au prezentat în forma cea mai simpla pentru a se înţelege principala lor funcţie. Forma completa se găseşte în HELP.

După ce s-au încercat toate comenzile pe exemplul prezentat se va crea un nou fişier care să utilizeze toate tipurile de date la definirea câmpurilor. Se vor deschide si utiliza pe rând cele doua fişiere.

1. 4. Expresii, Operatori, Variabile, Funcţii

1. 4.1. Expresii şi operatori

Datele de diferite tipuri apar în constante, variabile si câmpurile fişierelor deschise.
Constantele admise sunt de tip:

CHARACTER	" SURUB", 'CAL',[ROATA]
NUMERIC	5.241, - 7.51, 62, 0.514
FLOAT	5.1E-3, -1.3E04, 6.1E3
LOGICAL	.T., .t., .Y., .F., .N., .n.,
DATE	{02/15/92},{07/29/51}

Variabilele de memorie (max.500) se identifica prin nume si se iniţializează în general prin comanda de atribuire:

FAC = 'MECANICA'	- variabila şir
R = 5.542	- variabila numerica
R1 = l.5 E-4	- variabila în flotant
C1 = .T.	- variabila logica ce memorează o condiţie
D_N = {05/27/51}	- variabila de tip data

Numele de câmpuri pot apare în expresii si reprezintă valoarea conţinută în înregistrarea curenta a fişierului activ.
IMPOZ = 0.2 * SALAR - salarul corespunzător înregistrării curente

Datele de acelaşi tip (constante, variabile, câmpuri) pot fi combinate în expresii folosind operatori specifici:

Operatori numerici

+, -, *, /	adunare, scădere, înmulţire, împărţire
** sau ^	ridicare la putere

Operatori de tip şir

+ concatenare de doua şiruri
- concatenare cu eliminare spatii dintre şiruri

Operatorii de relaţie servesc la compararea a doi operanzi numerici, de tip sir sau data, dând expresii logice simple (condiţii)

< , > , + , <> sau #, <= , >=

Doua şiruri sunt egale daca sunt identice, sau al doilea e cuprins în prima parte a primului.

$ - operator care verifica daca sir1 e cuprins în sir2
d1='mar' $ 'ian feb mar apr...nov dec' - va da rezultat .T.

Operatorii logici- realizează funcţiile NU, SI, SAU care permit generarea unor condiţii complexe:
.NOT.,.AND.,.OR.
Disp For Bursa >0 .AND. Bursa<200 .OR. CAS
La orice operaţie de atribuire se afişează si rezultatul. Evitarea acestor afişări in timpul programului se face cu:
SET TALK OFF

Calculul unor expresii si afişarea rezultatele începând cu un rând nou se face prin comanda " ? " ,care are sintaxa:
? exp1, exp2, exp3,... - expresiile pot fi de tipuri diferite

Ex: ? 5.8*(SIN(0.1) + COS(0.5)/2), NUME + ADRESA, 'SALAR:',SALAR
? 'NR.ZILE', {05/25/92} - DATA_N - rezulta număr de zile
? 'POPOVICI' = 'POP' - rezultatul va fi .T.
? 'POP' = 'POPOVICI' - rezultatul va fi .F.

SET EXACT ON - cere compararea şirurilor pe toata lungimea.
? 'RD' $ 'ARDEAL' - rezultat .T., sir1 exista în sir2
? 'POPA' > 'POPOV' - rezultatul este .F. si este dat de primul caracter diferit
? 'NUME:'+NUME, 'SALAR:', SALAR, 'IMPOZIT:',0.2*SALAR
? 'SALAR:'+SALAR - greşit, operanzi de tipuri diferite.

La calculul expresiilor se respecta ordinea operaţiilor si se poate forţa prin () ordinea dorita. Ordinea implicita este:
- operaţii aritmetice - ridicări la putere
- înmulţire sau împărţire
- adunare sau scădere
- operaţii logice - operatori de relaţie
- operatori logici .NOT. .AND. .OR.

Comanda **??** are acelaşi efect cu cea precedenta, dar afişarea se va face la poziţia cursorului fără a trece la linie noua.
Comanda **???** are aceeaşi sintaxa si efect tipărind valorile expresiilor.

Comanda STORE atribuie valoarea expresiei calculate mai multor variabile existente, sau nou create de acelaşi tip cu expresia:

STORE expr TO var1,var2,var3,...

STORE 1.5 TO A1,B2,C7,ALFA - memorează 1.5 în variabilele din lista
STORE 25200 TO SALAR - este greşit daca SALAR e nume de câmp
STORE {01/15/71} TO VDATA - memorează intr-o variabila data
STORE SALAR >150000 TO C3 - memorează rezultatul logic în C3

Comanda **DISP MEMO** - afişează variabilele de memorie
Comanda **RELEASE** - şterge unele variabile :
RELEASE ALL - şterge toate variabilele
RELEASE var1,var2,.... - şterge variabilele specificate în lista
RELEASE ALL LIKE generic - toate care respecta genericul
RELEASE ALL EXCEPT generic - fac excepţie cele din generic.
RELEASE ALL LIKE ?A* - şterge var. care au A pe poziţia 2

Comanda **SAVE** salvează variabilele pe un fişier pentru a putea fi refolosite si generează un fişier cu extensia .MEM.
SAVE TO fişier - salvează toate variabilele
SAVE TO fişier ALL LIKE generic - toate de forma generic
SAVE TO fişier ALL EXCEPT generic - toate exceptând cele de forma
Comanda **RESTORE** reface variabilele dintr-un fişier MEM
RESTORE FROM fişier - citeşte toate variabilele din fişier si şterge pe cele existente.
RESTORE FROM fişier ADDITIVE - adaugă variabilele din fişier la cele existente

1.4.2. Funcţii dBASE

In afara de constante, variabile si câmpuri în expresii se pot utiliza ca operatori funcţii, care pot fi de tip numeric, şir, data si logic. Pentru a indica tipul unei expresii vom folosi în continuare notaţiile:

expN - expresie numerica
expC - expresie de tip caracter (şir)
expD - expresie de tip data
expL - expresie logica (condiţie simpla sau complexa)
expF - expresie în flotant
exp - expresie de orice tip

Functiile numerice au ca rezultat un număr. Numărul de cifre şi de zecimale, care se utilizează la afişare se specifică prin cenzile:

SET PRECISION TO n - unde n=10-19 cu valoarea implicită 16

SET DECIMAL TO k - unde k =0-18 reprezintă nr de zecimale afişate (implicit 2)

- ABS(expN) - valoarea absoluta dintr-un număr
- INT(expN) - întregul din număr
- ROUND(expN,N2) - valoare rotunjita a numărului cu N2 zec.
- LEN(expC) - lungimea şirului specificat
- LOG(expN) - logaritm natural din număr real pozitiv
- LOG10(expN) - logaritm în baza 10 din număr real
- MOD(N1,N2) - restul împărţirii N1/N2
- SQRT(expN) - radical din număr
- EXP(expN) - calculează e la puterea x
- FIXED(expF) - conversie din flotant în numeric
- FLOAT(expN) - conversie din numeric în flotant
- SIN(expN) - sinus trigonometric (argument în radiani)
- COS(expN) - cosinus
- TAN(expN) - tangenta unghiului
- ACOS(expN) - arcul în radiani al cărui cos e dat de expN
- ASIN(expN) - arcsinus
- ATAN(expN) - arctangenta
- ATAN2(expNl,expN2) - arctan cunoscând sin=expN1,cos= expN2
- MAX(expN1,expN2) - maximul din cele doua valori
- MIN (expN1,expN2) - minimul din cele doua valori
- Pi() - valoarea lui Pi = 3,14
- RANDOM(K) - generează un număr aleator plecând de valoarea K
- VAL(expC) - valoarea numerica a şirului de cifre până la primul caracter nenumeric.
- AT(expC1,expC2) - indica pozitia (1... n) de unde sir1 este gasit în sir2
  (0 daca sir1 nu e inclus în sir2
- ASC(expC) -codul ASCII(1-256) a primului caracter din sir2
  ASC('A')=41H = 65
- ERROR() - codul numeric al erorii detectate

Funcţii care se refera la fişierul curent deschis:

- RECNO() - numărul înregistrării curente din fişierul activ

- RECCOUNT() - numărul de înregistrări din fişier
- RECSIZE() - lungimea înregistrării
- MEMORY() - memoria RAM disponibila în Kbyte
- DISKSPACE() - spațiul disponibil(liber) pe discul curent.

Funcțiile referitoare la lucrul cu imprimanta:

- PCOL() - numărul coloanei curente la imprimanta
- PROW () - număr rând curent la imprimanta

Funcțiile de tip şir dau ca rezultat un şir de caractere:

- CHR(expN) - caracter dat prin cod ASCII(1-128) utilizat pt. a transmite caractere de comanda, ce nu pot fi tastate (ESC,LF,CR,...)
- DBF() - da numele fişierului activ
- FIELD(expN) - da numele câmpului din poziția n
- LOWER(expC) - transforma în litere mici
- UPPER(expC) - transforma în litere mari
- LTRIM(expC) - suprima spatiile din stânga şirului
- TRIM(expC) - suprima spatiile din dreapta şirului
- OS() - numele sistemului de operare
- TIME() - ora curenta sistem hh.mm.ss
- SPACE(expN) - generează N spatii
- REPLICATE(expC,N) - multiplicarea şirului de N ori
- RIGHT(expC,N) - ultimele N caractere din şir
- LEFT(expC,N) - primele N caractere din şir
- SUBSTR(expC,N,L)- subşirul de lungime L începând cu caracterul N
- TYPE(expC) - tipul variabilei sau câmpului (C,N,L,D,M)
- STUFF(sir1,N1,L1,sir 2) - subşirul începând cu caracterul din poziția N1 de lungime L1 din sir1 se suprima si se înlocuieşte cu şirul 2.
- STR(expN,L,Dec) - converteşte în şir un număr - lungime L şi Dec nr. zecimale
- & - funcția substituție realizează adresarea indirecta

Ex: A1='ALFA'
ALFA=52
? &A1 - ne va afişa 52

- FKLABEL(N) - da şirul de caractere "memorat" în tasta funcționala Fn, care se realizează cu:

SET FUNCTION FKLABEL(n) TO 'şir'

SET FUNCTION FKLABEL(5) TO 'LIST' -- Apăsarea tastei F5 va genera c-da LIST

Funcțiile logice au ca rezultat True sau False

- BOF() -.T. daca s-a atins începutul de fişier (beginning of file)
- EOF() -.T. dacă s-a atins sfârşit fişier (end of file)
- FOUND() -.T. dacă articolul cu cheia data a fost găsit
- DELETED() -.T. dacă articolul curent e marcat pentru ştergere
- ISCOLOR() - dacă monitorul e color
- NETWORK() - dacă se lucrează pe rețea
- ISALPHA(expC) - .T. dacă primul caracter e litera
- ISLOWER(expC) -.T. dacă primul caracter e litera mica
- ISUPPER(expC) - .T. dacă primul caracter e litera mare
- FILE('fisier') - .T. dacă fişierul specificat exista.

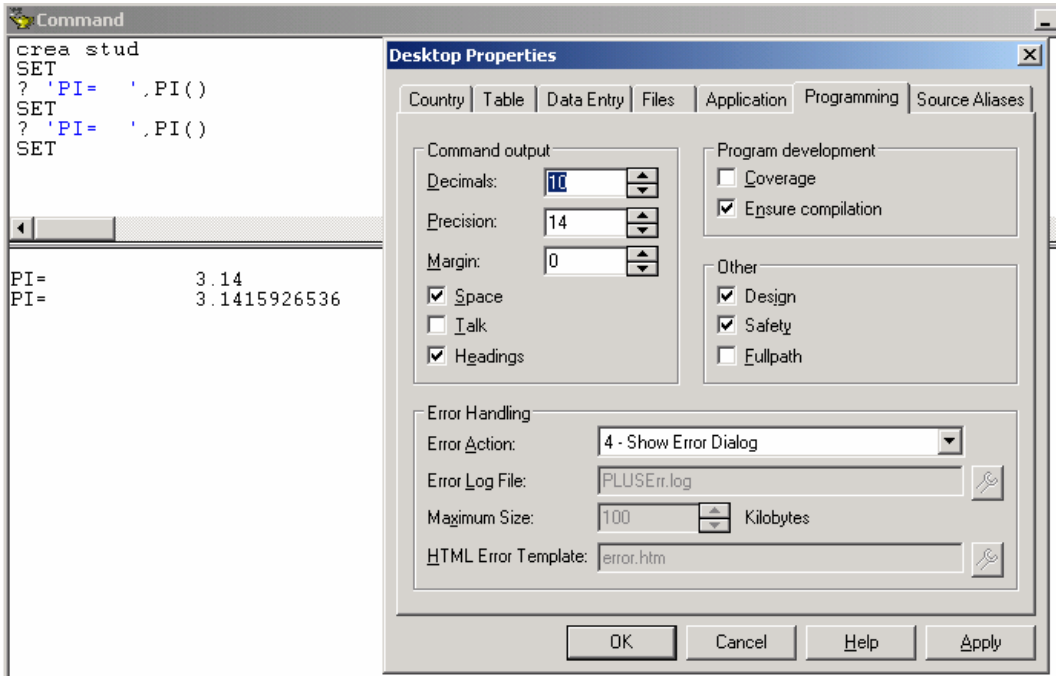
Funcțiile data considera data de forma ll/zz/aa (american)

- DATE() - data zilei curente din sistem
- YEAR(expD) - anul din data din forma 1940
- MONTH(expD) - luna din data (număr)
- DAY(expD) - ziua din luna (număr)
- CMONTH(expD) - numele lunii din data (engleza)
- CDOW(expD) - numele zilei din săptămâna
- DOW(expD) - numărul zilei din săptămâna
- CTOD(expC) - transformă un şir în data - CTOD('02/15/72')
- DTOC(expD) - transformă o data în şir - DTOC(date())
- LUPDATA() - data ultimei actualizări a fişierului activ
- Time() - returnează timpul curent
- TtoC(t) - conversie timp în sir de cactere de forma HH:MM:SS
- DateTime() - data si timpul din acel moment
- DTtoD(x) - retine data dintr-o valoare de tip Date Time
- DTtoT(x) - retine timpul dintr-o valoare de tip Date Time
- DtoDT(x) - transforma variabila date în DateTime
- DTtoC(x) - conversie din DateTime în sir de caractere
- DTOR(x) - Conversie grade în radiani (DegreeTo Radian)

La tipul dată poate să apară sau nu si secolul:

SET CENTURY ON - determina anul pe 4 cifre
SET CENTURY OFF - anul cu 2 cifre (implicit)
Modul de afisare al datei poate fi modificat prin:
SET DATE AMERICAN - implicit ll/zz/aa
MDY - ll/zz/aa (month, day, year)
DMY - zz/ll/aa
YMD - aa/ll/zz

Setările sunt foarte multe şi se pot afişa din Help prin Search. Toate setările se pot modifica simplu prin fereastra **Desktop Properties** activată prin **comanda SET** sau **Properties** din din bara de meniu principal. Prin tag-urile din partea de jos a ferestrei se alege categoria de setări.



Folosind funcţiile descrise se va verifica modul de funcţionare utilizând comanda **?** de afişare valorile unei liste de expresii astfel:
? DATE(), CMONTH(DATE ()), CDOW(DATE()), YEAR(DATE())
Afişează: 25/02/2014 February Tuesday 2014

Set Decimal to 10 - am setat numărul de cifre zecimale pentru precizie PI.
? sin(0514), 'Pi=', PI(), 'Pi=', 4*ATAN(1.)
Afişează: -0.8462043419 Pi= 3.1415926536 Pi= 3.1415926536

2. CONSULTAREA SECVENTIALA SI INTERACTIVA A BD

2.1. Selecţie înregistrări prin domeniu şi condiţie

Comanda DISP permite afişarea numai anumitor înregistrări din fişierul deschis.
Clauzele care delimitează **un domeniu:**

DISP - afişare, înregistrarea curenta
DISP NEXT n - afişează următoarele n înregistrări
DISP ALL - afişează toate articolele din fişier
DISP REST - afişează de la înregistrarea curenta la sfârşit
DISP RECORD n - afişează înregistrarea n din fişier

Aceste afişări nu ţin cont de conţinutul înregistrărilor ci numai de poziţia lor în fişier.

Putem selecta pentru a fi prelucrate numai înregistrările care îndeplinesc o anumita condiţie prin utilizarea clauzei **FOR cond:**

DISP FOR RECNO() < 15 - afişează înregistrările 1 la 15
DISP FOR DELETED() - numai înregistrările marcate pentru ştergere
DISP FOR .NOT.DELETED() - numai înregistrările nemarcate
DISP FOR CAS - afişează datele studenţilor căsătoriţi
DISP FOR BURSA>0 - afişează datele studenţilor bursieri
DISP FOR NUME='POP' - studenţii a căror nume încep cu 'POP'
DISP FOR DATA_N>{10/20/65} - studenţii născuţi înainte de 20 oct 1965.
DISP FOR BURSA>0.AND.CAS - studenţii căsătoriţi si bursieri

Daca se utilizează **clauza FOR domeniul implicit este ALL**. Se poate utiliza si un alt domeniu pe care sa se aplice condiţia: DISP NEXT 10 FOR CAS - din următoarele 10 articole se afişează studenţii căsătoriţi.
Utilizând o codificare corecta a studenţilor CODS se pot selecta anumite grupe:
DISP FOR CODS='CC4' - studenţii Fac .Calculatoare, secţia calculatoare anul 4
DISP FOR CODS='H' - toţi studenţii facultăţii Hidrotehnica
Folosind codul complet de 5 caractere se selectează un student.

Cu aceste precizări sintaxa extinsa a comenzii DISPLAY este:

DISP [dom] [lista_expr][FOR cond][OFF][TO print]

DISP 'NUME:', NUME, 'ADRESA:', ADR, 'CASATORIT'FOR CAS

Precizam ca la clauza FOR trecerea la înregistrarea următoare se face automat pentru întregul domeniu chiar daca condiția nu este îndeplinită. În locul clauzei FOR se poate folosi şi clauza WHILE, caz în care operația de selectare **se opreşte la prima înregistrare care nu îndeplineşte condiția:**
DISP WHILE CAS - se opreşte la primul student necăsătorit.

Clauza FOR şi domeniu se poate utiliza la multe comenzi:
LIST, DELETE, RECALL, EDIT, REPLACE, APPEND, COPY, ...

Comanda REPLACE permite modificarea conținutului unor câmpuri din înregistrarea selectata având sintaxa:

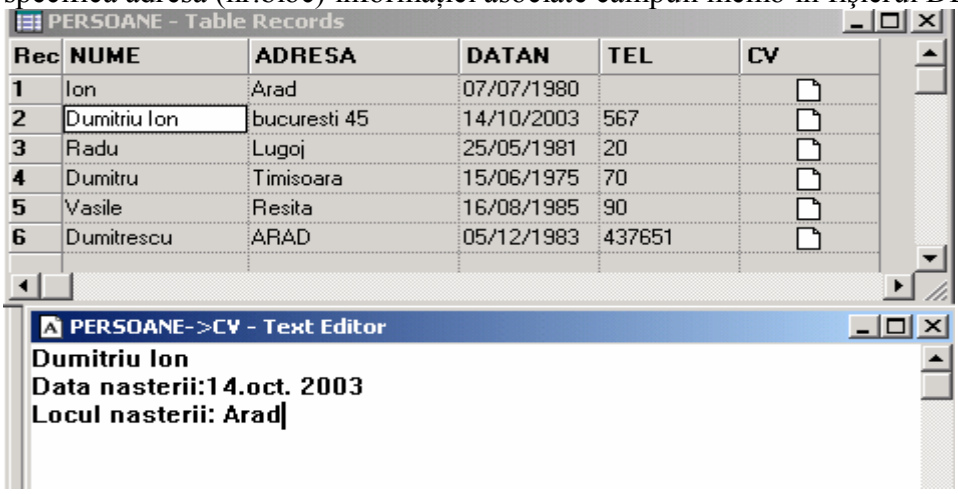
REPL [dom] cimp1 WITH expr1,cimp2 WITH expr2,...[FOR cond]

REPL ALL BURSA WITH BURSA * 1.2 - mărirea burselor cu 20%

Precizam ca modificarea unor câmpuri nu se poate face cu STORE, care se refera la variabile de memorie.

2.2. Câmpuri MEMO

Câmpurile MEMO conțin texte de lungime variabila (comentarii, observații, rezumate), care sunt păstrate în fişiere tip DBT cu acelaşi nume cu fişierul DBF, care conține câmpul. Câmpul MEMO este un pointer pe 10 caractere, care specifica adresa (nr.bloc) informației asociate câmpuli memo în fişierul DBT.



Fiecare informaţie MEMO are rezervat cel puţin 512 octeţi (un sector disc). La APPEND,EDIT sau BROWSE intrarea în câmpul MEMO se face dând DblClick pe câmp. Se deschide o fereastră şi editarea se face folosind funcţiile normale. Terminarea editării câmpului MEMO CV se face prin închiderea ferestrei.

Un câmp MEMO se poate memora intr-o variabila si prelucra corespunzător:
STORE REZUMAT TO VREZ - câmp MEMO REZUMAT memorat
REPL REZUMAT WITH SUBSTR (VREZ,1,55)+'ROMAN POLITIST'

Câmpurile MEMO nu pot intra direct în expresii dar se pot afişa
? 'Titlul:'+TITLU - titlul cărţii din înregistrarea curenta
? 'Rezumat: ',REZUMAT - afişare câmp memo REZUMAT sau
DISP 'Titlul: '+TITLU
DISP 'Rezumat: ',REZUMAT

Daca la afişare nu se specifica explicit numele câmpului memo conţinutul lui nu se afişează ci doar cuvântul "memo":
LIST - nu afişează câmpul memo
LIST titlu, autor, rezumat - afişează si conţinutul câmpului memo

Afişarea unui câmp memo se face pe linii de lungime fixa (implicit 50 caractere),care se stabileşte prin comanda:
SET MEMOWIDTH TO expN - valoare intre 8 si 255

Funcţia **MEMLINES(cimp_memo)** da numărul de linii al câmpului memo al înregistrării curente, de lungimea stabilita prin comanda anterioară. Afişarea unei singure linii se face prin funcţia:
MLINE(cimp_memo,N) - returnează linia N din câmpul memo
? MLINE(REZUMAT,5) - se afişează linia 5 din REZUMAT

2.3.Utilizare programe şi dialog cu utilizatorul

Un program dBASE conţine o secvenţa de comenzi memorate intr-un fişier cu extensia PRG. Fişierul program se poate realiza cu orice editor de program (TotalComandor, Word - nondocument, etc) sau cu editorul propriu (NotePad) lansat prin comanda MODIFY COMMAND care deschide o fereastră în care se scrie programul.

MODI COMM PROG1 - creează sau editează fişierul PROG1.PRG

Terminarea editării se face prin închiderea ferestrei.

DO PROG1 - compilează programul sursă PROG1.PRG generând un program obiect PROG1.DBO care se lansează în execuţie.

RETURN - comanda de revenire în programul chemător

CANCEL - comanda de revenire în dBASE

QUIT - comanda de revenire în MS-DOS

SUSPEND - opreşte execuţia programului (similar cu ESC), iar reluarea lui poate fi făcută prin comanda **RESUME**

COMPILE PROG1 - compilează PROG1 fără a lansa fişierul DBO creat.

Un program poate chema un alt program pentru o programare modulară. Pot fi utilizate si subprograme cu parametrii.

Intr-un program pot apare orice comenzi, dar nu se recomanda APPEND, DISP, EDIT, BROWSE, LIST, CREATE, care sunt folosite în general în mod comanda pentru încercări si depanare, în modul comanda.

Pentru ca programul sa fie mai general se utilizează comenzi de dialog cu utilizatorul prin tastatură si monitor, care creează sau modifica variabile.

In versiunile anteriore de dBase până la Visual dBase 5.5 existau comenzile **ACCEPT si INPUT** pentru introducerea datelor de la tastatura. Aceste comenzi au fost eliminate în dBase Plus, pentru a se utiliza numai programarea orientată pe obiecte Windows.

****

ACCEPT 'mesaj ' TO varC - afişează mesajul explicativ si aşteaptă introducerea unui şir de caractere terminat cu ENTER ce se va memora în variabila (şirul nu se delimitează cu apostroafe).

ACCEPT 'Nume student: ' TO VNUME - cere introducerea numelui

****

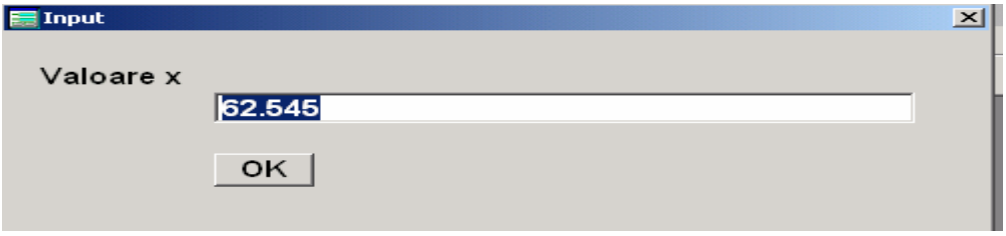
Folosirea interfeţei grafice fiind dificilă pentru început, vom folosi o **funcţie ACCEPT(),** care permite introducerea unui şir de caractere într-o variabilă. Ca parametru se da mesajul explicativ afişat în fereastra. Fişierul care conţine procedura, cu numele ACCEPT trebuie să fie copiat în directorul în care se găsesc programele aplicaţiei.

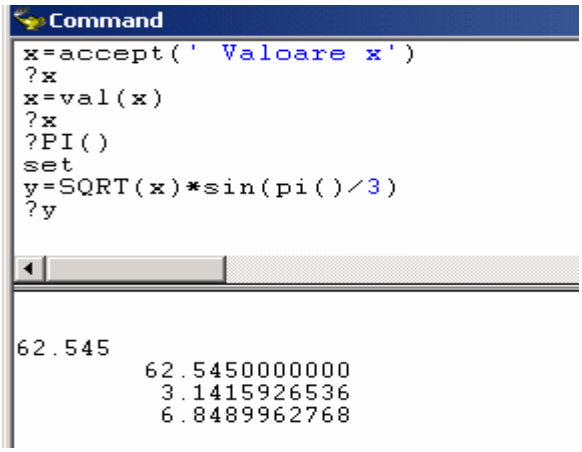
x= ACCEPT('Valoare x') // fereastra pentru introducerea unui sir

? x // afiseaza valoarea lui x introdusa

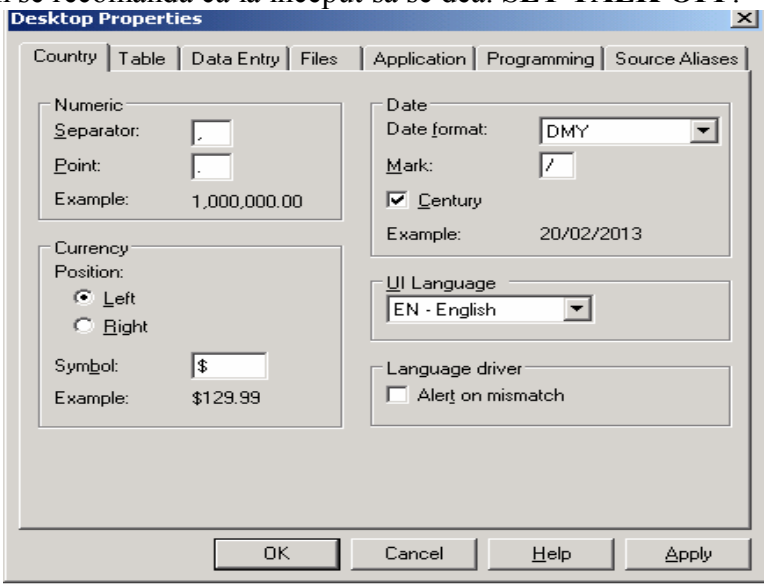
x=val(x) // converteste x in numar

? x // afiseaza valoarea lui x ca numar real





Pentru a nu apare pe ecran si alte mesaje decât cele specificate explicit în program se recomanda ca la început sa se dea: **SET TALK OFF**.



Comanda SET permite modificarea parametrilor aplicației şi in exemplul dat afişarea valorii lui PI se face pe 14 poziții cu 10 cifre zecimale (implicit 2 zecimale).

Daca se aşteaptă un singur caracter (pentru selecţia unei funcţii program de executat, pentru a pune în aşteptare programul, se utilizează comanda WAIT.

WAIT - afişează " tastaţi orice caracter" si aşteaptă apăsarea unei taste
WAIT 'mesaj' TO varC - afişeaza mesajul si memorează caracterul tastat în variabila de tip şir pe care o creaza daca nu exista.

WAIT ' Continuati? D/N ' TO r // introduce caracterul tastat in variabila r

IF r='D'
....secventa de continuare
ENDIF

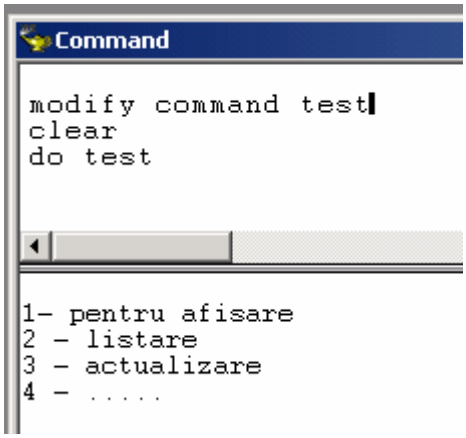
Exemplu de program care caută un student după nume:

MODI COMM CAUTS - deschide fisierul program Cauts.prg
* Creare fişier program (CAUTS.PRG)
SET TALK OFF - eliminarea mesaje suplimentare
CLEAR - şterge ecranul
USE STUD - deschide fişierul studenţi
? 'Programul afiseaza datele unui student cu nume dat'
?
VNUME =ACCEPT('Nume student') - cere nume student cu maxim 30 caractere
VNUME=TRIM(VNUME) -suprima spatiile din dreapta
DISP FOR Nume = VNume - afiseaza datele studentului(studentilor)
USE - inchide fisierul
RETURN - revenire în dBASE
* Se va inchide fereastra pentru terminare editare

DO CAUTS - lansare program dupa compilare

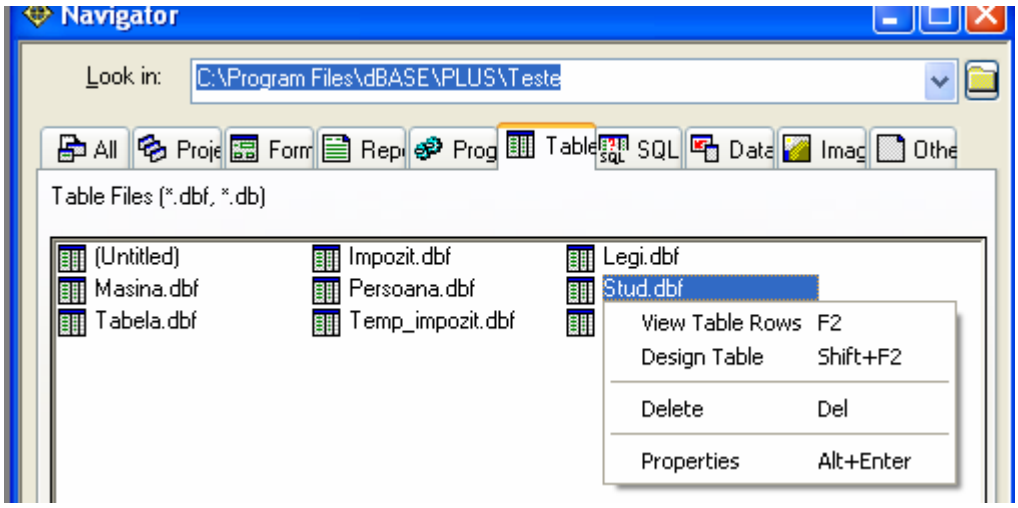
Comanda **TEXT .....ENDTEXT** delimitează un text afişat in fereastra de date pe mai multe randuri incepind cu poziţia curenta a cursorului.

```
TEXT
1- pentru afisare
2 - listare
3 - actualizare
4 - .....
ENDTEXT
```



Modificarea structurii unui fişier DBF se poate face dacă fişierul a fost deschis:

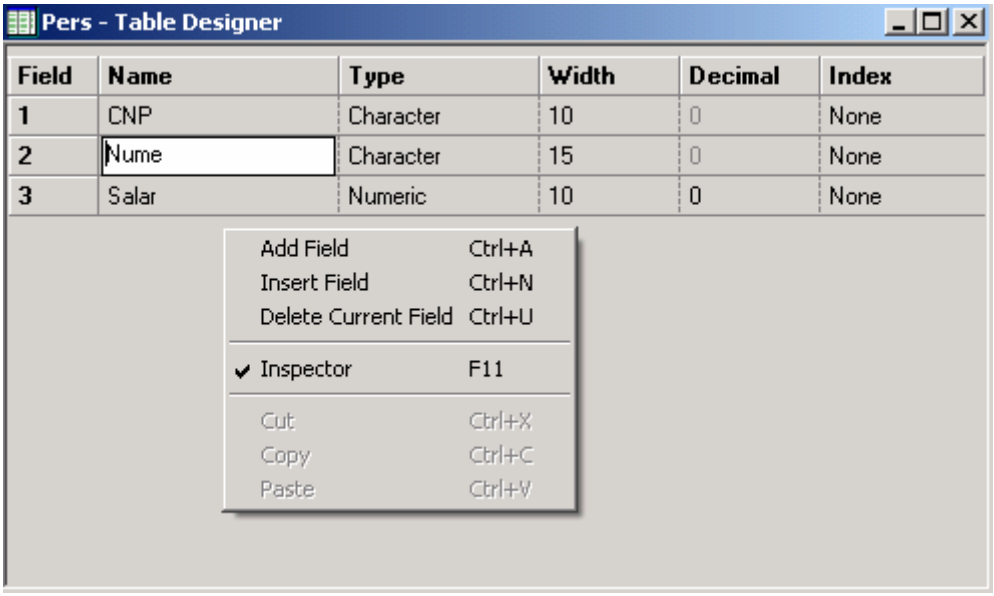
- folosind comanda **MODIFY STRUCTURE**
- **click dreapta** pe numele fişierului in **Navigator** si se selecteaza **Design Table** (fişierul trebuie să fie închis).



USE STUD Exclusive
MODI STRU - se afişează structura fişierului deschis într-o fereastră
Se pot face orice modificări a structurii în mod ecran:

- adăugare câmpuri (adăugați câmpul MEMO CV pentru caracterizare student)
- modificări lungimi sau tipuri
- ştergere câmpuri cu CTRL/U
- inserare câmpuri cu CTRL/N
- schimbare ordine câmp prin ştergere, plus inserare sau adăugare câmp
- terminarea modificărilor se indica prin CTRL/END sau prin închidere fereastră.

Tipul modificării de structură se poate alege si prin click dreapta pe fereastra de afişare a structurii.



După modificarea structurii automat se restructurează şi informațiile din articolele fişierului, prin recopiere. Câmpurile nou adăugate se considera "albe" (zero sau spatii). **Dacă se schimba numele si poziția unui câmp se consideră câmp nou si informațiile corespunzătoare se pierd.**

Introduceți informații în câmpurile CV de tip MEMO din fiecare înregistrare utilizând EDIT sau BROWSE. Observați că pe câmpurile MEMO completate apare un **A.**

Secvența următoare afişează numele fiecărui student selectat, adresa si prima propoziție din caracterizare delimitata de punct.

```
* Afisare Nume, Adresa si prima propozitie din caracterizare
USE STUD
VNume =TRIM(ACCEPT('Nume student: '))
LOCATE FOR Nume = VNume        // pozitionare pe inregistrarea indicata
DISP NUME,ADRESA,CV            // afisare date despre student
STORE CV TO CVAR          // memorare caracterizare intr-o variabila
N=AT('.', VCAR)                    //determina poziția punctului în caracterizare
? 'CARACTERIZARE: '+ substr(cvar,1,n)    // afişează o propoziție
USE                        //inchide fisierul
RETURN                     //terminare program
```

Căutarea secvențiala după un criteriu (condiție) se poate face si utilizând comanda LOCATE care are sintaxa:

LOCATE [domeniu] FOR conditie

Pentru a nu repeta condiția la căutarea următoarei înregistrări, care îndeplineşte condiția, se utilizează **CONTINUE**.

```
LOCATE FOR DATA_N > {10/25/70}   - determina prima înregistrare
DISP             - afiseaza inregistrarea gasita ce indeplineste conditia
CONTINUE    - cauta urmatoarea inregistrare cu aceeasi conditie
DISP             - afişează următoarea înregistrare găsită
```

Căutarea se continua până la atingerea sfârşitului de fişier.

In cazul în care condiția se dă intr-o ramura a programului prin LOCATE, ea poate fi folosita ulterior pe o ramura comuna cu CONTINUE.

Filtre de câmpuri din fişier -operație relațională de **proiectie**

Dacă într-o aplicație se cer frecvent numai anumite câmpuri pentru a fi afişate sau prelucrate intr-o ordine dată, aceasta se poate specifica prin:

```
SET FIELD TO lista_cimpuri      && celelalte campuri nu mai pot fi utilizate
SET FIELD ON       && validare utilizare lista de cimpuri definite
SET FIELD  OFF     && invalideaza utilizarea listei de cimpuri
```

SET FIELD TO NUME,BURSA,ADRESA && definire lista cimpuri
DISP && afiseaza doar cimpurile specificate
SET FIELD TO && desactivare lista de cimpuri)
SET FIELD TO cimp/R -cimpul se utilizeaza doar în citire
SET FIELD TO cimp_calculat = expr && definire cimp calculat

Câmpurile adăugate în lista de câmpuri pot fi din orice fişier deschis (prin prefixare) si se refera la toate comenzile ce admit lista de câmpuri (DISP,LIST, CHANGE, COPY TO,COPY STRU).

Filtre de înregistrări (rânduri din tabele)

O condiție de selecție a înregistrărilor din fişier, se poate păstra prin **SET FILTER**, încât clauza FOR nu mai trebuie sa fie specificată. In continuare la toate comenzile ce pot conține FOR se selectează numai înregistrările specificate prin "filtru" (inclusiv EDIT si BROWSE). Selectarea înregistrărilor se face secvențial parcurgând tot fişierul şi nu se recomandă în aplicații.

SET FILTER TO conditie && definire conditie filtru
SET FILTER TO && deactivare filtru pentru fisierul curent
SET FILTER TO FILE fisier.QRY -validare fisier conditii

2.4.Comenzi de calcul secvențial în fişierele de date

Exista un set de comenzi statistice (de tip agregat) care se execută pe un grup de înregistrări din fişier selectate prin clauza FOR. Aceste comenzi se vor evita fiindcă parcurg secvențial tot fişierul.

Pentru a număra înregistrările din fişier care îndeplinesc o condiție data se foloseşte COUNT care parcurge secvențial fişierul sau domeniul specificat:

COUNT [domeniu] [FOR cond] [TO varN]

COUNT && numara inregistrarile din fisier si afiseaza valoarea
COUNT FOR VCODS='CC4' TO NR4 && numără câti studenți sunt în anul 4, secția Calculatoare si memorează în variabila NR4;

Comanda SUM si AVERAGE calculează sume (totaluri) si respectiv medii, asupra unor câmpuri numerice din înregistrările selectate.

SUM [domeniu] lista_exprN [FOR cond] TO lista_varN
AVERAGE

SUM BURSA FOR CODS='C' TO CBURSA && calculează suma burselor studenților de la fac. Calculatoare si o memorează în CBURSA
AVERAGE BURSA FOR CODS = 'C' TO MCBURSA && calculează media burselor

Asupra unui fişier de personal se poate face calculul sumei salariilor, impozitelor, contribuţie asigurări sociale, fond şomaj, etc.

```
Vsec=ACCEPT( 'sectia: ')      -introducere cod sectie
SUM SALAR,SALAR*0.25,SALAR*0.2,SALAR*0.04 ;
        FOR SECTIA=VSEC TO TS,TI,TC,TS

? 'SECTIA:' + VSEC
? 'TOTAL SALARII= ',TS
? 'TOTAL IMPOZIT= ',TI
? 'TOTAL contrib.asig.sociale= ',TC
? 'total fond somaj= ',ts
```

Daca mai multe totaluri de acest gen sunt necesare., fişierul este parcurs de mai multe ori. Aceste comenzi se pot combina în CALCULATE:

CALCULATE [domeniu] lista_optiuni [FOR cond] TO lista_var

Lista de opţiuni poate cuprinde elemente de forma:

SUM(expN) - expresie sau câmp din care se calculează suma totala
AVG(expN) - expresie din care se calculează media
CNT() - numărul elementelor selectate
MAX(expN) - valoarea maxima determinata
MIN(expN) - valoarea minima determinata
STD(expN) - deviatia standard pentru valorile unui câmp al tabelei
VAR(expN) - varianta valorilor unui câmp numeric dintr-o tabelă

Asupra aceluiaşi fişier de persoane se poate determina intr-o singura trecere numărul salariaţilor dintr-o secţie, salariul maxim si minim, media si suma salariilor.

```
CALCUL
CNT(),MAX(SALAR),MIN(SALAR),AVG(SALAR),SUM(SALAR);
        FOR SECTIA = VSEC TO NR,SMAX,SMIN,MSAL,TSAL
```

În lista de opțiuni mai pot intra si alte funcții statistice si contabile specifice. Amânunte de utilizare în HELP.

```
CALC VAR(bursa) , STD(bursa) to var, standv

var(bursa)    std(bursa)
  17401.61        131.92
```

Se va evita utilizarea comenzilor de calcul secvențial de tip agregat in programe, fiindcă fiecare face o parcurgere completă a fişierului.

3. PROGRAME CICLICE SI RAMIFICATE

3.1.Comenzi procedurale

Pentru realizarea programelor ramificate si ciclice există comenzile IF, SCAN, DO WHILE, CASE.

Comanda SCAN parcurge toate articolele din fişier, sau din domeniul specificat si executa comenzile din secvența care urmează numai pentru cele care îndeplinesc condiția data prin FOR. Sfârşitul secvenței asociate se indica prin ENDSCAN.

Comanda **LOOP** forțează reluarea secvenței SCAN de la început.
Comanda **EXIT** forțează ieşirea din secvența SCAN.

SCAN [domeniu] FOR conditie
---------- - secventa comenzi executate daca conditia este indeplinita
----
[LOOP] && reluare fortata secventa SCAN
[EXIT] && iesire fortata din SCAN (dupa ENDSCAN)

ENDSCAN && sfirsit secventa SCAN (executa SKIP)

La terminarea secvenței SCAN se trece automat la articolul următor din fişierul activ (fără sa se dea SKIP în program) si se reia ciclul daca nu s-a atins EOF.

Comanda IF verifica îndeplinirea unei condiții si are sintaxa

IF conditie
---------- - secventa comenzi pentru conditie indeplinita

[ELSE]
---------- - secventa pentru conditie neindeplinita
ENDIF - sfirsit comanda IF

Comenzile de ciclare pentru o secvență de comenzi sunt:

- cu numar fix de cicluri **FOR .....NEXT**
- cu numar variabil de cicluri cu verificare condiție
  - - la inceputul ciclului **DO WHILE** conditie.....**ENDO**
  - **-** la sfârşitul ciclului **DO .....UNTIL** condiție

Comanda FOR execută secvența care urmează şi care se încheie cu **NEXT** modificând valoarea variabilei K de la K1 la K2 cu pasul T:

FOR K=K1 TO K2 STEP T
..... secvența de comenzi care utilizează K
NEXT K

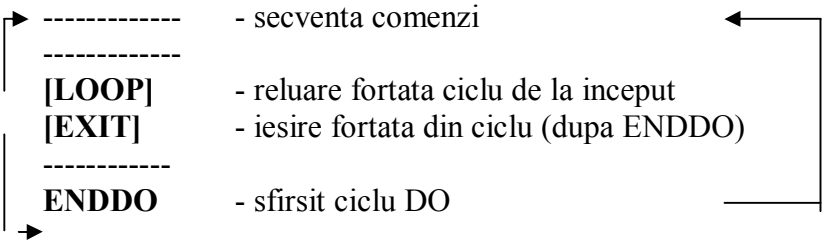
*** PFor Program** de calcul pentru suma unui sir de numere

```
K1=VAL(accept('Introduceti K1'))            && conversie valoare introdusa
K2=VAL(accept('Introduceti K2'))
T=VAL(accept('Introduceti pasul'))   && pasul
S=0
FOR k=k1 TO k2 STEP t
        S=S+K
NEXT  k        && se mareste variabila k cu pasul t
? 'Suma numerelor de la ', k1, 'la  ', k2, 'cu pasul ', t, ' este: ', S
Return
```

Suma numerelor de la 4 la 87 cu pasul 3 este: 1246

Comanda DO WHILE conditie, execută secvența de comenzi care urmează atât timp cât condiția este îndeplinita, reluând secvența de la început la atingerea lui ENDDO. Comanda efectuează un număr variabil de cicluri cu verificarea condiției la începutul ciclului. În fişier se pleacă de la înregistrarea curentă.

DO WHILE conditie



Precizăm că dacă condiția nu este îndeplinită secvența DO nu se mai reia, spre deosebire de SCAN, care neglijează înregistrările în care condiția nu e îndeplinită şi continuă cu următoarele înregistrări. Comanda DO nu este legată de fişier, încât dacă se prelucrează înregistrări, trecerea la următoarea înregistrare trebuie făcuta cu SKIP şi condiția trebuie să conțină testarea sfârşitului de fişier.

Pentru a afişa toți studenții bursieri cu DO WHILE trebuie să folosim secvența:

```
GO 1
DO WHILE .NOT. EOF() && Se parcurge fisierul pana la sfarsit din pozitia curenta
  IF bursa> 0
        DISP            && afisare stundent daca are bursa
  ENDIF
SKIP            && trece la inregistrarea urmatoare
ENDDO           && se reia ciclul daca nu este sfarsit de fisier – EOF()
Return
```

Secvența următoare este greşită fiindcă se iese din ciclu la primul student care nu are bursă:

```
DO WHILE bursa > 0
        DISP
        SKIP
ENDDO
```

Programul prezentat pentru FOR se va scrie cu DO WHILE sub forma:

```
* PDo Program de calcul pentru suma unui sir de numere
K1=VAL(accept('Introduceti K1'))            && conversie valoare introdusa
K2=VAL(accept('Introduceti K2'))
T=VAL(accept('Introduceti pasul'))   && pasul
S=0
K=k1
DO WHILE K <= k2  && se executa atata timp cat conditia  e indeplinita
        S=S+K
        K= K+t          && se mareste variabila k cu pasul t
ENDDO       && iesire din ciclu daca nu este indeplinita conditia
? 'Suma numerelor de la ', k1, 'la  ', k2, 'cu pasul ', t, ' este: ', S
Return
```

Suma numerelor de la 7 la 45 cu pasul 3 este: 325

Comanda DO ..... UNTIL condiție, execută secvența de comenzi care urmează şi verifică îndeplinirea condiției la sfârşitul ciclului. **Dacă nu este îndeplinită condiția se reia secvența** de la început, iar dacă este îndeplinită se abandonează ciclul. Secvența se execută la primul ciclu chiar dacă este îndeplinită condiția.

```
DO              -- secventa se executa cat timp nu este indeplinita conditia
.....
..... Secventa de comenzi
.....
UNTIL conditie      -- conditie pentru iesire din ciclu
```

Programul prezentat pentru FOR se va scrie cu DO .. UNTIL sub forma:

```
* PUntil Program de calcul pentru suma unui sir de numere
Clear
Set Talk off
K1=VAL(accept('Introduceti K1'))            && conversie valoare introdusa
K2=VAL(accept('Introduceti K2'))
T=VAL(accept('Introduceti pasul'))   && pasul
S=0
K=k1
DO                  && se executa secventa atata timp cat conditia nu e
indeplinita
        S=S+K
        K= K+t      && se mareste variabila k cu pasul t
UNTIL K>k2          && conditie de iesire din ciclu
? 'Suma numerelor de la ', k1, 'la  ', k2, 'cu pasul ', t, ' este: ', S
Return
```

Suma numerelor de la 2 la 34 cu pasul 2 este: 306

Comanda DO CASE verifica mai multe condiții si în fiecare caz executa o secvența specifica. Daca nici o condiție nu este îndeplinita se executa secvența OTHERWISE dacă clauza este specificata. După executarea unei secvențe se continua automat cu ce urmează după ENDCASE fără a mai verifica celelalte condiții.

DO CASE
  CASE cond1
  ..... - secventa1 executata daca conditia1 este îndeplinită
  CASE cond2
  ..... - secventa2 executata daca conditia2 este îndeplinită
  .....
  [OTHERWISE]
  ..... - secvență dacă nici o condiție nu e îndeplinită
ENDCASE

Se observa ca în dBASE se utilizează programarea structurată, nu exista etichete şi instrucțiunea GO TO. Din orice program sau secvență de program se poate lansa un alt program în regim de subprogram prin comanda DO fis.PRG.

Împreună cu aceste comenzi de ciclare si cele de dialog cu utilizator se folosesc si comenzile:

APPEND BLANK - adaugă o înregistrare alba (goala) la sfârşitul fişierului

Aceste înregistrări albe vor fi completare prin REPLACE utilizând datele obținute de la consola prin dialog.

3.2. Exemple de programe

*** Procedura de adaugare de noi inregistrari**
SET TALK OFF
USE STUD && deschide fisierul
R1 =`D` && forteaza executia primului ciclu
DO WHILE UPPER (R1)=`D` && reia ciclu daca s-a raspus D
CLEAR && sterge ecranul
? `Procedura de adaugare inregistrari `
VNUME= TRIM(ACCEPT(` Nume student: `)) && asteptare nume student
VADR=TRIM(ACCEPT(`Adresa: `))
VBURSA=VAL(ACCEPT(`BURSA: `))
CODS =TRIM(ACCEPT(`COD student (fac,sectie,an,gr.,nr.) `))
VDAT=CTOD(ACCEPT('Data nasterii: sub forma {ll/zz/aa}`))
&&conversieCharToDate

APPEND BLANK && adauga inregistrare alba
* Valorile din variabile vor modifica cimpurile inregistrarii
REPL NUME WITH VNUME,ADRESA WITH VADR, ;

```
BURSA WITH VBURSA,  DATA_N WITH VDAT, CODS WITH VCODS
? `CONTINUATI (D/N)`
WAIT TO R1          && asteapta confirmarea continuarii ciclului
ENDDO               && reluare ciclu daca s-a raspuns `D`
USE                 && inchide fisierul
RETURN              && terminare program
```

S-au introdus datele despre un student în variabile, a căror tip va depinde de modul de introducere a valorilor, s-a adăugat o înregistrare alba, care s-a completat apoi prin REPLACE cu datele din variabile. Prin dialog s-a putut explicita ce reprezintă acele date fata de APPEND simplu.

Pentru o afişare mai explicita fata de cea realizata prin DISP putem utiliza procedura.

```
*       Procedura afisare studenti bursieri
SET TALK OFF
 USE STUD
SET DATE TO DMY       && data sub forma zz/ll/aa
SCAN FOR BURSA>0      && secventa se executa numai pentru bursieri.
CLEAR
? `Nume student :` + NUME
? 'Adresa:` + ADRESA
? `Data nasterii` + DTOC(DATA_N)     && conversie data în sir
?? `(`,  DAY(DATA_N), CMONTH(DATA_N), YEAR(DATA_N),`-`;
   CDOW(DATA_N),`)`
? `Bursa`+STR(BURSA,9,2)          && conversie numeric în sir
? `casatorit `, CAS               && tip logic
WAIT                && asteptare
ENDSCAN             && trece la inregistrarea urmatoare si reia ciclu
USE                 && inchide fisier
RETURN              && sfirsit program
```

Acelaşi program se poate realiza cu DO WHILE astfel:

```
USE STUD
DO WHILE .NOT.EOF()   && ciclul se reia daca nu este sfirsit fisier
 IF BURSA>O           && secventa se face pentru bursieri
 CLEAR
 ? `Nume student: ',NUME
 ? 'Adresa: `,ADRESA
```

```
 ? `Data nasterii: `,DATA_N  && s-a scris ca lista de expresii
 ? `Bursa:`, BURSA           && care pot fi de tip diferit
 ? `CASATORIT:`, cas
 WAIT                        && asteapta apasarea unei taste
 ENDIF
SKIP                              && trece la inregistrarea urmatoare.
ENDDO                        && reluare ciclu
USE
RETURN
```

S-a folosit IF pentru testarea condiţiei BURSA>0 deoarece DO WHILE BURSA > 0 ar fi greşită. Se va ieşi din ciclul DO la primul student nebursier.

Următorul program exemplifica utilizarea comenzii LOOP pentru reluarea fortata a ciclului DO. Se afişează datele studenţilor specificaţi prin nume.

*** Afişare date studenţi selectaţi prin nume**

```
USE STUD
R1 = `D`
DO WHILE R1 $ `DdYy`    && raspunsuri afirmative recunoscute
 CLEAR
VNUME= TRIM( ACCEPT(`Numele studentului: `))
 LOCATE FOR NUME = VNUME&& cauta dupa NUME
 IF EOF()                          && studentul nu s-a gasit
  ? `studentul ` + VNUME + ` nu exista`
  WAIT
 LOOP                 && forteaza reluarea ultimului ciclu DO
 ENDIF
 ? ` Nume studenti:, Nume  ? `Adresa:,` Adresa
 ? ` Data nasterii:`, DATA_N
  WAIT  `Continuati (D/N)?` TO R1
ENDDO
 USE
RETURN
```

Vom prezenta în continuare un exemplu mai complet de prelucrare a unui fişier de date, în cel mai simplu mod folosind DO CASE şi principalele comenzi de creare şi actualizare, selectate printr-un "meniu".

