Programarea rapidă a aplicațiilor pentru baze de date relaționale

Lorentz JÄNTSCHI

Mădălina VĂLEANU Sorana BOLBOACĂ

AcademicDirect & Academic Pres

2006

Cuprins

I. Prefață

Lucrarea *Programarea rapidă a aplicațiilor pentru baze de date relaționale* este rezultatul experienței didactice de peste 10 ani în domeniul creării şi exploatării bazelor de date relaționale, a implementării de aplicații dedicate domeniilor cercetare științifică (baze de date pentru chimie şi medicină), biblioteconomie (baze de date cu publicații ştiințifice), management (gestiune şi contabilitate) şi nu în ultimul rând educație.

Lucrarea se adresează celor care doresc să-şi formeze priceperile şi deprinderile utilizării aplicațiilor dedicate pentru baze de date, şi în special a aplicațiilor din categoria *Rapid Application Development (RAD)* - în traducere *Dezvoltarea rapidă a aplicațiilor*. Din această categorie de aplicații, cea mai populară este FoxPro; aceasta este şi aplicația care este dezvoltată cel mai în amănunțime pe parcursul lucrării.

Din punctul de vedere al stilului programării, lucrarea oferă soluția *Microsoft*, fiind discutate trei aplicații dedicate bazelor de date oferite de firma *Microsoft*. În ordinea complexității mediului de programare oferit, acestea sunt: *Excel*, *Access* şi *Visual FoxPro*. În ceea ce priveşte versiunile aplicațiilor discutate, acestea nu sunt nici ultimele versiuni de pe piață, dar nici primele. Referindu-ne strict la Visual FoxPro, discuția este purtată exemplificând cu aplicații pentru versiunea 6 (an de apariție 1998), discuția fiind însă perfect valabilă şi pentru versiunea 5 şi pentru 7.

Axa demersului didactic al lucrării este orientată de la *problemă* către *soluție*, trecând prin *model matematic*, *algoritm* şi *implementare*.

Noțiunile teoretice fundamentale de baze de date nu lipsesc din demersul ştiințific al lucrării; dimpotrivă, sunt în amănunt discutate problemele de stocare atât la nivel fizic cât şi la nivel logic, problemele de consistență şi integritate sunt exemplificate pe parcursul lucrării, iar problemele fundamentale de securitate, coerență, restricții şi tranzacțiile sunt tratate datorită specificului aparte separat către sfârşitul lucrării (*Capitole speciale de baze de date*).

Lucrarea are un profund caracter formativ. Abilitățile care lucrarea doreşte a le forma sunt: proiectarea, implementarea, normalizarea şi asigurarea integrității referențiale unei baze de date relaționale şi implementarea aplicațiilor de gestiune; utilizarea controalelor puse la dispoziție de un mediu de programare vizual pentru dezvoltarea rapidă a aplicațiilor; realizarea de aplicații complexe client-server pentru gestiunea bazelor de date distribuite. Nu în ultimul rând se situează ingineria programării, tehnica de abordare a problemei propuse spre rezolvare de sus în jos şi de jos în sus (*top-down* şi *bottom-up* - în engleză) folosind binecunoscutul dicton latin *divide et impera* (*desparte şi stăpâneşte* - în traducere).

Cluj-Napoca, August 2006
Lorentz JÄNTSCHI

II. Despre autori

Sorana Daniela BOLBOACĂ

- s-a născut în 1973, în Cluj-Napoca, Cluj; a absolvit pe rând *Specializarea Medicină Generală*, în 1998, *Masterul de Informatică Medicală şi Biostatistică*, în 2001, *Facultatea de Medicină* la Universitatea de Medicină şi Farmacie I. Haţieganu Cluj-Napoca;
- din anul 2006, *Doctor în Ştiinţe Medicale*, *Specializarea Informatică Medicală*, la Universitatea de Medicină şi Farmacie I. Haţieganu Cluj-Napoca, cu Teza de Doctorat cu titlul *Practica Medicală Bazată pe Evidenţe: Logistică şi Implementare*, efectuată sub îndrumarea Prof. Univ. Dr. Andrei Achimaş Cadariu;
- *medic stagiar* şi apoi *medic rezident* (la Spitalul Clinic Judeţean Cluj-Napoca, 1999-2000-2005), *medic specialist* (radiologie şi imagistică medicală, din 2005), *asistent* şi apoi *şef de lucrări* (informatică medicală, 2001, 2006) la Universitatea de Medicină şi Farmacie I. Haţieganu Cluj-Napoca, *cercetător ştiinţific* (chimie, 2006) la Universitatea Tehnică din Cluj-Napoca.

Mădălina Ana VĂLEANU

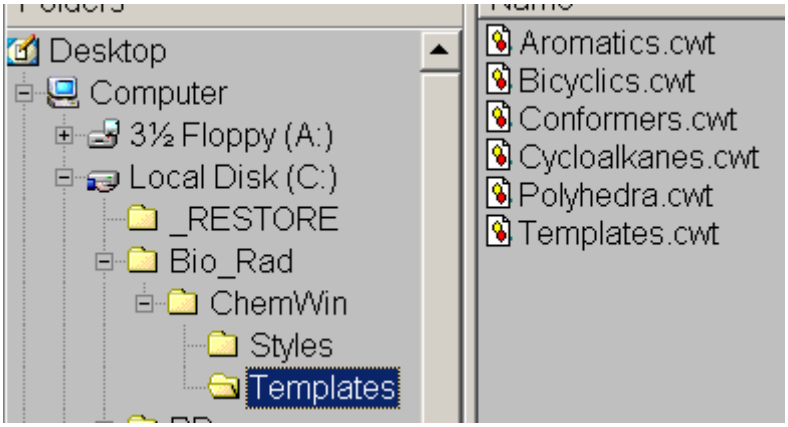
- s-a născut în 1972, în Sebeş, Alba; a absolvit pe rând *Specializarea Informatică*, în 1995, *Masterul de Informatică Distribuită*, în 1996, *Facultatea de Matematică şi Informatică* la Universitatea „Babeş-Bolyai" Cluj-Napoca;
- din anul 2004, *Doctor în Matematică*, *Specializarea Informatică*, la Universitatea „Babeş-Bolyai" Cluj-Napoca, cu Teza de Doctorat cu titlul *Problema integrităţii în baze de date distribuite*, efectuată sub îndrumarea Prof. Univ. Dr. Leon Ţâmbulea;
- analist programator (informatică, la S.C. Nethrom Software Cluj-Napoca, 1995-2001), asistent şi apoi şef de lucrări (informatică medicală, 2001, 2006) la Universitatea de Medicină şi Farmacie I. Haţieganu Cluj-Napoca.

Lorentz JÄNTSCHI

- s-a născut în 1973, în Făgăraş, Braşov; a absolvit pe rând *Specializarea Informatică*, *Facultatea de Matematică şi Informatică*, în 1995, *Specializarea Chimie - Fizică*, *Facultatea de Chimie şi Inginerie Chimică*, în 1997, la Universitatea „Babeş-Bolyai" Cluj-Napoca, *Masterul de Ameliorarea plantelor şi controlul calităţii seminţelor şi materialului săditor*, Facultatea de Agronomie, în 2002, la Universitatea de Ştiinţe Agricole şi Medicină Veterinară din Cluj-Napoca;
- din anul 2000, *Doctor în Chimie*, *Specializarea Chimie Organică*, la Universitatea „Babeş-Bolyai" Cluj-Napoca, cu Teza de Doctorat cu titlul *Predicţia proprietăţilor fizico-chimice şi biologice folosind descriptori matematici*, efectuată sub îndrumarea Prof. Univ. Dr. Mircea V. Diudea;
- *profesor de liceu* (informatică, titular la Colegiul Naţional G. Barițiu Cluj-Napoca, 1995-1999), *doctorand* (chimie organică, 1998-2000), *preparator* şi apoi *şef de lucrări* (chimie, 1999, 2000) la Universitatea Tehnică din Cluj-Napoca, *cadru didactic asociat* (informatică, 2000-2006) la Şcoala Academică Postuniversitară de Informatică Aplicată şi Programare de pe lângă UTCN;
- *a înfiinţat două reviste* (2002), şi *o editură* (2003), toate în formă electronică.

1. Baze de date şi SGBD

Calculatoarele au fost folosite încă din 1950 pentru *stocarea şi procesarea datelor*. Un deziderat major al *sistemelor informatice* este de a realiza produse software care să localizeze eficient datele pe suportul fizic şi să-l încarce rapid în memoria internă pentru procesare. La baza unui sistem informatic se află un *set de fişiere* memorate permanent pe unul sau mai multe suporturi fizice.



Gama largă de aplicații ale informaticii necesită acces rapid la mari cantități de date. Iată câteva exemple:

- sistemele computerizate de marcare din supermarketuri trebuie să traverseze întreaga linie de produse din magazin;
- sistemele de rezervare a locurilor la liniile aeriene sunt folosite în mai multe locuri simultan pentru a plasa pasageri la numeroase zboruri la date diferite;
- calculatoarele din biblioteci stochează milioane de intrări şi accesează citații din sute de publicații;
- sistemele de procesare a tranzacțiilor în bănci şi casele de brokeraj păstrează conturi care generează fluxul mondial de capital;
- motoarele de căutare World Wide Web scanează sute de pagini Web pentru a produce răspunsuri cantitative la interogări aproape instantaneu;
- sute de mici întreprinzători şi organizații utilizează bazele de date pentru a stoca orice de la inventare şi personal la secvențe ADN şi informații despre obiecte provenite din săpături arheologice.

Un produs software care presupune managementul fişierelor suportă descompunerea logică a unui fişier în *înregistrări*. Fiecare înregistrare descrie o entitate şi constă dintr-un număr de *câmpuri*, unde fiecare câmp dă valori unei anumite proprietăți (sau atribut) al entității.

Last_name	First_name	Acct_nbr	Address_1	City
Davis	Jennifer	1023495.0000	100 Cranberry St.	Wellesley
Jones	Arthur	2094056.0000	10 Hunnewell St	Los Altos
Parker	Debra	1209395.0000	74 South St	Atherton
Sawyer	Dave	3094095.0000	101 Oakland St	Los Altos
White	Cindy	1024034.0000	1 Wentworth Dr	Los Altos

Un fişier simplu cu înregistrări este adecvat pentru date comerciale cu complexitate redusă, cum ar fi inventarul dintr-un magazin sau o colecție de conturi curente pentru clienți.

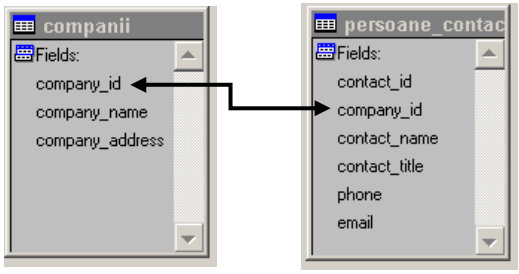
Un *index* al unui fişier constă dintr-o *listă de identificatori* (care disting înregistrările) *împreună cu adresele înregistrărilor*. De exemplu numele poate fi folosit pentru a identifica înregistrările unor persoane. Deoarece indexurile pot fi mari ele sunt uzual structurate într-o formă ierarhică şi sunt navigate cu ajutorul pointerilor. Formele ierarhice arborescente sunt frecvent folosite datorită vitezei mari de traversare.

Problemele reale ale procesării datelor solicită frecvent legarea datelor din mai multe fişiere. Astfel, în mod natural s-au conceput structuri de date şi programe de manipulare a datelor care să suporte legarea înregistrărilor din fişiere diferite.

3 modele de baze de date au fost create pentru a suporta legarea înregistrărilor de tipuri diferite:

- *modelul ierarhic*: tipurile înregistrărilor sunt legate într-o structură arborescentă (de exemplu înregistrările unor angajați s-ar putea grupa după o înregistrare care să descrie departamentele în care aceştia lucrează); IMS (Information Management System produs de IBM) este un exemplu de astfel de sistem;
- *modelul rețea*: se pot crea legături arbitrare între diferitele tipuri de înregistrări (de exemplu înregistrările unor angajați s-ar putea lega pe de o parte de o înregistrare care să descrie departamentele în care aceştia lucrează şi pe de altă parte supervizorii acestora care sunt de asemenea angajați);
- *modelul relațional*: în care toate datele sunt reprezentate într-o formă tabelată simplă.

În modelul relațional descrierea unei entități particulare este dată de setul valorilor atributelor, stocate sub forma unei linii în tabel şi numită relație. Această legare a n valori de atribute furnizează cea mai potrivită descriere a entităților din lumea reală.



Modelul relațional suportă *interogări* (cereri de informații) care implică mai multe tabele prin asigurarea unor legături între tabele (operația *join*) care combină înregistrări cu valori identice ale unor atribute ale acestora.

Statele de plată, de exemplu, pot fi stocate într-un tabel iar datele personalului beneficiar în altul. Informațiile complete pentru un angajat pot fi obținute prin reunirea acestor tabele (*join*) pe baza numărului personal de identificare.

Pentru a suporta o varietate de astfel de structuri de baze de date, o largă varietate a software denumită *sistem de gestiune a bazelor de date* este necesară pentru a stoca şi reda datele şi pentru a pune la dispoziția utilizatorului posibilitatea de a interoga şi actualiza baza de date.

Gestiunea datelor presupune o structurare a acestora realizată prin definirea bazelor de date. Pentru ca exploatarea bazelor de date să fie eficientă, e necesar ca acestea să aibă un grad înalt de abstractizare. Din punct de vedere practic, este normal să se definească mai multe nivele de abstractizare. Putem lua în considerare:

- *Nivelul fizic (sau intern).* La acest nivel se găsesc toate detaliile legate de reprezentarea datelor pe un suport de memorie;
- *Nivelul logic (sau conceptual).* Se ia in considerare aspectul semantic al datelor; contează conținutul efectiv al lor, precum şi relațiile (legăturile) dintre acestea; se descriu toate bazele de date folosind structuri relativ simple în funcție de necesitățile impuse de anumite aplicații;
- *Nivelul extern.* Acest nivel de abstractizare este cel în care se poate descrie conținutul unor baze de date; are în vedere simplificarea interacțiunii utilizator - bază de date.

Pentru descrierea bazelor de date facem apel la noțiunea de *structură* de date care reprezintă un ansamblu de instrumente conceptuale care permit descrierea datelor, a legăturilor dintre ele, semantica lor sau *constrângerile* la care ele sunt supuse.

Bazele de date evoluează în timp. Mulțimea informațiilor conținute în baza de date la un moment dat defineşte *instanțierea* bazei de date.

În 1970, Ted Codd (IBM, părintele SQL), nemulțumit de performanțele COBOL şi IMS formulează principiul de lucru al bazelor de date relaționale. Codd afirmă că SGBD trebuie să recunoască comenzi simple şi trebuie să fie aproape de utilizatori prin punerea împreună a comenzilor potrivite pentru găsirea a ceea ce se doreşte. Ceea ce Codd numeşte model relațional se bazează pe două puncte cheie:

- să furnizeze un mod de descriere a datelor cu numai cu structura lor naturală, ceea ce înseamnă că trebuie realizat acest lucru fără impunerea nici unei structuri adiționale pentru scopuri de reprezentare în calculator;

- de asemenea, să furnizeze baza pentru un limbaj de date de nivel înalt care va conduce la o maximă independență între programe, pe de o parte, şi reprezentarea în calculator, pe de altă parte.

O bază de date relațională extinde conceptul de tabele; este compusă dintr-o mulțime de tabele între care se definesc relații în sens matematic.

Să presupunem că avem T_1, T_2, ..., T_m m *tabele* într-o *bază de date*. Fiecare dintre aceste tabele are o *structură* ($T_i = \{C_{i0}, C_{i1}, ...\}$) ce conține *câmpuri* ($C_{ij}$). Pentru a defini *relații* ($R \subseteq T_1 \times ... T_m$) între aceste tabele, este necesar ca cel puțin un câmp din fiecare tabelă să suporte o *relație de ordine strictă* (nota bene: nu e necesară existența logică a acestei construcții; ea se poate construi şi din structura fizică a informației din tabele, cum ar fi numărul înregistrării). Fie aceste câmpuri C_{i0}. Asta înseamnă că *valorile* (v_{i0k}, k=1,...) din înregistrările corespunzătoare acestor câmpuri C_{i0} sunt *ordonate strict* ($v_{i01} < v_{i02} < ...$). Nota bene: nu e necesar ca relația de ordonare strictă să fie strict crescătoare, cum nu e necesar ca valorile v_{i01}, v_{i02}, ... să fie stocate în înregistrări consecutive; este necesară doar existența relației de ordine strictă, care să permită referirea individuală a fiecărei valori, şi prin aceasta, identificarea în mod unic a fiecărei înregistrări k: (v_{i0k}, v_{i1k}, ...). Relația R între tabele este în fapt o submulțime a $C_{10} \times C_{20} \times ... C_{m0}$. Reprezentarea figurativă a relației R este:

R	C_{10}	...	C_{m0}
r_1	c_{101}	...	c_{m01}
...	...	...	...
r_n	c_{10n}	...	c_{m0n}

În mod uzual, pentru mulțimea $T_1 \times ... T_m$ se foloseşte noțiunea de univers (U). Elementele universului U se numesc atribute. Câmpurile C_{i0} se notează (pentru simplitate) A_i. Mulțimea valorilor atributelor A_i (v_{i0k}, $k \geq 1$) se notează cu D_i. Elementele relației r_1,..., r_n se numesc tuple şi se notează cu t_1, ..., t_n. Folosind aceste notații, relația R devine:

R	A_1/D_1	...	A_m/D_m
t_1	a_{11}	...	a_{1m}
...	...	...	...
t_n	c_{n1}	...	a_{nm}

Coloanele acestui tablou se identifică prin atributele A_i şi domeniile corespunzătoare D_i, scriind A_i/D_i ($1 \leq i \leq m$). Mulțimea ordonată a atributelor $A = A_1, ..., A_m$ care definesc relația R se numeşte *schemă relațională*. Facem distincție între *schema relațională* A şi instanțierea acesteia (t_1, ... t_n). Convenim să notăm relația R de schemă *A*, sub forma: r(*A*) sau r(A_1, A_2, …, A_m). Dacă luam în considerare tuplul t_i care defineşte linia i din tabloul R de mai

sus, adică $t_i \Leftrightarrow a_{i1}...a_{im}$, convenim ca să folosim aceeaşi notație t_i pentru $t_i = (a_{i1}, ..., a_{im}) \in D_1 \times ... D_m$. Convenim, de asemenea să notăm $t_i[A_j] = a_{ij} \in D_j$, $1 \leq i \leq n$, $1 \leq j \leq m$. De asemenea, dacă avem $K = (A_{j1}, A_{j2}, ..., A_{jk})$, $k \leq m$, atunci $t_i[K] = (a_{ij_1}, a_{ij_2}, ..., a_{ij_k})$.

Într-o bază de date relațională noțiunea de cheie are un rol important. Numim cheie a unei relații R de schemă A, adică r(A), o *submulțime minimală* K, $K \subseteq A$ cu proprietatea că $t_i[K] \neq t_j[K]$, pentru $i \neq j$, $1 \leq i \leq n$, $1 \leq j \leq m$.

Recapitulând, principalele concepte utilizate la descrierea logică (conceptuală), respectiv formală, apoi uzuală şi fizică a elementelor de organizare a datelor sunt:

formal	*uzual*	*fizică*
relație	tablou	fişier
tuplu	linie	înregistrare
atribut	coloană	câmp
domeniu	tip de dată	tip de dată

Cu alte cuvinte modelul relațional consistă din:

- independența datelor față de hardware şi modul de memorare;
- navigarea automată sau un limbaj de nivel înalt neprocedural pentru accesarea datelor;

În loc ca să se proceseze câte o înregistrare, programatorul utilizează limbajul pentru a specifica operații unice care trebuie realizate asupra întregului set de date.

Limbajele de generația a 4-a (4^{th} GLs) sunt mai aproape de limbajul uman ca limbajele de nivel înalt (de generația a 3-a, 3^{th} GLs). Primele dintre acestea sunt FOCUS (Information Builders) SQL (IBM), QBE (Querry by example, IBM), dBASE (succesorul lui SQL).

Necesitatea pentru mai multă flexibilitate şi performanță din partea modelelor de date cum ar fi de a suporta aplicațiile ştiințifice sau inginereşti a făcut ca să se extindă conceptul de model relațional aşa încât intrările în tabele să nu mai fie simple valori ci să poată fi programe, texte, date nestructurate mari în formă binară sau orice alt format solicitat de utilizator. Un alt progres s-a făcut prin încorporarea conceptului de *obiect* devenit esențial în limbajele de programare. În bazele de *date orientate obiect* toate datele sunt obiecte. Obiecte se pot lega între ele printr-o *relație de apartenență* pentru a forma o familie mai largă şi mai diversă de obiecte (în anii '90 au fost lansate primele sisteme de management orientat obiect OODMS). Datele care descriu un transport pot fi stocate, de exemplu, ca familie mai largă care poate conține automobile, vapoare, vagoane, avioane. Clasele de obiecte pot forma *ierarhii* în care obiecte individuale pot moşteni proprietăți de la obiectele situate deasupra în ierarhie. Bazele de date multimedia, în care vocea, muzica şi informația video se stochează

împreună cu informațiile de tip text, devin tot mai frecvente şi îşi imprimă trendul în dezvoltarea sistemelor de gestiune a bazelor de date orientate obiect.

O secvență tipică pentru un limbaj 4^{th} GL este:

FIND ALL RECORDS WHERE NAME IS "TUCKER"

SQL (Structured Query Language) este un limbaj standard industrial pentru crearea, actualizarea şi interogarea sistemelor de management ale bazelor de date relaționale.

Prima versiune standardizată a SQL a apărut în 1986 şi conține construcțiile de bază ale limbajului pentru definirea şi manipularea tabelelor de date. O revizie în 1989 a adăugat limbajului extensii pentru integritatea referențială şi generalizează constrângerile de integritate. O altă extensie în 1992 furnizează facilități în manipularea schemelor şi administrarea datelor şi de asemenea substanțiale îmbunătățiri în ceea ce priveşte definirea şi manipularea datelor. Dezvoltarea sistemului este în desfăşurare pentru a face din acesta un limbaj computațional complet pentru definirea şi managementul obiectelor complexe persistente. Aceasta include generalizarea şi specializarea ierarhiilor, moştenire multiplă, tipuri de dată utilizator, generatoare şi construcții declarative, suport pentru sistemele bazate pe cunoştințe, expresii interogative recursive şi instrumente adiționale de administrare a datelor. Include de asemenea tipuri abstracte de date, identificatori de obiecte, metode, moştenire, polimorfism, încapsulare şi toate celelalte facilități care sunt asociate uzual cu managementul datelor de tip obiect.

În prezent, industria bazelor de date, ca segment al industriei de software generează anual aproximativ 8 miliarde de dolari. Companiile care dețin supremația pe acest segment de piață sunt IBM, Oracle, Informix, Sybase, Teradata (NCR), Microsoft, Borland.

2. Formele Backus-Naur

BNF (Backus-Naur Form, numite originar Backus Normal Form şi redenumite apoi la sugestia lui Donald Knuth) formează o metasintaxă utilizată pentru a exprima gramatici independente de context. BNF este unul dintre cele mai utilizate notații metasintactice pentru specificarea sintaxei limbajelor de programare şi seturile de comenzi ale acestora. pentru detalii suplimentare vezi „http://src.doc.ic.ac.uk/computing/internet/rfc/rfc2234.txt”.

Fie o formă BNF a unei adrese poştale din U.S.:

```
<postal-address> ::= <name-part> <street-address> <zip-part>
<personal-part> ::= <name> | <initial> "."
<name-part> ::= <personal-part> <last-name> [<jr-part>] <EOL>
              | <personal-part> <name-part>
<street-address> ::= [<apt>] <house-num> <street-name> <EOL>
<zip-part> ::= <town-name> "," <state-code> <ZIP-code> <EOL>
```

Aceasta se traduce prin: „O adresă poştală constă dintr-o parte de nume, urmată de o parte de adresă stradală şi urmată de o parte de cod poştal. O parte personală constă din prenume sau dintr-o inițială urmată de un punct. O parte de nume constă din următoarele: o parte personală urmată de nume urmat de un opțional <jr-part> (Jr., Sr., sau numărul dinastiei) şi sfârşit de linie sau o parte personală urmată de o parte de nume (aceasta ilustrează recursivitatea în formele BN, acoperind cazul persoanelor care folosesc mai multe nume sau prenume şi/sau inițiale). O adresă stradală constă dintr-un specificator opțional de apartament, urmat de număr şi numele străzii. Partea de cod poştal constă din numele oraşului, urmat de virgulă, urmat de codul statului şi oraşului urmat de sfârşit de linie.”

De observat că multe lucruri (cum ar fi formatul părții personale, specificatorul de apartament sau de codul oraşului au rămas nespecificate. Aceste detalii lexicale sunt presupuse evidente din context sau sunt specificate în altă parte.

Sunt multe variante şi extensii ale BNF, de exemplu prin introducerea wildcardurilor ? şi *. Două dintre acestea sunt EBNF şi ABNF.

3. Baze de date relaționale

Fie un simplu exemplu de carte de adrese, nimic prea complex, doar ceva care memorează nume, adrese, numere de telefon, emailuri şi atât. Să memorăm acum astfel de informații într-un fişier text cu delimitatori. Dacă prima linie serveşte ca cap de tabel şi virgula este folosită ca separator, acest fişier ar putea arăta ca mai jos:

```
Name, Addr1, Addr2, City, State, Zip, Phone, E-mail
Jay Greenspan, 211 Some St, Apt 2, San Francisco, CA, 94107,
4155551212, jgreen_1@yahoo.com
Brad Bulger, 411 Some St, Apt 6, San Francisco, CA, 94109,
4155552222, bbulger@yahoo.com
John Doe, 444 Madison Ave, , New York, NY, 11234, 2125556666,
nobody@hotmail.com
```

Nu este mult de văzut la acesta, însă este cel puțin încărcabil în calculator. Utilizând orice limbaj de programare (din generația a 3-a) se poate scrie un cod care să deschidă acest fişier şi să preia informația. Oricum am pune problema însă în implementarea acestui cod, se arată a fi o bună bucată de cod de scris. Dacă se doreşte ca această informație să se poată ordona şi interoga după o varietate de criterii, care de exemplu să sorteze alfabetic după nume sau să găsească toți oamenii care au o anumită valoare a codului de localitate, este într-adevăr dureros. Ne putem lovi acum de o altă problemă majoră dacă datele dorim să fie utilizate în cadrul unei rețele de o mulțime de utilizatori, cum ar fi modificările pe care le-ar efectua un utilizator în timp ce alt utilizator se poziționează pe o înregistrare mai jos în fişier. Probabil va trebui să blocăm la scriere fişierul atunci când mai mulți utilizatori îl accesează.

Este evident deci că soluția memorării acestuia sub formă de fişier text nu a fost fericită. Este nevoie de *un sistem de stocare care să reducă cantitatea de prelucrări şi accesări* ale informației din fişier de către programul responsabil cu gestiunea acestuia. Un tabel simplu ca tabelul 1 ar trebui să funcționeze tocmai bine:

Tabelul 1. Tabel simplu pentru stocarea datelor

name	addr1	addr2	city	state	zip	phone	e-mail
Jay Greenspan	211 Some St	Apt 2	San Francisco	CA	94107	4155551212	jgreen_1@yahoo.com
Brad Bulger	411 Some St	Apt 6	San Francisco	CA	94109	4155552222	bbulger@yahoo.com
John Doe	444 Madison Ave		New York	NY	11234	2125556666	nobody@hotmail.com

Acum acesta este aproape convenabil. Este uşor de ajuns la sfârşit şi de verificat dacă vreun program accesează acest tabel. Este uşor de accesat o linie din acest tabel odată fără a afecta pe ceilalți. În acest sens, dacă 2 sau mai mulți utilizatori doresc să insereze informații în această tabelă ei nu se vor suprapune în acțiunea lor. Dacă se va dori extragerea unor informații din tabel, cum ar fi toate persoanele care sunt din California, nu va fi necesar să se

prelucreze şi ordoneze fişierul. Un program care ar opera pe această tabelă va trebui doar să rezolve următoarea subproblemă: *afişează toate liniile la care conţinutul coloanei **State** este egal cu **'CA'***. Da, este frumos, însă nu este destul.

Obiectivul Dr. Codd, părintele SQL, a fost de a avea un model al informației care să nu creeze anomalii. Se pot identifica 3 situații de anomalie: la Actualizare, Ştergere şi Inserare.

Să presupunem că o structură tabelată poate rapid şi uşor gestiona cereri multiple şi să analiză ce se întâmplă dacă informația devine ceva mai complexă, cum ar fi cazul ilustrat în tabelul 2:

Tabelul 2. Tabelă cu stocare problematică

id	company_name	company_address	contact_name	contact_title	phone	email
1	BigCo Company	1121 43rd St	Jay Greenspan	Vice President	4155551212	jgreen_1@yahoo.com
2	BigCo Company	1121 43rd St	Brad Bulger	President	4155552222	bbulger@yahoo.com
3	LittleCo Company	4444 44th St	John Doe	Lackey	2125556666	nobody@hotmail.com

Ce se întâmplă dacă de exemplu, firma BigCo se hotărăşte să-şi schimbe sediul? Va trebui să actualizăm adresa sa în două linii. Poate fi şi o sursă de erori dacă modificarea se face manual în fiecare linie şi cineva introduce greşit una din cele două noi adrese. Rezultă deci că o cale mai bună de a manipula aceste date este de a lua numele companiei şi adresa acesteia şi a le pune într-o tabelă separată. Rezultatul poate fi ca în tabelele 3 şi 4.

Tabelul 3. Companii

company_id	company_name	company_address
1	BigCo Company	1121 43rd St
2	LittleCo Company	4444 44th St

Tabelul 4. Persoane de contact

contact_id	company_id	contact_name	contact_title	phone	email
1	1	Jay Greenspan	Vice President	4155551212	jgreen_1@yahoo.com
2	1	Brad Bulger	President	4155552222	bbulger@yahoo.com
3	2	John Doe	Lackey	2125556666	nobody@hotmail.com

Ceea ce s-a realizat în acest caz prin separarea celor două categorii de informații şi introducerea unei coloane de legătură (company_id) este că s-a creat o *relație între cele două tabele* şi de aici vine numele de bază de date relațională.

Deşi avem exact aceleaşi informații ca la început, totuşi, există o diferență, faptul că tocmai le-am segmentat. Putem acum schimba adresa atât pentru Jay cât şi pentru Brad prin modificarea unei singure linii. Acesta este un fapt convenabil, oricum.

Să presupunem că se întâmplă ca d-nul Doe să fie şters din baza de date în forma din tabelul 2. Dar se poate întâmpla ca cineva să vrea să ceară lista tuturor companiilor cu care ai avut contact anul trecut. În forma curentă, ştergându-l pe Doe, vom şterge şi informația despre companie odată cu el. Această problemă se numeşte ştergerea anormală.

Tabelul 2. Tabelă cu ştergere anormală

id	company_name	company_address	contact_name	contact_title	phone	email
1	BigCo Company	1121 43rd St	Jay Greenspan	Vice President	4155551212	jgreen_1@yahoo.com
2	BigCo Company	1121 43rd St	Brad Bulger	President	4155552222	bbulger@yahoo.com
3	LittleCo Company	4444 44th St	John Doe	Lackey	2125556666	nobody@hotmail.com

Dacă însă se păstrează structura din tabelele 3 şi 4, se poate face ştergerea numai din tabela 4, şi înregistrarea cu compania poate să rămână în tabela 3, aşa încât în acest caz problema ştergerii anormale nu mai există.

Privind din nou datele din tabelul 2 putem observa că scopul principal al acestei tabele este de a stoca contacte şi nu companii. Situația devine paradoxală atunci când dorim să inserăm o companie dar nu şi persoana de contact. De cele mai multe ori, ar trebui să aşteptăm până când avem date specifice de contact pentru ca să putem adăuga în baza de date. Este evident o restricție ridicolă.

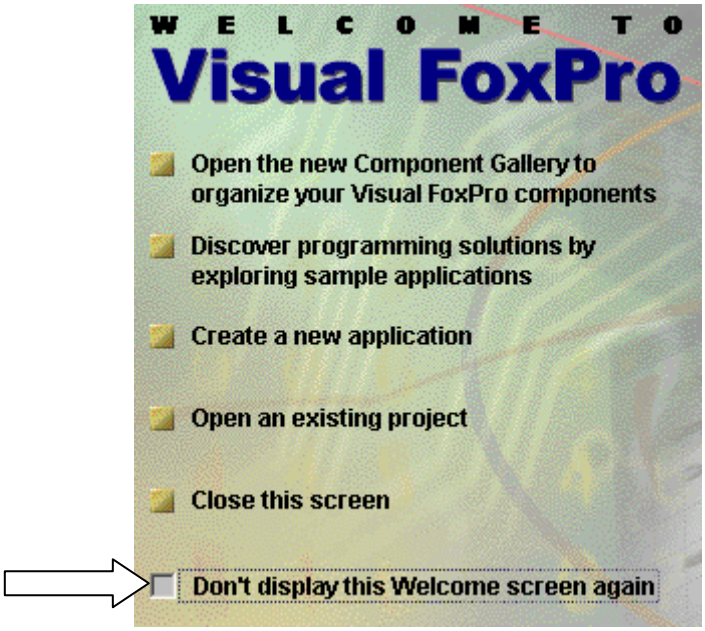
4. Microsoft Visual FoxPro

Microsoft Visual FoxPro face parte din pachetul Microsoft Visual Studio distribuit de firma Microsoft (TM).

Începând cu versiunea 6.0, Microsoft Visual Studio este un pachet integrat, care conține Visual Basic, Visual C++, Visual FoxPro, InterDev, Visual J++, SourceSafe şi o bogată documentație denumită MSDN (Microsoft Developer Network Library).

Lansarea aplicației în execuție se poate face din Taskbar, Start→Programs→Microsoft Visual Studio 6.0→Microsoft Visual FoxPro 6.0 sau dacă a fost instalat cu opțiunile implicite de instalare din locația: C:\Program Files\Microsoft Visual Studio\Vfp98\VFP6.EXE".

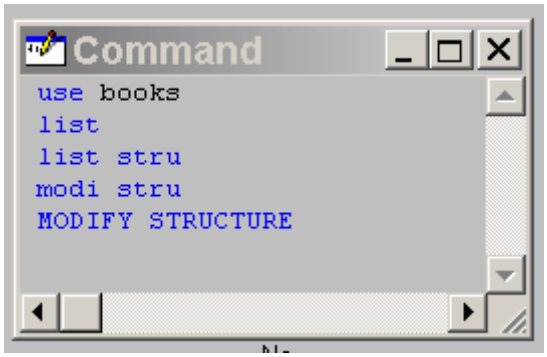
La pornire se activează un Wizard opțional:



care se poate dezactiva prin marcarea Checkbox-ului.

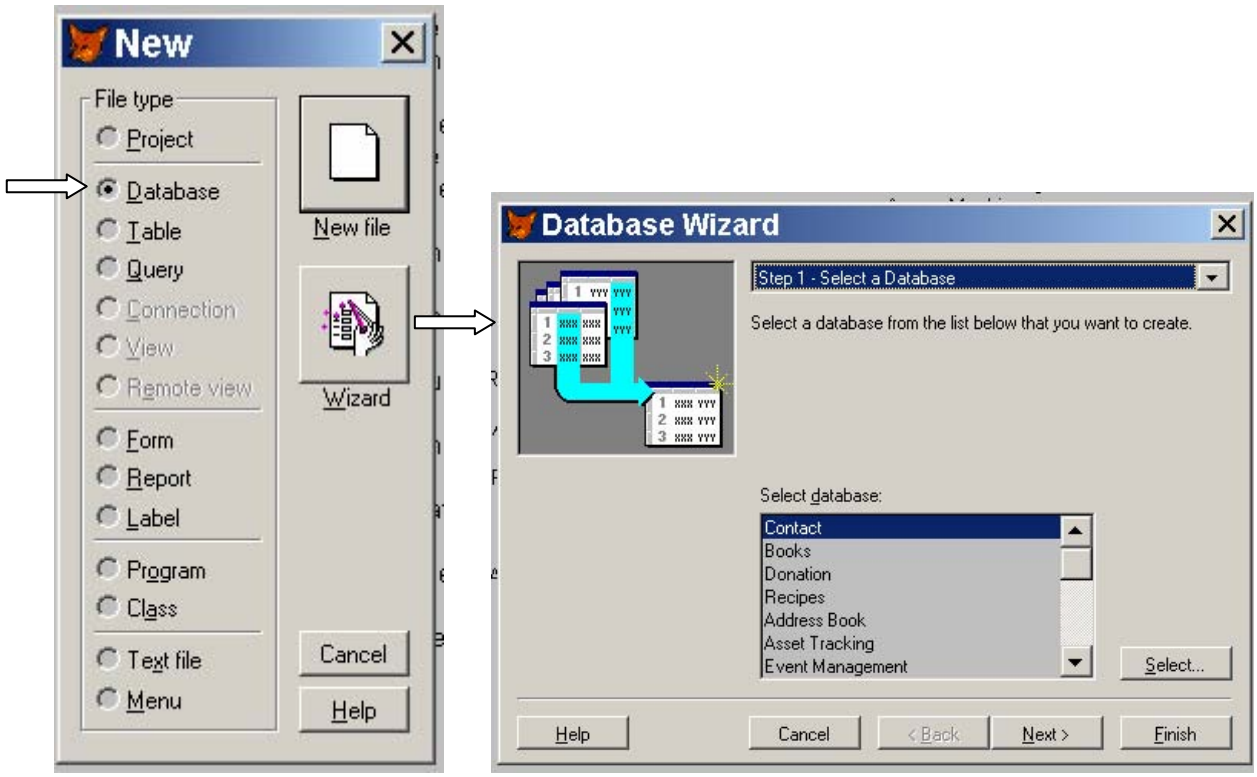
Pentru a ajunge la meniul aplicației se alege opțiunea **Close this screen**. Cu ajutorul barei de instrumente se pot crea sau deschide baze de date, tabele, interogări, forme, rapoarte, etichete, programe, clase sau proiecte. Oricare din operațiile efectuate cu ajutorul barei de instrumente se pot efectua şi din fereastra de comenzi; ea se activează/dezactivează ca în fig.

Operațiile asupra bazelor de date create pe care le efectuăm din bara de instrumente sunt oricum înregistrate în fereastra de comenzi, aceasta păstrând istoria activității sesiunii de lucru curente:



5. Crearea unei baze de date

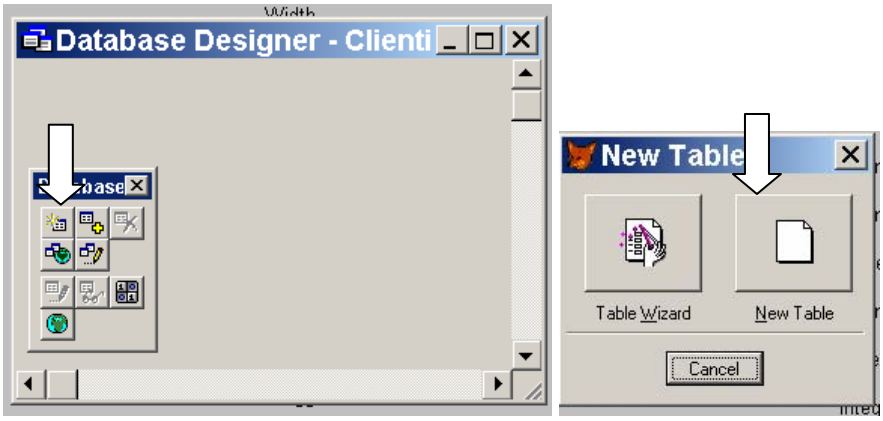
Cu ajutorul butonului New se activează o fereastră cu butoane radio ca în figură:



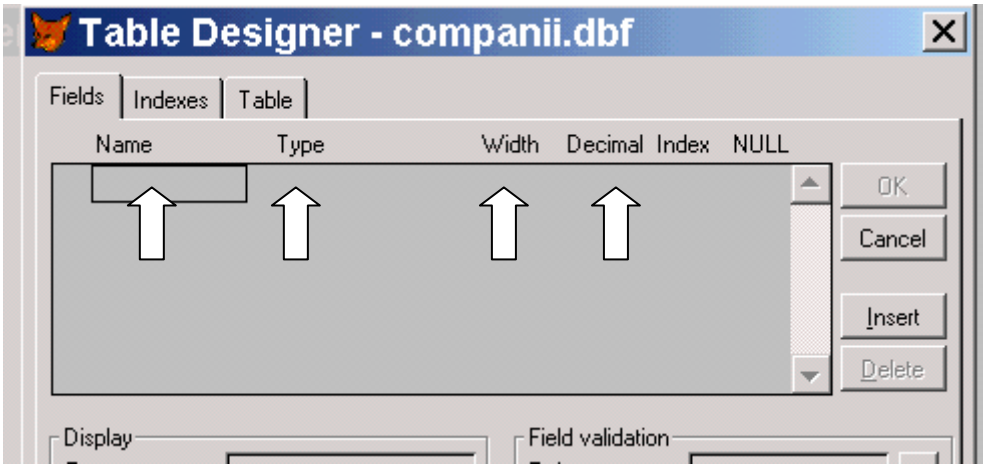
Se poate alege a se crea o bază de date cu ajutorul wizardului ca în figură în care într-o succesiune de 5 paşi se precizează caracteristicile noii baza de date care se doreşte să se creeze.