***Program simplu pentru crearea si actualizarea unui fisier**

```
SET TALK OFF
DO WHILE.T.                    && repeta la infinit ciclul
 CLEAR
TEXT
 FUNCTII OFERITE:
        1- Creare structura fisier studenti
        2- Adaugare inregistrari
        3- Modificare inregistrare data prin numar
        4- Modificare sau afisare inreg. data prin NUME student.
        5- Stergere inregistrare
        6- Modificare structura BD
        7- Listare fisier
        8- Compactare fisier - PACK
        9- Revalidare inregistrare marcata
        T- Terminare program
ENDTEXT
WAIT TO R
IF.NOT. R $ `123456789Tt`          && verificare functie admisa
 ? `FUNCTIA ` + R + ` inexistenta`
 WAIT                              && asteptare pentru citire mesaj
 LOOP                              && reluare ciclu
ENDIF
DO CASE
  CASE R=`1`
CREATE STUD                        && creare initiala DB
CLEAR
? ` S-a creat fisierul STUDENTI avind structura`
DISP STRU                          && afisare structura
WAIT
  CASE R=`2`
USE STUD
APPEND                             && adaugare inregistrari de la tastatura
  CASE R=`3`
USE STUD                           && deschidere fisier
N=VAL(ACCEPT(Numarul inregistrarii: '))
EDIT N                             && trece în editare inregistrarea N
  CASE R = `4`
USE STUD
```

```
VNUME=TRIM( ACCEPT(`Numele studentului: `)
LOCATE FOR NUME = VNUME    && cauta inregistrarea IF EOF()
        ? `Studentul cu numele ` + VNUME + ` nu exista`
        WAIT
        LOOP
ENDIF
EDIT                                && editare inregistrare gasita
  CASE R=`5`
USE STUD
N=VAL(ACCEPT('NR. inregistrare pentru stergere: `))
DELETE RECORD N                     && stergere inregistrare
  CASE R=`6`
USE STUD
? ` Puteti modifica structura BD`
MODI STRU
  CASE R=`7`
USE STUD
LIST                                && listare fisier
  CASE R=`8`
USE STUD
? ` Se elimina inregistrarile marcate pentru stergere`
PACK
WAIT
? ` Noul fisier este:`
LIST                                && listare noul fisier
  CASE R='9'
USE STUD
N=VAL(ACCEPT( 'Nr. articol ce se revalideaza: '))
RECALL RECORD N
  CASE R $ `Tt`
USE
RETURN                              && terminare program
ENDCASE
ENDDO                               && reluare ciclu
```

3.3. Programe de calcul funcţii prin descompunere în serie

Calculul funcţiei exponenţiale şi funcţiilor trigonometrice se poate face iterativ prin descompunere în serie, cu o anumită precizie. Se poate lua un număr dat de termeni k sau până termenul calculat devine foarte mic T<eps.

Pentru comparaţie vom afişa valoarea calculata si cea obţinută prin utilizarea functiilor dBase exp, sin, cos.

Funcţia exponenţială e^x unde x este un numar real are descompunerea în serie:

$e^x = 1+x/1!+x^2/2!+ x^3/3!+...$

*** Program calcul functie exponentiala**

```
set talk off
clear

x = val( accept('Introduceti x= ') )
E = 1                   // valoarea initiala suma termeni egala cu T1
T = 1                   // valoarea initiala termen curent egala cu T1
limita = 50             // numar de termeni

for k=1 to limita step 1
        T = T * (x / k) // calcul urmatorul termen fata de cel precedent
  E = E+T
endfor

? 'exp(x) =  ',  exp(x) // valoarea calculata cu functia existenta EXP
?'E=  ', E              // valoarea calculata prin program folosind
descompunerea in serie
return
```

Dupa rulare program pentru x=1 obţinem valoarea lui e:

exp(x) = 2.7182818285
E= 2.7182818285

Funcția cos(x) are descompunerea în serie:

cos(x)=1-x^2/2!+x^4/4!-x^6/6!+x^8/8!-…….

Notand cu C - suma termenilor, T - un termen, k – indicele, algoritmul de calcul va fi:

```
* Program calcul cos(x)
set talk off
clear

x = val( accept('x in grade ') )
x = dtor(x)       // conversie x  din grade in radiani

c=1               // valoarea initiala suma termeni egala cu T1
t=1               // valoarea initiala termen curent egala cu T1
limita = 25       // numar de termeni

for k=2 to limita step 2
        t = (-t*x*x)/(k*(k-1))  // calcul urmatorul termen fata de cel precedent
  c = c + t
endfor

? ' cos(x)=  ', cos(x)            // valoarea calculata cu functia existenta COS
? 'cos(x) calculat =  ',c         // valoarea calculata in program
return
```

Dupa rulare program pentru x=60 grade obținem:

cos(x)= 0.5000000000
cos(x) calculat = 0.5000000000

Funcția sin(x) are descompunerea în serie:

$\sin(x)=x-x^3/3!+x^5/5!-x^7/7!+\ldots$.

Notand cu S - suma termenilor, T - un termen, k – indicele, programul de calcul va fi:

```
* Program calcul sin(x)
set talk off
clear

x = val( accept('Dati x in grade:  ') )
x = dtor(x)     // conversie x  din grade in radiani
s=x
t=x
limita = 15         // numar maxim de cicluri
eps = 1e-12          // precizie

k=3                 // nr. termen curent
//for k=3 to limita step 2
do while abs(t)>eps
          t=(-t*x*x)/(k*(k-1))
  s=s+t
  k = k+2
enddo

? ' sin(x):  ', sin(x)
? 'Sinus calculat:  ',s
? 'numar de termeni:  ', ceil((k-3)/2)              // numar de termeni
return
```

Dupa rulare program pentru x=30 grade obținem:

sin(x): 0.5000000000
Sinus calculat: 0.5000000000
numar de termeni: 6.0000000000

Descompunerea in serie a functiilor hiperbolice este:
$ch(x)= 1+x^2/2!+x^4/4!+x^6/6!+x^8/8!+\ldots\ldots$
$sh(x)= x+x^3/3!+x^5/5!+x^7/7!+\ldots$.

4. INDEXAREA SI SORTAREA FISIERELOR

4.1.Sortare şi indexare

Crearea de noi înregistrări într-un fişier se face de regulă la sfârşit prin adăugare, folosind comanda **APPEND**, şi nu prin înserări intre cele existente prin INSERT. Inserarea nu se recomandă fiindcă necesită mutarea tuturor înregistrărilor care urmează. La adăugare nu se ţine în general cont de conţinutul câmpurilor înregistrării. Se admite şi concatenarea unor fişiere total sau parţial prin:

APPEND FROM fis [FOR cond] [TYPE SDF |
DELIMITED [WITH <char> | BLANK]] [REINDEX]

Înregistrările fişierului specificat, care îndeplinesc condiţia, sunt adăugate la fişierul curent deschis. Daca fişierul nu este de tip DBF, se va specifica si tipul său:
SDF – ASCII text file (System Data Format) în care delimitarea înregistrărilor se face prin CR- Carage Return si LF -Line Field
DELIMITED presupune un fişier text în care câmpurile sunt delimitate prin virgula CSV (Comma Separated Value) iar înregistrările prin CR.
DELIMITED [WITH <char> | BLANK]] Câmpurile delimitate prin caracterul specificat sau spatiu (BLANK).
[REINDEX] – indică actualizarea fişierelor index după terminarea comenzii

Pentru a realiza diferite situaţii si rapoarte, articolele trebuie sa fie ordonate în fişier după anumite criterii: pe secţii, facultăţi, an grupa, cod de material sau piesa, conturi, cod salariat, etc.

Ordonarea completă sau parţiala a unui fişier după unul sau mai multe câmpuri (sau chiar porţiuni de câmp), se realizează prin comanda SORT, care generează un nou fişier sortat, cel curent rămânând nemodificat.

SORT [domeniu] TO fis_sortat ON cimp1 [/A][/C][/D],
cimp2 [/A].....[FOR cond][ASCEN/DESC]

Sortarea in ordinea cheilor se face:
/A - crescător ţinând cont de codul ASCII pentru şiruri de caractere
/C - fără deosebire intre literele mari si mici
/D - ordine descrescătoare

ASCE/DESC - ordinea de sortare este aceeaşi pentru toate cheile

```
USE STUD      && deschide fişierul nesortat
LIST          && listare fişier nesortat
SORT TO STUD_S ON NUME      && sortare după câmpul Nume în fişierul
Stud_S
USE STUD_S          && deschidere fişier sortat
LIST                && afişare fişier sortat
```

Numărul maxim de chei de sortare este 10 si nu pot fi de tip logic, sau memo.

Sortarea necesita încă 2 fişiere de manevra şi spaţiu pentru fişierul sortat si nesortat pe disc. Pentru fiecare situaţie listată este necesară o altă sortare. Pentru fişirul studenţi se poate cere listarea în ordine alfabetică, în ordinea mediilor, în ordinea codului, pe grupe, etc.

Pentru a putea parcurge fişierul în ordine după chei diferite se recomandă utilizarea indexării, care este mai rapidă si mai economica sub aspectul spaţiului şi vitezei de prelucrare.

Un fişier poate fi indexat după maxim 47 de chei, care pot fi combinaţii de câmpuri. Pentru fiecare cheie se creează un fişier index **(.NDX**), care conţine valorile sortate ale câmpului si numărul de ordine al înregistrării corespunzătoare din fişier. Pentru a simplifica utilizarea indecşilor s-a introdus gruparea indecşilor pentru un fişier de date într-un singur fişier cu extensia .MDX.

4.2. Metode de indexare

Pentru regăsirea rapidă în acces direct a informaţiilor din fişierele unei baze de date (BD), pe baza unei chei simbolice, se folosesc metode de indexare sau randomizare a cheii. În majoritatea sistemelor de gestiune a bazelor de date (SGBD) se utilizează metode de indexare care s-au perfecţionat mult. Indexarea multinivel permite o regăsirea unei înregistrări cu valoare dată a cheii prin 3-4 poziţionări pe disc. Pentru un timp de acces disc care este în prezent de cca. 10 ms, aceasta înseamnă regăsirea unei înregistrări dintr-o BD cu zeci de milioane de articole în mai puţin de 1 secundă. Avantajele indexării faţă de utilizarea hashingului constau în :

- posibilitatea citirii ordonate a înregistrărilor, secvenţial în ordinea cheii de indexare;
- permite accesul direct la o înregistrare şi parcurgerea în continuare a fişierului, secvenţial din acel punct, pentru a citi înregistrări care au egală o

parte a cheii (studenţi din aceeaşi secţie şi an);

- posibilitatea creării mai multor fişiere index, pentru acelaşi fişier de date după diferite câmpuri, care sunt chei secundare (există mai multe înregistrări cu aceeaşi valoare a câmpului index).

Pentru regăsirea informaţiilor în acces direct, performanţele tuturor metodelor de indexare sunt bune, dar indexarea are şi dezavantaje care trebuie menţionate:

- utilizarea fişierelor index multinivel (cele mai des folosite), consumă mult timp pentru actualizarea fişierelor index la adăugarea de noi înregistrări în BD mari, care limitează utilizarea lor mai ales când există mai multe fişiere index pentru un fişier de date;
- spaţiul disc consumat de fişierele index este de acelaşi ordin de mărime cu cel al fişierelor de date şi mult mai mare decât cel utilizat de indexii de hashing.

Primul dezavantaj se elimină prin utilizarea unor algoritmi de indexare dinamică cum sunt cei cu arbori echilibraţi B^+ cu încărcare incompletă a nodurilor, care se vor prezenta în continuare. Această metodă complică procedurile de căutare şi actualizare a fişierului index. Spaţiul disc folosit pentru fişierele index poate fi redus funcţie de aplicaţie prin utilizarea unor structuri de BD care folosesc liste înlănţuite cu pointeri sau fişiere de legături. Se elimină în acest caz spaţiul ocupat de cheie în înregistrarea de index.

4.2.1. Indexarea multinivel nedensă după cheia primară

Fişierele secvenţial indexate clasice au fost realizate pe un fişier secvenţial sortat, format din blocuri grupate pe cilindri. Pentru fiecare cilindru s-a creat o tabelă **index înregistrări pe cilindru**, formată din câte o intrare pentru fiecare bloc de date. O înregistrare de index conţine valoarea cheii primare maxime din bloc şi adresa blocului la care se referă. Tabela este plasată în unul sau mai multe blocuri la sfârşitul cilindrului.

Pentru întregul fişier disc (volum) se realizează o tabelă index cilindri pe fişier, având câte o intrare pentru fiecare cilindru. Intrarea conţine cheia maximă pe cilindru şi adresa tabelei index a cilindrului. Dacă o tabelă de index conţine mai multe blocuri, se va crea o tabelă index rezumat urmată de blocurile tabelei detaliu. O intrare din tabela rezumat conţine cheia maximă dintr-un bloc detaliu şi adresa blocului respectiv. Tabela rezumat este separată de tabela detaliu prin unul sau mai multe blocuri libere. La consultare se citeşte tabela rezumat şi se determină blocul din tabela detaliu care conţine cheia căutată. Acesta se poate citi în aceeaşi tură de pistă prin folosirea blocurilor libere, ce vor fi parcurse pe timpul căutării în tabela rezumat. Citirea tabelei cilindru şi a

blocului ce conţine înregistrarea cu cheia căutată se va face printr-o singură poziţionare pe disc, deoarece ele se găsesc pe acelaşi cilindru.

cil. 0	bl. 0	record 1	record 2	…	
	bl. 1				
	bl. 2				
	…				
		Tabela	**index**	**înreg. pe**	**cil.0**
cil. 1					
		Tabela	**index**	**înreg. pe**	**cil.1**
		…			
cil. K					
	bl. N				
		Tabela	**index**	**înreg. pe**	**cil.k**
		Tabela	index	cil. pe	fişier

Tabelă index detaliu

Tab.index rezumat				

Fig.4.1. Indexare multinivel nedensă

Metoda foloseşte indexarea nedensă, deoarece numărul de chei din tabela index este mult mai mic decât numărul de chei din fişierul de date. Spaţiul ocupat de index este de cca. 10 % din spaţiul ocupat de fişierul de date. Aceasta duce la un acces rapid la înregistrări cu maxim 2 accese disc. Metoda a fost totuşi abandonată, deoarece adăugările ulterioare de înregistrări se fac într-o zonă de depăşire unde se înlănţuie cu pointer în înregistrarea precedentă existentă în zona principală a fişierului. După crearea fişierului, tabelele index rămân nemodificate, nu se admit valori duble pentru chei (cheie primară).

4.2.2. Fişiere index dense multinivel

SGBD-urile relaţionale folosesc **fişiere index "dense" aplicate pe fişiere de date neordonate.** Se pot construi fişiere index pentru mai multe câmpuri din fişier, fără ca acestea să fie câmpuri cheie (chei secundare). Fişierul index va conţine câte o intrare pentru fiecare înregistrare din fişier. Intrarea de index este de forma **(** K_i**,**P_i**)** unde K_i este valoarea cheii, iar P_i un pointer ce indică adresa înregistrării ce conţine cheia în fişierul de date. Adresa este relativă în fişier şi poate fi:

- adresa în octeţi a înregistrării în fişier;
- adresa sector început bloc plus adresa octet în bloc a înregistrării (.NDX);
- numărul înregistrării în fişier (fişiere .MDX în dBASE).

Ultima formă ocupă doar 4 octeţi şi presupune înregistrări cu lungime fixă, condiţie impusă în unele BD relaţionale. Tabela index obţinută se sortează în ordinea valorii cheilor şi face referiri la înregistrările din fişierul de date prin pointer.

Dacă tabela index conţine mai mult de un bloc se creează o tabelă rezumat de nivel 2 (fig.4.2), în care pentru fiecare bloc din nivelul 1 se creează o intrare ce conţine cheia maximă din acel bloc şi numărul sectorului unde începe blocul. Fiecare bloc din tabela de nivel 2 se plasează după blocurile de nivel 1, la care se referă. În acelaşi mod se formează tabele index de nivel 3, ş.a.m.d.

Căutarea unei înregistrări pentru o cheie dată presupune citirea a câte un bloc din fiecare nivel, plus blocul din fişierul de date. Căutarea unei chei în blocurile index se face rapid în memoria centrală şi timpul se neglijează. Pentru un fişier de 30 de mii de articole cu lungimea înregistrării index de 16 octeţi (12 cheie +4 adresă) rezultă 3 nivele de index, deci 4 poziţionări pe disc. Timpul de acces este sub 0,1 secunde pentru discuri cu timp de acces sub 20 ms.

Fişierele index multinivel necesită mult timp pentru actualizare când sunt de mari dimensiuni. Adăugarea unei noi chei duce la o reorganizare a tabelelor index, prin inserare într-un tabel secvenţial ordonat. Pentru a păstra consistenţa BD pentru orice actualizare în fişierul de date, se actualizează toate fişierele index asociate, dacă în dBASE se utilizează fişiere .MDX.

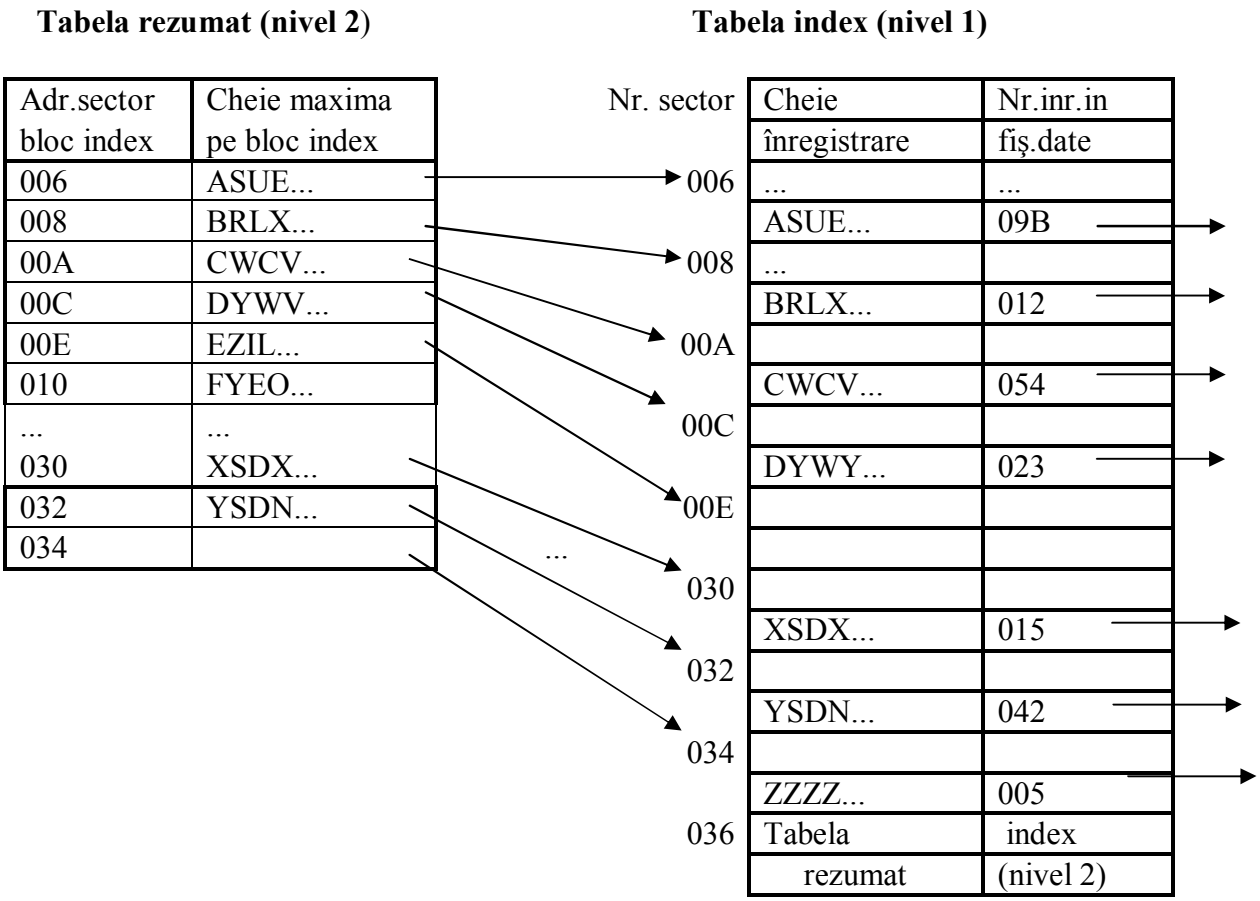


Fig. 4.2. Fişier index dens multinivel

La folosirea fişierelor de tip NDX, se recomanda ca la adăugări numeroase de înregistrări, fişierele index să nu fie deschise. Refacerea fişierelor index se va face la sfârşitul lucrării prin reindexare şi nu după fiecare înregistrare adăugată. **Dacă se folosesc grupări de fişiere index de tip MDX sau CDX, la deschiderea fişierului de date se deschid automat toate fişierele index asociate**. După orice actualizare a unei înregistrări din fişierului de date care afectează un câmp cheie, se restructurează întregul fişier index.

4.3. Comenzi de indexare

Indexarea se face în dBase folosind comanda INDEX unde se specifică câmpul cheie după care se face indexarea şi numele fişierului index generat.

INDEX ON cheie TO fis_index [UNIQUE][DESCENDING]

Clauza UNIQUE nu acceptă chei cu aceeaşi valoare în mai multe înregistrări, iar DESC specifică sortarea fişierului index in ordine descrescătoare.

Daca fişierul index este deschis parcurgerea secvenţială a înregistrărilor din fişier se va face în ordinea dată de fişierul index (DISP, LIST, EDIT, BROWSE,
SKIP, LOCATE).

Deschiderea fişierelor index se poate face odată cu fişierul de date sau separat prin :

SET INDEX TO lista_fis_index

Pentru un fişier se pot crea mai multe fişiere index după mai multe câmpuri (Cods, Nume, Telefon) fără a sorta înregistrările care rămân în ordinea în care au fost introduse.

```
USE STUD     && deschide fişierul
LIST         && listează fişierul in ordine naturala
INDEX ON  Nume TO INUME && indexare după nume student
LIST         && listare in ordinea numerelor
INDEX ON CODS TO ICODS
LIST         && listare in ordinea codurilor (facultate, secţie)
USE STUD     && deschidere fără fişiere index
LIST         && listare in ordine naturala
SET INDEX TO ICODS       && deschidere fişier index ICODS
LIST         && listare in ordinea codurilor
USE          && închidere fişier de date si  index asociate
USE STUD     && deschide fişierul de date fără fişiere index
LIST         && listează fişierul in ordine naturala
```

La închiderea unui fişier de date se închid si toate fişierele index asociate lui. Un fişier DBF indexat poate fi exploatat şi fără fişiere index sau numai cu unele din acestea. Daca se deschid simultan mai multe fişiere index,

atunci **primul din lista este index master si va determina ordinea de parcurgere a înregistrărilor** la accesul secvențial sau cheia de selecție in acces direct. Oricare fişier index poate fi specificat ca master prin comanda:

SET ORDER TO n - unde n este numărul de ordine din lista indexi în care s-a deschis.
SET ORDER TO 0 - reprezintă ordinea naturală
SET ORDER Cods - pentru tag index in .MDX

Dacă se fac modificări in fişierul de date (adăugări, ştergeri, modificări de înregistrări), se actualizează automat toate fişierele index deschise pentru toate comenzile (APPEND, EDIT, BROWSE, REPL,PACK).

```
USE STUD INDEX ICODS,INUME          && deschide fişier DBF si index;
LIST                    && afişare in ordinea codurilor studenților
SET ORDER TO 2          && fişier index master INUME
LIST                    && afişare in ordinea numelor.
```

Selecția in acces direct. In fişierele indexate se poate caută o înregistrare după cheia din fişierul index master deschis utilizând comenzile:

SEEK expr - caută înregistrarea cu cheia dată prin expresie
FIND cheie - caută înregistrarea cu cheia specificata direct

Dacă înregistrarea cu cheia specificată exista se poziționează funcțiile **FOUND() =.T.** si **EOF()** = .F. iar dacă înregistrarea nu exista valorile vor fi invers. După orice căutare se va verifica una din funcții si daca EOF()=.F., înseamnă că s-a poziționat pe înregistrarea căutată. Compararea cheii cu câmpul cheie se face pentru şiruri funcție de modul de comparație SET EXACT ON sau OFF.

```
USE STUD INDEX INUME
SET EXACT OFF       && căutare după primele caractere din cheie
SEEK 'POP'          && se caută primul care are primele litere POP
* FIND POP          && dacă se utilizează comanda FIND

IF EOF ()           && verifica dacă s-a găsit cheia
 ? ` Nu exista student cu numele cerut `
 RETURN
ENDIF
DO WHILE Nume =`POP` .AND. .NOT. EOF().
```

```
DISP Nume, Adresa   && afişează toţi studenţii  a căror nume începe cu POP..
SKIP                && trece la următoarea înregistrare conform fişierului
index
ENDDO
RETURN
```

Acelaşi exemplu se poate generaliza pentru dialog:

```
USE STUD INDEX INUME
VNUME =ACCEPT( `Nume student: `)          && cere nume student
căutat
* FIND &VNUME     && se specifica cheia indirect
SEEK VNUME        && in SEEK se specifică numele variabilei
IF EOF()          && verificare dacă studentul există
  ? ` Studentul  ` + VNUME + ` nu exista`
  RETURN
ENDIF
DO WHILE NUME=VNUME .AND..NOT. EOF()
DISP          && afişează toţi studenţii cu numele dat
SKIP          && trece la următorul student conform fişierului index
ENDDO
```

S-a introdus si condiţia .NOT. EOF() în DO pentru a se opri ciclul în cazul în care studenţii căutaţi sunt la sfârşitul fişierului.

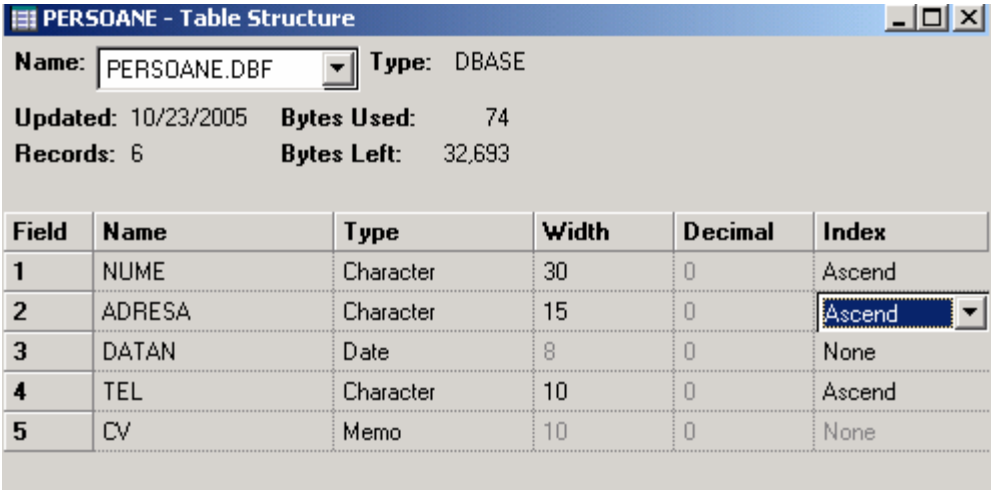
Comanda FIND are exact aceeaşi funcţie dar diferă puţin scrierea. Pentru ultimul exemplu va fi o substituţie cu valoarea introdusă în variabila VNUME:

```
FIND &VNUME      - iar restul rămâne la fel.
```

Grup de fisiere index. In variante actuale de dBASE s-a introdus noţiunea de grup de fişiere index (Multiple Index) care are extensia .MDX si care poate conţine, mai multe fişiere index, de obicei pentru acelaşi fişier de date (DBF).

Fiecare fişier index se numeşte TAG si are un nume propriu, Daca la crearea structurii fişierului DBF se completează coloana Index cu Ascending din dreptul unui câmp, el devin automat cheie (implicit valoarea este None). Pentru fişierul DBF se generează automat un fişier MDX cu acelaşi nume ca al fişierului (STUD.MDX), care conţine pentru fiecare cheie specificată un fişier index TAG cu numele câmpului.

La orice deschidere a fişierului DBF se deschid automat toate fişierele index din fisierul multiplu MDX asociat, asigurând automat actualizarea fişierelor index componente.


PERSOANE - Table Structure

Name: PERSOANE.DBF Type: DBASE

Updated: 10/23/2005 Bytes Used: 74

Records: 6 Bytes Left: 32,693

Field	Name	Type	Width	Decimal	Index
1	NUME	Character	30	0	Ascend
2	ADRESA	Character	15	0	Ascend
3	DATAN	Date	8	0	None
4	TEL	Character	10	0	Ascend
5	CV	Memo	10	0	None

Fişierul index (TAG) master se specifică prin:

SET ORDER TO TAG nume_tag - index din fişierul MDX activ
SET ORDER TO TAG nume_tag OF fis.MDX - dacă indexul este în alt fişier MDX

La crearea ulterioara a unor fişiere index TAG se va folosi:
INDEX ON cheie TO TAG nume_tag OF fis.MDX.

In acest fel se pot genera tag_uri index pentru fişiere DBF diferite in acelaşi fişier MDX. Deschiderea fişierului si a indecşilor se face prin specificarea listei_Tag si grupului MDX.

USE fisier INDEX lista_tag OF fis.MDX [ORDER nume_tag].

OBSERVATIE: Utilizarea fişierelor index MDX este contraproductivă, deoarece toate fişierele TAG se deschid la deschiderea fişierului DBF si vor duce la actualizarea "ON LINE" a fişierelor index, care consumă mult timp.

Reindexarea fişierului se poate face pentru **toate fişierele index deschise** prin comanda **REINDEX**. Reindexarea este obligatorie si după comanda PACK, care modifica numărul de ordine al înregistrărilor după eliminarea celor marcate.

```
USE STUD
PACK            && eliminare înregistrări marcate
SET INDEX TO INUME,ICOD      && deschide fisiere index
REINDEX         && reindexeaza fişierele index  deschise
USE             && închide fişierul
```

La comenzile de actualizare s-a introdus clauza REINDEX, care cere ca reindexarea să se facă doar la terminarea comenzii si nu pentru fiecare înregistrare modificata (APPEND FROM fisier , REPLACE).

4.4.Utilizarea simultană a mai multor fişiere din BD

Baza de date (BD) este formată din mai multe tabele (fişiere) legate între ele prin chei simbolice şi care urmăresc reducerea redundanţei informaţiilor. Până in prezent a fost folosit un singur fişier deschis, la care se refereau toate comenzile.

Deschiderea altui fişier în aceeaşi zonă implică închiderea celui deschis. Toate datele despre acest fişier se păstrau intr-o zona de memorie de lucru pe care o vom numi zona 1. In dBASE se pot utiliza implicit maxim 40 fişiere deschise simultan, fiecare în altă zonă de lucru.

Fiecare zona de lucru din memorie păstrează aceleaşi informaţii despre fişierul deschis:

- numele fişierului DBF deschis
- informaţiile privind structura înregistrărilor
- numele fişierelor index, report, format asociate.
- contorul de înregistrări şi înregistrarea curenta.
- zona tampon pentru citirea din fişier
- adresa blocului curent citit din fişier

Zona 1	**Zona 2**	**Zona3**	**Zona 4 .....**
Student	**Masini**	**Accidente**	

Orice fişier poate fi deschis in orice zona care se selectează cu **SELECT nr_zona**. Zona in care se deschide fişierul poate fi specificata prin comanda USE utilizând **clauza IN:**

```
SELECT 3                    && selecţie zona 3
USE STUD INDEX INUME        && deschide fişier studenţi in zona 3
SELECT 1                    && selecţie zona 1
USE CURS INDEX ICODC        && deschide fişier cursuri
USE MASINI IN 4 INDEX INRM  && deschide fişier maşini in zona 4
LIST                        && afişează lista cursuri
SELECT 3          && selectare zona 3
LIST              && afişare lista studenţi
SELECT 4          && selectare zona 4
LIST              && afişează lista masini
```

Prelucrări ale unui fişier intr-o zona (SKIP, GO, FIND, APPEND...) nu afectează fişierele din alte zone, care rămân la forma si poziţia ultimei prelucrări.

Forma generala a comenzii USE este:

USE fisier [INDEX lista fisiere_index][IN nr_zona][ALIAS alias] [AGAIN][EXCLUSIVE | SHARED] [NOUPDATE]

Clauza **ALIAS** specifica un nume prescurtat al fişierului ce poate fi folosit pentru referire in program la fişier sau la zona in care este deschis.
Se permite deschiderea simultana a unui fişier in 2 zone folosind clauza **AGAIN**.
Clauza **NOUPDATE** nu permite modificări în fişier

Închiderea fişierelor se poate face prin:

USE - închide fişierul DBF si cele index din zona curenta
USE IN nr_zona - închide fişierul din zona specificata
CLOSE DATABASE - închide toate fişierele DBF din toate zonele
CLOSE ALL - închide toate fişierele DBF,NDX,FMT, FRM, din toate zonele.

Intr-o zona se pot utiliza câmpuri ale înregistrării curente din alta zonă prin **prefixare** cu numele fişierului sau un nume alias astfel:

```
USE STUD INDEX INUME ALIAS ST      && fişier Stud indexat după nume
USE MASINI IN 2 INDEX INRM ALIAS MS  && fişier  indexat după NRM
VNUME= ACCEPT( `Nume student`)
SEEK VNUME                         && caută studentul
DISP          // afisare date student selectat
* Presupunem ca in fişierul STUD exista un numar masina NRM
SELE 2        // selectare zona cu fisier  Masini
SEEK ST->NRM  && cauta dupa NRM din fisierul STUD in fisierul MASINI
? ST->NUME, ST->ADR, NRM, TIP_M   && afisare date student si masina
Return
```

4.5. Legaturi între fişiere deschise în zone diferite

In fişiere diferite pot exista nume de câmpuri identice (NRM).

Intre doua fişiere deschise in zone diferite se poate stabili o relaţie încît la poziţionarea pe înregistrarea din fişierul master sa se realizeze automat poziţionarea pe o înregistrare corespunzătoare din fişierul 2. Relaţia se poate specifica prin:

- număr de înregistrare ce poate fi dată printr-o expresie
- nume de câmp comun celor doua fişiere, care in fişierul 2 este cheie pentru un fişier index master deschis (NRM)

```
USE STUD IN 1              - deschide fişierul studenţi in zona 1
USE MASINI IN 2            - deschide fişierul studenţi in zona 2
SET RELATION TO RECNO() INTO MASINI
GO 5                       - poziţionează pe înregistrarea 5 in STUD si
MASINI
```

In exemplul dat pentru orice poziţionare in fişierul master studenţi, se cere automat poziţionarea pe înregistrarea cu acelaşi număr din fişierul maşini (RecNo()).

```
USE STUD IN 1                  - deschide fisierul master Stud
USE MASINI INDEX INRM IN 2     - deschide fisierul referit Masini indexat
dupa NRM
SET RELATION TO NRM INTO MASINI - definire relaţie intre fisierul Stud
şi Masini
```

Relaţia definită cere ca la orice poziţionare pe o înregistrare din fişierul master să se selecteze automat maşina cu numărul NRM. Câmpul NRM trebuie să existe in ambele fişiere, iar în fişierul referit (fiu) să fie cheie primară (index master).
In clauza TO se poate specifica si un pointer.

Pentru a putea exploata mai uşor mai multe fişiere cu relaţii intre ele au fost modificate si anumite comenzi si funcţii.

EOF(zona) - permite testarea indicatorului EOF din zona specificata
FOUND(zona) - indicatorul logic FOUND() din zona indicata
SKIP n IN alias - poziţionare în fişier din alta zona fata de înregistrarea crt.
GO n IN alias - poziţionare pe înregistrarea n din fişierul indicat
RECNO(zona) - numărul înregistrării curente din zona
RECSIZE(zona) - lungime înregistrare fişier din zona
RECCOUNT(zona) - număr de înregistrări in fişier din zona
SEEK(cheie, zona) - funcţie de căutare (pozitionare) în altă zonă
ALIAS(zona) - afişează numele sau aliasul fisierului deschis in zona

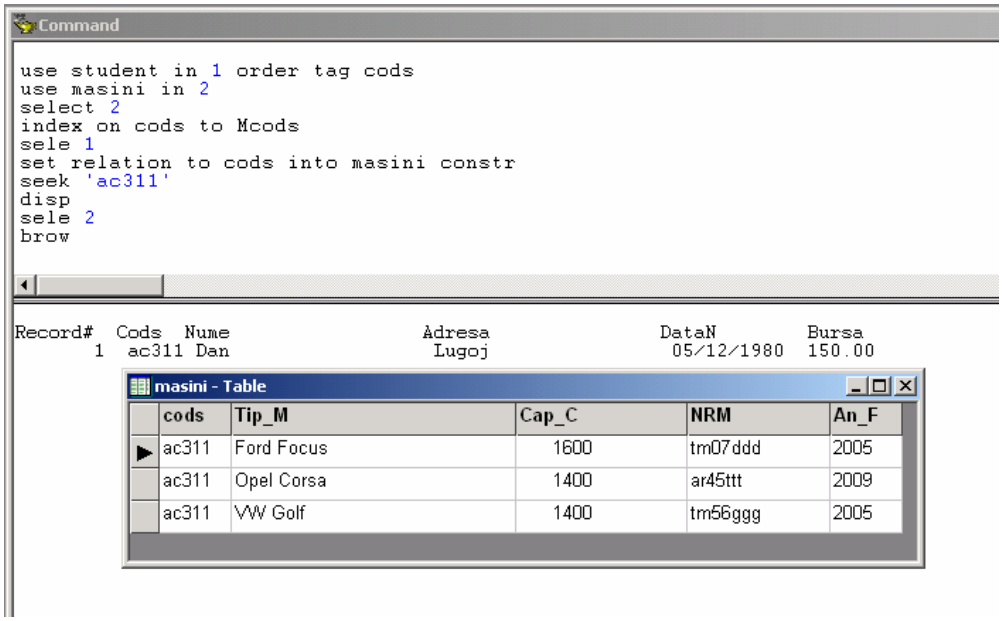
EOF(), ALIAS(), RECNO(),... – se refera la zona curentă

Folosind SET RELATION se pot crea BD complexe, formate din mai multe fişiere, care au relaţii intre ele si pot fi consultate simultan fără complicaţii de programare.
Atenţie: Se interzice folosirea literelor ca alias, sau variabile deoarece sunt implicit considerate nume de zone.
Forma generala a comenzii SET RELATION permite legaturi cu mai multe fişiere:

SET RELATION TO exp1 INTO alias1 [CONSTRAIN], exp2 INTO alias2,.... [ADDITIV]

La un moment dat dintr-o zona numai ultima relaţie definita este operaţională dacă nu se utilizează clauza ADDITIV.
Clauza **CONSTRAIN** selectează numai înregistrările referite de înregistrarea părinte.

Consideram tabelele Student si Masini indexata după cods (index Mcods) putem afişa toate maşinile unui student prin secvenţa care foloseşte Set relation Stud- Masini şi Constrain.



4.6. Exemple de programe

Exemplul 1

Considerăm fişierele care conţin câmpurile specificate:
STUD: NUME, ADRESA, DATA_N, CODS, BURSA, NRM
MASINI: NRM, TIP, AN_F, CAP_C ,PUTERE
Fişierul STUD îl presupunem indexat după NUME, CODS si NRM iar MASINI după număr maşina(NRM). Programul va permite căutarea unui student după oricare din chei si afişarea datelor personale si ale maşinii.

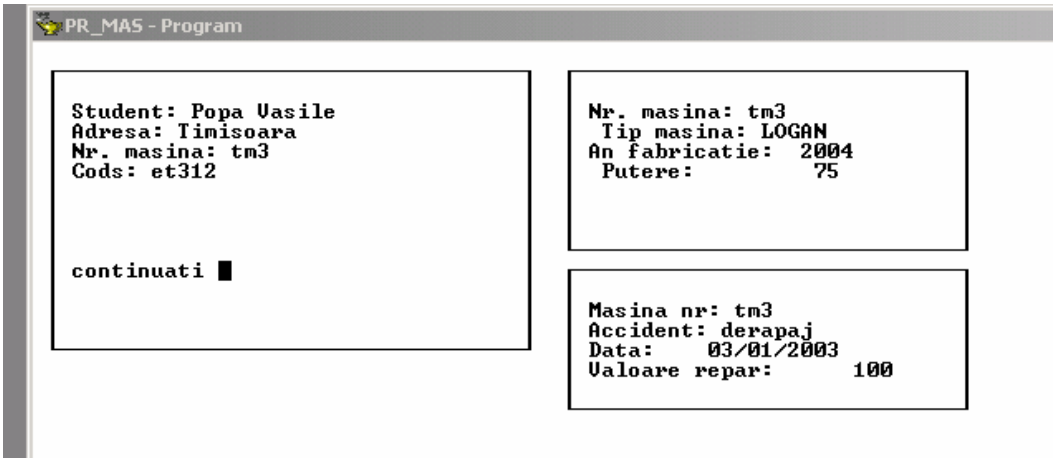
```
*=============================================
* PREL * Program pentru exemplificare legaturi intre fisiere
*=============================================
SET TALK OFF
USE STUD INDEX  INUME, ICODS, INRM     && deschis in zona 1
USE MASINI IN 2 INDEX INR ALIAS MS     && deschis in zona 2
SET RELATION TO NRM INTO MS            && precizare relatie
```

```
DO WHILE .T.        && ciclu infinit cu iesire conditionata
CLEAR
TEXT
        Cautarea se poate face dupa:
        1- Nume student
        2- Cod student
        3- Numar masina
        4- terminare program
ENDTEXT
WAIT TO R
DO CASE
  CASE R=`1`
     SET ORDER TO 1                 && selectare fisier index master INUME
     C1=`NUME STUDENT: `            && C1 mesaj parametric de dialog
     C2=`NUME`                      && C2 nume cimp utilizat in DO WHILE
  CASE R=`2`
     C1='COD STUDENT: `
     C2=`CODS`                      && C2 numele cimpului cod student
     SET ORDER TO 2                 && selectare fisier index  ICODS
  CASE R=`3`
     C1=`NR. MASINA: `
     C2=`NRM`                       && cimp numar masina
     SET ORDER TO 3                  && selectare fisier index INRM ca
master
  CASE R=`4`
     CLOSE ALL                      &&inchide fisiere si sfirsit program
     RETURN
  OTHER
   ? `Functia` + R + ` inexistenta`
   WAIT
   LOOP                         && reluare dialog
ENDCASE
CH =TRIM(ACCEPT( C1))           && introducere cheie (C1 - mesaj
corespunzator)
SEEK CH                         && cautare student dupa cheie
IF EOF()                        && student negasit
 ? `Studentul precizat prin `+ C1 + CH + `nu exista`
  WAIT
  LOOP        &&reluare dialog
ENDIF
```

```
DO WHILE &C2=CH        && cautare toti studentii cu cheia data
   DISP                && afisare date despre student
  IF EOF(2)            && verificare daca s-a gasit masina cu NRM in
zona 2
    ? `Studentul nu are masina
   ELSE
  ? MS->NRM, MS->AN_F, MS->CAP_C, MS->PUTERE    && date
despre masina
 ENDIF
WAIT
SKIP                   && trece la studentul urmator cu aceeasi cheie
ENDDO
  ENDOO
```

După încercarea exemplului prezentat încercaţi modificarea lui pentru a se selecta maşina după NRM si apoi datele studentului inversând relaţia.

Exemplu 2 de program care utilizează 3 fişiere legate între ele STUD, MASINI şi ACCIDENT. Pentru fiecare student selectat se afişează datele studentului, ale maşinii şi accidentele maşinii respective.



*** Pr_mas** Program afisare persoane, masini si accidente

```
set talk off
Clear
set date to dmy
```

```
use stud index inume, icods, inrm alias st
use masini in 2 index inm alias m1
set relation to nrm into m1
use accident in 3 index inr alias ac
select 2
set relation to nrm into ac
select 1
do while .t.
clear
text
  Cautarea se face dupa:
    1. Nume student
    2. Cod student
    3. Numar masina
    4. Terminare
endtext
wait 'Dati optiunea: ' to r

do case
  case r='1'
    set order to 1
    c1='Nume student:'
    c2='Nume'
  case r='2'
    set order to 2
    c1='cod student:'
    c2='Cods'
  case r='3'
    set order to 3
    c1='Nr. masina:'
    c2='Nrm'
  case r='4'
    close all
    clear all
    return
  other
  ? 'Functia '+r+' inexistenta'
    wait
    loop
endcase
```

```
clear    && sterge fl
ch =accept( c1)
seek trim(ch)
if eof()
  msgbox( 'Studentul precizat prin '+c1+ch+' nu exista',1)
   wait
   loop
endif

 do while &c2=ch
   clear            && sterge fl
 ? 'Student: ' +nume
 ? 'Adresa: ' + adresa
 ? 'Nr. masina: ' + nrm
 ? 'Cods: ' +cods
if eof(2)
   ?  'Studentul nu are masina'
   else
   * ? m1->nrm, m1->tip, m1->an_f, m1 ->cap_c, m1 ->putere
   ? 'Nr. masina: ' +m1->nrm
? ' Tip masina: '+ m1->tip
?  'An fabricatie: ' + str(m1->an_f,5)
? ' Putere: ' + STR(m1->putere)
          if .not. eof(3)
? 'Masina nr: ' +ac->nrm        //afisare accidente
? 'Accident: ' +ac->cauza
   ? 'Data: ' + DTOC(ac->data)
   ? 'Valoare repar: ' +str(ac->valoare,8)
           endif
   endif
  ?
Wait 'continuati' to r
  skip
          enddo
enddo
Return
```

Structura si conţinutul fişierelor de date utilizate

Structura fisier STUD.DBF

Field	Field Name	Type	Length	Dec	Index
1	NUME	CHARACTER	10		N
2	ADRESA	CHARACTER	10		N
3	BURSA	NUMERIC	6		N
4	DATA	DATE	8		N
5	CODS	CHARACTER	10		N
6	NRM	CHARACTER	10		N

Fişier STUD.DBF

	NUME	ADRESA	BURSA	DATA	CODS	NRM
1	adi	tm	2592	03	ac231	tm1
2	deni	TM	4074	12	ac427	tm5
3	dan	tm	32442	02	ac163	tm3
4	virgil	TM	3241	12	ac523	tm4
5	pop	ar	57438	12	ac234	ar1
6	vlad	ar	342	12	ac235	ar2
7	liviu	DEVA	657	02	ac123	ar6
8	stefan	TM	3402	10	ac124	tm5
12	radu	tm	12345	11	ac125	tm2
13	gelu	tim	500		ac126	ar5
14	asul	tim	2345	12	ac127	tm10
16	ralu	ar	9876	05	ac236	ar3
17	vasile	ar	10123	12	ac238	ar4

Structura fişier MASINI.DBF

Field	Field Name	Type	Length	Dec	Index
1	NRM	CHARACTER	10		N
2	TIP	CHARACTER	10		N
3	AN_F	NUMERIC	4		N
4	CAP_C	NUMERIC	7	2	N
5	PUTERE	NUMERIC	10		N

Fişier MASINI.DBF

	NRM	TIP	AN_F	CAP_C	PUTERE
1	tm5	opel	1993	1575	85
2	tm4	golf	2000	1600	90
3	tm3	logan	2004	1500	75
4	tm2	audi	2001	1700	85

5	tm1	dacia	1996	1400	70
6	ar1	bmw	2000	2200	110
7	ar2	mercedes	2001	2300	120
8	ar3	dacia	2002	1400	70
9	ar4	ford fiest	2004	1400	120
10	ar5	nissan	1999	1600	130
11	tm10	ford musta	1988	1480	210

Structura fisier ACCIDENT.DBF

	Field Name	Type	Length	Dec	Index
1	NRM	CHARACTER	10		
2	DATA	DATE	8		
3	CAUZA	CHARACTER	20		
4	VALOARE	NUMERIC	10		

Fisier ACCIDENT.DBF

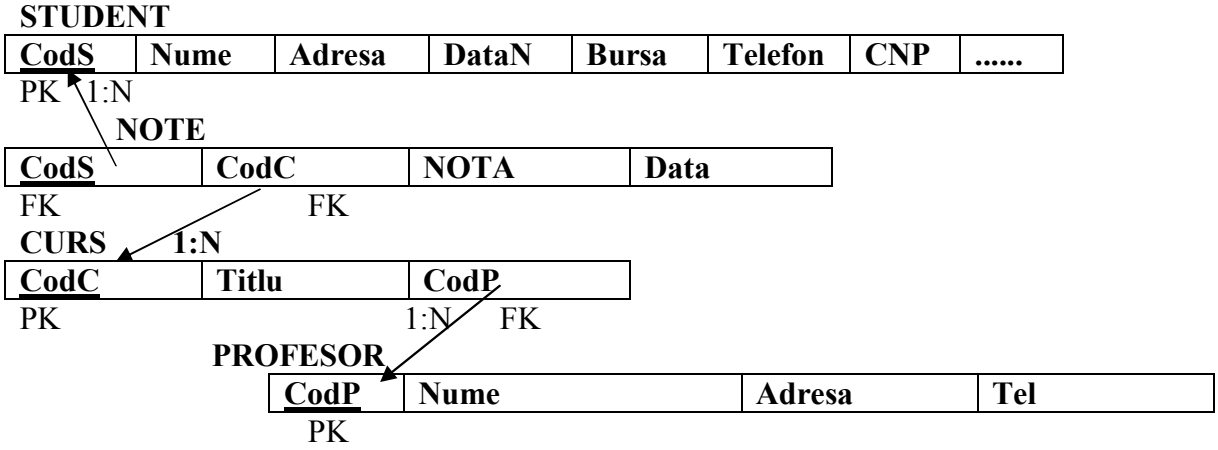
	NRM	DATA	CAUZA	VALOARE
1	tm3	01	derapaj	1000
2	tm1	10	coliziune	2500
3	ar1	02	viteza 150	3100
4	tm2		prioritate	200

4.7. Chei primare şi integritate referenţială

Considerăm o bază de date universitară normalizată (fără redundanţă), care are structura din figură. Avem tabelele **Student**, **Curs** şi **Profesor,** care reprezintă **entităţi** cunoscute între care există legături de tipul N:M (Many- To-Many) sau 1:N, greu de implementat prin liste înlănţuite. Un student participă la N cursuri, iar la un curs participă M studenţi. La fiecare curs studentul are o notă.

- Fiecărui student îi corespunde o înregistrare cu datele personale şi o **cheie unică de identificare Cods** numită **cheie primară PK** (Primary Key),.
- Fiecare curs din planul de învăţământ are un **Codc** unic ca o cheie simbolică primară PK şi are o înregistrare în care se specifică Titlul cursului, cod profesor **CodP**, dacă există o tabelă cu datele personale ale profesorilor.
- Implementarea relaţională introduce o tabelă de legătură **Note,** care conţine codul studentului Cods, codul cursului Codc, ca informaţii de

legătură între tabela Student si Curs, iar Nota şi Data sunt informaţii cantitative. Cods şi Codc se numesc chei externe FK (Forein Key) şi pot lua numai valorile existente în tabelele Student respectiv Curs. Se permit note numai pentru studenţii şi cursurile care există.



Prin tabela Note, legătura M:N se sparge în 2 legaturi 1:N.

Pentru o căutare rapidă:

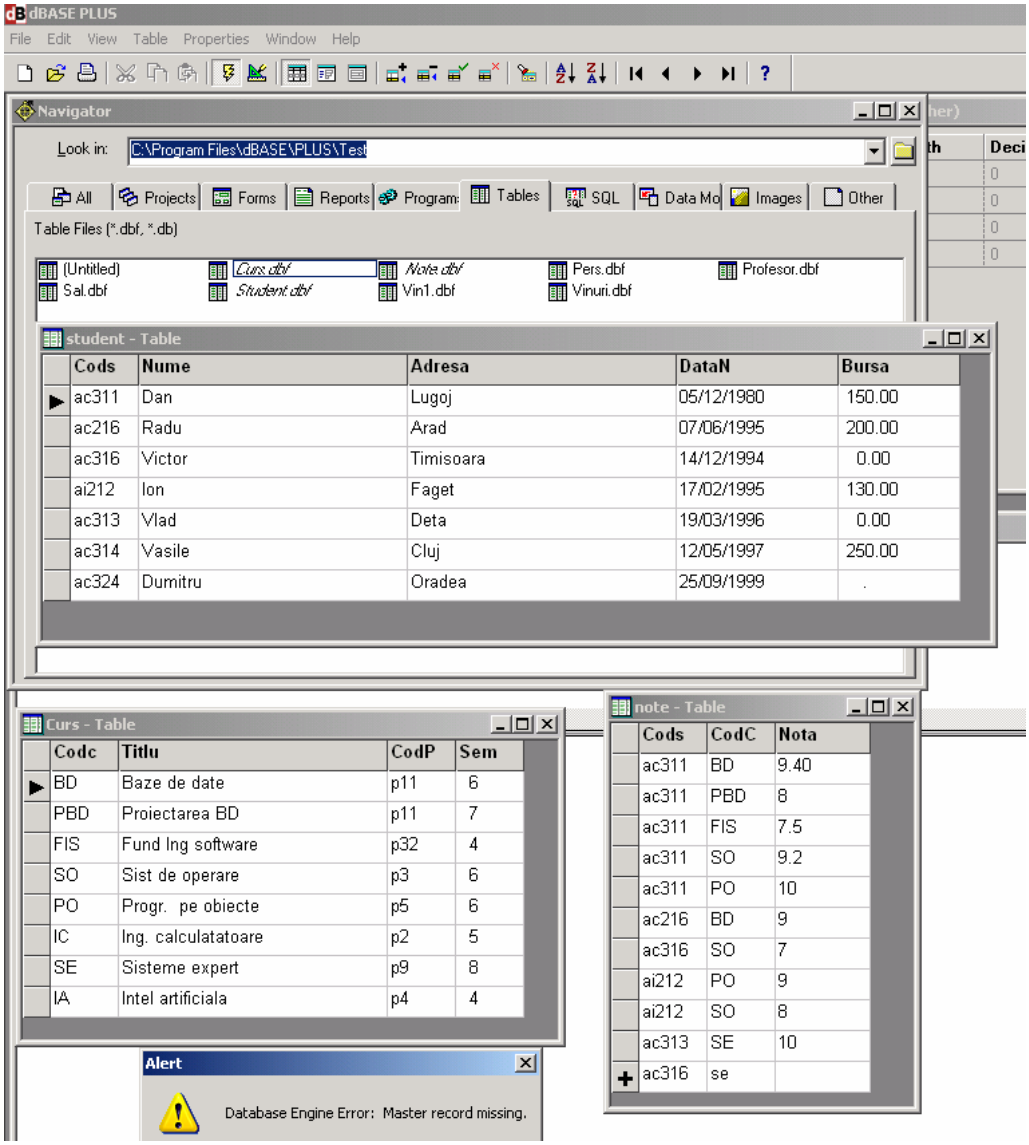
- fişierul Student se va indexa după Cods,
- fişierul Curs după câmpul Codc,
- fişierul Profesor după codP.
- fişierul Note se va indexa după Cods şi Codc, fiindcă la un moment dat trebuie să găsim toate notele unui student

În aceşti **indecşi din Note** numiţi **secundari** poate apare o valoare de mai multe ori. Aici Cods şi Codc vor apare ca şi chei externe Forein Key (FK), care fac referinţă spre cheia primară din tabela părinte (master) în fişierele Student şi respectiv Curs.

În bazele de date avem:

- **indecşi după cheia primară** în care fiecare cheie este unică
- **indecşi secundari**, după orice câmp în care valorile cheii se pot repeta

Se prezintă mai jos tabelele completate în care au fost definite fişierele index primare şi secundare şi s-au definit condiţiile de integritate a referinţei. Se observa mesajul care apare dacă se dă o valoare a cheii Codc (se), care nu există în Curs.