Baza de date aşa cum s-a putut constata în exemplul prezentat anterior este compusă din mai multe tabele. Să presupunem că vrem să memorăm informațiile prezentate în tabelele 3 şi 4. În acest caz, putem alege să construim o nouă bază de date pe baza creări unui nou fişier în care apoi să adăugăm cele două tabele, şi anume COMPANII şi PERSOANE_CONTACT.

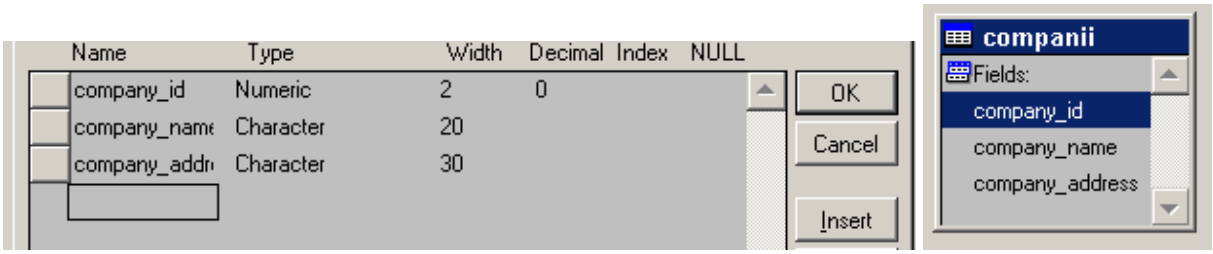
Alegem deci New/Database, New File, introducem numele noii baze de date (să spunem CLIENTI când se generează următorul rezultat:



În Această coală avem posibilitatea să construim structura celor două tabele (COMPANII şi PERSOANE_CONTACT). Activarea consecutivă a butoanelor New Table şi apoi din nou New Table urmat de introducerea numelui primei tabele (COMPANII) duce la activarea constructorului de tabele:

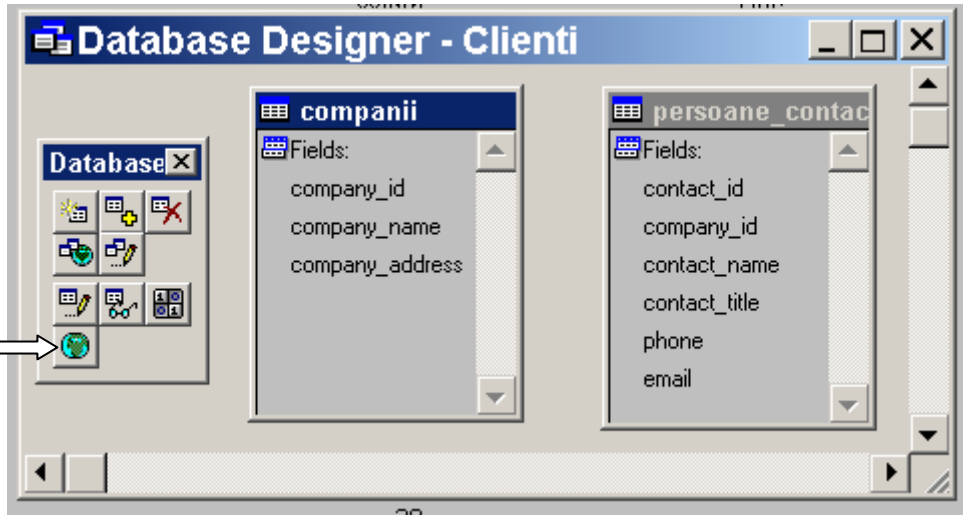


Se precizează succesiv în această tabelă numele, tipul, lungimea şi precizia zecimală acolo unde este cazul pentru fiecare câmp al bazei de date, până când se obține un rezultat ca în figura următoare:



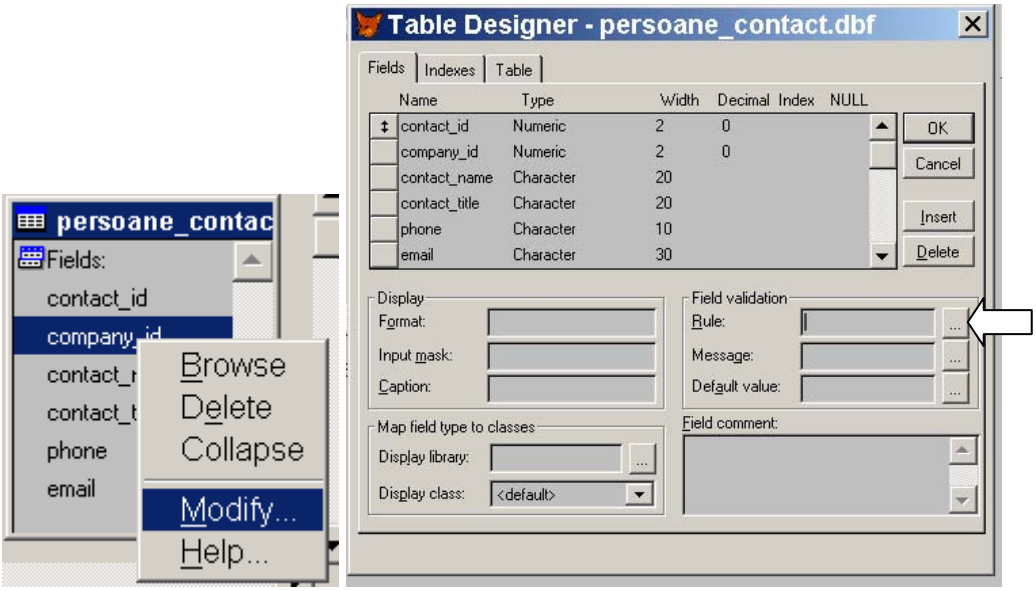
Se va acționa butonul Ok şi în acest moment se va adăuga automat pe foaia de lucru a bazei de date desenul din dreapta care reprezintă tabela creată.

Se repetă procedura şi pentru cea de-a doua bază de date, când se obține:

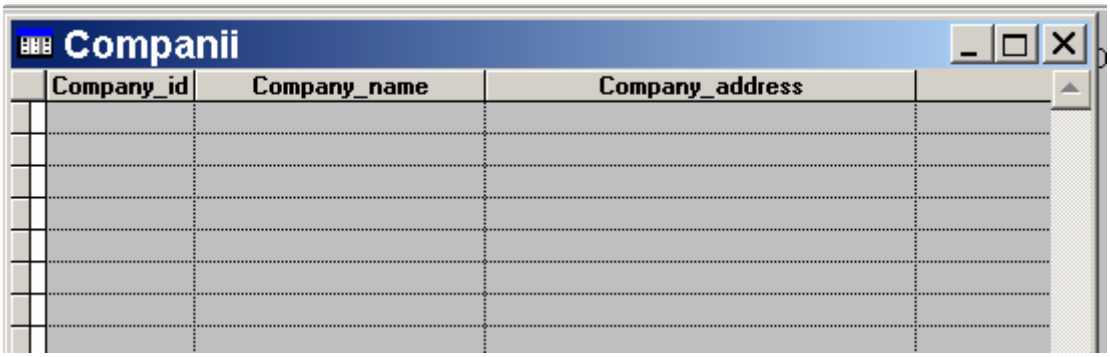


Se poate stabili acum o relație de validare la completare. Se activează cu click dreapta ca în figură proprietățile câmpului contact_id din persoane_contact şi din fereastra de modify se stabileşte regula:

Companii.company_id > 0



Se completează apoi bazele de date cu informațiile necesare prin dublu click asupra tabelelor, când se activează ferestre de tip browse în care se adaugă câte o înregistrare cu CTRL+Y sau Table/Append New Record:



După aceasta oricând ne putem întoarce la foaia tabelei pentru a vizualiza conținutul tabelelor sau pentru a face adăugări sau modificări.

6. Normalizarea unei baze de date

Aşa cum s-a putut vedea în exemplul prezentat, crearea unei baze de date relaționale presupune identificarea tuturor relațiilor între atributele entităților care sunt stocate sau se vor stoca în baza de date. Se poate ca aceste relații să se identifice după ce structura bazei de date a fost creată. Oricum, procesul prin care se elimină cele 3 anomalii (la modificare, la ştergere şi la adăugare) se numeşte *normalizare*. Înțelegerea normalizării este vitală pentru lucrul cu *baze de date relaționale*.

Normalizarea nu este un proces cu care se începe sau se termină designul bazei de date. Este un proces care se aplică oricând se identifică anomalii. Experiența şi instinctul totdeauna joacă un rol important în crearea unei bune baze de date.

Normalizarea se poate realiza prin trecerea succesivă a datelor prin câteva *forme normale*. Până în prezent s-au stabilit 7 astfel de forme normale, dintre care primele 3 asigură o calitate destul de bună a organizării relaționale a bazei de date şi majoritatea bazelor de date relaționale sunt organizate până la această formă.

Prima formă normală a datelor necesită ca:

- datele să fie structurate într-un tabel;
- fiecare coloană trebuie să conțină o singură valoare de un singur tip, adică să existe o singură valoare în fiecare celulă; nu sunt permise şiruri sau alte forme de reprezentare a mai mult de o valoare pe celulă;
- fiecare coloană trebuie să aibă un nume unic;
- tabelul trebuie să aibă un set de valori care identifică în mod unic o linie; valorile din această coloană se numesc *chei primare* pentru tabel;
- nu trebuie să existe două linii identice în tabel;
- nu sunt permise grupuri repetitive de date;

Ultima afirmație necesită explicații. Fie datele din tabelul 5:

Tabelul 5. Tabel cu grupuri repetitive de date

company_id	company_ name	company_ address	contact_ name	contact_ title	phone	email
1	BigCo Company	1121 43rd St	Jay Greenspan	Vice President	4155551212	jgreen_1@yahoo.com
2	BigCo Company	1121 43rd St	Brad Bulger	President	4155552222	bbulger@yahoo.com
3	LittleCo Company	4444 44th St	John Doe	Lackey	2125556666	nobody@hotmail.com

Cum se observă, zona marcată conține informații identice. Ea formează un grup repetitiv. După ce vom înlătura aceste coloane şi le plasăm în propriul lor tabel se ajunge la prima formă normală.

Cheile primare sunt o coloană sau un set de coloane care au pe fiecare linie o valoare unică în şirul valorilor coloanei respective. În tabelul 5 se poate vedea cum s-a inclus o astfel de coloană (*company_id*). Toate browserele de baze de date posedă un instrument de a defini o astfel de coloană. De exemplu, în MySQL aceasta se numeşte câmp auto_increment. Pot însă fi *chei primare* seriile de buletin, adresele email sau URL-urile. Singura condiție este ca datele să fie unice.

De exemplu, dacă informațiile de contact dintr-o astfel de agendă de adrese presupun memorarea de informații pentru companii cu mai multe reprezentanțe, probabil cea mai bună soluție este de a memora identificatorul reprezentanței şi adresa într-un tabel separat în care *company_id* şi eventual *company_city* să formeze cheia primară.

Dependența datelor este un element esențial în organizarea lor relațională. O coloană este *dependentă* (de cheia primară) dacă ea nu poate exista în tabel când cheia primară este înlăturată.

A doua formă normală intervine când la sfârşitul primei normalizări obținem o cheie primară formată din mai multe coloane. Să presupunem că în urma procesului de separare a tabelei de adrese (v. tabelul 5) obținem o tabelă în forma:

Tabelul 6. Tabel care nu e în forma normală 2

company_name	company_location	company_ceo	company_address
BigCo Company	San Francisco	Bill Hurt	1121 43rd St
LittleCo Company	LA	LittleCo Company	4444 44th st

Aici, company_name şi company_location formează cheia primară multiplă.

O *adăugare anormală* se produce atunci când dorim să adăugăm o nouă adresă pentru BigCo Co. Vom avea numele CEO (Chief Executive Officer) repetat în linia adăugată.

Transformăm acest tabel în a doua formă normală prin eliminarea liniilor care sunt doar parțial dependente de cheia primară. CEO este dependent doar de coloana *company_name* şi nu este dependent de coloana *company_location*. Pentru a ajunge la a 2-a formă normală, se mută liniile care sunt doar parțial dependente de cheia primară multi-coloană într-un tabel propriu.

A doua formă normală nu se aplică pentru tabelele care au o cheie primară formată de o singură coloană.

A 3-a formă normală rezolvă *dependențele tranzitive*. O *dependență tranzitivă* este atunci când o coloană există în tabel dar nu este condiționată direct de cheia primară. În loc de aceasta, ea este condiționată de alte câmpuri, care la rândul lor sunt condiționate de cheia primară.

O cale rapidă de a se ajunge la a 3-a formă normală este de a ne uita la toate câmpurile din tabel şi de a ne întreba dacă aceste câmpuri descriu cheia primară. Dacă nu, locul lor nu este aici (în această tabelă). Dacă se doreşte ca agenda să memoreze mai multe informații de contact, ca în tabela următoare:

Tabelul 7. Tabel care nu e în a 3-a formă normală

contact_id	contact_ name	contact_ phone	assistant_ name	assistant_ phone
1	Bill Jones	4155555555	John Bills	2025554444
2	Carol Shaw	2015556666	Shawn Carlo	6505556666

atunci se pune problema dacă nu cumva prin adăugarea acestora nu se alterează normalizarea acesteia. Este posibil, chiar probabil ca un asistent să deservească mai multe reprezentanțe, ceea ce face ca *assistant_name* şi *assistant_phone* să apară în tabel mai mult decât odată. Acestea vor forma atunci un *grup repetitiv*, care deja a fost discutat cum se elimină.

7. Tipuri de relații

Este esențial a se crea un grup de tabele care să nu aibă anomalii. Acestea totdeauna includ coloane care mențin (definesc) relațiile între aceste tabele. Sunt 3 tipuri de relații în domeniul bazelor de date:

- *relații 1 la n*: de departe cel mai frecvent caz, când o valoare dintr-o coloană referă mai multe câmpuri într-un alt tabel:

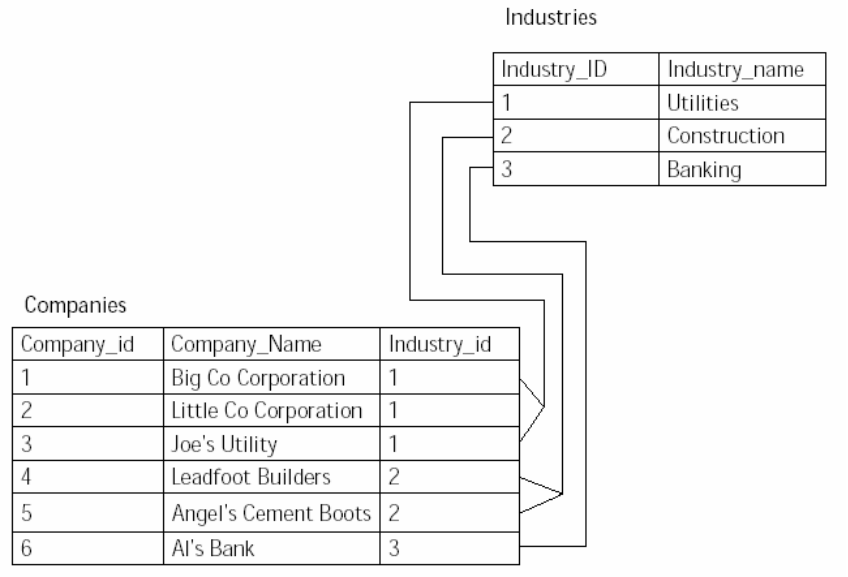
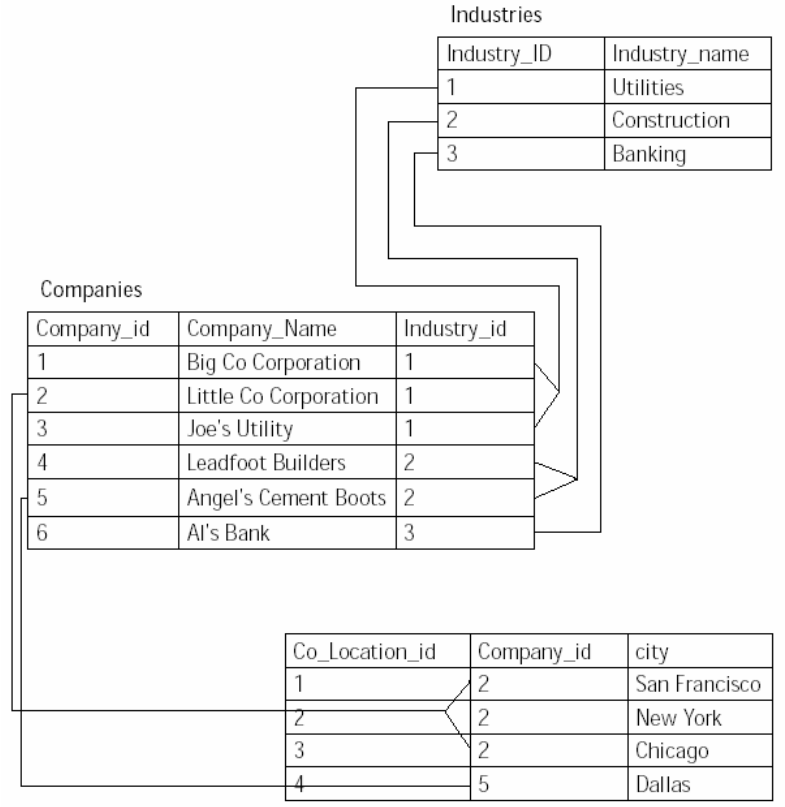
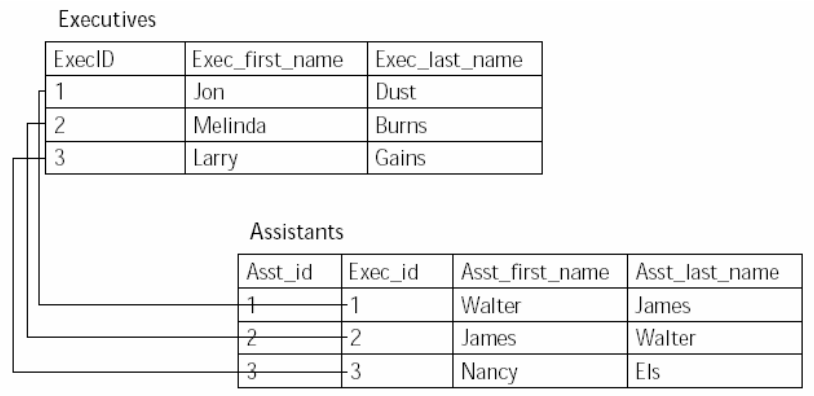


Figura următoare arată cum tabela de Companii poate fi unul din capetele încă unei relații 1 la n cu tabela reprezentanțelor:



- *relații 1 la 1*: sunt de fapt relații 1 la n în care o linie dintr-un tabel este legată cu o linie dintr-un alt tabel; ele rezultă de obicei în urma proceselor de normalizare, aşa cum ar putea rezulta în urma separării datelor din tabelul 7:



- *relații m la n*: acestea lucrează diferit față de cele două anterioare; să presupunem că compania păstrează informații despre o varietate de publicații care pot fi distribuite persoanelor de contact, în funcție de cererea acestora; pentru început, se creează tabela care va memora tipurile de publicații:

Tabelul 8. Tabela publicațiilor

newsletter_id	newsletter_name
1	Weekly
2	Monthly
3	Bi-monthly
4	Annual

Se poate acum adăuga o coloană în tabelul de persoane de contact pentru a preciza publicațiile care se vor trimite, ca în tabelul 9:

Tabelul 9. Persoane de contact

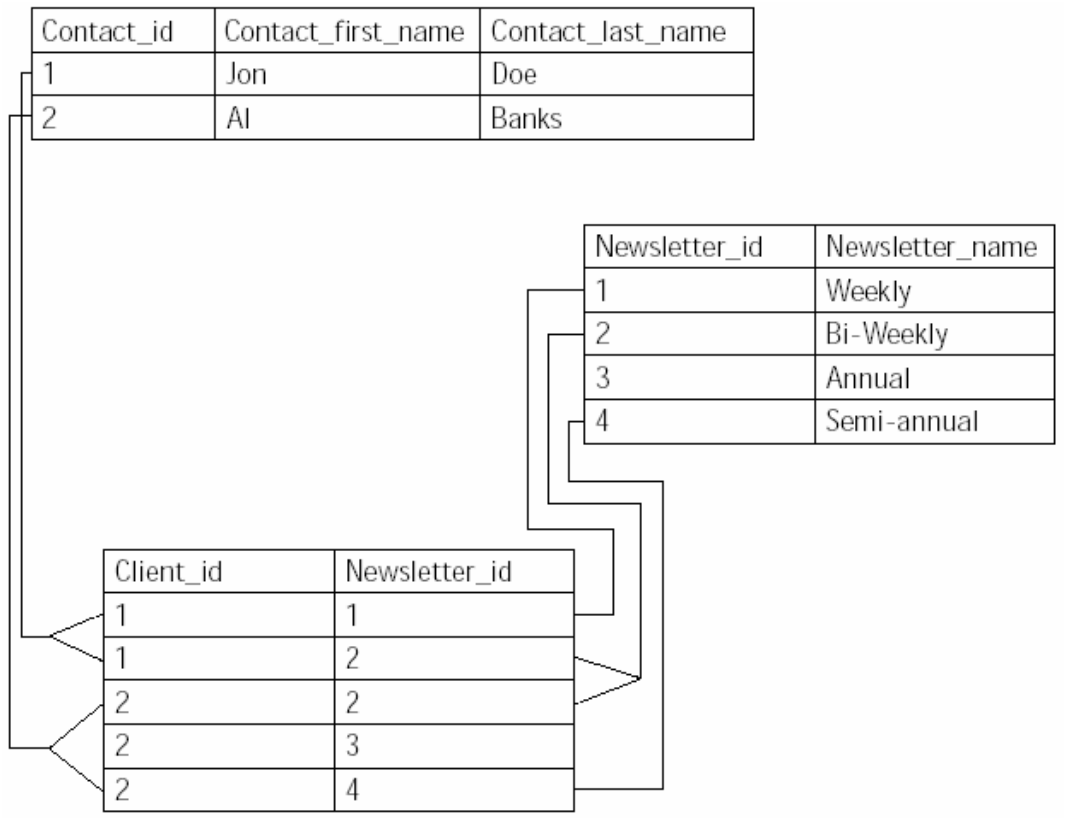
contact_id	contact_first_name	contact_last_name	newsletters
1	Jon	Doe	1,3,4
2	Al	Banks	2,3,4

Coloana *newsletters* conține mai mult de o valoare, fiind de fapt un şir de valori. Aşa cum s-a menționat la *prima formă normală* această situație nu trebuie să apară niciodată în baza de date. Soluția este crearea încă unei tabele pentru a memora aceste informații, aşa cum se exemplifică în tabelul 10.

Tabelul 10. Clienți ai publicațiilor

Client_id	Newsletter_id
1	1
1	2
2	2
2	3
2	4

Iată acum că această tabelă constituie baza unei relații m la n în baza de date:



8. Aspecte ale stocării datelor în BD relaționale

Integritatea referențială. Exemplele discutate până acum au folosit *chei străine*. O cheie străină este o coloană care referă o cheie primară dintr-o altă tabelă a bazei de date cu ajutorul căreia se realizează relația între cele două tabele. De exemplu, tabelele în 3 şi 4:

company_id	company_name	company_address
1	BigCo Company	1121 43rd St
2	LittleCo Company	4444 44th St

contact_id	company_id	contact_name	contact_title	phone	email
1	1	Jay Greenspan	Vice President	4155551212	jgreen_1@yahoo.com
2	1	Brad Bulger	President	4155552222	bbulger@yahoo.com
3	2	John Doe	Lackey	2125556666	nobody@hotmail.com

coloana *company_id* din tabela de contacte este o *cheie străină* către tabela de companii.

În anumite SGBD-uri, ca VFP, Oracle, Sybase sau PostGres, tabelele pot fi create cu definirea explicită a cheilor străine (*field validation/rule* în VFP) aşa încât adăugarea va fi acceptată de către sistem doar dacă valoarea *cheii străine* există în tabela referită ca *cheie primară*.

În alte SGBD-uri, ca MySQL, cheile străine nu au această semnificație. În acest caz, programatorul trebuie să efectueze câțiva paşi suplimentari înainte de a adăuga sau modifica înregistrări, cum ar fi (pentru tabelele 3 şi 4):

- preia toate valorile pentru *company_id* din tabela de companii;
- verifică dacă valoarea pentru *company_id* pe care intenționezi să o inserezi în tabela de contacte există în şirul de date obținut mai sus;
- dacă ea există, inserează valorile;

Tranzacții. În BD relaționale au loc schimbări de apartenențe în grupuri. Multe schimbări necesită ca liniile să fie actualizate în mai multe tabele deodată. În unele cazuri acest lucru se face printr-o succesiune de instrucțiuni care preiau datele acolo unde trebuie să fie stocate.

Un site pentru comerțul electronic poate conține un cod care să efectueze operațiile:

1. adaugă clientul în tabela de clienți;
2. adaugă lista de cumpărături în tabela de cumpărături;
3. scade cantitățile comandate din tabela de stocuri.

Când se lucrează cu o serie de paşi ca aceasta, există un potențial pentru probleme care pot apare. Dacă sistemul se blochează sau iese în decor între paşii 2 şi 3, atunci baza de date conține date eronate.

Pentru a preveni o astfel de situație, unele SGBD-uri folosesc conceptul de *tranzacții*. Cu acesta, programatorul poate identifica un grup de comenzi. Dacă una dintre aceste comenzi eşuează în înregistrare, întregul grup este respins şi baza de date este restaurată la starea sa înainte de efectuarea grupului de comenzi (tehnologia COMMIT/ROLLBACK).

Proceduri de memorare. SGBD-urile cu mai mare flexibilitate permit inserarea de cod procedural în baza de date (ceva foarte asemănător cu PHP şi Perl). Sunt câteva avantaje care acest fapt le conferă:

- se poate reduce cantitatea de cod necesar pentru aplicațiile de mărime medie;
- se asigură baza de cunoştințe (proceduri) pentru execuția interogărilor şi tranzacțiilor de pe sisteme de operare diferite şi din SGBD-uri diferite în ceea ce priveşte sintaxa.

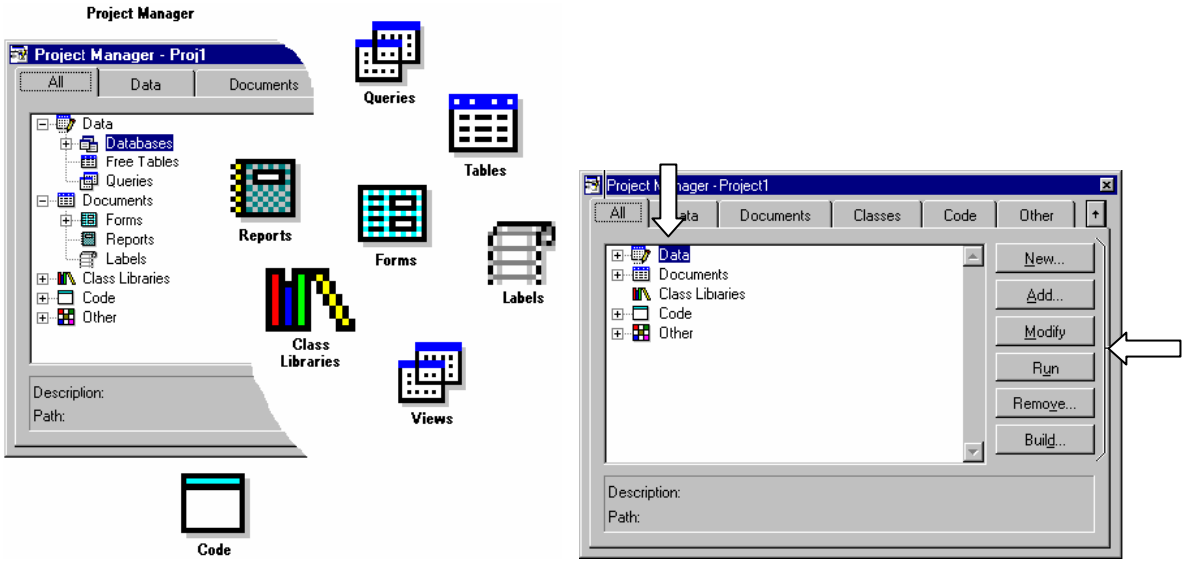
9. Lucrul cu Project Manager în VFP

Project Manager este instrumentul de organizare primară al lucrului cu date şi obiecte în VFP. Un *proiect* este o colecție de fişiere, date, documente şi obiecte VFP care sunt salvate ca un fişier cu extensia *.PJX*.

Se poate utiliza *Project Manager* pentru a organiza şi manipula fişiere, crea tabele şi baze de date, scrie interogări, defini forme şi rapoarte şi construi aplicații. Pentru construcția de aplicații, un bun îndrumar este *Programmer's Guide* din *MSDN*.

Pentru a crea un nou proiect, se urmează succesiunea de paşi:

1. Se acționează butonul *New* din bara de instrumente;
2. Se selectează *Project* din caseta de butoane radio;
3. Se apasă butonul *New File* când se va activa o fereastră de dialog;
4. În caseta de editare cu eticheta *Enter project* se introduce numele dorit pentru proiect;
5. Se apasă butonul *Save* când se va genera un nou proiect cu numele ales.



Se pot acum adăuga proiectului tabele (fişiere .DBF) sau baze de date (fişiere .DBC).

În *Project Manager* datele sunt grupate pe categorii şi prezentate într-o formă ierarhică. Pentru a urmări un tip de fişier sau obiect se activează boxa ⊞ a grupului corespunzător. Activând tabulatorul *Data* putem restrânge domeniul de vizualizare al componentelor proiectului la nivelul de date, şi anume: bazele de date, tabelele, interogările şi vizualizările. Acționând unul din butoanele *New…*, *Add…*, *Modify*, *Run*, *Remove…* sau *Build…* se alege operația specifică asupra componentei sau categoriei selectate.

Acțiunile care se pot efectua asupra datelor sunt descrise în continuare.

10. Crearea tabelelor şi indexurilor

Aşa cum s-a arătat, pentru crearea unei tabele se poate folosi ***Table Wizard*** sau se poate lucra direct cu ***Table Designer***.

Tipurile de date permise sunt descrise în tabelul următor.

Tabelul 11. Tipuri de date în VFP

Tip dată	Descriere	Exemplu
Character	Text alfanumeric	Adresa unui client
Currency	Unități monetare	Preț de cumpărare
Numeric	Numere întregi sau cu zecimele	Numărul de produse comandate
Float	La fel cu Numeric	
Date	Lună, zi şi an	Data la care a fost făcută comanda
DateTime	Lună, zi, an, oră, minut şi secundă	Data, ora la care un angajat vine la serviciu
Double	Număr în dublă precizie	Date provenite din experimente ce necesită înalt grad de precizie
Integer	Valori numerice fără zecimale	Numărul de înregistrare al unei comenzi
Logical	*True* sau *False*	Dacă o comandă a fost sau nu făcută
Memo	Text alfanumeric de lungime necunoscută	Lista apelurilor telefonice efectuate
General	OLE	Foaie de calcul din Excel
Character (Binary)	La fel cu Character dar valorile nu sunt translatate atunci când se schimbă codul de pagină	Parolele unor utilizatori stocate într-o tabelă şi folosite în diferite țări
Memo (Binary)	La fel cu Memo dar valorile nu sunt translatate cu schimbarea codului de pagină	Scriptul de logare al unui utilizator (în diferite țări)

Dacă se doreşte ca câmpul să accepte introducerea de valori nule, se check-ează butonul de pe coloana *NULL* din *Table Designer*.

Pentru a adăuga înregistrări în tabelă din *Project Manager* se parcurg următorii paşi:

1. În *Project Manager* se selectează numele tabelei;
2. Se apasă butonul *Browse*;
3. Din meniu, se selectează *View/Append Mode*;
4. Se introduc valorile noii înregistrări în fereastra Browse;

Dacă se doreşte vizualizarea fiecărui câmp pe linie separată se selectează din meniu *View/Edit*; revenirea la starea anterioară se face tot din meniu *View/Browse*.

În formatul *View/Browse* fiecare linie reprezintă o *înregistrare* iar fiecare coloană reprezintă *câmpurile* înregistrării.

Se pot crea (după cum se poate vedea din *Project Manager*) două tipuri de tabele: tabelele încorporate într-o bază de date (*database table*) şi tabelele libere (*free table*), care sunt independente de orice bază de date.

Deplasarea într-o tabelă. Utilizând barele de defilare orizontale şi verticale ne putem deplasa şi vizualiza câmpuri diferite şi înregistrări diferite. Se pot folosi de asemenea săgeţile şi TAB-ul. Pentru a accesa o anume înregistrare, pentru a ne deplasa de la o înregistrare la alta, pentru a ajunge la începutul sau sfârşitul tabelei se poate accesa din meniu *Table/Got o Record* > când se activează un submeniu din care se alege opțiunea dorită.

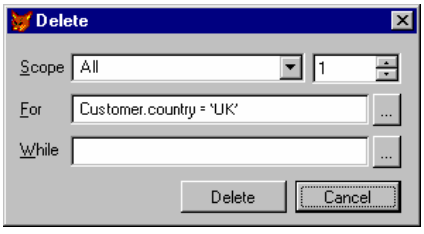
Pentru a edita câmpurile de tip Character, Numeric, Logical, Date sau DateTime este suficient să plasăm cursorul în celula dorită şi să scriem valoarea dorită. Editarea câmpurilor Memo se face cu dublu click în câmpul dorit (sau CTRL+PgDn). Atunci se activează o fereastră de editare pentru conținutul câmpului memo. Un câmp general conține un obiect OLE (detalii la „http://www.microsoft.com/data/oledb/prodinfo.htm") legat sau încapsulat. Se poate edita acest obiect cu dublu click în celula sa, când se va activa (sau se va încerca activarea) aplicației sale asociate pentru editare.

Ştergerea înregistrărilor. Ştergerea înregistrărilor dintr-o tabelă este un proces în 2 paşi în VFP. Primul, marcarea înregistrărilor pentru ştergere de exemplu cu click pe boxa din stânga fiecărei înregistrări de şters. Al doilea, din meniu, *Table/Remove Deleted Records* care va şterge înregistrările marcate pentru ştergere. Aceasta este o ştergere permanentă şi nu poate fi restaurată, spre deosebire de prima care este doar o ştergere logică şi poate fi restaurată prin demarcarea înregistrărilor prin acelaşi procedeu. Eliminarea înregistrărilor şterse va închide tabela aşa încât aceasta trebuie redeschisă pentru a putea fi utilizată.

Pentru a şterge mai multe înregistrări care verifică o condiție se poate folosi fereastra de dialog care se activează din meniu *Table/Delete Records.…*

Analog, pentru a restaura înregistrări care verifică o condiție se poate accesa din meniu *Table/Recall Records...*

De exemplu, ca în figură, se pot selecta toate înregistrările care au valoarea UK în câmpul Country, folosind expresia **FOR Country = 'UK'**:



Fereastra de Browse se poate configura după dorință, prin (re)dimensionarea lățimii coloanelor, activarea sau dezactivarea liniilor de grid sau divizarea ferestrei de Browse în două porțiuni. Aceste operațiuni nu vor afecta structura actuală a tabelei.

Pentru a modifica structura unei tabele, se alege în *Project Manager* opțiunea *Modify*. Structura tabelei este încărcată atunci în *Table Designer* şi utilizatorul are posibilitatea să

efectueze modificările dorite. Pentru a şterge un câmp din tabelă se selectează câmpul şi se apasă butonul *Delete*.

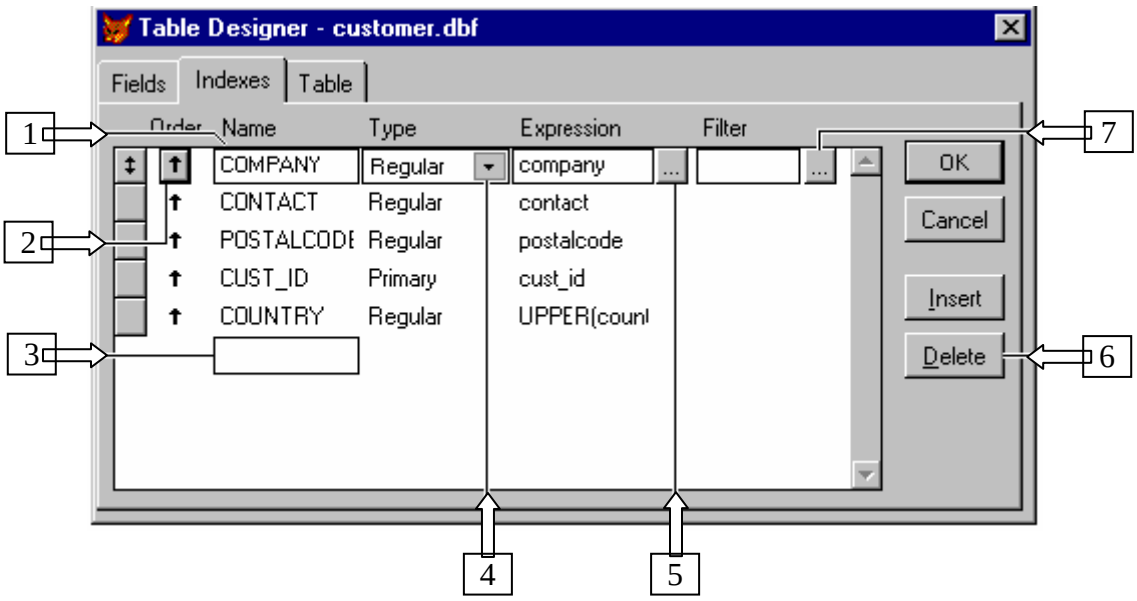
Afişarea înregistrărilor în fereastra de Browse se poate face selectiv după o condiție impusă înregistrărilor. De asemenea se poate limita accesul la anumite câmpuri ale tabelei prin setarea unui filtru de câmpuri. Pentru impunerea unui filtru, din meniu, *Table/Properties*:

Odată ce a fost creată o tabelă, aceasta se poate ordona pentru a accelera regăsirea informației prin folosirea ***indexurilor***. Cu indexurile, se pot procesa rapid înregistrările pentru afişare, interogare sau tipărire. Se pot selecta înregistrări, urmări dacă există valori duplicat într-un câmp şi suportă stabilirea de relații între tabele în cadrul bazelor de date. Un index în VFP este o listă de numere de înregistrări care pointează către înregistrările corespunzătoare şi astfel determină ordinea de procesare a înregistrărilor. Un index nu modifică ordinea în care înregistrările sunt stocate în tabelă, ci modifică doar ordinea în care acestea sunt citite de VFP.

Se poate crea mai mult de un index pentru o tabelă (fiecare index reprezentând câte o ordine de procesare a înregistrărilor din tabelă). Indexurile care se crează sunt stocate într-un fişier index cu structură compusă care este deschis şi actualizat oricând tabela este folosită.

Numele fişierului index este identic cu numele tabelei iar extensia sa este .*CDX*. Un index se poate crea uşor, aşa încât frecvent se defineşte câte un index după fiecare câmp al tabelei. Un index se poate crea după un *câmp* sau după o *expresie*.

Din *Project Manager* se selectează tabela, se apasă butonul *Modify*, apoi în *Table Designer* se selectează tabulatorul *Indexes* când se activează o fereastră de forma:



în care avem posibilitatea:

- 1 să modificăm numele pentru un index;
- 4 să alegem tipul indexului după cum urmează:
  - *index primar* (cheie primară) unde doar valori unice sunt permise în câmp şi determină ordinea de procesare a înregistrărilor; se pot crea câte un index primar pentru fiecare tabelă dintr-o bază de date; dacă tabela are deja un index primar, se poate crea atunci un *index candidat*;
  - *index candidat* care de asemenea necesită valori unice şi determină ordinea de procesare a înregistrărilor; pot exista mai multe indexuri candidate pentru 1 tabelă;
  - *index regular* care permite duplicarea valorilor ce intră într-un câmp, este folosit pentru a determina ordinea de procesare şi poate exista mai mult de 1 index regular pentru o tabelă;
  - *index unic* păstrat pentru compatibilitate cu versiunile mai vechi care selectează şi ordonează un subset de înregistrări bazat pe prima apariție a unei valori într-un câmp specificat;
- 5 să definim o expresie simplă (formată din numele unui câmp) sau compusă (o expresie în care intervin nume de câmpuri din tabelă) după cum urmează:
  - câmpurile sunt evaluate în ordinea în care apar în expresie;
  - operatorul „+” aplicat la câmpuri numerice va aduna valorile din câmpuri:

    employ.salary + employ.prime
  - operatorul „+” aplicat la câmpuri caracter va concatena şirurile de caractere din câmpuri:

    customer.country + customer.postalcode + customer.company
  - sunt permise conversiile la tipul şir de caractere prin intermediul funcției STR(·):

STR(customer.maxordamt,20,4) + customer.company

- 7 să filtrăm înregistrările supuse indexării după o condiție logică:

employ.prime > 0 sau *customer.country = "Canada"*

- 2 să alegem o ordonare descendentă (↓) sau ascendentă (↑);
- 3 să adăugăm un nou index;
- 6 să ştergem un index;

Indexurile pot îndeplini diferite roluri, în funcție de tipul acestora:

Dacă vrei să	Foloseşte
Ordonezi înregistrările pentru a mări viteza de afișare, interogare sau tipărire	Un index regular, primar sau candidat
Controlul intrării valorilor duplicat într-un câmp şi ordonarea înregistrărilor	Un index primar sau candidat pentru tabelele libere

Odată creat, un index poate fi folosit la deschiderea unei tabele, în forma:

USE customer ORDER Cust_Id

când la activarea ferestrei de Browse se vor afişa înregistrările în ordinea specificată.