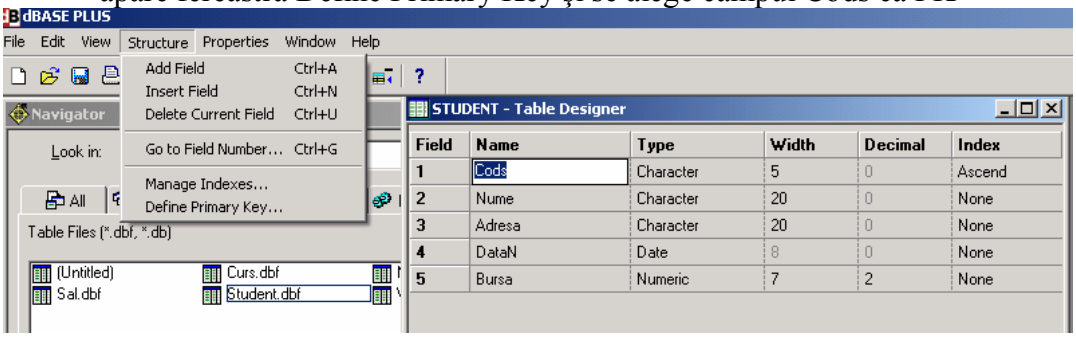


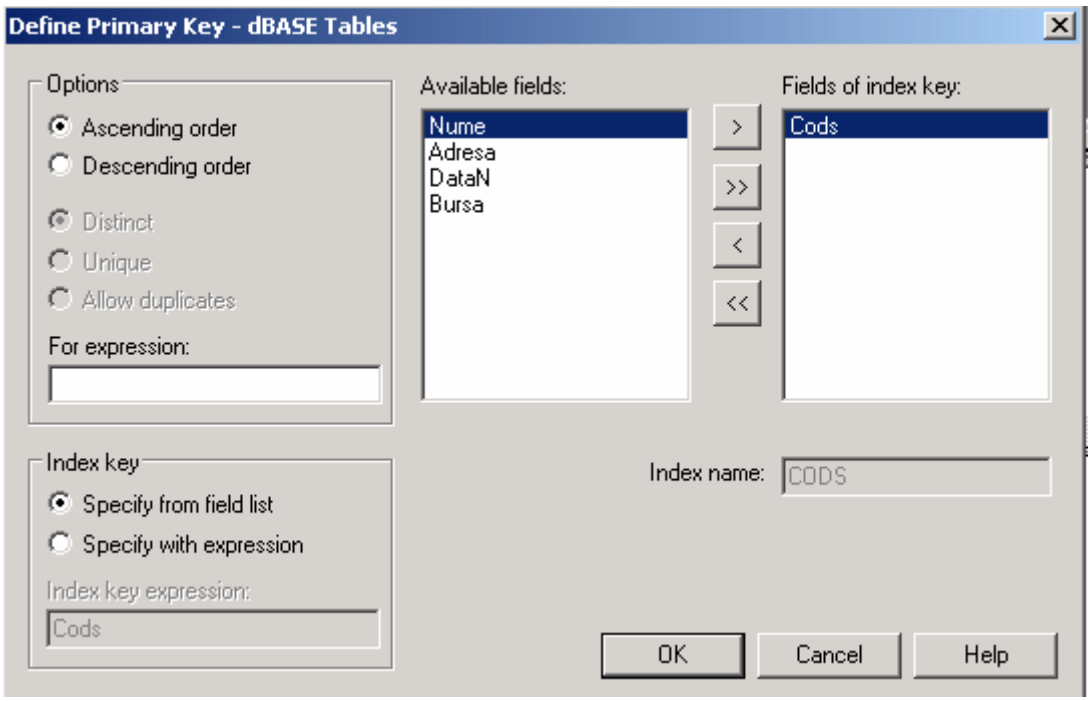
Vom prezenta in continuare modul de definire a cheilor primare şi integritatea referinţei în dBase. O tabelă poate avea o singură cheie primară şi mai multe chei secundare pentru căutare. Indexarea după un câmp cheie primară se face prin:

USE Student
INDEX ON cods TAG cods **PRIMARY**

Definirea cheii primare pentru tabela Student se poate face şi prin meniuri:

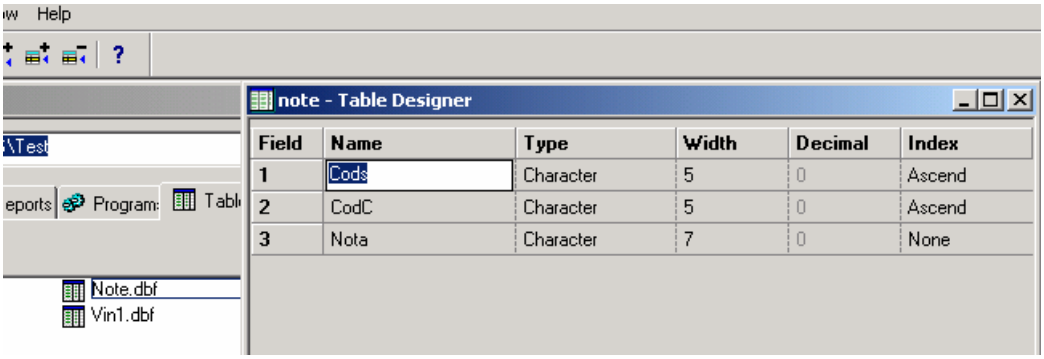
- se selectează fişierul student şi se dă click dreapta
- se selectează Design Table afişând structura fişierului (Table Designer)
- se selectează din meniul principal Structure şi apoi Define Primary Key
- apare fereastra Define Primary Key şi se alege campul Cods ca PK





La fel se se defineşte cheia primară Codc pentru tabela Curs

Se definesc indecşi secundari Cods şi Codc pentru tabela Note deschizând fereastra **Designe Table** unde se aleg Ascend în coloana index.



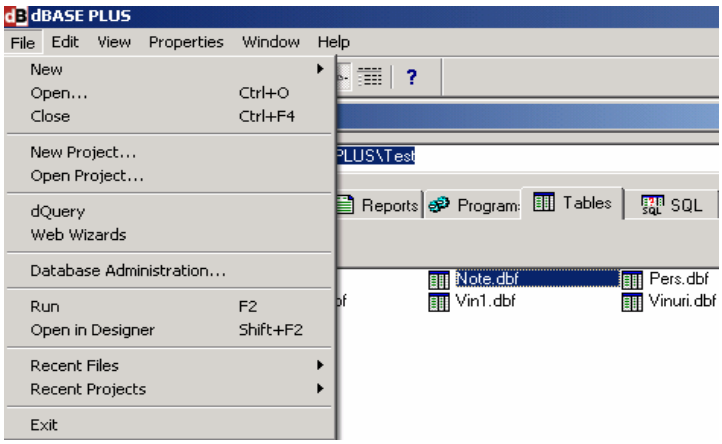
Definire integritate referențială

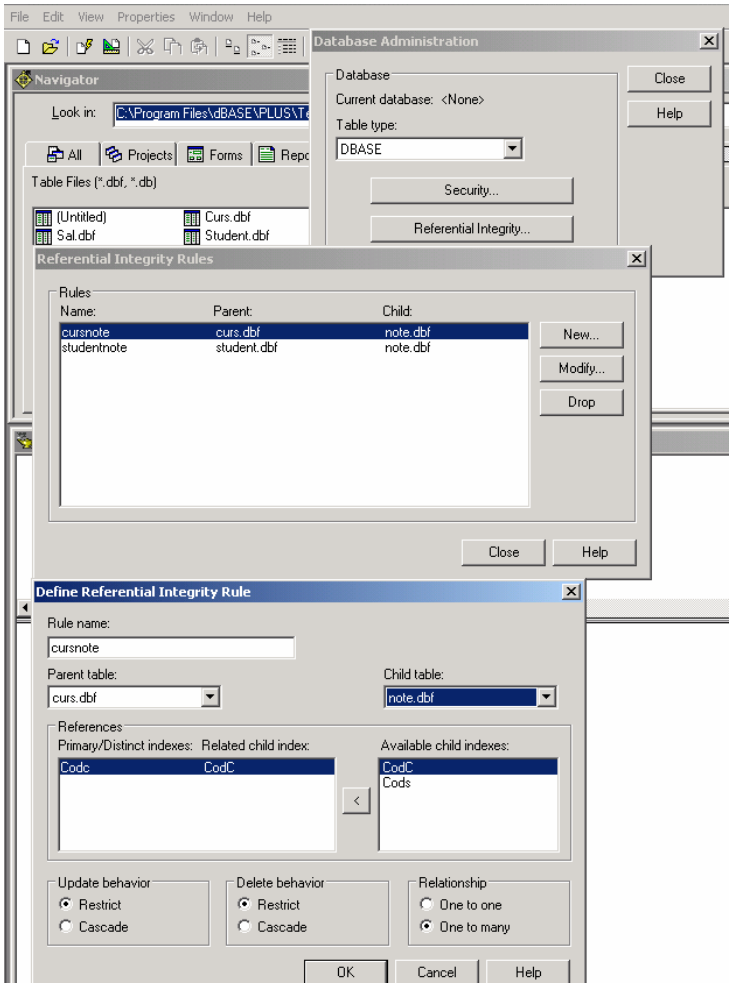
Se trece la definirea restricției de Integritate referențială în care tabelele Student și Curs sunt **părinte** și au cheile primare Cods și Codc definite anterior. **Tabela fiu** va fi Note care are cheile secundare Cods și Codc și referă cheile primare din tabelele părinte.

Definirea integrității referențiale se face astfel:

- se selectează din meniul principal **File** și opțiunea **Data Administration**
- va apare fereastra **Data Administration** și se selectează **Referential Integrity**
- in fereastra **Referential Integrity** se selectează butonul **New**
- se va deschide fereastra **Define Referential Integrity Rule** unde se va selecta tabela părinte și fiu și câmpurile cheie din cele 2 tabele.

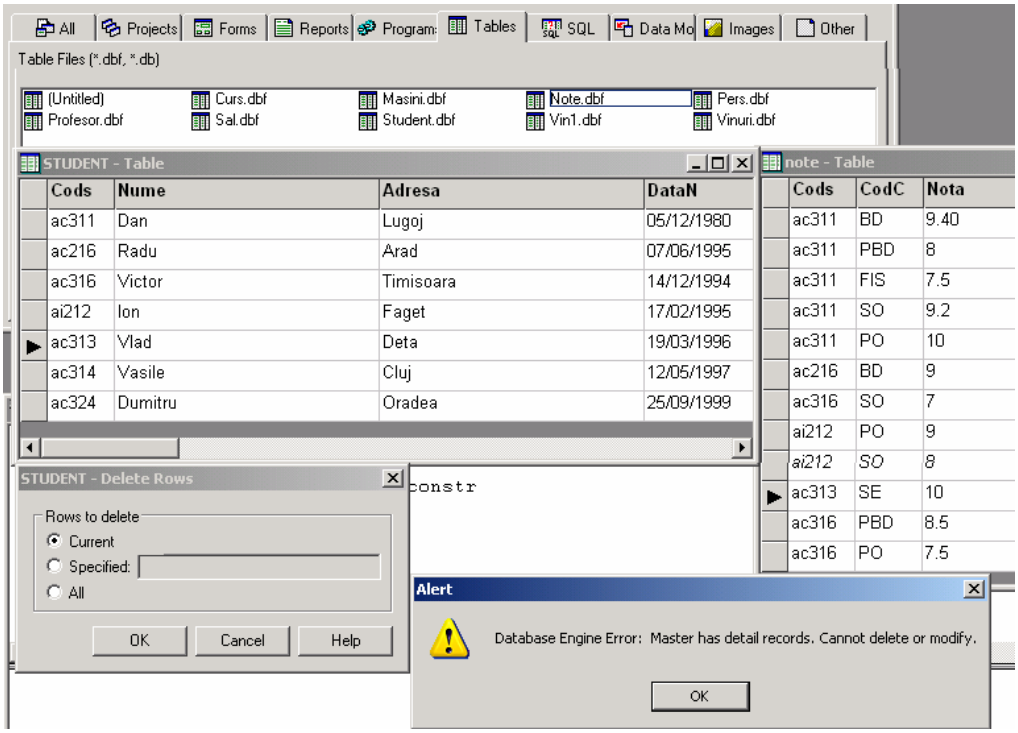
După definirea restricțiilor de integritatea a referinței la orice notă adaugată se va verifica dacă există studentul cu acel Cods și cursul cu acel Codc.





- Opţiunea **Restrict** pentru comanda Delete interzice ştergerea unei înregistrări părinte dacă are înregistrări fiu. Se interzice ştergerea unui student care are note în fişierul Note. Întâi se şterg toate notele şi apoi studentul
- Opţiunea **Cascade** permite ştergerea unui student, dar se şterg în cascadă şi notele studentului respectiv din tabela Note. Astfel nu se admit înregistrări fiu fără înregistrări părinte.

Deschidem fişierul Student şi din meniul principal Table selectăm Delete Row. Confirmăm cu OK ştergerea studentului Vlad având codul ac313, care are nota la disciplina SE. Mesajul de eroare ne interzice fiindcă înregistrarea master are înregistrări fiu (detailed records).
Dacă ştergem notele studentului Vlad putem să-l ştergem din fişierul Student.



Dacă la restricția de integritate a legăturii Student-Note vom da Cascade, putem sterge orice student dar automat i se vor şterge toate notele.

4.8. Program - Afişare note pentru un student

Programul AfisN va afişa toate notele unui student pentru care se dă Cods. Reindexaţi înainte fişierele Student, Curs, Profesor si Note pentru a actualiza indecşi dacă s-au făcut adăugări de înregistrări sau modificări. Se folosesc indexari de tip MDX.

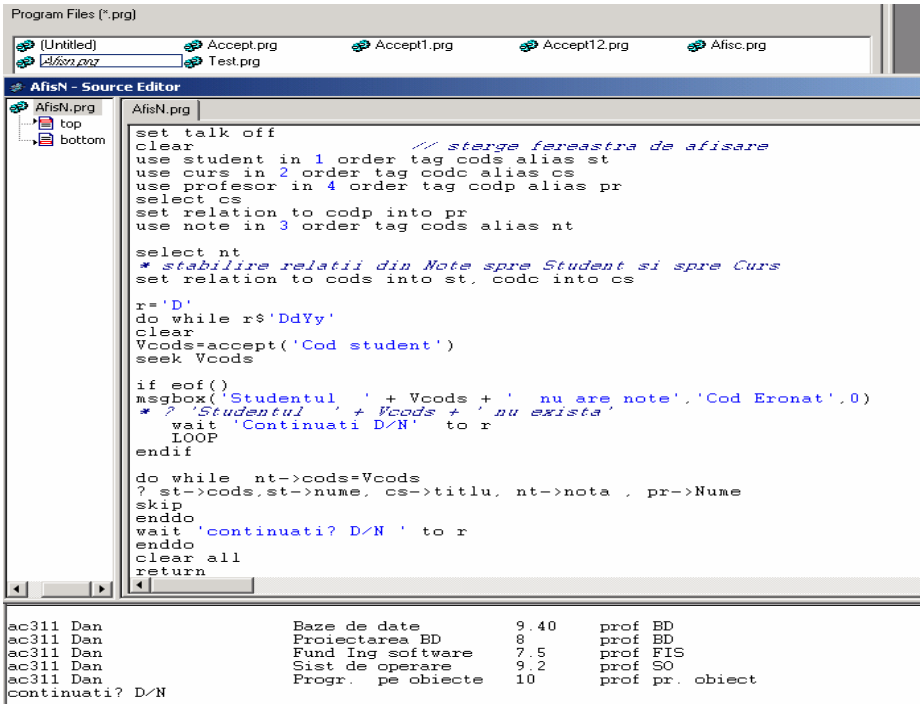
```
Use Student order tag cods exclusiv  // deschidere exclusive fisier student
Reindex                              // reindexare
Use Note order tag cods exclusiv
Reindex
```

Atentie! Cele 2 legături Set relation pentru Note se scriu in aceeaşi comandă, fiindcă se ia ultimul Set relation dacă nu are clauza ADDITIV

In caz de cheie eronată EOF()=.T. inainte de Loop se pune Wait

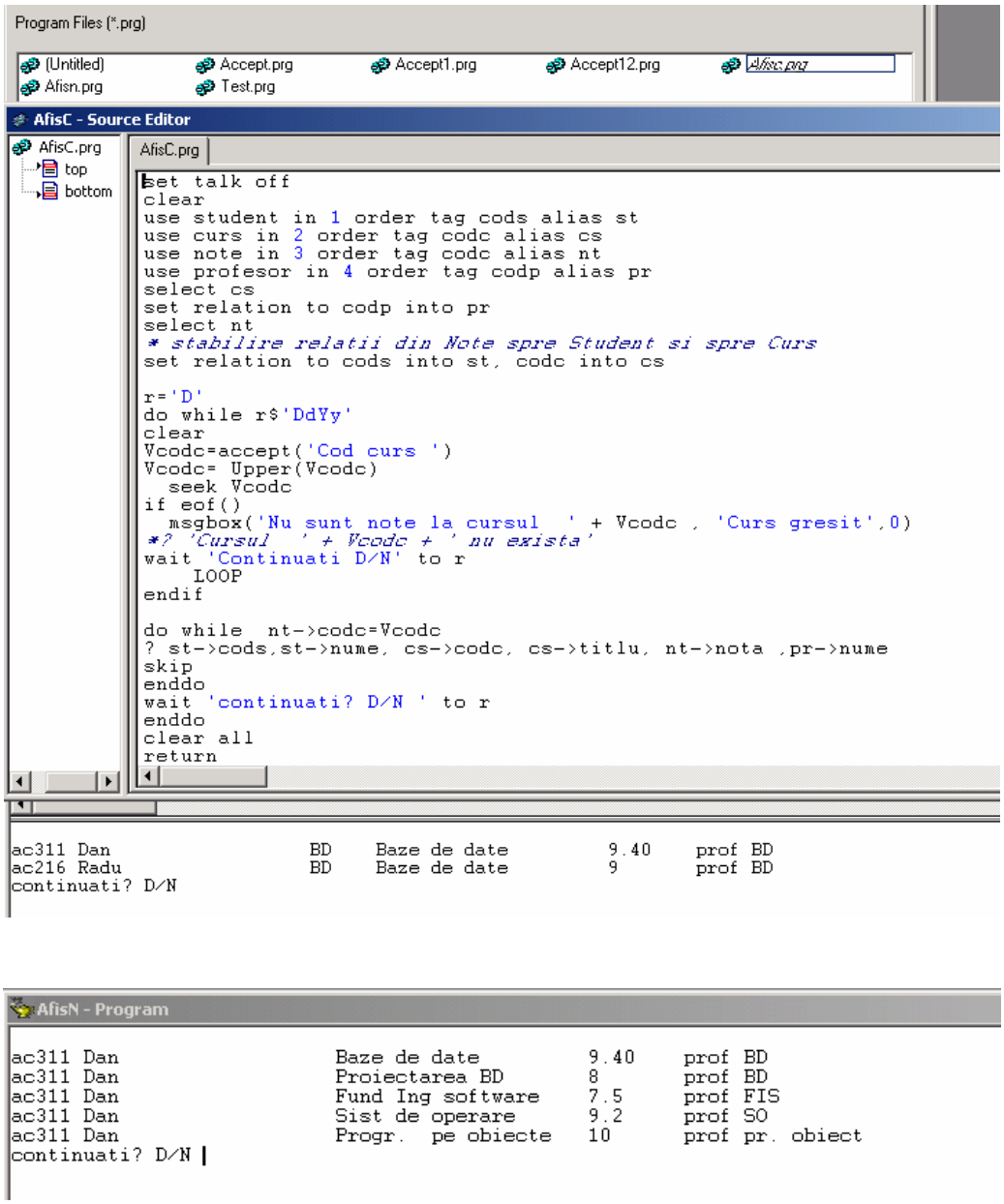
Dacă studentul nu are nici o notă se dă un mesaj prin funcția msgbox, care deschide o fereastra si unde 'Cod eronat' apare in titlul ferestrei.
Programul poate fi dezvoltat prin adăugarea unei tabele
Profesor(Codp, NumeP, Tel), care să aibă cheia primară CodP şi să fie părinte pentru tabela Curs pentru care se va da:
Use Profesor in 4 order tag codp alias PR
Set Relation To Codp Into Pr

În programele de afişare se va adăuga şi câmpul **pr->Nume**, selectarea profesorului făcându-se automat prin Set relation din fisierul Curs.

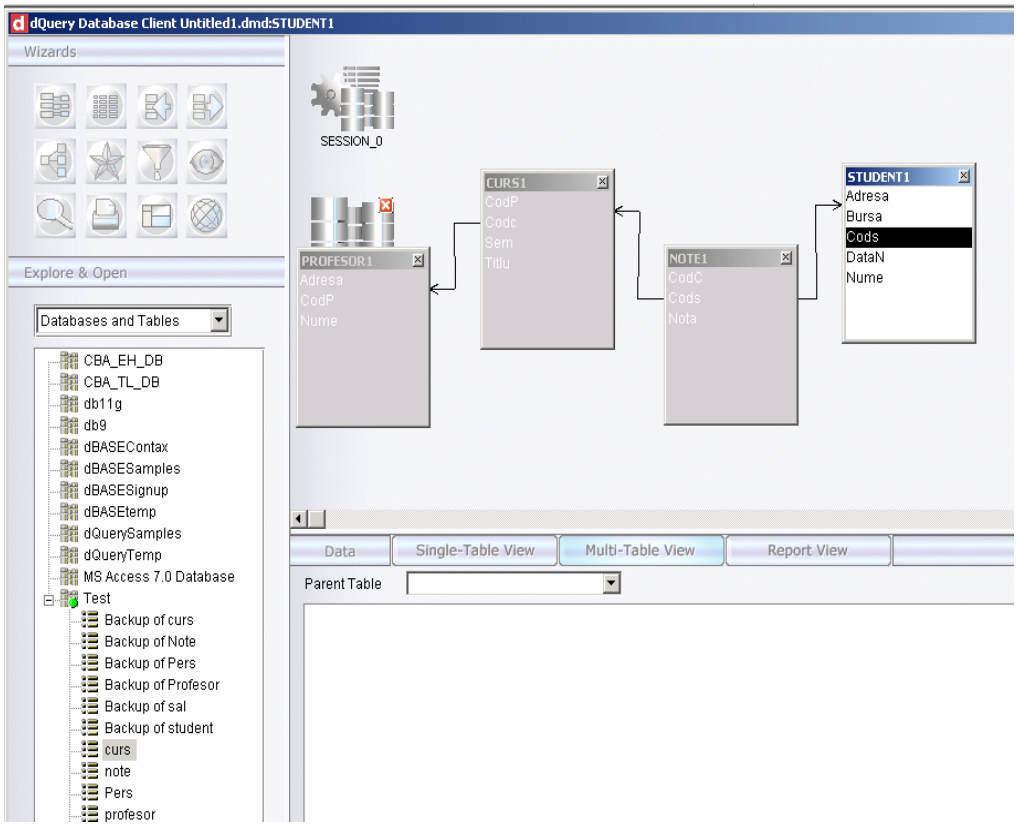


Afişare studenți şi notele obținute la un curs dat prin Codc

Modificând puțin programul se pot afişa cu programul AfisC toti studenții si notele obținute la un anumit curs specificat prin Codc (BD în exemplul dat). Codul cursului introdus se converteşte în litere mari (UPPER) fiindcă în fişierul Curs codurile sunt cu majuscule.



Structura logică a bazei de date se poate reprezenta printr-un utilitar din dQuery o componentă dBase Plus.


dQuery Database Client Untitled1.dmd:STUDENT1
Wizards
Explore & Open
Databases and Tables
CBA_EH_DB
CBA_TL_DB
db11g
db9
dBASEContax
dBASESamples
dBASESignup
dBASEtemp
dQuerySamples
dQueryTemp
MS Access 7.0 Database
Test
Backup of curs
Backup of Note
Backup of Pers
Backup of Profesor
Backup of sal
Backup of student
curs
note
Pers
profesor
SESSION_0
CURS1
CodP
Codc
Sem
Titlu
PROFESOR1
Adresa
CodP
Nume
NOTE1
CodC
Cods
Nota
STUDENT1
Adresa
Bursa
Cods
DataN
Nume
Data
Single-Table View
Multi-Table View
Report View
Parent Table

5. UTILIZARE SUBPROGRAME

5.1. Proceduri şi funcţii utilizator

Pentru a realiza o programare modulară, programe uşor urmărit, de depanat şi dezvoltat ulterior prin adăugări sau modificări, se recomandă scrierea unui program principal din care se cheamă Proceduri şi Funcţii. Fiecare procedura sau funcţie va implementa un algoritm bine determinate şi va avea parametrii propri de intrare şi ieşire.

In subprograme (proceduri sau funcţii) se utilizează:

- parametrii formali specificaţi prin comanda PARAMETER, care sunt **variabile locale** ale subprogramului şi nu pot fi folosiţi la chemarea altor subprograme,
- variabile private definite în subprogram ca variabile de lucru,
- nume de câmpuri de fişier din orice zonă (utilizând prefixarea),
- variabile globale (Public) definite într-o zonă comună accesibile din orice modul de program unde au fost definite.

Variabilele globale se declara in secţiunile unde se folosesc (program principal sau subprogram) prin:

PUBLIC lista _var

Variabilele globale se folosesc ca variantă de transmitere a parametrilor spre subprograme, mai ales când lista de parametrii este mare.

O procedura are structura:

PROCEDURE nume_proced
PARAMETER lista_var_formale
PUBLIC lista_var_globale
PRIVATE lista_var_locale
---------- - secvenţa comenzi definite procedura

RETURN - sfârşit de procedura

Numele procedurilor nu pot fi cuvinte rezervate ale limbajului dBase (nume de

comenzi, clauze sau funcţii standard).
Variabilele PRIVATE (implicit cele utilizate în procedură)se pot utiliza numai in procedura unde sunt definite, sunt alocate in stiva şi dispar după execuţia procedurii.

PRIVATE lista_var ALL [LIKE/EXCEPT generic]

Parametrii formali ai unei proceduri sau funcţii se dau prin:

PARAMETER lista_var_formale

Sfârşitul de procedura se marchează prin începutul altei proceduri sau funcţii.
O procedura se cheamă prin:

DO nume_proced WITH lista_param

Parametrii de apel trebuie sa fie de acelaşi tip cu cei formali din definiţie.
Variabilele publice nu se pot şterge cu RELEASE lista_var ci numai cu CLEAR ALL

Funcţia poate avea mai mulţi parametrii de intrare dar numai o valoare transmisă la ieşire prin numele funcţiei.
Structura unei funcţii este asemănătoare cu a procedurii, dar valoarea de ieşire se transmite în argumentul comenzii RETURN:

FUNCTION nume_functie (lista_var_formale)
PUBLIC lista_var_globale
---------- - secvenţă comenzi definiţie funcţie

RETURN (exp) - ieşire cu transmitere valoare calculata.

O funcţie poate fi utilizată in expresii cu operanzi de acelaşi tip cu ea.
Tipul funcţiei e dat de tipul expresie din RETURN.
Utilizarea procedurilor permite refolosirea unor secvenţe de program.
Procedurile pot apela proceduri chiar şi recursiv.
Procedurile se pot scrie după programul principal sau se pot grupa într-un fişier de proceduri ce se va declara prin:

SET PROCEDURE TO fis_proc [ADDITIV]
SET LIBRARY TO

Pentru a deschide mai multe fişiere pentru proceduri se va specifica clauza ADDITIV, altfel se va lua numai ultima specificaţie SET PROCEDURE.
Închiderea unui fişier de proceduri se face prin:

SET PROCEDURE TO sau
CLOSE PROCEDURE

Atentie! Comanda CLEAR ALL şterge toate setările, toate variabilele din memorie, închide fişierele de date şi selectează prima zona de lucru.

5.2. Proiectare aplicaţie simplă de Bază de Date universitară

La proiectarea unei aplicaţii se recomanda următoarele faze:

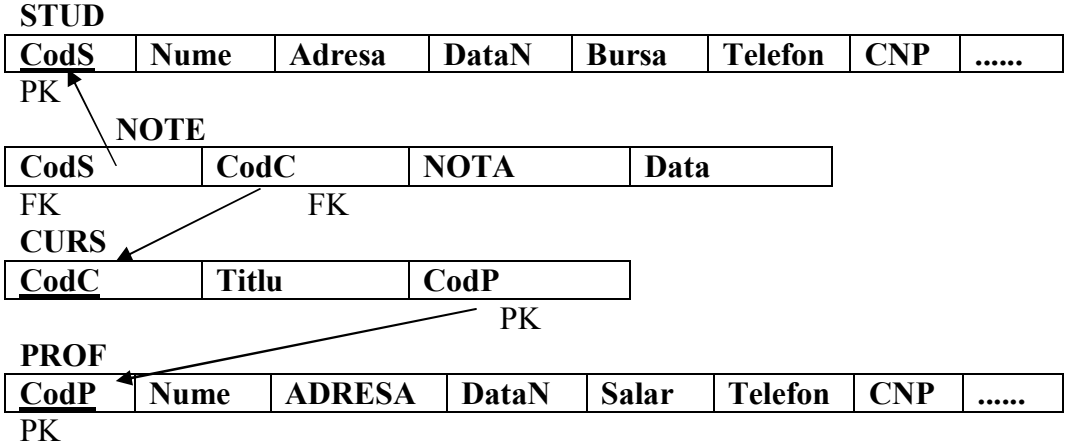
- **Analiza aplicaţiei** şi alegerea tabelelor(fişierelor) utilizate
- **Stabilirea structurii** tabelelor componente ale BD şi a legăturilor dintre ele (câmpuri, fişiere index, relaţii). Pentru fiecare fişier se poate stabili un câmp cheie primară - PK, care să identifice univoc fiecare înregistrare şi câmpurile care fac referinţă spre alte tabele (foreign key - FK)
- **Stabilirea funcţiilor** care trebuie realizate (cerute de beneficiar)
- **Stabilirea procedurilor** ce realizează fiecare funcţie si a algoritmilor folosiţi
- **Stabilirea meniurilor** de selecţie a funcţiilor si ierarhia meniurilor;
- **Stabilirea ierarhiei** de chemare a **procedurilor** pentru fiecare funcţie
- **Scrierea programului principal** care cuprinde meniul principal si secvenţa de selecţie.
- **Scrierea si testarea fiecărei proceduri** si a programului in ansamblu.

Se considera o Bază de date pentru evidenţă universitară care conţine informaţii despre studenţi, cursuri şi profesori care sunt entităţi bine definite.

Structura BD se proiectează urmărind eliminarea redundanţei informaţiilor.

Se vor alege fişierele componente în care informaţiile nu trebuie să se repete şi spaţiul liber trebuie eliminat. Suntem tentaţi să creăm un fişier de cursuri, care să conţină numele profesorului şi gradul său. Numele se regăseşte şi în fişierul de profesori unde se găsesc toate datele personale (CodP, data naşterii, salar, locul naşterii, adresa, telefon acasă, telefon la servici, mobil,..). Atunci în fişierul de cursuri vom introduce numai codul profesorului (CodP). Se elimină astfel şi repetare numelui dacă un profesor este titular la mai multe cursuri.
In fişierul de studenţi vom introduce datele personale ale studentului (CodS, nume, adresa, data naşterii, telefon, bursa,..). Pentru fiecare student trebuie

introduse toate notele şi cursul la care au fost obţinute.
Pentru a simplifica lucrurile vom introduce entitatea note, ca o entitate de legătură între studenţi şi cursuri, care va conţine CodS, CodC, Nota, Data. CodS şi CodC sunt referinţe care definesc o legătură între fişierul STUD şi CURS, iar ultimele 2 câmpuri sunt informaţii cantitative ce caracterizează legătura. Structura BD fără redundanţe va fi cea de mai jos:



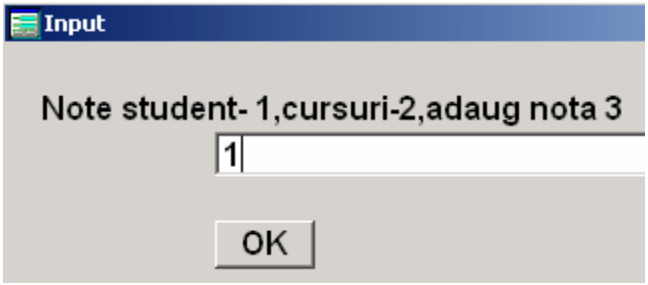
Câmpuri cheie sunt CodS în fişierul STUD, CodC în CURS, CodP în PROF şi pentru ele se pot crea fişiere index pentru căutare în acces direct şi parcurgere ordonată. Se pot crea fişiere index secundare şi pentru câmpul Nume şi Telefon în fişierele STUD şi PROF Titlu în CURS, dar ele nu sunt chei pentru fişier. Cheie ar putea fi câmpul CNP (Cod Numeric Personal) din STUD şi PROF, dar ele nu sunt funcţionale pentru această aplicaţie.
In fişierul NOTE cheia poate fi formată din concatenarea câmpurilor CodS+CodC care identifică univoc o notă.

Prezentăm mai jos un exemplu în care programele de afişare note studenţi AfisN şi afişare note la cursuri AfisC, sunt transformate în proceduri care se apelează prin dialog dintr-un program principal Puniv. Programul poate fi ulterior dezvoltat pentru aplicaţia BD universitară prin adăugarea de noi funcţionalităţi prin scrierea unor noi proceduri:

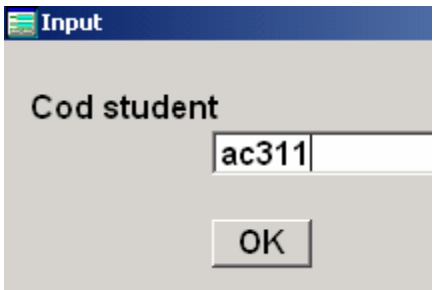
- Afişare sau modificare date personale studenţi si profesor
- Afişare sau modificare informaţii despre cursuri
- Adăugare noi înregistrări în fişierele Student, Curs, Profesor
- Adăugare note pentru studenţi la discipline prin verificarea integrităţii referinţei
- Crearea iniţială aBD prin copierea unor structuri de fişiere memorate

***Puniv** program principal afisare note studenti si note la cursuri

```
* Se utilizeaza procedurile AfisNote si AfisNcurs
set talk off
clear              // sterge fereastra de afisare

* Deschidere fisiere si indexi in diferite zone de lucru
* Stabilire legaturi prin Set Relation
use student in 1 order tag cods alias st
use curs in 2 order tag codc alias cs
use profesor in 4 order tag codp alias pr
select cs
set relation to codp into pr
use note in 3 order tag cods alias nt
select nt
* stabilire relatii din Note spre Student si spre Curs
set relation to cods into st, codc into cs
r=accept(' Note student- 1,cursuri-2,adaug nota 3')
```

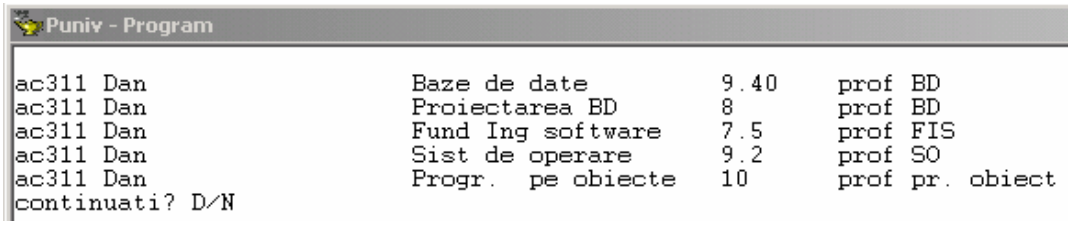



```
Do case
  case r='1'
  Do AfisNote
  case r='2'
  Do AfisNcurs
  case r='3'
  select 3
  browse          // afisare tabel note pentru completare
other
msgbox('Functie eronata: '
+r,'Eroare',0)   // fereastra mesaj de
eroare
clear all          // inchidere fisiere
cancel
endcase
return
```




```
 Procedure AfisNote                                    //
tabela Cursuri si Note pentru un student
  select 3
set order to  cods              // index master cods in Note
```

```
r='D'
do while r$'DdYy'
clear

Vcods=accept('Cod student')
seek Vcods

if eof()
msgbox('Studentul  ' + Vcods + '  nu are note','Cod Eronat',0)
* ? 'Studentul  ' + Vcods + ' nu exista'
   wait 'Continuati D/N'  to r
   LOOP
endif

do while  nt->cods=Vcods
? st->cods,st->nume, cs->titlu, nt->nota , pr->Nume
```




```
skip
enddo
wait 'continuati? D/N ' to r
enddo
clear all
return

Procedure AfisNcurs          // afisare studenti si notele obtinute la un curs
  select 3
  set order to codc          // index master codc in Note
r='D'
do while r$'DdYy'
clear
Vcodc=accept('Cod curs ')
Vcodc= Upper(Vcodc)
  seek Vcodc
```

```
if eof()
  msgbox('Nu sunt note la cursul  ' + Vcodc , 'Curs gresit',0)
*? 'Cursul  ' + Vcodc + ' nu exista'
wait 'Continuati D/N' to r
   LOOP
endif

do while  nt->codc=Vcodc
? st->cods,st->nume, cs->codc, cs->titlu, nt->nota ,pr->nume
skip
enddo
wait 'continuati? D/N ' to r
enddo
clear all
return
```

5.3 Funcția Msgbox()

Pentru afişarea rapidă a erorilor şi cu un design plăcut se va folosi funcția Msgbox(), care permite afişarea într-un Form în care apar 2 mesaje şi unul sau mai multe butoane:

Msgbox(‘Mesaj_eroare’,’Titlul_ferestrei’,Cod)

Exemplu:

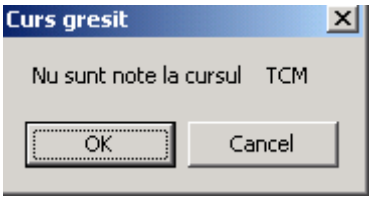
```
 msgbox('Nu sunt note la cursul  ' + Vcodc , 'Curs gresit',0)  // Cod=0 un buton OK
```

Codul funcției apelate specifică ce butoane vor fi afişate:

0	OK
1	OK, Cancel

2	Abort, Retry, Ignore
3	Yes, No, Cancel
4	Yes, No
5	Retry, Cancel

.....
r=msgbox('Nu sunt note la cursul ' + Vcodc , 'Curs gresit',1)
IF r=1
Loop
Else
Cancel
EndIf
….



Prin variabila r se returnează codul butonului selectat, care poate fi testat pentru a alege modul de continuare a programului.

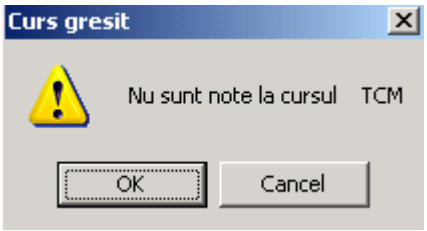
În tabelul următor sunt date codurile butoanelor returnat de funcția msgbox.

Pushbutton	Return value
OK	1
Cancel	2
Abort	3
Retry	4
Ignore	5
Yes	6
No	7

Dacă se doreşte afişarea unei figuri de atenţionare în fereastra de eroare la cod se adaugă 16, 32, 48, 64:

Number to add	Icon displayed
16	
32	
48	
64	

r=msgbox('Nu sunt note la cursul ' + Vcodc , 'Curs gresit',1+48)



5.4. Functia ACCEPT pentru dialog cu utilizatorul

Dăm mai jos programul sursă pentru funcţia ACCEPT utilizată deja care utilizează Proceduri şi Funcţii, dar şi obiecte Windows care vor fi discutate ulterior.
Comunicarea între proceduri se face prin variabila Publica V returnată şi prin parametrul p (mesaj).
Formal Procedurile se pot înlocui cu Funcţii, dar înţelegerea este uneori mai greoaie.
Programul ACCEPT poate fi modificat şi simplificat transformând procedura ACCEPT12 în functia ACCEPT1 şi se elimină prima funcţie.

accept1.prg ACCEPT.prg

```
function accept(p)          // parametru p este mesajul afisat
public v                    //   v este valoarea introdusa
do accept12   with p
return v

Procedure accept12
param p
public   v
v=space(30)
define  form f1
   f1.Text ='Input'
   f1.mdi=.f.       // tipul ferestrei este modala pentru dialog
   f1.height=10
   f1.width=70
defi text t1 of f1 at 1,2 prop Text 'Mesaj', fontsize 12, fontbold .t.
   f1.t1.width=50
   f1.t1.text=p             //  text t1 contine mesajul p
defi entryfield e1 of  f1 at 2,15 prop width 20, fontsize 12, fontbold .t.
   f1.e1.width=50
   f1.e1.datalink='v'       // valoarea returnata  asociata cu EntryField E1
defi pushbutton b1 of f1 at 4,15 prop text 'OK', fontsize 12, fontbold .t.

   f1.b1.onclick ={;do pOK }
 f1.readmodal()          // deschidere fereastra modala
   return

procedure pOK
public v
v= f1.e1.value         // Valoarea introdusa atribuita lui V
f1.close()
return
```

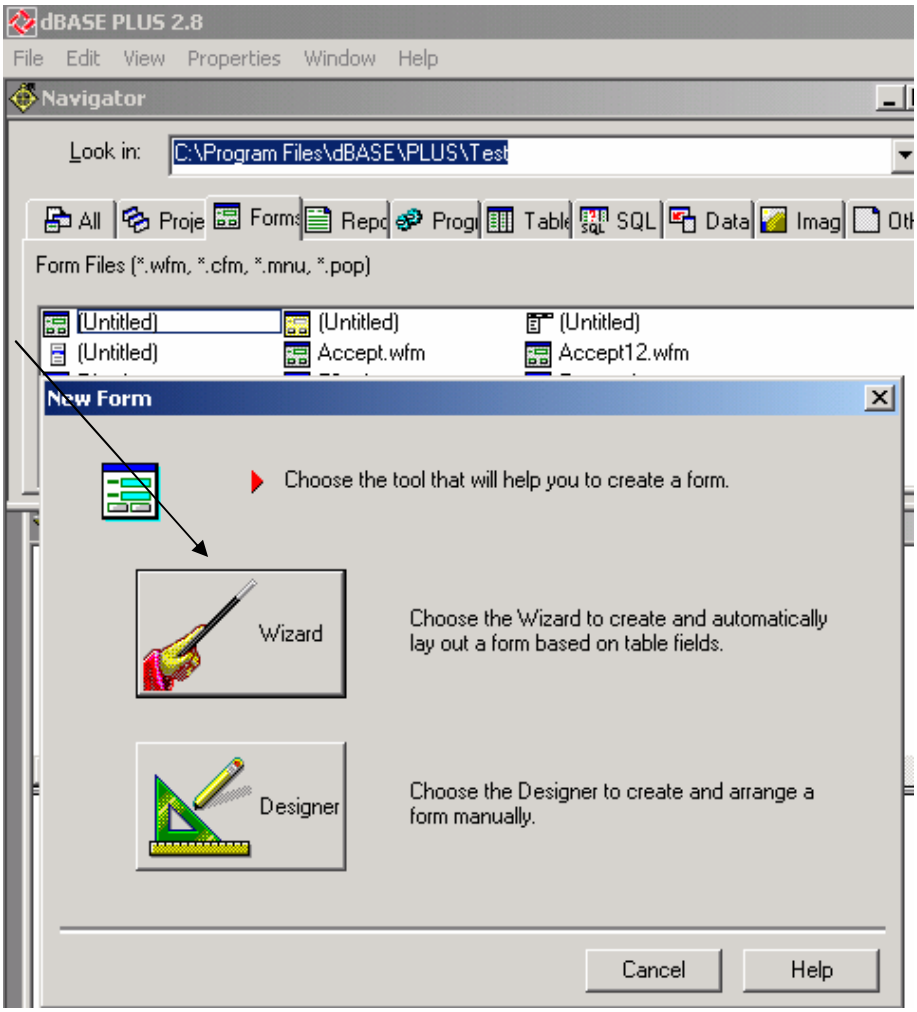
accept1.prg

```
Function accept1(p)          // parametru p este mesajul afisat
public   v                    //  v este valoarea introdusa
v=space(30)

define  form f1
   f1.Text ='Input'
   f1.mdi=.f.          // tipul ferestrei este modala pentru dialog
   f1.height=10
   f1.width=70
defi text t1 of f1 at 1,2 prop Text 'Mesaj', fontsize 12, fontbold .t.
   f1.t1.text=p        //  text t1 contine mesajul p
   f1.t1.width=50
defi entryfield e1 of  f1 at 2,15 prop width 20, fontsize 12, fontbold .t.
   f1.e1.width=50
   f1.e1.datalink='v'          // valoarea returnat  asociata cu EntryField E1
defi pushbutton b1 of f1 at 4,15 prop text 'OK', fontsize 12, fontbold .t.

f1.b1.onclick ={;do pOK }
    f1.readmodal()             // deschidere fereastra modala
return   v

procedure pOK
public v
v= f1.e1.value           // Valoarea introdusa atribuita lui v
f1.close()
return
```

6. PROIECTARE INTERFETE GRAFICE CU DESIGNER

6.1. Proiectare FORM (fereastră)

In dBase există un Designer (wizard) pentru proiectarea interfețelor grafice, care ajută la realizarea unor programe ce folosesc **obiecte standard Windows** şi utilizează **programarea orientată pe evenimente**. Până acum s-a folosit programarea clasică care utilizează comenzile procedurale pentru a selecta diferite funcții(DO CASE,DO WHILE).



Interfața grafică permite utilizarea obiectelor **TEXT** la care se pot alege diferite proprietăți ca dimensiuni, fonturi, culori,.... Designer-ul este un generator de cod de program dBase, care definește o clasă **FORM** pe care se găsesc obiecte derivate din clase Windows (Text, EntryField, PushButtone, RadioButton, ListBox, ComboBox, Browse, Image,...). Programatorul proiectează o machetă de formular (FORM) pe care plasează obiectele de pe o **Paletă de obiecte,** pe care le personalizează modificându-le proprietățile cu ajutorul unei **Palete de proprietăți** afișate de **Inspector** (obiect al designer-ului).

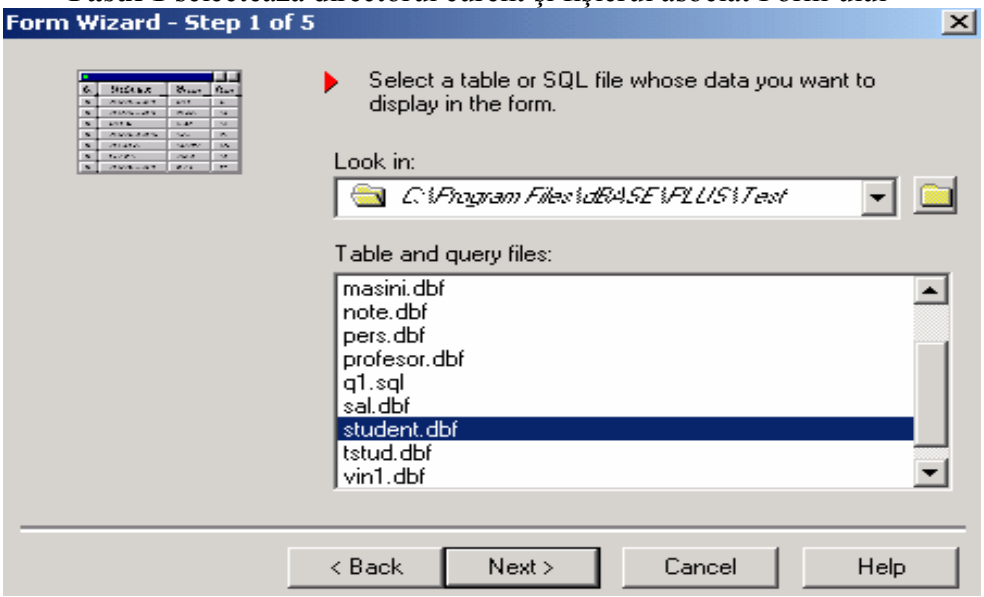
Activarea Designer-ului se poate face:

- Prin selectare FORMS în Navigator şi DblClick pe Untitled
- Din meniul FILE se selectează NEW şi apoi FORM

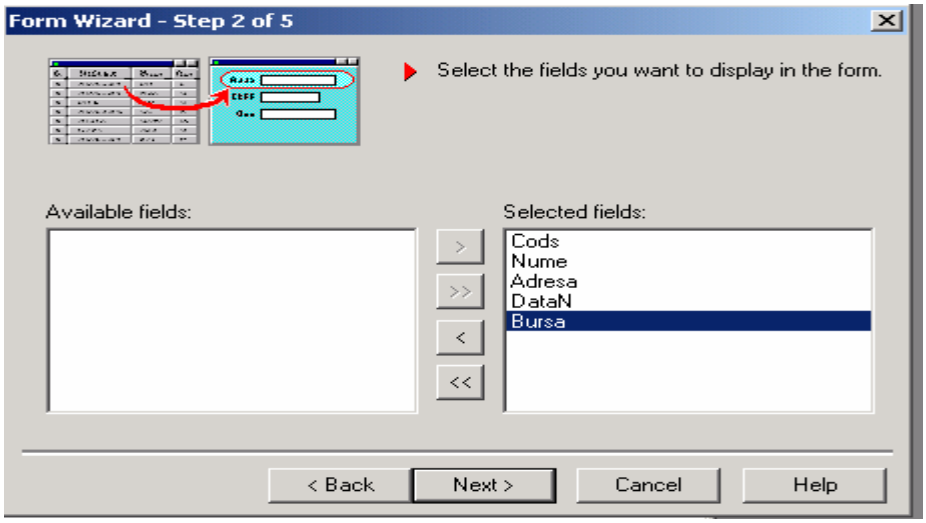
Se va afişa **meniul New Form** în care butonul DESIGNER deschide un Form gol pe care se pot completa obiecte de către programator cu o asistență minimă.

Cea mai simpla proiectare se realizează selectând butonul **Wizard** care permite legarea Form-ului de un fişier şi referirea directă la câmpurile acestuia. Acest meniu are 5 paşi.

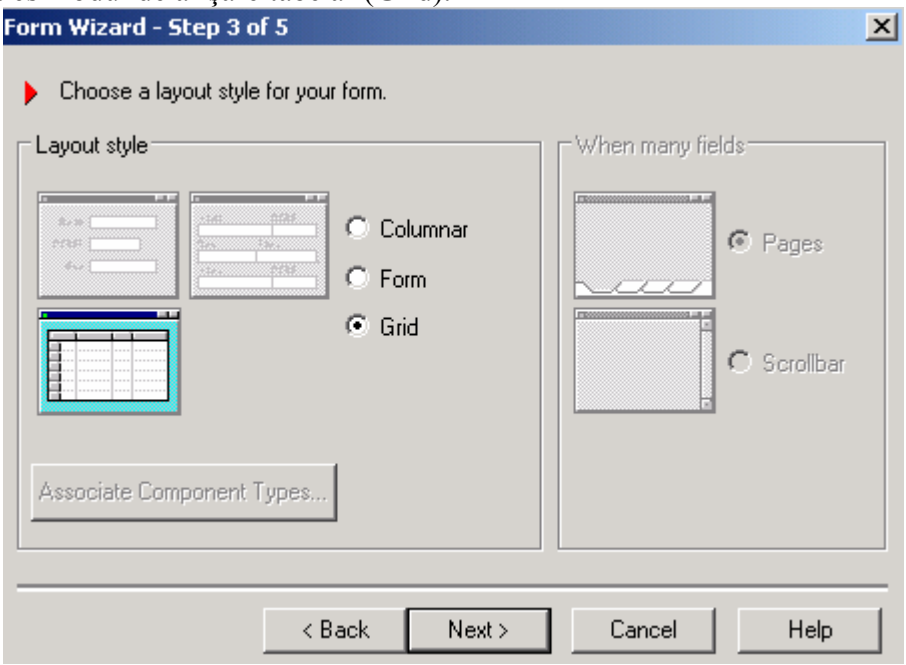
Pasul 1 selectează directorul curent şi fişierul asociat Form-ului



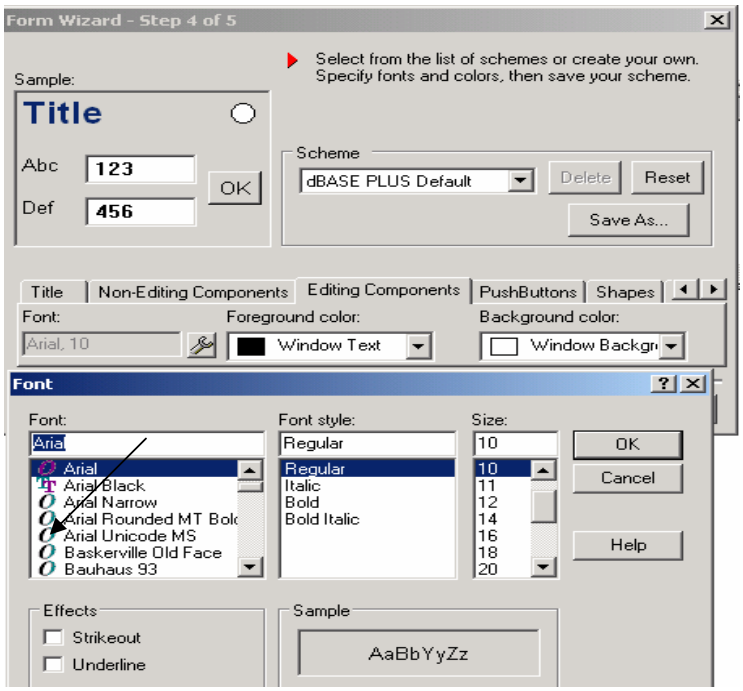
Pasul 2 permite selectarea cu butoane a câmpurilor care vor fi utilizate in aplicație şi vor fi trecute în **paleta de câmpuri**.



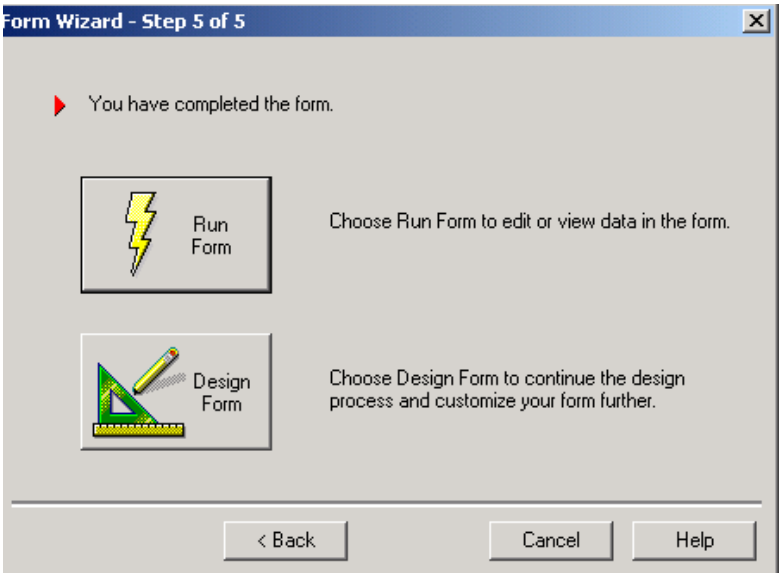
Pasul 3 permite alegerea modului de afişare a înregistrărilor din fişier pe Form. S-a ales modul de afişare tabelar (Grid).



Pasul 4 permite alegerea culorilor şi fonturilor pentru Titlu, pentru Text(Non Editing), EntryField (Editing components), PushButton, Form.