11. Colectarea tabelelor într-o bază de date

Punând tabelele într-o bază de date, se poate reduce stocarea datelor redundante şi se poate proteja *integritatea* datelor.

Un SGBD cum este VFP furnizează mediul de lucru pentru:

- lucrul cu tabele;
- stabilirea relațiilor între tabele;
- setarea proprietăților şi regulilor de validare pentru date.

Următorul exemplu arată cum se realizează o bază de date.

Problema:

Să presupunem că se dorește realizarea unei baze de date care să conțină informații despre universități şi persoane de contact în cadrul acestora.

Aşa cum s-a discutat până acum aceasta presupune crearea a cel puțin două tabele în cadrul bazei de date între care să se stabilească relații.

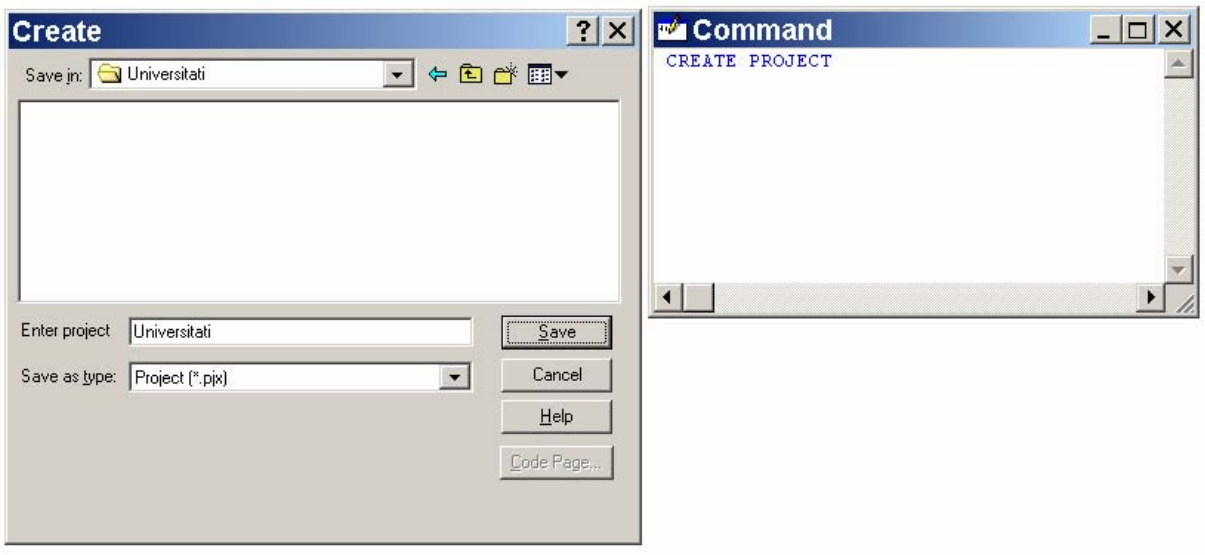
Obiective:

- crearea unui proiect (*Project Manager*);
- crearea unei baze de date (*Database Designer*);

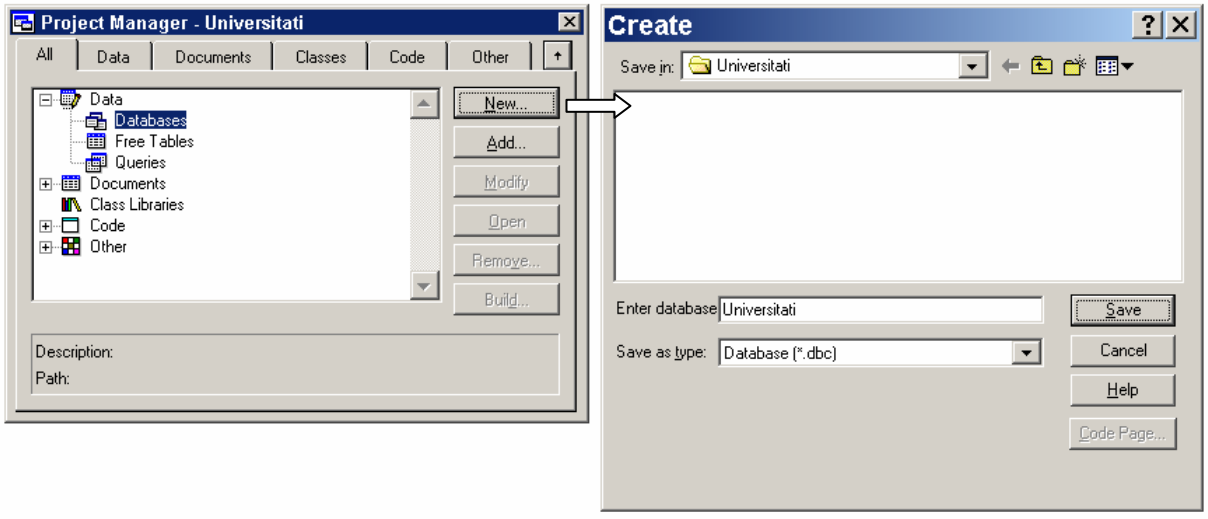
Implementare:

- crearea proiectului

1. Din meniu sau din bara de instrumente se selectează *File/New (File Type: Project)/New file;*
2. Se explorează calculatorul şi se alege locul unde se dorește stocarea proiectului pe disc;
3. Se creează un director pentru stocarea componentelor proiectului;
4. Se dă nume proiectului; fie de exemplu numele Universitati; acesta se va salva pe disc sub numele *Universitati.pjx*;



- crearea bazei de date

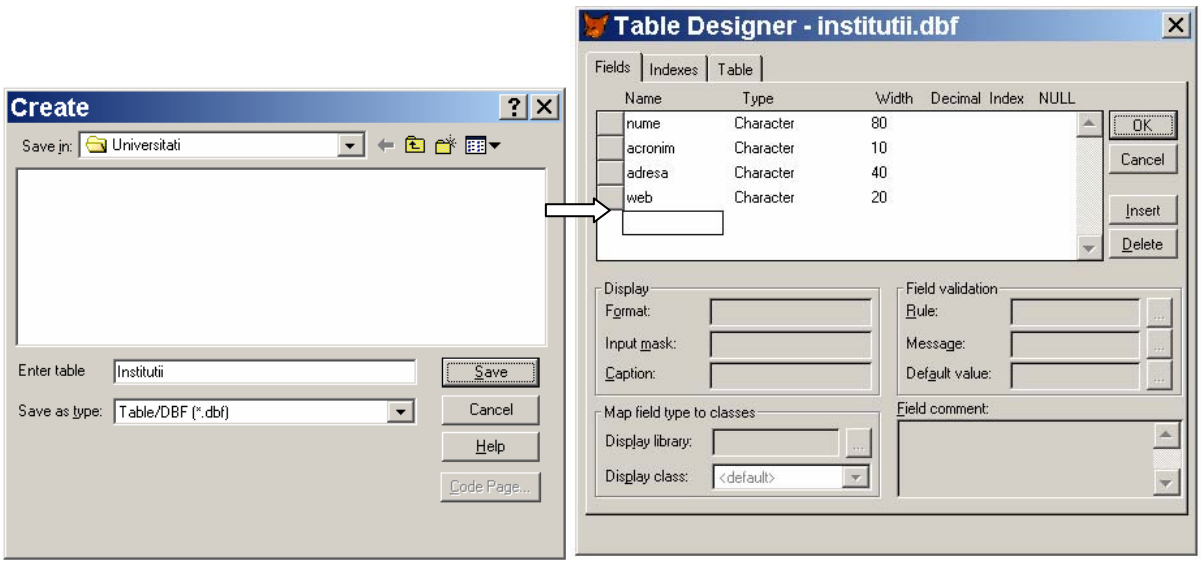


5. Din *Project Manager* din categoria *Data* se selectează *Databases* şi se apasă butonul *New* şi apoi *New Database*;
6. Se introduce un nume pentru noua bază de date din cadrul proiectului; fie aceasta Universitati; aceasta se va salva pe disc sub numele *Universitati.dbc*; se va încărca în mod automat aplicația expert *Database Designer*; cu ajutorul ei se construiesc tabelele bazei de date şi se definesc relațiile între tabele; *Database Designer* mai permite crearea de vederi locale şi de la distanță, editarea procedurilor stocate în baza de date (baza de cunoştințe pentru execuția interogărilor şi tranzacțiilor de pe sisteme de operare diferite şi din SGBD-uri diferite), realizarea de conectări la distanță;

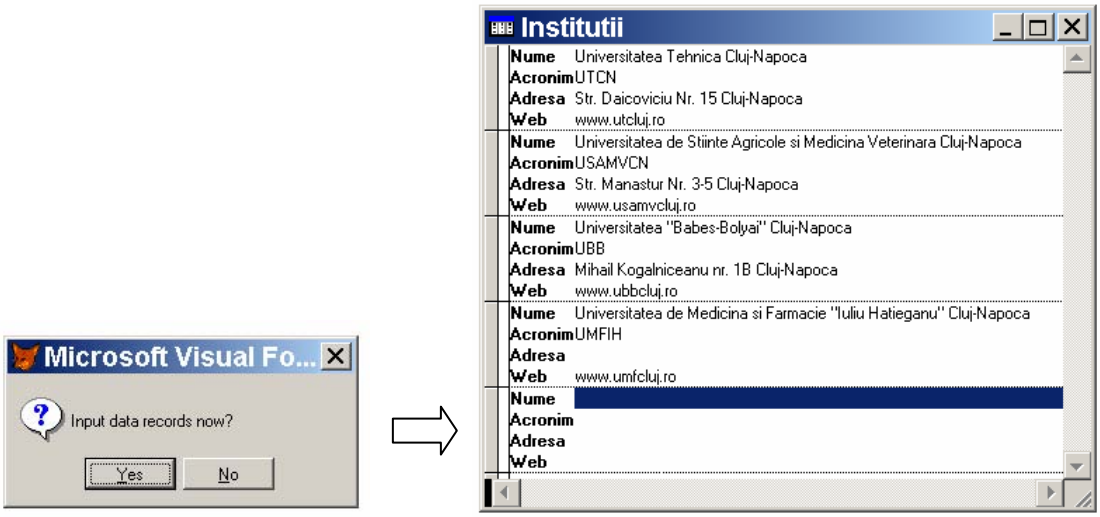
- crearea tabelelor

Se enumeră informațiile care se doresc memorate în baza de date. Acestea ar pute fi: nume universitate, acronim, adresă, nume rector, prorectori, adrese email; se identifică faptul că pentru o universitate avem de memorat informații proprii instituției (nume, acronim, adresa, pagina web) şi informații despre persoanele aflate la conducerea acesteia (nume, funcție, adresa email). Se desprind astfel în mod natural două tabele în baza de date: tabela institutii şi tabela contacte.

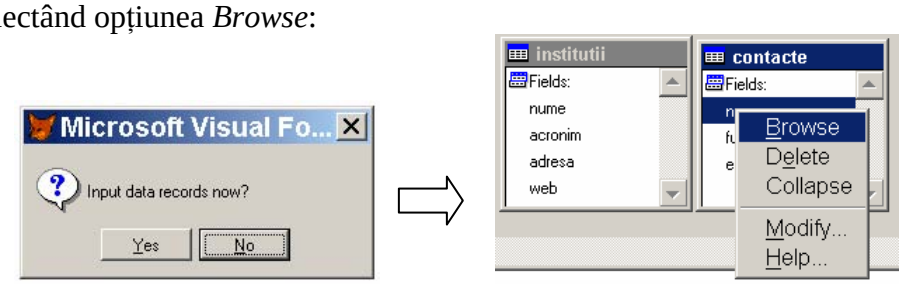
7. Din aplicația expert *Database Designer* se selectează butonul *New Table* şi apoi din nou *New Table*; se va lansa în execuție în mod automat aplicația expert *Table Designer*;
8. Cu ajutorul aplicației *Table Designer* se creează tabela *institutii* cu structura *nume char(80), acronim char(10), adresa char(40), web char (20)*; aceasta se va salva pe disc cu numele *Institutii.dbf*;



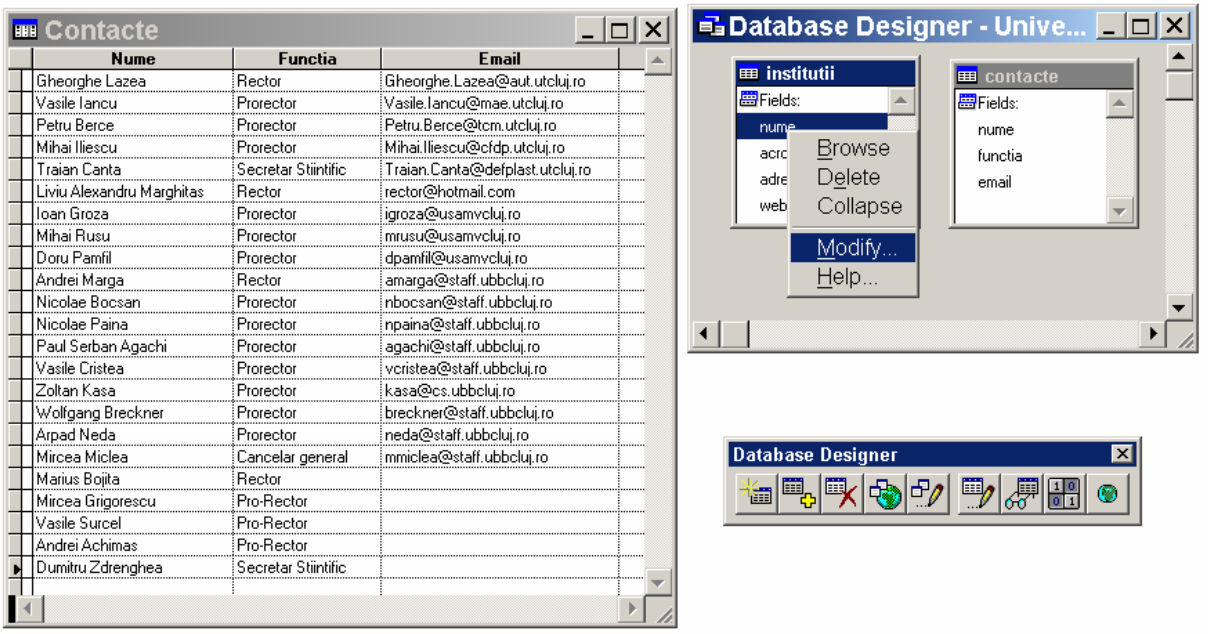
9. Se apasă butonul *Ok* când se activează o fereastră de dialog; aici se poate alege dacă să se introducă acum informațiile despre instituții sau mai târziu; să alegem introducerea acum;
10. Se activează o fereastră care permite introducerea informațiilor; se introduc informațiile despre universități;



11. Din *Database Designer* se încarcă din nou *Table Designer* pentru crearea celei de-a doua tabele;
12. Se creează tabela *Contacte* cu structura: *nume char(25), functia char(25), email char(32)*;
13. Se alege să se introducă informațiile despre contacte mai târziu;
14. Se pot acum introduce informații în tabela *Contacte* activând click dreapta pe tabelă și selectând opțiunea *Browse*:

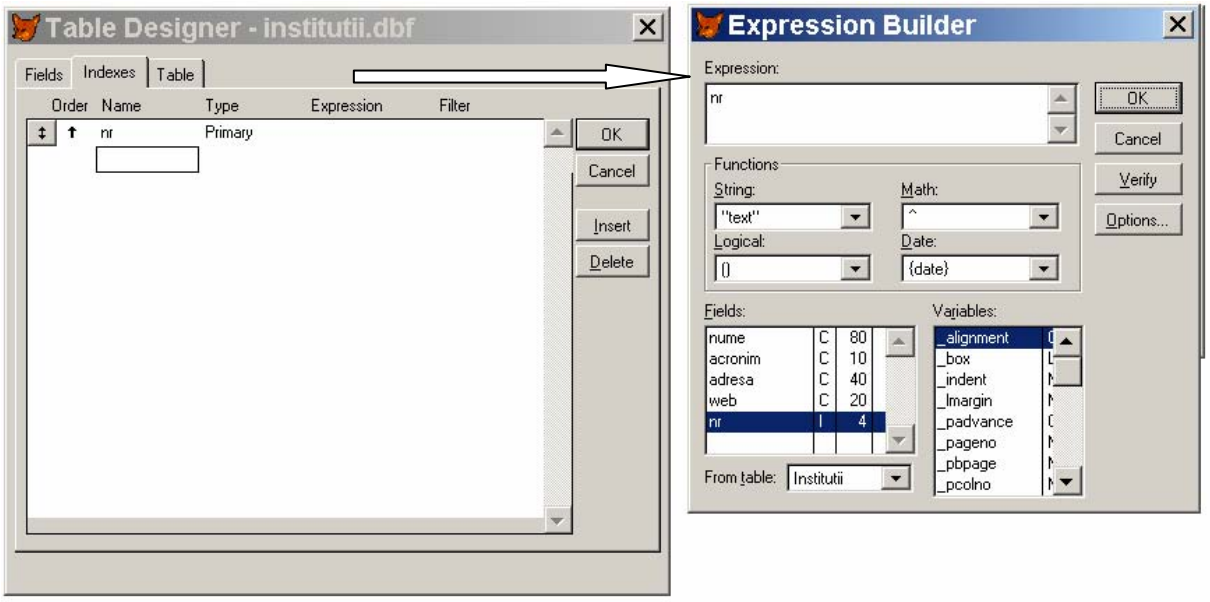


15. Se activează o fereastră *Browse*; se adaugă câte o înregistrare (din meniu, *Table/Append New Record* sau de la tastatură *Ctrl+Y*);
16. Se doreşte stabilirea de relații între instituții şi contacte; pentru aceasta este necesară adăugarea unor câmpuri numerice în tabele; deoarece la o instituție avem mai multe persoane de contact relația între tabele este de tipul 1 la n; pentru stabilirea relației este necesar ca valorile câmpului numeric din tabela *institutii* (fie acesta *nr int*) să fie distincte; se modifică tabela şi se creează acest câmp în consecință;



17. Se adaugă un câmp numeric (de preferință cu acelaşi nume) în tabela *contacte* ale căror valori se vor completa ținând seama de apartenența persoanelor de contact la instituții;

- crearea indexurilor

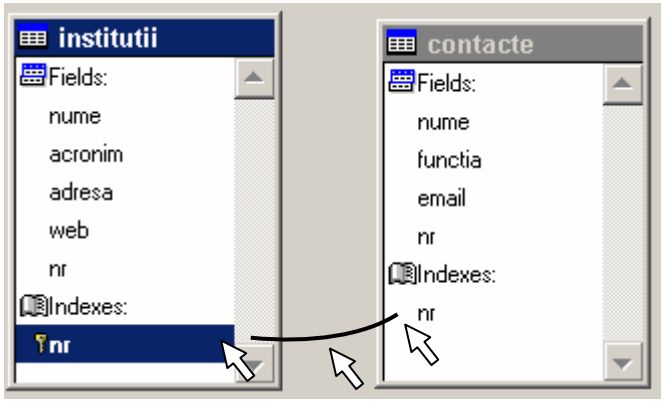


18. Câmpul *nr* din tabela *institutii* se numeşte cheie primară; pentru stabilirea unei relații 1 la n după acest câmp tabela va trebui să fie indexată după acest câmp cu un *index primar*;

din nou aplicăm *Modify* la tabela *Institutii* şi îi asociem un index primar cu acelaşi nume cu câmpul după care se face indexarea: *nr*; se foloseşte aici aplicația expert *Expression Builder*; se va memora în mod automat pe disc indexul sub numele *Institutii.cdx*;

19. Câmpul *nr* din tabela *contacte* se numeşte cheie străină; valorile din această coloană a tabelei contacte nu sunt toate diferite între ele; după acest câmp tabela se poate indexa cu un index regular; se creează indexul; fie numele acestuia *nr*;

- stabilirea relației între tabele

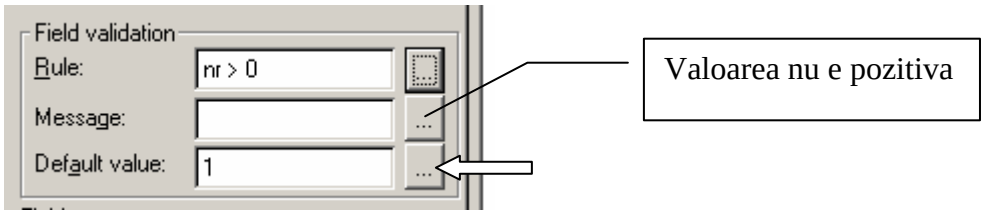


20. Se selectează indexul primar şi se efectuează *drag and drop* cu mouse-ul peste indexul regular (cheia primară peste cheia străină);

12. Validarea datelor la adăugare sau modificare

Pentru a controla tipul informației introduse de utilizator într-un *câmp* dintr-o tabelă se poate valida data independent de orice altă intrare în înregistrare, se poate realiza o ***validare la nivel de câmp*** a datelor. De exemplu, pentru a ne asigura că utilizatorul nu introduce o valoare negativă într-un câmp care trebuie să conțină doar valori pozitive.

Fie cazul când dorim să validăm printr-o regulă de acest tip câmpul *nr* din tabela *Contacte* a bazei de date *Universitati*. O soluție ar arăta astfel:

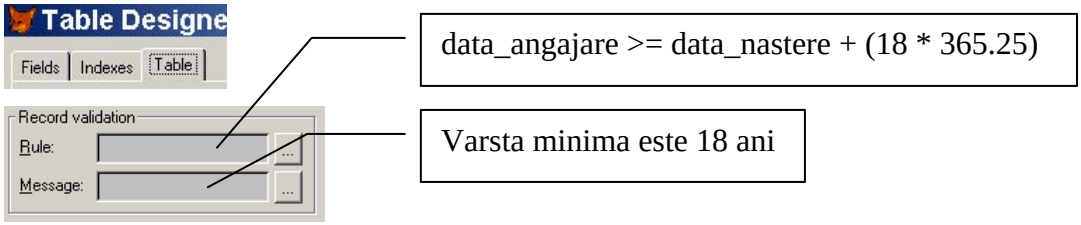


Butoanele „...” lansează aplicația expert *Expression Builder* cu ajutorul căreia se poate construi expresia pentru regula de validare a datelor, valoarea implicită a câmpului la adăugarea unei noi înregistrări şi mesajul care se va afişa în caz de introducere eronată.

O altă posibilitate este de a stabili reguli de ***validare la nivel de înregistrare*** într-o tabelă. Presupunând că într-o tabelă *angajati* avem două câmpuri *data_nastere* şi *data_angajare* de tip *date* atunci se poate stabili următoarea regulă de validare la nivel de înregistrare:

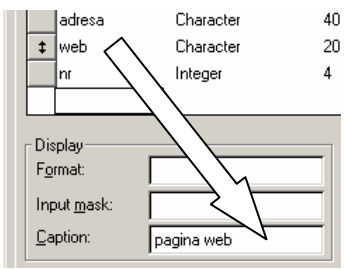
*data_angajare >= data_nastere + (18 * 365.25)*

care să verifice că la angajare viitorul angajat are peste 18 ani:



Incluzând comenzi sau funcții în regulile de validare care încearcă să mute pointerul de înregistrare în zona de lucru se pot provoca erori în stabilirea condițiilor.

Se poate defini un nume de câmp care va fi afişat (diferit de numele acestuia din tabelă) prin definirea acestuia în caseta de editare *Caption* din tabulatorul *Fields* al lui *Table Designer*:

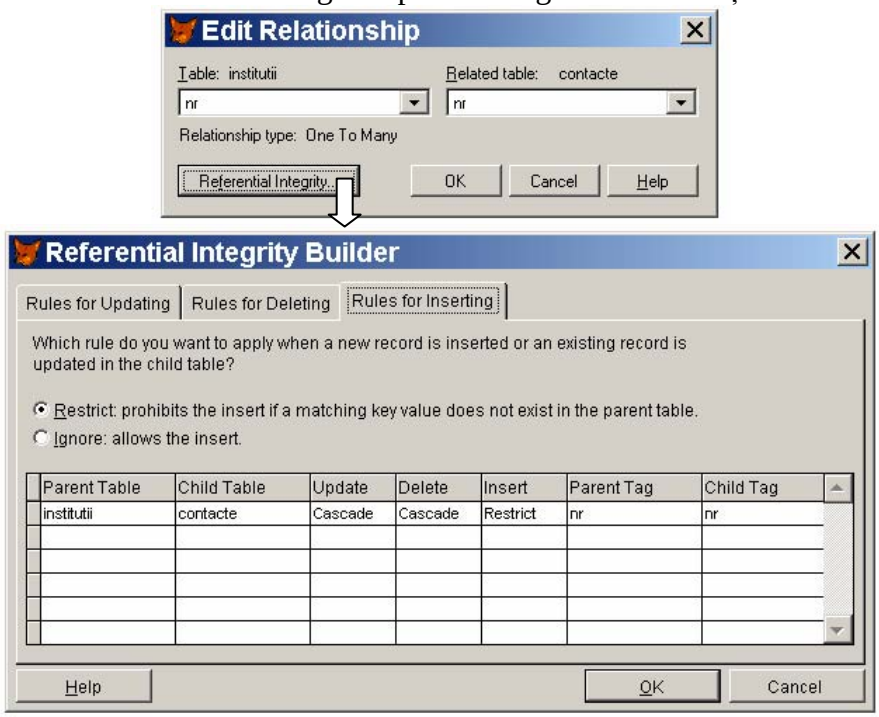


13. Manipularea înregistrărilor în baza de date şi integritatea referențială

După ce au fost stabilite relațiile, se pot de asemenea defini reguli pentru manipularea înregistrărilor în baza de date. Acestea asigură *integritatea referențială*.

De exemplu, dacă se adaugă o companie în tabela *companii*, se doreşte adăugarea automată a informațiilor despre persoana de contact în tabela *persoane_contact*. Pentru aceasta se foloseşte *Referential Integrity Builder*. Se urmează paşii:

- cu baza de date deschisă din meniu se selectează *Database/Clean Up Database*;
- cu dublu click pe linia ce marchează relația se activează fereastra *Edit Relationship*;
- se apasă butonul *Referential Integrity...*;
- în fereastra creată se selectează regulile pentru integritatea referențială:

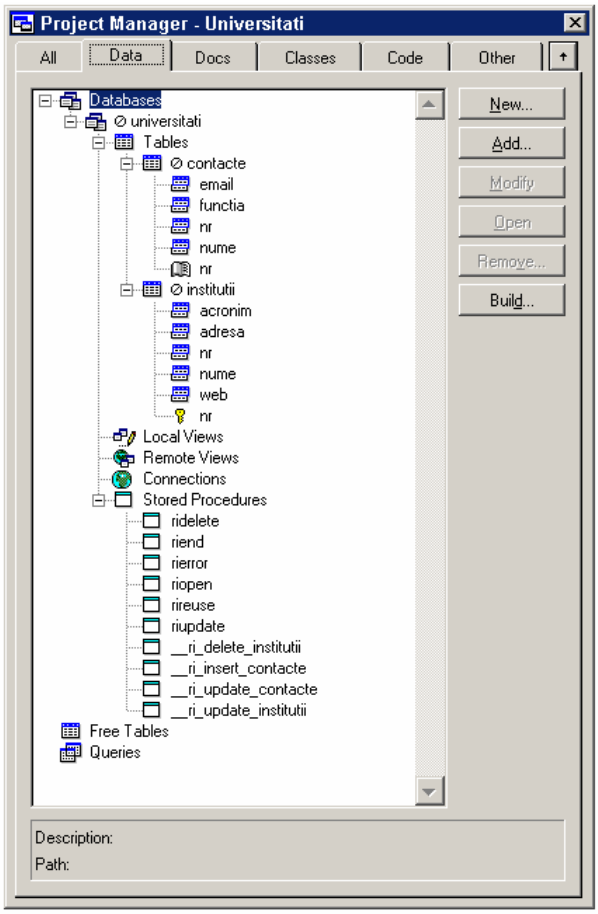


Referential Integrity Builder permite definirea următoarelor reguli:

Updating	Cascade	Atunci când o valoare a cheii primare este modificată actualizează toate înregistrările din tabela cu cheia străină corespunzătoare
	Restrict	Nu lasă modificarea valorii cheii primare atunci când aceasta are corespondente valori ale cheii străine în tabela cu cheia străină
	Ignore	Permite modificările în tabela cu cheia primară şi lasă nereferite înregistrările din tabela cu cheia străină
Deleting	Cascade	Atunci când este ştearsă o înregistrare din tabela cu cheia primară şterge toate înregistrările din tabela cu cheia străină a căror valoare a cheii străine corespunde cu valoarea respectivă a cheii primare
	Restrict	Nu permite ştergerea unei înregistrări din tabela cu cheia primară atunci când valoarea cheii acesteia este regăsită printre valorile cheii străine din tabela cu cheia străină;
	Ignore	Permite ştergerile în tabela cu cheia primară şi lasă nereferite înregistrările din tabela cu cheia străină
Inserting	Restrict	Atunci când o nouă înregistrare este adăugată sau una existentă este

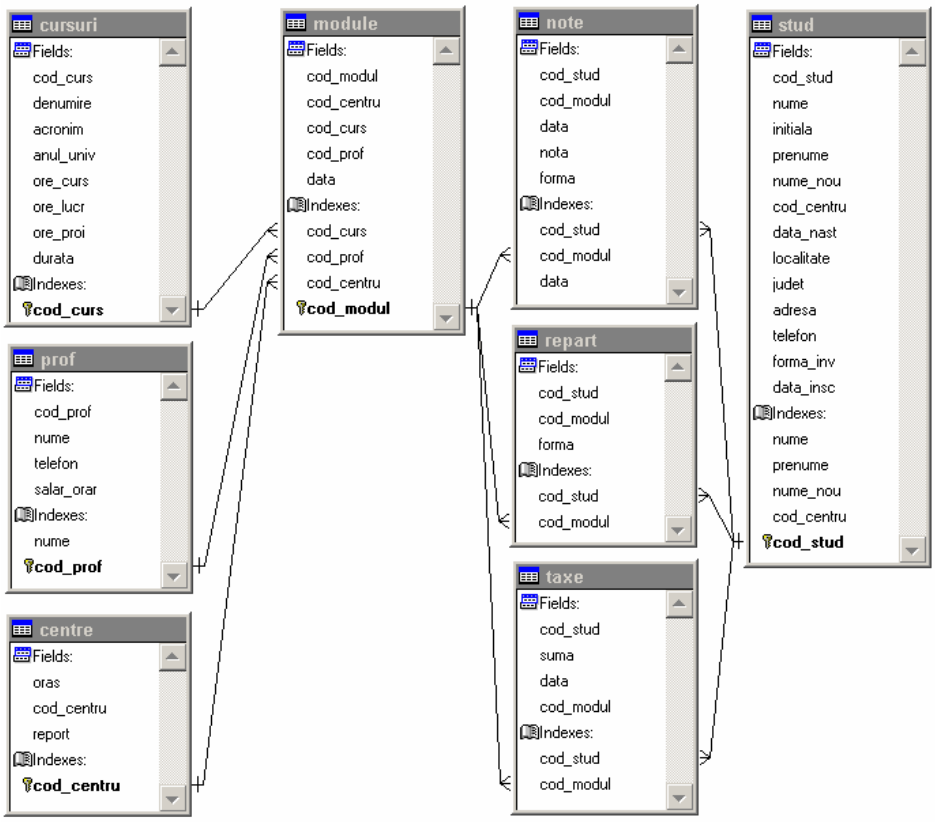
		modificată în tabela cu cheia străină se verifică dacă există valoarea introdusă pentru cheia străină printre valorile cheii primare din tabela cu cheia primară şi nu permite inserarea sau modificarea dacă această valoare nu este găsită
	Ignore	Permite adăugarea sau modificarea fără respectarea integrității referențiale

Referential Integrity Builder va genera cod de integritate referențială care va fi inclus în baza de date sub formă de proceduri, aşa cum se vede din *Project Manager*.



Aplicație:

Să se creeze baza de date cu cursanții unei Şcoli de Informatică Aplicată şi Programare.



Soluție:

Se creează o bază de date ce conține următoarele tabele (în care ↑ exprimă existența unui index după câmpul specificat; tipul indexului se va stabili în funcție de tipul relației tabelului respectiv cu celelalte în cadrul bazei de date).

Fișiere de date

9. Cursuri.dbf			
COD_CURS	N	5	↑
DENUMIRE	C	70	
ACRONIM	C	10	
ANUL_UNIV	C	12	
ORE_CURS	N	2	
ORE_LUCR	N	2	
ORE_PROI	N	2	
DURATA	N	2	

2. Stud.dbf			
COD_STUD	N	5	↑
NUME	C	20	↑
INITIALA	C	1	
PRENUME	C	30	↑
NUME_NOU	C	20	↑
COD_CENTRU	N	5	↑
DATA_NAST	D	8	
LOCALITATE	C	20	
JUDET	C	20	
ADRESA	M	10	
TELEFON	N	10	
FORMA_INV	C	1	

S-scoala M-module

3. Repart.dbf			
COD_STUD	N	5	↑
COD_MODUL	N	5	↑
FORMA	C	1	

4. Taxe.dbf			
COD_STUD	N	5	↑
SUMA	N	10	
DATA	D	8	

(**N**ormal sau **D**istanță)				COD_MODUL	N	5	↑
7. Module.dbf				***8. Prof.dbf***			
COD_MODUL	N	5	↑	COD_PROF	N	5	↑
COD_CENTRU	N	5	↑	NUME	C	20	↑
COD_CURS	N	5	↑	TELEFON	N	10	
COD_PROF	N	5	↑	SALAR_ORAR	N	10	
DATA	D	8					

12. Note.dbf				***10. Centre.dbf***			
COD_STUD	N	5	↑	ORAS	C	12	↑
COD_MODUL	N	5	↑	COD_CENTRU	N	5	↑
DATA	D			REPORT	N	12	
NOTA	N	2					

Rezultatul analizei problemei se prezintă în următorul tabel:

Nr	Nume	Index	Tip
12	note.dbf	Cod_stud	Regular
		Cod_modul	Regular
10	centre.dbf	Cod_centru	Primar
		Oras	Regular
9	cursuri.dbf	Cod_curs	Primar
8	prof.dbf	Cod_prof	Primar
		Nume	Regular
7	module.dbf	Cod_modul	Primar
		Cod_centru	Regular
		Cod_curs	Regular
		Cod_prof	Regular
		Natura	Regular
		Cod_prof	Regular
		Localitate	Regular
4	taxe.dbf	Cod_stud	Regular
		Cod_modul	Regular
3	repart.dbf	Cod_stud	Regular
		Cod_modul	Regular
2	stud.dbf	Cod_stud	Primar
		Nume	Regular
		Prenume	Regular
		Cod_centru	Regular

Analiza se face în modul următor (de exemplu pentru tabela 12): Tabela 12 (note.dbf) conține 2 indexuri: unul după cod student şi unul după cod modul; valorile acestor câmpuri se pot repeta în tabelă: un student ia mai multe note (la diferite module) şi un modul este frecventat de mai mulți studenți; indexurile sunt atunci regulare.

După ce s-au stabilit relațiile se impun:

- regulile de validare pentru câmpuri;
- regulile de validare pentru înregistrări;
- regulile de integritate referențială.

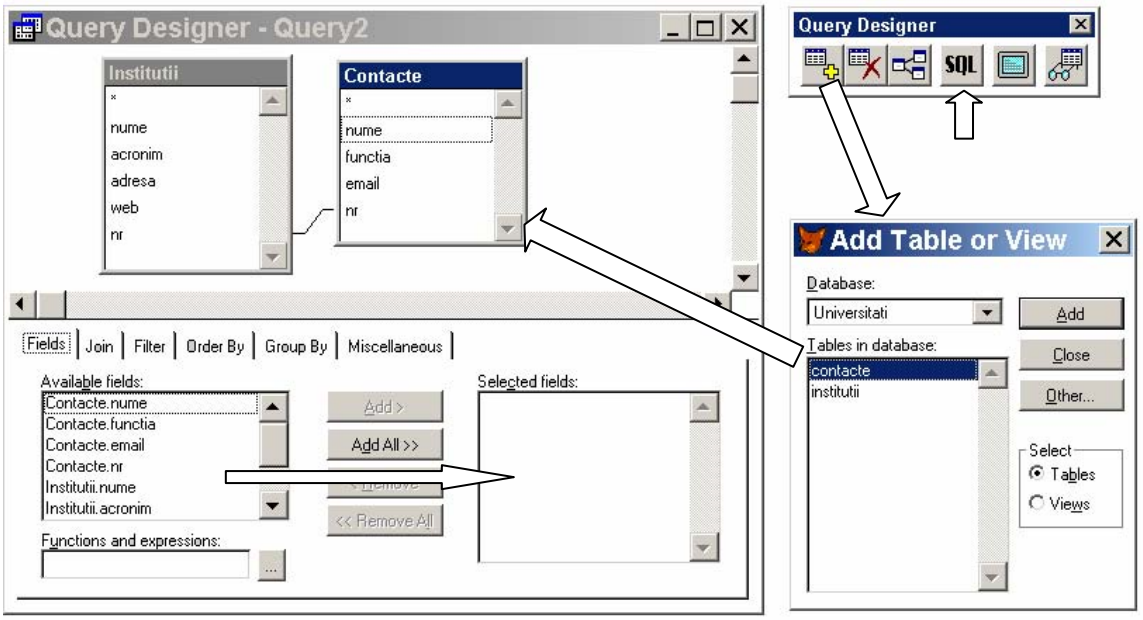
14. Interogarea unei baze de date şi limbajul SQL

SQL este limbaj structurat pe interogări, bază de date interogativă şi limbaj de programare. Utilizarea setului de *instrucțiuni SQL* este legată de crearea de interogări (*Query Designer*), definirea de vederi (*View Designer*), editarea de rapoarte (*Report Designer*). De asemenea, o comandă SQL poate fi dată în fereastra *Command* (*Window/Command Window*) sau inclusă în proceduri şi programe.

Crearea de interogări şi vizualizări

După ce a fost creată baza de date, au fost definite tabelele şi relațiile între acestea, cum este cazul pentru baza de date *universitati* în care cheia primară pentru tabela *institutii* este *nr* iar tabela *contacte* are asociată cheia străină *nr* (care este un index regular la tabelă), iar relația între cele două tabele este definită pe baza acestei relații, se poate crea o nouă interogare.

Pentru crearea unei noi interogări, din *Project Manager* se selectează *Queries*, se apasă butonul *New*... apoi *New Query*. Se lansează atunci aplicația expert *Query Designer*:



Practic, operațiile efectuate până în acest moment sunt rezultatul unor instrucțiuni simple în limbajul SQL, specificate prin intermediul wizard-ului pe care sistemul le-a executat pentru noi, instrucțiuni care se pot vizualiza de la butonul SQL:

FROM clienti!companii INNER JOIN clienti!persoane_contact ;

ON Companii.company_id = Persoane_contact.company_id

Se poate observa sintaxa acestor instrucțiuni:

FROM <nume_bd>!<nume_tabela_1> INNER JOIN <nume_bd>!<nume_tabela_2>;

ON <nume_tabela_1>.<nume_câmp> = <nume_tabela_2>.<nume_câmp>

Se pot *selecta* acum câmpurile care să participe la interogare, ca în exemplul:

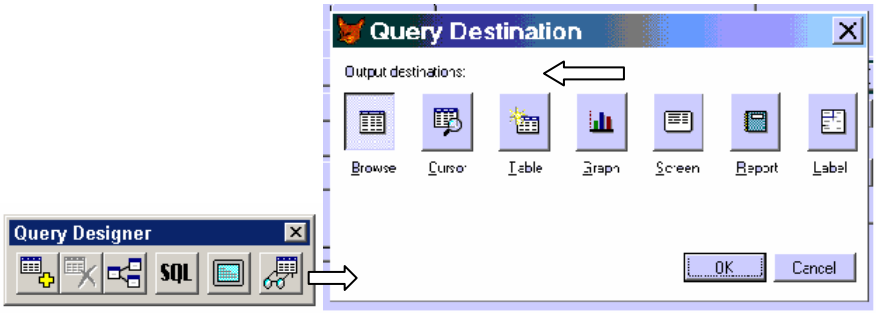
Din tabulatorul *Order By* se poate defini criteriul de ordonare în interogare. Să alegem *Institutii.nr* ca criteriu de ordonare. De asemenea, din tabulatorul Group_By se poate defini un criteriu de grupare a rezultatului interogării. Să alegem ca criteriu de grupare *Institutii.nr*.

În acest caz interogarea va furniza ultima persoană de contact înregistrată pentru fiecare instituție. Dacă se renunță la grupare şi se execută din nou interogarea se vor lista toate persoanele de contact împreună cu nume instituțiilor pe care le reprezintă.

Până aici, sistemul a complectat pentru noi comenzile SQL corespunzătoare:

SELECT Contacte.nume, Contacte.functia, Institutii.nume, Institutii.nr,;
Contacte.poz, Contacte.nr;
FROM universitati!institutii INNER JOIN universitati!contacte ;
ON Institutii.nr = Contacte.nr

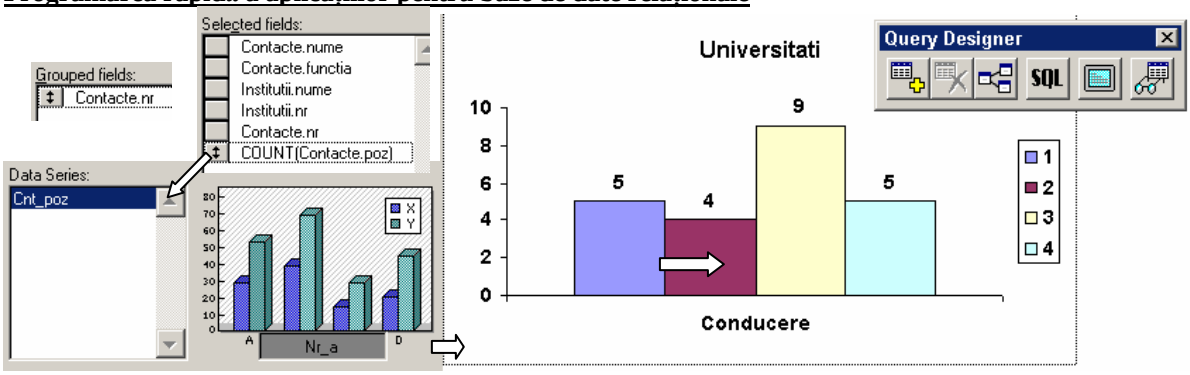
Se poate acum specifica destinația interogării din *Query Designer*, când se activează o fereastră în forma:



Între câmpuri numerice din tabele se pot crea grafice de corespondență (Butonul Graph). Astfel, pentru a face o reprezentare grafică din baza de date Universități să adăugăm câmpul *poz* (*int*) în tabela Contacte, şi să atribuim câte o poziție fiecărei înregistrări.

Lansarea în execuție a interogării se poate face din meniu (*Query/Run Query*) sau din tastatură (*ctrl+Q*).

Astfel, dacă se selectează câmpurile *Institutii.nr* şi *Count(Contacte.poz)* se poate obține cu ajutorul lui Graph Wizard un grafic de dependență a identificatorilor din tabele:



În *Query Destination* se poate preciza ca destinație o fereastră de browse (*Butnul Browse*), caz în care rezultatul afișării este vizualizat într-o fereastră în care se pot parcurge înregistrările şi câmpurile.

Opțiunea Cursor (*Butonul Cursor*) doar selectează înregistrările care corespund condițiilor specificate în interogare, fără a efectua afisări. Opțiunea Table (*Butonul Table*) permite salvarea rezultatului interogării într-o tabelă al cărui nume se va introduce la execuția interogării. Opțiunea Screen (*Butonul Screen*) permite afişarea rezultatului interogării la imprimantă sau trimiterea sa într-un fişier text. Opțiunile Report şi Label (*Butoanele Report* şi *Label*) permite trimiterea rezultatului interogării către un raport sau etichetă, a căror construcție se va discuta mai târziu.

Aceeaşi paşi care au fost urmați în crearea unei interogări se efectuează şi pentru crearea unei vederi.

Vederile combină calitățile tabelelor cu cele ale interogărilor; la fel ca în cazul interogărilor, se poate crea o vedere pentru extragerea unui set de date din una sau mai multe tabele asociate; la fel ca la o tabelă, o vedere poate fi folosită pentru actualizarea informațiilor şi stocarea permanentă a informațiilor pe hard-disk. Vederile pot fi însă folosite şi la culegerea şi modificarea datelor off-line în afara sistemului principal.

Dacă se doreşte accesul la date stocate pe un server la distanță, trebuie creată o vedere externă. În acest scop, trebuie întâi să ne conectăm la o sursă de date. O sursă de date externă este de obicei un server extern pentru care este instalat un driver ODBC (open database connectivity) şi s-a atribuit un nume sursei de date ODBC. Pentru a beneficia de o sursă de date, trebuie să fie instalat driverul ODBC. Din mediul VFP se poate atunci defini sursa de date şi conexiunile.

15. Crearea de vederi locale

O vedere locală permite să extragem înregistrări dintr-o tabelă, să efectuăm modificări asupra înregistrărilor extrase şi să transmitem modificările în tabelele sursă.

Se pot crea vederi din tabele locale, din alte vederi, din tabele stocate pe un server, din surse de date situate la distanță, cum ar fi Serverul SQL Mirosoft prin intermediul protocolului ODBC. Se poate configura VFP să efectueze modificările în tabelele sursă în momentul efectuării modificărilor în vedere.

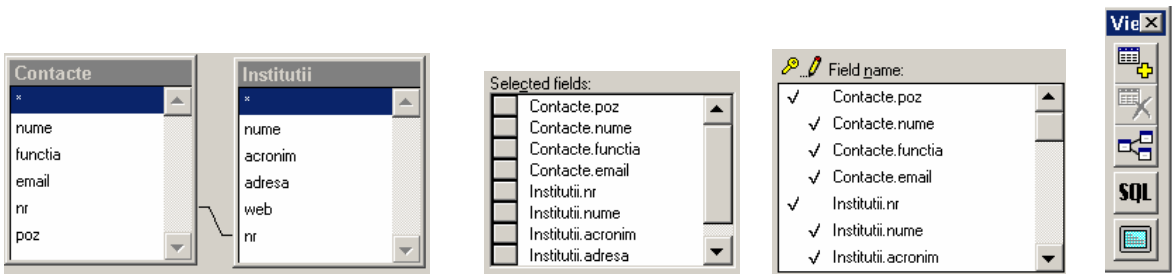
O vedere locală se poate crea cu ajutorul aplicației expert *View Designer*. Astfel, avem următoarele posibilități:

1. Din fereastra de comenzi (*Window/Command Window*) cu comanda *create sql view*;
2. Din bara de instrumente: *New/View/New file* sau din meniu *File/New/View/New file*;
3. Din *Project Manager* se selectează o bază de date, se alege *Local Views* şi butonul *New*;

Se poate folosi *View Designer* pentru a crea vederi din baza de date *Universitati.dbc*. Se pot construi două vederi: una care să redea conținutul bazei de date ordonat alfabetic crescător după persoanele de contact şi alta care să redea conținutul ordonat alfabetic crescător după numele universității şi apoi alfabetic descrescător după funcție.

Vederea bazei de date Universitati după persoanele de contact

1. Se deschide proiectul *Universitati.pjx* (*Open/Project*);
2. Se selectează baza de date *Universitati*;
3. Se expandează conținutul bazei de date (⊞ → ⊟);
4. Se selectează din aceasta *Local views*;
5. Se apasă butonul *New...* şi se alege opțiunea *New View*;
6. La fel ca la interogări, se includ tabelele *contacte* şi *institutii* în *View Designer*;

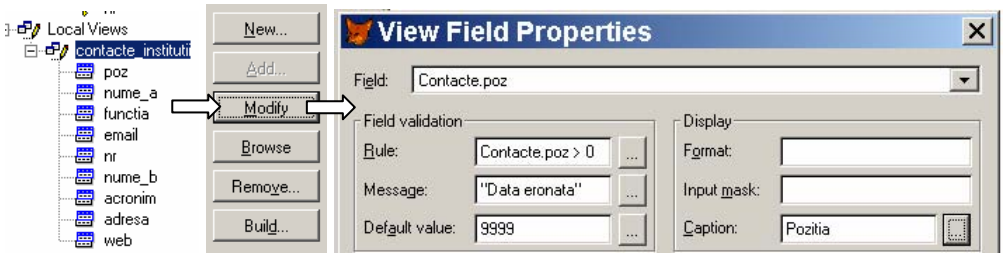


7. Se selectează toate câmpurile, mai puțin *Contacte.nr*;
8. Se alege relația de ordine dorită (*Order By/Ordering criteria: Contacte.nume* ↑);
9. În tabulatorul *Update Criteria* se precizează cheile primare *Contacte.poz* şi *Institutii.nr*;
10. Se apasă butonul *Update All* pentru a face posibilă modificarea tuturor câmpurilor din tabele pe baza modificărilor efectuate în vedere; comanda SQL transmisă sistemului se poate vedea din bara de instrumente a lui *View Designer*.

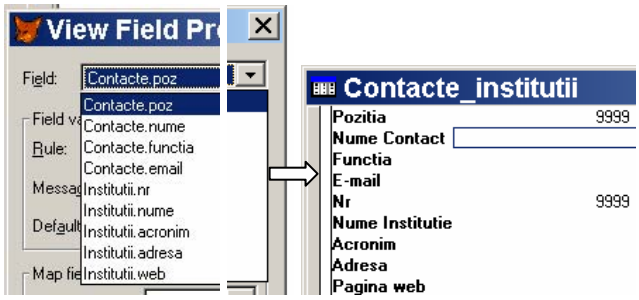
SELECT Contacte.poz, Contacte.nume, Contacte.functia, Contacte.email,;

Institutii.nr, Institutii.nume, Institutii.acronim, Institutii.adresa,;
Institutii.web;
FROM universitati!institutii INNER JOIN universitati!contacte ;
ON Institutii.nr = Contacte.nr;
ORDER BY Contacte.nume

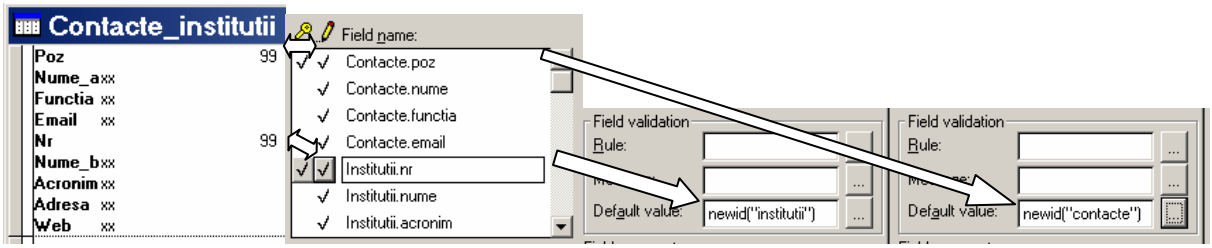
11. Se salvează vederea din bara de instrumente (*Save*) sau din meniu (*File/Save*) de exemplu cu numele *contacte_institutii*;
12. Se pot acum defini opțiuni de vizualizare ale câmpurilor vederii; se selectează vederea *contacte_institutii* din *Project Manager* şi se apasă *Modify*;



13. În *View Designer* la tabulatorul *Fields* se apasă butonul *Properties*;
14. Se pot aici defini reguli de validare pentru câmpurile din vedere (*Field validation*) la fel ca la tabele şi etichete pentru afişarea lor în fereastra de *Browse* (*Display/Caption*);
15. Se setează din această fereastră de dialog proprietățile pentru toate câmpurile vederii;



16. Se pot adăuga acum persoane de contact şi instituțiile corespunzătoare dacă se lansează în execuție vederea creată: *Query/Run Query* când se poate afişa în forma *Append Mode* şi *Edit* din meniu de la *View*;
17. Se poate seta opțiunea pentru a efectua modificările în tabelele incluse în vedere; pentru aceasta se selectează opțiunea *Send SQL updates* din tabulatorul *Update Criteria* al vederii *contacte_institutii* cu ajutorul lui *View Designer*;
18. Se pot acum adăuga persoane de contact şi instituții, modificarea efectuându-se automat în tabelele bazei de date; pentru aceasta, se execută vederea (*Ctrl+Q*) şi se adaugă o nouă înregistrare în aceasta (*Ctrl+Y*);
19. Pentru ca modificările să devină active trebuie închis proiectul (*Close all data* în fereastra de comenzi);
20. La o nouă deschidere a acestuia se pot observa modificările în tabele;



21. Pentru a insera noile chei pentru instituție și contact este necesară selectarea explicită a modificării în vedere (*View Designer/Update Criteria/Field name*);
22. Sistemul salvează vederea în baza de date sub forma unui tabel liber cu același nume cu vederea, care se poate observa în *Project Manager* la încărcarea aplicației expert *Database Designer*;

Vederea bazei de date Universitati după institutii și apoi persoane de contact

23. Se urmează același succesiune de pași ca mai sus, însă se alege la relația de ordine dorită (*Order By/Ordering criteria: Institutii.nume* ↑ și *Contacte.nume* ↓);

Stocarea de proceduri în baza de date pentru câmpurile autoincrement

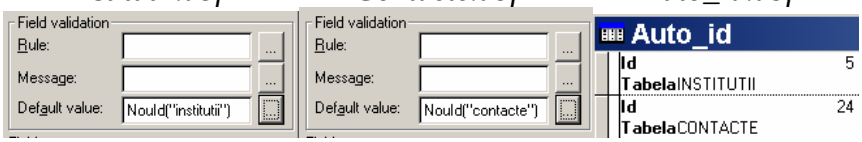
Baza de date *Universitati* este acum pregătită pentru a construi o metodă de autoincrementare a valorilor pentru cheile primare. Este necesară crearea în baza de date a unei tabele care să memoreze viitoarele valori ale cheilor primare pentru fiecare tabelă în parte. Ulterior, se folosește această tabelă de către o procedură stocată în baza de date pentru a stabili valorile de increment. Se urmează pașii:

1. Se adaugă în *Universitati.bdc* tabela auto_id cu structura (*id int; tabela char(50)*);
2. Se completează *Auto_id.dbf* cu valorile corespunzătoare;
3. Se indexează *Auto_id.dbf* după cele două câmpuri ale sale cu indecși regulari;
4. Se creează o funcție pentru autoincrementare; fie aceasta *NouId()*; Se poate selecta codul procedurii *NewId()* din baza de date *Books.dbc* din subdirectorul de instalare al *VFP*: *Wizards/Template/Books/Data* și copia în baza de date *Universitati.dbc*; Acesta se modifică corespunzător, ținând cont de numele actuale:

```
FUNCTION NouID(tcAlias)
 LOCAL lcAlias, ;
              lnID, ;
    lcOldReprocess, ;
    lnOldArea
 lnOldArea = SELECT()
 lnID = 0
 IF PARAMETERS() < 1
  lcAlias = UPPER(ALIAS())
 ELSE
  lcAlias = UPPER(tcAlias)
 ENDIF
 lcOldReprocess = SET('REPROCESS')
 *-- Lock until user presses Esc
 SET REPROCESS TO AUTOMATIC
 IF !USED("AUTO_ID")
  USE universitati!auto_id IN 0
 ENDIF
 SELECT auto_id
 IF SEEK(lcAlias, "auto_id", "tabela")
  IF RLOCK()
   lnID = auto_id.id
   REPLACE auto_id.id WITH auto_id.id + 1
   UNLOCK
  ENDIF
 ENDIF
 SELECT (lnOldArea)
 SET REPROCESS TO lcOldReprocess
 RETURN lnID
ENDFUNC
```

5. Se exploatează funcția *NouId()* prin definirea autoincrementelor în tabelele *Institutii* şi *Contacte*;

Institutii.dbf *Contacte.dbf* *Auto_Id.dbf*

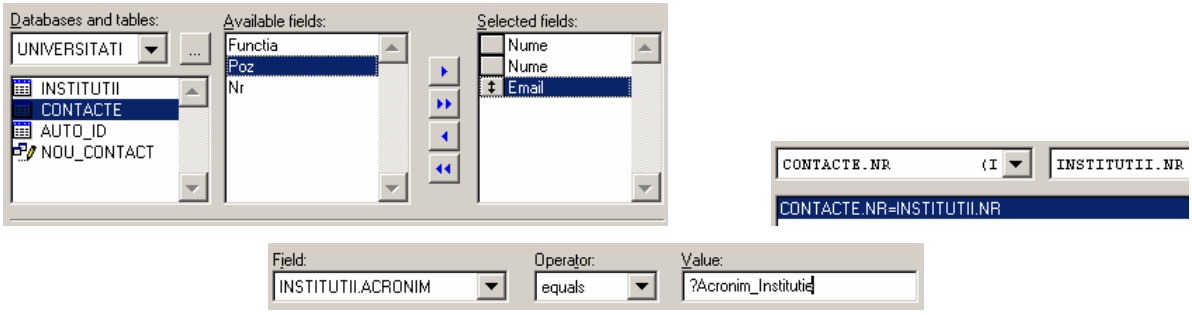


6. Se construieşte vederea *Nou_Contact* pe baza tabelelor *Contacte* şi *Institutii*; se includ toate câmpurile din tabela *Contacte* şi câmpul nume din tabela *Institutii* şi se procedează identic cu cazul anterior;
7. Se construieşte vederea *Noua_Institutie* pe baza tabelei *Institutie*, în care se includ toate câmpurile din tabelă; se procedează la fel cu cazurile anterioare;

Utilizarea Wizard-ului pentru crearea de vederi

Pentru vizualizarea tuturor persoanelor care aparţin unei instituții cu un anumit acronim se poate realiza o vedere parametrizată care să le includă. Se urmează paşii:

1. Se selectează *Local Views*; Se activează *New../View Wizard*;
2. Se aleg câmpurile *contacte.nume*, *institutii.nume*, *contacte.email*, *institutii.nr* şi *contacte.nr*;



3. Se defineşte relația *institutii.nr = contacte.nr*;
4. Se includ toate liniile din tabela *Contacte* (*All Rows from this table*);
5. Se defineşte interogarea în tabela *Institutii* după *Acronim* în forma *?Acronim_Institutie*;
6. Se salvează vederea sub numele *CautContact*;
7. Se execută interogarea (*Run Query*);

În acelaşi mod se poate defini vederi care să extragă o persoană de contact pentru o anume universitate sau să găsească pagina web a unei universități.

16. Lucrul cu fereastra de comenzi

Pe lângă facilitățile oferite de mediul vizual, sistemul VFP pune la dispoziția utilizatorului şi o interfață de tip text, în care utilizatorul are posibilitatea să lanseze comenzi sistemului. Există un set de comenzi pe care sistemul le acceptă şi le execută din fereastra de comenzi.

Se pot lansa comenzi practic pentru orice proces care poate fi accesat din meniuri. Cele mai importante comenzi sunt redate în continuare.

Vizualizarea structurii tabelei curente se realizează cu comenzile:

LIST STRUCTURE [TO PRINTER [PROMPT] | TO FILE <file>]

DISPLAY STRUCTURE [IN <expN> | <expC>] [TO PRINTER [PROMPT] | TO FILE <file>]

unde:

IN <expN | expC> specifică numărul zonei de lucru sau aliasul tabelei,

TO PRINTER trimite informația la imprimantă,

PROMPT introduce dialog (confirmare) înainte de imprimare,

TO FILE <nume_fis> salvare informație în fişier,

Exemplu:

DISPLAY STRUCTURE IN 3 TO FILE <str_baz1.txt>

Aliasul tabelei se poate vizualiza din meniu la opțiunea *Window/Data Session*.

Structura unei baze se poate ulterior modifica în sensul adăugării/eliminării unor câmpuri sau modificarea mărimii câmpurilor prin comanda:

MODIFY STRUCTURE

Comenzile DISPLAY şi LIST sunt folosite pentru afişarea conținutului tabelelor.

DISPLAY [[FIELDS] <lista campurilor>]

[<domeniu>] [FOR <expL1>] [WHILE <expL2>] [OFF]

[TO PRINTER [PROMPT] | TO FILE <numefis>] [NOCONSOLE] [NOOPTIMIZE]

Exemplu:

USE Contacte

DISPLAY ALL

afişează toate câmpurile şi înregistrările cu oprire la fiecare pagină. Pentru afişarea numai a câmpurilor specificate lansăm de exemplu comanda:

DISPLAY ALL FIELDS NUME, EMAIL

Afişarea înregistrărilor care îndeplinesc o condiție (au valoarea câmpului POZ mai mare decât 10):

DISPLAY ALL FOR POZ > 10

Afişarea înregistrărilor care au şirul de caractere din câmpul nume mai mare sau egal decât şirul Ha:

DISPLAY ALL FOR NUME >= “Ha”

DISPLAY ALL WHILE NUME > „Ha”

afişează înregistrările cât timp expresia logică NUME > „Ha” este adevărată.

DISPLAY ALL OFF

nu se afişează coloana 0 ce conține numărul de ordine al înregistrărilor

DISPLAY

afişează numai înregistrarea curentă.

În cadrul expresiei logice pot fi folosiți operatorii logici AND, OR, NOT

DISPLAY NEXT 5 FOR nume < ’R’ AND nume > ’A’

Comanda LIST are sintaxa:

LIST

[FIELDS <expr list>]

[<scope>] [FOR <expL1>] [WHILE<expL1>] [OFF]

[NOCONSOLE] [NOOPTIMIZE] [TO PRINTER [PROMPT] | TO FILE <file>]

LIST afişează toate înregistrările fiind echivalentă cu comanda DISPLAY ALL;

LIST NEXT 3 afişează următoarele 3 înregistrări şi mută indicatorul corespunzător (pe a 3-a înregistrare);

Exemple:

LIST FOR NOT NUME= ‘ Giurgiu’

LIST FOR "Gi" $ nume

afişează înregistrările ce conțin şirul “Gi” în câmpul nume.

LIST FOR LIKE ("*iurg*", nume)

Zona de lucru este o zonă de memorie rezervată pentru păstrarea informațiilor necesare lucrului cu o tabelă. Zona de lucru 1 este curentă implicit după lansarea FoxPro. La un moment dat o singură zonă este activă.

Prin comanda SELECT <expN> | <expC> se stabileşte zona de lucru curentă:

SELECT 7

Zona de lucru indice 7 a devenit curentă, activată.

Funcția SELECT () returnează numărul zonei de lucru curente.

Operatorul ? este folosit pentru afişarea unei valori pe ecran. Astfel,

? SELECT( )

va afişa:

7

Pentru a lucra cu o tabelă ea trebuie deschisă. Deschiderea tabelei înseamnă înscrierea caracteristicilor esențiale ale bazei (structura, numărul înregistrărilor, etc.) într-o anumită zonă de lucru, nu neapărat cea curentă.

Funcția USED (expN | expC) returnează .T. dacă în zona de lucru specificată prin argument este deschisă o tabelă (în caz contrar returnează .F.). Argumentul expC al funcției se referă la aliasul tabelei (un alt nume al tabelei).

```
? USED(3)
.F.
```

Funcția DBF(expN | expC) returnează numele tabelei deschise în zona specificată prin argument sau numele bazei de alias expC:

```
SELECT 3
? SELECT()              && se afişează 3
USE tabela1
LIST STRUCTURE
? USED(3)               && se afişează .T.
? DBF(3)                && se afişează tabela1
USE                     && se închide tabela1 din 3
? USED(3)                   && afişează .F.
USE tabela1 IN 4            && se deschide tabela1 în zona de lucru 4
LIST STRUCTURE              && nu se listează structura
USE tabela1 IN 3 AGAIN      && se deschide tabela tabela1 în zona de lucru curentă 3 && rămânând deschisă şi în zona 4
```

```
Close all data
```

închide toate tabelele din mediul de lucru şi odată cu acestea toate bazele de date şi proiectele ce le utilizează.

Poziționarea pe o înregistrare se poate face cu ajutorul comenzilor GO, GOTO şi SKIP. FoxPro atribuie în mod reflex un număr de înregistrare fiecărui articol din fişierul deschis.

```
GO [RECORD] <expN1> [IN <expN2> | IN <expC>]
GO TOP | BOTTOM [IN <expN2> | IN <expC>]
GOTO [RECORD] <expN1> [IN <expN2> | IN <expC>]
GOTO TOP | BOTTOM [IN <expN2> | IN <expC>]
```

Comanda GO număr_articol permite trecerea imediată pe articolul având numărul specificat (expN1) :

```
Go 20
Disp
```

Este suficient să precizăm primele 4 litere din numele comenzii pentru ca sistemul să recunoască comanda.

Pentru saltul peste un anumit număr de înregistrări se poate folosi comanda SKIP

```
SKIP  –1        && acceptă şi valori negative, pentru saltul înapoi;
```

Cu ajutorul comenzii SET se poate defini o listă de câmpuri implicită la afişare. De exemplu:

```
SET FIELDS TO NUME, EMAIL
LIST
SET FIELDS OFF
LIST
```

Filtrele pentru înregistrări se definesc tot cu ajutorul comenzii SET:

```
SET FILTER TO POZ>10
LIST
SET FILTER TO
LIST
```

Pentru modificarea înregistrărilor din fişier se pot folosi comenzile EDIT, BROWSE, REPLACE şi CHANGE.

Comanda EDIT modifică înregistrarea curentă. Comanda EDIT 4 modifică înregistrarea 4.

EDIT FIELDS NUME, EMAIL (restricționare la câmpurile enumerate)

EDIT FOR nr = 4 (restricționare la înregistrările care satisfac condiția).

Comanda REPLACE permite un alt mod de modificare a valorii unor câmpuri. Comanda are sintaxa :

REPLACE <field1> WITH <expr1> [ADDITIVE] [, <field2> WITH <expr2> [ADDITIVE]] ... [<scope>] [FOR <expL1>] [WHILE <expL2>]

unde <scope>: ALL, NEXT <expN>, RECORD <expN>, sau REST

Exemple:

```
use Institutii
goto 3
REPLACE nume WITH "Popescu"
use Contacte
REPLACE ALL poz WITH poz*2
REPLACE ALL nume WITH UPPER(nume)
```

Opțiunea ADITIVE operează numai pentru câmpuri de tip memo. Dacă ea este folosită, valoarea indicată după WITH va fi adăugată la sfârşitul conținutului curent al câmpului.

Suprimarea înregistrărilor (se realizează în doi paşi):

Comanda DELETE marchează înregistrările ca fiind suprimate dar nu le şterge efectiv din fişier, şi apoi cu comanda PACK se reorganizează fişierul suprimând efectiv articolele marcate (marcarea înregistrărilor se poate face şi din fereastra comenzii BROWSE).

Comanda DELETE permite ştergerea unuia sau mai multor articole:

DELETE [<scope>] [FOR <expL1>] [WHILE <expL2>]

<scope>: ALL, NEXT <expN>, RECORD <expN>, sau REST

Exemple :

DELETE RECORD 12 şterge articolul 12

DELETE NEXT 4 şterge următoarele 4 articole

DELETE ALL FOR nr=4

DELETE ALL şterge toate articolele

DELETE REST şterge articolele de la poziția curentă până la sfârşitul fişierului.

Comanda RECALL restabileşte înregistrările suprimate cu DELETE.

RECALL [<scope>] [FOR <expL1>] [WHILE <expL2>]

<scope>: ALL, NEXT <expN>, RECORD <expN>, sau REST

Indexarea unui tabel se poate realiza din fereastra de comenzi.

INDEX ON câmp TAG câmp DESCENDING

Dacă se doreşte o indexare după mai multe câmpuri este necesară o comandă de tipul:

INDEX ON câmp1+câmp2 TAG nume

De exemplu, dacă se doreşte o indexare după nume şi apoi, pentru acelaşi nume după cifra de afaceri, se poate folosi o indexare în forma:

INDEX ON nume+STR(ca) TAG x

În exemplu s-a convertit cifra de afaceri ca în şir caractere (cu ajutorul funcției STR) şi s-au concatenat şirurile de caractere nume şi STR(ca). La redeschiderea fişierului astfel indexat se va putea folosi de exemplu :

USE parteneri ORDER TAG x

Dacă în timpul exploatării se doreşte trecerea la indexarea după un alt index se foloseşte comanda SET ORDER TO:

```
USE contacte ORDER nume
LIST
SET ORDER TO poz
LIST
```

Într-un fişier indexat se pot efectua căutări cu comanda SEEK:

```
USE contacte ORDER nr
SEEK 1
```

În exemplul precedent numărătorul de înregistrări se poziționează pe articolul a cărui câmp nr = 1. Dacă un astfel de articol nu există contorul rămâne neschimbat şi funcția FOUND() returnează .F.. Comenzile SET EXACT ON/OFF modifică modul de operare a comenzii SEEK. Dacă este activă comanda SET EXACT OFF căutarea se va face numai după primele caractere ale câmpului cheie.

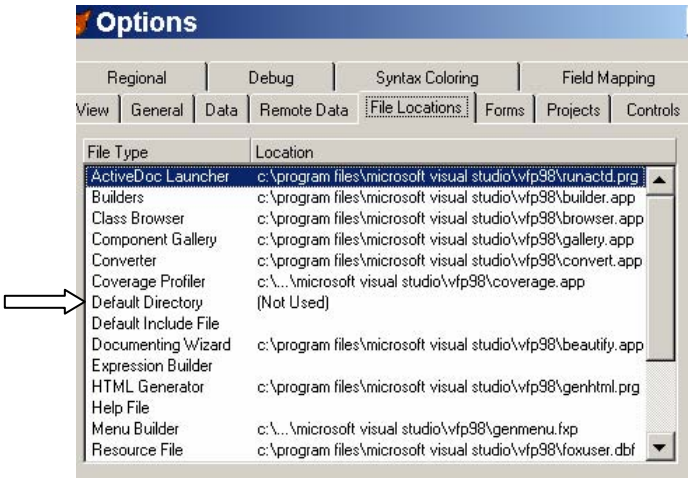
Se poate defini directorul implicit pentru fişiere. Pentru a defini un director implicit pentru sesiunea de lucru curentă se foloseşte comanda:

SET DEFAULT TO <nume_director>

De exemplu:

SET DEFAULT TO C:\UTILIZATORI\STUDENT\VASILE

Pentru a defini directorul implicit care să fie încărcat la fiecare pornire a VFP, din meniu, Tools/Options/File Locations, când se activează o fereastră în forma:



Comenzile se pot integra în fişiere de comenzi sau programe, care au facilități suplimentare față de facilitățile oferite de fereastra de comenzi. Oricum, oricare din succesiunea de comenzi (sau toate la un loc) care au fost lansate în fereastra de comenzi se pot salva într-un program. Un nou program se creează din *New/Program/New File*.

Se lansează în execuție cu comanda DO:

DO ProgramName1 | ProcedureName [IN ProgramName2] [WITH ParameterList]

Argumente:

- ProgramName1 – specifică numele programului de executat;

Dacă nu se include nici o extensie, FVP caută să execute în următoarea ordine:

ProgramName1.exe (versiunea executabilă)

ProgramName1.app (o aplicație)

ProgramName1.fxp (versiunea compilată)

ProgramName1.prg (programul)

Pentru a executa un program de tip meniu, o formă, o interogare, trebuie să i se includă şi extensia (*.mpr*, *.spr*, sau *.qpr*).

ProcedureName – specifică numele procedurii de executat din programul curent.

IN ProgramName2 clauză care comunică VFP să caute procedura într-un fişier anume;

WITH ParameterList – specifică parametrii de transmis programului sau procedurii.

Se poate merge într-un program la apel recusiv la peste 128 de apeluri DO.

17. Expresii

Expresiile care pot fi construite cu ajutorul tipurilor de dată predefinite puse la dispoziție de mediul VFP (de exemplu în *Expresion Builder*) se găsesc documentate în MSDN, *Active Subset: Visual Fox Pro Documentation/Contents/Reference/Language Overview/Overview of the Language/Building Expresions*, sau dacă este instalată versiunea MSDN'98, atunci se poate încărca în *Internet Explorer* pagina *mk:@MSITStore:C:\Program%20Files\Microsoft%20Visual%20Studio\MSDN98\98VSa\1033\foxhelp.chm::/html/conbuilding_expressions.htm*.

- Operatorii pe expresii de tip caracter sunt descrişi în:

  foxhelp.chm::/html/tblcharacter_operators.htm

- Operatorii pe expresii de tip dată şi oră sunt descrişi în:

  foxhelp.chm::/html/tbldate_and_time_operators.htm

- Operatorii numerici:

  foxhelp.chm::/html/tblnumeric_operators.htm

- Operatorii logici:

  foxhelp.chm::/html/tbllogical_operators.htm

18. Lucrul cu funcțiile FVP – exemple de utilizare

ABS()

```
? ABS(-45)       && Afişează 45
? ABS(10-30)     && Afişează 20
? ABS(30-10)     && Afişează 20
STORE 40 TO gnNumber1
STORE 2 TO gnNumber2
? ABS(gnNumber2-gnNumber1)     && Afişează 38
```

CHR() – într-un program:

```
CLEAR
FOR nCOUNT = 65 TO 75
  ? nCount                    && Display numeric value
  ?? ' ' + CHR(nCount)        && Display character
ENDFOR
```

DATE()

```
CLEAR
SET CENTURY OFF
? DATE( )  && Afişează today's date without the century
SET CENTURY ON
? DATE( )  && Afişează today's date with the century
? DATE(1998, 02, 16)  && Afişează a year 2000-compliant Date value
```

DATETIME()

```
tNewyear = DATETIME(YEAR(DATE( ) ) + 1, 1, 1)  && Next New Year
tToday = DATETIME( )
nSecondstonewyear = tNewyear - tToday
CLEAR
? "There are " + ALLTRIM (STR(nSecondstonewyear)) ;
  + " seconds to the next New Year."
CLEAR
SET CENTURY ON
SET DATE TO AMERICAN
? DATETIME(1998, 02, 16, 12, 34, 56) && Afişează 02/16/1998 12:34:56 PM
STORE {^1998-03-05} TO gdBDate
```

DAY()

```
CLEAR
? CDOW(gdBDate)  && Afişează Thursday
? DAY(gdBDate) && Afişează 5
? 'That date is ', CMONTH(gdBDate), STR(DAY(gdBDate),2)
```

DBUSED() && Returnează true (.T.) dacă baza de date specificată este deschisă.

```
DBUSED('Universitati')
```

DELETED()

DELETED([cTableAlias | nWorkArea]) Returnează o valoare logică ce indică dacă înregistrarea curentă este marcată pentru ştergere.

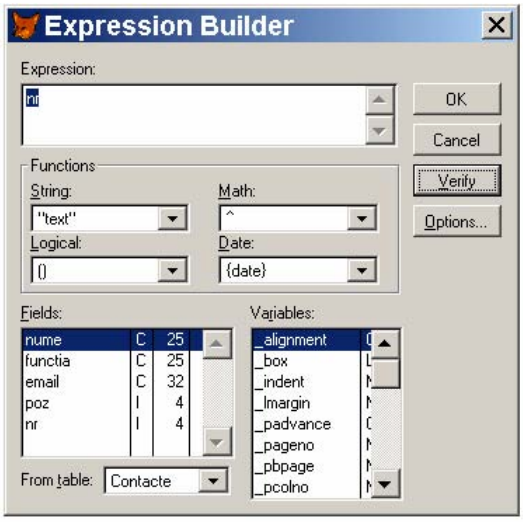
DMY() && expresie de tip caracter în formatul day-month-year

```
CLEAR
SET CENTURY OFF
? DMY(DATE( ))
SET CENTURY ON
? DMY(DATE( ))
```

19. Constructorul de expresii (*Expression Builder)*

Constructorul de expresii este un instrument esențial în exploatarea unei baze de date. În elementele discutate el intervine în:

- definirea expresiilor pentru indecşi;
- definirea regulilor de validare, mesajelor şi valorilor implicite pentru câmpuri şi înregistrări;
- definirea funcțiilor şi expresiilor pentru noi câmpuri în interogări şi vederi;
- definirea etichetelor de câmpuri la vederi;



Constructorul de expresii poate fi accesat din aplicațiile expert, ferestre, constructoare şi wizard-uri. Permite definirea de expresii în care intervin câmpuri din tabele şi vederi. Tipul expresiei construite trebuie să fie compatibil cu tipul acceptat de caseta de editare din care a fost încărcat. De exemplu, dacă a fost încărcat dintr-o casetă de editare în care se acceptă o valoare de tip şir de caractere (cum este cazul pentru *Fields Caption*) atunci valoarea expresiei construite trebuie să fie de tip şir de caractere.

O expresie poate fi:

- simplă (numele unui câmp);
- complexă (implicând de exemplu un IF *imediat*);

Pentru a construi expresii, se pot scrie în caseta de editare pentru expresii sau se pot selecta intrări din lista *drop-down* a funcțiilor pentru a se insera în caseta de editare.

Pentru operații cu şiruri de caractere sau câmpuri ce conțin şiruri de caractere, sunt utile funcțiile *ALLTRIM(), LTRIM(), RTRIM(), PADL(), PADR(), PADC(), SUBSTR(), LEFT(), RIGHT(), UPPER(), LOWER(), PROPER(), STR()*.

Să presupunem că avem şirul de caractere: " ABBA ". Atunci:

```
? ALLTRIM("  ABBA  ss ")      && va afişa "ABBA  ss"
? LTRIM("  ABBA  ")           && va afişa "ABBA  "
```

```
? RTRIM("    ABBA      ")          && va afişa "    ABBA"
? PADL("    ABBA      ",12,"X")    && va afişa "XXX    ABBA      "
? PADR("    ABBA      ",12,"X")    && va afişa "    ABBA      XXX"
? PADC("    ABBA      ",12,"X")    && va afişa "X    ABBA      XX"
? SUBSTR("    ABBA      ",2,4)     && va afişa "  ABB"
? SUBSTR("    ABBA      ",2)       && va afişa "  ABBA      "
? LEFT("    ABBA      ",3)         && va afişa "  A"
? RIGHT("    ABBA      ",5)        && va afişa "BA      "
? LOWER("    ABBA      ")          && va afişa "    abba      "
? UPPER(LOWER("    ABBA      ")) && va afişa "    ABBA      "
? PROPER("    ABBA    ss  ")      && va afişa "    Abba    Ss  "
? STR(12+13)                      && va afişa "          25"
? STR(12+13,1)                    && va afişa "*"
? STR(12+13,2)                    && va afişa "25"
? STR(12.2)                       && va afişa "          12"
? STR(12.2,8,3)                   && va afişa "   12.200"
? STR(12.2,6,3)                   && va afişa "12.200"
? STR(12.2,5,3)                   && va afişa "12.20"
```

Pentru orice funcție selectată din *Expression Builder* sistemul VFP afișează pe linia de stare (*Status Bar*, ultima linie din fereastra aplicației VFP) informații cu privire la aceasta.

Din caseta *From table* a *Expression Builder* este permisă selectarea unei tabele din cele deschise (cu *USE*).

Caseta *Variables* permite utilizarea în definirea expresiei a variabilelor sistem, a tablourilor şi variabilelor obişnuite anterior definite de utilizator.

Opțiunea *Verify* verifică sintaxa expresiei. Este însă doar o verificare formală, în timpul execuției nu este neapărat ca eroarea semnalată să fie reală. De exemplu definirea unei etichete în forma *Nume Contact* în caseta vederii *CautContact* nu este o eroare în execuție deşi opțiunea *Verify* semnalează absența ghilimelelor: "*Nume Contact*".

Aplicație: se pot lista toate instituțiile au ca ultime două litere în acronim CN:

```
use institutii
browse font "Courier New" fields nume, acronim for upper(right(rtrim(acronim),2)) = "CN"
```

Rezultatul este în forma:

Institutii

Nume	Acronim
Universitatea Tehnica Cluj-Napoca	UTCN
Universitatea de Stiinte Agricole si Medicina Veterinara Cluj-Napoca	USAMVCN

20. Programare

În general, orice ce poate fi făcut cu un program, se poate face manual dacă avem destul timp. De exemplu, dacă căutăm o persoană de contact în tabela *contacte* de exemplu Gheorghe Lazea, acest lucru poate fi făcut manual urmând secvența de instrucțiuni:

1. Din meniul *File* selectăm *Open*;
2. Din caseta *File of type* selectăm *Table*;
3. Selectăm tabela *Contacte.dbf* în lista fişierelor;
4. Din meniul *View* selectăm *Browse*;
5. Parcurgând conținutul tabelei, urmărind câmpul *Nume* identificăm înregistrarea pentru *Gheorghe Lazea*;

Din fereastra de comenzi, poate fi făcut acelaşi lucru scriind următoarele instrucțiuni:

USE Customer

LOCATE FOR Nume = "Gheorghe Lazea"

BROWSE

După ce am localizat înregistrarea putem să îi modificăm conținutul pentru a scrie de exemplu numele cu litere mari:

REPLACE nume WITH UPPER(nume)

sau să revenim la scrierea cu prima literă din nume mare:

REPLACE nume WITH PROPER(nume)

Instrucțiunile şi comenzile pot fi integrate în *programe*. Acest fapt conferă unele avantaje:

- programele pot fi modificate şi executate din nou;
- se pot executa programele din meniuri, forme şi bare de instrumente;
- programele pot executa alte programe;

Un program în VFP se poate crea pe calea *New/Program,New File* sau din fereastra de comenzi cu comanda *modify command*.

Un program simplu care să schimbe toate numele la litere mari este:

use contacte

scan

replace nume with ;

upper(nume)

endscan

unde *scan* parcurge toate înregistrările din tabel şi execută instrucțiunile între *scan* şi *endscan* iar *;* indică că o comandă (comanda *replace*) se continuă pe linia următoare. Pe lângă comenzi, programele permit scrierea şi de instrucțiuni (cum este cazul instrucțiunii *scan*).

Pentru a restaura situația anterioară în fișier este suficient ca să modificăm programul înlocuind funcția *UPPER(nume)* cu funcția *PROPER(nume)*.

Pentru a executa o succesiune de comenzi şi instrucțiuni în fereastra de comenzi, se introduc acestea în fereastra de comenzi (de exemplu cu *Copy* şi *Paste*), se selectează (de exemplu de pe tastatură cu *Shift* şi *săgeți*) după care se apasă enter (permite şi execuția de instrucțiuni).