Pasul 5 permite generarea, compilarea şi rularea programului (Run Form) sau continuarea proiectării Form-ului pentru a plasa obiecte pe el (Design Form).
Vom continua cu Design Form pentru a personaliza şi completa aplicația.

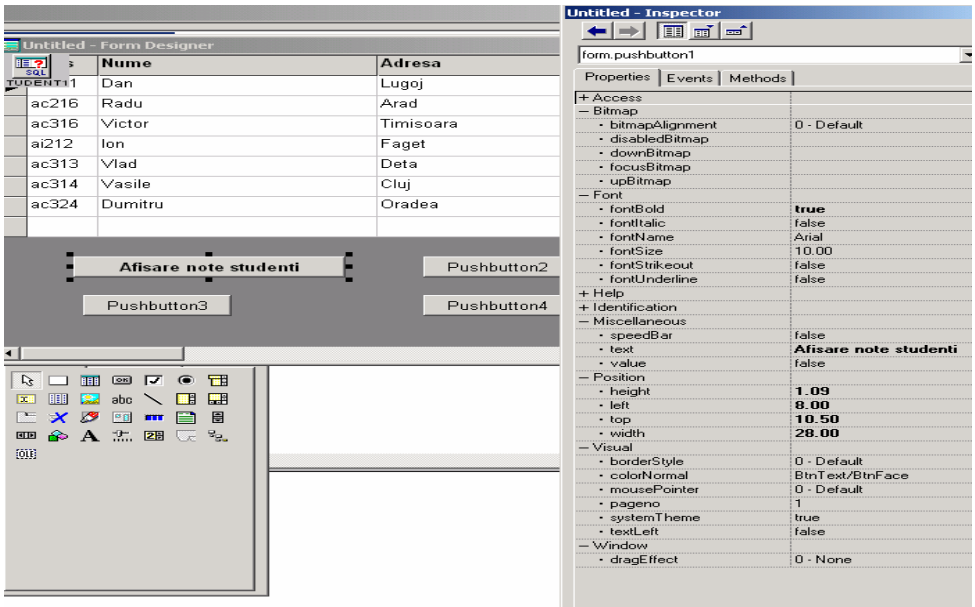


Pe ecran apare macheta într-o forma generală cu afişarea conţinutului fişierului.
Dimensiunile Form-ului şi tabelului de afişare pot fi modificate cu mouse-ul.

Lângă Form se găseşte **paleta de obiecte** de unde putem selecta cu mouse-ul obiecte pe care să le punem pe Form. S-au plasat pe Form câteva butoane care sunt de aceeaşi dimensiune şi denumite automat cu numele clasei şi un număr. Ele sunt simple figuri inerte care trebuie personalizate prin modificarea proprietăţilor.

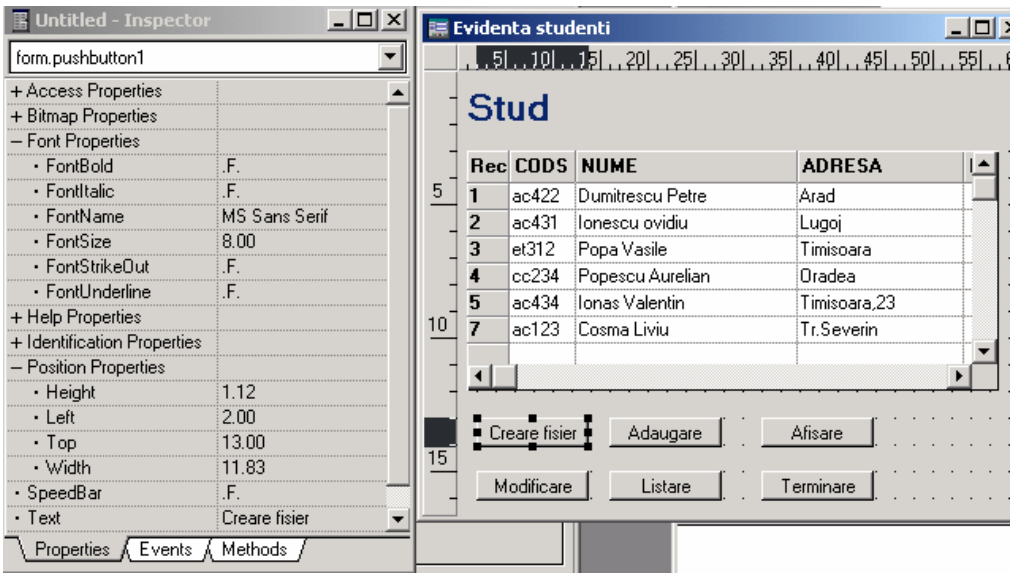
6.2. Modificare proprietăţi obiecte

Modificarea obiectelor se face prin selectarea lor cu mouse-ul şi afişarea paletei de proprietăţi folosind unealta **Inspector** care se activează cu **click dreapta** pe Form.
S-a modificat Titlul Form-ului (proprietatea Text) unde s-a introdus Evidenta studenti.
Se poate modifica dimensiunea şi coordonatele Form-ului.



Selecţia unui alt obiect de pe Form se poate face prin Click cu mouse-ul pe obiect sau din ComboBoxul din partea de sus din Inspector. Selectăm PushButon1 îi modificăm dimensiunea cu mouse-ul, textul afişat pe el, culoarea, tipul şi dimensiunea font-ului folosit din Inspector. **Grupele de proprietăţi se afişează cu + şi se închid cu -.** Dimensiunea şi poziţia pot fi

modificate şi în paleta de proprietăţi din Inspector.
La fel se procedează şi cu celelalte butoane.



6.3. Modificare proprietăţi comportamentale

În programarea orientată pe obiecte, obiectele au proprietăţi care sunt încapsulate şi se pot moşteni la obiectele din subclasele derivate. Proprietăţile obiectelor sunt:

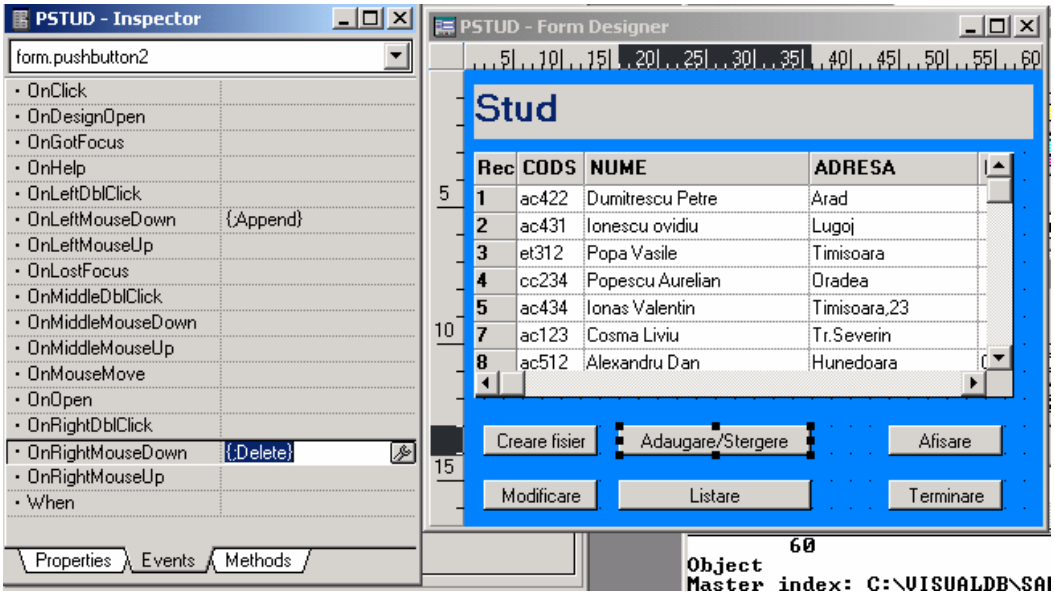
- **Proprietăţi structurale** (Properties) care descriu obiectul (nume, adresa, data naşterii, salar, culoare, tip font, înălţime, lăţime, greutate, distanţă,..)
- **Proprietăţi comportamentale** care caracterizează **reacţia obiectelor la evenimente**. Ele se precizează prin asocierea unei comenzi sau proceduri unui eveniment din tabela Events din Inspector. Evenimentele pot fi click stânga sau dreapta pe mouse (OnLeftMouseDown), deschidere sau închidere Form (OnOpen, OnClose), mişcare mouse pe Form (OnNavigate), la mişcare obiect (OnMove), sau selectare sau deselectare obiect (OnGotFocus, OnLostFocus), schimbare dimensiune (OnSize), la schimbarea unor valori dintr-un EntryField (OnChange).
- **Metodele** care sun proceduri interne ale obiectului care pot fi utilizate în prelucrare. Pentru Form sunt deschidere, închidere sau stergere Form (F1.Open(), F1.Close(), F1.Release()), reafişare Form (F1.Refreash())

Programarea bazată pe evenimente presupune ataşarea unor proceduri la evenimente. Procedurile se lansează astfel asincron la apariţia evenimentului asociat obiectului. Un obiect poate reacţiona diferit la diferite evenimente. Fiecărui eveniment care acţionează asupra obiectului (ex.buton) i se poate ataşa o procedură distinctă. Deci procedura nu i se ataşează obiectului ci evenimentului care acţionează asupra acelui obiect.

La proiectarea unei aplicaţii se recomandă:

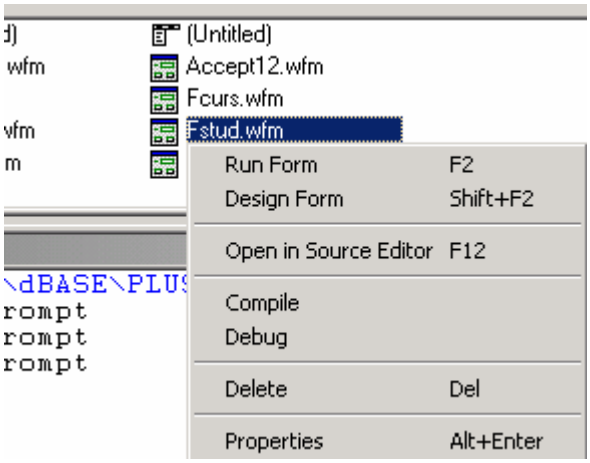
- La **OnOpen** pentru obiectul Form să se asocieze o procedură în care **se deschid toate fişierele** (tabelele) bazei de date.
- La **OnClose** pentru Form **se închid toate fişierele** bazei de date (Close Database)

In exemplul prezentat s-au asociat comenzi simple butoanelor Adăugare - Append Afisare - Browse, Modificare – Edit, Listare – List, Terminare – Close database. S-a modificat butonul Adaugare in Adaugare/Stergere şi s-a asociat evenimentului **Click stânga** comanda **Append** iar pentru **click dreapta** comanda **Delete.** Dacă evenimentului i se asociază o singură comandă ea se pune direct în tabelul Inspector sub forma **{;Append}.** Dacă evenimentului i se asociază o procedură, se va da un click pe unealta din dreapta rândului, care va deschide o procedură care va fi scrisă de programator.



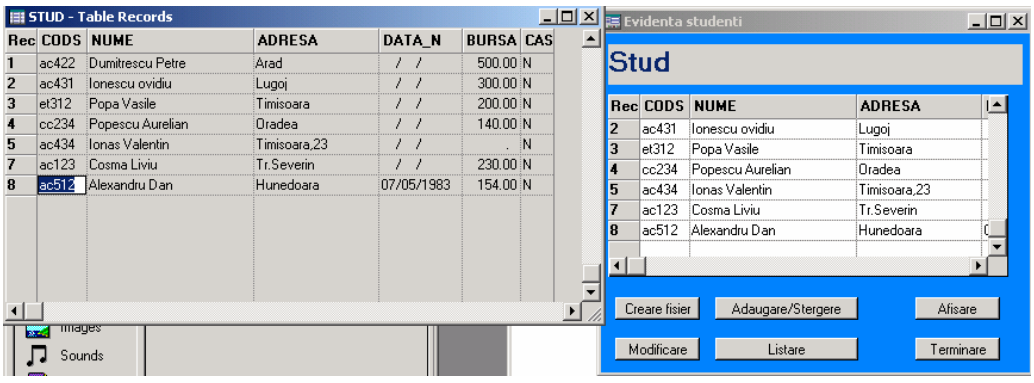
Macheta proiectată cu Design-erul va servi la generarea unei clase obiect Form care conține toate obiectele adăugate. La închiderea ferestrei Form Designer se precizează numele programului Fstud, care va avea extensia **WFM (Windows Form).**

Programul va fi generat, compilat şi pus în execuție prin deschiderea Form-ului dând click dreapta pe programul Fstud.wfm din navigator şi selectând **RunForm**.



Un Form poate fi deschis din nou în modul design pentru completare sau corectare alegând Design Form în meniu.

Butonul Creare fişier poate să aibă pe click stânga procedura de creare care copiază structura fişierului de referință Rstud.



Designer-ului acceptă un singur fişier deschis, care este dat ca o proprietate **rowset** definită printr-o comandă SELECT de SQL.

Este preferabilă proiectarea completă a Form-ului de către programator.

Programul generat poate fi afişat şi modificat selectând **Open source Editor** pe click dreapta.

```
local f
Use stud
f = new PSUD1FORM()
  f.Open()
.....
```

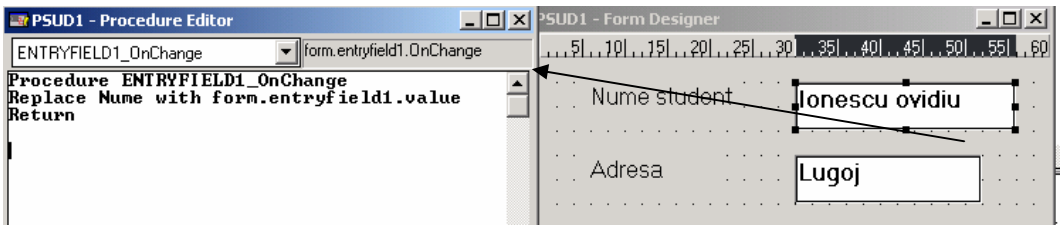
6.4. Proiectare Form folosind Designer-ul

Mai jos se prezintă un exemplu de Form realizat direct cu Designer-ul care generează un obiect Form pe care sunt plasate obiectele Texte, EntryField şi Pushbutton.
Programul principal conţine numai comanda de creare a unui obiect Form şi deschiderea lui.

In EntryField-uri se va afişa valoarea câmpului asociat din înregistrarea curentă. Pentru aceasta se modifică programul generat adăugând la evenimentul OnOpen o procedură care deschide fişierul folosit şi stabilesc câmpurile asociate celor două EntryField-uri.

```
Procedure OnOpen
USE Stud
form.entryfield1.DataLink =’stud->Nume’
form.entryfield1.DataLink =’stud->adresa’
return
```

Menţionăm că **valorile din obiectele EntryField se pot modifica, dar nu se modifică direct valoarea câmpului asociat din fişier**. Acesta se poate modifica cel mai simplu folosind **evenimentul OnChange** din obiectul EntryField căruia îi asociem o procedură, care înlocuieşte valoarea câmpului cu proprietatea obiectului EntryField.Value, care este specifică obiectelor EntryField.



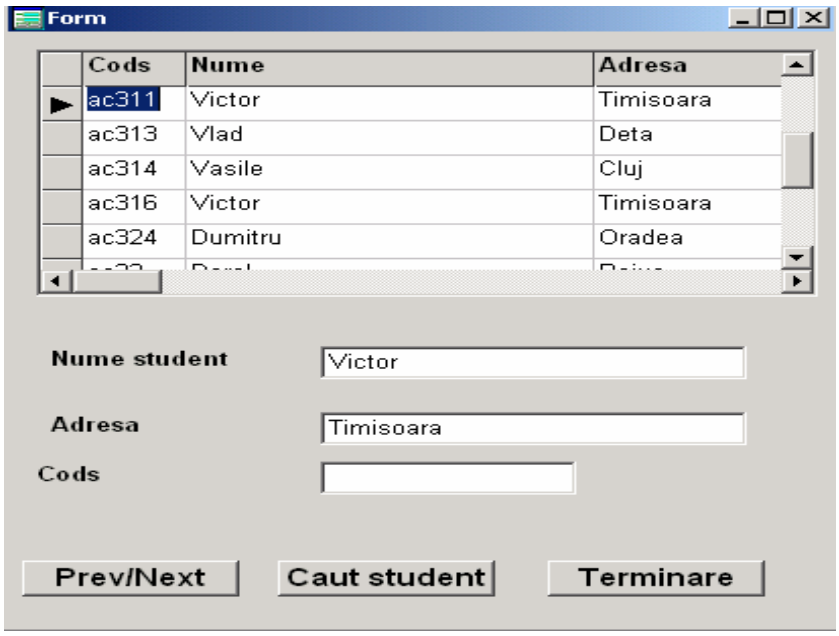
Asemănător se scrie o procedură pentru modificarea câmpului Adresa dacă se modifică pe ecran valoarea obiectului EntryField Adresa:

Procedure ENTRYFIELD2_OnLeftMouseDown
Replace Adresa whith form.entryfield2.Value
Return

Proiectare Form pentru gestiune studenti cu Designer

Se va proiecta Formul după modelul de mai jos care conține obiectele obținute cu **paleta de obiecte** şi **paleta de proprietăți** – Inspector :

- Browse1 pentru afişare studenți din tabela Student
- 3 obiecte text1, text2,text3 pentru nume student, adresa si cods
- 3 obiecte EntryField 1,2,3 pentru a afişa valoarea câmpurilor corespunzătoare din fişier. Ele vor avea proprietatea Value blank
- 3 butoane 1,2,3 pentru funcțiile dorite



Obiectele se iau de pe paleta de obiecte.

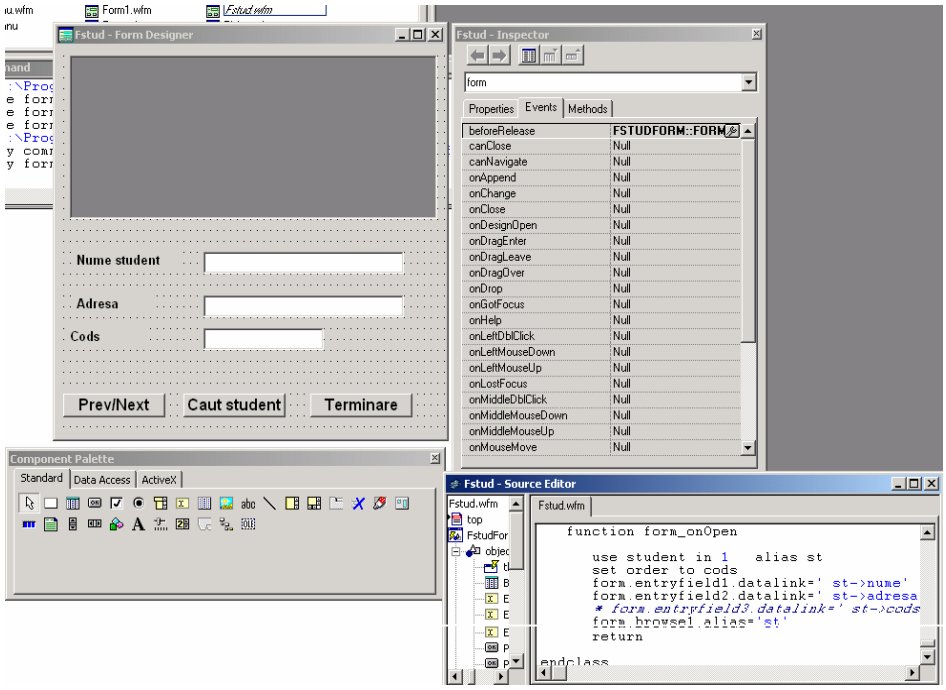
Proprietățile obiectelor şi comportamentul lor se va stabili selectând obiectul şi modificând proprietățile din inspector.

Procedurile asociate obiectelor se dau pentru evenimentele asociate: OnClick, OnLeftMouseDown, On RightMouse Down, OnOpen pentru Form.

În momentul lansării programului pe evenimentul OnOpen din Form atașăm procedura care realizează:

- deschidere fişierul care are indexi de tipul MDX după Cods şi Nume
- stabilim indexul master Cods
- Asociem obiectelor EntryField câmpurile Nume şi adresă din fisier
- EntryFieldul pentru Cods este lăsat alb pentru a introduce codul studentului căutat

Fereastra de Design este prezentată mai jos.



Dăm mai jos procedurile asociate evenimentelor si obiectelor.

function form_onOpen

use student in 1 alias st //deschidere fisier si indexi MDX asociati
set order to cods // setare index master
form.entryfield1.datalink=' st->nume' // setare campuri asociate cu entryFied-uri
form.entryfield2.datalink=' st->adresa'
* form.entryfield3.datalink=' st->cods'
form.browse1.alias='st' // setare fisier asociat cu obiectul Browse
return

```
function PUSHBUTTON1_onLeftMouseDown(flags, col, row)
     skip -1                // pentru a trece la inregistrarea uprecedenta
     return

  function PUSHBUTTON1_onRightMouseDown(flags, col, row)
     skip   1      // pentru a trece la inregistrarea urmatoare
     return

  function PUSHBUTTON2_onClick
    // procedura de cautare student folosind valoarea Cods introdusă in
EntryFied
Vcods= form.entryfield3.value
    seek vcods
     if eof()
    msgbox(' Studentul'+ vcods+'nu exista','Cods gresit',0)
    disp
     endif
     return

  function PUSHBUTTON3_onClick
     close database              // procedura asociata butonului Terminare
     form.close()                // inchidere form
     return

  function PUSHBUTTON3_onClose
  close database         // pentru inchidere form din coltul din dreapta sus
     return
```

Se poate adăuga un buton afişare care asociat evenimentelor Left si Right Mouse să activeze programul afişare note discutat in cursul trecut.

Programul principal generat ca o clasa este dat mai jos.

```
** END HEADER -- do not remove this line
//
// Generated on 31/03/2013
//
parameter bModal
```

```
local f
f = new FstudForm()
if (bModal)
  f.mdi = false // ensure not MDI
  f.readModal()
else
  f.open()
endif
 *  Definire clase
class FstudForm of FORM
  with (this)
    onOpen = class::FORM_ONOPEN
    height = 18.0
    left = 71.0
    top = 0.0
    width = 58.1429
    text = ""
  endwith

  this.PUSHBUTTON1 = new PUSHBUTTON(this)
  with (this.PUSHBUTTON1)
    onLeftMouseDown =
class::PUSHBUTTON1_ONLEFTMOUSEDOWN
    onRightMouseDown =
class::PUSHBUTTON1_ONRIGHTMOUSEDOWN
    height = 1.0909
    left = 1.0
    top = 16.0
    width = 15.2857
    text = "Prev/Next"
    fontSize = 12.0
    fontBold = true
  endwith

  this.PUSHBUTTON2 = new PUSHBUTTON(this)
  with (this.PUSHBUTTON2)
    onClick = class::PUSHBUTTON2_ONCLICK
    height = 1.0909
    left = 19.0
    top = 16.0
```

```
  width = 15.2857
  text = "Caut student"
  fontSize = 12.0
  fontBold = true
endwith

this.PUSHBUTTON3 = new PUSHBUTTON(this)
with (this.PUSHBUTTON3)
  onClick = class::PUSHBUTTON3_ONCLICK
  onClose = class::PUSHBUTTON3_ONCLOSE
  height = 1.0909
  left = 38.0
  top = 16.0
  width = 15.2857
  text = "Terminare"
  fontSize = 12.0
  fontBold = true
endwith

this.BROWSE1 = new BROWSE(this)
with (this.BROWSE1)
  height = 7.5
  left = 2.0
  top = 0.5
  width = 55.0
endwith

this.TEXT1 = new TEXT(this)
with (this.TEXT1)
  height = 1.0
  left = 3.0
  top = 9.5
  width = 16.0
  fontBold = true
  text = "Nume student"
endwith

this.TEXT2 = new TEXT(this)
with (this.TEXT2)
  height = 1.0
```

```
  left = 3.0
  top = 11.5
  width = 12.0
  fontBold = true
  text = "Adresa"
endwith

this.ENTRYFIELD1 = new ENTRYFIELD(this)
with (this.ENTRYFIELD1)
  height = 1.0
  left = 22.0
  top = 9.5
  width = 30.0
  value = " "
endwith

this.ENTRYFIELD2 = new ENTRYFIELD(this)
with (this.ENTRYFIELD2)
  height = 1.0
  left = 22.0
  top = 11.5
  width = 30.0
  value = " "
endwith

this.TEXT3 = new TEXT(this)
with (this.TEXT3)
  height = 1.0
  left = 2.0
  top = 13.0
  width = 12.0
  fontBold = true
  text = "Cods"
endwith
```

7. PROIECTAREA UNOR APLICAȚII COMPLEXE

7.1. Defiirea prin program a obiectelor grafice standard

Folosind designer-ul se pot crea Form-uri care să conțină mai multe obiecte de tipuri diferite, care se utilizează în aplicație pentru a asigura o interfață flexibilă cu utilizatorul, dar se pot proiecta numai aplicații simple. Proiectarea clasică permite utilizarea **unui singur fişier** care apare ca parametru al obiectului Form pentru proprietatea View (this.View = "stud.dbf"). Deschiderea indexată a fişierului, sau deschiderea altor fişiere prin procedurile programului duce la anomalii. Nu se acceptă alte câmpuri decât cele din fişierul asociat Form-ului, în proprietățile DataLink din obiectele Entryfield şi SpinBox, DataSource din ListBox şi ComboBox, Fields din Browse.

Designer-ul crează o singură clasă Form cu obiectele şi proprietățile încapsulate şi fixe, fără posibilitatea de a avea acces la obiecte din exterior. Prin crearea unui obiect Form din clasa creată se copiază proprietățile implicite ale clasei. Punerea în funcțiune a programului echivalează cu crearea unui obiect din clasa definită şi deschiderea lui.

Definirea obiectelor dintr-o clasă existentă se poate face direct in program şi proprietățile obiectelor pot fi modificate dinamic. Obiectelor definite li se pot asocia evenimente care declanşează execuția unor proceduri. În aceste proceduri pot fi deschise mai multe tabele (fişiere).

Obiectele de interfață grafică standard existente în Windows pot fi create în programe sau proceduri folosind comanda DEFINE.

Orice obiect grafic dintr-o clasa Windows se defineşte în cadrul unui container care este de regula o fereastra (FORM). Comenzile pentru definire, afişare, modificare se pot da in linia de comanda, dar in general in program.

Proprietățile obiectelor standard sunt:

- Proprietăți structurale (dimensiune, nume, adresa, culoare, ..)
- Metode – funcții care acționează asupra obiectelor clasei (Open(), Close(),Release()).
- Proprietăți comportamentale definite prin subrutine asociate unor evenimente.

Se recomanda folosirea documentației disponibile in HELP la comanda DEFINE, fiindcă sunt multe proprietăți comune sau specifice fiecărui obiect.

Definirea unui obiect dintr-o clasa existentă se face cu comanda DEFINE care are sintaxa:

DEFINE <nume_clasa> <nume_obiect**> [OF <**obiect_container>]
[FROM <linie, coloana> **TO** < linie,coloana > | **<AT** <linie, coloana>**]**
[PROPERTY <lista_proprietati>**]**
[CUSTOM <custom property list>**]**

Unde **<nume_clasa>** este clasa obiectului care se doreşte a fi definit.
FROM sau **AT** se utilizează pentru a preciza poziția obiectului în fereastră.
Lista de proprietăți conține valorile proprietăților specifice obiectului.
OF obiect container - precizează formul pe care se va plasa obiectul definit
Există proprietăți comune pentru obiecte şi specifice unei clase.
Obiectele standard Windows principale sunt:

Browse	**Checkbox**	**Combobox**
DDELink	DDETopic	**Editor**
Entryfield	**Form**	**Image**
Line	**Listbox**	**Menu**
Menubar	Object	Ole
Paintbox	**Popup**	**Pushbutton**
Radiobutton	Rectangle	Scrollbar
Shape	**Spinbox**	Tabbox
Text		

Definirea unui form

La definirea unui Form, care este o fereastră Windows, trebuie precizată poziția pe ecran, înălțimea, lățimea, titlul, font-ul titlului, culoarea fondului, proceduri atașate la evenimente şi alte proprietăți. Dacă nu se specifică nimic la definirea unui obiect dintr-o clasă se iau toate proprietățile implicite.
Pentru exemplele care urmează se recomandă utilizarea lor din fereastra de comandă, pentru a observa că efectul comenzilor este imediat vizibil (schimbare dimensiuni, culori, creare de obiecte).

//Se deschide unul sau mai multe fisiere care se vor utiliza

DEFINE FORM f1 PROPERTY TEXT 'Titlul ferestrei', height 20,width 80

// Creează obiectul Form f1 in memorie
F1.OPEN() // afişează fereastra f1 deja definita
F1.CLOSE() // închide fereastra
F1.RELEASE() // şterge obiectul din memorie

Modificarea parametrilor unui obiect existent

F1.TEXT='Noul titlu al ferestrei' //atributul titlului (textul afişat)
F1.HEIGHT=10 //atributul înălţime
F1.WIDTH=10 //atributul lăţime
F1.LEFT=5 // poziţie fata de stânga ecran
F1.TOP=2 //număr linie fata de latura de sus a ecranului
F1.VISIBLE=.F. //afişează sau nu obiectul
F1.ESCEXIT=.T. //se poate închide fereastra cu ESC

Tratarea evenimentelor

F1.OnOpen= {; Use stud in 1 index inume alias St}
//Se deschide unul sau mai multe fisiere care se vor utiliza in aplicatie la deschiderea formului f1
F1.ONMOVE={;?' Se mişcă fereastra!'} // la miscarea ferestrei f1 se afisează textul
F1.OnLeftMouseDown={; Append} // la click stanga pe f1 se adauga o inregistrare

Afişarea atributelor obiectelor se poate face ca pentru expresii

? F1.TEXT, F1.WIDTH, F1.TOP // returnează valoarea atributului indicat

Asocierea unui fişier la o fereastra

F1.VIEW='STUDENTI.DBF'
// Asociază un fişier ferestrei care va fi deschis odată cu fereastra; se recomandă OnOpen()
INSPECT(F1) // Afişează lista de proprietăţi pentru obiectul F1

7.2. Crearea de obiecte pe Form

Pe Form-ul F1 creat se pot defini alte obiecte de interfață standard Windows. Orice obiect de interfață trebuie definit obligatoriu pe un Form, care este considerat container (părinte).

Definire obiecte TEXT

Obiectele Text înlocuiesc afişările normale cu Say din DOS. Obiectele Text permit o scriere sofisticată, care foloseşte toate facilitățile sistemului Windows. Textul va fi considerat ca un Banner pentru care se dă lungime, lațime, culoare, poziție pe form (left,top), tip, dimensiune şi culori de fonturi, precizate prin proprietăți.

Prin clauza OF se indică formul pe care se va plasa textul. Se pot defini obiecte cu acelaş nume pe form-uri diferite.

DEFINE TEXT T1 **OF** F1 **PROPERTY** TEXT **'Nume Student: '** // definire text T1

Putem modifica proprietățile obiectului precizat prin nume_parinte.nume_obiect(F1.T1)

```
F1.T1.WIDTH=30              // lățime text
F1.T1.HEIGHT=2              // înălțime text
F1.T1.BORDER=.T.            // bordura in jurul textului
F1.T1.FONTNAME="ARIAL"      // Nume font folosit
F1.T1.FONTSIZE=14           // dimensiune font
F1.T1.FONTBOLD=.T.          // utilizare fond Bold
F1.T1.ColorNormal='GR/GR+'  // culoare verde/galben
F1.T1.left =5          // 5 pozitii fata de  marginea stanga  a ferestrei
F1T1.top=3             // 3 pozitii fata de  marginea de sus  a ferestrei
```

Definire obiecte Button

Obiectele Button sunt funcțional texte scrise pe un suport grafic de formă tri-dimensională şi au toate proprietățile obiectelor Text. Ele se folosesc mai frecvent pentru comenzi fiindcă sugerează butoanele unui tablou de comandă. Trebuie să precizăm că **pentru orice obiect** (Text, EntryField,Image..) **se pot asocia proceduri** ataşate unor evenimente.

Define PushButton B1 **At 8,5 OF** F1 **Prop Text** 'Adaugare' // butonul B1 cu textul adaugare
F1.B1.OnLeftMouseDown={;Append} // la apasarea butonului se executa Append

Definire obiecte ENTRYFIELD

Obiectele EntryField înlocuiesc Get-urile din varianta DOS şi permit o scriere cu fonturi a valorilor afişate sau introduse de utilizator. Are proprietăți similare ca şi obiectele Text, dar valoarea se poate modifica şi se poate asocia unui câmp din fişierul curent deschis.

DEFINE ENTRYFIELD E1 OF F1
F1.E1.**DATALINK='ST->NUME'** // camp asociat din fisierul Stud deschis anterior cu alias ST
F1.E1.WIDTH=20
F1.E1.TOP =5
DEFINE ENTRYFIELD E2 OF F1
F1.E2.DATALINK='ST->ADRESA' // camp asociat Adresa din fisierul Stud deschis anterior F1.E1.TOP =6

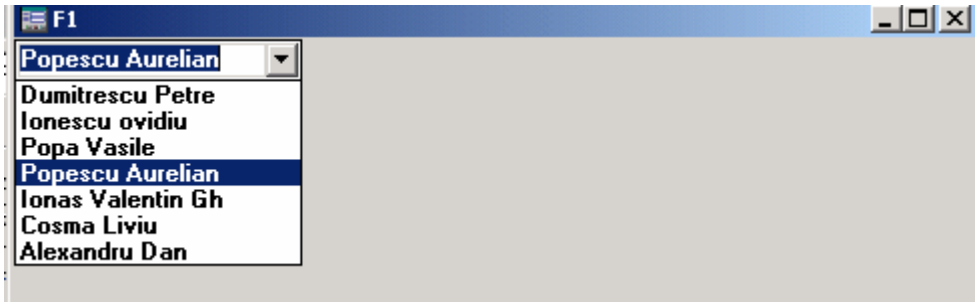
In proprietatea **DataLink** se poate da un **nume de câmp sau un nume de variabilă**, a cărei valoare se afişează şi poate fi modificată. La modificarea valorii unui EntryField nu se **modifică valoarea câmpului** afişat. Modificarea câmpului se poate face folosind comanda Replace într-o procedură asociată evenimentului **On Change** asociat EntryField-ului.

Definire obiecte Combobox

Define COMBOBOX CB1 of F1 Prop Datasource 'Field STUD->Nume'
F1.CB1.Datasource ='Field Stud->Adresa' // schimbă câmpul afişat
? F1.CB1.value // afişează valoarea curentă a obiectului CB1

La acționarea unui obiect Combobox se vor afişa într-o listă valorile câmpului asociat şi va putea fi selectată una din valori, care va deveni valoarea curentă a obiectului şi va putea fi apelată prin atributul ComBobox.Value (vezi selecția tipului şi dimensiunii de font în Word). Din acest punct de vedere este asemănător cu EntryField, mai ales că valoarea curentă poate fi dată prin tastare directă.

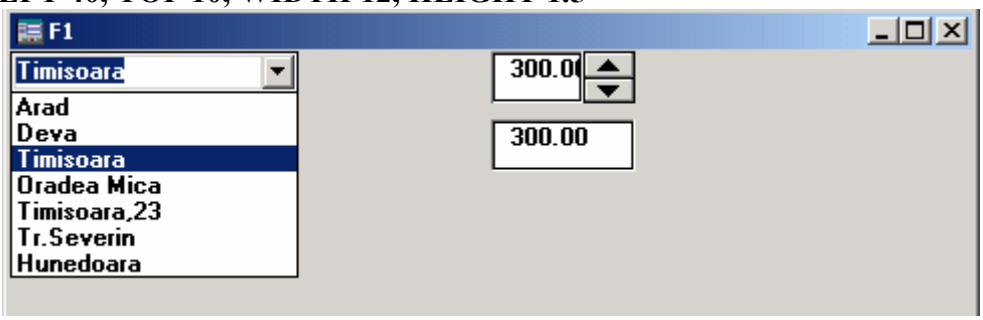
Prin Combobox se selectează o valoare şi **nu se face poziţionarea pe înregistrarea** care conţine valoarea selectată. Este utilă la selectarea numelui complet al unei instituţii. Poziţionarea pe înregistrarea selectată prin valoarea unui câmp se face cu ListBox.



Definire obiecte SpinBox

Obiectele SpinBox afişează valoarea curentă a unui câmp sau variabilă numerică, care poate fi modificată direct, sau prin butonul de incrementare / decrementare asociat. In rest se comportă ca un EntryField după cum se vede în figură.

DEFINE SpinBox SB1 OF F1 Property DataLink ” Stud->BURSA”,; LEFT 40, TOP 10, WIDTH 12, HEIGHT 1.5



Definire obiecte Listbox

Un obiect ListBox este:

- listă de valori ale unui **singur câmp** din fişierul curent deschis dacă se foloseşte în proprietatea DataSource argumentul **FIELD**,
- elementele unui tablou linear dacă argumentul este **ARRAY**.

Nu se recomandă să fie folosit ListBox pentru afişarea unor rezultate.
ListBox este folosit la selectarea unei înregistrări pe baza valorii unui **câmp.**

DEFINE LISTBOX LB1 OF F1
F1.LB1.DATASOURCE= "FIELD STUD->NUME"
F1.LB1.TOP=3
Obiectele din form se sincronizează. La selecţia unei înregistrări din Listbox valoarea proprietăţii obiectelor EntryField se va modifica corespunzător înregistrării pe care s-a poziţionat.

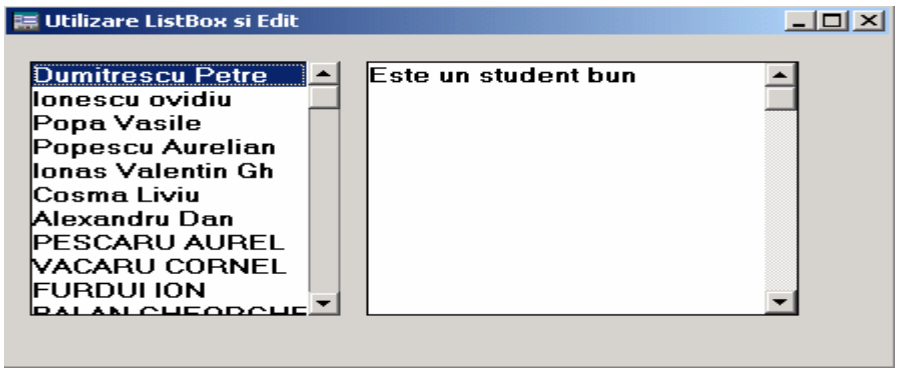
Definire obiecte Editor

Obiectele EDITOR sunt ferestre de editare pentru câmpuri memo. Proprietăţile unui obiect Editor sunt asemănătoare cu cele ale unui Form pentru dimensiuni şi poziţie. Editorul este ataşat unui câmp memo printr-un DataLink ca şi un EntryField. In interiorul ferestrei Editor se pot edita texte folosind posibilităţile editorului NotePad.

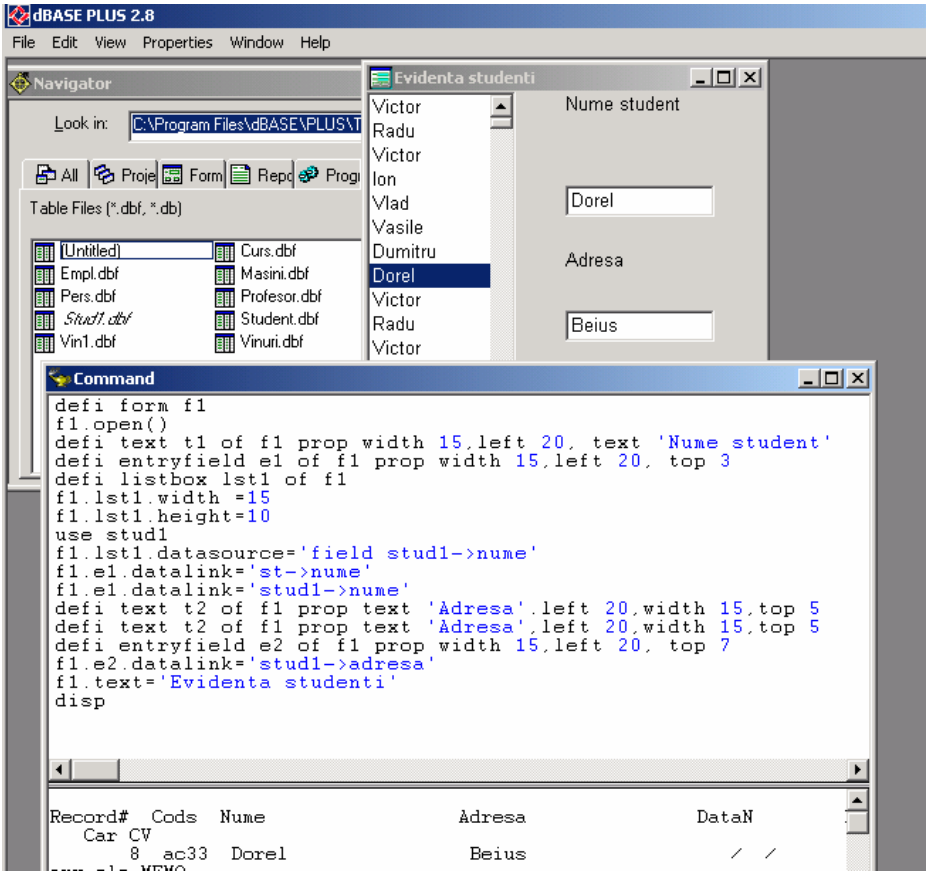
DEFINE EDITOR Ed1 OF F1
F1.Ed1.DATALINK='Stud->CV'
F1.Ed1.Height=20
F1.Ed1.Left=50

Realizăm un Form cu Designer-ul, pe care plasăm un ListBox şi un Editor.
In **OnOpen** vom deschide fişierul Stud, se va ataşa la ListBox câmpul Nume şi la Editor câmpul memoCV.

Procedure Form_OnOpen1
use stud
Form.Listbox1.DataSource='Field Stud->Nume'
Form.Editor1.DataLink='Stud->CV'
return



Exemplu de program scris direct in linia de comanda folosind text, entryfield, listbox:



Definire obiecte Browse

Obiectul Browse se foloseşte pentru afişarea sub formă de tabel a valorii unor câmpuri dintr-un fişier, specificate în atributul Fields. Intr-o aplicaţie în care se selectează un număr de înregistrări printr-o procedură, se recomandă ca înregistrările să fie memorate într-un fişier intermediar de rezultate care să se afişeze prin declararea unui Browse.

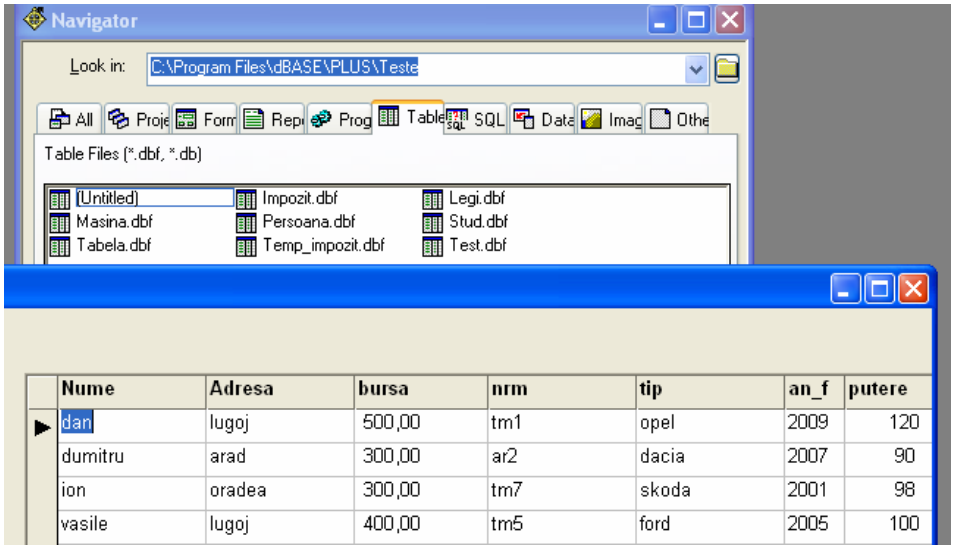
Define BROWSE BR1 of F1 Prop Text 'Evidenta Studenti ' ,Height 20,; Width 40, Left 15,Top 2, Fields ' Nume, Adresa, Bursa'

Intr-un Browse se poate specifica o listă de câmpuri din mai multe

fişiere deschise simultan în zone de lucru diferite, care sunt legate între ele prin Set Relation. Dacă se consideră fişierul de studenţi Stud şi Maşina cu aliasul MS putem defini un Browse care să afişeze atât datele personale cât şi ale maşinii.

```
* Programul Prel.wfm
Set Talk off
Clear all              // sterge toate obiectele si inchide fisierele
Use stud in 1  Index Inume alias st
Use masina in 2 Index INrm alias MS
select 1
Set Relation to NRM Into MS

Defi form f1 prop width 100
F1.Open()
Define BROWSE BR1 of  F1 Prop Height 20,;
Width 110, Left 15,Top 2

 f1.br1.alias ='st'       //precizare fisier asociat browse Br1
    f1.br1.Fields =' st->Nume, st->Adresa, st->Bursa,;      // lista campuri de
afisat
 MS->NRM, MS->TIP, MS->AN_F,MS->putere'

Wait              //asteptare apasare tasta  sau click pe Form pentru terminare
program
f1.Close()    //inchide fereastra
close all      //Inchide fisierele
          Return
```

De remarcat că aşteptarea pentru Wait se termină la apăsarea unei taste sau click pe Form.



Proprietati comune pentru obiecte windows:

Property	Event	Method

before borderStyle enabled fontBold fontItalic fontName fontSize fontStrikeout fontUnderline form height ColorNormal 'r/b ' ColorHighlight helpFile helpId hWnd movable sizeable	id left mousePointer name pageno parent printable statusMessage tabStop text top visible value valid conditie ValidErrorMsg	OnLostFocus OnLeftMouseDown onLeftDblClick onRightMouseDown onLeftMouseUp onMouseMove onOpen onRender onRightDblClick onRightMouseDown onRightMouseUp when move() onGotFocus onLostFocus OnChange	release() setFocus() Open() Close() Release() Copy() Paste() Cut() Undo() Print()

7.3. Crearea unor aplicații complexe folosind Designer-ul

Dacă se foloseşte în exclusivitate **Designer**-ul se pot realiza doar **aplicații simple cu un singur fişier** deschis specificat în atributul View din Form. Scrierea manuală a programelor prin definirea directă a tuturor obiectelor de interfață grafică este laborioasă.
La aplicații complexe se poate combina utilizarea Designer-ului cu definirea şi manipularea directă dinamică a obiectelor în proceduri. Se evită astfel operațiile de rutină şi se acționează direct asupra obiectelor pentru a realiza operații mai complicate.

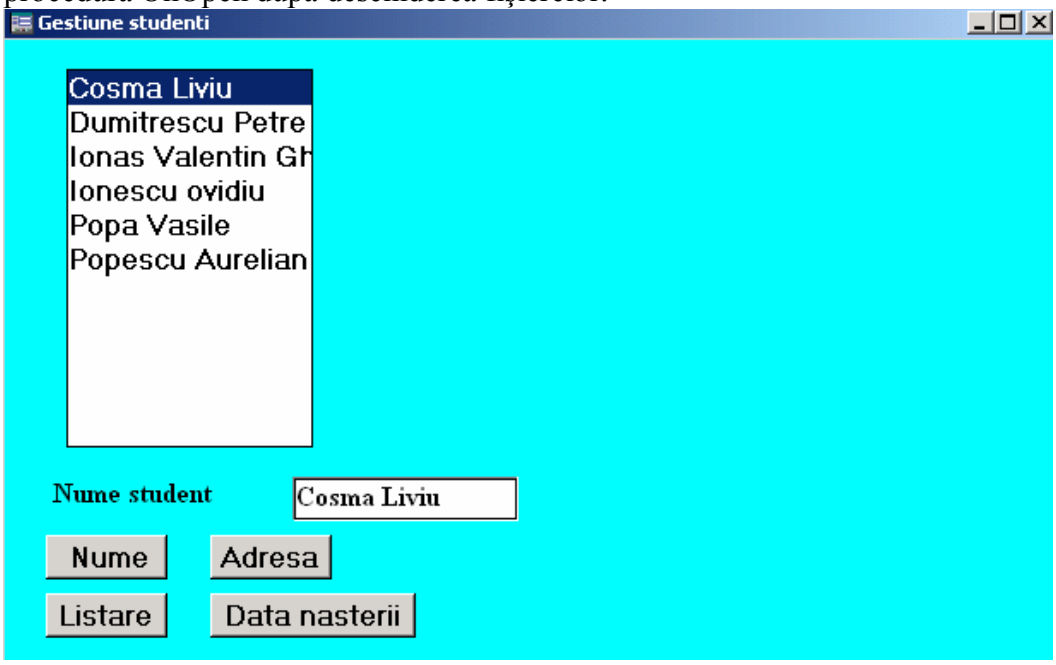
In continuare se va prezenta o metodă prin care se pot utiliza dinamic mai multe fişiere folosind Designer-ul, dar fără a folosi proprietatea View din Form.

- Se va genera un Form care să conțină mai multe obiecte
- In procedura asociată evenimentului OnOpen se vor deschide fişierele de date şi index, se vor face setări (Set Talk off, Clear, Set Date to DMY, Set Decimal To) şi se vor defini relațiile dintre fişiere (Set Relation)
- Obiectelor li se vor ataşa proceduri prin evenimente.
- In cadrul procedurilor se vor ataşa proprietăți de legătură cu fişierele pentru obiectele de pe Form (DataLink din obiectele Entryfield şi SpinBox, DataSource din ListBox şi ComboBox, Fields din Browse)
- In proceduri se vor defini şi şterge obiecte de tip ListBox şi Browse
- In proceduri se pot chema alte programe care se găsesc pe alte Form-uri

Se va exemplifica în continuare cu o aplicație, în care pe un Form la care nu s-a asociat nici un fişier se pun cu Designer-ul Texte, Butoane, Entryfield şi ListBox pentru care nu se precizează atributele DataLink şi respectiv DataSource. La deschiderea Form-ului pe OnOpen se deschide fişierul STUD cu indexul Inume, încât câmpurile acestuia vor fi recunoscute în continuare şi se poate consulta în acces direct. Închiderea fişierului se va face pe evenimentul OnClose la închiderea Form-ului. Atributele DataLink şi Data Source se vor adăuga prin procedurile asociate obiectelor pe evenimentul Click stânga din EntryField şi respectiv ListBox. La deschiderea Form-ului Entryfield-ul şi ListBoxul rămân albe şi se vor completa dacă dăm click pe ele. In acest fel nu trebuie intervenit în codul generat de Designer ci numai în procedurile asociate evenimentelor ataşate obiectelor.

```
Procedure ENTRYFIELD1_OnLeftMouseDown(flags, col, row)
  form.entryfield1.datalink='stud->nume'
  return

  Procedure LISTBOX1_OnLeftMouseDown(flags, col, row)
  form.listbox1.datasource='field stud->nume'
  return
```

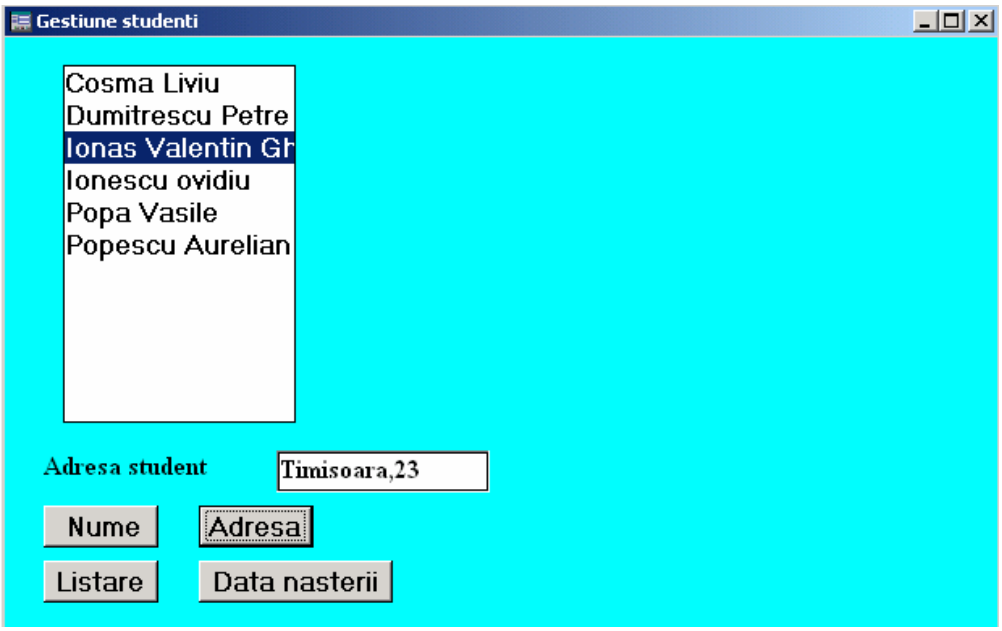
Specificarea argumentelor Datalink şi DataSource se poate face şi în procedura OnOpen după deschiderea fişierelor.