- pentru a crea un program: *New/Program,New File* sau comanda *modify command*;
- pentru a salva un program: *File/Save*;
- pentru a deschide un program: *File/Open/File type: program/Open* sau *modi comm <nume_program>*; se poate face şi *modi comm ?* când se activează o fereastră de dialog;
- pentru a executa un program: *Program/Do...* sau cu comanda *do <nume_program>*;

Din perspectiva VFP există următoarele containere pentru date:

- variabile (elemente individuale de date stocate în RAM); pentru a crea o variabilă este suficient să precizăm numele variabilei şi valoarea corespunzătoare; dacă nu există sau este de tip incompatibil, aceasta va fi creată când se distruge variabila anterioară cu acelaşi nume; alternativa pentru *store* este operatorul = cu semnificația cunoscută din limbajul *C*; variabilele pot fi publice, locale şi private;
- de exemplu:

```
STORE DATE( ) TO local gdDate
STORE 50 TO gnNumeric
STORE 'Hello' TO gcCharacter
STORE .T. TO glLogical
STORE $19.99 TO gyCurrency
DIMENSION gaMyArray(2,2)
SET COMPATIBLE OFF
STORE 2 TO gaMyArray
CLEAR
DISPLAY MEMORY LIKE g*
```

- şiruri sau matrice (serii ordonate de elemente, stocate în RAM); se declară cu una din comenzile *DIMENSION* sau *DECLARE*; de exemplu:

```
DIMENSION ArrayName[5,2]
ArrayName[1,2] = 966789
? ArrayName[1,2]
```

- tabele şi înregistrări; o înregistrare poate avea peste 255 de câmpuri;

Pentru lucrul cu tabele sunt utile comenzile *SCATTER, GATHER, COPY TO ARRAY,* şi *APPEND FROM ARRAY*.

Instrucțiunea IF are sintaxa:

IF <expresie_logică> [THEN]
 [<comenzi şi instrucțiuni>]
[ELSE
 <comenzi şi instrucțiuni>]
ENDIF

Un exemplu de program cu instrucțiunea IF: să se afişeze înregistrările care îndeplinesc o condiție oarecare.

```
CLOSE DATABASES
OPEN DATABASE ('Universitati')
USE Contacte                                  && deschide tabela contacte
GETEXPR 'Introdu conditia de localizare ' TO gcTemp;
  TYPE 'L' DEFAULT 'email = """'
LOCATE FOR &gcTemp                            && Enter LOCATE expression
IF FOUND( )                                   && Este găsit?
  DISPLAY                                     && Dacă da, afişează înregistrarea
ELSE                                          && Dacă nu e găsit
  ? 'Conditia ' + gcTemp + ' nu a fost gasita'    && afişează un mesaj
ENDIF
USE
```

Instrucțiunea CASE are sintaxa:

DO CASE
 CASE <expresie_logică_1>
  <*comenzi şi instrucțiuni*>
 [CASE <expresie_logică_2>
<*comenzi şi instrucțiuni*>
  ...
 CASE <expresie_logică_N>
<*comenzi şi instrucțiuni*>]
 [OTHERWISE
<*comenzi şi instrucțiuni*>]
ENDCASE

Un exemplu de folosire al acesteia este următorul:

```
STORE CMONTH(DATE( )) TO luna                && luna curentă
DO CASE                                      && Începerea buclei case
  CASE INLIST(luna,'January','February','March')
    STORE 'Primul sfert al anului' TO rpt_titlu
  CASE INLIST(luna,'April','May','June')
    STORE 'Al doilea sfert al anului' TO rpt_titlu
  CASE INLIST(luna,'July','August','September')
    STORE 'Al treilea sfert al anului' TO rpt_titlu
  OTHERWISE
    STORE 'Al patrulea sfert al anului' TO rpt_titlu
ENDCASE                                      && Sfârşit buclă case
WAIT WINDOW rpt_titlu NOWAIT                 && afişare mesaj
```

Instrucțiuni de ciclare: SCAN, FOR, DO WHILE

```
OPEN DATABASE ('Universitati')               && Exemplu SCAN
USE Institutii in 1                          && Deschide tabela Institutii
USE Contacte in 2                            && Deschide tabela Contacte
CLEAR
SCAN FOR institutii.nr = 1 and contacte.nr = 1   && Permite şi EXIT şi LOOP (continuă)
  ? contacte.nume, institutii.nume, contacte.email
ENDSCAN
SET TALK OFF                                 && Exemplu FOR
CLOSE DATABASES
OPEN DATABASE ('Universitati')
USE Contacte
STORE 2 TO gnI                               && Valoare inițială
STORE 10 TO gnJ                              && Valoare finală
STORE 2 TO K                                 && Pas
FOR gnCount = gnI TO gnJ STEP K              && Permite şi EXIT şi LOOP (continuă)
  GOTO gnCount                               && Mută pointerul de înregistrare
  DISPLAY nume                               && Afişează numele
ENDFOR                                       && Sau NEXT
CLOSE DATABASES                        && Exemplu DO WHILE
OPEN DATABASE ('Universitati')
USE contacte
SET TALK OFF
```

```
DIMENSION A(4)
STORE 0 TO A
max = 0
DO WHILE .T.                          && Startul buclei DO WHILE
  IF EOF( )                           && Testează sfârşitul de fişier
    EXIT
  ENDIF
  IF poz < 10
    SKIP
    LOOP                              && Trece la testarea din nou a condiţiei
  ENDIF
  A(nr) = A(nr) + 1                   && Numără ...; cel mult institutii.nr = 9
  IF max < nr                         && Urmăreşte nr. maxim de instituţii
     max = nr
  ENDIF
  SKIP
ENDDO                                 && Sfârşit buclă
CLEAR
? 'Statistica contactelor inregistrate mai putin primele 10:'
FOR zz = 1 TO max
?? A(zz)
ENDFOR
```

21. Proceduri şi funcţii

În VFP procedurile şi funcţiile au sintaxa în forma:

PROCEDURE <nume_procedură>
<comenzi şi instrucţiuni>
ENDPROC

FUNCTION <nume_funcţie>
<comenzi şi instrucţiuni>
ENDFUNC

iar apelul acestora este în forma:

DO <nume_procedură>

DO <nume_funcţie>

Următoarea procedură afişează un mesaj iar următoarea funcţie arată o altă modalitate de a defini funcţii şi proceduri:

```
PROCEDURE myproc( cString )
  MESSAGEBOX ("myproc" + cString)
ENDPROC
```

```
FUNCTION plus2saptamani
PARAMETERS dDate
  RETURN dDate + 14
ENDFUNC
```

În cazul în care procedura are parametrii, cum este cazul:

```
PROCEDURE procedura( dData, cSir, nOriTipar )
  FOR nCnt = 1 to nOriTipar
    ? DTOC(dData) + " " + cSir + " " + STR(nCnt)
  ENDFOR
ENDPROC
```

atunci apelul se face în forma:

```
DO procedura WITH DATE(), "Salut Prieteni!", 10
```

22. Rapoarte şi etichete

5 forme de prezentare a rapoartelor sunt frecvente:

raport pe coloane raport pe linii raport 1 la n raport multi coloană etichetă

Un raport se salvează pe disc cu extensia **.frx* şi are de asemenea asociat şi un fişier cu extensia **.frt*. Într-un raport se specifică câmpurile dorite, textul de imprimat şi plasarea informației în pagină. Informația efectivă din raport se încarcă în momentul procesării raportului pentru imprimare şi este conformă cu modificările care survin în tabelele şi vederile folosite în întocmirea raportului.

Se poate crea un raport pe 3 căi: *Report Wizard* (pentru rapoarte dintr-o singură tabelă sau din mai multe tabele şi/sau vederi), *Quick Report* (dintr-o singură tabelă sau vedere) şi cu ajutorul lui *Report Designer* (rapoarte definite în întregime de utilizator).

Utilizarea lui Report Wizard

În *Project Manager* se selectează *Reports/New/Report Wizard*. Se selectează tipul de raport care se doreşte a se crea şi apoi se urmăresc instrucțiunile din ferestrele interogative.

Aplicație: realizarea unui raport cu persoanele contact din baza de date *Universitati*.

Rezolvare: se urmează paşii:

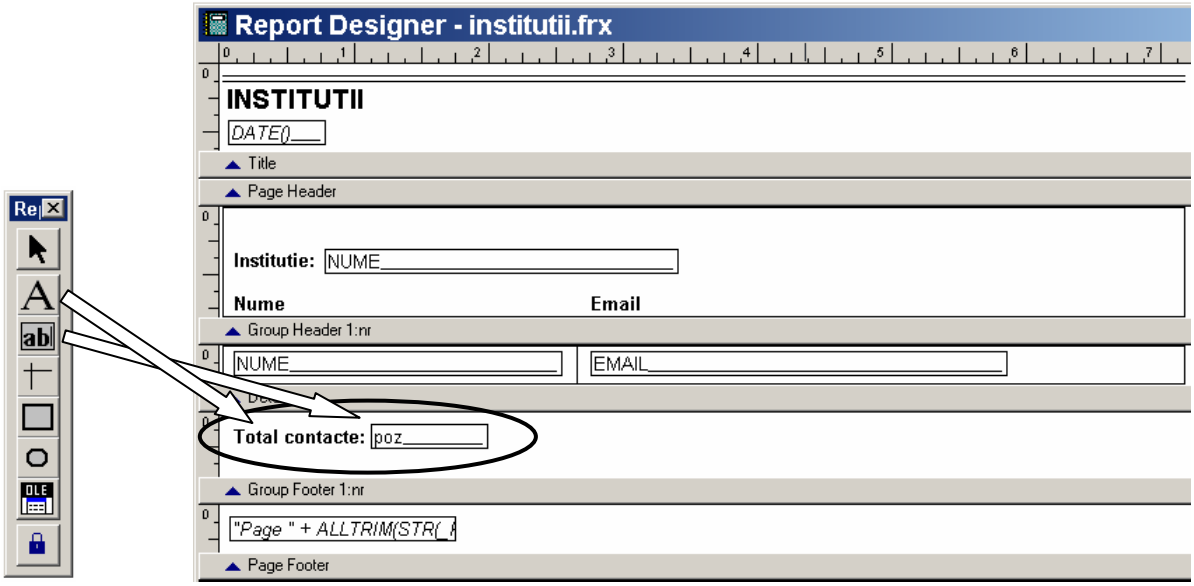
1. Se deschide proiectul *Universitati.pjx* (*File/Open...*);
2. Se încarcă baza de date *Universitati.dbc* (optiunea *Modify*);
3. Se selectează tabulatorul *Documents* şi de aici *Reports*;
4. Se acționează butonul *New...* şi apoi se selectează *Report Wizard*;
5. Se selectează *One-to-Many Report Wizard* şi se activează *Ok*;
6. Se selectează din tabela părinte (*Institutii*) câmpul *nume*;
7. Se selectează din tabela fiu (Contacte) câmpurile *nume* şi *email*;
8. Se acceptă relația *Institutii.nr = Contacte.nr* care leagă cele două tabele;
9. Se alege o regulă de ordonare în raport; fie aceasta după *acronim*;
10. Se alege o formă de prezentare; fie aceasta *Ledger*;
11. Se alege setarea de pagină; fie aceasta *Landscape*;
12. Se alege titlul raportului; fie acesta *institutii.frx*;
13. Se activează butonul *Preview* pentru a executa raportul.

Utilizarea lui Report Designer

Aplicație: totaluri pe categorii; pentru a realiza totaluri pe categorii este necesar să includem câmpuri numerice în raport, cum ar fi câmpul poz din tabela contacte.

Rezolvare: se modifică raportul *Institutii*:

1. Se deschide proiectul *Universitati.pjx* (*File/Open...*);
2. Se selectează tabulatorul *Documents* şi de aici *Reports*;
3. Se acționează butonul *New...* şi apoi se selectează *Report Wizard*;



4. Se modifică raportul cu ajutorul instrumentelor din *Report Controls* adăugând totalul după *poz* în *Group Footer*.

Realizarea unui raport dintr-o vedere şi Quick Report

1. Se deschide proiectul *Universitati.pjx* (*File/Open...*);
2. Se încarcă baza de date *Universitati.dbc* (optiunea *Modify*);
3. Se selectează tabulatorul *Documents* şi de aici *Reports*;
4. Se acționează butonul *New...* şi apoi se selectează *Report Wizard*;
5. Se selectează *Report Wizard* şi se activează *Ok*;
6. Se selectează vederea *contacte_institutii* cu structura: *contact.nume*, *contact.email*, *institutii.nume* (*contacte_institutii.nume_a*, *contacte_institutii.email*, *contacte_institutii.email*);
7. Se lasă negrupate contactele;
8. Se sortează după *contacte_institutii.nume_a* şi apoi după *contacte_institutii.nume_b*;
9. Se selectează boxa *Use display settings stored in the database*;
10. Se salvează raportul;
11. Se selectează raportul *Contacte_institutii* din tabulatorul *Docs* (*Documents*);

12. Se apasă butonul *Preview...*;

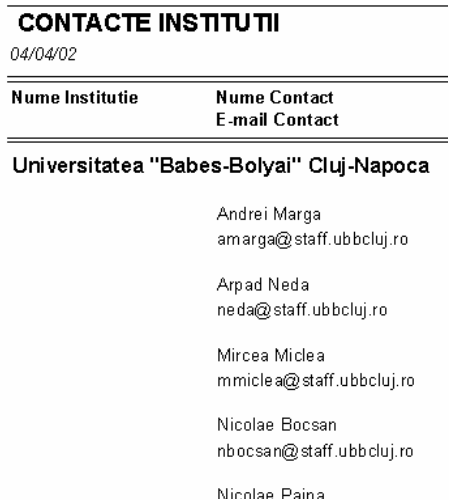
CONTACTE INSTITUTII

04/04/02

Nume Contact Nume Institutie	E-mail Contact
Andrei Achimas Universitatea de Medicina si Farmacie "Iuliu Hatieganu" Cluj-Napoca	
Andrei Marga Universitatea "Babes-Bolyai" Cluj-Napoca	amarga@staff.ubbcluj.ro
Arpad Neda Universitatea "Babes-Bolyai" Cluj-Napoca	neda@staff.ubbcluj.ro
Doru Pamfil Universitatea de Stiinte Agricole si Medicina Veterinara	dpamfil@usamvcluj.ro

Varianta 2

1. Se modifică vederea prin definirea ordinii de afişare *institutii.nume*, *contacte.nume*;
2. Se urmează paşii 1-6 ca la varianta 1;
3. Se grupează înregistrările după *nume_b*;
4. Se alege stilul *Executive*;
5. Se sortează după *nume_a*;
6. Se selectează boxa *Use display settings stored in the database*;
7. Se apasă *Finish* şi se salvează raportul cu numele *contacte_institutii1*;



CONTACTE INSTITUTII

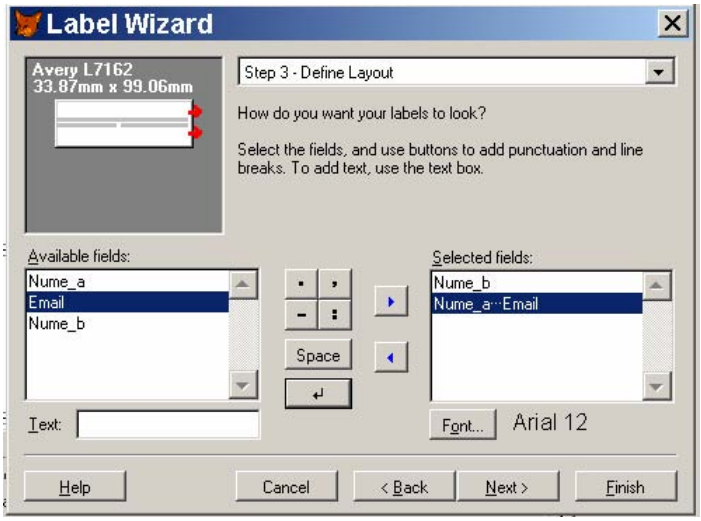
04/04/02

Nume Institutie	Nume Contact E-mail Contact
Universitatea "Babes-Bolyai" Cluj-Napoca	
	Andrei Marga amarga@staff.ubbcluj.ro
	Arpad Neda neda@staff.ubbcluj.ro
	Mircea Miclea mmiclea@staff.ubbcluj.ro
	Nicolae Bocsan nbocsan@staff.ubbcluj.ro
	Nicolae Paina

Etichete

Se urmează paşii:

1. Se selectează *Labels*, se apasă *New...*;
2. Se alege vederea *contacte_institutii*;
3. Se alege sistemul *metric*;
4. Se alege o formă de etichetă pe două coloane (de exemplu L7162);
5. Se defineşte forma etichetei (plasarea informațiilor în cadrul zonei etichetei);



6. Se defineşte fontul (de exemplu *Arial 12*);
7. Se salvează raportul;
8. Se vizualizează cu preview;

Universitatea "Babes-Bolyai" Cluj-Napoca
Andrei Marga amarga@staff.ubbcluj.ro

Universitatea "Babes-Bolyai" Cluj-Napoca
Arpad Neda neda@staff.ubbcluj.ro

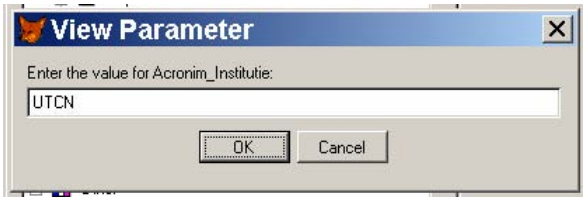
Universitatea "Babes-Bolyai" Cluj-Napoca
Mircea Miclea mmiclea@staff.ubbcluj.ro

Universitatea "Babes-Bolyai" Cluj-Napoca
Nicolae Bocsan nbocsan@staff.ubbcluj.ro

Crearea unei etichete dintr-o vedere cu parametru

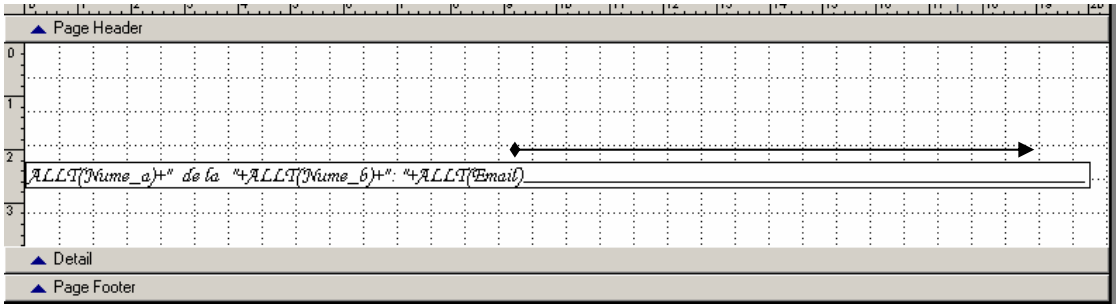
Se poate folosi vederea *caut_contact* din baza de date universitati. Se urmează paşii:

1. În *Label Wizard* se alege vederea *caut_contact*;
2. Se alege sistemul *metric* şi tipul *Avey EAL 04*;
3. Se defineşte forma (de exemplu: `nume_a de la nume_b : Email`);
4. Se alege fontul (de exemplu Monotype Corsiva, 12);
5. Se salvează eticheta caut_contact;
6. Se execută vederea de la butonul *Preview...* când se activează mesajul interogativ;



7. Pentru că spațiul alocat implicit de Wizard este insuficient pentru afişarea tuturor informațiilor (*email*) se alege modificarea raportului (*Modify*);
8. Se poate definii pagina *Landscape* (din *File/Page Setup/Print Setup/Orientation*);

9. Se măreşte caseta alocată textului din Label Designer;



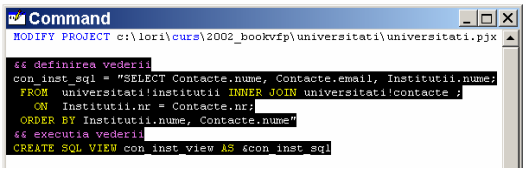
23. Macrosubstituție

Macrosubstituția este înlocuirea numelor cu variabile. În VFP se plasează operatorul & înaintea unei variabile pentru a folosi valoarea acestei variabile (care trebuie să fie un şir de caractere ce respectă sintaxa VFP) ca un nume. O comandă sau o funcție ce conține un nume se execută mai rapid ca una ce conține o macrosubstituție însă utilizarea macrosubstituțiilor conferă avantaje de flexibilitate în codul de executat.

Un exemplu de macrosubstituție poate fi crearea unei vederi folosind un cod SQL stocat într-o variabilă din care apoi poate fi chemat.

Codul se scrie într-un program (fişier de comenzi) sau se copiază în *Command*:

```
&& definirea vederii
con_inst_sql = "SELECT Contacte.nume, Contacte.email, Institutii.nume;
 FROM  universitati!institutii INNER JOIN universitati!contacte ;
  ON  Institutii.nr = Contacte.nr;
 ORDER BY Institutii.nume, Contacte.nume"
&& execuția vederii
CREATE SQL VIEW con_inst_view AS &con_inst_sql
```



După execuție, în *Project Manager* apare vederea.

Aceasta se poate apoi vizualiza (Browse).

24. Formulare

Aşa cum rapoartele permit facila tipărire a tabelelor, interogărilor şi vederilor, formularele sunt o cale eficientă pentru afişarea, introducerea şi editarea informațiilor din baza de date.

Se pot crea formulare interactive din tabele şi vederi, utilizând wizard-ul, constructorul şi aplicația expert Form Designer.

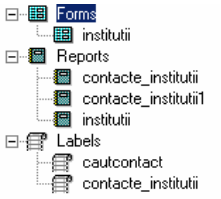
Form Wizard

Din *Project Manager* din tabulatorul *Documents* se selectează *Forms* apoi *New* şi *Form Wizard* după care se urmează instrucțiunile din ferestrele interogative următoare.

Aplicația 1. Să se creeze un formular pentru parcurgerea şi modificarea datelor pentru instituțiile existente şi pentru introducerea datelor pentru o nouă instituție în baza de date *universitati.dbc*.

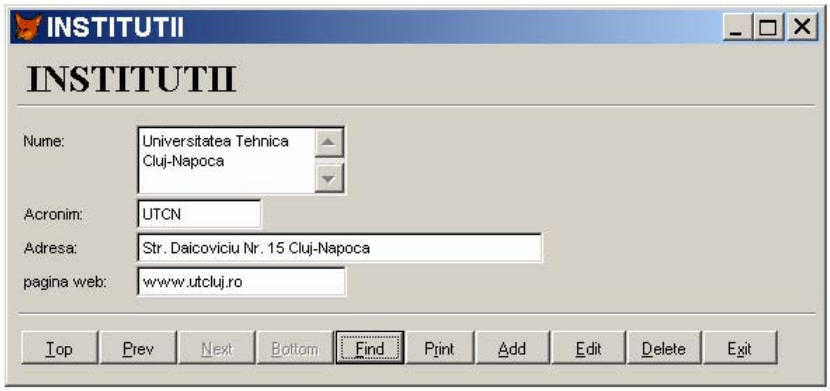
Rezolvare. Se urmează paşii:

1. Se alege *Form Wizard* în fereastra de dialog *Wizard Selection*;
2. Se alege tabela *Institutii.dbc*; se includ câmpurile acesteia în caseta *Selected Fields* mai puțin câmpul autoincrement *universitati!institutii.nr*;
3. Se alege stilul formei; de exemplu *Embossed*; se alege tipul butoanelor de navigare; de exemplu *Text buttons*; se alege indexul după care să se facă ordonarea la afişare; de exemplu *acronim*;
4. Se salvează forma pentru a fi utilizată ulterior; se alege numele acesteia *institutii*; ea se va salva pe disc cu numele *institutii.scx*;

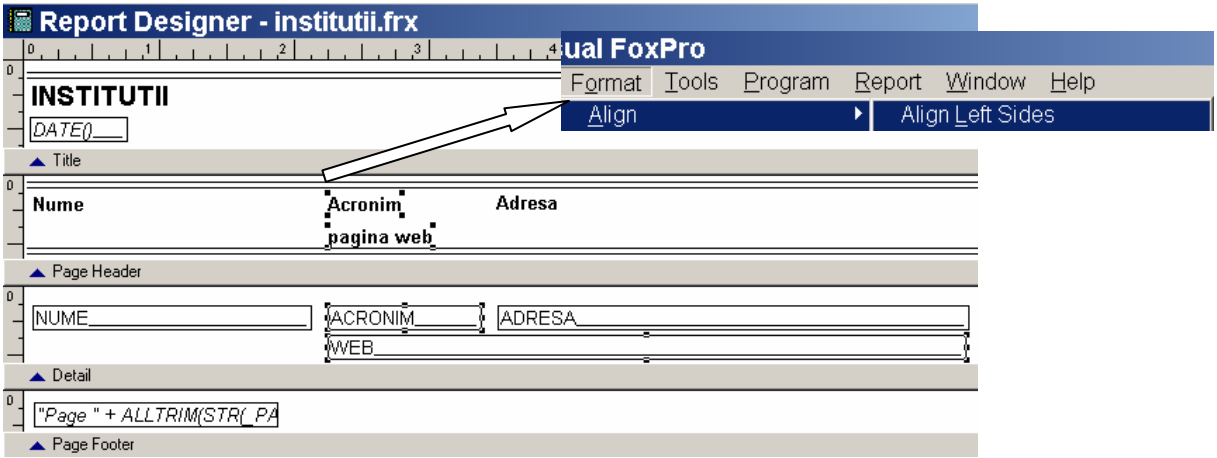


5. În *Project Manager* va apare în grupul *Forms* noua formă creată; se selectează forma *institutii* şi se lansează în execuție (butonul *Run*).

Noua formă creată conține 5 butoane pentru deplasare (*Top*, *Next*, *Prev*, *Bottom*, *Find*), un buton pentru imprimare (*Print*), 3 butoane pentru operații de intrare-ieşire (*Add* pentru adăugarea unei noi înregistrări, *Edit* pentru modificarea unei înregistrări existente şi *Delete* pentru ştergere) şi un buton pentru ieşire (*Exit*).

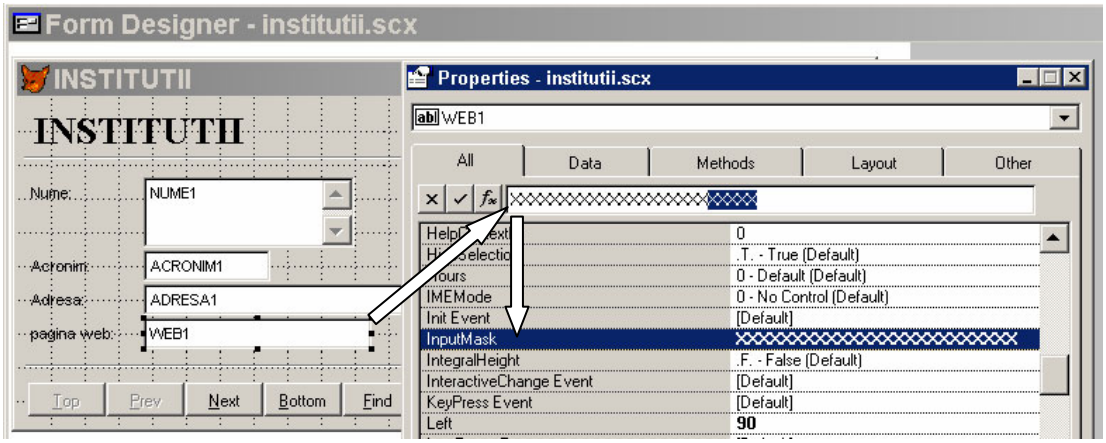


Pentru tipărirea informațiilor despre instituții se poate crea un raport *institutii.frx*; se poate folosi wizard-ul pentru generarea raportului incluzând câmpurile nume, acronim, adresa şi web, selecta opțiunea de preluare a etichetelor pentru câmpuri din tabela *institutii.dbf* şi apoi se poate modifica încât să se prezinte astfel:

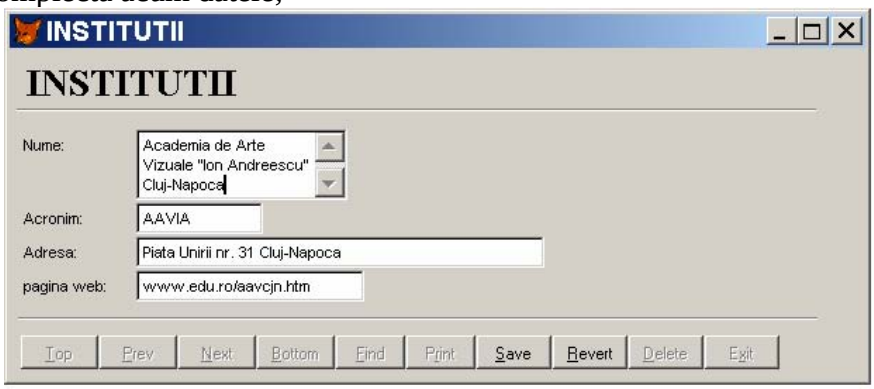


Se poate folosi formularul pentru introducerea unei noi instituții (de exemplu Academia de Arte Vizuale „Ion Andreescu” Cluj-Napoca). Tentativa de a completa adresa web a acesteia este sortită eşecului dacă câmpul pentru pagina web are 20 de caractere. Sunt necesare atunci următoarele operațiuni:

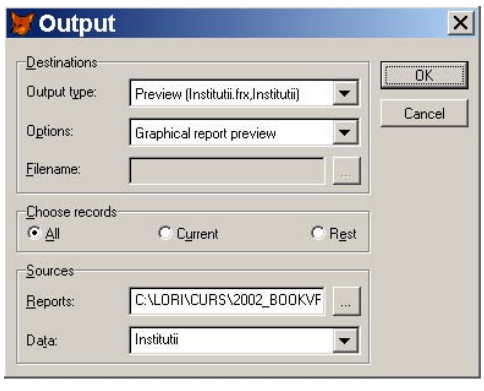
1. Se modifică structura tabelei *institutii.dbf* prin mărirea câmpului de la 20 la 25 de caractere (*Project Manager*/*Databases*/*Universitati*/*Tables*/*Institutii*/*Modify*);
2. Se modifică formularul *institutii.scx* (se măreşte caseta de editare pentru pagina web) pe calea *Project Manager*/*Docs*/*Forms*/*Institutii*/*Modify*;
3. Se activează proprietățile casetei de editare WEB1 (*click dreapta* pe caseta corespunzătoare paginii web şi apoi *Properties*);
4. Se modifică macheta de introducere a casetei prin inserarea a încă 5 "X";

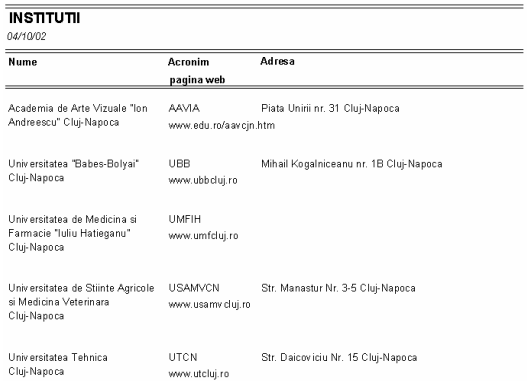


5. Pentru a executa formularul este necesar acum să închidem tabela *institutii.dbf* deschisă de sistem la modificarea structurii (execuția necesită accesul exclusiv la tabelă); se poate face acest lucru din *Data Session* sau din fereastra de comenzi *Close all* urmată de *Modi Project*);
6. Se execută formularul pe calea *Project Manager*/*Docs*/*Forms*/*Institutii*/*Run*;
7. Se pot complecta acum datele;



8. Pentru inserarea datelor în tabela *institutii.dbf* se apasă butonul *Save* şi apoi *Exit* când la o nouă execuție a formularului sau la o deschidere a tabelei *institutii* într-o fereastră *Browse* apare noua instituție înregistrată; dacă a fost definită funcția de autoincrementare pentru câmpul cheie *institutii.nr* acesta va avea valoarea generată automat în tabelă (vezi fereastra *Browse* pentru tabela *institutii*);
9. Tipărirea instituțiilor cu ajutorul formei *institutii.scx* se poate face acum prin selectarea raportului *institutii.frx*: *Project Manager/Docs/Forms/Institutii/Run/Print*;



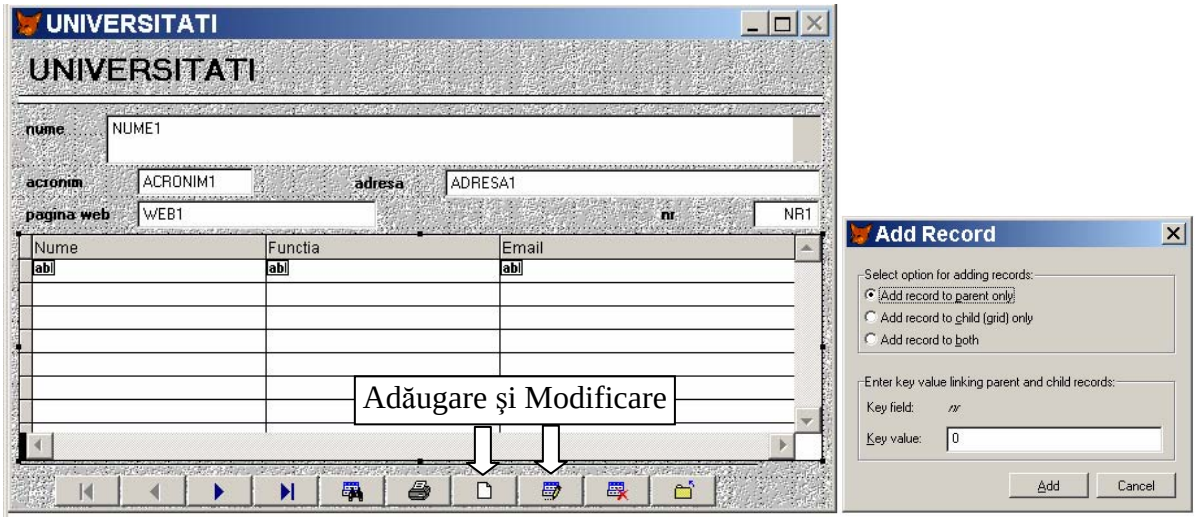


Aplicația 2. Să se creeze un formular pentru parcurgerea şi modificarea datelor simultan pentru instituțiile şi persoanele de contact din baza de date *universitati.dbc*.

Rezolvare. Se urmează paşii:

1. Se alege *One-to-many Form Wizard* în fereastra de dialog *Wizard Selection*;
2. Se alege tabela părinte: *Institutii.dbc*; se includ câmpurile acesteia în caseta *Selected Fields* mai puțin câmpul autoincrement *universitati!institutii.nr*;
3. Se alege tabela fiu: *Contacte.dbf*; se includ câmpurile acesteia în caseta *Selected Fields* mai puțin câmpurile cheie străină (*Contacte.poz*) şi autoincrement (*Contacte.nr*);
4. Se acceptă relația între tabele pe baza cheii primare *Institutii.nr* şi străine *Contacte.nr*;
5. Se alege stilul formei; de exemplu *Stone*; se alege tipul butoanelor de navigare; de exemplu *Picture buttons*; se alege indexul după care să se facă ordonarea la afişare pentru înregistrările din tabela părinte; de exemplu *nr*;
6. Se salvează forma pentru a fi utilizată ulterior; se alege numele acesteia *universitati*; ea se va salva pe disc cu numele *universitati.scx*;

În *Project Manager* va apare în grupul *Forms* noua formă creată; se selectează forma *universitati* şi se modifică ca mai jos (butonul *Modify*); apoi se lansează în execuție;



Formularul permite modificarea datelor pentru o instituție; prezența codului relației de integritate face ca modificările făcute valorii cheii primare (*Institutii.nr*) să se transmită şi în tabela de contacte, fapt care se poate observa cu ajutorul formularului. De asemenea, modificările efectuate asupra înregistrărilor din tabela de contacte (mai exact modificările asupra cheii străine) sunt controlate de codul relației de integritate. O modificare a cheii străine la o valoare care nu se regăseşte în tabela *institutii* este sortită eşecului.

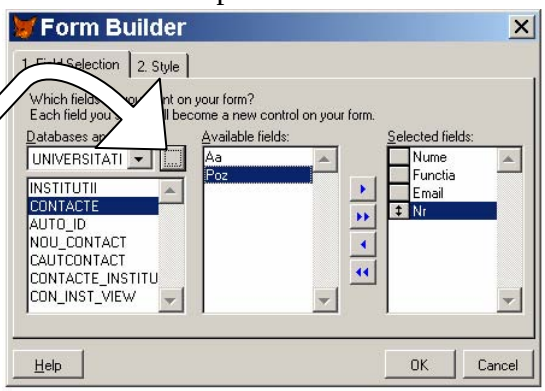
Formularul permite adăugarea unei instituții sau a unei persoane de contact. Prezența relației de integritate interzice însă adăugarea înregistrărilor simultan în ambele tabele. Este posibilă adăugarea simultană numai prin renunțarea la codul relației de integritate.

Form Builder

Constructorul de formulare (*Form Builder*) se poate activa pe următoarea succesiune de paşi:

1. Se încarcă proiectul;
2. Se selectează opțiunea *Forms*;
3. Se apasă *New/New form*;
4. Din bara de instrumente a lui *Form Desiner* se selectează *Form Builder* sau din meniul VFP, *Form/Quick Form*;

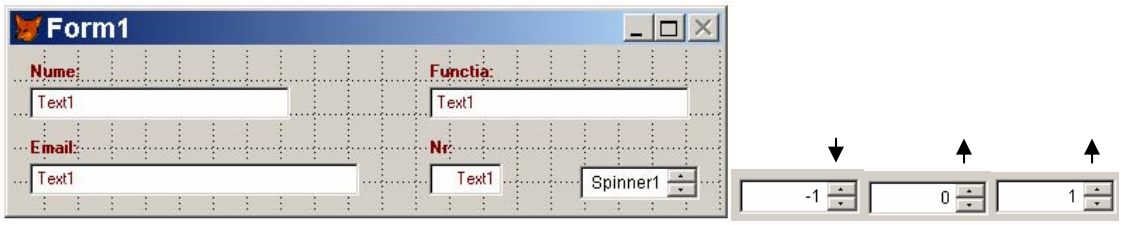
Aplicația 3. Să se construiască un formular pentru modificarea unei persoane de contact.



Rezolvare. Se urmează pașii:

1. Se încarcă *Form Builder*;
2. Se deschide *universitati.dbc* sau *contacte.dbf*;
3. Se selectează câmpurile din *contacte.dbf*;
4. Se alege stilul (de exemplu *Colorful*);
5. Se salvează formularul;
6. Se lansează în execuție;

Formularul va încărca din tabela *Contacte* prima înregistrare care va putea fi modificată după dorință. La închiderea formularului modificările efectuate se vor transmite în tabela Contacte (dacă nu a apărut conflict la modificarea cheii străine *contacte.nr*). Pentru a îmbunătăți formularul creat este necesar să adăugăm controale pe acesta cu ajutorul barei de instrumente *Form Controls*. Din aceasta se poate adăuga un control *Spinner* (Spiner1).



La lansarea în execuție a formularului se poate observa că săgețile controlului Spinner1 (↑ şi ↓) incrementează sau decrementează valoarea din caseta de editare a acestuia fără însă ca odată cu aceasta să se deplaseze pointerul înregistrării curente în tabela *Contacte* şi să accesăm o altă înregistrare.

Este necesară setarea proprietăților controlului *Spinner1*. O posibilitate este ca mai jos:

- ControlSource **Contacte.poz**
- DownClick Event **(User Procedure)**
- SpinerHighValue = **RECCOUNT()**
- SpinnerLowValue = **1**
- UpClick Event **(User Procedure)**

```
v = val(Form2.Spinner1.Text)
if v >= 1 and v <= recsize()
  goto v
  Form2.NUME1.Text1.refresh
  Form2.FUNCTIA1.Text1.refresh
  Form2.EMAIL1.Text1.refresh
  Form2.NR1.Text1.refresh
else
  return to master
endif
```

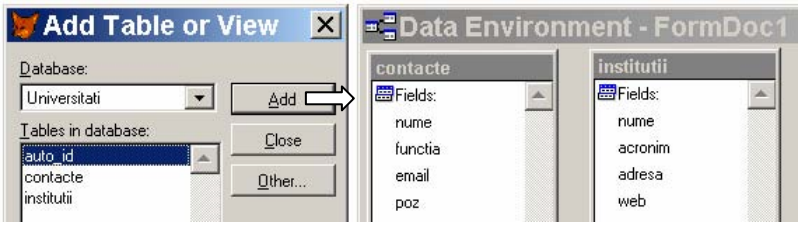
Form Designer

Pentru generarea manuală a formularelor sunt necesare bara de instrumente a lui *Form Desiner* şi a lui *Form Controls*.