Prin butoanele **Nume, Adresa, Data naşterii** s-au asociat proceduri care schimbă câmpul asociat obiectului Entryfield prin DataLink. La selecţia unui student din ListBox se va afişa în EntryField câmpul Nume, Adresa sau Data nasterii. Pentru a claritate se va schimba şi textul afişat în dreptul EntryField-ului.

```
Procedure PUSHBUTTON1_OnLeftMouseDown(flags, col, row)
  Form.text1.text='Nume student'
  Form.Entryfield1.datalink= ' stud->Nume'
  Return
```

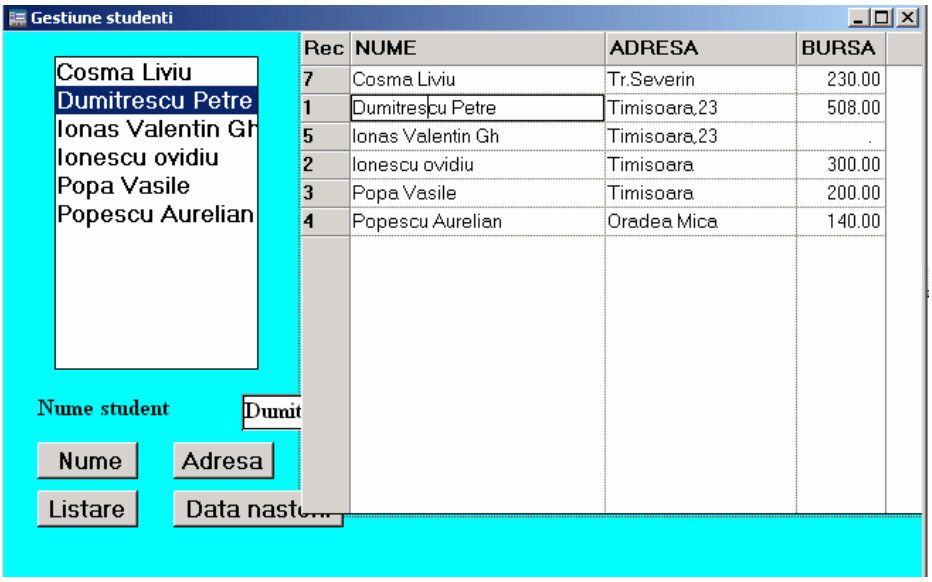
```
Procedure PUSHBUTTON2_OnLeftMouseDown(flags, col, row)
  Form.text1.text='Adresa student'
  Form.Entryfield1.datalink= ' stud->Adresa'
  Return

  Procedure PUSHBUTTON3_OnLeftMouseDown(flags, col, row)
  Form.text1.text='Data nasterii'
  Form.Entryfield1.datalink= ' stud->Data_n'
  Return
```



Prin butonul **Listare** se asociază o procedură care defineşte un Browse pe Form şi se afişează câmpurile Nume, Adresa, Data_n din fişierul Stud. Dând click dreapta pe acelaşi buton se şterge obiectul Browse. Obiectele create în afara clasei trebuie şterse înainte de relansarea programului sau cu Clear All în procedura asociată la OnOpen. Dacă nu sunt şterse, la următoarea rulare se va încerca crearea unui nou obiect cu acelaşi nume şi va apare eroare.

```
  Procedure PUSHBUTTON4_OnLeftMouseDown(flags, col, row)
    // Buton List
Form.whidth =120
  Defi browse br1 of form property Fields 'nume, adresa, bursa.data_n',;
  height 20, width 80,left 35, top 0,FontSize 11
  * Form.br1.OnrightMouseDown =form.br1.release()
```

```
Return

Procedure PUSHBUTTON4_OnRightMouseDown(flags, col, row)
Form.br1.release()
return
```



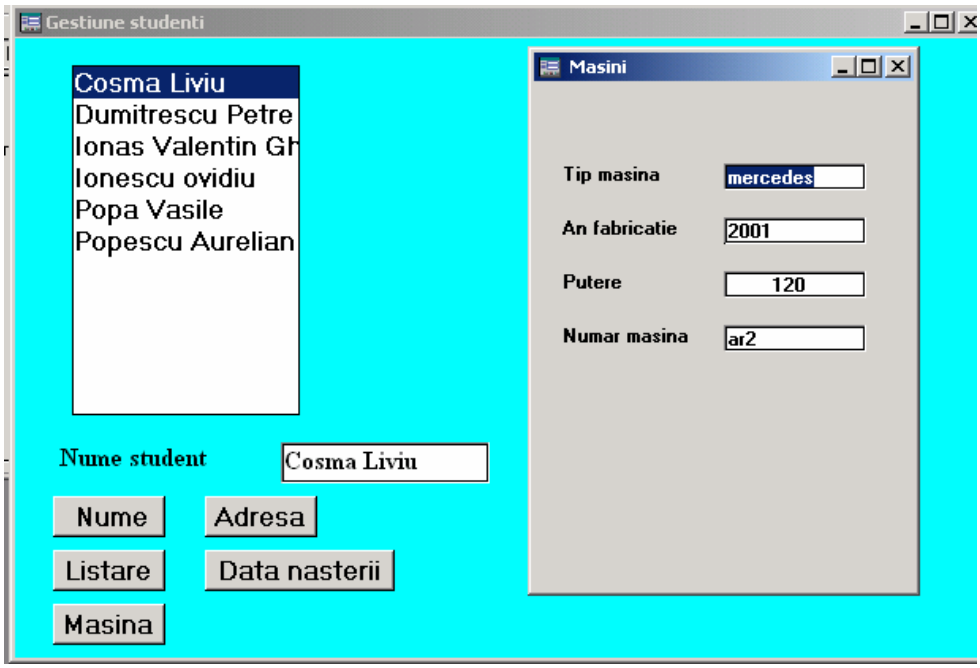
Se observă că toate obiectele s-au poziţionat pe acelaşi student indiferent de modul în care s-a făcut poziţionarea (prin ListBox, Browse, Go, Seek, Locate).

7.4. Definirea şi deschiderea unei noi ferestre

Dintr-o procedură asociată unui obiect dintr-un Form, se va defini şi deschide un nou Form F2, care conţine obiecte Text şi EntryField în care să se afişeze datele despre maşina studentului selectat.

Vom defini un nou buton Masina la care vom ataşa procedura, care defineşte o nouă fereastră F2 pe care vom afişa datele maşinii. Maşina va fi identificată printr-o legătură de Set Relation între fişierul Stud şi Masini în procedura ataşată la OnOpen unde se deschid şi cele 2 fişiere indexate. Dacă dorim ca datele maşinii să fie permanent afişate vom ataşa la OnOpen din Form-ul principal procedura următoare, care defineşte şi Formul F2 ce conţine obiecte Text şi EntryField referitoare la maşina studentului curent selectat.

```
Procedure Form_OnOpen
   Use Stud in 1 Index Inume
   Use Masini in 2 Index INrm alias MS
   Set Relation to NRM Into MS
   Defi Form F2 Prop width 40, Text " Masini",left 50
   F2.OnClose ={;f2.release()}
   Defi TEXT Tip of F2 Prop Text 'Tip masina ',left 3, top 3,width 15
   Defi TEXT AnFabr of F2 Prop Text 'An fabricatie ',left 3, top 5,width 15
   Defi TEXT Putere of F2 Prop Text 'Putere ',left 3, top 7,width 15
   Defi TEXT Numar of F2 Prop Text 'Numar masina ',left 3, top 9,width 15
Defi EntryField Etip of F2 Prop left 20, top 3, width 15, DataLink 'MS->TIP'
Defi EntryField EAnf of F2 Prop left 20, top 5, width 15, DataLink 'MS->An_F'
 Defi EntryField Etip1 of F2 Prop left 20, top 7, width 15, DataLink 'MS->Putere'
Defi EntryField Enrm of F2 Prop left 20, top 9, width 15, DataLink 'MS->Nrm'
   F2.Open()
   Return
```



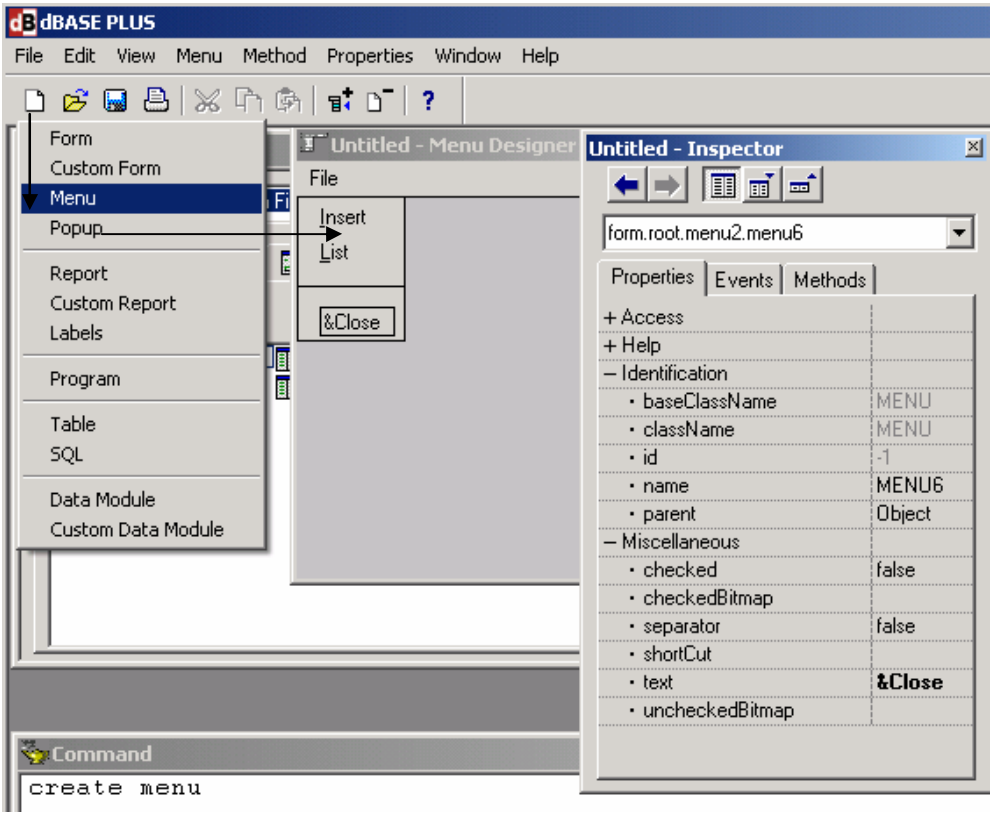
Folosind această metodă de lucru se poate realiza proiectarea grafică a Form-urilor, care este o muncă de uzură, folosind Designer-ul, iar în procedurile asociate evenimentelor pentru diferite obiecte se vor preciza elementele de legătură cu fişierele bazei de date.

8. MENIURI si OBIECTE MULTIMEDIA

8.1. Meniuri Windows

Meniurile Windows îmbină conceptul de menu bară şi popup cu caracteristicile programării pe obiecte utilizată în interfeţele grafice.
La proiectarea unui meniu vom folosi Designer-ul de meniu care se activează din:

- Fereastra Command **CREATE MENU**;
- Navigator grupa Forms şi Untitled Menu-Designer
- Bara de meniu principal File/New/Menu
- Speed Bar şi New/Menu



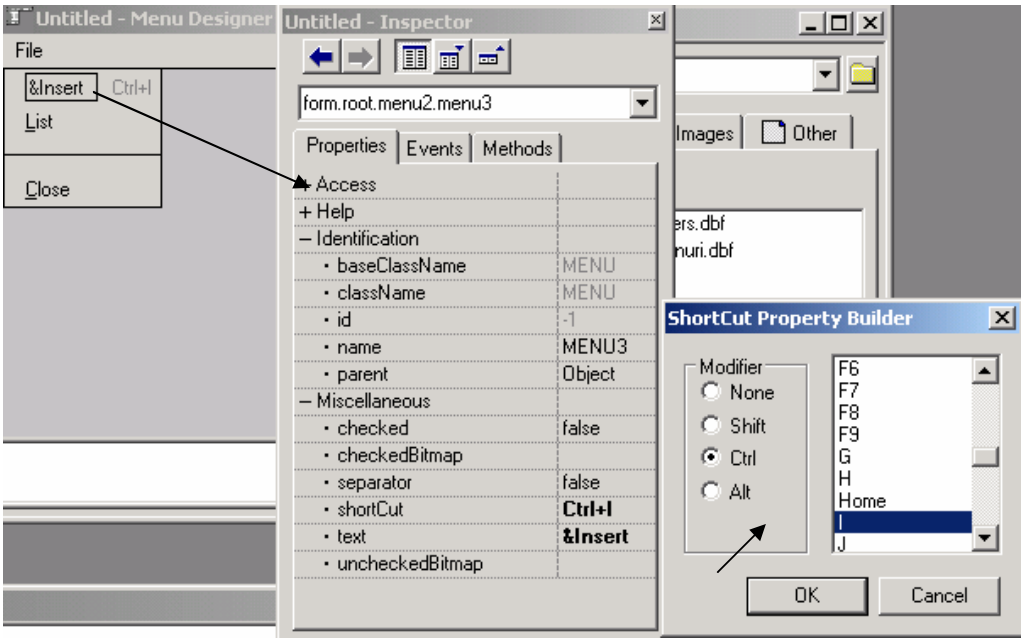
Se va deschide o fereastră de proiectare **Untitled Menu-Designer** şi

paleta de proprietăți (Inspector). Se pot plasa mai multe meniuri fiecare având un nume şi mai multe bare de comanda. Fiecărei bare de comandă i se va ataşa printr-un eveniment o procedură.

Se vor prezenta printr-un exemplu fazele de definire a meniurilor:

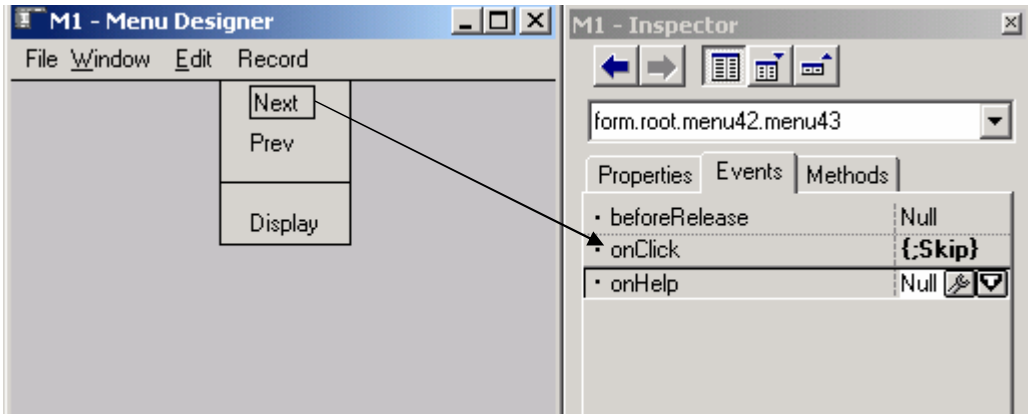
1. Apare in fereastră primul meniu şi se scrie **FILE** – numele meniului
2. Se tastează **săgeată verticală** şi apare prima bara unde scriem **&Insert**, apoi următoarele bare unde scriem **&List şi &Close.** Semnul & precizează caracterul de selecție rapidă şi va fi subliniat în meniul afişat **Insert.**
3. Fiecărui element de meniu (bară) i se poate asocia un **ShortCut** de tipul CTR+I..

Barele unui meniu pot fi grupate printr-o bară separator, dacă bara respectivă are proprietatea **Separator = .T**.



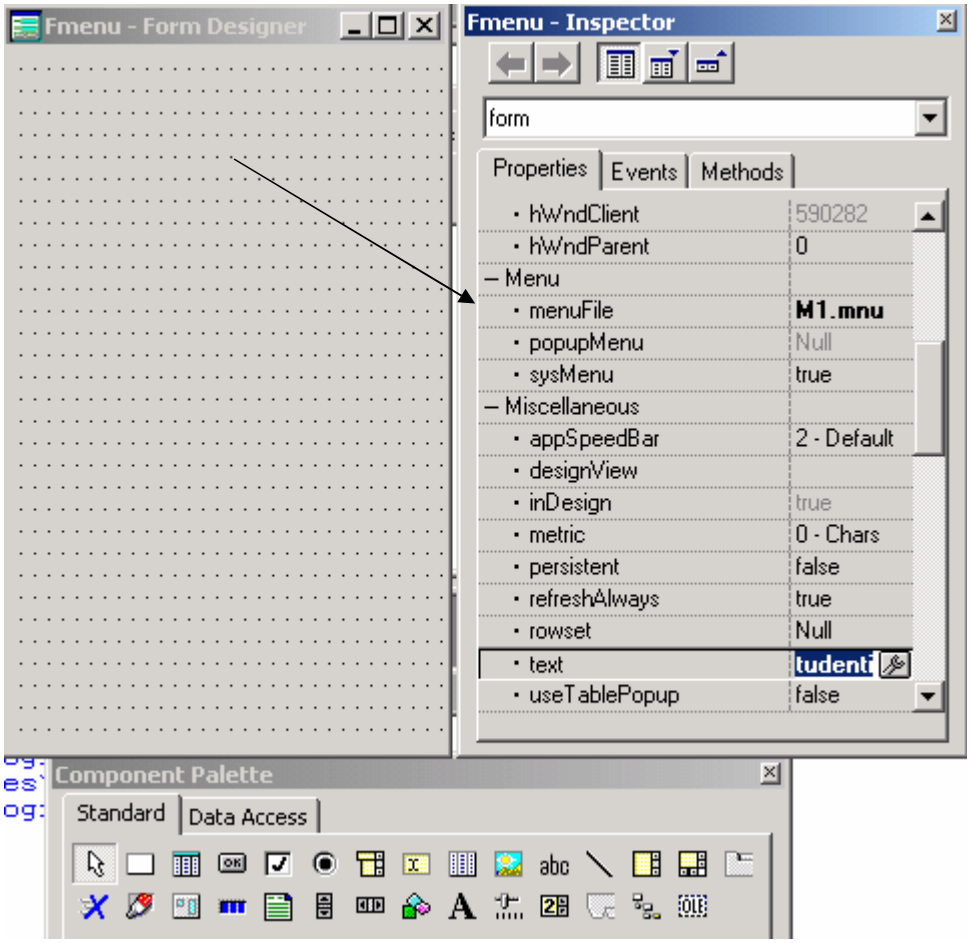
4. Se dă click pe File şi **Tab** pentru a **trece la următorul meniu**. Se selectează din bara de meniu **Menu** şi Insert Edit şi se vor insera automat comenzile de editare Windows **Undo, Cut, Copy, Paste** cu numele de meniu **Edit**.
5. Se dă Tab (nou meniu) şi se selectează din **Menu** Insert Windows, care ne va permite în timpul rulării afişarea ferestrelor deschise pentru a le selecta (ca şi Windows din meniul principal dBase)

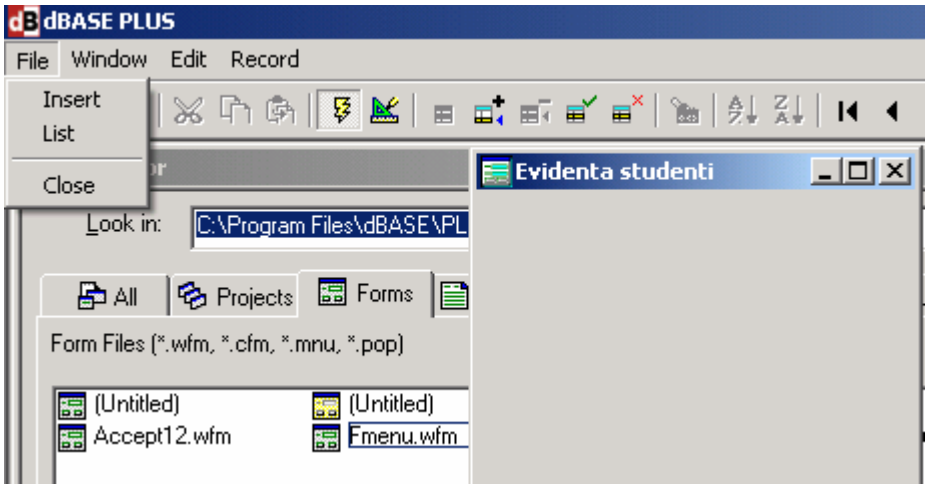
6. Se dă Tab şi se completează meniul **&Record** cu barele **&Next, &Prev şi &Disp.**
7. Se selectează pe rând câte **o bară** din meniurile definite (exceptând Edit şi Windows) şi cu paleta de proprietăţi **Events** i se va asocia câte o procedură sau comandă asociată:
8. La închiderea ferestrei Menu Designer se va da meniului proiectat un nume, care va avea ex: **M1.mnu** şi după compilare se plasează în grupul Forms.
9. **Meniul proiectat se asociază unui Form în proprietatea MenuFile din Inspector.**
10. La lansarea Form-ului pe care se plasează meniul se va afişa meniul în locul barei principale de meniu (nu pe Form).
11. Selectând un meniu se vor afişa barele componente care pot fi selectate cu click de mouse pentru a activa procedurile asociate



Se deschide cu Designer un Form cu titlul Evidenta studenţi şi numele **Fmenu** la care se ataşează prin proprietatea **MenuFile** meniul proiectat anterior **M1.mnu**.
La evenimentul **OnOpen** din Form se va asocia comanda **Use Student**, iar la **OnClose** se va asocia **Close Database**

La deschiderea Formului Fmenu.wfm se va afişa meniul care ne va permite să selectăm procedurile care execută funcţiile dorite. Se remarcă faptul că meniul nu apare pe Form ci pe bara de meniu principal.


Fmenu - Form Designer
Fmenu - Inspector
form
Properties
Events
Methods
hWndClient
590282
hWndParent
0
Menu
menuFile
M1.mnu
popupMenu
Null
sysMenu
true
Miscellaneous
appSpeedBar
2 - Default
designView
inDesign
true
metric
0 - Chars
persistent
false
refreshAlways
true
rowset
Null
text
tudent
useTablePopup
false
Component Palette
Standard
Data Access


dBASE PLUS
File
Window
Edit
Record
Insert
List
Close
Evidenta studenti
Look in:
C:\Program Files\dBASE\PL
All
Projects
Forms
Form Files (*.wfm, *.cfm, *.mnu, *.pop)
(Untitled)
(Untitled)
Accept12.wfm
Fmenu.wfm

Dacă selectăm **List** se va afişa conţinutul fişierului Student (Browse) Câmpul memo CV poate fi afişat şi modificat prin DblClick care va deschide o fereastră Editor.

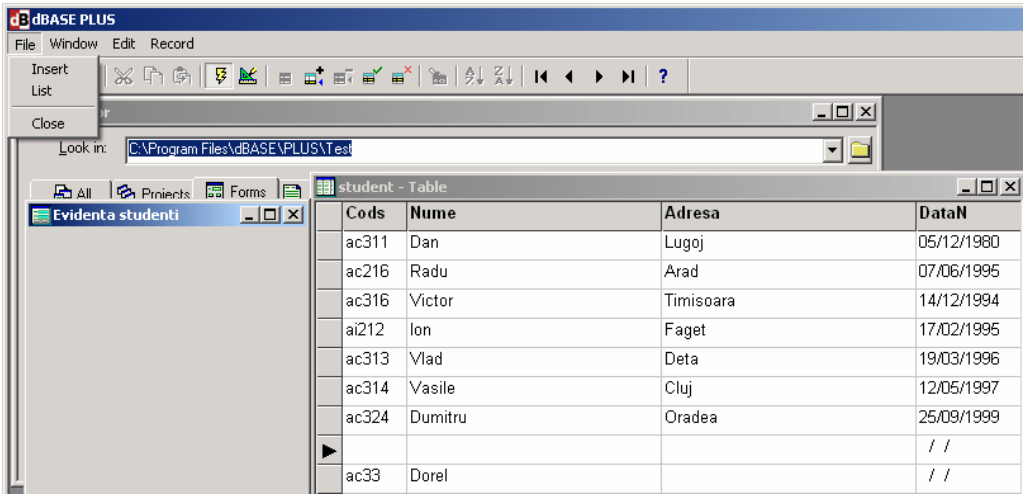
Cu **Insert** se adaugă înregistrări noi (Append).

Din meniul **Record** selectând barele

Next trece la înregistrarea următoare (Skip)

Prev revine la înregistrarea precedentă(Skip -1)

Disp afişează şi permite editarea înregistrării curente (Edit)



Utilizarea meniurilor poate fi combinată cu plasarea pe Form a unor obiecte Text, EntryField, Butoane, ListBox discutate in cursul anterior.

8.2. Definire şi utilizare obiecte multimedia

In dBase se pot păstra în câmpurile unor fişiere obiecte multimedia ca imagini, sunete sau obiecte de tip OLE. Pentru aceasta se folosesc câmpurile de tip BIN şi OLE care fac referinţă la fişierul care conţine obiectul.

8.2.1. Imagini şi sunete

Imaginile se păstrează în fişiere speciale ale sistemului de operare .JPG, .GIF, TIF, BMP, iar sunetele în fişiere Wav.

In fişierul de date dBase imaginile şi sunetele se pot păstra în **câmpuri de tip BIN.**

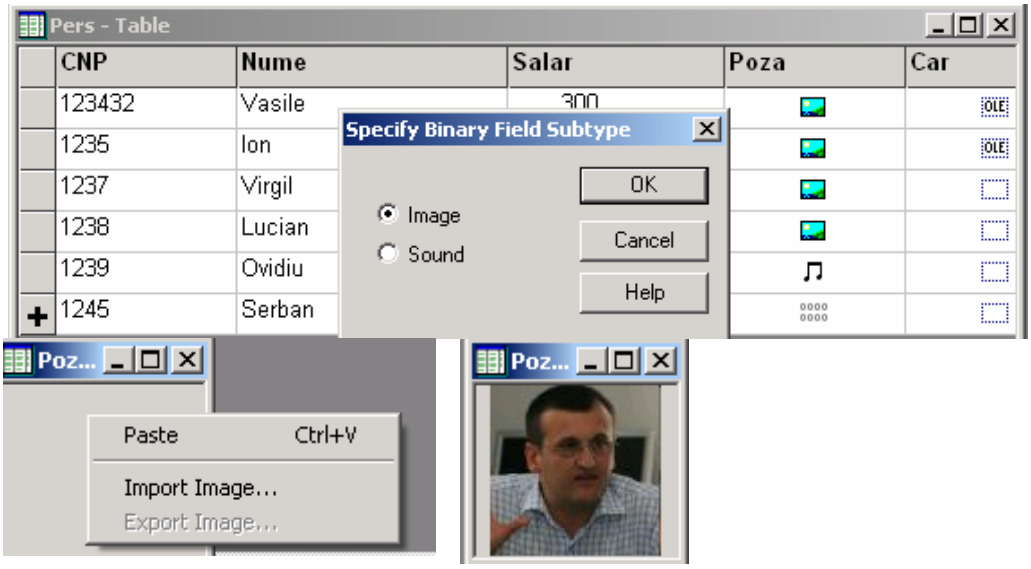
La crearea unei înregistrări câmpurile BIN sunt goale. Ele trebuie să facă referință la fișierele care conțin efectiv pozele sau sunetele.
Se consideră fișierul de studenți la care prin comanda Modi Struc vom adăuga:

- câmpul BIN **Poza** ce va conține poza studentului
- câmpul OLE Car ce va conține o caricatură executată în PaintBrash, sau o caracterizare făcută în Word.

Adăugarea unei poze se face prin:

- Se deschide fișierul şi se dă Browse.
- Se selectează câmpul BIN Poza pentru un student cu DblClick sau F9
- Apare fereastra Empty Binary Field în care se selectează Image Viewer şi OK.
- Va apare fereastra Image Viewer în care se introduce imaginea selectată prin Click dreapta şi Import Image şi se caută imaginea din orice director (se pot lua imagini din directorul Windows).
- Inchizând fereastra, poza rămâne ataşată câmpului şi se afişează cu DblCick.

La fel se procedează şi la schimbarea unei poze.

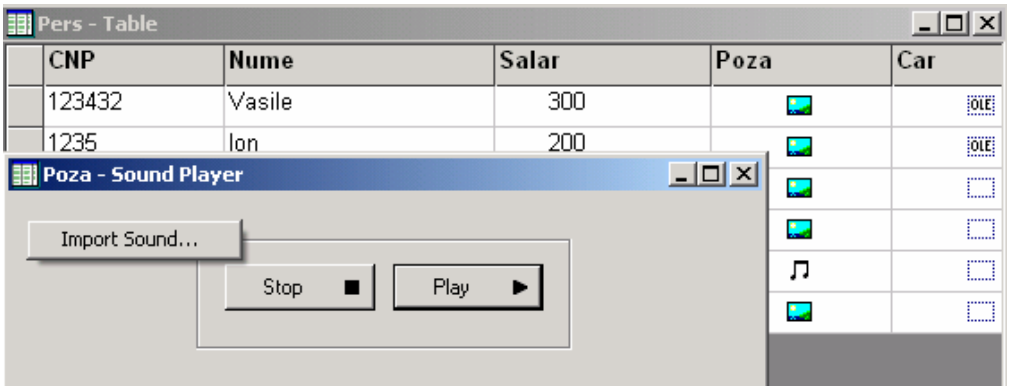


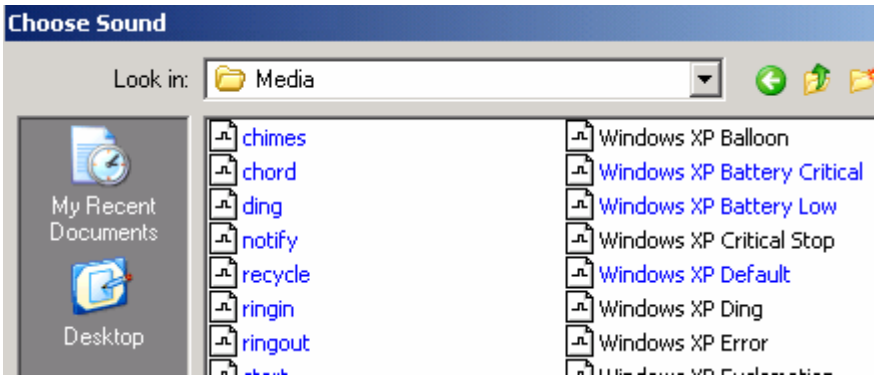
Pentru adăugare fişier de sunet la un câmp BIN

- se selectează Sound Player şi OK
- va apare Stop şi Play.
- se selectează fişierul de sunet (.wav) cu:
  Click dreapta pe fereastra Sound Player/ Import Sound se căută un

fişier în directoarele disponibile şi OK.

Se observă că am introdus sunete în câmpul poza, fiindcă în ambele cazuri tipul câmpului este BIN. Acestui câmp i se asociază un fişier care poate fi de imagini sau sunete.Redarea sunetului ataşat câmpului se face prin DblClick pe câmp şi PLAY.





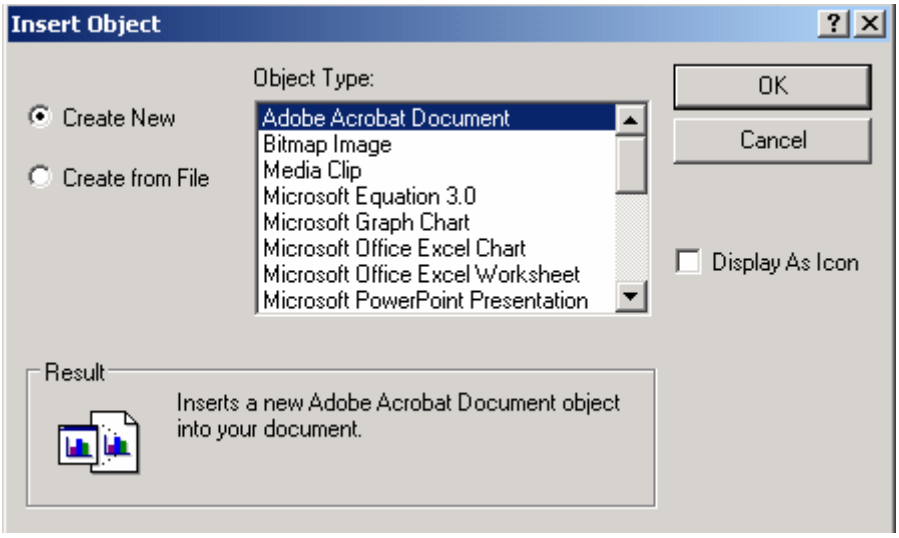
8.2.2. Câmpuri OLE (Object Linking and Embedding)

Câmpurile de tip OLE referă obiecte care nu sunt recunoscute direct de Windows. In aceste obiecte se consideră încapsulate obiecte Windows de diferite tipuri. Ele sun create de **constructori** care permit şi prelucrarea sau dezvoltarea lor. Acestea pot fi documente Word, pagini de calcul Excel, imagini realizate cu Adobe foto, CorelDraw sau PaintBrush. Dacă se dă DblClick pe un asemenea câmp se activează constructorul (Word, Excel, Internet Explorer,..) care primeşte adresa obiectului care va fi deschis şi eventual dezvoltat şi prelucrat. Se poate ataşa astfel un CV redactat în Word care oricând poate fi

afişat dar şi modificat.

Ataşarea acestor obiecte la câmpurile OLE se face astfel.

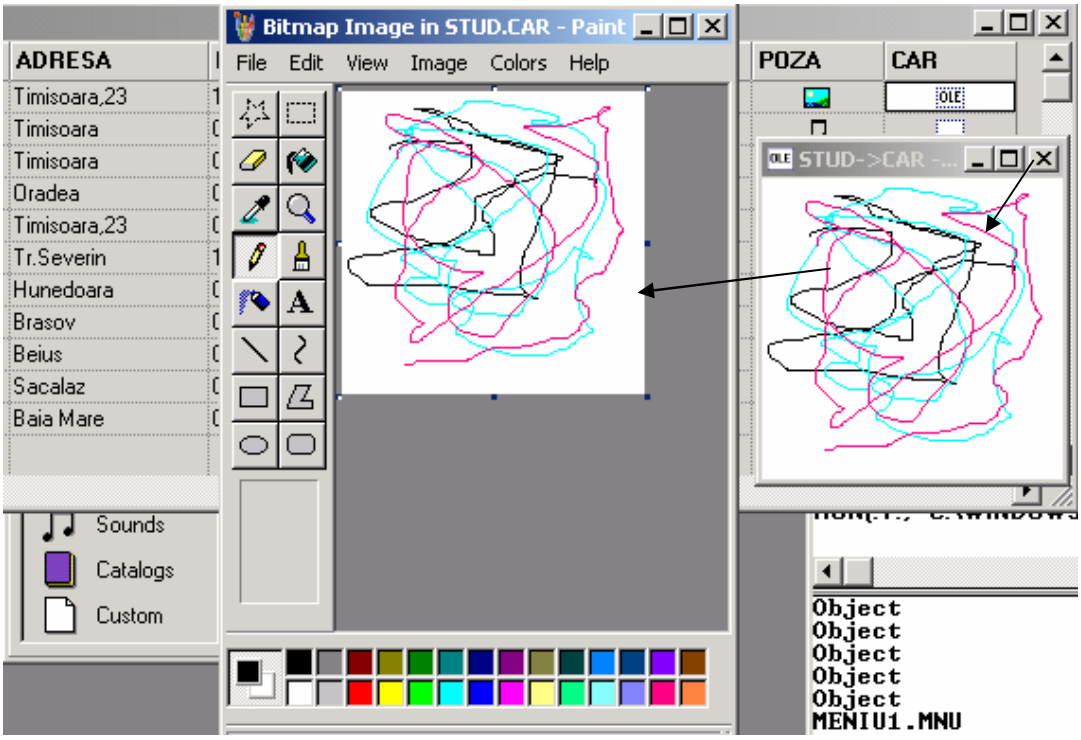
- Se dă Dblclick pe câmpul OLE ce trebuie iniţializat sau modificat.
- Apare fereastra OLE Viewer
- Se selectează constructorul cu **EDIT/Insert OLE (PaintBrush, Word, Excel,..)**



- Se creează sau se modifică obiectul
- La închiderea obiectului se va memora legătura în câmpul OLE din fişier.

La modificarea unui obiect OLE:

- se dă DblClick pe câmpul OLE ce conţine obiectul
- obiectul va fi afişat într-o fereastră
- se dă DblClick pe fereastra ce conţine obiectul şi se va activa programul constructor (PaintBrush în exemplul nostru)
- se modifică obiectul şi se închide programul constructor memorând în câmp legătura spre noul obiect modificat



8.2.3 . Afişarea imaginilor şi redarea sunetelor prin program

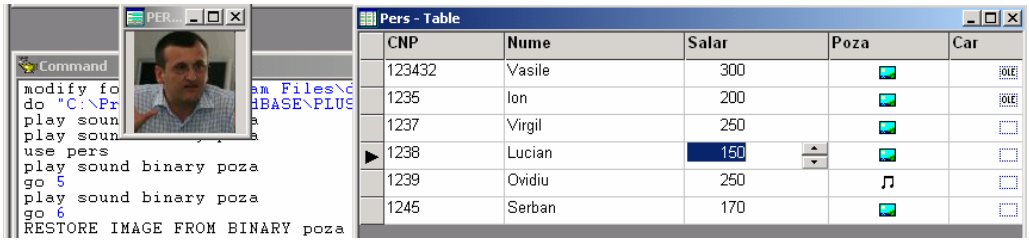
Afişarea imaginilor prin program se face folosind comanda:

RESTORE IMAGE FROM fisier.bmp dacă se specifică direct fişierul sursă

BINARY camp dacă se dă un câmp BIN din fişier

Ex:

```
Use pers
Go 5
RESTORE IMAGE FROM BINARY poza
```



RESTORE IMAGE FROM C:\VISUALDB\SAMPLES\AIRBRLN2

Un câmp BIN ce conține o poză poate fi ataşat unui obiect Windows IMAGE, care poate fi considerat un container pentru poză, fiindcă aceasta se va afişa la dimensiunile cerute.

DEFI IMAGE foto1 OF F1 AT 2,2 PROP width 20, height 10,;
DataSource 'BINARY Poza'
FILE fisier.bmp

Redarea sunetelor prin program se realizează prin comanda:

PLAY SOUND BINARY camp_sunet	pentru sunet ataşat la un câmp BIN
FILENAME fisier.wav	pentru un sunet dintr-un fişier.wav

Ex:
PLAY SOUND BINARY sunet
PLAY SOUND FILENAME c:\visualdb\samples\welcome.wav

Completarea dinamică în program a unui câmp BIN se face prin comanda REPLACE:

REPLACE BINARY poza FROM fisier.bmp
REPLACE BINARY sunet FROM fisier.wav

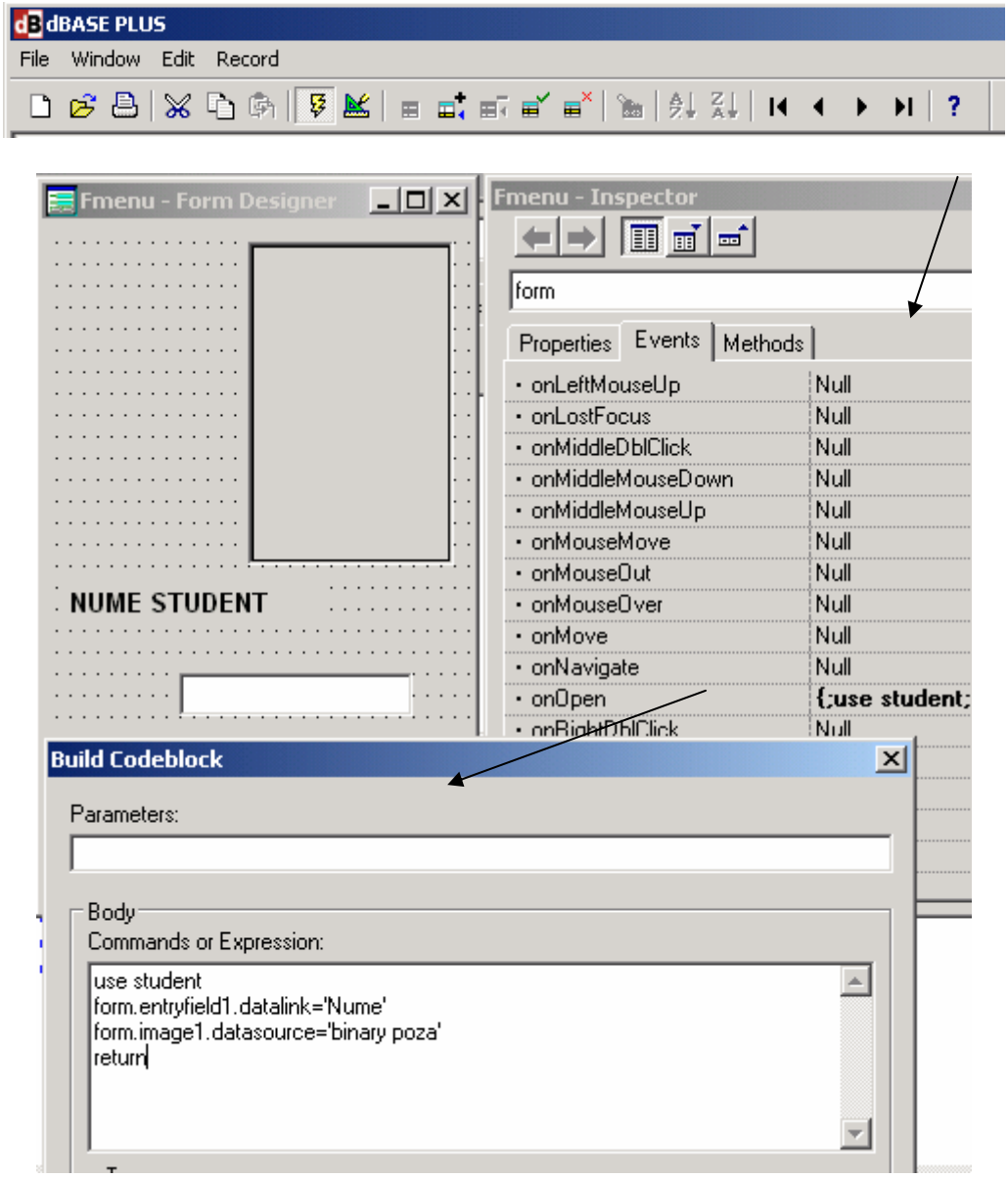
Copierea unei poze sau sunet dintr-un câmp într-un fişier se face prin:

COPY BINARY poza TO fisier.bmp
COPY BINARY sunet TO fisier.wav

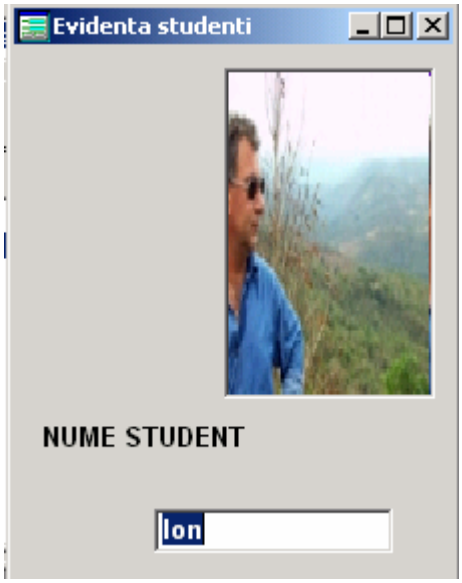
8.2.4. Afişare poze din câmp BIN intr-o fereastra

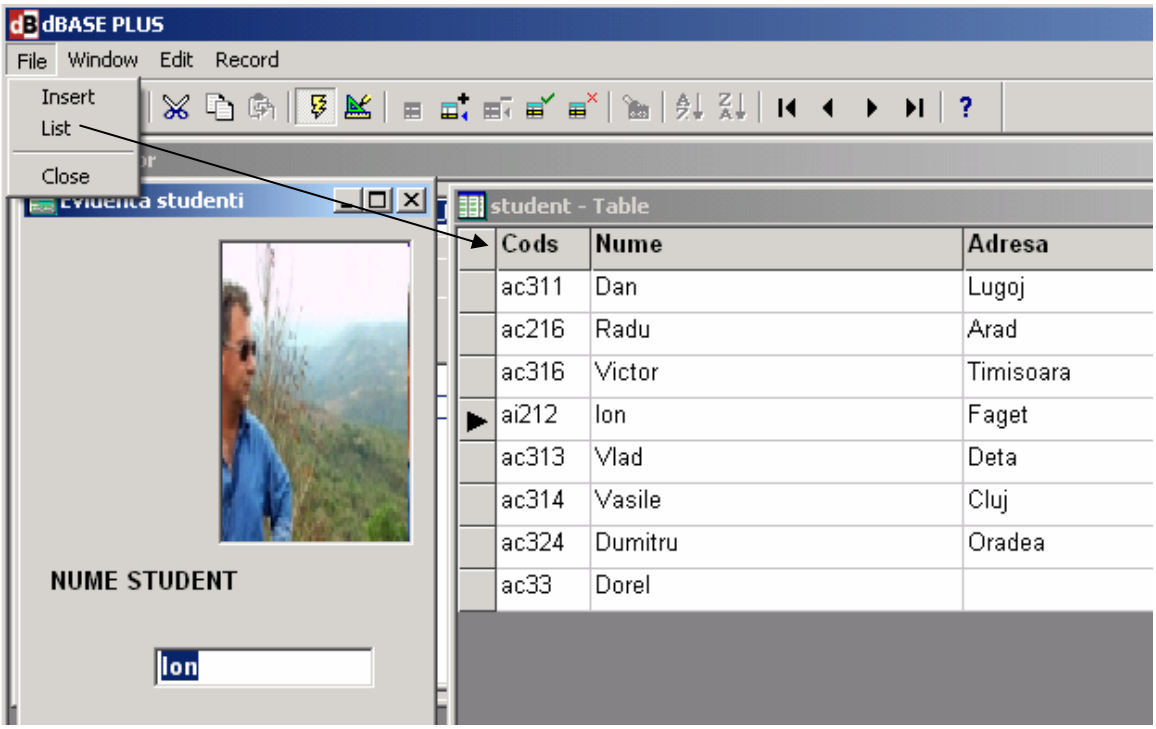
Pentru programul cu meniuri Fmenu proiectat anterior vom plasa pe FORM un obiect IMAGE, un text şi un EntryField. In procedura declanşată prin OnOpen la deschiderea Form-ului o sa-i ataşăm obiectului Image un câmp poza din fişierul Student, iar pentru obiectul EntryField se asociază câmpul Nume. În acest fel se va afişa în orice moment poza şi numele studentului din înregistrarea curentă. Selectarea înregistrării se poate face din obiectul Browse afişat din meniul File/List sau meniul Record/Next şi Record/Prev. Deplasarea

în fişier se poate face şi din bara de Speed menu folosind:



La deplasarea în fişier prin meniu (Next, Prev) sau cu Browse se va afişa poza studentului din înregistrarea curentă. In acest fel este posibil să se selecteze o înregistrare din fişier pe baza pozei persoanei, pentru a afişa datele personale.


Evidenta studenti
NUME STUDENT
Ion


dBASE PLUS
File
Window
Edit
Record
Insert
List
Close
Evidenta studenti
NUME STUDENT
Ion
student - Table
Cods
Nume
Adresa
ac311 Dan Lugoj
ac216 Radu Arad
ac316 Victor Timisoara
ai212 Ion Faget
ac313 Vlad Deta
ac314 Vasile Cluj
ac324 Dumitru Oradea
ac33 Dorel

Programul generat este:

```
parameter bModal
local f
f = new FmenuForm()
if (bModal)
  f.mdi = false // ensure not MDI
  f.readModal()
else
  f.open()
endif

class FmenuForm of FORM
  with (this)
    onOpen = {;use student; form.entryfield1.datalink='Nume';
form.image1.datasource='binary poza'; return}
    onClose = {;use}
    height = 16.0
    left = 24.4286
    top = 0.5
    width = 31.2857
    text = "Evidenta studenti"
    menuFile = "M1.mnu"
  endwith

 this.IMAGE1 = new IMAGE(this)
  with (this.IMAGE1)
    onLeftMouseDown = class::IMAGE1_ONLEFTMOUSEDOWN
    height = 7.5
    left = 15.0
    top = 0.5
    width = 15.0
  endwith

this.ENTRYFIELD1 = new ENTRYFIELD(this)
  with (this.ENTRYFIELD1)
    height = 1.0
    left = 10.0
    top = 10.5
    width = 17.0
    fontBold = true
```

```
    value = " "
  endwith

  this.TEXTLABEL1 = new TEXTLABEL(this)
  with (this.TEXTLABEL1)
    height = 1.0
    left = 2.0
    top = 8.5
    width = 19.0
    text = "NUME STUDENT"
    fontBold = true
  endwith
endclass

// Generated on 02/04/2012
//
parameter formObj
new M1MENU(formObj, "root")
class M1MENU(formObj, name) of MENUBAR(formObj, name)
  this.MENU2 = new MENU(this)
  with (this.MENU2)
    text = "File"
  endwith

 this.MENU2.MENU3 = new MENU(this.MENU2)
  with (this.MENU2.MENU3)
    onClick = {;Append}
    text = "&Insert"
  endwith

this.MENU2.MENU4 = new MENU(this.MENU2)
  with (this.MENU2.MENU4)
    onClick = {;Browse}
    text = "&List"
  endwith
  this.MENU2.MENU5 = new MENU(this.MENU2)
  with (this.MENU2.MENU5)
    text = ""
    separator = true
  endwith
```

```
this.MENU2.MENU6 = new MENU(this.MENU2)
  with (this.MENU2.MENU6)
    onClick = {;Close database}
    text = "Close"
  endwith


 this.MENU20 = new MENU(this)
  with (this.MENU20)
    text = "&Window"
  endwith

 this.MENU14 = new MENU(this)
  with (this.MENU14)
    text = "&Edit"
  endwith

this.MENU14.UNDO = new MENU(this.MENU14)
  with (this.MENU14.UNDO)
    text = "&Undo"
    shortCut = "Ctrl+Z"
  endwith

this.MENU14.CUT = new MENU(this.MENU14)
  with (this.MENU14.CUT)
    text = "Cu&t"
    shortCut = "Ctrl+X"
  endwith

this.MENU14.COPY = new MENU(this.MENU14)
  with (this.MENU14.COPY)
    text = "&Copy"
    shortCut = "Ctrl+C"
  endwith

 this.MENU14.PASTE = new MENU(this.MENU14)
  with (this.MENU14.PASTE)
    text = "&Paste"
    shortCut = "Ctrl+V"
```

```
  endwith

this.MENU42 = new MENU(this)
  with (this.MENU42)
    text = "Record"
  endwith

this.MENU42.MENU43 = new MENU(this.MENU42)
  with (this.MENU42.MENU43)
    onClick = {;Skip}
    text = "Next"
  endwith

 this.MENU42.MENU45 = new MENU(this.MENU42)
  with (this.MENU42.MENU45)
    onClick = {;Skip-1}
    text = "Prev"
 endwith

this.MENU42.MENU47 = new MENU(this.MENU42)
  with (this.MENU42.MENU47)
    text = "Disp"
    separator = true
  endwith

  this.MENU42.MENU49 = new MENU(this.MENU42)
  with (this.MENU42.MENU49)
    onClick = {;Disp}
    text = "Display"
  endwith

endclass
```

9. BAZE DE DATE RELAȚIONAL-OBIECTUALE

9.1. Crearea dinamica a interfeței grafice

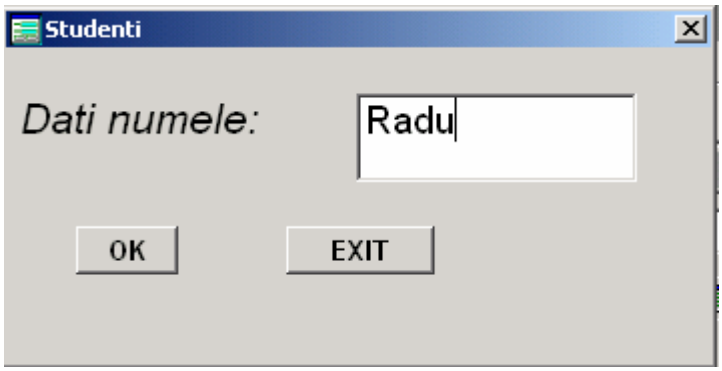
In continuare se va crea o aplicații care efectuează o căutare intr-o bază de date şi afişează rezultatele intr-o fereastra F2 folosind obiecte grafice şi multimedia Windows definite de utilizator in program, fără a utiliza Designer-ul din dBase.
Se recomanda consultarea documentației privitoare la DEFINE si MsgBox() din HELP.

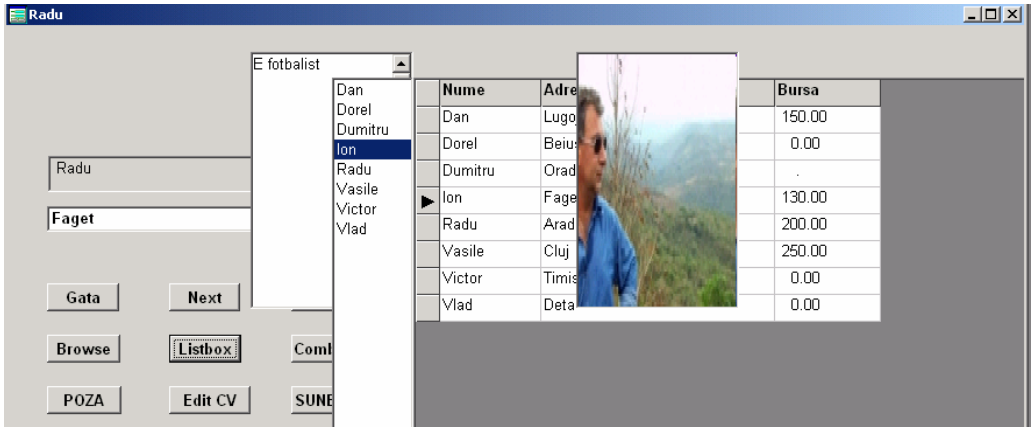
Fereastra modală are proprietatea MDI=.F. şi nu permite ieşirea din fereastră decât prin apăsarea unui buton, care închide fereastra şi returnează numele obiectului care a fost apăsat, pentru a continua programul pe o anumită ramură. Deschiderea ferestrei modale se face cu READMODAL(). Acest tip de fereastră se foloseşte în MsgBox() şi în toate mesajele de eroare ale diferitelor aplicații.

Se defineşte prima data o fereastră modală F1 pentru dialog cu utilizatorul, care cere numele unui student care trebuie căutat în baza de date prin index. Dacă studentul a fost găsit se defineşte şi se deschide fereastra F2 pe care se plasează mai multe butoane, care vor servi la lansarea mai multor proceduri care vor realiza diferite funcții:

- definire un browse care afişează conținutul fişierului Student
- definire şi activare un ListBox cu numele studenților
- Afişare poza studentului selectat într-un obiect Image creat
- Afişare CV student selectat intr-un obiect Editor creat
- O înregistrare sonoră pentru acel student
- Afişare student următor şi precedent, care are acelaşi nume

Numele studentului căutat, dat în EntryField-ul din fereastra F1, se va memora în variabila VN specificată ca DataLink . Pentru ca VN şi fereastra F2 să fie accesibile şi în proceduri vor fi declarate Public.





In program pentru a gestiona suprafața ecranului s-a prevăzut ca la click dreapata pe obiect (Editor,ListBox, Combobox, Imagine, Browse) sau butonul corespunzător sa se steargă obiectul cu Release.

*** Program creare interfata dinamica**
set talk off
public vn,f2,t1
use student index inume alias st
defi form f1 prop text 'Studenti',width 50,height 7,; // definire fereastra modala
top 0, left 0,mdi .f., escexit .f.