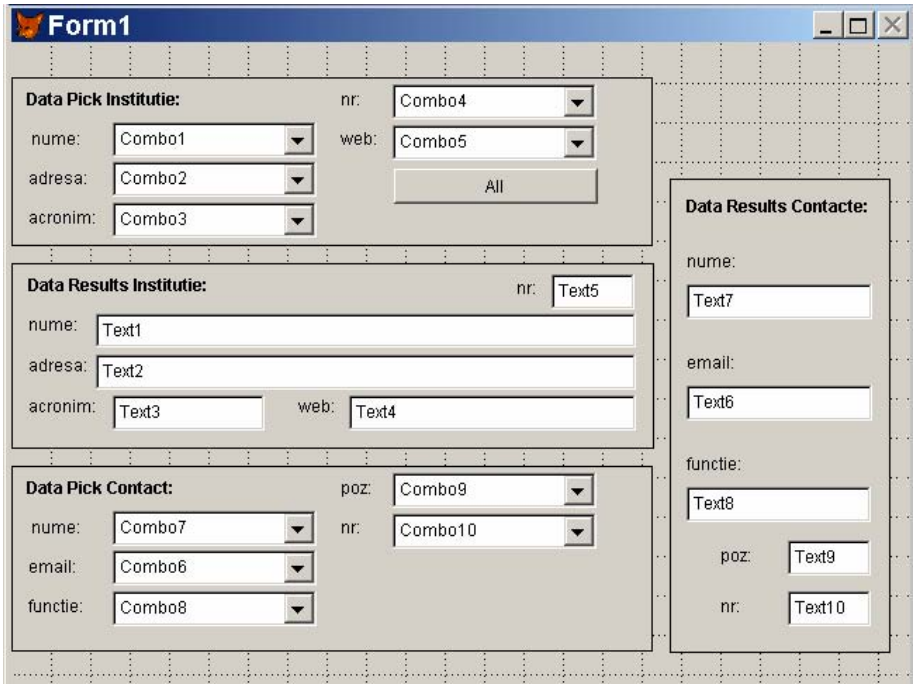
Aplicația 4. Să se construiască un formular după modelul:

Rezolvare. Se urmează paşii:

1. Cu *Form Designer* se inserează tabelele *institutii* şi *contacte* (butonul *Data Environment*);



2. Cu *Form Controls* se contruiesc butoanele, casetele de editare, listele combinate şi etichete; se defineşte mărimea formularului, se fac alinierile;



3. Se creează câte un index după fiecare câmp din tabelele *contacte* şi *institutii* (regular sau primar);
4. Se definesc proprietățile pentru fiecare *Combo* şi *Text*;
    a. Combo1
        i. Click Event

```
select institutii
set order to nume
seek ALLTRIM(Form1.Combo1.Text)
Form1.Text1.Refresh
Form1.Text2.Refresh
Form1.Text3.Refresh
Form1.Text4.Refresh
```

Form1.Text5.Refresh
select contacte
set filter to contacte.nr = institutii.nr
Form1.Combo6.Refresh
Form1.Combo7.Refresh
Form1.Combo8.Refresh
Form1.Combo9.Refresh
Form1.Combo10.Refresh

ii. RowSource: *Institutii.nume*

iii. Row Source Type: *6 – Fields*

b. Combo2

i. Click Event

select institutii
set order to adresa
seek ALLTRIM(Form1.Combo2.Text)
Form1.Text1.Refresh
Form1.Text2.Refresh
Form1.Text3.Refresh
Form1.Text4.Refresh
Form1.Text5.Refresh
select contacte
set filter to contacte.nr = institutii.nr
Form1.Combo6.Refresh
Form1.Combo7.Refresh
Form1.Combo8.Refresh
Form1.Combo9.Refresh
Form1.Combo10.Refresh

ii. RowSource: *Institutii.adresa*

iii. Row Source Type: *6 – Fields*

c. Analog *Combo3* şi *Combo5*

d. Combo4

select institutii
set order to nr
seek val(ALLTRIM(Form1.Combo4.Text))
Form1.Text1.Refresh
Form1.Text2.Refresh
Form1.Text3.Refresh
Form1.Text4.Refresh
Form1.Text5.Refresh
select contacte
set filter to contacte.nr = institutii.nr
Form1.Combo6.Refresh
Form1.Combo7.Refresh
Form1.Combo8.Refresh
Form1.Combo9.Refresh
Form1.Combo10.Refresh

e. Command1 (All)

i. Click Event

```
select contacte
set filter to
Form1.Combo6.Refresh
Form1.Combo7.Refresh
Form1.Combo8.Refresh
Form1.Combo9.Refresh
Form1.Combo10.Refresh
```

ii. Caption: *All*

f. Text1: ControlSource *Institutii.nume*;

g. Text2: ControlSource *Institutii.adresa*;

h. *Text3*, *Text4*, *Text5* analog;

i. Text7: ControlSource *Contacte.nume*;

j. Text6: ControlSource *Contacte.email*;

k. *Text8*, *Text9*, *Text10* analog;

l. Combo7

i. Click Event

```
select contacte
set order to nume
seek ALLTRIM(Form1.Combo7.Text)
Form1.Text6.Refresh
Form1.Text7.Refresh
Form1.Text8.Refresh
Form1.Text9.Refresh
Form1.Text10.Refresh
```

ii. RowSource: *Contacte.nume*

iii. Row Source Type: *6 – Fields*

m. *Combo6*, *Combo8* analog;

n. Combo9

i. Click Event

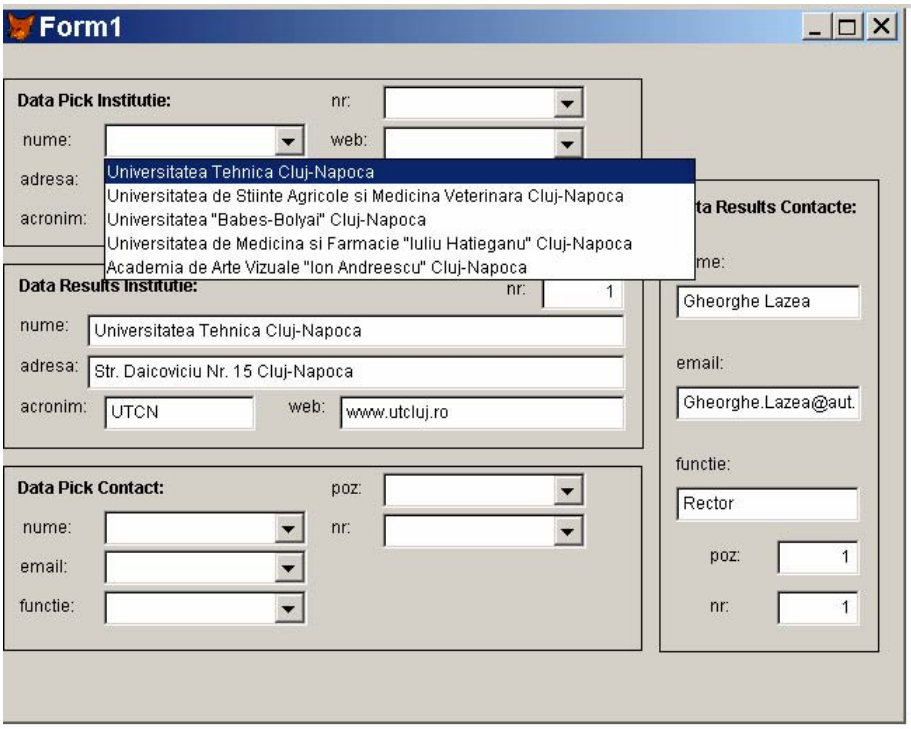
```
select contacte
set order to poz
seek val(ALLTRIM(Form1.Combo9.Text))
Form1.Text6.Refresh
Form1.Text7.Refresh
Form1.Text8.Refresh
Form1.Text9.Refresh
Form1.Text10.Refresh
```

ii. RowSource: *Contacte.poz*

iii. Row Source Type: *6 – Fields*

o. *Combo10* analog

5. Se salvează forma şi se lansează în execuție când se obține o fereastră de tipul:



6. Se pot adăuga butoane pentru adăugare instituție şi adăugare persoană de contact.
7. Pentru adăugare instituție: un buton Add care să aplice un *Append Blank* şi un buton Save care să aplice un Replace nume with AllTrim(Form1.Text1.Text) şi aşa mai departe;
8. Pentru adăugare contact: un buton care să aplice un *Append Blank* urmat de Form1.Text10.Text = Form1.Text5.Text şi un buton Save care să aplice un Replace nume with AllTrim(Form1.Text7.Text) şi aşa mai departe;
9. Noua formă creată este în conformitate cu relația de integritate; pentru protejarea cheilor la adăugare se poate seta proprietatea Enabled la .F.;
10. Valorile câmpurilor autoincrement de asemenea pot fi blocate la modificare cu ajutorul aceleiaşi proprietăți;
11. Cu ajutorul controlului *Image* şi apoi a proprietății *Picture* se pot insera poze în format recunoscut de sistem (bmp, gif, jpg, etc.);

Următoarele controale sunt disponibile în mod implicit la crearea de formulare:

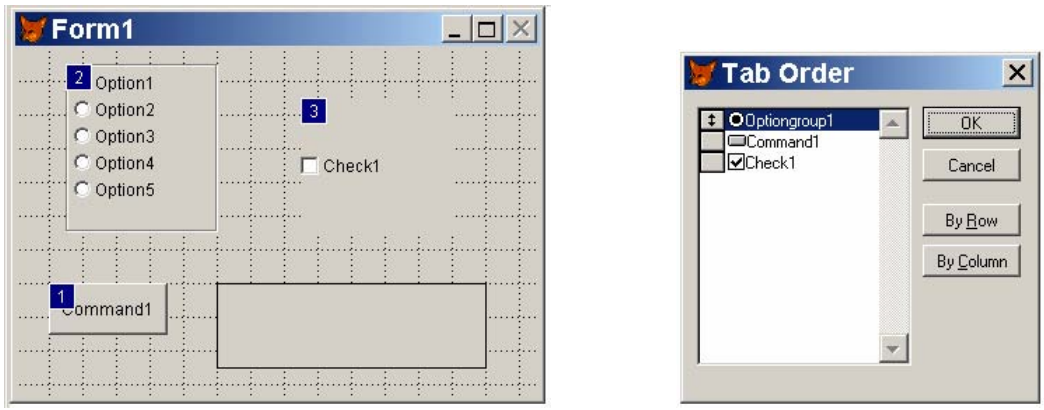
- *Label*: creează un control de tip etichetă;
- *Text Box*: creează un control casetă de editare cu introducerea de text pe o singură linie;
- *Edit Box*: creează un control casetă de editare cu introducerea de text pe mai multe linii;
- *Command Button*: creează un control de tip buton de comandă;
- *Command Group*: creează un control de tip grup de butoane de comandă;
- *Option Group*: crează un control de tip grup de butoane radio;

- *Check Box*: crează un control de tip casetă de selecție (opțiuni);
- *Combo Box*: control de tip listă ascunsă;
- *List Box*: control de tip listă vizibilă cu bare de defilare;
- *Spinner*: control de tip butoane incrementare/decrementare combinate cu o casetă de editare;
- *Grid Control*: un control de tip grid (vezi formularele one-to-many generate cu wizard-ul);
- *Image*: crează o imagine grafică;
- *Timer*: crează un control de timp;
- *Page Frame*: crează un grup de tabulatori;
- *Line* şi *Shape*: permit trasarea de curbe şi suprafețe pe formular;

Pentru alinierea şi ordonarea controalelor pe formular este utilă bara de instrumente *Layout* (*View/Toolbars...*):

Form Designer conține şi controale pentru definirea culorilor şi formatării formularelor. De asemenea, vizualizarea codului asociat unei proprietăți a formularului se poate face tot cu *Form Designer*.

Pentru a modifica ordinea de selecție a controalelor pe formular se selectează din meniu *View/Tab Order* când va apărea numărul de ordine al fiecărui control pe formular (numai pentru controalele care pot primi *focus*):

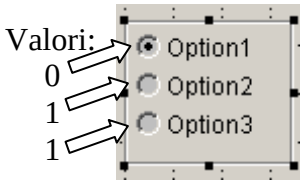


Schimbarea ordinii se face cu click pe control. Renumerotarea începând cu poziția 1 se face cu dublu click pe controlul care se dorește a fi primul; apoi cu click pe al doilea, şi aşa mai departe. O altă posibilitate este de a seta ordinea pe baza unei liste. Din *Tools/Options/Forms/Tab ordering* se setează *By List* şi apoi la *View/Tab Order* se activează o fereastră în care sunt aşezate controalele într-o listă în ordinea de selecție (Tab Order).

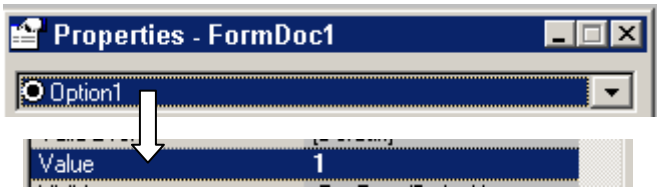
Setarea ariei maxime pe ecran (implicit 640×480) a unui formular se face din *Tools/Options/Forms/Maximum design area*.

25. Controale

Controalele sunt mediul de bază pentru interacțiunea cu utilizatorul. Acestea se aplică pe formulare şi pot avea asociate:



- valori (controlul buton de opțiune are valoarea implicită 0 dacă este selectat şi valoarea implicită 1 dacă nu este selectat); dacă numele formularului este *Form4* (proprietatea *Name*) iar numele grupului de butoane de opțiune este *OptionGroup1* atunci accesul la valoare se face pe calea *Form4.OptionGroup1.Option1.Value*;



- câmpuri (proprietatea *ControlSource*);
- variabile ....

Aplicația1

Să se realizeze o aplicație cu butoane de opțiune (radio) care să afişeze un mesaj la selectarea unei opțiuni din caseta de opțiuni.

Rezolvare. Se pot urma paşii:

1. Din *Project Manager* se selectează *Documents* (*Docs*) apoi *Forms*, *New...*, *New Form*;
2. Se stabileşte proprietatea *Caption* pentru formular (din *Form Designer*, *Properties Window*); fie *Caption* **Test**; acesta va fi afişat la execuția formularului;
3. Se stabileşte proprietatea *Name* pentru formular; acesta va fi folosit pentru a identifica formularul în proceduri; fie *Name* **Form_Test**;
4. Se salvează formularul pe disc (*File/Save*); se stabileşte un nume pentru formular; acesta va fi folosit pentru identificarea formularului în cadrul proiectului şi va fi numele sub care acesta se salvează pe disc; fie acesta *Form3.scx*;
5. Se adaugă acum din *Form Controls* un control de tipul *Option Group*;
6. Se selectează controlul *Option Group* creat; dacă a fost închisă fereastra de proprietăți se activează din nou pe aceeaşi cale; acum avem 4 obiecte: formularul (cu proprietățile sale) şi *OptionGroup1* (cu proprietățile sale) şi două obiecte de tip butoane de opțiune (*Option1* şi *Option2*); din caseta de proprietăți a lui *OptionGroup1* se selectează *Click Event* şi se accesează această proprietate (*dublu click*);
7. În fereastra *Form_test.Click* care se activează şi care conține codul procedurii care se execută la apăsarea unui buton de opțiune (vezi: *Object: Form_Test*; *Procedure: Click*) se introduce următorul cod:

```
store "Starea butoanelor de optiune:" to xx
For i = 1 to ThisForm.OptionGroup1.ButtonCount
  xx = xx + chr(13) + ThisForm.OptionGroup1.buttons[i].Caption +;
  ":" + str(ThisForm.OptionGroup1.Buttons[i].Value)
endfor
Messagebox(xx)
```

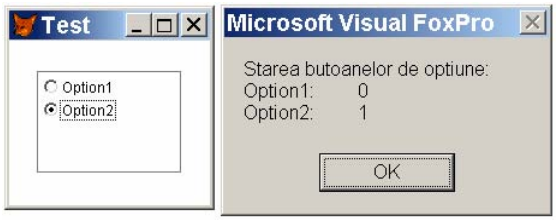
care pentru fiecare buton din grupul de butoane *OptionGroup1* extrage titlul acestuia (*Caption*) din tabloul (şirul) de butoane ale acestuia (*buttons*) şi apoi valoarea (*Value*) pe care o converteşte la şir de caractere (funcția *str(·)*) şi le memorează în şirul de caractere *xx*; funcția *chr(13)* este folosită pentru trecerea la linie nouă; la sfârşit este afişat un mesaj cu conținutul şirului *xx* iar descriptorul *ThisForm* este folosit pentru a specifica formularul curent;

8. Se poate seta proprietățile BackColor (Red,Green,Blue) ale obiectelor (forma, grup de butoane, butoane); se pot seta acestea la alb (255,255,255), roşu (255,0,0), etc.; pentru accesul la butoane se poate merge pe calea ilustrată mai jos:

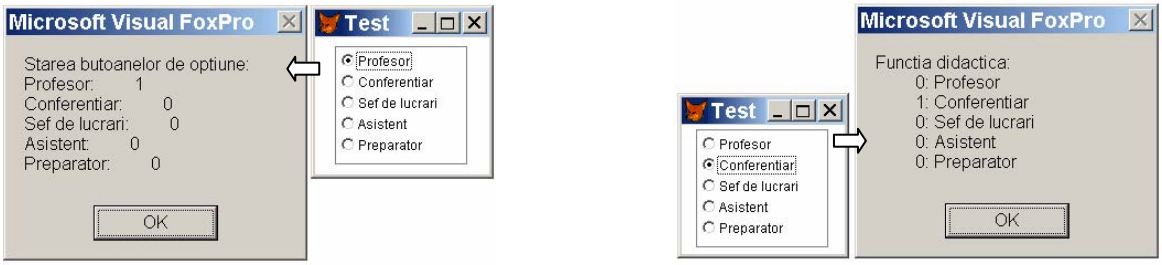
9. Pentru centrarea grupului de butoane pe formular se poate scrie următorul cod la proprietatea Activate Event (evenimentul de activare a formularului):

```
ThisForm.OptionGroup1.Left = (ThisForm.Width – ThisForm.OptionGroup1.Width)/2
ThisForm.OptionGroup1.Top = (ThisForm.Height – ThisForm.OptionGroup1.Height)/2
```

10. Pentru ca la redimensionarea formularului la execuție să se centreze automat grupul de butoane în fereastra formularului, este necesară introducerea aceluiaşi cod şi la evenimentul *Resize* al formularului;
11. Execuția formularului (butonul *!*) va duce la afişarea unui mesaj în forma:



12. Se pot acum modifica titlurile opțiunilor şi chiar numărul acestora fără ca codul să sufere modificări; se poate modifica şi mesajul implicit din fereastra de mesaj ca în figura:



unde alinierea în fereastra de mesaj se poate face dacă se inversează ordinea de afişare:

```
store "Functia didactica:" to xx
For i = 1 to THISFORM.OptionGroup1.ButtonCount
  xx = xx + chr(13) + str(THISFORM.OptionGroup1.Buttons[i].Value) +;
  ": " + THISFORM.OptionGroup1.buttons[i].Caption
endfor
Messagebox(xx)
```

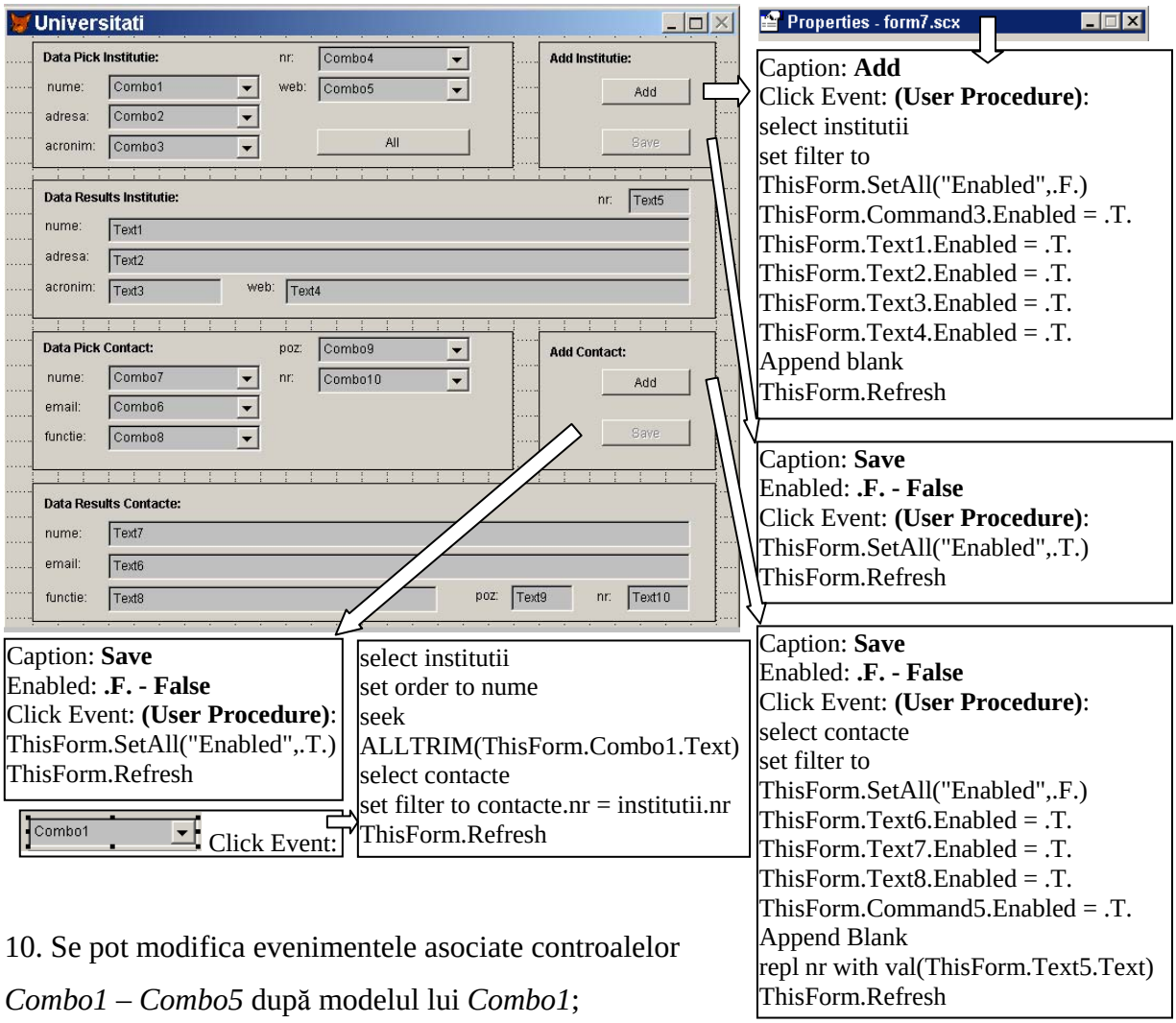
Aplicația2

Să se modifice formularul *Form1* (Aplicația 4, pag. 54) astfel încât să permită:

12. Pentru adăugare instituție: un buton Add care să aplice un *Append Blank* şi un buton Save care să aplice un Replace nume with AllTrim(ThisForm.Text1.Text) şi aşa mai departe;
13. Pentru adăugare contact: un buton care să aplice un *Append Blank* urmat de Form1.Text10.Text = Form1.Text5.Text şi un buton Save care să aplice un Replace nume with AllTrim(ThisForm.Text7.Text) şi aşa mai departe;
14. Noua formă creată este în conformitate cu relația de integritate; pentru protejarea cheilor la adăugare se poate seta proprietatea Enabled la .F.;
15. Valorile câmpurilor autoincrement de asemenea pot fi blocate la modificare cu ajutorul aceleiaşi proprietăți;

Rezolvare. Se urmează paşii:

1. Se modifică setarea ariei maxime a formularelor la dimensiunea de 800×600 (*Tools/Options/Forms/Maximum design area*);
2. Se creează un nou formular în cadrul proiectului şi se salvează cu un nume; fie acesta *form7.scx*;
3. Se selectează toate obiectele (controalele) de pe forma *form1.scx* cu ajutorul mouse-ului;
4. Se copiază în clipboard (*Edit/Copy*); se copiază pe forma *form7.scx* (*Edit/Paste*);
5. Se salvează din nou *form7*; se verifică existența tuturor procedurilor asociate evenimentelor de pe *form7*;
6. Se pot rearanja controalele pe formular;
7. Se includ tabelele *institutii* şi *contacte* în mediul de lucru al formei *form7* (*Form Designer*/*Data Environment*/*Add Table or View*/*Add institutii*, *contacte*);
8. Pentru a permite adăugarea unei persoane de contact în prezența relației de integritate se asociază o valoare implicită pentru câmpul *nr* din tabela *contacte* (*contacte/Modify*/*Fields/nr/Field validation/Default value: 1*);
9. Se adaugă câte două controale de tip *command button* pentru fiecare tabelă pe formular;



10. Se pot modifica evenimentele asociate controalelor *Combo1 – Combo5* după modelul lui *Combo1*;
11. Se salvează şi se execută formularul, care permite acum şi adăugarea unei instituții şi/sau persoane de contact;

26. Controale şi containere în FVP

Elementele puse la dispoziție de mediul VFP pentru dezvoltarea de aplicații se clasifică în controale şi containere care pot fi vizuale şi non-vizuale. Un container poate conține alte containere şi controale. Sunt astfel două tipuri de clase: clasele *container* şi clasele *control*. Orice container sau control care este folosit în aplicație reprezintă o instanțiere a clasei din care face parte şi se numeşte *obiect* (de tip *container* sau *control*).

Dezvoltarea unei aplicații presupune alegerea elementelor potrivite pentru realizarea obiectivelor dorite. Cea mai rapidă cale de a dezvolta aplicații este prin folosirea elementelor standard puse la dispoziție de mediul VFP. Fiecare *obiect* (*container* sau *control*) are asociate *proprietăți* şi *evenimente* accesabile atât în faza de construcție prin intermediul ferestrei de proprietăți asociate obiectului (*Properties Window*) cât şi în faza de execuție (prin intermediul operatorului "."). Următoarea schemă clasifică şi ierarhizează obiectele VFP:

- Obiecte VFP:
  - Controale:
    - Vizuale (pot fi vizibile la execuție):
      - Check Box ()
      - Combo Box ()
      - Command Button ()
      - Control (poate conține orice control)
      - Edit Box ()
      - Header (creează o casetă *Header* pentru o coloană într-un control de tip *grid*)
      - Hyperlink () creează o legătură către un *Document Activ* prin intermediul unui Container de Documente Active (ex: Microsoft Internet Explorer), legătură dată printr-un URL (*uniform resource locator*)
      - Image () creează o legătură către o imagine care este afişată în format BMP
      - Label () creează o etichetă care poate afişa un text
      - Line () creează un control care poate afişa o linie
      - List Box () poate afişa o listă de valori
      - OLE Bound Control () poate afişa un obiect de tip *general* într-un câmp dintr-un tabel (de exemplu provenit din Word sau Excel)

- OLE Container Control ()poate afişa un obiect de tip *general* în aplicație (de exemplu provenit din Word sau Excel)
- OptionButton (adaugă un buton de opțiune într-un container de tipul *Option Button Group*)
- Shape () crează un control care poate afişa o formă geometrică
- Spinner () creează un control care poate afişa o castă de editare de tip text pentru o valoare şi două butoane pentru parcurgerea valorilor unui domeniu de valori asociat casetei de tip text
- Text Box () creează un control care poate afişa o casetă de editare de tip text

▪ Non-vizuale:

- Active Doc (Crează un *document activ* care poate fi găzduit de un container de documente active ca *Microsoft Internet Explorer*)
- Custom (poate conține orice control, Pageframe, container, custom)
- Project Hook (poate conține fişiere şi servere)
- Timer () creează un control de timp

o Containere:

▪ Vizuale:

- Container (poate conține orice control, )
- Form (poate conține Pageframe, orice control, containere, Custom)
- Grid (poate conține *coloane grid*; *coloanele grid* fac legături la orice tip de obiecte exceptând formularele, seturile de formulare, barele de instrumente, controalele de timp şi alte *coloane de grid*); cu ajutorul *grid*-urilor () se pot afişa datele în linii şi coloane şi este similar cu apariția lor într-o fereastră Browse
- Column (creează o *coloană* într-un *grid*)
- Page (o *pagină* poate conține orice controale, containere, Custom)
- Toolbar (poate conține orice control, pagini, containere)
- Option Button Group (poate conține butoane de opțiune, )
- Command Button Group (poate conține butoane de comandă, )

▪ Non-vizuale:

- Form Set (poate conține formulare şi bare de instrumente)
- PageFrame (poate conține *pagini*)

Toate obiectele VFP au asociate următoarele evenimente (prin proceduri):

- Init: este executată atunci când obiectul este creat;
- Destroy: este executată atunci când obiectul este dealocat din memorie;
- Error: este executată atunci când o eroare apare la execuția procedurilor din obiect;

Pentru accesarea obiectelor VFP din aplicație în timpul execuției sunt utile următoarele proprietăți:

- Parent: containerul care conține obiectul care referă procedura; de exemplu pentru formularul *Test.scx* proprietatea *Parent* apelată în procedura Option1.Click va returna o referință către containerul acestui obiect (OptionGroup1);
- This: obiectul curent;
- ThisForm: formularul curent;
- ThisFormSet: FormSetul curent;

Exemplu:

1. Se selectează obiectul Option1 din formularul Test;
2. În caseta procedurii *Option1*.*Click* (*Click Event*) se introduce MessageBox("Container: "+This.Parent.Name + " Formularul: " + ThisForm.Caption);
3. Se execută formularul;

Un control poate fi legat de datele din tabele sau din variabilele din memorie pe baza proprietății *ControlSource*. Schematizarea efectelor valorilor atribuite proprietății *ControlSource* asupra controalelor:

- casetă de validare:
    - ControlSource = câmp de tabelă:
        - valorile NULL, valorile logice (.T. şi .F.) şi valorile numerice 0, 1 şi 2 determină: selectarea, deselectarea sau dezactivarea casetei de validare pe măsură ce indicatorul de înregistrări parcurge tabela;
        - exemplu: să se construiască o tabelă ce conține un câmp cu valori numerice de 0, 1 şi 2 şi să se construiască un formular care conține o casetă de validare legată cu *ControlSource* de acest câmp şi să se execute formularul;
- coloană:
    - ControlSource = câmp de tabelă:
        - utilizatorul editează direct valorile câmpului odată cu editarea valorilor din coloană; proprietatea se extinde la întreaga grilă cu ajutorul proprietății RecordSource a grilei;
        - exemplu: proprietatea *RecordSource* de la formularul *Universitati*, obiectul grid1;
- casetă combo sau list:

- ControlSource = variabilă:
  - valoarea aleasă de utilizator este păstrată în variabilă;
  - exemplu: se adaugă un control de tip *List* (*List1*) la formularul *Test*; la evenimentul de activare a formularului se adaugă instrucțiunile:

```
dimension a(10)
public i
store 2 to i
store " " to a
ThisForm.List1.additem(str(1))
```

la evenimentul DblClick al listei *List1* se introduc instrucțiunile:

```
ThisForm.List1.additem(str(i))
i = i + 1
```

se execută formularul; la dublu click se va insera câte un element în listă;

- buton de opțiune:
  - ControlSource = câmp numeric:
    - în câmp va fi inserată valoarea 0 (dacă este selectat butonul) sau 1 (dacă nu este selectat butonul);
  - ControlSource = câmp logic:
    - în câmp va fi inserată valoarea .T. (dacă este selectat butonul) sau .F. (dacă nu este selectat butonul);
    - dacă indicatorul de înregistrări parcurge tabela, valoarea butonului de opțiune se modifică pentru a reflecta noua valoare a câmpului;
    - exemplu: să se insereze un control de tip buton de opțiune pentru înregistrările şterse dintr-o tabelă;
- grup de opțiuni:
  - ControlSource = câmp de tip caracter:
    - se păstrează în câmp titlul butonului selectat;
    - opțiunea nu este valabilă şi pentru butoane individuale de opțiuni;
- spinner:
  - ControlSource = câmp sau variabilă numerică:
    - caseta de incrementare afişează şi scrie valori numerice din/în câmpul sau variabila asociate;
    - exemplu: formularul din aplicația 3 (pag. 53);
- text sau edit:
  - ControlSource = câmp:

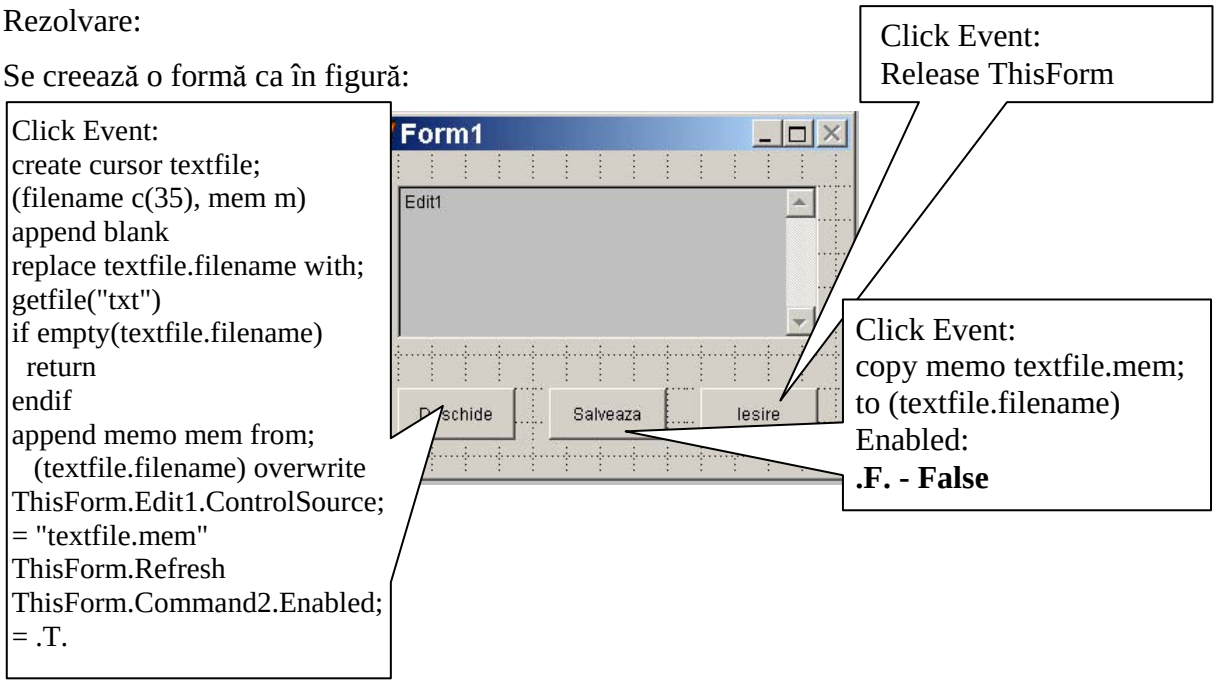
- în casetă este afișată valoarea câmpului din tabelă;
- modificările efectuate sunt inserate pe loc în tabelă;

Aplicația3

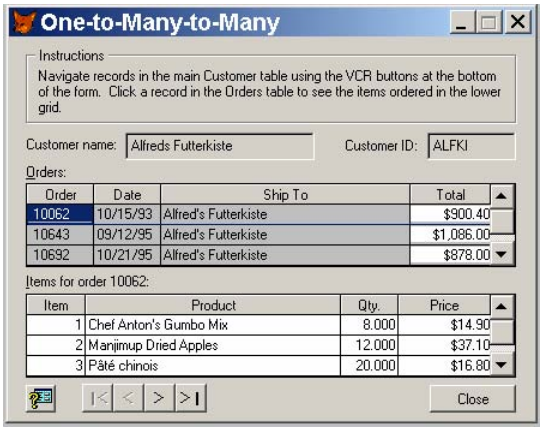
Să se folosească controlul Casetă de Editare pentru a edita un fişier text.

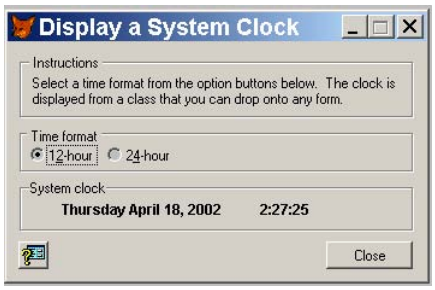
Rezolvare:

Se creează o formă ca în figură:

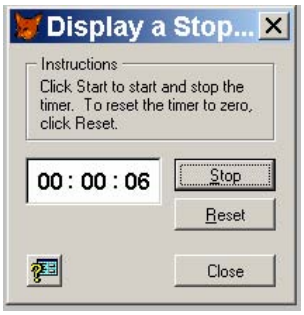


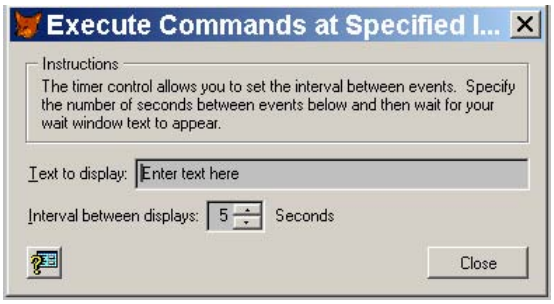
Cu ajutorul grid-urilor se pot crea formulare care să opereze cu informații din mai multe tabele relatate cu relații de tipul 1 la n, ca în exemplul:





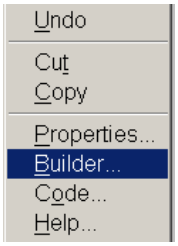
Cu ajutorul controlului de timp se pot construi formulare ca în exemplele:





27. Constructoarele de controale şi containere

Mediul VFP pune la dispoziția utilizatorului un set de constructoare pentru setarea proprietăților controalelor şi containerelor. Astfel există: *Combo Box Builder* (pentru liste ascunse), *Command Group Builder* (pentru grupuri de butoane de comandă), *Edit Box Builder* (pentru casete de tip Edit), *List Box Builder* (pentru liste), *Option Group Builder* (pentru casete de opțiune), *Text Box Builder* (pentru casete de tip Text), *AutoFormat Builder* (pentru grupuri de controale).

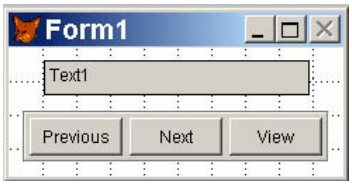


Pentru a accesa un constructor:

1. Se plasează controlul pe formular din bara de instrumente *Form Controls*;
2. Se selectează controlul; se apasă click dreapta;
3. Se selectează opțiunea *Builder…*;
4. Se aleg opțiunile dorite din aplicația expert corespunzătoare controlului sau containerului selectat;

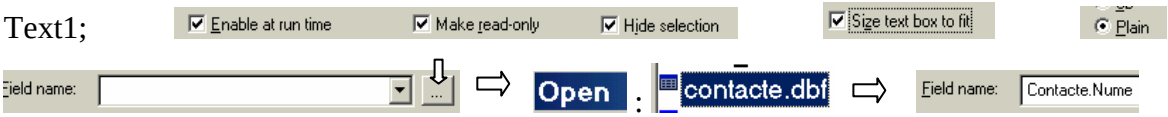
Aplicația 1.

Formular cu 3 butoane de comandă care să deschidă şi să parcurgă tabela *contacte*. La apăsarea unuia dintre butoane să se activeze o fereastră de mesaj cu conținutul înregistrării curente. Să se adauge apoi o casetă de tip Text care să conțină numele persoanei de contact selectate fără a permite modificarea.

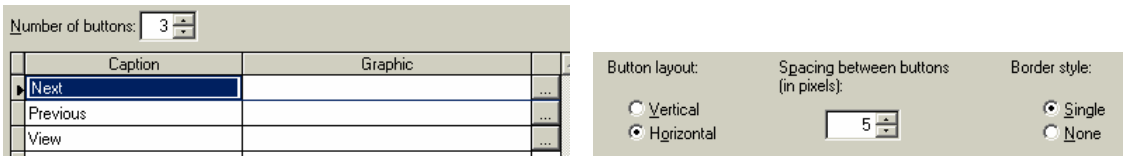


Rezolvare. Se pot urma paşii:

1. Se generează un nou formular (*Forms/New…/New Form*);
2. Se adaugă un control *Text* şi un container *CommandGroup*;
3. Se foloseşte aplicația expert *Text Box Builder* pentru a seta proprietăți pentru controlul Text1;



4. Odată selectată tabela contacte, aceasta va fi automat inclusă în mediul de lucru cu date al formularului; se poate vedea acest fapt pe calea *View/Data Environment* ...; de asemenea, în caseta *Field name:* se poate acum selecta orice câmp din tabelă; se lasă *contacte.nume*;
5. Se foloseşte aplicația expert *Command Group Builder* pentru a seta proprietăți pentru containerul *CommandGroup1*;



6. Se poate lansa în execuție formularul, când se observă că butoanele din container nu au asociate evenimente: apăsarea butoanelor nu produce acțiuni;
7. Se adaugă evenimente:
   a. pentru formular: *Activate Event*:

```
if not eof()
  thisform.commandgroup1.enabled = .T.
endif
```

   b. pentru commandgroup1 doar proprietatea *commandgroup1.enabled = .F.*
   c. pentru command1, procedura Command1.Click:

```
if recno() = 1
  This.Enabled = .F.
else
  skip -1
  ThisForm.Text1.Refresh
  ThisForm.CommandGroup1.Command2.Enabled = .T.
endif
```

   d. pentru command2, procedura Command2.Click:

```
if recno() = reccount()
  This.Enabled = .F.
else
  skip 1
  ThisForm.Text1.Refresh
  ThisForm.CommandGroup1.Command1.Enabled = .T.
endif
```

   e. pentru command3, procedura Command3.Click:

```
mesaj = alltrim(nume) + chr(13) + alltrim(functia) +;
chr(13) +alltrim(email) + chr(13) + "nr=" + alltrim(str(nr)) +;
chr(13) + "poz=" + alltrim(str(poz))
MessageBox(mesaj,64)
```

8. Se execută formularul; acționarea butoanelor de comandă produce acum evenimentele dorite.

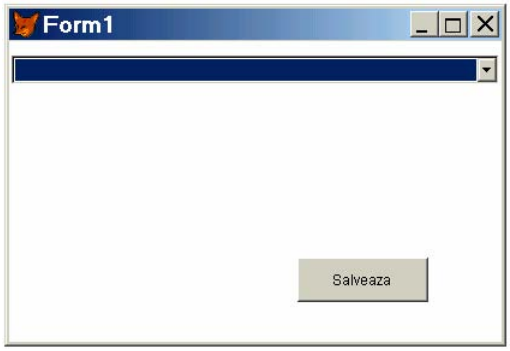
Aplicația 2.