* Fereastra initiala va fi de tip modal (mdi=.F.), ceea ce inseamna faptul ca iesirea se poate face
* doar prin închidere prin intermediul butoanelor OK sau Cancel.
* Deschiderea acestei ferestre se face cu READMODAL().

```
vn=space(10)
defi text t1 of f1 prop text 'Dati numele:',;
fontitalic .t., fontsize 16, top 1, left 1, ;
width 25, height 2
defi entryfield e1 of f1 at 1,25 prop width 20,;
height 2, border .t., fontsize 14, datalink 'vn',;
statusmessage 'Introduceti numele studentului'
* Variabila vn din program va prelua valoarea introdusa pentru numele cautat.

defi pushbutton OK of f1 at 4,5 prop text 'OK',fontbold .t.,;
 onclick {; f1.close()}
defi pushbutton CANCEL of f1 at 4,20 ;
 prop text 'EXIT', onclick {; f1.close()},fontbold .t.

* Readmodal()  Deschide fereastra iar la inchidere da numele obiectului care a
inchis-o
*          si se verifica daca butonul apasat a fost "OK".

Do case
   case f1.readmodal().name #'OK'   // 'CANCEL'  // deschidere fereastră
modală F1
// dacă obiectul selectat nu este OK atunci se trece la EndCase
   case seek (trim(vn))          // selectare primul student cu numele dat
// dacă studentul căutat nu există se returnează .False. şi se trece la EndCase

* Definire fereastră F2 şi obiectele conţinute împreună cu procedurile asociate
defi form F2 at 3,3 prop width 60, height 15,;
         text 'Modificare adresa',    onclose{;close tables}

* Daca cautarea s-a efectuat cu  success atunci se va defini si deschide fereastra
f2,care contine :
* Text cu nume, prenume si telefon
* Entryfield cu adresa ce se poate modifica
* Butoane pentru terminare(GATA), avans inainte(NEXT) sau inapoi (PREV)
in fisier.
* Butoane pentru definire unor obiecte: BROWSE, LISTBOX, COMBOBOX,
fotografie (POZA) si sunet

    f2.text=trim(vn)   // numele cautat in titlul ferestrei
defi text t1 of f2 at 5,5 prop text nume , ;
```

```
          width 100, border .t., height 1.5
defi entryfield e2 of f2 at 7,5 prop datalink 'adresa',;
 width 50, statusmessage 'Puteti modifica adresa', fontbold .t.
defi pushbutton poz1 of f2 at 14,5 ;
    prop text 'POZA', fontbold .t.,;      // nume buton afisare poza
          onclick pimag, ;          //procedura pentru afisare poza
    onrightmousedown{; f2.foto1.release()}              // sterg poza
defi pushbutton gata of f2 at 10,5 prop text 'Gata',fontbold .t.,;
          onclick{;form.close()} //inchidere Form F2
defi pushbutton next of f2 at 10,20 prop text 'Next',fontbold .t.,;
          onclick pnext   // procedura asociata
defi pushbutton prev of f2 at 10,35 prop text 'Prev',fontbold .t.,;
          onclick pprev    // procedura asociata Pprev
defi pushbutton brow of f2 at 12,5 prop text 'Browse',fontbold .t.,;
          onclick pbr,;     // procedura afisare Browse
onrightmousedown{;f2.b1.release()}  // sterge Browse
defi pushbutton listb of f2 at 12,20 prop text 'Listbox', fontbold .t.,;
         onclick plistb,onrightmousedown{;f2.l1.release()}
defi pushbutton combo of f2 at 12,35 prop text'ComboBox', fontbold .t.,;
        onclick pcb,;       // procedura afisare ComboBox
onrightmousedown{;f2.cb.release()} // sterge ComboBox
defi pushbutton CV of f2 at 14,20 prop text'Edit CV', width 10, fontbold .t.,;
        onclick pedit,;    // procedura afisare camp memo CV
onrightmousedown{;f2.ed1.release()}           // sterge Edit

* Definire buton SUNET care reda un fisier.wav asociat unui camp Binary din
fisier
defi pushbutton S1 of f2 at 14,35 prop text'SUNET', fontbold .t.,;
        onclick {;Play Sound Binary sunet}
 F2.Open()     // deschidere Form F2 si obiectele componente


  otherwise
     close tables          // daca nu s-a gasit inregistrarea
     mes='nu exista  '+ vn
     msgbox(mes,'Atentie',40) // afisare mesaj eroare
endcase
return
```

```
* Procedura Afisare inregistrare urmatoare. (Buton NEXT)
* Se da un mesaj  daca este ultima care indeplineste conditia de grup.
Proced Pnext
Public vn,f2,t1 // variabile globale care se pot referi
  skip 1                  // Inregistrarea urmatoare
  if nume = trim(vn)   // Verificare conditie
    f2.t1.text=nume
  else
    skip - 1
    msgbox ('Ultimul cu acest nume','gata',1)
  endif
return

* Procedura Afisare inregistrarea anterioara. (Buton PREV)
Proced Pprev
Public vn,f2,t1 // variabile globale care se pot referi
  skip - 1        // Inregistrarea precedenta
  if nume=trim(vn)     // Verificare conditie
    f2.t1.text=nume
  else
    skip +1
    msgbox ('Ultimul cu acest nume','gata',1)
  endif
return

   * Definire si afisare  obiect Browse pe F2
Proced Pbr
public f2                 // Obiect definit global ce poate fi referit
  f2.width=125          // modificare dimensiuni Form F2
  f2.height=22
defi browse b1 of f2 at 2,50 prop width 80,height 20,;
  fields 'nume,adresa,bursa',;   // campuri afisate
onrightmousedown{;f2.b1.release()}  //sterge ob. browse
f2.b1.alias='st'          //  fisierul din care se afiseaza
return

* Definire si afisare obiect Listbox pe F2
Proced Plistb
public f2                 // Obiect definit global ce poate fi referit
defi listbox l1 of f2 at 2,40 prop width 10,height 20,;
```

```
datasource 'field nume' // campul asociat la Listbox
onrightmousedown{;f2.l1.release()}  //Sterge listbox
return

* Definire si afisare ComboBox
Proced Pcb
public f2                  // Obiect definit global ce poate fi referit
defi combobox cb of f2 at 2,60 prop width 10,height 20,;
 datasource 'field nume',; // Camp asociat Combobox
 onrightmousedown{;f2.cb.release()}// sterge combobox
return

* Procedura de definire si afisare a unei imagini
Proced Pimag
public f2                  // Obiect definit global ce poate fi referit
defi image foto1 of f2 at 1,70 ;
       prop width 20, height 10,;
 datasource 'Binary poza',; // Camp Binary ce contine poza
      onrightmousedown{; f2.foto1.release()}  // sterge poza
return

* Procedura de definire a unui obiect EDITOR pentru afisare camp Memo
Proced Pedit
public f2                  // Obiect definit global ce poate fi referit
defi EDITOR Ed1 of f2 at 1,30 ;
       prop width 20, height 10,;
 DataLink 'CV', ;                  // Camp Memo ce contine CV
      onrightmousedown{; f2.Ed1.release()}  // sterge obiectul Edit
return
```

9.2. Definire clase utilizator

Până acum s-au folosit obiecte din clasele standard Windows. Utilizatorul poate să-şi definească noi clase pentru care va preciza prin constructorul:

CLASS…ENDCLASS :

- numele clasei şi parametrii formali dacă există ;
- numele superclasei căreia îi moşteneşte proprietăţile ;
- proprietăţile structurale, atributele şi valorile lor implicite ;

- metodele (funcțiile) de prelucrare asociate clasei de obiecte.

Sintaxa comenzii CLASS este :

```
CLASS  nume_clasa  [ ( parametrii)] [ CUSTOM]
        [ PARAMETERS  parametrii ]              -- parametrii formali
        [ OF  superclasa  [ ( parametrii)] ]    -- clasa părinte
        . . .
  secvență cod constructor                      -- secvență de comenzi ce se
execută la
        . . .                                   -- definirea obiectului
        . . .
        funcții membre                          -- funcțiile interne clasei sunt
proceduri şi
        . . .                                   -- funcții care vor forma metodele
clasei
ENDCLASS
```

Pentru exemplificare vom declara o clasă neconvențională PISICA şi o subclasă PISICUTA care se obține modificând anumite proprietăți ale clasei, sau adăugând proprietăți noi. Definirea claselor se va face după programul principal.

```
// Program care utilizeaza obiecte din clase definite de  utilizator
Set Talk off
Clear
// Definire obiecte din  clase utilizator declarate
// definire obiect din subclasa Pisicuta
Tom = New PISICUTA()      // definire obiect pisicuta
Tom.rasa='Maidaneza'

  ? Tom.rasa            // afisare rasa 'maidaneza' a a obiectului
        ? Tom.color             // afiseaza culoare 'neagra' implicita pentru
Pisicuta
        Tom.miau()              // proprietate comuna mieunat
Tom.prezent(' pe strada')
// Va afisa  'Eu sunt pisica maidaneza neagra Adresa:pe strada'

// Definire un obiect pisica
Lace = New PISICA()                 // creaza obiect pisica
```

```
        Lace.miau()                          // proprietate comuna mieunat
        ? Lace.color            // afiseaza 'gri' implicit
  ? Lace.name
  ?   Lace.color
  Lace.prezent('pe soba')                          // afiseaza culoare 'gri'
        Tom.color = 'maro'                   // schimbare culoare
        Tom.rasa = 'angora'                  // schimbare rasa
  ? Tom.color, Tom.rasa
Return

* Definire Clasa  pisici
CLASS PISICA
     this.Name='Pisica'                      // definire obiect pisica
        this.rasa = 'siameza'           // definire proprietati clasa pisica
        this.dim = 'mare'                    // dimensiune
        this.color = 'gri'                   // culoare
        this.adulta = .T.
// definire metoda pentru sunet pisica memorat intr-un fisier
        this.miau    ={;PLAY    SOUND    Filename    'C:\Media\Windows
Shutdown.wav' }
Procedure Prezent      // definire procedura specificare nume, rasa si loc
Parameter locul
? 'Eu sunt ',this.name, this.rasa, this.color
? ' Adresa este:  ', locul
return
ENDCLASS

// Definire subclasa Pisicuta care mosteneste proprietatile clasei Pisica
CLASS PISICUTA  OF PISICA            // definire subclasa
        this.adulta = .F.                    // modificare proprietati
        this.color = 'neagra'
        this.dresat=1           // adaugare proprietate noua
        This.dim='mica'
ENDCLASS
```

Prin rularea programului se obține pe monitor:

```
Maidaneza
neagra
Eu sunt  Pisica Maidaneza neagra
 Adresa este:    pe strada
```

gri
Pisica
gri
Eu sunt Pisica siameza gri
Adresa este: pe soba
maro angora

9.3. Implementarea bazelor de date relaţional obiectuale

Bazele de date relaţional-obiectuale presupun păstrarea calităţilor de flexibilitate şi robusteţe a bazelor de date relaţionale şi adăugarea unor facilităţi noi obiectuale. Se ţine cont de faptul că obiectele sunt perisabile (în toate limbajele), fiind definite în memoria centrală, dar au calitatea de a putea fi de mare complexitate. Gradul înalt de complexitate admis fac dificilă găsirea unor metode de memorare a obiectelor pe disc. Din acest motiv limbajele de programare permit definirea unor clase de obiecte standard sau utilizator în RAM, fără a furniza soluţii de memorare persistă pe disc. Baze de date pur obiectuale s-au realizat experimental, dar nu au ajuns să fie extinse comercial.

La proiectarea BD relaţional-obiectuale se îmbină posibilităţile de definire a unor structuri de **obiecte complexe perisabile**, cu posibilitatea de a memora **proprietăţile obiectelor într-o formă persistentă în fişiere de date** (tabele) pe disc.

BD relaţional obiectuale se implementează în programe astfel:

- Se definesc una sau mai multe clase de obiecte, care au proprietăţi structurale ale căror valori vor fi luate din câmpurile curente ale înregistrărilor unor fişiere de date.
- Clasele de obiecte definite vor avea metode care sunt proceduri de căutare, afişare, modificare în fişierele de date, pe baza unor valori introduse în obiectele EntryField de către utilizator. Valorile câmpurilor din înregistrările selectate vor servi la completarea proprietăţilor obiectelor (mapare).

Dacă se consideră o **clasă Pers**, ea va conţine ca atribute datele personale generale valabile pentru copii, elevi, muncitori, studenţi, funcţionari, pensionari, şomeri,..

CNP, Numele, Adresa, Data naşterii, Locul naşterii, iar ca metode:

Pcaut – căutare persoană cu numele dat (ex. PCaut(Vnume)
PAfis – afişare date pentru persoana selectată

Pe baza clasei Pers se pot defini alte **subclase** care se referă la anumite categorii de persoane pentru care la atributele clasei Pers se vor adăuga alte atribute specifice:
Student – CNP, Nume, Adresa, DataN, LoculN**, + Cods, Univ, Fac, Sectie, Media**
Metoda **AfisN -** afişare note
Salariat – CNP, Nume, Adresa, DataN, LoculN**, + Functie, Tel, Salar, Institutie**
Metoda **Safis -** va afişa Functie, Tel, Salar, Institutie
Profesor – CNP, Nume, Adresa, DataN, LoculN**,+ Grad, Vechime, Salar, Univ**

Pentru a afişa datele personale se va folosi pentru toţi metoda Pafis(), dar pentru afişare note pentru un student AfisN().

Pentru a memora informaţiile despre studenţi vom utiliza fişierul Stud, pentru salariaţi fişierul Sal, iar pentru profesori Prof. Toate aceste fişiere vor avea primele câmpuri ale clasei Pers iar următoarele specifice subclasei.
Se remarcă faptul că **pentru clasa Pers nu există un fişier** alocat, iar obiectele pers nu există decât generic.

Exemplu de implementare BD relaţional-obiectuală:
Vom defini o clasă de persoane **PERS** şi o subclasă **SALARIAT** care moşteneşte proprietăţile şi metodele. Deoarece secvenţa de cod constructor se execută la crearea obiectului, aici vom introduce definirea proprietăţilor, care vor fi variabile de memorie, corespunzătoare câmpurilor fişierului. Metodele clasei vor fi scrise în continuare ca funcţii, care vor realiza operaţiile asupra fişierului. În cadrul clasei se vor defini funcţiile de deschidere de fişier, închidere fişier, căutare după cheie, ştergere înregistrare, afişare date personale. Afişarea datelor suplimentare ale salariatului se va face în funcţia SAFIS.
Având în vedere că înregistrările pot conţine imagini, într-un fişier de persoane se pot memora fotografia, semnătura, amprenta.

Consideram fişierul care conţine date despre persoane având structura şi conţinutul:

Pers - Table Designer

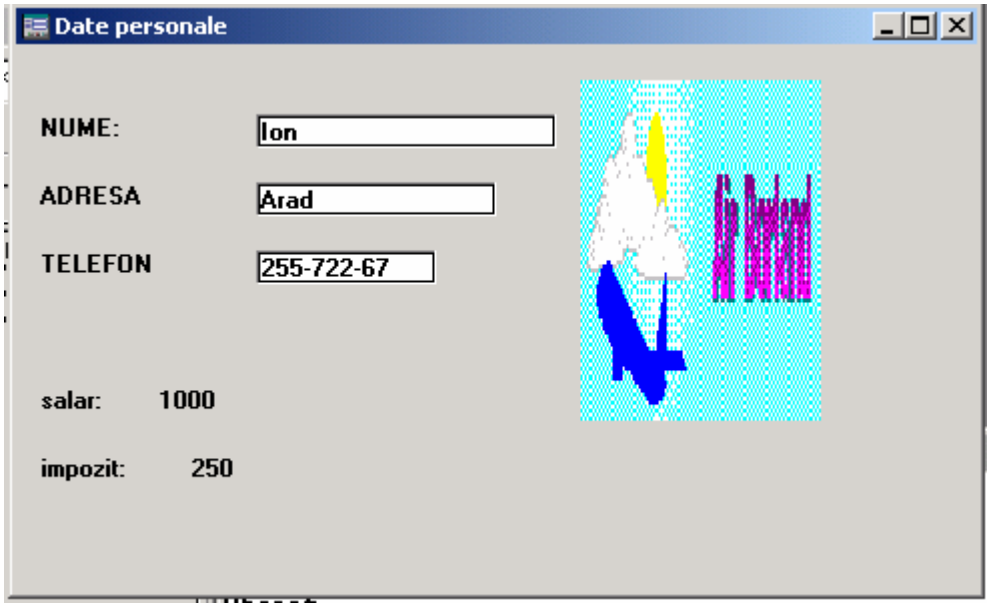
Field	Name	Type	Width	Decimal	Index
1	CNP	Character	10	0	None
2	Nume	Character	15	0	None
3	Adresa	Character	15	0	None
4	Tel	Character	10	0	None
5	Salar	Numeric	10	0	None
6	Poza	Binary	10	0	None
7	Car	OLE	10	0	None

Pers - Table

CNP	Nume	Adresa	Tel	Salar	Poza
123432	Vasile	Timisoara	123	300	
1235	Ion	Arad	234	200	
1237	Virgil	Beius	235	250	
1238	Lucian	Oradea	236	150	
1239	Ovidiu	Lipova	456	250	
1245	Serban	Deva	888	170	

Obiectul clasei **Pers** va conține proprietățile:
Nume - atribut ce reprezintă numele persoanei
Adresa – atribut adresa persoanei
Tel – atribut număr telefon
Form-ul pe care se afişează Nume, adresa şi poza persoanei luate din fişierul Pers
Metoda **Deschid()** – care deschide fişierul de persoane utilizat cu un index
Metoda **Caut**() – care caută o persoană prin indexul Inumep după numele specificat
Metoda **PAfis()** – care afişează datele personale din înregistrarea curentă

Obiectele **subclasei Salariat** din clasa Pers vor conține în plus proprietățile:
Salar – un text ce afişează salarul
Impozit – calculat ca 0.16 din salar
Telb – telefonul de la birou
Metoda Safis() care afişează pe acelaşi form Salarul, impozitul şi telefonul de la birou



*** Program principal**

```
p1=new pers()            // definire obiect P1 din clasa Pers

p1.deschid('Pers','Inumep')     // deschide un fisier  cu index
defi image foto1 of p1.f1 at 1,47 ;
      prop width 20, height 10,;
 datasource 'Binary poza',;      // Camp Binary ce contine poza
     onrightmousedown{; s1.f1.foto1.release()}         // sterge poza

p1.caut('Ion')   // cauta persoana cu numele
p1.f1.open()             // deschide fereastra f1 din obiectul P1
p1.pafis()               // afisare date personale din inregistrarea curenta
* p1.f1.Print() // afisare fereastra din obiectul P1
? p1.nume, p1.adresa  // afisare atribute obiect
wait
          P1.caut('Vasile')         // cauta alta persoana
P1.pafis()
* P1.f1.close() // inchidere fereastra din obiect
wait
 release object p1
use
clear all
return
```

```
s1=new salariat()                  // definire obiect salariat
s1.deschid ('Salariat','Inumes')
s1.caut('Sandu')
s1.f1.open()
s1.safis()

wait
use
release object s1
Return


// Definire clasa persoane

Class Pers
  this.nume=space(20)          // proprietati persoana
  this.adresa=space(15)
  this.tel=space(10)

  this.f1=new Form('Persoane')         // form ca o componenta clasa persoana
with(this.f1)
    height=16
    width=80
    text='Date personale'
    onclose={;Clear all}
    onopen={; f1.e1.datalink='nume';f1.e2.datalink='adresa';f1.e3.datalink='tel'}
endwith

 this.t1=new text(this.f1,'Nume')       //  Nume - text pe pers.Form
with(this.t1)
    top=2
    left=2
    width=20
    text='NUME:  '
endwith

  this.e1=new entryfield(this.f1,'enume')       // EntryField pentru Nume
with(this.e1)
    top=2
    left=20
```

```
      width=25
      datalink='this.nume'
 endwith

  this.t2=new text(this.f1)       // Adresa - text
with(this.t2)
      top=4
      left=2
      width=20
      text='ADRESA  '
    endwith

   this.e2=new entryfield(this.f1,'eadresa')      // EntryField pentru Adresa
   with(this.e2)
      top=4
      left=20
      width=20
      datalink='this.adresa'
    endwith

   this.t3=new text(this.f1)                 // Telefon - text
with(this.t3)
      top=6
      left=2
      width=20
      text='TELEFON  '
endwith

   this.e3=new entryfield(this.f1,'etel')           // EntryField pentru telefon
with(this.e3)
      top=6
      left=20
      width=15
      datalink='this.tel'
endwith

  Function Deschid    // deschidere fisier de persoane
      parameter fis,index          // nume fisier si index utilizat
      use &fis index &index in 1
   return 1
```

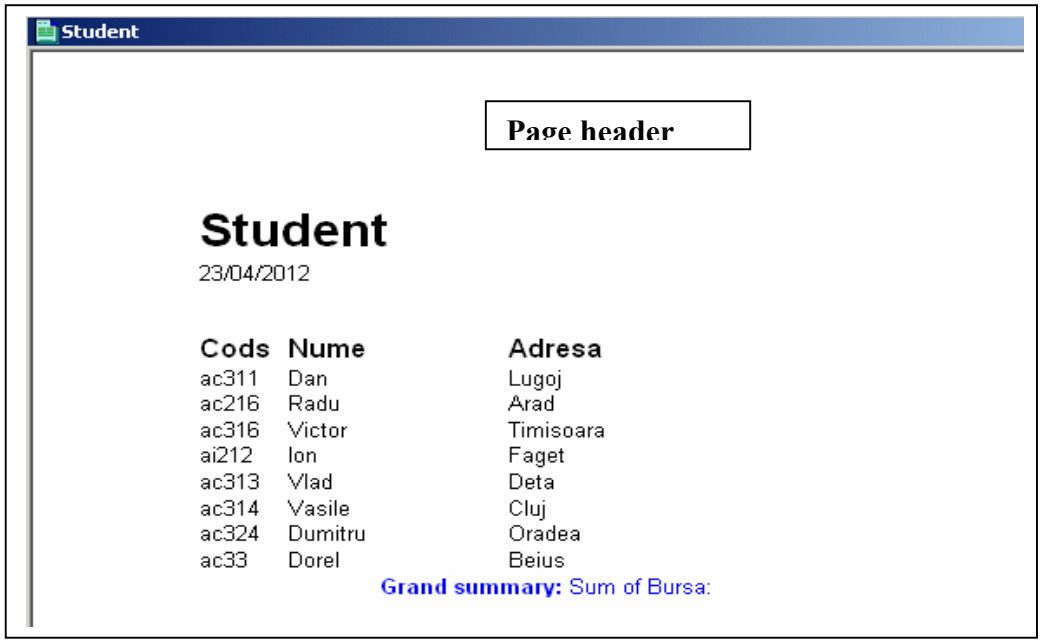
```
  Function Caut                  // cautare persoana in fisier
    parameter cheie    // parametru de cautare
    seek trim(cheie)
    if eof()
      x=msgbox(1, 'Atentie!', "Nu exista ! " + cheie)
    endif
  return 1
    Function Pafis               // metoda afisare date personale
    this.nume=nume   // completare valori atribute persoana
    this.adresa=adresa
    this.tel=tel
    this.f1.refresh()
  return 1
endclass
// Definire subclasa Salariat al clase Pers
Class Salariat of Pers
  this.salar=0             // definire atribute noi pentru salariat
  this.impozit=0
  this.telb=Space(10)

  this.t5=new text(this.f1,'salar')
  this.t5.width=30
  this.t5.top=10
  this.t5.left=2
    this.t6=new text(this.f1,'impozit')
  this.t6.width=30
  this.t6.top=12
  this.t6.left=2

  Function Safis        // definire metoda afisare date salariat
    this.Pafis() // afisare date personale
    this.salar=salar
    this.impozit=0.16 * this.salar
    this.t5.text='salar: '+str(this.salar)
    this.t6.text='impozit: '+str(this.impozit)
    this.f1.refresh()
  return (this.salar)

endclass
```

10. GENERARE RAPOARTE ŞI ETICHETE

10.1. Crearea de Rapoarte

Intr-o Bază de date se păstrează structurat sub formă de tabele un mare volum de informație. Prin programe se asigură crearea şi actualizarea înregistrărilor din tabele folosind proceduri, activitate prin comenzi procedurale, prin meniuri sau prin interfețe grafice. Regăsirea informațiilor din BD se face prin utilizarea fişierelor index şi a relațiilor dintre tabele. Interogarea BD presupune afişarea informațiilor dorite sub forma cerută pe ecran sau la imprimantă. Pe ecran valorile înregistrărilor se pot afişa pe formulare (Form) într-o formă grafică cerută (FontSize, FontType, Color), folosind obiectele windows discutate anterior (Text, EntryField, Image, OLE).

Cea mai simplă formă de afişare a unui fişier de date ca tabel se face folosind comanda LIST sau BROWSE, care afişează coloanele specificate ale înregistrărilor. In aplicații se cer frecvent liste de complexitate mai mare, care să cuprindă totaluri generale sau parțiale (pe facultăți, pe secții, pe meserii, pe produse), antet instituție, data curentă, paginare, sortare. Aceste liste se pot obține prin programe de complexitate mai mare, după modelul prezentat în capitolul proceduri. Scrierea unor asemenea programe necesită un efort mai mare de programare şi poate dura câteva zile sau săptămâni (state de plată, state de funcțiuni, balanțe de verificare).



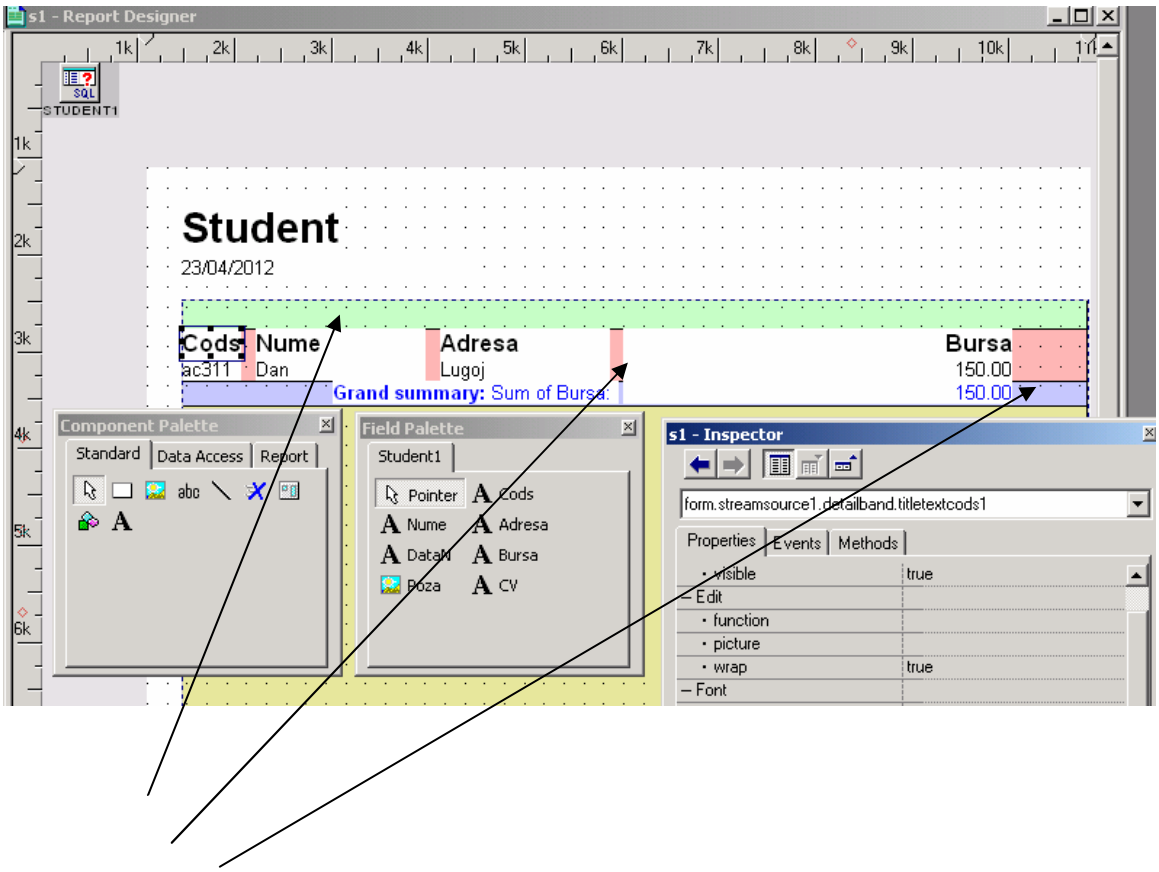
Pentru cazul în care asemenea liste se cer într-un timp scurt(o zi sau câteva ore) şi se referă în general la un fişier se poate recurge la un **generator de rapoarte**, care permite proiectarea rapidă a raportului în mediu Windows. Dacă se dă DblClick pe Untitled în grupul Report din Navigator se activează generatorul de rapoarte cu cele două moduri Expert şi Designer. Se generează un program cu extensia .REP. Modul Expert este cel mai simplu dar şi rezultatul este simplist. Rezultatul este mai simplu decât la comanda LIST.

Se poate lucra mai simplu în modul Des **Page detail** poate activa şi prin:

- File/New/Report din meniul principal
- Folosind New şi Report din Speed bar icon-ul
- Comanda CREATE REPORT r1.rep dacă există un fişier deschis

Page footer

Dacă am deschis anterior fişierul STUD prin comanda **CREATE REPORT s1** se deschide o fereastră cu programul pentru generare rapoarte.

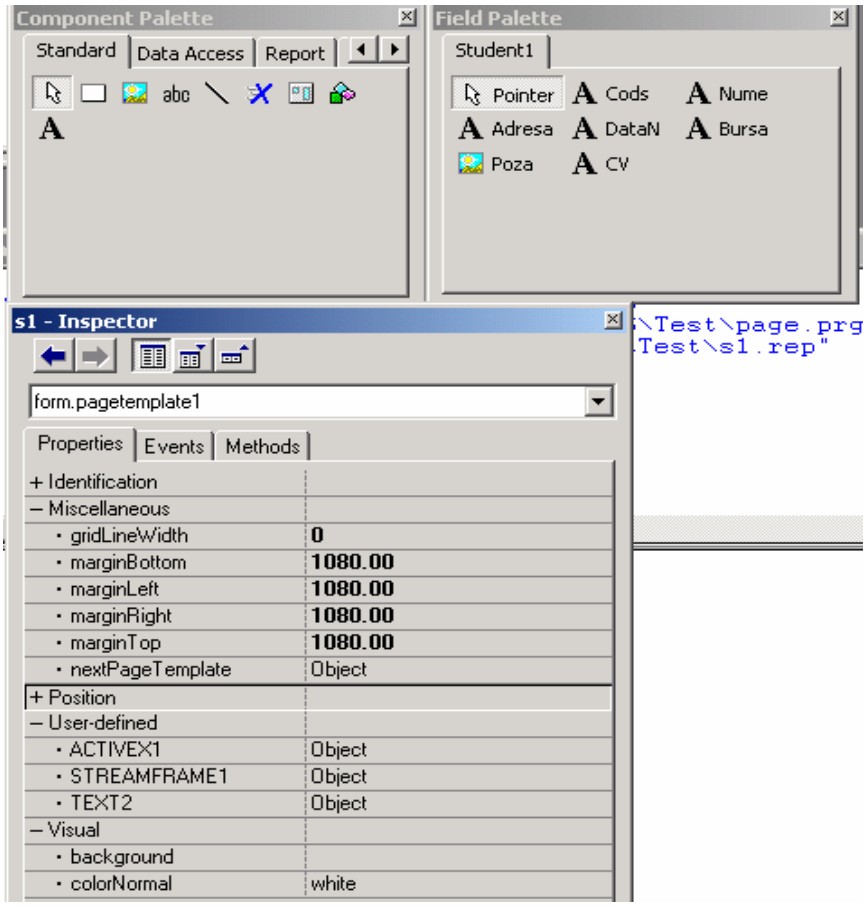


Formularul are 3 zone principale:

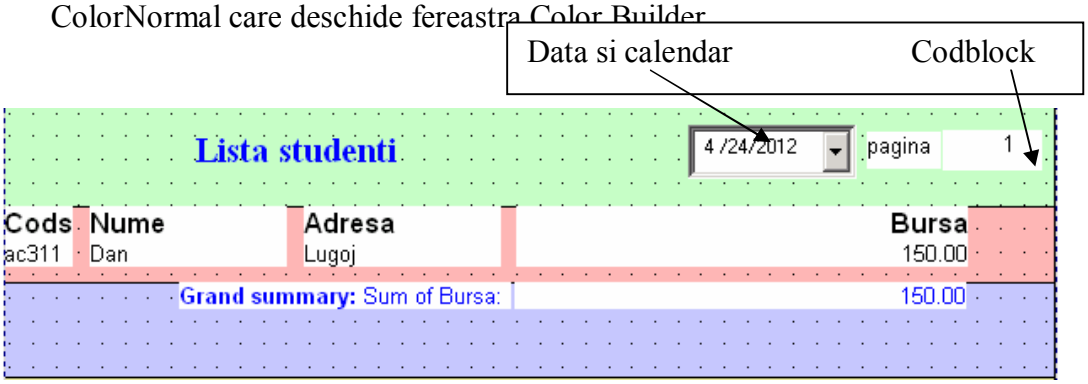
- **Page header** care cuprinde antetul paginii cu titlu si capul de tabel
- **Details** care va conține înregistrările din fișierul utilizat
- **Page footer** care specifică informațiile din partea de jos a paginii
- **Zone pentru sume** parțiale sau globale pentru anumite câmpuri (ex.Bursa)

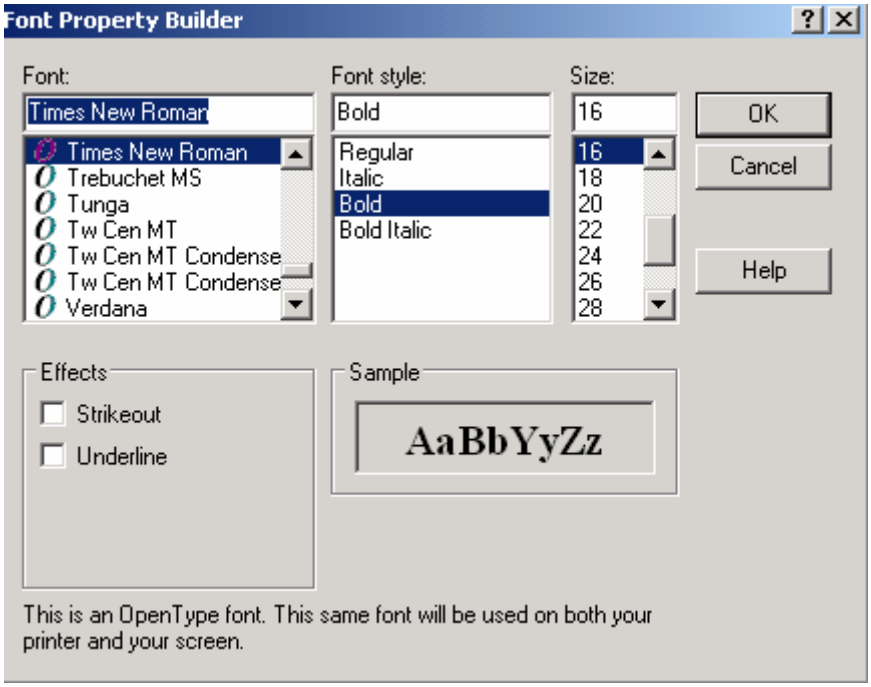
Completarea zonei de antet cu un titlu al raportului se poate face prin:

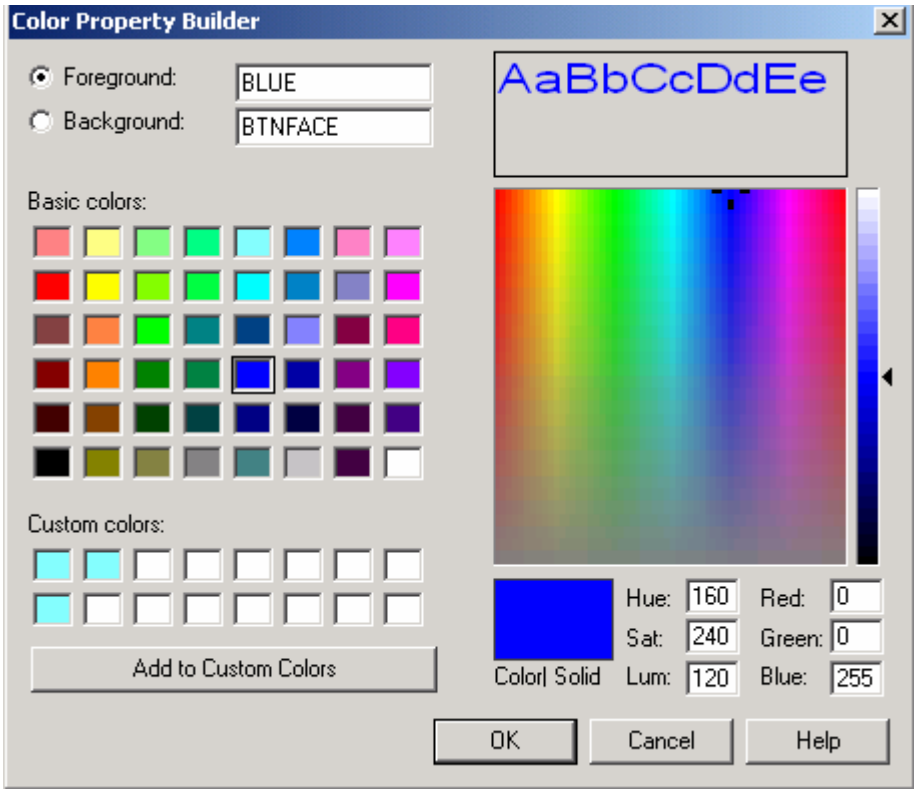
- Se deschid ferestrele de componente, de campuri şi inspector prin Click dreapta
- Se selectează obiectul text **A** (sau TextLabel **abc**) din paleta de componente şi se plasează pe form stabilind dimensiunile cu mouse-ul.
- Se completează apoi atributul Text în fereastra Inspector



- Se stabilesc prin atributul FontName tipul şi dimensiunea font-ului folosit prin fereastra Font Builder.
- Culoarea textului şi a fondului se stabileşte prin activarea proprietății ColorNormal care deschide fereastra Color Builder.

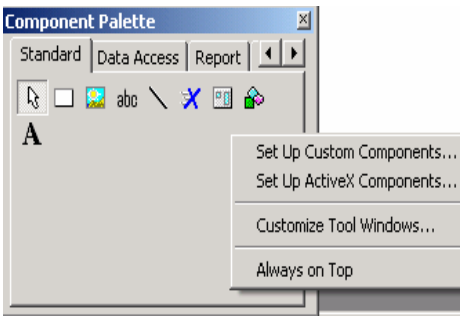


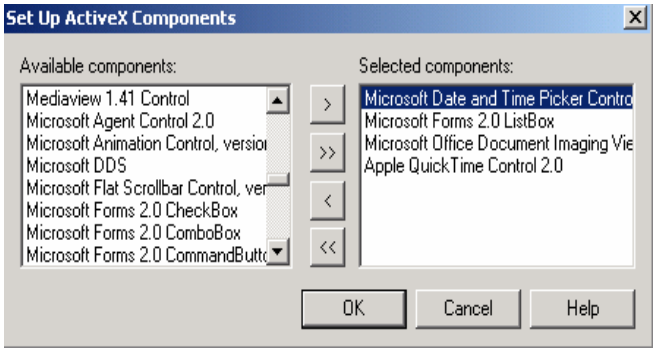




Numărul paginii se obţine printr-un obiect text în care atributul Text se setează tip Codblock, iar prin activarea uneltei se deschide o fereastra in care se poate scrie o secvenţă de comenzi. Comenzile se pot completa şi direct scriind în acest caz
{|| _pageno}. Mai multe comenzi se separă prin ; .Pentru dată se foloseşte funcţia Date().

Funcţia Data şi Calendar se poate obţine prin **obiectele ActivX,** care pot fi setate prin click dreapta pentru a fi plasate pe fereastra de componente de unde pot fi selectate şi plasate pe form:





La activarea raportului data va fi afişată şi dacă se doreşte calendarul:

Se pot insera imagini în antet prin obiectul imagine , pentru care se va seta fişierul sursă.

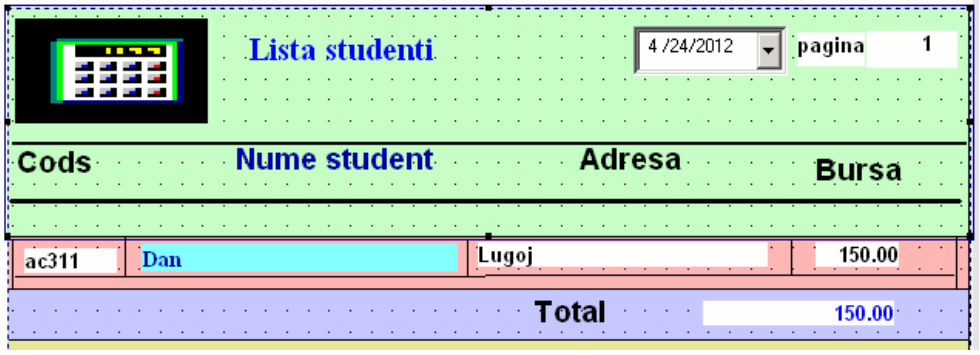
Completarea zonei detaliu

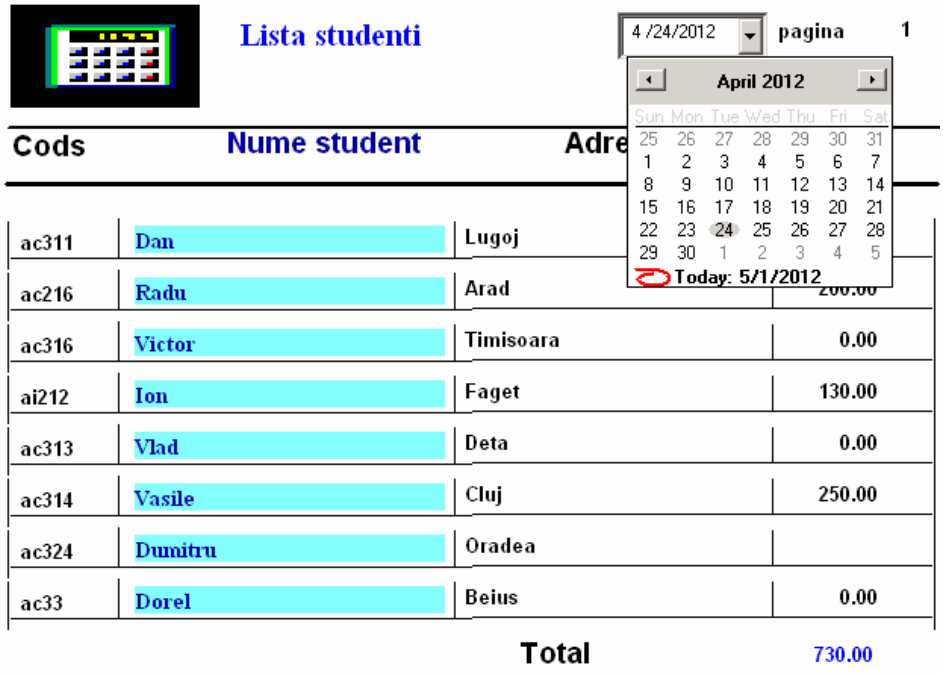
Zona detaliu se completează pentru formatul unui rând şi se aplică pentru toate înregistrările din fişierul asociat raportului. Zona conţine obiecte Text pentru capul de tabel care pot fi setate prin selectare proprietăţi din fereastra Inspector şi plasate unde se doreşte.
Capul de tabel poate fi delimitat cu linie, la fel ca şi fiecare linie de înregistrare folosind obiectul linie din fereastra de componente indicat prin / . Se trage linia cu mouse-ul şi apoi se selecteză şi setează proprietăţile din inspector (ColorNormal, Width,...).

Capul de tabel se recomandă să fie plasat în zona antet format din obiecte Text. Se vor şterge textele ataşate automat la EntryField-urile din zona detaliu.

Câmpurile din înregistrări vor selectate, modificate ca dimensiune şi vor fi setate ca şi textele folosind Inspector (poziţie, aliniere, font, culoare cerneală şi

fond, dimensiune). Ele sunt obiecte EntryField şi se pot lua din paleta de câmpuri.
Pentru a sublinia fiecare rând se va plasa o linie după câmpurile înregistrării, iar pentru separarea coloanelor se va plasa câte o linie verticală după fiecare câmp ca în exemplu, obținând forma raportului ca în figură.
Raportul afişat poate fi tipărit cu File/Print la imprimantă.





Afişarea unui raport se poate face şi dintr-un program prin:

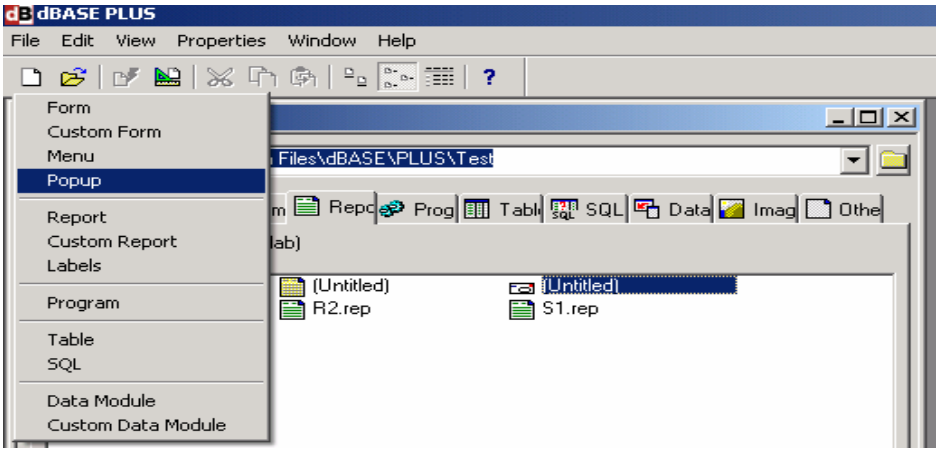
```
DO S1.rep          // unde S1 este numele raportului proiectat
```

10.2. Utilizare obiecte Label (Etichete)

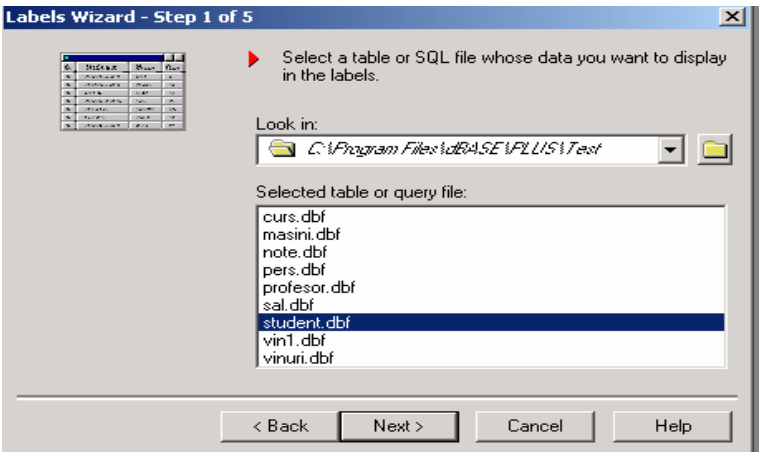
Obiectele Label folosesc informaţiile dintr-un fişier de date şi pot fi utilizate la:

- tipărire unor cărţi de vizită pentru toate persoanele din fişier.
- tipărirea de etichete pentru diverse produse dintr-un fişier.
- tipărirea unor invitaţii pentru un număr mare de persoane
- Etichete pentru CD, DVD, Casete
- Etichete cu adresa de corespondenţă

Toate etichetele au acelaşi format care se stabileşte în regimul Design activat prin File/New/Labels sau prin opţiunea Untitled din categoria Reports.

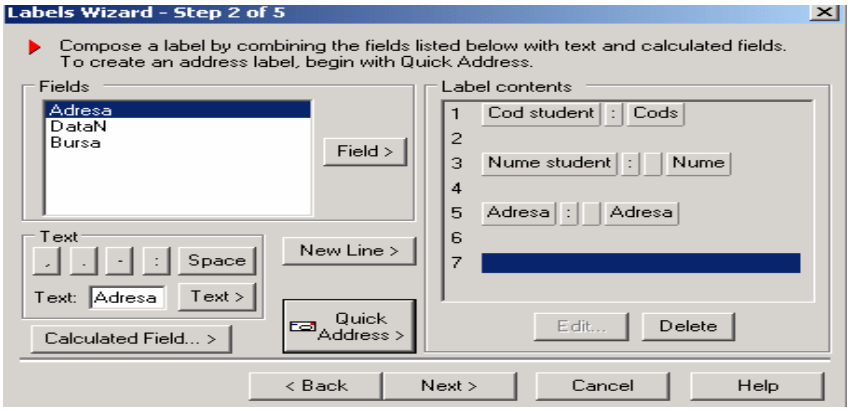


Primul pas stabileşte fişierul utilizat:



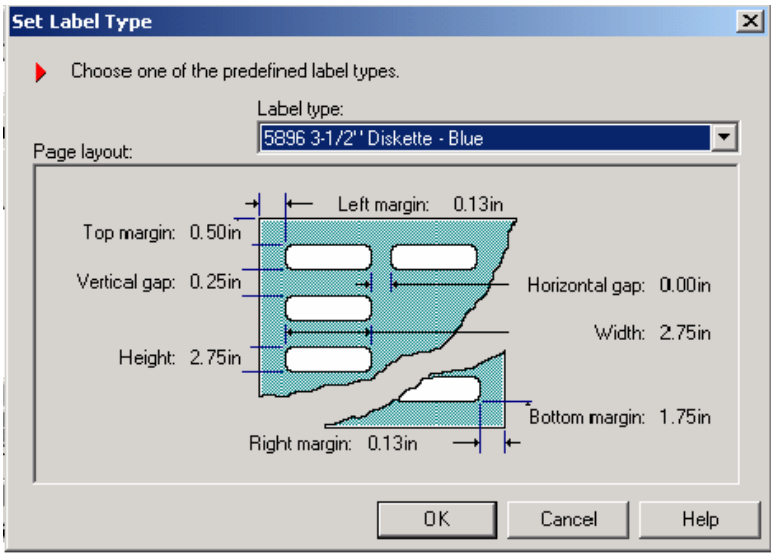
Proiectarea unei etichete se face într-un format ales completând rândurile machetei folosind:

- câmpurile selectate din fişierul ales combinate cu
- texte explicative,
- caractere speciale NewLine, Spaţiu, virgulă, punct, liniuţă, două puncte
- câmpuri calculate cu o formulă dată (impozit=salar*0.16)

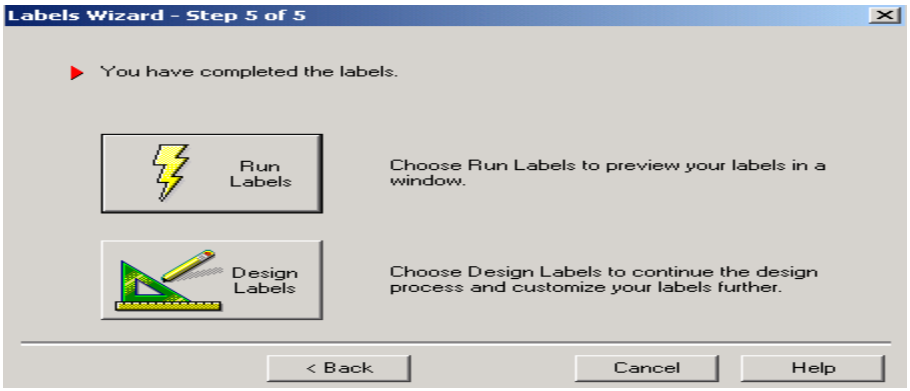


În pasul următor se alege dintr-o listă formatul paginii şi modul de plasare a etichetelor pe pagină. În figura afişată după selecţia tipului de machetă se dau toate dimensiunile care definesc eticheta şi plasarea etichetelor pe pagină.

Am ales o eticheta pentru dischete cu 3 coloane pe pagina:



În pasul următor se poate trece la faza Run Label sau la Design Label, caz în care fiecare element din etichetă poate fi finisat prin prin modificarea atributelor de tip Text (font name, font size, color, border,..)



CODS: ac316
Nume student: Victor
Adresa: Timisoara
Bursa: 0.00 Data nasterii: 14/12/1994

CODS: ai212
Nume student: Ion
Adresa: Faget
Bursa: 130.00 Data nasterii: 17/02/1995

CODS: ac313
Nume student: Vlad
Adresa: Deta
Bursa: 0.00 Data nasterii: 19/03/1996

CODS: ac314
Nume student: Vasile
Adresa: Cluj
Bursa: 250.00 Data nasterii: 12/05/1997

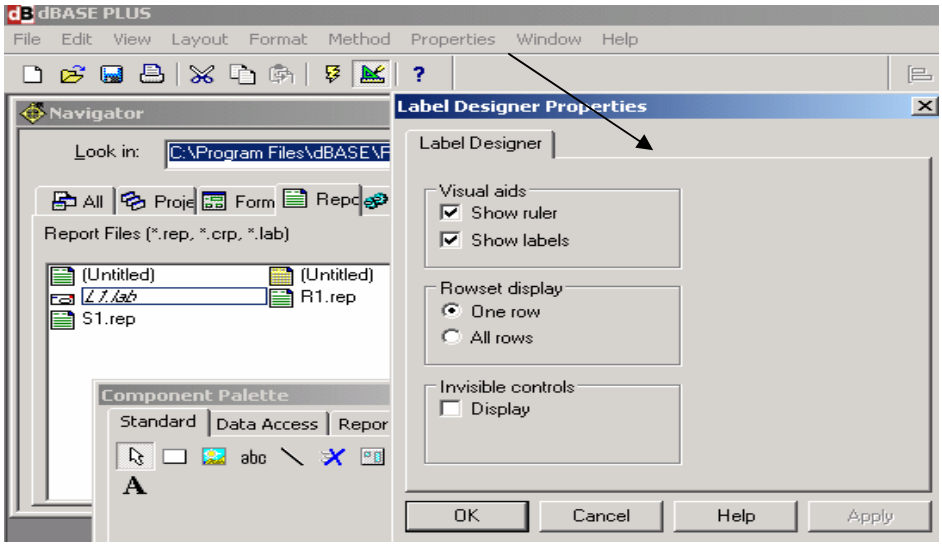
CODS: ac324
Nume student: Dumitru
Adresa: Oradea
Bursa: Data nasterii: 25/09/1999

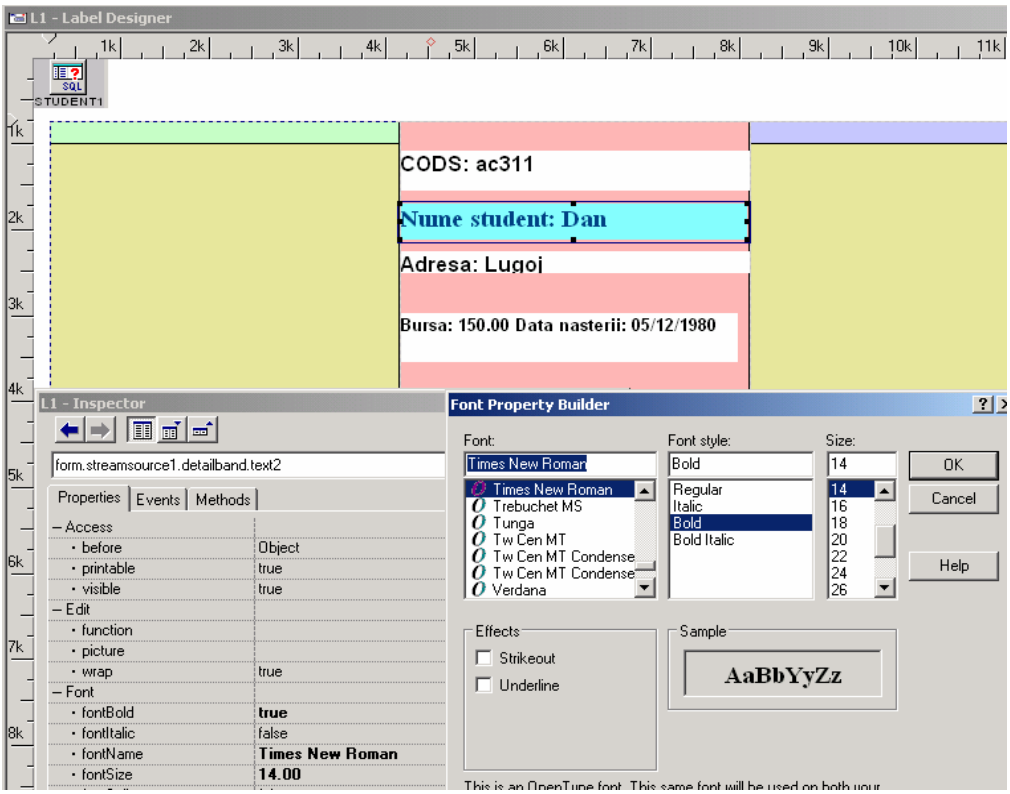
CODS: ac33
Nume student: Dorel
Adresa: Beius
Bursa: 0.00 Data nasterii:

Pentru a finisa etichetele vom deschide în mod Design programul L1.lab cu click dreapta. Vor apare cele 3 coloane de etichete şi vom lucra pe una singură dacă selectăm din meniul Properties-Label designer Properties optiunea **One row**.

Fiecare text din eticheta va fi dimensionat şi pozitionat.
Se va seta apoi din Inspector FontName şi se va stabili tipul şi dimensiunea fontului.

Folosind proprietatea ColorNormal vom stabili culoarea scrisului şi a fondului.





Rularea programului rezultat L1.lab se poate face cu dublu click sau F2 iar din program:

DO L1.lab // lansare program

CODS: ac316

Nume student: Victor

Adresa: Timisoara

Bursa: 0.00 Data nasterii: 14/12/1994

CODS: ai212

Nume student: Ion

Adresa: Faget

Bursa: 130.00 Data nasterii: 17/02/1995

CODS: ac313

Nume student: Vlad

Adresa: Deta

Bursa: 0.00 Data nasterii: 19/03/1996

CODS: ac314

Nume student: Vasile

Adresa: Cluj

Bursa: 250.00 Data nasterii: 12/05/1997

CODS: ac324

Nume student: Dumitru

Adresa: Oradea

Bursa: Data nasterii: 25/09/1999

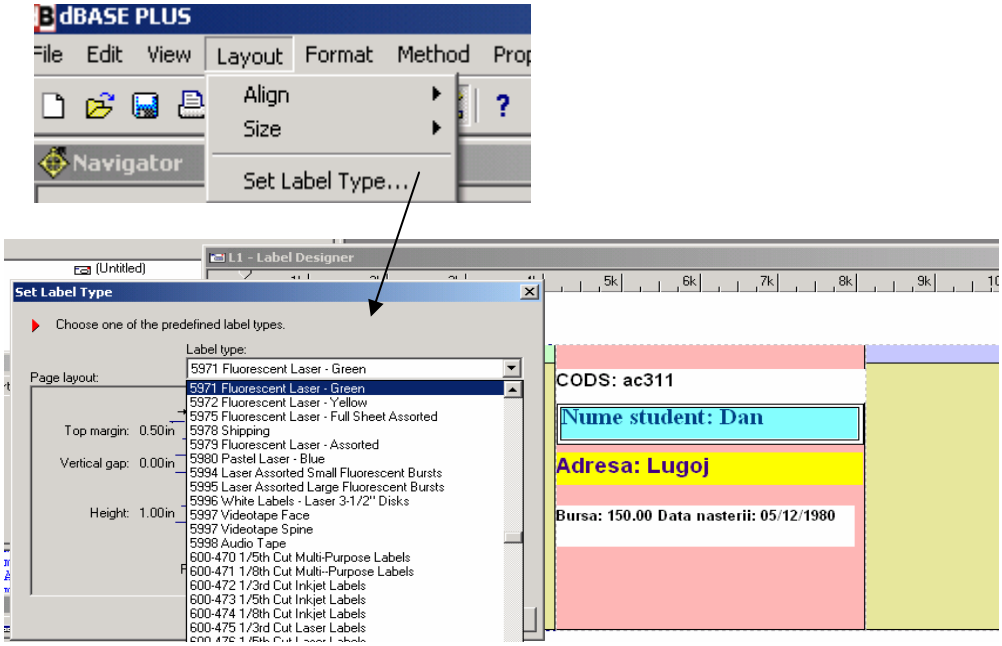
CODS: ac33

Nume student: Dorel

Adresa: Beius

Bursa: 0.00 Data nasterii:

Dacă formatul ales nu e potrivit pentru tipărire se poate schimba modelul prin meniul Layout/Set Label Type:



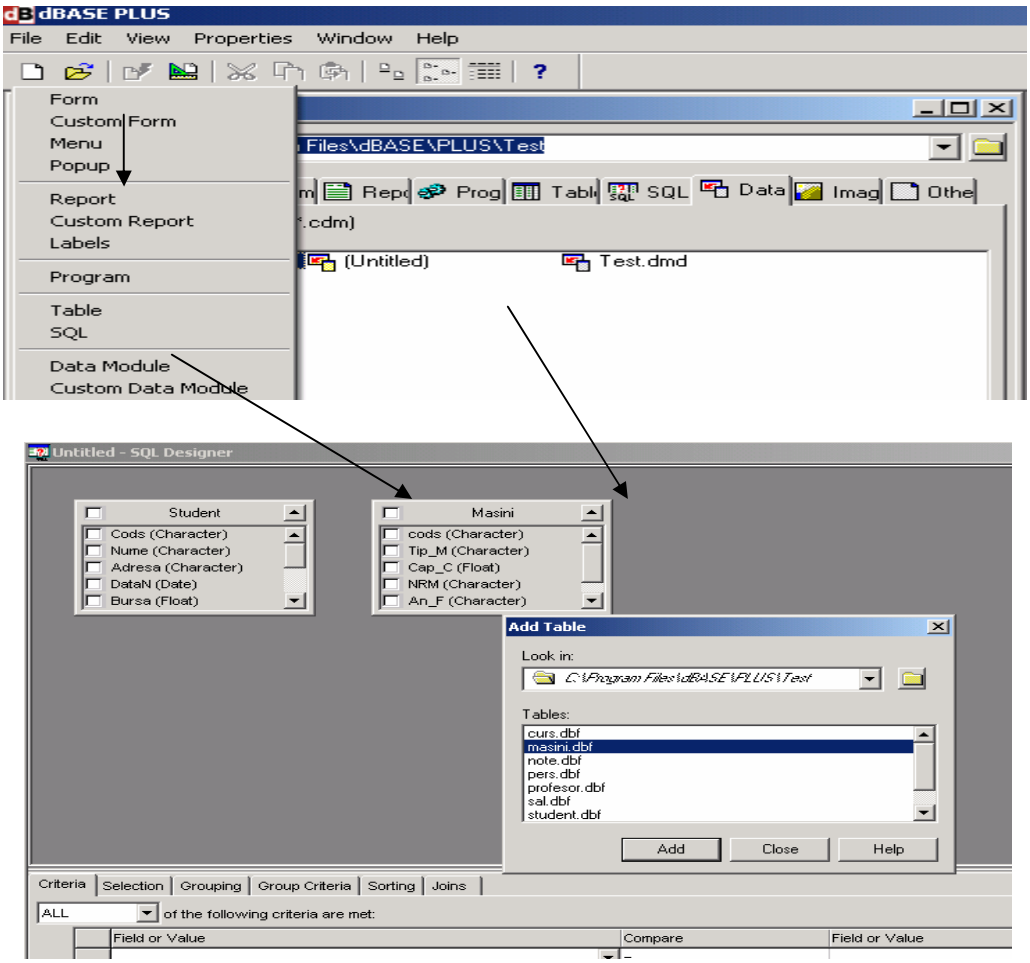
10.3. Definirea de interogari – Query

Interogările sunt programe (obiecte) care permit definirea unor vederi (View în SQL) mai complexe a bazei de date, care **se definesc în SQL utilizând operația de Join,** care combină mai multe tabele pe baza relațiilor dintre ele.

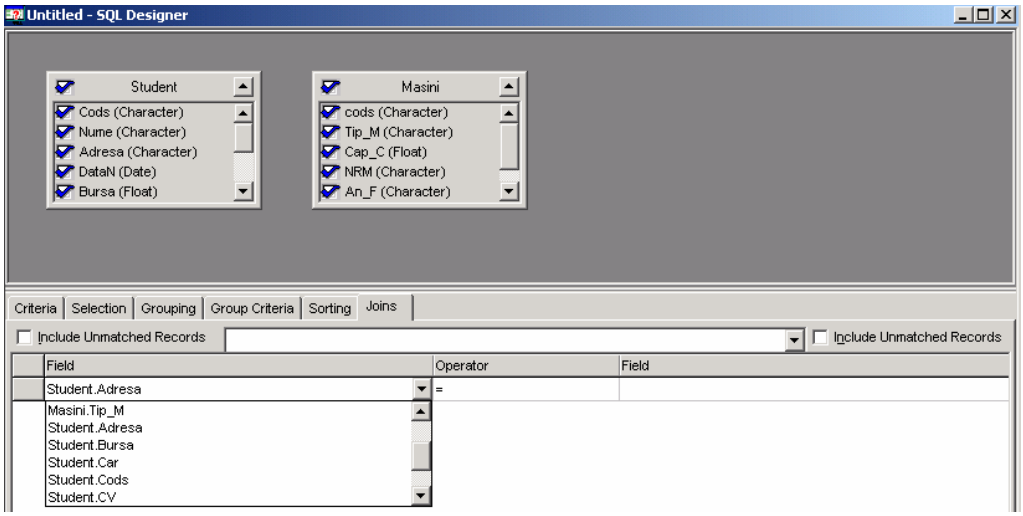
Orice interogare poate fi selectată ca şi o tabelă şi se pot folosi toate câmpurile definite în proiectarea de Form-uri, Rapoarte, Etichete.

Proiectarea unei interogări se poate face prin crearea unui obiect SQL, care permite tabelelor din directorul curent sau din baze de date externe pe servere Oracle, Ms SqlServer, DB2 – IBM, SYBASE, pentru care dBase poate fi folosit ca şi client.

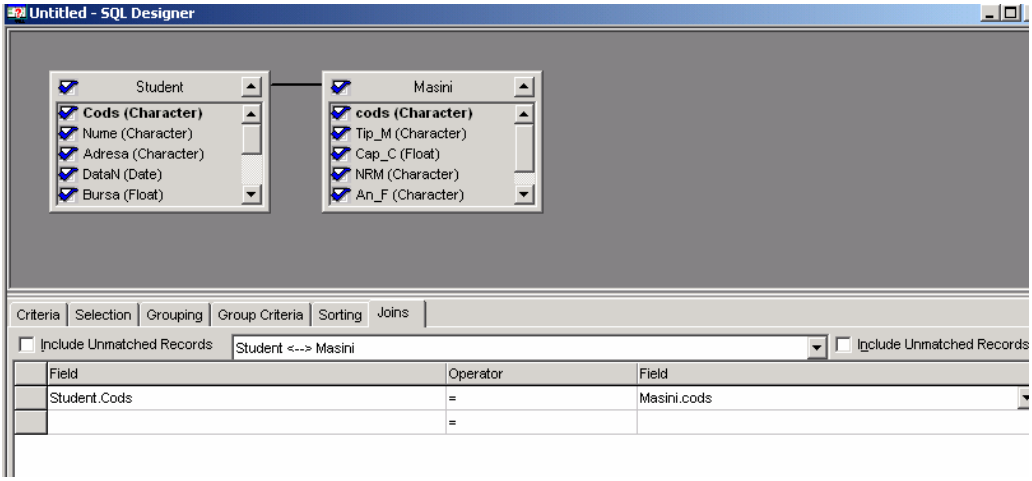
Se va selecta grupa SQL şi se va folosi un Designer:



Va apare fereastra de proiectare **SQL Designer** unde se vor alege cu optiunea ADD tabelele Student si Masini ce vor fi folosite din lista de tabele existente în directorul curent selectat Test. Dacă s-au ales tabelele se dă Close.
Se selectează apoi câmpurile folosite în intorogare prin bifare în structura tabelelor ce sunt afişate.
Se selectează opţiunea Join din operaţiile afişate şi câmpul Student.cods din fişierul părinte Student.



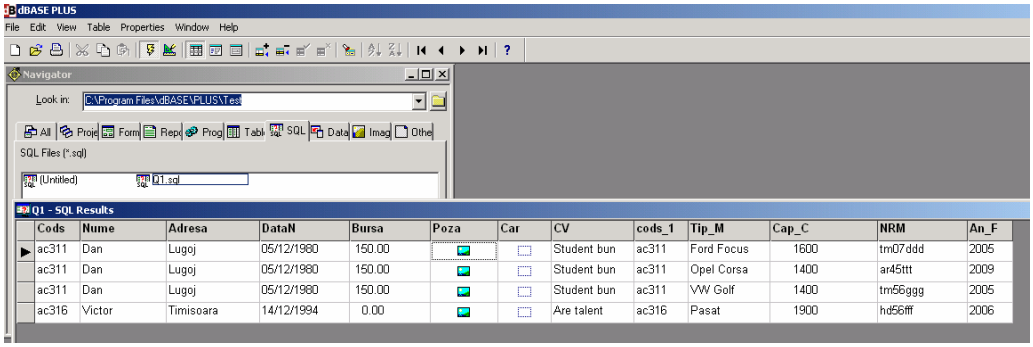
La fel se va selecta câmpul referit Masini.cods din fişierul fiu Masini. In figură se vede că s-a marcat legătura Student-Masini prin Join de egalitate între cpurile de legătură.



Se memorează interogarea cu numele Q1.SQL.

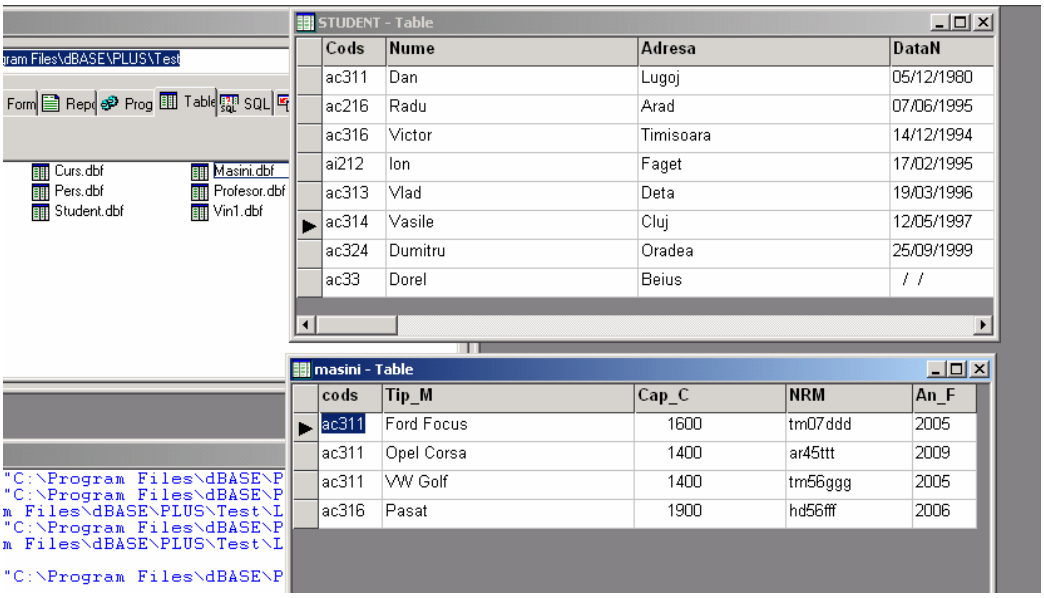
Interogarea Q1 se comportă ca o tabelă compusă, având toate câmpurile selectate în cele 2 tabele, pentru înregistrările care corespund relației. Pentru fiecare student se va trece datele maşinii.

Rezultatul se afişează ca Browse dacă se dă Dclick pe obiectul Q1.sql



Cods	Nume	Adresa	DataN	Bursa	Poza	Car	CV	cods_1	Tip_M	Cap_C	NRM	An_F
ac311	Dan	Lugoj	05/12/1980	150.00			Student bun	ac311	Ford Focus	1600	tm07ddd	2005
ac311	Dan	Lugoj	05/12/1980	150.00			Student bun	ac311	Opel Corsa	1400	ar45ttt	2009
ac311	Dan	Lugoj	05/12/1980	150.00			Student bun	ac311	VW Golf	1400	tm56ggg	2005
ac316	Victor	Timisoara	14/12/1994	0.00			Are talent	ac316	Pasat	1900	hd56fff	2006

Se observă că nu se afişează studenţii care nu au maşină.



Cods	Nume	Adresa	DataN
ac311	Dan	Lugoj	05/12/1980
ac216	Radu	Arad	07/06/1995
ac316	Victor	Timisoara	14/12/1994
ai212	Ion	Faget	17/02/1995
ac313	Vlad	Deta	19/03/1996
ac314	Vasile	Cluj	12/05/1997
ac324	Dumitru	Oradea	25/09/1999
ac33	Dorel	Beius	/ /

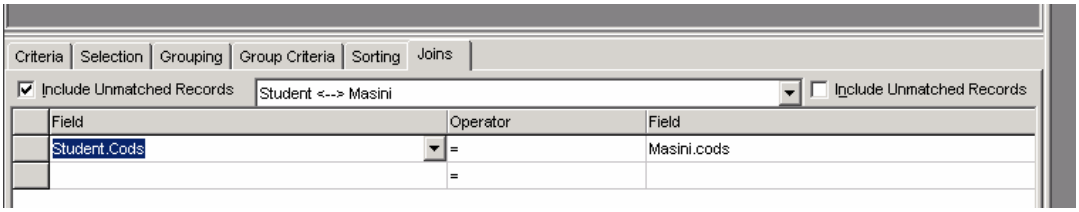
cods	Tip_M	Cap_C	NRM	An_F
ac311	Ford Focus	1600	tm07ddd	2005
ac311	Opel Corsa	1400	ar45ttt	2009
ac311	VW Golf	1400	tm56ggg	2005
ac316	Pasat	1900	hd56fff	2006

Programul sursă de interogare Q1.sql se poate afişa şi este un Join SQL

Q1.sql

```
SELECT Student.Cods, Student.Nume, Student.Adresa, Student.DataN,
Student.Bursa, Student.Poza, Student.Car, Student.CV, Masini.cods,
Masini.Tip_M, Masini.Cap_C, Masini.NRM, Masini.An_F
FROM "C:\Program Files\dBASE\PLUS\Test\student.dbf" Student
   INNER JOIN "C:\Program Files\dBASE\PLUS\Test\masini.dbf" Masini
   ON  (Student.Cods = Masini.cods)
```

Dacă se bifează Incude Unmatched Record in fereastra de design se obține Left Outer Join, care afişează toate înregistrările din tabela din stânga.



Q1 - SQL Results

Cods	Nume	Adresa	DataN	Bursa	Poza	Car	CV	cods_1	Tip_M	Cap_C	NRM	An_F
ac216	Radu	Arad	07/06/1995	200.00			Are simt artistic					
ac311	Dan	Lugoj	05/12/1980	150.00			Student bun	ac311	VW Golf	1400	tm56ggg	2005
ac311	Dan	Lugoj	05/12/1980	150.00			Student bun	ac311	Ford Focus	1600	tm07ddd	2005
ac311	Dan	Lugoj	05/12/1980	150.00			Student bun	ac311	Opel Corsa	1400	ar45ttt	2009
ac313	Vlad	Deta	19/03/1996	0.00			Are spirit practic					
ac314	Vasile	Cluj	12/05/1997	250.00			E sportiv					
ac316	Victor	Timisoara	14/12/1994	0.00			Are talent	ac316	Pasat	1900	hd56fff	2006
ac324	Dumitru	Oradea	25/09/1999	.			Student neglijent					
ac33	Dorel	Beius	/ /	0.00			student de nota 8					
ai212	Ion	Faget	17/02/1995	130.00			E fotbalist					

Programul rezultat este:

Q1.sql

```
SELECT Student.Cods, Student.Nume, Student.Adresa, Student.DataN,
 Student.Bursa, Student.Poza, Student.Car, Student.CV, Masini.cods,
 Masini.Tip_M, Masini.Cap_C, Masini.NRM, Masini.An_F
FROM "C:\Program Files\dBASE\PLUS\Test\student.dbf" Student
   LEFT OUTER JOIN "C:\Program Files\dBASE\PLUS\Test\masini.dbf" Masini
   ON  (Student.Cods = Masini.cods)
```

11. BAZE DE DATE RELAȚIONALE

11.1. Elemente de Algebră Relațională

Principiile algebrei relaționale au fost stabilite de E.F. Codd în 1970, pentru a asigura o fundamentare matematică a bazelor de date relaționale. În concepția relațională o bază de date este formată dintr-o **colecție de relații** (tabele, fișiere de date) asupra cărora se aplică **o colecție de operatori** pentru a gestiona datele conținute în tabele.

Un operator relațional se aplică asupra unor tabele şi va avea ca rezultat tot o tabelă. In algebra relațională nu este permis accesul direct asupra înregistrărilor dintr-o tabelă, sau poziționarea pe o anumită înregistrare.

Se definesc mai jos principalii termeni folosiți în algebra relațională.

Constituant (câmp, atribut,coloana, caracteristică) este informația elementară (atomică) a unei relații. Notăm constituanții cu **X1, X2, X3, ….Xn**
Exemplu : CodP, Nume, Adresa, Salar, Data_nasterii

Domeniul (tipul) este ansamblul valorilor pe care le poate lua un constituant şi se notează **dom(Xi).** Domeniul este un set de valori atomice şi poate fi definit ca un **tip abstract**.
Mai mulți constituanți pot avea acelaşi domeniu : Tel_acasa, Tel_serv, Fax

N-upletul de constituanți este un ansamblu de constituanți **(X_1, X_2, …, X_N)** ce se poate nota cu **X** şi se poate considera un constituant compus.

N-upletul (de date) este un set de valori **(a_1, a_2, …, a_n)** unde **$a_i \in$ dom(X_i).**
N-upletul este un articol dintr-un fişier (record), sau un rând dintr-un tabel (row).

O **relație N-ară R(X)** se defineşte prin trei elemente:

- precizarea unui N-uplet de constituanți **(X_1, X_2, …, X_N);**
- definirea domeniului pentru fiecare constituant X_i;
- definirea unui **predicat** logic care pentru orice N-uplet de date (a_1, a_2,

…, a_n) unde $a_i \in dom(X_i)$ pentru i=1..n dă o propoziție adevărată sau falsă.

Notăm această relație **R(X1,X2,X3,….Xn)** sau **R(X).**

Relația **R(X)** este formată din ansamblul N-upleților pentru care predicatul dă propoziții adevărate.

Relația STUD(Cods, Nume, Adresa, Data_n, Bursa) va cuprinde toți N-upleții (înregistrările) pentru care condițiile (predicatul) de student sunt îndeplinite (admis, neexmatriculat, netransferat, nu a absolvit,etc.). Nu orice combinație de valori, care aparține domeniilor constituanților ne dă un n-uplet valid.

Notăm cu **//R(a_1,a_2,a_3,...a_n) //** o evaluare a predicatului pentru un n-uplet.

Dacă n-upletul aparține relației atunci:

//R(a_1,a_2,a_3,...a_n) // = DA

O relație este o tabelă de valori unde fiecare rând reprezintă un set al valorilor datelor din relație. Valorile pot fi interpretate ca descriind **o entitate (**Persoana, Student, Masina**)** sau **o legătură dată** (Note, Orar, Rută). Numele tabelei şi coloanelor servesc la interpretarea sensului valorilor din fiecare rând al tabelei.

Gradul unei relații este dat de numărul atributelor ce formează relația.

Modelul (schema) relației R este notat **R(X1,X2,X3,….Xn)** şi este un un set de atribute (constituanți) R= **{X1,X2,X3,….Xn}.** Un atribut Xi este numele care indică un anumit domeniu $D_i = dom(X_i)$ într-un model de relație R.

Modelul de relație:

STUD(Cods,Nume, Adresa, Data_n, Bursa)

este abstract şi se referă la mai multe relații concrete, care cuprind studenții unor universități diferite.

O relație concretă (instance) r a modelului de relație **R(X1,X2,X3,….Xn)** notată **r(R)** este un set de n-upleți r = {t_1,t_2,t_3,.....t_m), unde fiecare n-uplet este o listă ordonată de n valori:

t = (v_1,v_2,v_3,....v_n) unde fiecare valoare $v_i \in dom(X_i)$ pentru i = 1...n sau v_i=nul.

O relație r reflectă numai n-upleții valizi, care reprezintă o stare a lumii reale, ce se poate modifica în timp. Modelul relației R rămâne stabil până la modificarea atributelor sau predicatului.

Principalele caracteristici ale unei relații sunt:

- **N-upleții din relație nu sunt ordonați** ca elementele unei mulțimi matematice. Relația se memorează ca un fişier unde unui n-uplet îi corespunde o înregistrare cu un număr de ordine, rezultat arbitrar în momentul creierii. O relație ca tabel poate fi afişată în orice ordine a rândurilor. Relația poate fi ordonată logic după orice atribut.
- **Ordinea valorilor în N-upleți** este dată de ordinea definirii atributelor în modelul relației. Toți n-upleții unui fişier (tabelă) vor avea aceeaşi ordine a valorilor. Ordinea valorilor din n-uplet se precizează într-o tabelă prin ordinea numelor atributelor dată la descriere.
- **Valorile atributelor** din N-upleți sunt atomice. Un atribut nu poate avea valori multiple. Sunt permise valorile nule care pot fi necunoscute (nu ştim numărul de telefon al unei persoane) sau inexistente (persoana nu are telefon).
- O relație poate fi privită ca o specificare a unui tip compus. Definiția tipului este dată de structura relației. N-upleții relației sunt realizări ale entității sau legăturii.

Atribute cheie în relație

O relație este definită ca un set de N-upleți (câmpuri) distincți (din acest punct de vedere corespunde tipului algebric mulțime). Nu pot exista doi n-upleți care au toate valorile atributelor identice. Uzual există grupe de atribute pentru care valorile diferă şi nu există două valori identice în n-upleți diferiți.

Supercheia (SK) este un grup de atribute care identifică în mod unic N-upleții relației.
$\mathbf{t_i(SK) \# t_j(SK)}$ pentru orice i,j unde i # j
Ex: Nume, Data_N, Adresa
Pot exista relații care au o singură supercheie formată din toate atributele.

Cheie (K) într-o relației **R** este o **supercheie minimă**, care are proprietatea că, înlocuind sau ştergând orice atribut A din K cu A' obținem un set de atribute K', care nu mai sunt supercheie în relația R.. Mulțimea cheilor unei relații formează cheile candidat din care trebuie aleasă o cheie primară.

Cheia primară (**PK – primary key**) este o cheie aleasă de administratorul bazei de date pentru a identifica înregistrările. Se alege o cheie cu un număr minim de atribute, dacă este posibil chiar un singur atribut. Se

recomandă ca PK să aibă şi un sens funcţional (Cods, CNP=cod numeric personal). În modelul bazei de date cheile primare se subliniază. Se defineşte ca **atribut prim** orice atribut care face parte din cheia primară.

Cheie externă (**FK – foreign key**) este un grup de atribute care este cheie primară într-o altă relaţie şi serveşte la realizarea unor legături între înregistrările celor două relaţii.

Definiţie: Un set de atribute FK din relaţia R1 este cheie externă în R1 dacă satisface condiţiile:

- Atributele din FK au acelaşi domeniu cu atributele cheii primare din relaţia R2. Cheia FK referă relaţia R2.
- O valoare a lui FK într-un n-uplet $\mathbf{t_i}$ din R1 să găsească o valoare a cheii PK pentru un anumit n-uplet $\mathbf{t_j}$ din R2 sau FK este nul.

$\mathbf{t_i(FK) = t_j(PK)}$ pentru orice i = 1..n

O cheie externă poate referi chiar şi propria relaţie (cod_şef, cod_tata, cod_mama). La adăugarea înregistrărilor trebuie verificate valorile cheilor externe în relaţia referită, pentru a asigura coerenţa bazei de date (**integritatea referinţei**).

Operaţiile algebrei relaţionale

Algebra relaţională cuprinde o colecţie de operatori, care sunt aplicaţi pe relaţii întregi obţinând o Relaţie rezultat. Relaţiile rezultate pot fi supuse la noi operaţii. Operaţiile realizează:

- **Selecţia** n-upleţilor dintr-o relaţie pe baza unor condiţii:

DISP FOR Bursa > 0 .AND. Cods = ‘AC4’ - in XBase
SELECT * FROM Stud WHERE Bursa > 0 AND Cods LIKE ‘AC4%’; - in SQL

- **Proiecţia** selectează numai anumite coloane dintr-o relaţie:

LIST **Nume, Adresa, Data_n** - în XBase
SELECT Nume, Adresa, Data_n FROM Stud; - în SQL

- **JOIN (reuniune) c**ombină n-upleţii din două relaţii pentru a răspunde la întrebări în BD:

SELECT ST.Nume, ST.Adresa, NT.Nota FROM Stud ST, Note NT
WHERE ST.Cods = NT.CodS AND Bursa > 0;

Condiția **ST.Bursa > 0** este o **condiție de selecție** care se aplică unei singure tabele, iar condiția **ST.CodS = NT.CodS** este o **condiție de JOIN** între două câmpuri care au acelaşi domeniu şi sunt din două tabele diferite.

- **Operațiile din teoria mulțimilor** se pot realiza între 2 **relații UNION compatibile** (care au acelaşi număr de atribute şi de acelaşi tip)

UNION, INTERSECT, MINUS şi PRODUS CARTEZIAN

Exemplu de selecție a studenților de la facultatea AC, mai puțin cei din anul 1 folosind 2 comenzi de SELECT se realizează prin operația de MINUS între cele 2 mulțimi de înregistrări selectate.

SELECT * FROM Stud WHERE Cods LIKE 'AC%' MINUS
SELECT * FROM Stud WHERE Cods LIKE 'AC1%';

Selecția se notează cu $\sigma_{cond}(R)$ şi se aplică pe o singură relație, verificând fiecare
N-uplet dacă îndeplineşte condiția de selecție şi îl extrage în relația rezultat. Gradul relației rezultate (număr de atribute) este acelaşi cu cel al relației inițiale R pe care se aplică. Numărul de n-upleți rezultați (cardinalitatea) este mai mic, cel mult egal cu cel al relației inițiale R.

- Operația de selecție este comutativă privind ordinea de evaluare a condițiilor:

$$\sigma_{cond1}(\sigma_{cond2}(R)) = \sigma_{cond2}(\sigma_{cond1}(R)$$

- Se pot combina în cascadă condițiile într-un singur SELECT cu operatorul de relație AND.

$$\sigma_{cond1}(\sigma_{cond2}(...(\sigma_{condn}(R))) = \sigma_{cond1\ AND\ cond3\ AND....condn}(R)$$

Proiecția se notează cu $\Pi_{lista\_atribute}(R)$ selectează numai anumite coloane dintr-o relație eliminându-le pe celelalte. Relația rezultat va avea ca atribute pe cele din lista de proiecție din relația R, în ordinea din listă. Gradul

relației rezultat este gradul listei de atribute. Dacă lista nu conține nici un atribut cheie este posibil să obținem n-upleți identici şi **se elimină duplicații**. Dacă se utilizează atribute cheie numărul de elemente este acelaşi cu cel din relația inițială. Proiecția nu este comutativă.

11.2. Normalizarea bazelor de date

Normalizarea unei baze de date constă în principal în descompunerea modelului bazei de date în mai multe relații astfel încât să se reducă la maxim redundanța datelor şi implicit să elimine anomaliile de actualizare. Operația de normalizare se bazează pe dependențele funcționale care există între datele unei aplicații.
Concret, paşii care trebuie făcuți la proiectarea unei baze de date sunt următorii:

- Analiza aplicației: analiza circuitului informațional, studierea intrărilor şi ieşirilor, stabilirea claselor de utilizatori;
- Analiza semanticii atributelor din entități: identificarea atributelor şi a sensului lor funcțional, gruparea atributelor în relații pe entități, stabilirea cheilor primare şi externe;
- Normalizarea relațiilor obținute la punctul anterior: micşorarea redundanței prin gruparea atributelor în relații conform definițiilor pentru formele normale, stabilirea de constrângeri pentru eliminarea anomaliilor de actualizare;
- Scoaterea din relațiile principale a atributelor care au peste 70% valori nule.

Dependențe funcționale (FD)

Reducerea redundanței în baza de date şi eliminarea anomaliilor de actualizare (adăugare, ştergere) se face riguros prin normalizarea BD pe baza analizei dependențelor funcționale. Prin definiție dacă se consideră un model de BD descrisă printr-o singură relație universală $\mathbf{R(A_1,A_2,A_3,...A_n)}$, nu este obligatoriu să fie memorată printr-o singură tabelă.

O dependență funcțională, notată $\mathbf{X \rightarrow Y}$, între două seturi de atribute a unei relații **R**, specifică o constrângere asupra n-upleților posibili. Ea se defineşte în felul următor:

$\forall\ \mathbf{t_1,\ t_2 \in R,\ pentru\ t_1(X) = t_2(X) \Rightarrow t_1(Y) = t_2(Y)}$

Se spune în acest caz că X determină funcțional pe Y sau că Y este dependent funcțional de X. Faptul că X nu determină funcțional pe Z se va nota **X $|\rightarrow$ Z.**
O dependență funcțională FD este o proprietate a sensului semantic al atributelor din relația R. Notăm cu **F** setul dependențelor funcționale existente între atributele relației R. Din acestea se pot deduce prin inferență (deducție) altele.
O dependență funcțională FD care face parte din setul F al relației R, se poate deduce că există pentru orice realizare (instanța) a relație **r$\subset$ R.**
Notăm cu $\mathbf{F}^+$ setul dependențelor funcționale deduse din F.
Notăm cu F |= X$\rightarrow$ Y dependența funcțională dedusă din F.

Reguli de inferență (IR)

Notăm concatenarea atributelor într-o dependență funcțională **XY$\rightarrow$Z** în sensul că atributele X şi Y concatenate determină pe Z. Pentru determinarea dependențelor funcționale se pot aplica următoarele reguli de inferență:

1) Regula reflexivă: $X \supseteq Y \Rightarrow X \rightarrow Y$
2) Regula de mărire: $\{ X \rightarrow Y \} \Rightarrow XZ \rightarrow Y$
3) Regula tranzitivă: $\{ X \rightarrow Y \wedge Y \rightarrow Z \} \Rightarrow X \rightarrow Z$
4) Regula de decompoziție: $\{ X \rightarrow YZ \} \Rightarrow X \rightarrow Y$
5) Regula de reuniune: $\{ X \rightarrow Y \wedge X \rightarrow Z \} \Rightarrow X \rightarrow YZ$
6) Regula pseudotranzitivă: $\{ X \rightarrow Y \wedge WY \rightarrow Z \} \Rightarrow WX \rightarrow Z$

Normalizarea relațiilor

Forma normală urmăreşte **eliminarea redundanței informațiilor** din baza de date. **Normalizarea bazei** de date presupune aducerea relațiilor gradual pe diverse forme normale. Fiecare formă normală preia constrângerile formei anterioare la care adaugă noi condiții. Formele normale impun constrângeri dar nu dau soluții. Structura BD se proiectează de administratorul BD şi proiectantul aplicației pe baza experienței şi se verifică dacă fiecare tabelă respectă cerințele de normalizare.

Forma normală 1 (1NF) cere:

- domeniul atributelor să cuprindă valori atomice (indivizible); se interzic câmpurile compuse sau "relații în relație"
- fiecare atribut din N-uplet trebuie să aibă o singură valoare în domeniu.

Exemple:

Nu se permite ca la o persoană să se introducă mai multe numere de telefon. Se pot adăuga noi câmpuri ca Tel_mobil, Tel_serv, Tel_acasa, Fax.

Dacă o persoană poate avea mai multe maşini se va schimba structura BD, eliminând câmpul Nrm (număr masină) din tabela **Pers** şi se va introduce CodP (cod proprietar) în tabela **Maşini**. Pentru câmpul Codp se poate crea un fişier index prin care se vor găsi toate maşinile ce aparţin unei persoane

Forma normală 2 (2NF) cere ca relaţia să fie în 1NF şi se bazează pe dependenţa funcţională completă (full) faţă de cheia primară.

Y este complet dependent funcţional de X, X→Y dacă prin eliminarea unui atribut oarecare A din X **(X –A) |→ Y** nu mai determină funcţional pe Y.

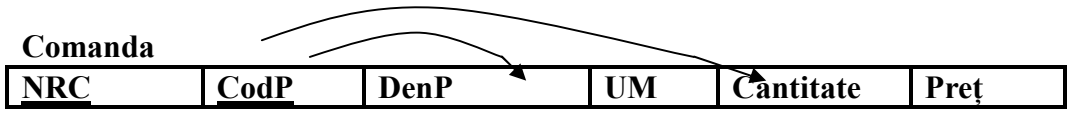
O relaţie R este în 2NF dacă orice atribut neprim (care nu face parte din cheia primară) este complet dependent funcţional de cheia primară.

Variantă: Se cere să nu existe atribute care să depindă numai de o parte a cheii primare.

Dacă nu este îndeplinită condiţia se vor plasa acele atribute în altă relaţie la care se va adăuga atributul din cheia primară care le determină.

Exemplu. Se consideră o tabelă de comenzi care are Cheia primară NRC +CodP. Câmpul CodP determină DenP şi UM (unitate de măsură). Deci cele două câmpuri depind numai de o parte a cheii primare şi tabela nu este în forma normală 2.

Comanda



NRC	**CodP**	**DenP**	**UM**	**Cantitate**	**Preţ**
Nr.comanda	Cod produs	Denumire produs			

Pentru a ajunge în 2NF se vor elimina câmpurile DenP şi UM din tabela **Comada**, care vor fi plasate într-o altă tabelă **Produse** în care cheia primară va fi câmpul CodP.

Comanda

NRC	**CodP**	**Cantitate**	**Pret**

Produse

Codp	**DenP**	**UM**

Forma normală 3 (3NF) este bazat pe conceptul de dependență tranzitivă.
O dependență X→Y din R este tranzitivă, dacă există un set de atribute Z care nu este cheie în R şi există **X→Z şi Z→Y**

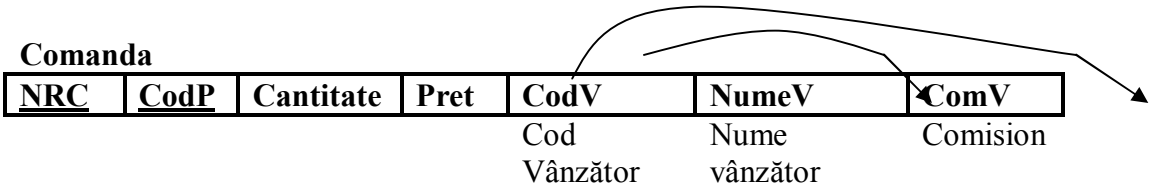
O relație R este în 3NF dacă este în 2NF şi nu există nici un atribut neprim din R care să fie dependent tranzitiv de cheia primară.

Variantă: nu se permit atribute ale relației care nu fac parte din cheia primară şi care determină alte atribute (fac excepție cheile candidat). Atributele care determină alte atribute se numesc **determinanți.**
Definiție generalizată pentru forma 3NF. Orice atribut al relației îndeplineşte:

- este complet dependent funcțional de orice cheie din R;
- este dependent netranzitiv de orice cheie din R.

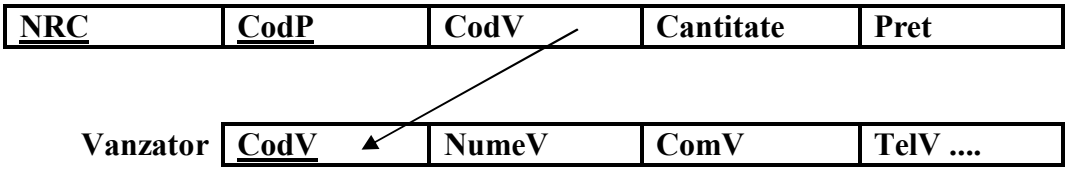
Exemplu: Se consideră o tabelă **Comanda** în care câmpul **CodV** nu face parte din cheia primară şi determină câmpurile NumeV şi ComV (comision)

Comanda



Pentru a aduce relația în 3NF se vor elimina câmpurile NumeV şi ComV din tabela **Comanda** şi se va crea o altă tabelă **Vanzator**, care va avea cheia primară CodV. Se observa că pentru a ajunge la forma **3NF** dintr-o tabelă Comanda s-a ajuns la 3 tabele. Prin normalizare se fragmentează BD dar se elimina redundanța şi anomaliile de actualizare.

Comanda



BCNF – Boyce Codd Normal Form este variantă mai restrictivă. O relație este în BCNF dacă, pentru orice dependență X→Y din R, X este o cheie

candidat a lui R.
Condiția pentru cele 3 forme normale se poate rezuma într-o propoziție:
"Orice câmp dintr-o relație trebuie să depindă de PK, de întreaga PK şi numai de PK."

Forma normală 4 (4NF) elimină anomaliile datorate dependențelor funcționale multivaloare.
Relația R(A,B,C…) conține o dependență multivaloare dacă:
- A nu determină univoc pe B şi C (A $|\rightarrow$ B şi A $|\rightarrow$ C)
- A conduce la valori multiple a lui B (A$\rightarrow\rightarrow$B)
- A conduce la valori multiple a lui C (A$\rightarrow\rightarrow$C)
- B şi C sunt independente între ele.

O relație este în 4NF dacă este în BCNF şi nu are dependențe multivaloare.

Aducerea unei baze de date într-o formă normală superioară presupune extragerea unor atribute din relațiile existente şi crearea pe baza lor a unor noi relații, astfel încât rezultatul să respecte condițiile de normalizare. Acest lucru duce la fragmentarea bazei de date, dar elimină din anomaliile de actualizare şi reduce spațiul pierdut datorită redundanței datelor.

11.3. Modelul relațional al BD

Modelul relațional al BD a fost propus de E. Codd din 1971 şi s-a bucurat de succes teoretic dar nu şi practic. El formalizează matematic prin algebra relațională teoria BD pornind de la câteva principii:

- O BD relationala este formată din **tabele (** corespunzătoare relațiilor din algebra relațională) care cuprind un număr de **coloane** (câmpuri) şi **rânduri** (înregistrări).
- Pentru fiecare tabelă se defineşte o **cheie simbolică** care identifică univoc un rând (t-uplet) şi nu poate fi nulă (integritatea entității).
- Împărțirea BD în tabele se face din considerente semantice şi trebuie să respecte cerințele de normalizare :
    - 1NF- să nu existe decât o valoare pentru fiecare câmp dintr-o înregistrare;
    - 2NF- să nu existe câmpuri care depind numai parțial de cheia primară (PK);
    - 3NF- să nu existe câmpuri, care nu fac parte din cheia primară şi care să determine alt câmp;
    - 4NF- să nu existe dependențe multivaloare în BD.

- Legăturile între tabelele BD se fac prin **chei externe** (FK) cărora trebuie să le corespundă în tabela referită o **cheie primară** (integritatea referinței).
- Asupra tabelelor ce compun BD să se accepte operatorii :
  - relaționali SELECT, PROJECT, JOIN, PRODUS CARTEZIAN;
  - din teoria mulțimilor UNION, INTERSECT, MINUS.
- Să nu se utilizeze pointeri pentru accesul la înregistrări sau legături vizibile pentru utilizator.
- Se acceptă indexarea tabelelor după anumite câmpuri, dar utilizatorul nu poate face referire explicită la index. El servește la optimizarea accesului, făcut automat de sistem.
- Întrebările asupra BD să fie exprimate fără iterații sau recursivitate.
- Redundanța informației este minimă dacă BD este normalizată.
- Modificarea şi dezvoltarea structurii BD este permisă ulterior după creare prin adăugări de tabele sau câmpuri în tabele.

11.4. Limbajul SQL

Limbajul SQL (Structured Query Language) este limbajul neprocedural (declarativ) cel mai utilizat la interogarea bazelor de date relaționale. El respectă cel mai bine specificațiile algebrei relaționale.
SQL cuprinde un set redus de comenzi care însă sunt suficiente pentru:
- crearea, modificarea şi ştergerea elementelor unei baze de date: tabele, vederi şi indecşi
- adăugarea, modificarea şi ştergerea înregistrărilor din tabele
- interogarea bazei de date prin întrebări structurate
- comenzi de securizare, control al accesului concurent şi asigurarea integrității

Toate informațiile referitoare la o bază de date se rețin în tabele sistem: structură de tabele şi vederi, definiții de câmpuri din tabele, fişiere index, informații de acces etc.

Limbajul SQL fiind un limbaj declarativ nu permite implementarea unor algoritmi de prelucrare care să parcurgă înregistrările într-o anumită secvență. Din acest motiv toate implementările practice combină limbajul SQL cu un limbaj procedural (dBase, Oracle PL/SQL, Visual Basic).

Limbajul SQL standard defineşte mai multe tipuri de interogări:

- pentru crearea structurii bazei de date: *CREATE DATABASE, CREATE TABLE, CREATE VIEW, DROP TABLE, DROP VIEW, ALTER TABLE etc.*

- pentru gestionarea bazei de date: *START DATABASE, STOP DATABASE, GRANT, REVOKE PRIVILEGE, SET TRANSACTION, ROLLBACK, COMMIT*
- pentru gestionarea înregistrărilor: *INSERT, DELETE, UPDATE*
- pentru căutări sau interogări de date: *SELECT*.

Implementarea SQL în dBASE Plus

In dBase Plus se implementează două variante de limbaj SQL:

- **Local SQL** care se aplică asupra fişierelor de date existente în dBase şi care sunt văzute ca tabele ale Bazei de date
- **SQL extern** care transmite o comandă de interogare spre un server extern de Baze de Date (Oracle, Microsoft SQL Server, DB2 IBM), care va fi executată pe serverul extern şi se va primi rezultatul ca o tabelă. Conectarea la serverul extern se face prin creearea unui Data Module, care va fi prezentat mai tarziu

Implementarea Local SQL din dBase Plus combină într-o formă simplă un dialect de SQL cu limbajul xBase.

În acest fel se pot mixa comenzi SQL cu comenzi xBase pentru accesul la tabelele locale.

Se recomandă:

- Comenzile xBase pentru crearea tabelelor, inserarea, afişarea, ştergerea şi actualizarea înregistrărilor prin dialog cu utilizatorul folosind interfață grafică
- Comanda SELECT din SQL pentru interogări complexe, care returnează rezultatul
  într-o tabelă ce poate fi afişată folosind comenzi xBase

Comenzile implementate de Local SQL pentru accesul la tabele locale (DBF) sunt următoarele:

CREATE TABLE	- crearea unei noi tabele DBF
DROP TABLE	- ştergerea unei tabele
ALTER TABLE	- permite modificarea câmpurilor unei tabele
CREATE INDEX	- crearea unor indecşi în fişierul multiindex asociat
DROP INDEX	- ştergerea unui index din fişierul multiidex asociat
INSERT INTO	- adăugare de înregistrări într-o tabelă
DELETE FROM	- ştergerea condiționată a unor înregistrări
UPDATE	- modificarea condiționată a datelor dintr-o tabelă

SELECT **- selectarea datelor dintr-o tabelă**

Cea mai utilă comandă, prin prisma avantajelor față de comenzile xBase este comanda ***SELECT***. Ea va fi prezentată mai pe larg în continuare, pentru restul comenzilor putând fi consultată documentația Help a sistemului.

Baza de date pentru local SQL poate fi considerata formată din toate fişiere de date (.dbf) numite tabele şi de indexi create anterior folosind comenzile xBase. Aceste tabele se pot prelucra folosind atât comenzi xBase cât şi comenzi SQL.

Comanda ***SELECT*** din Local SQL permite extragerea datelor din una sau mai multe tabele bazată pe anumite criterii. O comandă SELECT care extrage date din mai mult de o tabelă realizează operația de ***JOIN*** *din* ***algebra relațională***.

Rezultatul unei comenzi SELECT este o nouă tabelă temporară. Comanda SELECT acționează ca şi comanda USE, prin crearea structurii tabelei temporare şi completarea ei cu rezultatul selecției. Aceasta va fi deschisă în prima zonă liberă, care va deveni automat zona curentă. După execuția ei în această zonă vor putea fi folosite toate comenzile xBase obişnuite pentru accesarea înregistrărilor. Tabela temporară va fi ştearsă automat când va fi închisă din zona respectivă. Din motive de optimizări nu tot timpul se va crea o tabelă temporară. Astfel, dacă se va interoga o singură tabelă fără a se redenumi câmpurile, se va deschide chiar tabela în cauză iar clauza de selecție va fi simulată prin setări *SET FILTER* şi *SET FIELDS*.

Sintaxa comenziiSELECT

SELECT **[*DISTINCT*] listă_coloane [*AS* redenumire_coloane]**
FROM **<listă_tabele>**
[*WHERE* condiție_de_selecție]
[*GROUP BY* listă_coloane]
[*ORDER BY* listă_coloane]
[*HAVING* condiție_de_grup]
[*SAVE TO* nume_fişier]
[*ALIAS* nume_alias]

Parametrii au următoarele semnificații:
DISTINCT – va elimina rândurile care au valori duplicate în coloanele

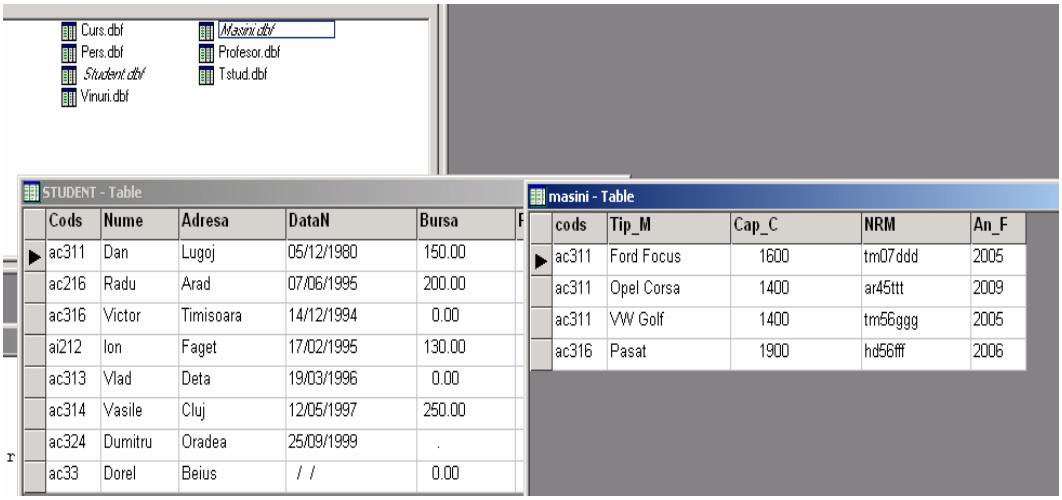
specificate în lista_col.
În lista_coloane câmpurile vor fi prefixate dacă este necesar de numele tabelei de care aparţin. Pentru a include în rezultat toate câmpurile se poate folosi notaţia ”*”.
AS redenumire_coloane – folosită dacă se doreşte redenumirea unor coloane în rezultat.
FROM <listă_tabele> - specifică tabelele din care sunt obţinute datele. Pe post de tabele se pot folosi şi rezultate ale unor operaţii de *outer* sau *inner join* conform SQL-92.
WHERE condiţie_de_selecţie – specifică condiţia care dictează ce înregistrări vor fi incluse în rezultat. Se poate folosi operatorul *IN* pentru a testa apartenenţa unui câmp la o mulţime precizată (ex. ”an_studiu IN (1,3,4)”). Sintaxa Local SQL nu permite includerea de subinterogări în clauza WHERE.
GROUP BY listă_coloane – specifică condiţii de grupare a înregistrărilor pentru funcţiile de agregare. **Numai câmpurile specificate aici pot să apară şi în lista *SELECT***.
ORDER BY listă_coloane – specifică ordonarea înregistrărilor în tabela rezultat
HAVING condiţie_de_grup – specifică o condiţie de grup ce trebuie indeplinită. Trebuie să aibă valoarea .T. pentru grupările care se doresc selectate. Poate apărea doar în combinaţie cu parametrul *GROUP BY*.
SAVE TO nume_fişier – specifică un fişier unde se va depune tabela rezultat. În caz contrar se va crea câte o tabelă temporară cu numele SQL_1,2,... pentru fiecare SELECT care va fi ştearsă la închidere.
ALIAS nume_alias – specifică un alias care va fi dat zonei în care se depune rezultatul. În felul acesta rezultatul se poate utiliza uşor în corelaţie cu alte tabele din baza de date.

```
select * from masini              // correct local memoreaza in urmatoarea zona
libera
? work()                  // afiseaza zona curenta in care s-a memorat
list
select * from student             // rezultatul in urmatoarea zona libera
browse
Close all                 // inchide toate fisierele din toate zonele
```

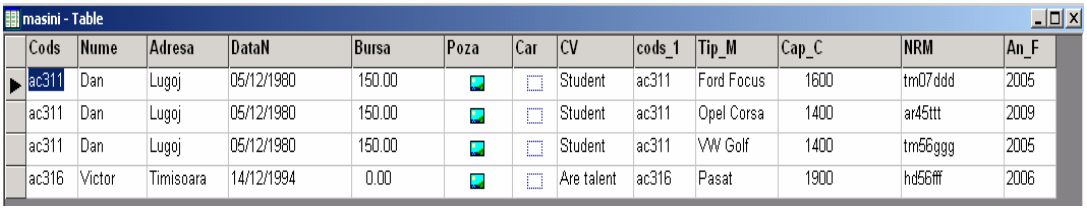
Fiecare Select se face in urmatoarea zona libera si denumite cu alias-ul _2, _3, ...

```
// Functioneaza dar nu se recomanda daca se reia comanda
select * from student save to Tstud    // correct nume fisier rezultat
clear all         //inchide
use tstud         // deschide intr-o zona libera  dar poate apare eroare daca e
deschis deja
browse

// se recomanda fiindca se rescrie peste in zona anterioara
select * from student alias TS          // utilizare alias pentru tab rezultat
? work()
select ts
browse

select 3                  // in zona 3 nu este nimic deschis
select * from student alias TS          // memorare rezultat tot in zona 1
anterioara
? work()                  // afiseaza 3 zona curenta
List              // eroare nici un fisier deschis in 3
Select TS         // selecteaza zona cu alias TS
? alias(1)        // afiseaza alias-ul fisierului din zona 1         - TS
List              // afiseaza rezultatul interogarii
? select()        // afisare urmatoarea zona libera
```

Operatia de Inner Join intre tabelele Student cu alias S si Masini cu alias M pe campul cods, va afişa toti studentii care au masina şi datele maşinilor. Nu se vor afişa studentii care nu au maşină.