Formular care să permită adăugarea unei persoane de contact după o machetă predefinită.

Rezolvare. Se creează un nou formular. Se urmează pașii:

1. se adaugă pe formular un control de listă ascunsă, un container de tipul butoane de opțiune, 4 casete de tip text și un buton de comandă;
2. se folosește aplicația expert *Combo Box Builder* pentru a seta proprietăți pentru Combo1; din baza de date *universitati* și tabela *contacte* se alege câmpul *nume* pentru a i se lega valorile cu lista ascunsă; la stilul listei se alege *Drop-down list*;

3. se foloseşte aplicația expert *Option Group Builder* pentru a genera containerul OptionGroup1; se introduce numărul de butoane (6) şi denumirile acestora (vezi figura); se alege poziționarea *verticală* a butoanelor, şi *spațiere* de 5 puncte între acestea;
4. se foloseşte aplicația expert Text Box Builder pentru a seta proprietăți pentru controalele Text2-Text4; acestea se setează astfel: aliniament: Text2: *right*; Text3: *left*, Text4: *center*; de asemenea, pentru Text4: 1. Format: ☑ *Make read only*; Input Mask: @;
5. se setează manual celelalte proprietăți şi evenimentele pentru controale după cum urmează:



6. procedura Combo1.Click:

```
thisform.optiongroup1.Visible = .T.
thisform.text1.Visible = .T.
```

7. proprietatea OptionGroup1.Visible: *.F. – False*;
8. procedura Text1.GotFocus:

```
thisform.text2.visible = .T.
```

9. proprietatea Text1.MaxLength: *=fsize('nume','contacte')*

10. proprietatea Text1.Visible: *.F. – False*;
11. proprietatea Text2.InteractiveChange:

```
if "@" $ this.text
  thisform.text4.Visible = .T.
  thisform.text3.Visible = .T.
  v_email = substr(this.text, 1,at("@",this.text)-1)
  this.refresh
  thisform.text3.MaxLength = fsize('email','contacte');
  - len(alltrim(v_email))
  thisform.text3.setfocus
endif
if len(alltrim(this.text)) = this.maxlength - 2
  MessageBox("Lungimea maxima pentru acest camp a fost atinsa."+;
  chr(13)+"Pentru a putea adauga noi caractere mariti dimensiunea campului in tabela")
endif
```

12. proprietatea Text2.MaxLength: *=fsize('email','contacte')*;
13. proprietatea Text2.ControlSource: *v_email*

14. proprietatea Text2.Visible: *.F. – False*;
15. proprietatea Text4.Visible: *.F. – False*;
16. procedura Text3.GotFocus:

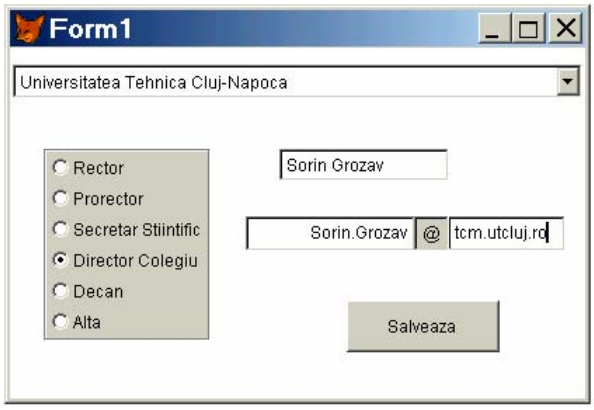
```
thisform.command1.visible = .T.
```

17. procedura Text3.InteractiveChange:

```
if len(alltrim(this.text)) = this.maxlength - 1
  MessageBox("Lungimea maxima pentru acest camp a fost atinsa."+;
  chr(13)+"Pentru a putea adauga noi caractere mariti dimensiunea campului in tabela")
endif
```

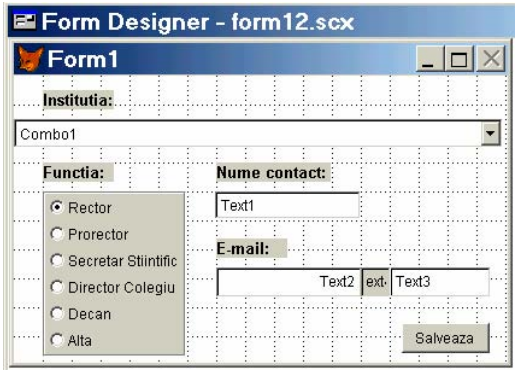
18. proprietatea Text3.Visible: *.F. – False*;

19. proprietatea Command1.Caption: *Salveaza*;



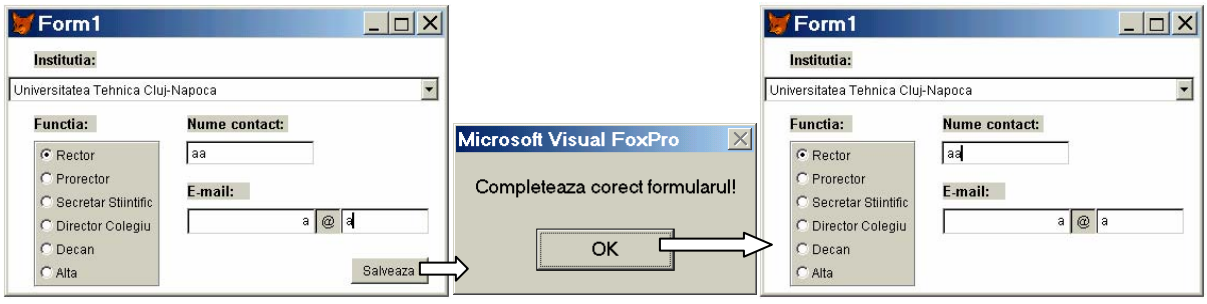
20. procedura Command1.Click:

```
if (len(alltrim(thisform.text1.text))>3)and;
  (len(alltrim(thisform.text2.text))>0)and;
  (len(alltrim(thisform.text3.text))>0)
    select contacte
    append blank
    replace nr with institutii.nr,;
    nume with thisform.text1.text,;
    email with alltrim(thisform.text2.text)+;
    thisform.text4.text+alltrim(thisform.text3.text)
    for i = 1 to thisform.optiongroup1.buttoncount
      if thisform.optiongroup1.buttons[i].value = 1
        replace functia with thisform.optiongroup1.buttons[i].Caption
      endif
    endfor
else
  this.visible = .F.
  MessageBox("Completeaza corect formularul!")
  if len(alltrim(thisform.text3.text))=0
    thisform.text3.visible = .F.
    thisform.text2.setfocus
  endif
  if (len(alltrim(thisform.text2.text))=0)or;
     (len(alltrim(thisform.text1.text))<4)
    thisform.text1.setfocus
  endif
endif
```



21. Se modifică formularul adăugându-se etichetele corespunzătoare;

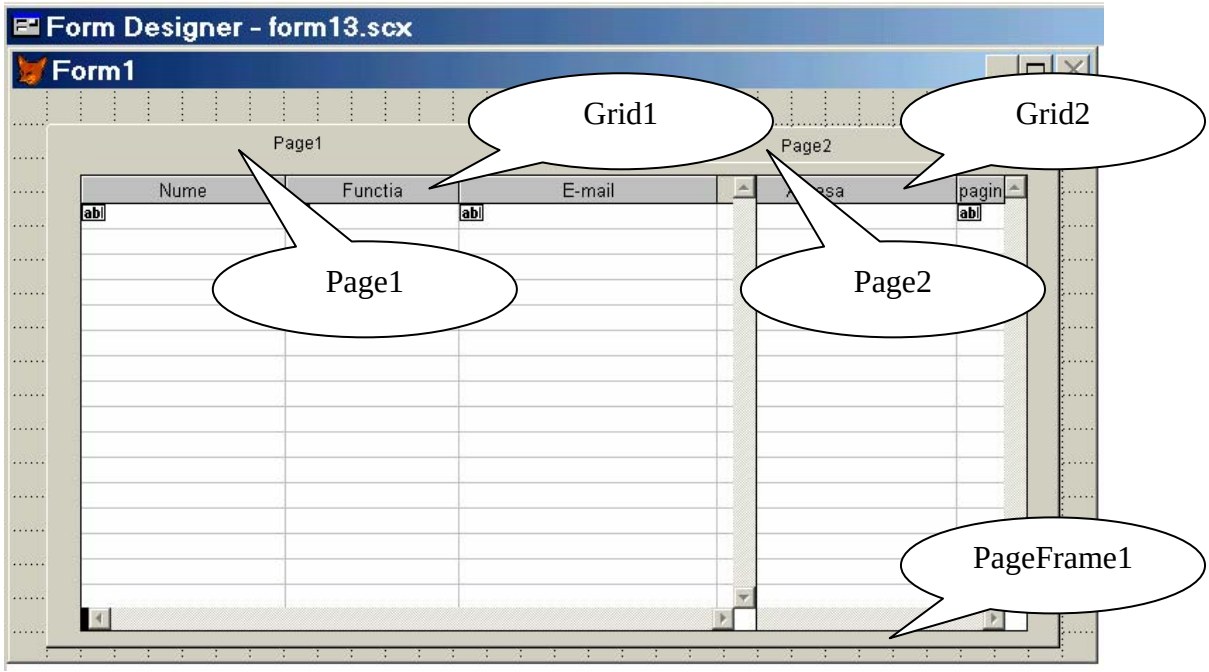
22. La execuția formularului, dacă se introduc date incorecte (vezi procedura *Command1.Click*) atunci va apare o fereastră de mesaj şi utilizatorul este întors la completarea casetei cu date incorecte:



Aplicația 3.

Să se folosească containerele *PageFrame* şi *Grid* pentru vizualizarea datelor din baza de date *universitati*.

Rezolvare: se poate construi un formular după modelul:



Se urmează paşii:

1. Se creează un nou formular (Form1);
2. Se adaugă un container *PageFrame* (PageFrame1);
3. Se adaugă un container *Grid* (Grid1);
4. Se adaugă al doilea container *Grid* (Grid2);
5. Se selectează din fereastra de proprietăți containerul Grid1 (pentru instituții);
6. Se lansează aplicația expert *Grid Builder* pentru definirea proprietăților containerului Grid1; în tabulatorul *Grid Items* se selectează tabela *institutii* şi din caseta *Available fields* se includ în caseta *Selected fields* câmpurile: *Nume*, *Acronim*, *Adresa*, *Web*;
7. Se selectează din fereastra de proprietăți containerul Grid2 (pentru contacte);
8. Se lansează aplicația expert *Grid Builder* pentru definirea proprietăților containerului Grid1; în tabulatorul *Grid Items* se selectează tabela *contacte* şi din caseta *Available fields* se includ în caseta *Selected fields* câmpurile: *Nume*, *Functia*, *Email*;
9. În tabulatorul *Relationship* în caseta *Key field in parent table:* se selectează *Institutii.nr* iar în caseta *Related index in child table:* se selectează câmpul nr;
10. Se setează manual proprietățile şi evenimentele controalelor şi containerelor:

    10.1. evenimentul Form1.Activate:

ThisForm.Grid1.Visible = .T.;

    10.2. evenimentul Page1.Click:

ThisForm.Grid2.Visible = .F.
ThisForm.Grid1.Visible = .T.

    10.3. evenimentul Page2.Click:

ThisForm.Grid1.Visible = .F.
ThisForm.Grid2.Visible = .T.

10.4. proprietatea Grid1.Visible = .F.

10.5. proprietatea Grid2.Visible = .F.

10.6. proprietatea Grid1.ReadOnly = .T.

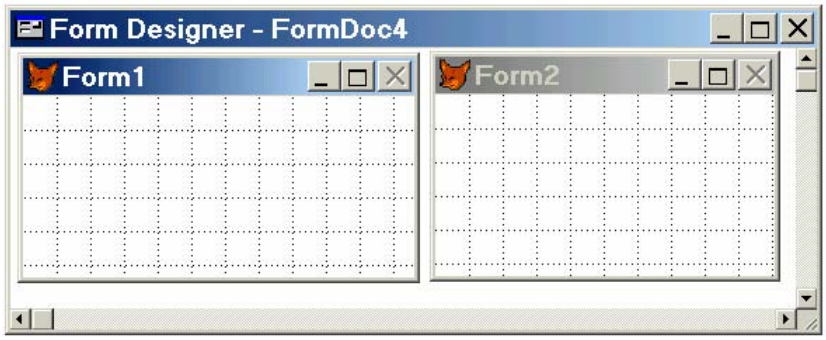
10.7. proprietatea Grid2.ReadOnly = .T.

Aplicația 4. Seturi de formulare

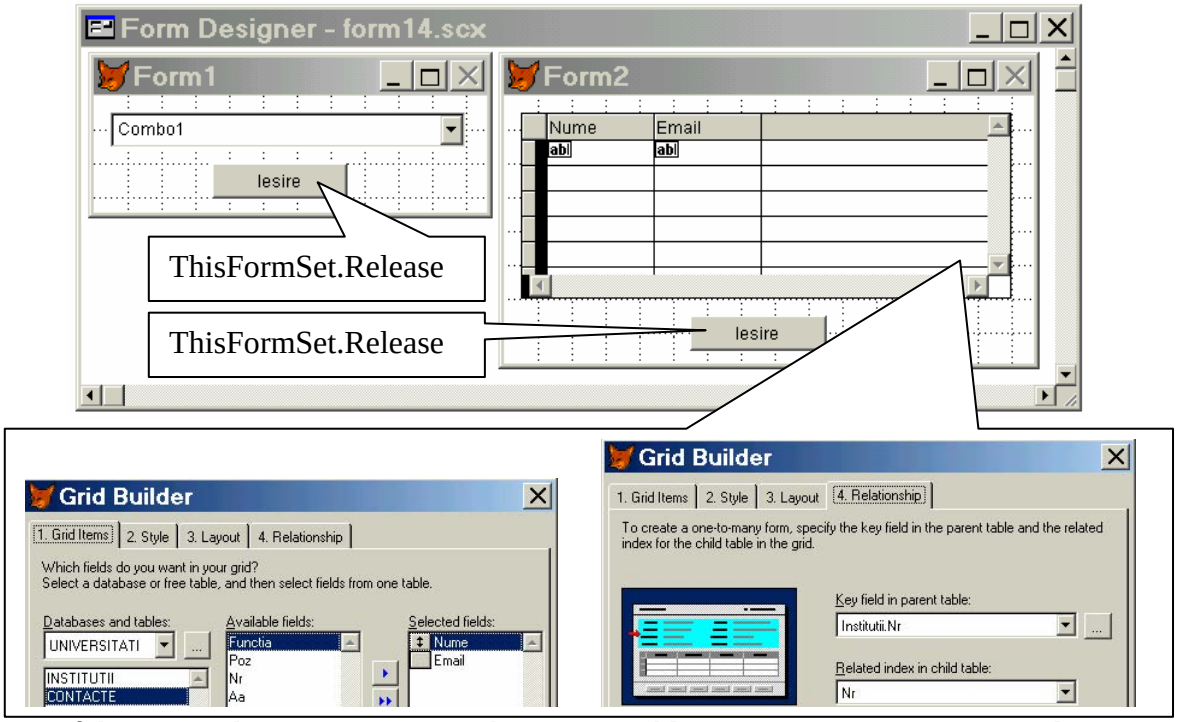
Să se realizeze o aplicație care să folosească două formulare care să conțină informațiile din baza de date *universitati*.

Rezolvare. Se urmează pașii:

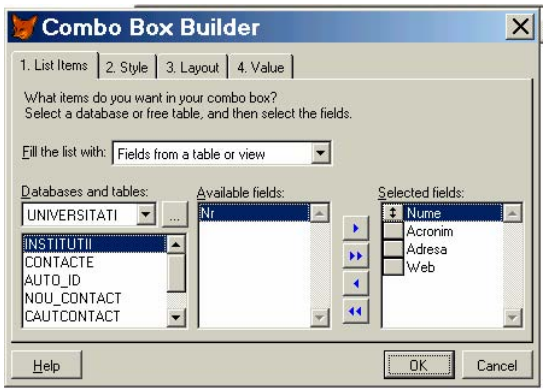
1. Din *Project Manager* se selectează *Forms/New.../New Form*;
2. Din meniu, în categoria *Form* se selectează opțiunea *Create Form Set*;
3. Pe aceeași cale, se adaugă un nou formular: *Form/Add New Form*;

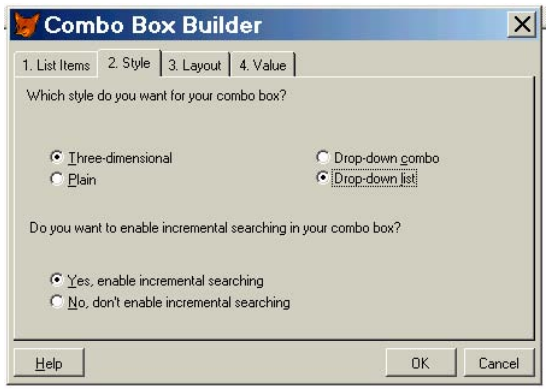


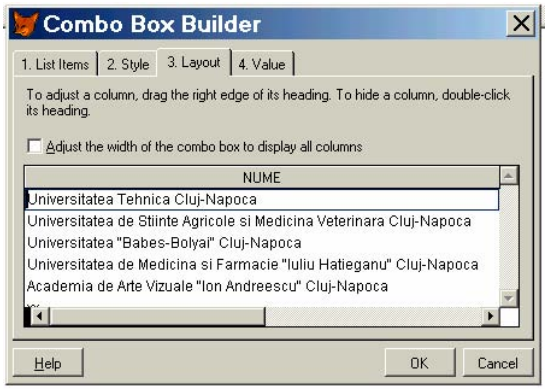
4. Se adaugă tabelele *institutii* și *contacte* în *Data Environment* (*View/Data Environment...*);
5. Se adaugă contoale și containere ca în figură:

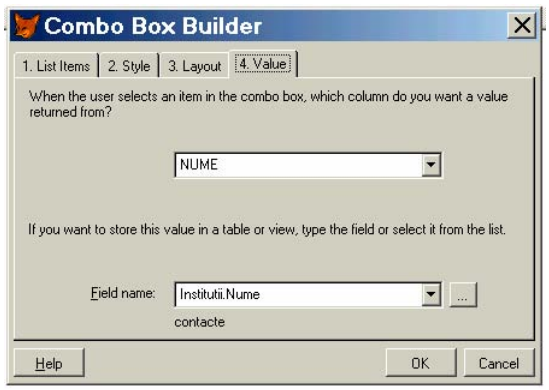


6. Se folosește aplicația expert *Combo Box Builder* pentru a seta proprietățile casetei Combo1:







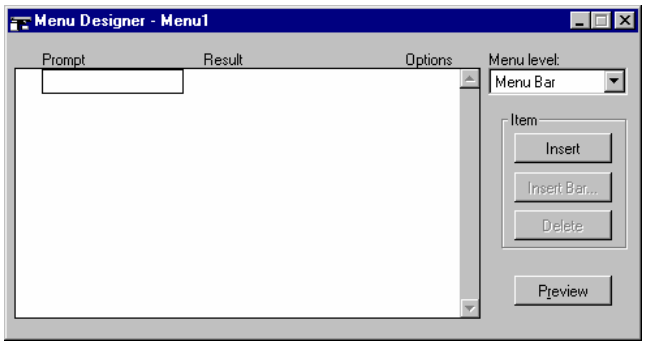


7. Se execută setul de formulare.

28. Meniuri

În VFP, folosirea meniurilor în aplicații face ca rezultatul obținut (produsul program) să îşi mărească calitatea în exploatare.

Fiecare parte a unei aplicații VFP poate avea propriul său meniu sistem sau un set de meniuri. Pentru a crea un meniu, se foloseşte aplicația expert *Menu Designer*.

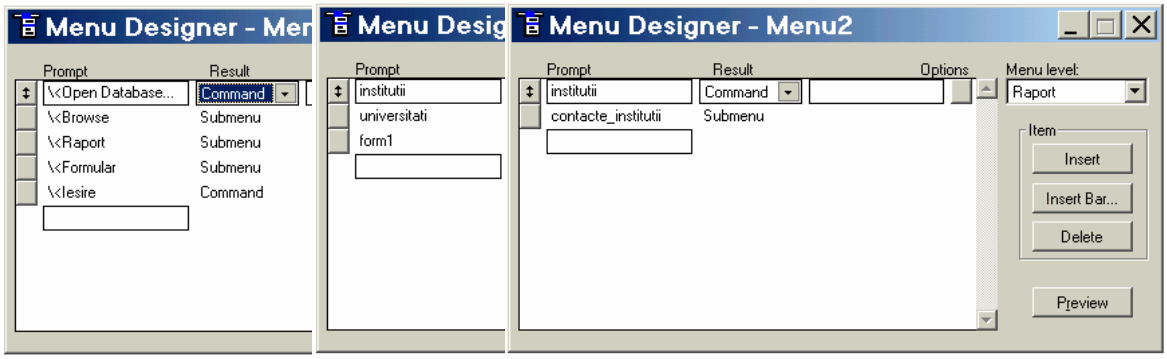


Crearea unui meniu sistem implică câțiva paşi. În funcție de mărimea aplicației şi complexitatea meniurilor care se intenționează a se folosi, aceştia sunt:

1. Planificarea şi construcția sistemului (decide ce meniuri ai nevoie, dacă acestea vor apărea în interfață, care dintre acestea necesită submeniuri, şi aşa mai departe);
2. Crearea meniurilor şi submeniurilor (definirea titlurilor pentru meniu, pentru elementele din meniu, utilizând *Menu Designer*);
3. Asocierea de evenimente astfel încât sistemul să facă ce dorim (specificarea acțiunilor pentru meniuri de efectuat; adițional, se poate include cod de inițializare şi cod de curățire); codul de inițializare se execută înainte ca meniul sistem să apară şi se poate include cod pentru deschiderea de fişiere, declararea de variabile; codul de curățire conține ceea ce se va executa după codul de definire a meniului şi poate face meniul sau elementele din meniu selectabile sau neselectabile;
4. Generarea programului pentru meniu;
5. Execuția programului pentru a testa sistemul.

Aplicația 5.

Să se creeze un meiul după modelul:



29. Dezvoltarea de meniuri pentru aplicații

Crearea unui meniu pentru o aplicație este asociată de obicei cu integrarea tuturor componentelor aplicației într-un ansamblu care să permită execuția fiecărei componente. Finalitatea acestui proces reprezintă sistemul de gestiune al bazei (sau bazelor) de date client.

Pentru construcția meniului sistemului se poate folosi aplicația expert *Menu Builder* sau se poate construi direct prin instrucțiuni, așa cum se va vedea în codul generat de VFP în urma generării unui meniu.

Aplicația 1.

Să se construiască un meniu cu opțiuni de comandă pentru acțiunile: deschidere baza de date *Universitati*, afişare ferestre de Browse pentru tabelele *Institutii* şi *Contacte* şi o opțiune Help About.

Rezolvare. Se urmează paşii:

1. Se creează un director pentru stocarea fişierelor aplicației;
2. Se creează un nou proiect (*New/Project/New file*); fie acesta *proj1.pjx*;
3. În fereastra de comenzi se testează funcția GetDir(), Sys(5) şi Sys(2003): *GETDIR(“”,”default for the application”)* respectiv *Sys(5)* şi *Sys(2003)*;
4. În *Project Manager* la categoria *Code* se alege *Programs* şi aici *New...*;
5. În fereastra pe care sistemul *VFP* o deschide se va crea un nou program care va gestiona proiectul; fie acesta *Program1*;
6. Se introduc următoarele comenzi:

```
cale_veche = sys(5)+sys(2003)
messagebox("cale veche: "+cale_veche)
cale = GETDIR("","default for the application")
if like(cale,"")=.F.
  set default to &cale
  messagebox("noua cale: "+cale)
endif
* comenzi pentru executia aplicatiei
* ...
* sfarsit aplicatie
```

7. Se salvează fişierul; ne asigurăm că acesta a fost salvat în directorul proiectului *proj1.pjx*;
8. În *Project Manager* se selectează programul *program1* şi apoi din meniul VFP la *Project* se verifică că programul *program1* are opțiunea Set Main selectată (✓*Set Main*); în caz contrar se selectează; se execută şi testează programul (*Project Manager/Code/Programs/Program1/Run*);
9. Se creează un nou meniu sistem (*Project Manager/Other/Menus/New.../Menu*) cu ajutorul aplicației expert *Menu Designer*; acesta se poate realiza după structura:

\<File (Submenu)
\<Open... (Procedure)
\<Quit (Procedure)
\<Browse (Command) Browse
\<Help (Submenu)
\<About... (Procedure)

10. Cu *Menu Designer – menu1.mnx* deschis se selectează din meniul VFP *Menu/Generate...* pentru a genera codul programului de meniu; acesta conține definițiile pentru comenzi (*BAR*) şi submeniuri (*PAD/POPUP*); dintre fişierele generate de VFP sunt esențiale pentru sistem: *menu1.MNX* (care este o tabelă cu informații despre structura şi conținutul meniului), *menu1.MNT* (un fişier memo) şi *menu1.mpr* (care conține comenzile generate de aplicația expert pentru crearea meniului); în acest din urmă fişier se pot vedea comenzile generate (se poate deschide cu *Notepad*); de menționat că sistemul nu permite modificarea directă a acestui fişier, el fiind actualizat la execuție în conformitate cu definițiile din fişierul *MNX*;
11. În orice moment al construcției sale, se poate vedea configurația meniului (*Menu Designer – menu1.mnx/Preview*);
12. Execuția meniului (*Project Manager/Other/Menus/Menu1/Run*) este sortită eşecului; sistemul lansează meniul în execuție însă nu este pregătit pentru gestiunea acestuia;
13. Se aplică următoarele modificări meniului (din *Menu Designer – menu1*):
    a. Se editează conținutul procedurii pentru *Help/About*:

```
MessageBox("First menu VFP");
```

    b. Se editează conținutul procedurii pentru *File/Quit*:

```
MessageBox("Now exit the application")
set default to cale_veche+"\"
MessageBox("restaurare cale: "+sys(5)+sys(2003))
Cancel
```

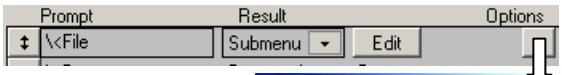
    c. Din meniul VFP se selectează *View/General Options...* şi apoi din containerul de butoane de opțiune *Location* se selectează opțiunea *Append*;
    d. Se salvează meniul (*File/Save*) şi se generează noul cod (*Menu/Generate...*);
14. Se aplică următoarele modificări programului *program1*:

```
* comenzi pentru executia aplicatiei
do menu1.mpr
read events
* sfarsit aplicatie
```

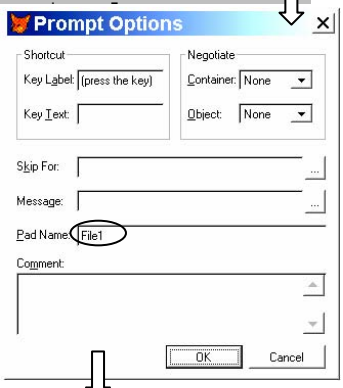
15. Tentativa de a executa meniul este în continuare sortită eşecului; se poate însă acum executa *program1* (*Project Manager/Code/Programs/Program1/Run*);
16. Programul produce afişarea casetelor de mesaj dorite (*Help/About...* şi *File/Quit*); observație: meniul aplicației se adaugă meniului sistemului *VFP* şi la ieşire rămâne activat

devenind parte integrantă a mediului VFP şi după terminarea aplicației (comanda *Cancel*); pentru a ajusta această situație trebuie scrise comenzile corespunzătoare în procedura pentru comanda *File/Quit*; un alt inconvenient este starea activă a comenzii (opțiunii) *Browse* chiar dacă nu a fost selectată nici o tabelă (*File/Open...*);

17. Se modifică meniul pentru a asocia nume *PAD*-urilor (*Menu Designer – menu1*); se ține seama de faptul că numele implicite ale meniurilor VFP sunt în forma: (Edit: nume PAD: _MSM_EDIT) şi nu se denumesc la fel!

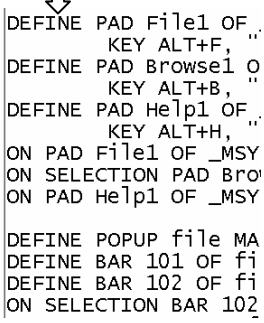


18. Astfel, se pot denumi pe calea:
    a. \<File (Submenu) *Options/Pad Name:* File1;
    b. \<Browse (Command) *Options/Pad Name:* Browse1;
    c. \<Help (Submenu) *Options/Pad Name:* Help1;



19. Se pot asocia (însă numere) şi pentru BAR-uri (BAR #);
    a. \<Open... (Procedure) *Options/Bar #:* 101;
    b. \<Quit (Procedure) *Options/Bar #:* 102;
    c. \<About... (Procedure) *Options/Bar #:* 103;

20. Se generează codul (*Menu/Generate...*);



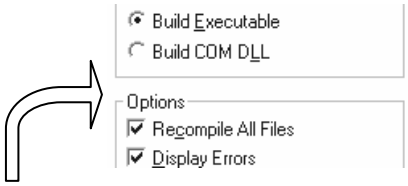
```
DEFINE PAD File1 OF
        KEY ALT+F, "
DEFINE PAD Browse1 O
        KEY ALT+B, "
DEFINE PAD Help1 OF
        KEY ALT+H, "
ON PAD File1 OF _MSY
ON SELECTION PAD Bro
ON PAD Help1 OF _MSY

DEFINE POPUP file MA
DEFINE BAR 101 OF fi
DEFINE BAR 102 OF fi
ON SELECTION BAR 102
```

21. Se vizualizează codul generat pentru a ne asigura de modificări;
22. Pentru dezactivarea lui *Browse* din meniu în *program1*:

do menu1.mpr
Set Skip of Pad Browse1 of _MSYSMENU .T.
read events

23. Se execută programul program1;
24. Pentru revenirea la meniul normal este acum necesară reîncărcarea aplicației *MVFP*;
25. Pentru ca aplicația să restaureze mediul VFP implicit se va modifica sfârşitul procedurii corespunzătoare comenzii *File/Quit* astfel:

messagebox("restaurare cale: "+sys(5)+sys(2003))
SET SYSMENU TO DEFAULT
cancel



26. Se poate genera acum aplicația executabilă pe calea *Project Manager – proj1/Code/Programs/program1/Build...*;
27. Se testează aplicația (din *Windows Explorer*); se completează acum cu procedurile lipsă;
28. La începutul programului *program1* se va adăuga o variabilă *tabela_use*:

tabela_use = ""
cale_veche = sys(5)+sys(2003)

29. Se încarcă din nou *menu1* în *Menu Designer*; se completează procedura pentru *File/Open...*: 


```
tabela_use1 = GetFile('DBF', 'Browse a Table:', 'Browse', 0, 'Browse')
if like(tabela_use, tabela_use1)
        Return                          && do nothing
```

```
endif
if like(tabela_use1, "")
        Return                          && do nothing
endif
if Used(tabela_use1)
        use                             && inchide tabela precedenta
        i = 1
        On Error i = 0
                USE &tabela_use1 in 0 again shared
        On Error
        if (i = 0)
                use &tabela_use         && redeschide tabela precedenta
                MessageBox("Table opened in exclusive mode by another program")
                Return
        else
                tabela_use = tabela_use1
                Set Skip of Pad Browse1 of _MSYSMENU .F.
                return
        endif
endif
use                                     && inchide tabela precedenta
i = 1
On Error i = 0
        USE &tabela_use1
On Error
if (i = 0)
        use &tabela_use                 && redeschide tabela precedenta
        MessageBox("Table opened in exclusive mode by another program")
        Return
else
        tabela_use = tabela_use1
        Set Skip of Pad Browse1 of _MSYSMENU .F.
endif
```

30. Se poate adăuga un BAR *Close* (*File/Close*) a cărui procedură să dezactiveze pe *Browse*:

```
use
tabela_use = ""
Set Skip of Pad Browse1 of _MSYSMENU .T.
```

31. Se lansează în execuție şi se testează aplicația *program1*; se generează executabilul;

32. Se pot elibera din memorie inclusiv elementele sistem, ca în exemplul:

RELEASE PAD _MEDIT OF _MSYSMENU

Aplicația 2.

Un meniu creat fără constructorul de meniuri.

Soluție. Se creează un nou program (File/New/Program/New file); se salvează cu numele Definpad.prg.

```
filen = Locfile("definpad", "prg,app,exe", "Locate definpad.prg")
set default to substr(filen,1,RAT("DEFINPAD.",filen)-1)
Clear
```

```
Set Talk Off
Set Sysmenu Save
Set Sysmenu To
Public Markpad
Markpad = .T.
Define Pad Syspad  Of _Msysmenu Prompt '\<System'  Color Scheme 3 ;
  Key Alt+S, ''
Define Pad Editpad Of _Msysmenu Prompt '\<Edit'  Color Scheme 3 ;
  Key Alt+E, ''
Define Pad Recordpad Of _Msysmenu Prompt '\<Record'  Color Scheme 3 Key Alt+R, ''
Define Pad Windowpad Of _Msysmenu Prompt '\<Window'  Color Scheme 3 ;
  Key Alt+W, ''
Define Pad Reportpad Of _Msysmenu Prompt 'Re\<Ports' Color Scheme 3 ;
  Key Alt+P, ''
Define Pad Exitpad Of _Msysmenu Prompt 'E\<Xit'  Color Scheme 3 ;
  Key Alt+X, ''
On Selection Menu _Msysmenu ;
  Do Choice In Definpad With Pad( ), Menu( )
Procedure Choice
Parameter Mpad, Mmenu
Wait Window 'You Chose ' + Mpad + ;
  ' From Menu ' + Mmenu Nowait
Set Mark Of Pad (Mpad) Of _Msysmenu To ;
  ! Mrkpad('_Msysmenu', Mpad)
Markpad = ! Markpad
If Mpad = 'EXITPAD'
  Set Sysmenu To Default
Endif
```

Aplicația 3.

Un meniu creat fără constructorul de meniuri.

Soluție. Se creează un nou program (File/New/Program/New file); se salvează cu numele Definbar.prg.

```
filen = Locfile("definbar", "prg,app,exe", "Locate definbar.prg")
set default to substr(filen,1,RAT("DEFINBAR.",filen)-1)
CLEAR
SET SYSMENU SAVE
SET SYSMENU TO
DEFINE PAD convpad OF _MSYSMENU PROMPT '\<Conversions' COLOR SCHEME 3 ;
  KEY ALT+C, ''
DEFINE PAD cardpad OF _MSYSMENU PROMPT 'Card \<Info' COLOR SCHEME 3 ;
  KEY ALT+I, ''
ON PAD convpad OF _MSYSMENU ACTIVATE POPUP conversion
ON PAD cardpad OF _MSYSMENU ACTIVATE POPUP cardinfo
DEFINE POPUP conversion MARGIN RELATIVE COLOR SCHEME 4
DEFINE BAR 1 OF conversion PROMPT 'Ar\<ea' KEY CTRL+E, '^E'
DEFINE BAR 2 OF conversion PROMPT '\<Length' ;
  KEY CTRL+L, '^L'
DEFINE BAR 3 OF conversion PROMPT 'Ma\<ss' ;
  KEY CTRL+S, '^S'
```

```
DEFINE BAR 4 OF conversion PROMPT 'Spee\<d' ;
  KEY CTRL+D, '^D'
DEFINE BAR 5 OF conversion PROMPT '\<Temperature' ;
  KEY CTRL+T, '^T'
DEFINE BAR 6 OF conversion PROMPT 'T\<ime' ;
  KEY CTRL+I, '^I'
DEFINE BAR 7 OF conversion PROMPT 'Volu\<me' ;
  KEY CTRL+M, '^M'
ON SELECTION POPUP conversion;
  DO choice IN definbar WITH PROMPT( ), POPUP( )
DEFINE POPUP cardinfo MARGIN RELATIVE COLOR SCHEME 4
DEFINE BAR 1 OF cardinfo PROMPT '\<View Charges' ;
  KEY ALT+V, ''
DEFINE BAR 2 OF cardinfo PROMPT 'View \<Payments' ;
  KEY ALT+P, ''
DEFINE BAR 3 OF cardinfo PROMPT 'Vie\<w Users' KEY ALT+W, ''
DEFINE BAR 4 OF cardinfo PROMPT '\-'
DEFINE BAR 5 OF cardinfo PROMPT '\<Charges '
DEFINE BAR 6 OF cardinfo PROMPT '\-'
DEFINE BAR 7 OF cardinfo PROMPT 'E\<xit '
ON SELECTION POPUP cardinfo;
  DO choice IN definbar WITH PROMPT( ), POPUP( )
PROCEDURE choice
PARAMETERS mprompt, mpopup
WAIT WINDOW 'You chose ' + mprompt + ;
  ' from popup ' + mpopup NOWAIT
IF mprompt = 'Exit'
  SET SYSMENU TO DEFAULT
ENDIF
```

Aplicația 4.

Un meniu creat fără constructorul de meniuri.

Soluție. Se creează un nou program (File/New/Program/New file); se salvează cu numele Definmenu.prg.

```
filen = Locfile("definmenu", "prg,app,exe", "Locate definmenu.prg")
set default to substr(filen,1,RAT("DEFINMENU.",filen)-1)
CLEAR
SET SYSMENU SAVE
SET SYSMENU TO
ON KEY LABEL ESC KEYBOARD CHR(13)
DEFINE MENU example BAR AT LINE 1
DEFINE PAD convpad OF example PROMPT '\<Conversions' COLOR SCHEME 3 ;
  KEY ALT+C, ''
DEFINE PAD cardpad OF example PROMPT 'Card \<Info' COLOR SCHEME 3 ;
  KEY ALT+I, ''
ON PAD convpad OF example ACTIVATE POPUP conversion
ON PAD cardpad OF example ACTIVATE POPUP cardinfo
DEFINE POPUP conversion MARGIN RELATIVE COLOR SCHEME 4
DEFINE BAR 1 OF conversion PROMPT 'Ar\<ea' ;
```

```
   KEY CTRL+E, '^E'
DEFINE BAR 2 OF conversion PROMPT '\<Length' ;
   KEY CTRL+L, '^L'
DEFINE BAR 3 OF conversion PROMPT 'Ma\<ss' ;
   KEY CTRL+S, '^S'
DEFINE BAR 4 OF conversion PROMPT 'Spee\<d' ;
   KEY CTRL+D, '^D'
DEFINE BAR 5 OF conversion PROMPT '\<Temperature' ;
   KEY CTRL+T, '^T'
DEFINE BAR 6 OF conversion PROMPT 'T\<ime' ;
   KEY CTRL+I, '^I'
DEFINE BAR 7 OF conversion PROMPT 'Volu\<me' ;
   KEY CTRL+M, '^M'
ON SELECTION POPUP conversion DO choice IN defimenu WITH PROMPT( ), POPUP( )
DEFINE POPUP cardinfo MARGIN RELATIVE COLOR SCHEME 4
DEFINE BAR 1 OF cardinfo PROMPT '\<View Charges' ;
   KEY ALT+V, ''
DEFINE BAR 2 OF cardinfo PROMPT 'View \<Payments' ;
   KEY ALT+P, ''
DEFINE BAR 3 OF cardinfo PROMPT 'Vie\<w Users' ;
   KEY ALT+W, ''
DEFINE BAR 4 OF cardinfo PROMPT '\-'
DEFINE BAR 5 OF cardinfo PROMPT '\<Charges '
ON SELECTION POPUP cardinfo;
   DO choice IN defimenu WITH PROMPT( ), POPUP( )
ACTIVATE MENU example
DEACTIVATE MENU example
RELEASE MENU example EXTENDED
SET SYSMENU TO DEFAULT
ON KEY LABEL ESC
PROCEDURE choice
PARAMETERS mprompt, mpopup
WAIT WINDOW 'You chose ' + mprompt + ;
    ' from popup ' + mpopup NOWAIT
```

Aplicația 5.

Un meniu creat fără constructorul de meniuri cu submeniuri.