Select * From Student s, Masini m where **s.cods = m.cods**
// Conditi de Join este **s.cods = m.cods**
? work() // afiseaza numarul zonei 7
? alias(7) // afiseaza alias-ul tabelei deschisein zona _7
Browse // afisare zona curenta



Daca dorim sa afişăm numai anumite câmpuri le putem specifica într-o listă, iar dacă numele câmpului este în ambele liste trebuie prefixat cu alias-ul tabelei din care face parte (S.Cods, M.cap_c). Fiecare câmp afişat poate sa aibă un alias după numele câmpului, care se va afişa in Browse: Nume Nume_student,
tip_m Marca

Select S.Cods,**Nume Nume_student**, Adresa, Bursa,tip_m Marca,M.Cap_c, nrm,an_f
From Student s, Masini m WHERE **s.cods = m.cods**

masini - Table

Cods	Nume_student	Adresa	Bursa	Marca	Cap_c	nrm	an_f
ac311	Dan	Lugoj	150.00	Ford Focus	1600	tm07ddd	2005
ac311	Dan	Lugoj	150.00	Opel Corsa	1400	ar45ttt	2009
ac311	Dan	Lugoj	150.00	VW Golf	1400	tm56ggg	2005
ac316	Victor	Timisoara	0.00	Pasat	1900	hd56fff	2006

In format standard forma interogării, care dă acelaşi rezultat este:

Select S.Cods,Nume Nume_student, Adresa, Bursa,tip_m Marca,M.Cap_c, nrm,an_f from Student S **Inner Join** Masini M ON **s.cods = m.cods** Alias SM

Pentru a afişa şi studenţii care nu au maşină, lăsând libere coloane cu datele maşinii se foloseşte
Left Outer Join şi s-a adăugat un **alias Outer** pentru a refolosi aceeaşi zonă (s-a ajuns la 14)

Select S.Cods,Nume Nume_student, Adresa, Bursa, tip_m Marca, M.Cap_c, nrm, an_f
from student s **Left Outer Join** masini m on **s.cods = m.cods** alias **Outer**

Pentru a afişa toate maşinile indiferent dacă au sau nu proprietari se foloşte **Right Outer Join**

Select S.Cods,Nume Nume_student, Adresa, Bursa,tip_m Marca,M.Cap_c, nrm,an_f from student s **Right Outer Join** masini m on **s.cods = m.cods** alias **Outer**

Se va folosi **Full Outer Join** dacă se iau toate înregistrările şi din tabela din stânga şi din cea din dreapta.

SQL_14 - Table

	cods	Nume_student	Adresa	Bursa	Marca	Cap_c	nrm	an_f
▶	ac216	Radu	Arad	200.00				
	ac311	Dan	Lugoj	150.00	VW Golf	1400	tm56ggg	2005
	ac311	Dan	Lugoj	150.00	Ford Focus	1600	tm07ddd	2005
	ac311	Dan	Lugoj	150.00	Opel Corsa	1400	ar45ttt	2009
	ac313	Vlad	Deta	0.00				
	ac314	Vasile	Cluj	250.00				
	ac316	Victor	Timisoara	0.00	Pasat	1900	hd56fff	2006
	ac324	Dumitru	Oradea	.				
	ac33	Dorel	Beius	0.00				
	ai212	Ion	Faget	130.00				

In limbajul SQL nu existe variabile dar se pot utiliza variabile din dBase ca variabile externe, care sun prefixate în comenzile Sql cu :var.
Se pot utiliza variabile a căror valoare poate fi setată prin dialog utilizând funcția Accept()

x=accept('vbursa') //se introduce valoarea bursei
x=val(x) // se transformă în tip numeric pentru a putea fi comparată cu Bursa
?x
select * from student where **bursa >:x**
browse

student - Table

	Cods	Nume	Adresa	DataN	Bursa	Poz
▶	ac311	Dan	Lugoj	05/12/1980	150.00	
	ac216	Radu	Arad	07/06/1995	200.00	
	ai212	Ion	Faget	17/02/1995	130.00	
	ac314	Vasile	Cluj	12/05/1997	250.00	

Studenții selecți pot fi afișați în ordinea alfabetică a numelor.

Select * from student where **bursa >:x Order by Nume**

SQL_3 - Table

	Cods	Nume	Adresa	DataN
▶	ac311	Dan	Lugoj	05/12/1980
	ai212	Ion	Faget	17/02/1995
	ac216	Radu	Arad	07/06/1995
	ac314	Vasile	Cluj	12/05/1997

Clauza DISTINCT se foloseşte pentru a afişa numai valorile distincte ale unor câmpuri.
Pentru tabela Note putem afişa numai odată fiecare din codurile studenţilor fără a le repeta, ţinând cont de faptul ca un student are mai multe note.

Select DISTINCT cods from Note alias lc
Browse

note - Table

Cods	CodC	Nota
ac311	BD	9.40
ac311	PBD	8
ac311	FIS	7.5
ac311	SO	9.2
ac311	PO	10
ac216	BD	9
ac316	SO	7
ai212	PO	9
ai212	SO	8
ac316	PBD	8.5
ac316	PO	7.5
ac316	FIS	9
ac316	SE	8.5
ac313	SE	10

SQL_1 - Table

cods
ac216
ac311
ac313
ac316
ai212

Select Distinct cods, codc from note

Comanda va fi folosită numai pentru a verifica dacă sunt, din greşeală, două note la acelaşi curs.

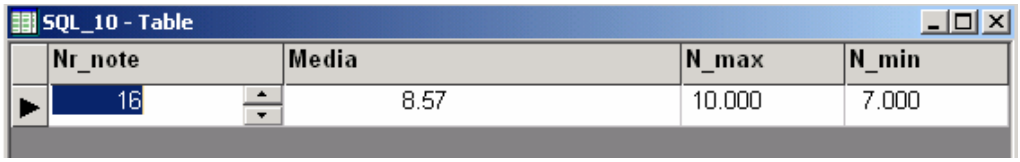
Utilizare funcţii de grup (agregat)

Se permite efectuarea unor calcule statistice pentru înregistrări utilizând valorile unui câmp.
Funţiile agregat sunt:
MIN(nota) - calculează valoarea minimă a câmpului nota

MAX(nota) - calculează valoarea maximă a câmpului nota
AVG(nota) - calculează valoarea medie a câmpului nota
COUNT(nota) - calculează numărul de valori ne nule ale câmpului nota

Pentru ca sensul să fie cât mai clar se poate ataşa câte un alias pentru fiecare funcţie:

Select count(nota) Nr_note, avg(nota) Media, max(nota) N_max,min(nota) N_min from Note
Browse
Dacă nu s-a specificat nici o grupare a înregistrărilor, funcţiile agregat se aplică pe toate înregistrările din fişier. In lista de selecţie pot apare numai funcţii agregat şi nici un câmp.


SQL_10 - Table

Nr_note	Media	N_max	N_min
16	8.57	10.000	7.000

Dacă se doreşte gruparea înregistrărilor după valoarea unui câmp, funcţiile agregat se aplică pe fiecare grup. Dacă alegem gruparea după Cods funcţiile se vor aplica pentru notele fiecărui student şi câmpul Cods poate fi afişat.

Select **cods**, count(nota) Nr_note,avg(nota) Media, max(nota) N_max,min(nota) N_min
From note **Group by cods**
Browse

SQL_11 - Table

cods	Nr_note	Media	N_max	N_min
ac216	1	9.00	9.000	9.000
ac311	5	8.82	10.000	7.500
ac313	1	10.00	10.000	10.000
ac316	5	8.10	9.000	7.000
ai212	4	8.38	9.500	7.000

Se pot selecta numai studenţii de la secţia ac prin condiţia where cods like ‘ac%’, unde % indică oricâte caractere oarecare, iar prin _ se acceptă un singur

caracter oarecare
'a_2%' reprezintă toți studenții din anul 2 toate secțiile.

Select cods, count(nota) Nr_note,avg(nota) Media, max(nota) N_max,min(nota) N_min
From note Where **cods like 'ac%'** Group by cods
Browse

SQL_2 - Table

cods	Nr_note	Media	N_max	N_min
ac216	1	9.00	9.000	9.000
ac311	5	8.82	10.000	7.500
ac313	1	10.00	10.000	10.000
ac316	5	8.10	9.000	7.000

Select cods, count(nota) Nr_note,avg(nota) Media, max(nota) N_max,min(nota) N_min
From note Where **cods like 'a_2%'** Group by cods
Browse

SQL_3 - Table

cods	Nr_note	Media	N_max	N_min
ac216	1	9.00	9.000	9.000
ai212	4	8.38	9.500	7.000

Se admit în lista de selecție numai funcții agregat şi câmpuri din lista de grupare.

Se permite calculul mediei şi pentru note corectate:

Select **cods,** count(nota) Nr_note,avg(nota+0.51) Media, max(nota) N_max,min(nota) N_min
From note **Group by cods**
browse

SQL_13 - Table

	cods	Nr_note	Media	N_max	N_min
▶	ac216	1	9.51	9.000	9.000
	ac311	5	9.33	10.000	7.500
	ac313	1	10.51	10.000	10.000
	ac316	5	8.61	9.000	7.000
	ai212	4	8.88	9.500	7.000

Pentru a afişa numai studenţii cu medii peste o valoare se foloseşte clauza HAVING care se completează cu o condiţie de grup.

Select cods, count(nota) Nr_note,avg(nota) Media, max(nota) N_max,min(nota) N_min
From note Group by cods **Having avg(nota)>8.40**
Browse

SQL_14 - Table

	cods	Nr_note	Media	N_max	N_min
▶	ac216	1	9.00	9.000	9.000
	ac311	5	8.82	10.000	7.500
	ac313	1	10.00	10.000	10.000

Funcţiile agregat pot fi folosite şi în operaţiile de Join, caz în care se aplică pe rezultatul final.
Vom aplica aceleaşi funcţii agregat pentru a afişa şi datele despre studenţi, obţinute prin Join între tabelele Student şi Note.

Select S.cods, S.Nume Nume_student,S.Adresa,S.Bursa,count(nota) Nr_note,avg(nota) Media,
max(nota) N_max,min(nota) N_min
From Student S, Note N Where s.cods=n.cods
Group by s.cods,s.Nume,adresa,bursa
browse

SQL_2 - Table

cods	Nume_student	Adresa	Bursa	Nr_note	Media	N_max	N_min
ac216	Radu	Arad	200.00	1	9.00	9.000	9.000
ac311	Dan	Lugoj	150.00	5	8.82	10.000	7.500
ac313	Vlad	Deta	0.00	1	10.00	10.000	10.000
ac316	Victor	Timisoara	0.00	5	8.10	9.000	7.000
ai212	Ion	Faget	130.00	4	8.38	9.500	7.000

Câmpurile Nume, Adresa, Bursa din tabela Student pot fi afişate numai dacă sunt specificate în clauza Group By. Dacă unul lipseşte apare eroarea:

Dacă se doreşte o situaţie a notelor pe cursuri, se păstrează toate funcţiile agregat şi se face un Join între tabela Curs şi Note, având condiţia de Join c.codc=n.codc şi gruparea pe codc,Titlu curs şi codp (cod profesor).

```
Select   C.codc, C.Titlu Titlu_Curs,count(nota) Nr_note,avg(nota) Media,
max(nota) N_max,
        min(nota) N_min
        From Curs C, Note N
        Where C.codc=n.codc
         Group by C.codc,C.Titlu,C.codp
```

browse

SQL_4 - Table

codc	Titlu_curs	Nr_note	Media	N_max	N_min
BD	Baze de date	3	9.30	9.500	9.000
FIS	Fund Ing software	2	8.25	9.000	7.500
PBD	Proiectarea BD	2	8.25	8.500	8.000
PO	Progr. pe obiecte	3	8.83	10.000	7.500
SE	Sisteme expert	3	8.50	10.000	7.000
SO	Sist de operare	3	8.07	9.200	7.000

11.5. SQL extern din dBase Plus

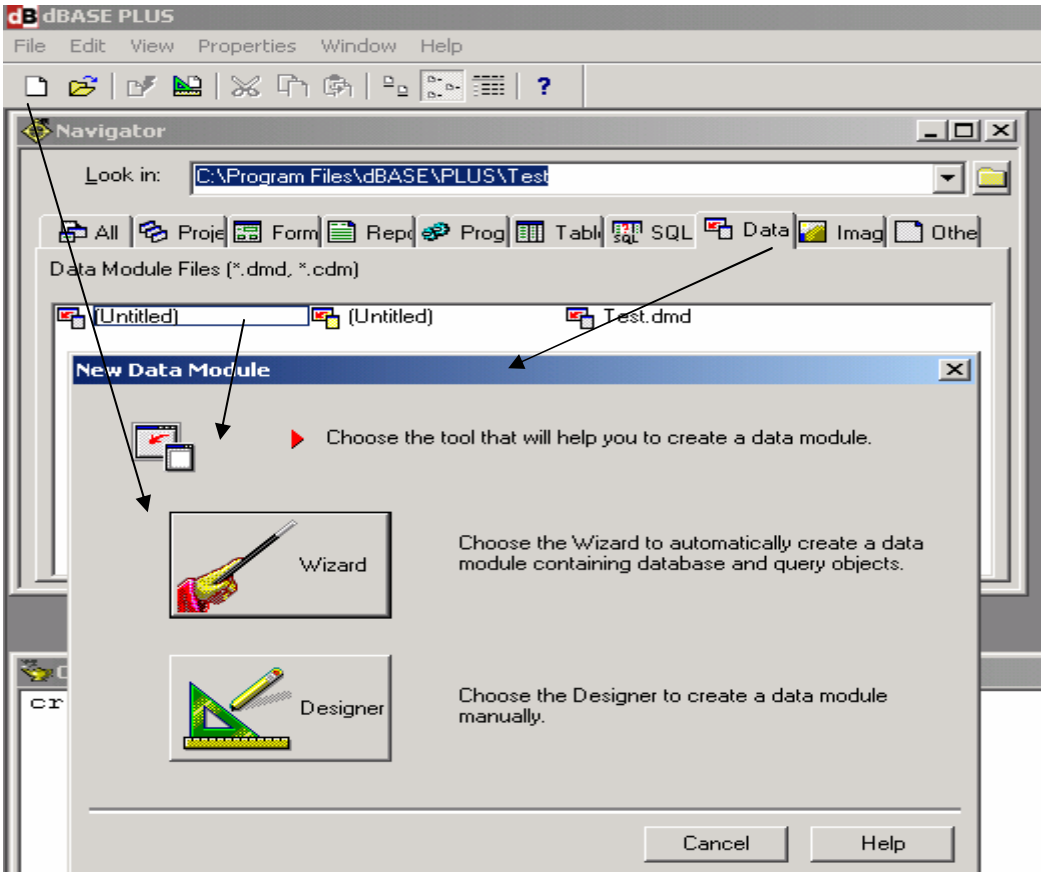
Din dBase Plus se pot face una sau mai multe **conexiuni la Servere de BD** Oracle, DB2-IBM, Sybase, Informix.

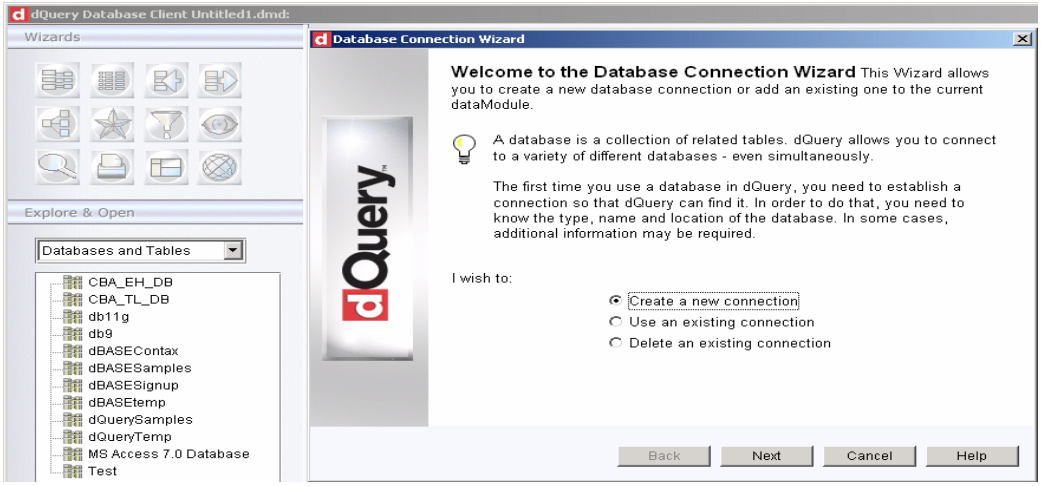
- Conexiunea la o BD externa se creează printr-un obiect Data Module.
- Deschiderea BD externe se face prin:

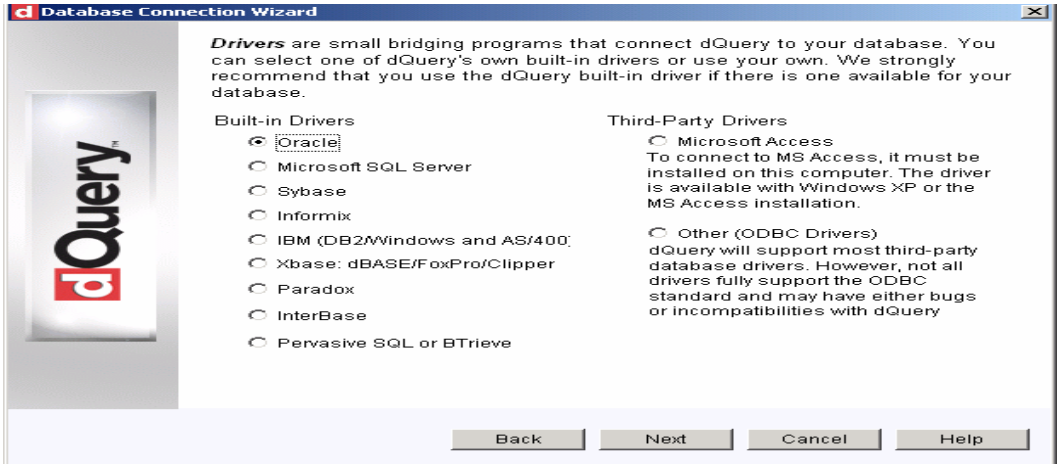
  Open database nume_BD
- Se pot face interogări ale tabelelor din BD prin prefixare cu numele BD
- Tabelele BD se pot utiliza şi utilizând comenzile dBase (Use, Browse, Edit,..)

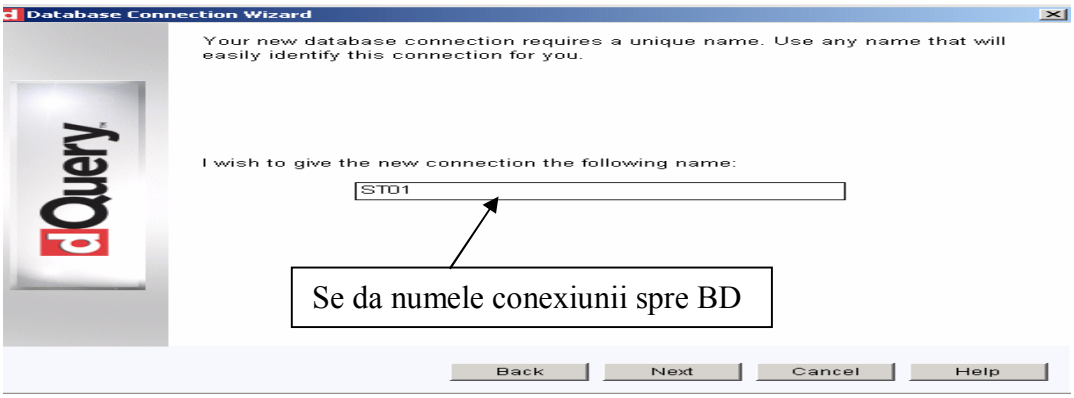
Crearea unei conexiuni la o BD externa

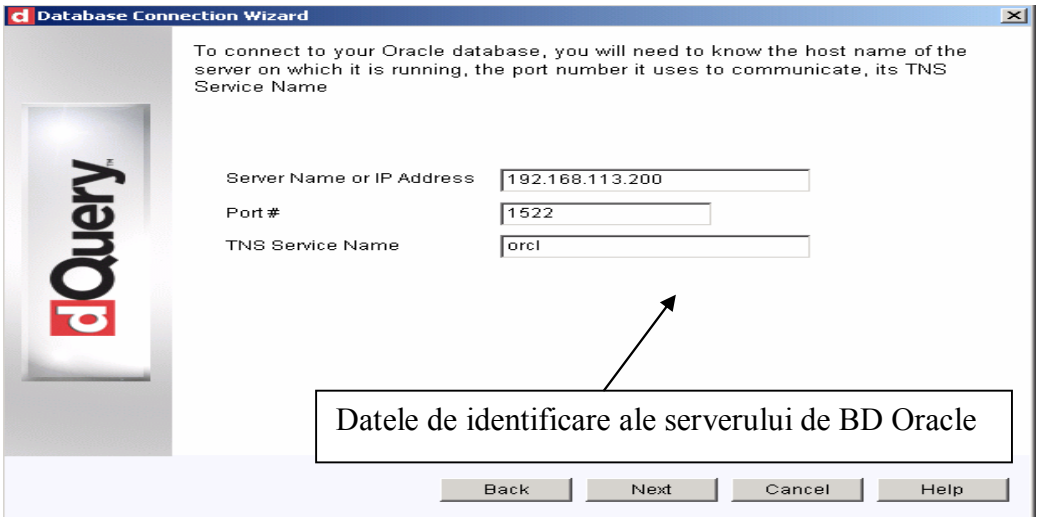
Se selectează crearea unui Data Module nou si se utilizează Wizard, care va activa dQuery.

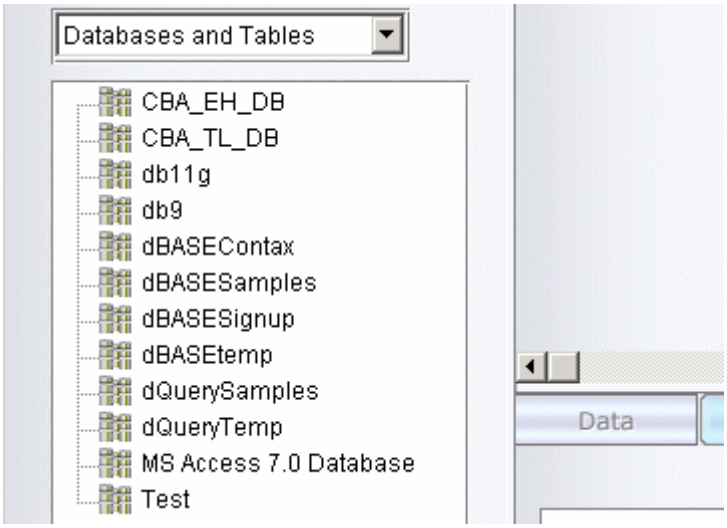



dQuery Database Client Untitled1.dmd:
Wizards
Explore & Open
Databases and Tables
CBA_EH_DB
CBA_TL_DB
db11g
db9
dBASEContax
dBASESamples
dBASESignup
dBASEtemp
dQuerySamples
dQueryTemp
MS Access 7.0 Database
Test
Database Connection Wizard
Welcome to the Database Connection Wizard This Wizard allows you to create a new database connection or add an existing one to the current dataModule.
A database is a collection of related tables. dQuery allows you to connect to a variety of different databases - even simultaneously.
The first time you use a database in dQuery, you need to establish a connection so that dQuery can find it. In order to do that, you need to know the type, name and location of the database. In some cases, additional information may be required.
I wish to:
Create a new connection
Use an existing connection
Delete an existing connection
Back
Next
Cancel
Help


Database Connection Wizard
Drivers are small bridging programs that connect dQuery to your database. You can select one of dQuery's own built-in drivers or use your own. We strongly recommend that you use the dQuery built-in driver if there is one available for your database.
Built-in Drivers
Oracle
Microsoft SQL Server
Sybase
Informix
IBM (DB2/Windows and AS/400)
Xbase: dBASE/FoxPro/Clipper
Paradox
InterBase
Pervasive SQL or BTrieve
Third-Party Drivers
Microsoft Access
To connect to MS Access, it must be installed on this computer. The driver is available with Windows XP or the MS Access installation.
Other (ODBC Drivers)
dQuery will support most third-party database drivers. However, not all drivers fully support the ODBC standard and may have either bugs or incompatibilities with dQuery
Back
Next
Cancel
Help


Database Connection Wizard
Your new database connection requires a unique name. Use any name that will easily identify this connection for you.
I wish to give the new connection the following name:
ST01
Se da numele conexiunii spre BD
Back
Next
Cancel
Help

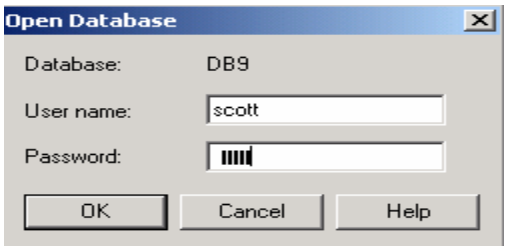




S-a creat conexiunea DB9 spre o BD Oracle şi conexiunea Test spre directorul Test, care este văzut ca o Bd de tip XBase si conține tabelele Student, Note, Cursuri, Profesor utilizate deja.

Exemple de utilizare SQL extern

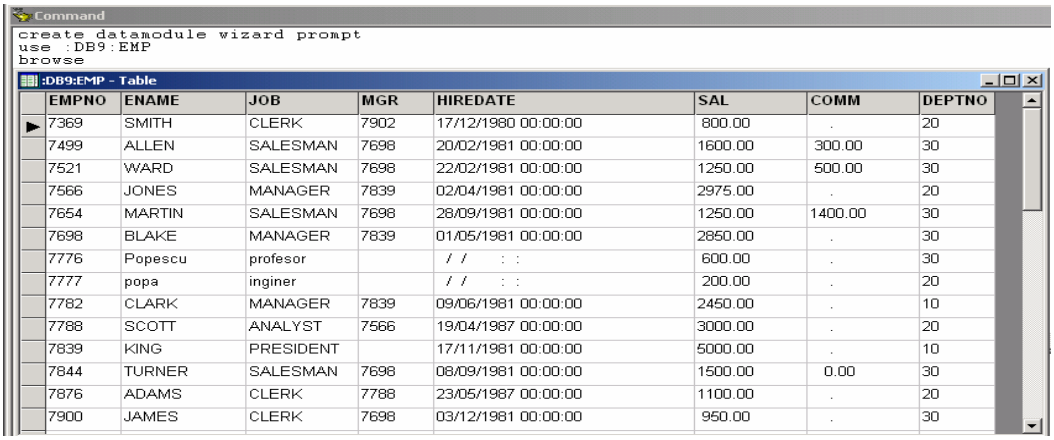
select * from **emp** // gresit! Trebuie numele bazei de date, deja deschisă
open database db9 // deschidere BD prin conexiunea db9



Userul si parola intrare in Oracle.
Pentru laborator:User: st01 -- schema in care ne conectam
Parola: st01

use **:DB9:EMP** // correct cu use, list, browse.
Browse // sau List

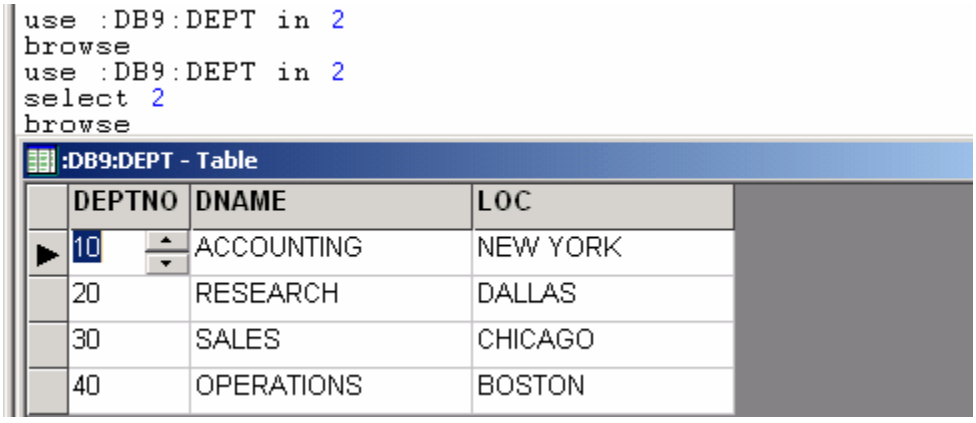
Numele BD DB9 si tabelei EMP se prefixează cu „:" fiind externe dBase



EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17/12/1980 00:00:00	800.00	.	20
7499	ALLEN	SALESMAN	7698	20/02/1981 00:00:00	1600.00	300.00	30
7521	WARD	SALESMAN	7698	22/02/1981 00:00:00	1250.00	500.00	30
7566	JONES	MANAGER	7839	02/04/1981 00:00:00	2975.00	.	20
7654	MARTIN	SALESMAN	7698	28/09/1981 00:00:00	1250.00	1400.00	30
7698	BLAKE	MANAGER	7839	01/05/1981 00:00:00	2850.00	.	30
7776	Popescu	profesor		/ / : :	600.00	.	30
7777	popa	inginer		/ / : :	200.00	.	20
7782	CLARK	MANAGER	7839	09/06/1981 00:00:00	2450.00	.	10
7788	SCOTT	ANALYST	7566	19/04/1987 00:00:00	3000.00	.	20
7839	KING	PRESIDENT		17/11/1981 00:00:00	5000.00	.	10
7844	TURNER	SALESMAN	7698	08/09/1981 00:00:00	1500.00	0.00	30
7876	ADAMS	CLERK	7788	23/05/1987 00:00:00	1100.00	.	20
7900	JAMES	CLERK	7698	03/12/1981 00:00:00	950.00	.	30

use EMP // gresit

use **:DB9:DEPT** in 2 // deschidere tabel departamente in zona 2
select 2
Browse



DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

open database db9 // correct deschidere BD Oracle
Select * from :DB9:EMP

Browse

EMP - Table

	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
▶	7369	SMITH	CLERK	7902	17/12/1980 00:00:00	800.00	.	20
	7499	ALLEN	SALESMAN	7698	20/02/1981 00:00:00	1600.00	300.00	30
	7521	WARD	SALESMAN	7698	22/02/1981 00:00:00	1250.00	500.00	30
	7566	JONES	MANAGER	7839	02/04/1981 00:00:00	2975.00	.	20

Copierea unor tabele din Oracle in dBase

Prelucrările mai complexe a tabelelor direct pe BD Oracle pot să pună probleme în cazul utilizării tabelelor locale şi externe în operațiile de Join. Din acest motiv este indicat ca anumite tabele Oracle să fie copiate în tabele locale şi operațiile de combinare să se facă local.
Copierea se face în 2 faze:

- deschiderea tabelei externe într-o zonă de lucru Oracle
- crearea unei copy a tabelei externe deschise într-un director dBase

Tabela astfel creată poate fi deschisă într-o zonă de lucru sau utilizată în operații relationale prin SQL local. Vom exemplifica prin copierea tabelelor de salariați EMP si departamente Dept. Utilizarea directă a opțiunii Save To din SQL pe bază de date externă este ilegal

```
use :db9:emp        // deschidere tabela din dBase in zona curentă
copy to empl        // copiere tabela din zona curenta in fisierul Empl (local)
use empl in 2       // deschidere fisier local Empl în zona 2
select 2
browse
```

empl - Table

	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
▶	7369.00	SMITH	CLERK	7902.00	2/1980 00:00:00	800.00	.	20.00
	7499.00	ALLEN	SALESMAN	7698.00	2/1981 00:00:00	1600.00	300.00	30.00
	7521.00	WARD	SALESMAN	7698.00	2/1981 00:00:00	1250.00	500.00	30.00
	7566.00	JONES	MANAGER	7839.00	4/1981 00:00:00	2975.00	.	20.00
	7654.00	MARTIN	SALESMAN	7698.00	9/1981 00:00:00	1250.00	1400.00	30.00

```
use :db9:dept
copy to dept1
use dept1 in 3
browse
```

:db9:DEPT - Table

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

In continuare se cere afişarea Nume, Nr.departament, salar şi D.Dname Nume departament, care se obține printr-un Join intre tabelele EMP şi Dept din BD externă DB9.

Select E.ENAME Nume_salariat, E.DEPTNO Nr_Dept, E.sal Salar, D.DNAME
from :DB9:EMP E, :DB9:DEPT D
where E.DEPTNO = D.DEPTNO
browse

EMP - Table

Nume_salariat	Nr_Dept	Salar	DNAME
SMITH	20	800	RESEARCH
ALLEN	30	1600	SALES
WARD	30	1250	SALES
JONES	20	2975	RESEARCH
MARTIN	30	1250	SALES
BLAKE	30	2850	SALES
Popescu	30	600	SALES
popa	20	200	RESEARCH
CLARK	10	2450	ACCOUNTING
SCOTT	20	3000	RESEARCH

Self Join

Operația Self Join presupune executarea unui Join între câmpuri din aceeaşi tabelă, caz în care tabela se deschide de 2 ori cu alias diferit. Câmpul Mgr din tabela Empl specifică codul (Empno) al şefului care şi el este salariat.

Se deschide de 2 ori tabela Empl cu alias E respective S -sefi şi se face Join intre cele două tabele pentru conditia e.mgr=s.empno pentru a determina datele sefului.

select e.ename Nume_salariat, e.sal Salar,e.mgr,s.empno, s.ename

Nume_sef,s.sal salr_sef
from empl e, empl s where **e.mgr = s.empno**
browse

SQL_3 - Table

	Nume_salariat	Salar	mgr	empno	Nume_sef	salr_sef
▶	SCOTT	3000.00	7566.00	7566.00	JONES	2975.00
	FORD	3000.00	7566.00	7566.00	JONES	2975.00
	ALLEN	1600.00	7698.00	7698.00	BLAKE	2850.00
	WARD	1250.00	7698.00	7698.00	BLAKE	2850.00
	MARTIN	1250.00	7698.00	7698.00	BLAKE	2850.00
	TURNER	1500.00	7698.00	7698.00	BLAKE	2850.00

Nu se acceptă în present Self Join pe o tabelă externă, dar se acceptă cu o copie locală:

select e.ename Nume_salariat, e.sal Salar,e.mgr,s.empno, s.ename Nume_sef,s.sal salr_sef
from :db9:emp e, empl s where e.mgr=s.empno
browse

Se acceptă operaţii de Join între tabele locale şi tabele din BD externă Oracle. Vom face un Join între tabela externă :DB9:Emp şi tabela locală Deptl:

open database db9
use :db9:emp
select e.ename Nume,e.deptno, d.dname Departament
from :db9:emp E, deptl d where e.deptno=d.deptno
browse

Command

```
open database db9
use :db9:emp
select e.ename Nume,e.deptno, d.dname Departament from :db9:emp E, deptl d where e.deptno=d.deptno
browse
```

SQL_2 - Table

	Nume	deptno	Departament
▶	CLARK	10.00	ACCOUNTING
	MILLER	10.00	ACCOUNTING
	KING	10.00	ACCOUNTING
	FORD	20.00	RESEARCH
	ADAMS	20.00	RESEARCH

Self join poate simplifica proiectarea unei BD. Într-un tabel de persoane Pers se poate specifica:

- CNP - codul numeric personal pentru fiecare persoană
- CNPT - CNP-ul tatălui care se găseşte în tabelul Pers

- CNPM - CNP-ul mamei care se găseşte în tabelul Pers

CNP	Nume	DataN	Adresa	CNPT	CNPM

La afişarea datelor unei persoane se vor adăuga Numele, Adresa şi DataN tatălui şi mamei folosind un Self Join. Se va deschide de 3 ori tabela Pers cu aliasul P , T - taţi, M – mame.

```
Select P.Nume, P.DataN, P.Adresa, T.Nume Nume_tata, M.Nume
Nume_mama
        From Pers P, Pers T, Pers M
        Where P.CnpT=T.Cnp AND P.CnpM=M.Cnp
```

Select-uri imbricate

In SQL standard şi în dBase Plus se admit Select-uri imbricate, în care un atribut poate fi înlocuit cu rezultatul unui Select.

Pentru tabela de salariaţi Empl dorim să aflăm care salariaţi au aceeaşi meserie(job) ca şi un salariat dat prin Nume:

```
select * From empl Where job=(Select Job From empl Where Ename
='SCOTT')
browse
```

```
select * From empl Where job=(Select Job From empl Where Ename ='SCOTT')
browse
```

SQL_5 - Table

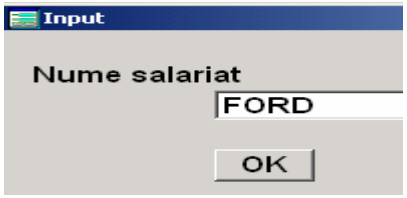
EMPNO	ENAME	JOB	MGR
7788.00	SCOTT	ANALYST	7566.00
7902.00	FORD	ANALYST	7566.00

În acest caz am avut un singur salariat selectat. Dacă din Select rezultă mai multe înregistrări selectate în faţa operaţiei select se va pune obligatoriu ANY, care indică toate valorile din înregistrările selectate.

Numele salariatului poate fi introdus de la tastatura ca variabilă externă SQL cu Accept

```
Vnume=Accept('Nume salariat')
```

select * From empl Where job=(Select Job From empl Where **Ename = :Vnume)**



SQL_9 - Table

	EMPNO	ENAME	JOB	MGR
▶	7788.00	SCOTT	ANALYST	7566.00
	7902.00	FORD	ANALYST	7566.00

Afişăm toţi studenţii care participă la cursuri cu titlul care începe cu Baze...

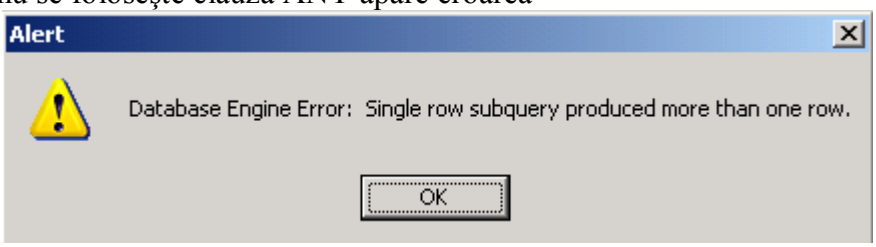
- vom determina printr-un Select codurile Cods ale cursurilor cu titlu 'Baze%' din tabela Curs
- toate aceste valori Cods vor apare în clauza Where ale unui Select pe Codc din tabela Note determinând valorile câmpului Cods pentru acele cursuri
- valorile Cods determinate vor apare în clauza Where a unui Select care va afişa datele studenţilor identificaţi

```
Select Cods,Nume, Adresa, Bursa, DataN
From Student where cods=
        any (Select cods from note where codc=
                any (Select codc  From Curs Where titlu like 'Baze%'))
browse
```

SQL_4 - Table

	Cods	Nume	Adresa	Bursa	DataN
▶	ac311	Dan	Lugoj	150.00	05/12/1980
	ac216	Radu	Arad	200.00	07/06/1995
	ai212	Ion	Faget	130.00	17/02/1995

Dacă nu se foloseşte clauza ANY apare eroarea



Selecturile imbricate sunt foarte rapide, dar putem afişa numai

câmpurile din prima tabelă (Student).

Dacă se foloseşte operația de Join se pot folosi câmpurile din toate tabelele implicate.

Select S.Nume,S.Bursa, N.Nota, C.Titlu from Student S, Note N,Curs C Where C.codc=N.Codc AND N.cods=S.Cods AND c.titlu Like 'Baze%'
browse

SQL_11 - Table

Nume	Bursa	Nota	Titlu
Dan	150.00	9.400	Baze de date
Radu	200.00	9.000	Baze de date
Ion	130.00	9.500	Baze de date

Select-urile imbricate sunt utile mai ales când se utilizează funcții agregat pentru comparații.
Afişăm toți salariații din tabela Empl care au salar mai mare decât media salariilor.

Select Ename Nume_salariat,sal Salar From Empl Where sal>(Select avg(sal) from empl)
browse
Select avg(sal) salar_mediu from empl

SQL_12 - Table

Nume_salariat	Salar
JONES	2975.00
BLAKE	2850.00
CLARK	2450.00
SCOTT	3000.00
KING	5000.00
FORD	3000.00

SQL_14 - Table

salar_mediu
1864.06

Afişam salariații care au salarii mai mari decât media salariilor unui departament:

Select Ename Nume, sal Salar, deptno from empl where sal >
(select avg(sal) from empl where deptno=10)

:TEST:SQL_13 - Table

	Nume	Salar	deptno
▶	JONES	2975.00	20.00
	SCOTT	3000.00	20.00
	KING	5000.00	10.00
	FORD	3000.00	20.00

In dBase Plus **nu se acceptă în select imbricat clauza Group By** curent utilizată în Oracle.
Afişarea salariaţilor care au salarii mai mari decât salariile medii pentru cel puţin un departament nu funcţionează:

```
Select Ename Nume, sal Salar, deptno from empl where sal>
        any(select avg(sal) from empl group by deptno )     -- cel puţin un departament
        all(select avg(sal) from empl group by deptno )     -- fata de toate departamentele
```

Afişăm toţi salariaţii care au salar >= decât cel puţin un Manager (ANY).

```
Select Ename Nume, sal Salar,job, deptno from empl where sal >=
        ANY(select sal from empl where job ='MANAGER' )
```

:TEST:SQL_20 - Table

	Nume	Salar	job	deptno
▶	JONES	2975.00	MANAGER	20.00
	BLAKE	2850.00	MANAGER	30.00
	CLARK	2450.00	MANAGER	10.00
	SCOTT	3000.00	ANALYST	20.00
	KING	5000.00	PRESIDENT	10.00
	FORD	3000.00	ANALYST	20.00

Pentru Salar >= decât salarul tuturor Managerilor (ALL)

```
Select Ename Nume, sal Salar,job, deptno from empl where sal >=
        ALL(select sal from empl where job ='MANAGER' )
```

:TEST:SQL_21 - Table

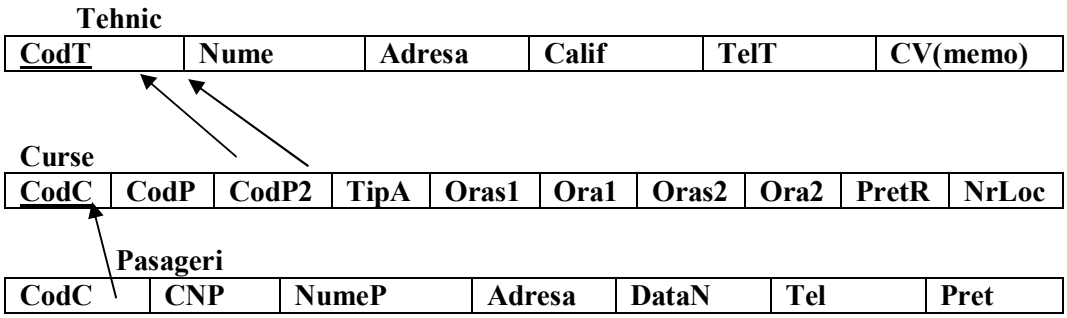
Nume	Salar	job	deptno
JONES	2975.00	MANAGER	20.00
SCOTT	3000.00	ANALYST	20.00
KING	5000.00	PRESIDENT	10.00
FORD	3000.00	ANALYST	20.00

Se observă că dintre manageri mai rămâne numai Jones, care are salarul cel mai mare.
Dacă condiția ar fi > atunci nu ar mai apare nici el.

12. Baza de date pentru rezervare bilete la avion

Se prezintă mai jos enunțul unui subiect de examen însoțit de indicații de implementare, exemplu de proiectare a formului şi procedurile folosite OnOpen, OnChange şi cele ataşate butoanelor.

Baza de date are structura normalizată şi este formata din următoarele tabele:
Curse – tabela care contine toate cursele de avion şi este indexată după CodC.
Codul cursei conține luna si ziua pentru rezervari pe un an.
Tehnic – Tabela conține tot personalul tehnic calificat, indifferent de profesie şi are cheia primară
CodT după care se indexează
Pasageri – tabela conține toți pasagerii tuturor curselor, care pot fi selectati prin CodC care este cheie secundară şi după ea se face indexarea. Se pot găsi astfel toți pasagerii unei curse.
Indexând după CNP pot fi afişate toate cursele pe care a circulat o persoană.



Folosind Designerul dBase să se realizeze un program care utilizează un Form ce conține Butoane,texte şi EntryField-uri pentru afişare conținut câmpuri, texte explicative şi care realizează fuctiile:

- **Rezervare –** la o cursă dată prin CodC cu afişare informații despre cursă
- **Afişare pasageri –** pentru o cursa dată prin CodC. Datele pasagerilor vor fi afisate intr-un Browse, dupa ce au fost selectați intr-un fisier temporar Ftemp atasat obiectului Browse.
- **Statistica** -Calculeaza in SQL şi afisează pentru fiecare cursa Codc, Oras1,Oras2 suma preturilor incasate pe biletele vandute, folosind comanda Select cu clauza **Group By** şi operația de **Join** intre tabelele Curse si Pasageri.

Butoanele folosite vor fi:

Rezervare – adaugă datele unui pasager din EntryField-urile din zona Date

pasager în fişierul Pasageri pentru o cursă CodC dacă există locuri pe acea cursă. După rezervare se decrementează NrLoc din fişierul Curse

Afisare Pasageri - pentru o cursă dată.

- Selectează cursa care are codul dat şi afişează toate datele cursei în EntryFielduri
- Selectează din fişierul Tehnic numele pilotului şi copilotului specificaţi în înregistrarea din fişierul Curse prinCodP şi CodP2 si le completează in form
- Selectează pasagerii din fişierul Pasageri care au codul cursei si ii plasează într-un fisier temporar Ftemp, de unde vor fi afişaţi în obiectul Browse specificând alias.

Statistica – va declanşa o procedură care utilizând limbajul SQL va face o prelucrare statistică folosind un Join între tabelele Curse şi Pasager, iar apoi va folosi un Group By pentru a grupa toţi pasagerii pe curse după codul cursei. Rezultatul comenzii Select folosite se va memora într-o tabelă pentru care se specifică un alias ce va fi folosit pentru afişare prin obiectul Browse.

Recomandam utilizarea unui Form ca cel de mai jos:

CodC	Oras1	Ora1	Oras2	Ora2	Pilot	Copilot	TipAv
Ro234-0609	**Timisoara TSR**	**8:30**	**Bucuresti**	**9:30**	**Popescu**	**Adam**	**B747**

CNP	**Nume Pas**	**Adresa**	**Telefon**	**DataN**	**Pret**

Lista Pasageri

CNP	**Nume Pas**	**Adresa**	**Telefon**	**DataN**	**Pret**

Rezervare loc **Afisare pasageri** **Statistica**

Formul şi obiectele plasate pe el vor fi realizate cu designer-ul.
Obiectele vor fi grupate:

- Datele cursei care se vor afişa automat prin procedura OnChange la introducerea unei noi curse în câmpul Codc. Procedura v-a lua codul cursei din Form.entryField1.value şi o va folosi pentru căutarea cursei in fişierul curse folosind indexul Icodc.
- Datele ce se completează pentru pasageri la rezervarea locului
- Un obiect Browse vid pe care se vor afişa pasagerii unei curse sau tabela statistică, modificând dinamic alias-ul asociat

In procedura OnOpen lansata automat la deschiderea Form-ului se deschid fişierele folosite Curse, Pasageri şi Tehnic şi se declara legătura dintre fişierul Curse şi Pasageri.
La închiderea Form-ului în procedura OnClose se inchid toate fişiere.

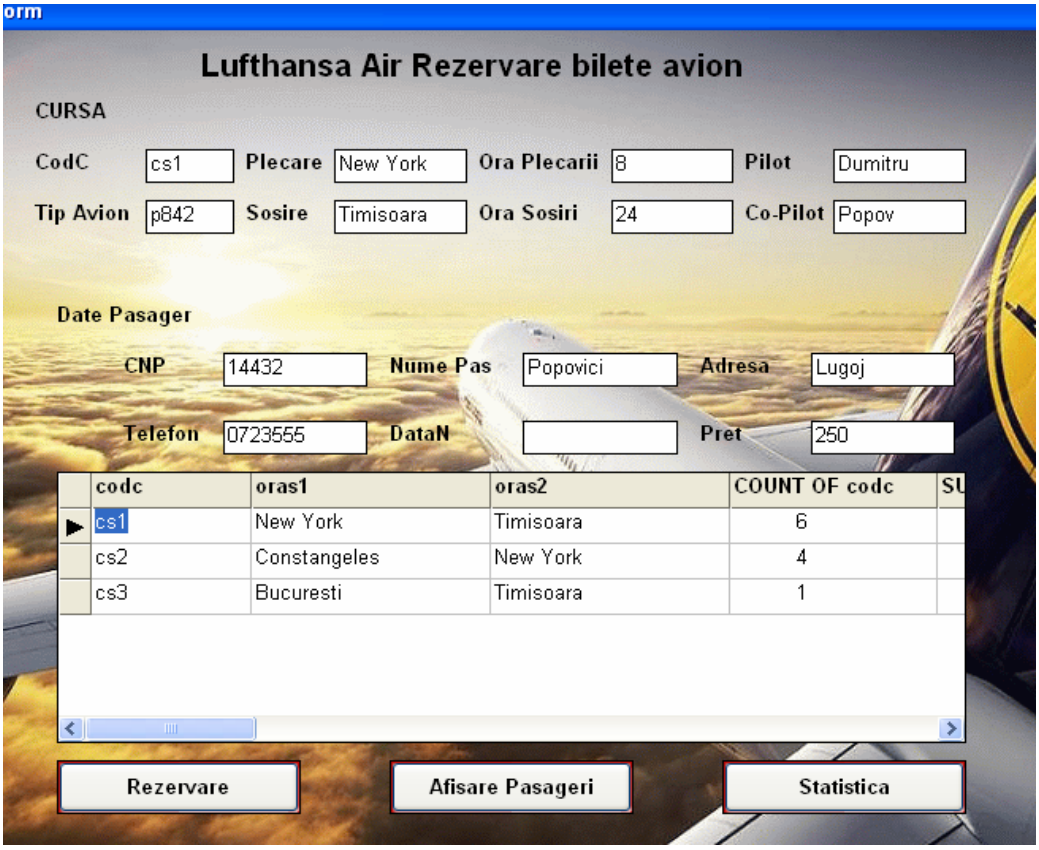
In figură se prezintă Form-ul activ pentru gestiunea biletelor la avion, unde:

- s-a selectat codul unei curse şi s-au afişat datele cursei
- s-a făcut o rezervare pentru un pasager completânt datele personale
- s-au afişat toti pasagerii cursei specificate prin Codc în Entryfield folosind obiectul Browse

Procedurile asociate evenimentelor

```
Procedure Form_OnOpen
  clear
  //  Deschidere fisiere stabilire legaturi intre fisiere
    use curse in 1 index iCodc alias curse
    use pasageri in 2 index iCodcP alias pasageri
    use tehnic in 3 index iCodt alias tehnic
      select curse
    set relation to codc into pasageri constrain
      return
  Procedure Form_OnClose
  close all
  return
```

In următoarea figură se vede cum obiectul Browse este folosit pentru afişarea rezultatului obţinut din interogarea SQL cerută pentru date statistice.



```
Procedure ENTRYFIELD1_OnChange
select 1        // Completare date despre cursa si avion
  cod=form.entryfield1.value
  seek cod              //Cautare cursa dupa codc din entryfield
  if found()
    form.entryfield2.value=oras1
    form.entryfield3.value=ora1
    form.entryfield4.value=oras2
    form.entryfield5.value=ora2
    form.entryfield8.value=TipA
    select 3      // completare date piloti din tabela Tehnic
      seek curse->codp
      form.entryfield6.value=nume
      seek curse->codp2
      form.entryfield7.value=nume
  else
    msgbox("Cursa "+cod+" nu exista!")
  endif
  return
 Procedure PUSHBUTTON1_OnClick
// Rezervare – completare date pasager
  select 2          // Tabel Pasageri
   codcc=form.entryfield1.value      // Cod cursa
  seek codcc      // Completare date pasager  pentru buton rezervare
  if found()

  append blank
  replace codc with codcc
  replace cnp with val(form.entryfield9.value)
  replace numep with form.entryfield10.value
  replace adresa with form.entryfield11.value
  replace datan with ctod(form.entryfield13.value)
  replace tel with val(form.entryfield12.value)
  replace pret with val(form.entryfield14.value)
  endif

   select 1    // tabel Curse
  seek codcc
  if found()
```

```
replace nrl with nrl-1          // actualizare locuri libere cursa
endif
  msgbox("Rezervare efectuata!")
return

Procedure PUSHBUTTON2_OnClick
// Afisare pasageri dintr-o cursa
select 2    // tabel pasageri

use ftemp exclusive in 4
// Fisierul s-a creat cu Use pasageri   Copy Stru To Ftemp
select 4     // fisier temporar pentru pasageri dintr-o cursa
//delete all
// pack
 Zap  // stergere fisier
cod=form.entryfield1.value       // Cod cursa

select 1          // fisier Curse
  cod=form.entryfield1.value
  seek cod
  if found()
// afisare date curse * se putea elimina fiind in  OnChange
     form.entryfield2.value=oras1
     form.entryfield3.value=ora1
     form.entryfield4.value=oras2
     form.entryfield5.value=ora2
     form.entryfield8.value=TipA
     select 3    //  Date piloti din fisierul Tehnic
       seek curse->codp       //cod pilot
       form.entryfield6.value=nume    // Nume pilot
       seek curse->codp2                // Cod copilot
       form.entryfield7.value=nume     // Nume copilot
       endif

select 2      // fisier pasageri
  seek cod      // cauta primul pasager din cursa
if found()
* Copiere pasgeri din cursa selectata in fisierul temporar Ftemp
do while .not. eof(2) .and. codc=cod  //ciclu pentru toti pasagerii cursei
select 4                  // fisier temporar pentru pasageri dintr-o cursa
```

```
append blank              // datele unui pasager din cursa
replace cnp with pasageri->cnp
replace numep with pasageri->numep
replace adresa with pasageri->adresa
replace datan with pasageri->datan
replace tel with pasageri->tel
replace pret with pasageri->pret

select 2
skip       //  urmatorul pasager  *se putea skip in 2
enddo

// Afisare fisier temporar in Browse
form.browse1.alias='Ftemp'   // atasare Fis Temporar la browse
endif

return

Procedure PUSHBUTTON3_OnClick
// procedura pentru Select care face statistica ceruta cu Group By

select c.codc,c.oras1,c.oras2,count(p.codc),sum(p.pret);
 from curse c,pasageri p where p.codc=c.codc;
  group by c.codc,c.oras1,c.oras2 alias stat

//Afisare fisierul rezultat din Select statistica
 *  use stat in 4
  form.browse1.alias='stat'  // Atasare fis stat la Browse

return

ENDCLASS
```