Soluție. Se creează un nou program (File/New/Program/New file). Se salvează şi se execută.

```
DEFINE WINDOW wOrder FROM 10,0 TO 13,39
DEFINE MENU mnuDinner
DEFINE PAD padOne OF mnuDinner PROMPT '\<Main Course' KEY ALT+M, ''
DEFINE PAD padTwo OF mnuDinner PROMPT '\<Dessert'   KEY ALT+D, ''
ON PAD padOne OF mnuDinner ACTIVATE POPUP popMainCourse
ON PAD padTwo OF mnuDinner ACTIVATE POPUP dessert
DEFINE POPUP popMainCourse MARGIN MESSAGE ;
   'We have burgers and pizza today'
DEFINE BAR 1 OF popMainCourse PROMPT '\<Hamburgers'
DEFINE BAR 2 OF popMainCourse PROMPT '\<Pizza'
ON BAR 1 OF popMainCourse ACTIVATE POPUP burger
ON BAR 2 OF popMainCourse ACTIVATE POPUP pizza
```

```
DEFINE POPUP burger MARGIN MESSAGE ;
  'What would you like on your burger?'
DEFINE BAR 1 OF burger PROMPT '\<Ketchup'
DEFINE BAR 2 OF burger PROMPT '\<Mustard'
DEFINE BAR 3 OF burger PROMPT '\<Onions'
DEFINE BAR 4 OF burger PROMPT '\<Pickles'
DEFINE POPUP pizza MARGIN MESSAGE 'Here are the available toppings'
DEFINE BAR 1 OF pizza PROMPT '\<Anchovies'
DEFINE BAR 2 OF pizza PROMPT '\<Green Peppers'
DEFINE BAR 3 OF pizza PROMPT '\<Olives'
DEFINE BAR 4 OF pizza PROMPT '\<Pepperoni'
ON BAR 3 OF pizza ACTIVATE POPUP olives
DEFINE POPUP olives MARGIN
DEFINE BAR 1 OF olives PROMPT '\<Black' MESSAGE 'Black olives?'
DEFINE BAR 2 OF olives PROMPT '\<Green' MESSAGE 'Green olives?'
DEFINE POPUP dessert MARGIN MESSAGE 'Our dessert offerings'
DEFINE BAR 1 OF dessert PROMPT '\<Brownies'
DEFINE BAR 2 OF dessert PROMPT '\<Cookies'
DEFINE BAR 3 OF dessert PROMPT '\<Ice Cream'
DEFINE BAR 4 OF dessert PROMPT '\<Pie'
ON BAR 4 OF dessert ACTIVATE POPUP pie
DEFINE POPUP pie MARGIN MESSAGE 'What kind of pie?'
DEFINE BAR 1 OF pie PROMPT '\<Blueberry'
DEFINE BAR 2 OF pie PROMPT '\<Cherry'
DEFINE BAR 3 OF pie PROMPT '\<Peach'
DEFINE BAR 4 OF pie PROMPT '\<Rhubarb'
ON SELECTION POPUP ALL DO yourchoice
ACTIVATE MENU mnuDinner
PROCEDURE yourchoice
ACTIVATE WINDOW wOrder
CLEAR
DO CASE
  CASE POPUP( ) = 'BURGER'
    @ 0,0 SAY 'A ' + POPUP( ) + ' order:'
    @ 1,0 SAY 'You ordered a burger with ' + LOWER(PROMPT( ))
  CASE POPUP( ) = 'PIZZA'
    @ 0,0 SAY 'A ' + POPUP( ) + ' order:'
    @ 1,0 SAY 'You ordered a pizza with ' + LOWER(PROMPT( ))
  CASE POPUP( ) = 'OLIVES'
    @ 0,0 SAY 'A ' + POPUP( ) + ' order:'
    @ 1,0 SAY 'You ordered a pizza with ' ;
      + LOWER(PROMPT( )) + ' olives'
  CASE POPUP( ) = 'DESSERT'
    @ 0,0 SAY 'A ' + POPUP( ) + ' order:'
    @ 1,0 SAY 'You ordered ' + LOWER(PROMPT( )) + ' for dessert'
  CASE POPUP( ) = 'PIE'
    @ 0,0 SAY 'A ' + POPUP( ) + ' order:'
    @ 1,0 SAY 'You ordered ' + LOWER(PROMPT( )) + ' pie'
ENDCASE
WAIT WINDOW
DEACTIVATE WINDOW wOrder
RETURN
```

Aplicația 6.

Un meniu creat fără constructorul de meniuri.

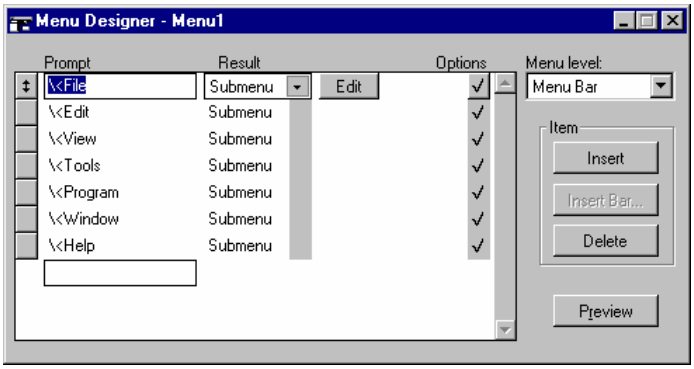
Soluție. Se creează un nou program (File/New/Program/New file); se salvează cu numele Definpop.prg.

```
filen = Locfile("definpop", "prg,app,exe", "Locate definpop.prg")
set default to substr(filen,1,RAT("DEFINPOP.",filen)-1)
CLEAR
SET SYSMENU SAVE
SET SYSMENU TO
DEFINE PAD convpad OF _MSYSMENU PROMPT '\<Conversions' COLOR SCHEME 3 ;
  KEY ALT+C, ''
DEFINE PAD cardpad OF _MSYSMENU PROMPT 'Card \<Info' COLOR SCHEME 3 ;
  KEY ALT+I, ''
ON PAD convpad OF _MSYSMENU ACTIVATE POPUP conversion
ON PAD cardpad OF _MSYSMENU ACTIVATE POPUP cardinfo
DEFINE POPUP conversion MARGIN RELATIVE COLOR SCHEME 4
DEFINE BAR 1 OF conversion PROMPT 'Ar\<ea' KEY CTRL+E, '^E'
DEFINE BAR 2 OF conversion PROMPT '\<Length' ;
  KEY CTRL+L, '^L'
DEFINE BAR 3 OF conversion PROMPT 'Ma\<ss' ;
  KEY CTRL+S, '^S'
DEFINE BAR 4 OF conversion PROMPT 'Spee\<d' ;
  KEY CTRL+D, '^D'
DEFINE BAR 5 OF conversion PROMPT '\<Temperature' ;
  KEY CTRL+T, '^T'
DEFINE BAR 6 OF conversion PROMPT 'T\<ime' ;
  KEY CTRL+I, '^I'
DEFINE BAR 7 OF conversion PROMPT 'Volu\<me' ;
  KEY CTRL+M, '^M'
ON SELECTION POPUP conversion;
  DO choice IN definpop WITH PROMPT( ), POPUP( )
DEFINE POPUP cardinfo MARGIN RELATIVE COLOR SCHEME 4
DEFINE BAR 1 OF cardinfo PROMPT '\<View Charges' ;
  KEY ALT+V, ''
DEFINE BAR 2 OF cardinfo PROMPT 'View \<Payments' ;
  KEY ALT+P, ''
DEFINE BAR 3 OF cardinfo PROMPT 'Vie\<w Users' ;
  KEY ALT+W, ''
DEFINE BAR 4 OF cardinfo PROMPT '\-'
DEFINE BAR 5 OF cardinfo PROMPT '\<Charges '
DEFINE BAR 6 OF cardinfo PROMPT '\-'
DEFINE BAR 7 OF cardinfo PROMPT 'E\<xit '
ON SELECTION POPUP cardinfo;
  DO choice IN definpop WITH PROMPT( ), POPUP( )
PROCEDURE choice
PARAMETERS mprompt, mpopup
WAIT WINDOW 'You chose ' + mprompt + ' from popup ' + mpopup NOWAIT
IF mprompt = 'Exit'
  SET SYSMENU TO DEFAULT
ENDIF
```

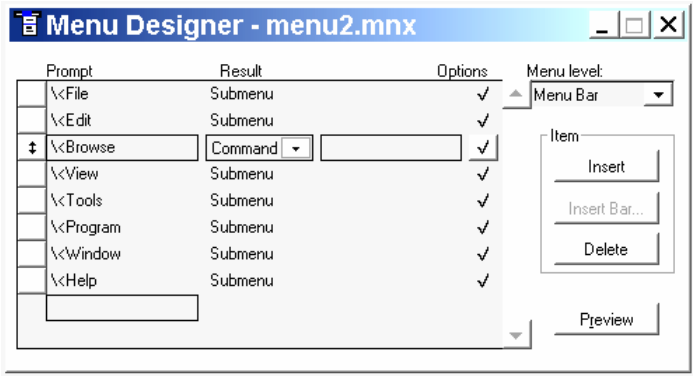
Generarea meniurilor cu *Quick Menu*

Sistemul VFP pune la dispoziția utilizatorului posibilitatea de a genera meniuri care să includă opțiunile de bază din mediul VFP. Este avantajos să se folosească această facilitate pentru a putea beneficia de resursele sistemului *Windows* (de exemplu de *Clipboard* prin intermediul meniului *Edit*). Pentru a folosi această resursă se poate merge pe calea:

1. Se creează un nou meniu în cadrul proiectului *proj1.pjx*; fie acesta *menu2.mnx* (vezi Aplicația 1); se salvează cu numele *menu2.mnx*;
2. Cu aplicația expert *Menu Designer – Menu1* activă se selectează *Quick Menu* din *Menu*;
3. Sistemul VFP va genera un meniu în forma:



4. Se poate acum personaliza acest meniu prin adăugarea de opțiuni şi asocierea de evenimente; de exemplu se poate adăuga la acest meniu opțiunea *Browse* şi modifica opțiunile *File* şi *Help* şi asocia la acestea evenimente la fel ca în *Aplicația 1* când meniul va arăta în forma:



5. Se generează meniul (din meniul VFP *Menu/Generate...*);
6. Se modifică programul *Program1* din proiectul *proj1* pentru a încărca acum meniul menu2.mpr:

```
tabela_use = ""
cale_veche = sys(5)+sys(2003)
messagebox("cale veche: "+cale_veche)
cale = GETDIR("","default for the application")
if like(cale,"")=.F.
  set default to &cale
  messagebox("noua cale: "+cale)
```

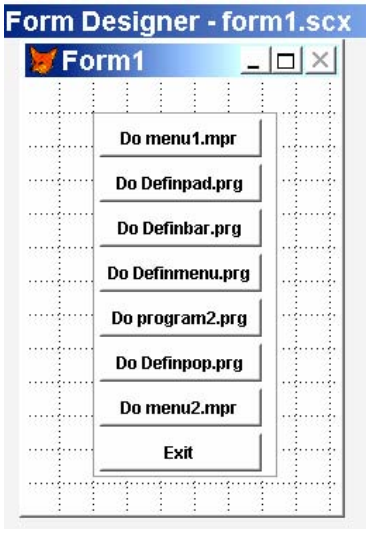
```
endif
* comenzi pentru executia aplicatiei
do menu2.mpr
Set Skip of Pad Browse1 of _MSYSMENU .T.
read events
* sfarsit aplicatie
```

7. Se execută programul *program1* (*Project Manager – proj1/Code/Programs/Program1/ Run*); se generează executabilul (*Project Manager – proj1/Code/Programs/Program1/ Build...*);

Execuția meniurilor din formulare

Odată create meniurile şi generat codul sursă al programului de meniu se poate folosi acesta pentru a executa meniul din formulare. Următoarea aplicație folosește codul programelor *menu1.mpr*, *Definpad.prg*, *Definbar.prg*, *Definmenu.prg*, *program2.prg* (Aplicația 5), *Definpop.prg* şi *menu2.mpr*. Se urmează paşii:

1. Se deschide proiectul *proj1.pjx* (Aplicația 1);
2. Se creează un nou formular (Project Manager – proj1/Documents/Forms/New.../New Form); se salvează cu numele *form1.scx*; se adaugă un container cu 8 butoane de comandă ca în figura:



3. Se asociază evenimente:
    a. Command1.Click:

```
Do menu1.mpr
Set Skip of Pad Browse1 of _MSYSMENU .T.
read events
```

    b. Command2.Click: *Do Definepad.prg*
    c. Command3.Click: *Do Definebar.prg*
    d. Command4.Click: *Do Definemenu.prg*
    e. Command5.Click: *Do program2.prg*

f. Command6.Click: *Do Definepop.prg*

g. Command7.Click: *Do menu2.mpr*

Do menu2.mpr
Set Skip of Pad Browse1 of _MSYSMENU .T.
read events

h. Command8.Click:

SET SYSMENU TO DEFAULT
ThisForm.Release
Cancel

4. Se modifică programul *program1*:

tabela_use = ""
public cale_veche
cale_veche = sys(5)+sys(2003)
...
** comenzi pentru executia aplicatiei*
Do Form form1.scx
Read events
set default to cale_veche+"\"
messagebox("restaurare cale: "+sys(5)+sys(2003))
** sfarsit aplicatie*

5. Se modifică procedurile Menu1/Quit şi Menu2/Quit şi se regenerează codul meniurilor:

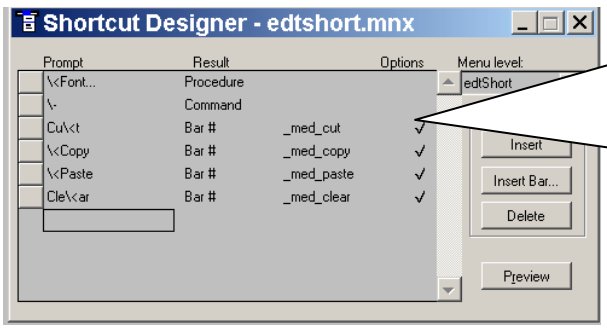
Messagebox("Now exit the application")
SET SYSMENU TO DEFAULT
Return to Master

6. Se execută formularul; se generează şi testează executabilul.

Generarea de meniuri contextuale

Un meniu contextual se activează atunci când se execută click dreapta pe un control sau pe un obiect şi reprezintă o modalitate rapidă de a afişa toate funcțiile aplicabile acelui obiect. Un meniu contextual se realizează pe calea *Project Manager/Other/Menus/New...*

Următorul exemplu implementează un meniu contextual pentru operații de editare. Poate fi preluat direct pe calea MSDN98\98VSa\1033\Samples\Solution\Vfp98\Solution când se copiază fişierele *Edtshort.mnx* şi *Edtshort.MNT* după care se adaugă la proiect (*Project Manager – proj1/Other/Menus/Add.../Edtshort.mnx*).



Message: "Removes the selection and places it onto the Clipboard"; Key Label: CTRL+X; Key Text: Ctrl+X
"Copies the selection onto the Clipboard"; CTRL+C; Ctrl+C
"Pastes the contents of the Clipboard"; CTRL+V; Ctrl+V
"Removes the selection and does not place it onto the Clipboard"

Shortcut Designer - edtshort - Font... Procedure

```
IF TYPE("m.oRef") = "O"
        m.cFont = GetFont()
        IF EMPTY(m.cFont)
                RETURN
        ENDIF
        m.commaLoc = AT(",",m.cFont)
        m.comma2Loc = AT(",",m.cFont,2)
        oRef.FontName = SUBSTR(m.cFont,1,m.commaLoc-1)
oRef.FontSize = VAL(SUBSTR(m.cFont,m.commaLoc+1,m.comma2Loc-m.commaLoc))
        oRef.FontBold = ATC("B",SUBSTR(m.cFont,m.comma2Loc))#0
        oRef.FontItalic = ATC("I",SUBSTR(m.cFont,m.comma2Loc))#0
ENDIF
```

7. Se generează codul meniului;
8. Se modifică formularul şi se adaugă evenimentul:

FORM1.RightClick

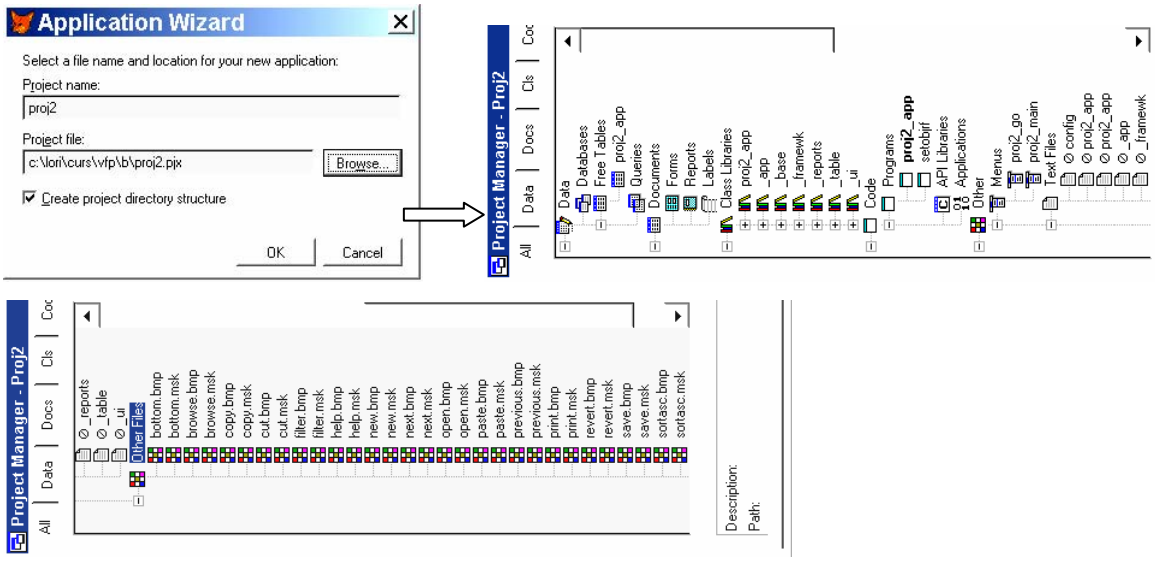
```
do Edtshort.mpr
read events
```

9. Se generează codul executabil şi se testează aplicația *proj1*.

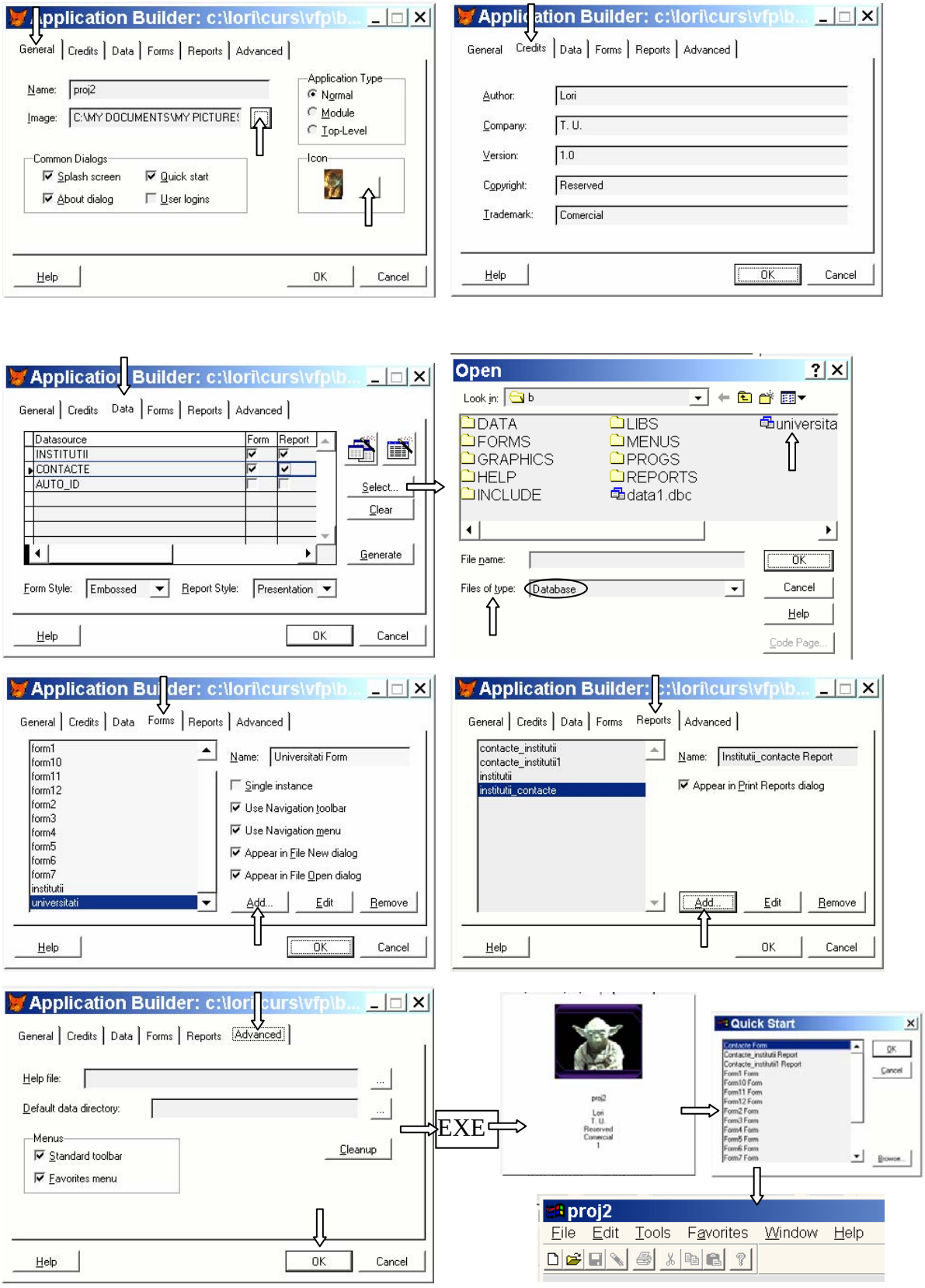
Generarea automată a aplicației cu aplicația expert *Application Wizard*

Se poate genera automat proiectul unei aplicații pe calea *File/New/Project/Wizard*. Următorul exemplu generează automat proiectul pentru baza de date *Universitati*. Se urmează paşii:

1. Se creează un nou director;
2. Se copiază fişierele din directorul proiectului Universitati mai puțin Universitati.PJT şi Universitati.pjx;
3. Se lansează în execuție aplicația expert *Application Wizard*;
4. Se definesc atributele proiectului:



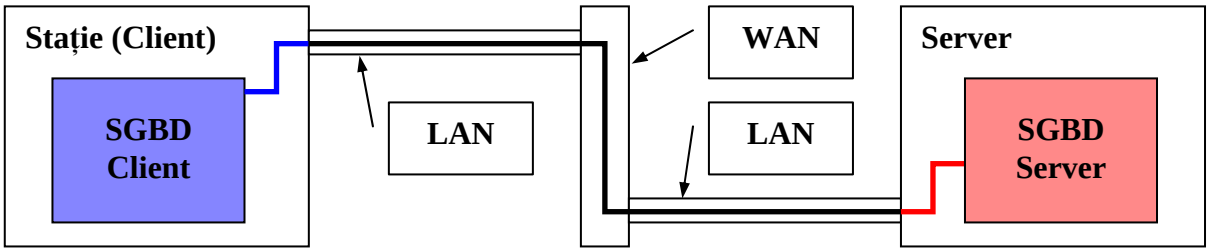
5. Sistemul VFP va genera un proiect GENERIC ca în figurile anterioare; în continuare cu ajutorul aplicației expert *Application Builder* se pot defini elementele constitutive ale proiectului:



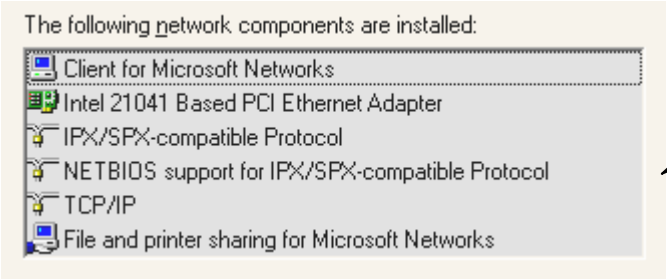
Noul proiect generat include acum formularele şi rapoartele create; se generează executabilul şi se testează aplicația.

30. Baze de date externe şi aplicații client – server

Aproape toate SGBD-urile care se execută sub sisteme de operare cu nucleu (kernel) client-server (Windows 3.11 NT, 9.x, NT 4.x, XP; Novell Netware 3.x, 4.x, 5.x, Linux, Unix, OS/2, BSD) au prevăzute protocoale de comunicare pentru baze de date situate la distanță (pe stații sau pe servere). Există o mare diversitate de sisteme de operare atât pentru stații (clienți) cât şi pentru servere. Din acest motiv, comunicarea se realizează prin intermediul unui protocol, care reprezintă de fapt implementarea unor convenții în ceea ce priveşte comunicarea de date sub formă de librării şi pachete de programe.



Protocoalele de transfer de date (şi driverele aferente) ale stației şi serverului sunt furnizate de către sistemul de operare, aşa cum se poate observa în figurile de mai jos. Acestea se configurează urmând prescripțiile din documentația sistemului de operare şi ale distribuitorului de internet (pentru WAN).



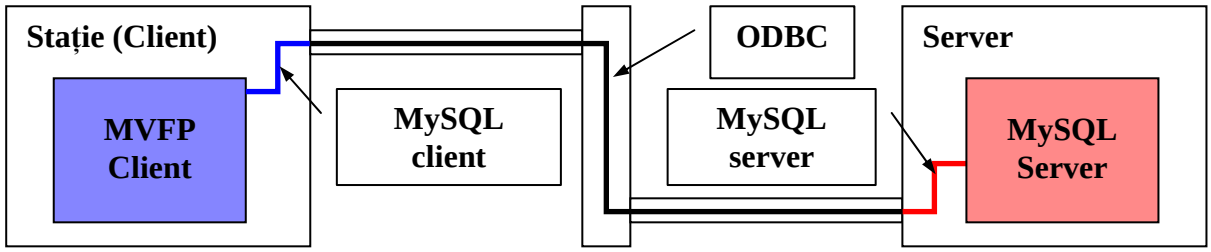
Win 9.x

Free BSD

```
bash-2.05a$ ifconfig
de0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> mtu 1500
        inet 193.226.7.140 netmask 0xffffff80 broadcast 193.226.7.255
        inet6 fe80::200:c0ff:fe91:ebf5%de0 prefixlen 64 scopeid 0x1
        ether 00:00:c0:91:eb:f5
        media: Ethernet autoselect (10baseT/UTP)
        status: active
lo0: flags=8049<UP,LOOPBACK,RUNNING,MULTICAST> mtu 16384
        inet6 ::1 prefixlen 128
        inet6 fe80::1%lo0 prefixlen 64 scopeid 0x2
        inet 127.0.0.1 netmask 0xff000000
ppp0: flags=8010<POINTOPOINT,MULTICAST> mtu 1500
sl0: flags=c010<POINTOPOINT,LINK2,MULTICAST> mtu 552
faith0: flags=8002<BROADCAST,MULTICAST> mtu 1500
```

Cazul cel mai complex este atunci când sistemele de operare sunt diferite şi SGBD-urile trebuiesc configurate pentru comunicare atât pe server cât şi pe stație.

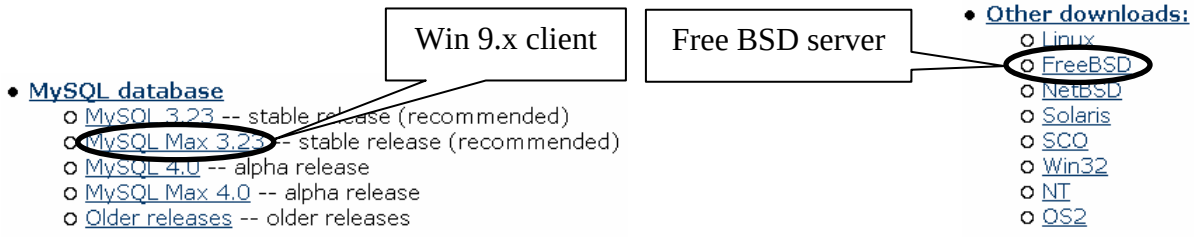
31. Configurarea unui client VFP/Win9.x pe un server MyQSL/FreeBSD



Pentru a începe configurarea unui sistem de exploatare a bazelor de date la distanță trebuie să ne asigurăm că dispunem de toate driverele necesare. În cazul în care am ales să instalăm un Server MySQL va trebui să aducem componentele necesare şi să le instalăm atât pe server cât şi pe stații.

Se parcurg următorii paşi:

1. Se alege pachetul de drivere şi programe corespunzătoare sistemului de operare (www.mysql.com/downloads):

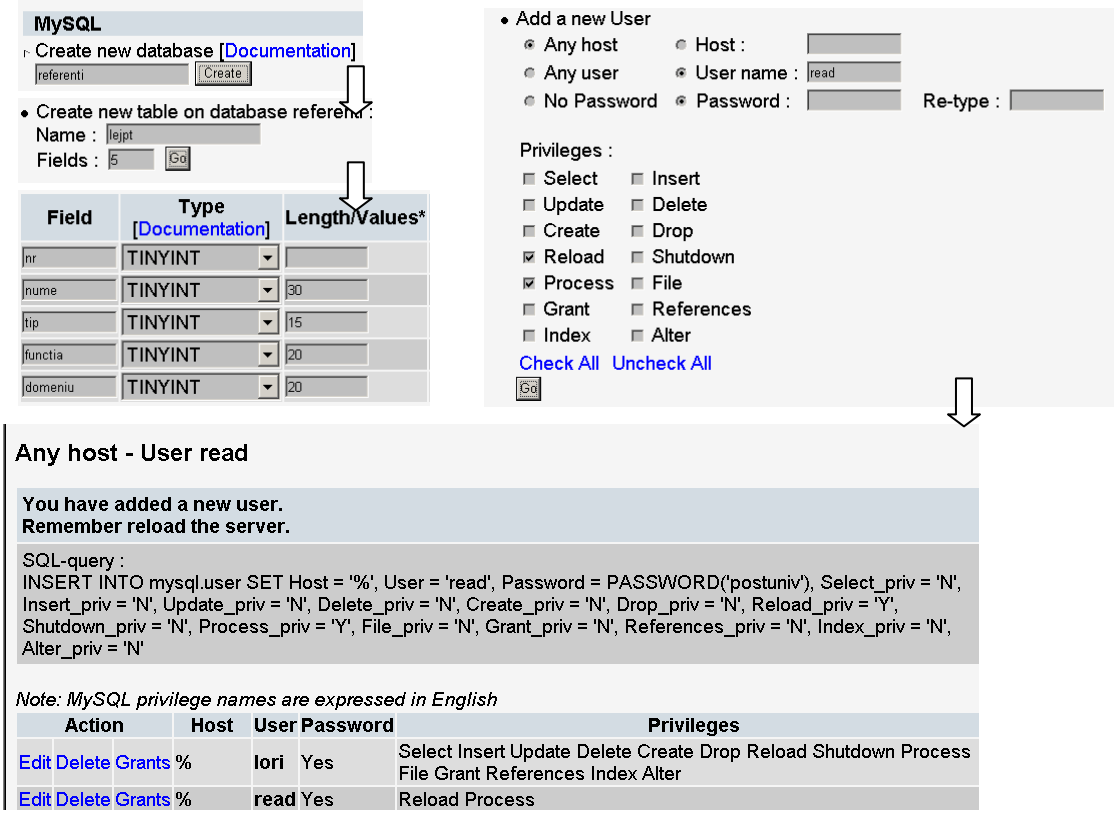


2. Se aduc şi se instalează acestea; MySQL server va descărca şi instala şi alte pachete adiționale pentru limbajul perl :

```
post-patch:
    ${PERL} -pi -e \
        "s@^(my $$Libs.*)-lpthread -ldl\";\$$@\1 ${PTHREAD_LIBS}\";@; \
        s@^(CFLAGS_GLOB.*)@\1 ${CFLAGS} ${PTHREAD_CFLAGS}@" \
    ${WRKSRC}/configure.pl
```

3. Dacă dorim să asigurăm o interfață grafică SGBD-ului de pe server, se poate instala acum phpMyAdmin (http://www.phpwizard.net/projects/phpMyAdmin/) care este un sistem care administrează MySQL prin protocolul http. Acesta oferă interfața grafică pentru:
    a. crearea şi ştergerea bazelor de date;
    b. crearea, copierea, ştergerea şi modificarea tabelelor;
    c. ştergerea, editarea şi adăugarea de câmpuri;
    d. executarea oricărei secvențe în limbajul SQL şi chiar a interogărilor înlănțuite;
    e. manipularea cheilor în câmpuri;
    f. încărcarea fişierelor text în tabele;
    g. crearea şi citirea de tabele;
    h. exportul datelor;
    i. administrarea multiplă de servere pe o singură bază de date.

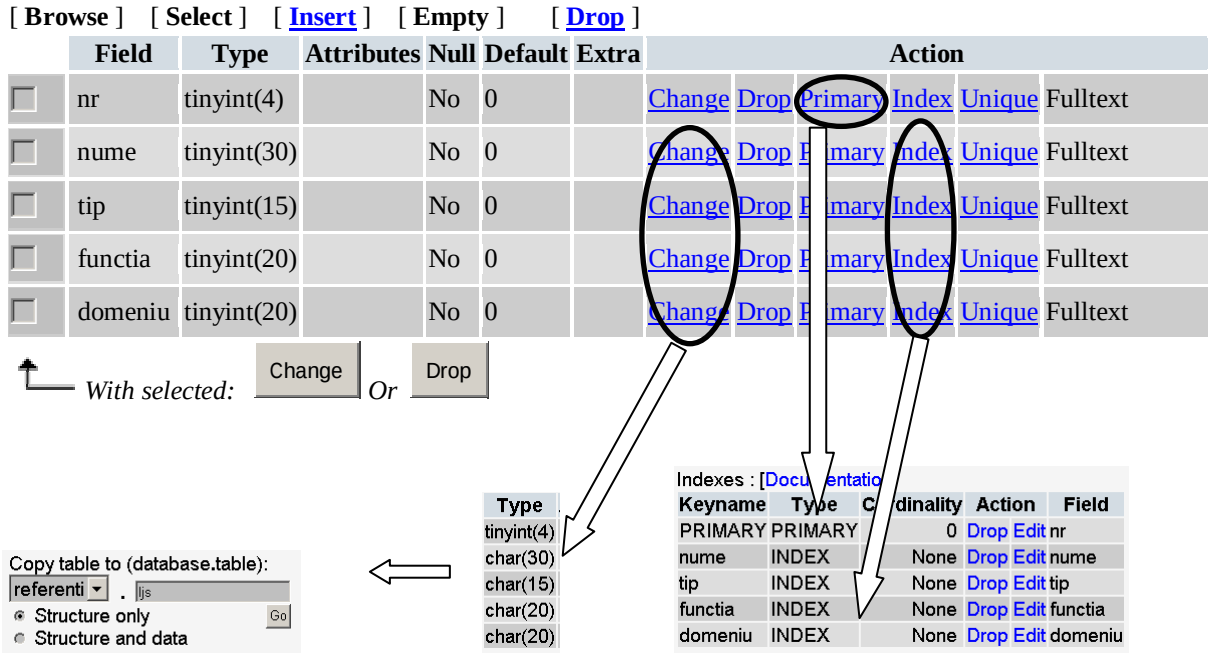
4. Cu ajutorul lui phpmyadmin se pot crea acum utilizatori care să acceseze bazele de date MySQL din interfața phpMyAdmin:



5. Interogarea SQL este în forma următoare:

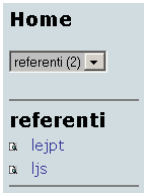
CREATE TABLE `lejpt` (`nr` TINYINT NOT NULL, `nume` TINYINT(30) NOT NULL, `tip` TINYINT(15) NOT NULL, `functia` TINYINT(20) NOT NULL, `domeniu` TINYINT(20) NOT NULL);

6. Se poate modifica acum structura tabelei. De exemplu se corectează câmpurile nume, tip, functia, domeniu din tinyint în char şi tipul indexului pentru câmpurile de tip caracter:



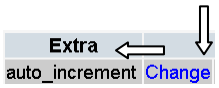
7. Se poate face o copie a structurii într-un nou tabel; fie acesta ljs; comanda SQL generată de phpMyadmin este:

```
CREATE TABLE `referenti`.`ljs` (`nr` tinyint(4) NOT NULL default '0',`nume` char(30) NOT NULL default '0',`tip` char(15) NOT NULL default '0',`functia` char(20) NOT NULL default '0',`domeniu` char(20) NOT NULL default '0',PRIMARY KEY (`nr`),KEY `nume` (`nume`),KEY `tip` (`tip`),KEY `functia` (`functia`),KEY `domeniu` (`domeniu`)) TYPE=MyISAM;
```



8. Acum baza de date referenti conține cele două tabele, lejpt și ljs;

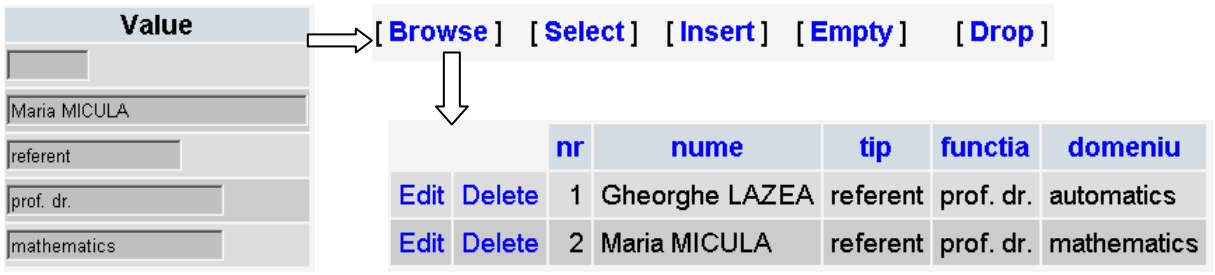
9. Se poate defini câmpul nr ca câmp autoincrement:




```
ALTER TABLE `referenti`.`lejpt` CHANGE `nr` `nr` TINYINT(4) DEFAULT '0' NOT NULL AUTO_INCREMENT
```

10. Se pot completa tabelele cu înregistrări pe calea:

11. Odată definită funcția autoincrement, nu mai este necesară completarea valorii pentru nr:



12. Comanda SQL este în forma:

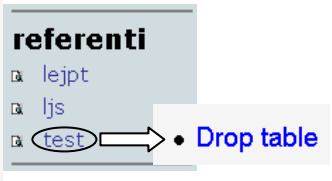
```
INSERT INTO `lejpt` (`nr`, `nume`, `tip`, `functia`, `domeniu`) VALUES ('', 'Valentin MILITARU', 'referent', 'prof. dr.', 'fizica');
```

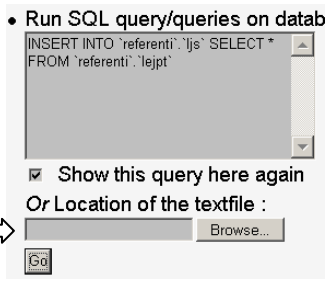
13. Corectarea unei valori introduse greșit se face cu comanda:

```
UPDATE `lejpt` SET `domeniu` = 'physics' WHERE `nr` = '3' LIMIT 1;
```

14. Rezultatul completării cu date poate arăta în felul următor:

		nr	nume	tip	functia	domeniu
Edit	Delete	1	Gheorghe LAZEA	referent	prof. dr.	automatics
Edit	Delete	2	Maria MICULA	referent	prof. dr.	mathematics
Edit	Delete	3	Valentin MILITARU	referent	prof. dr.	physics
Edit	Delete	4	Elena Maria PICA	referent	prof. dr.	chemistry
Edit	Delete	5	Tiberiu RUSU	referent	prof. dr.	engineering
Edit	Delete	6	Mircea SAVATTI	referent	prof. dr.	agronomy
Edit	Delete	7	Lorentz JANTSCHI	chief redactor	sef lucr. dr.	informatics
Edit	Delete	8	Florica ALDEA	redactor	sef lucr. dr.	applied mathematics
Edit	Delete	9	LASZLO Alexandru	redactor	prof. dr. gr. 1	linguistics

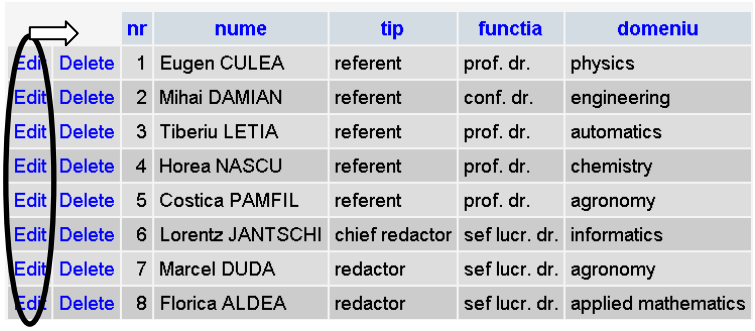




15. Se pot acum copia valorile din tabelul lejpt în tabelul ljs:

```
INSERT INTO `referenti`.`ljs` SELECT * FROM `referenti`.`lejpt`
```

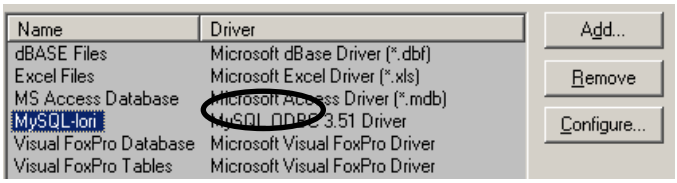
16. Dacă a fost creat un tabel care ulterior nu mai este de dorit a se păstra în baza de date se aplică comanda drop asupra acestuia;
17. După ce s-au copiat valorile din tabelul lejpt în tabelul ljs se pot modifica după dorință (întâi browse şi apoi edit):



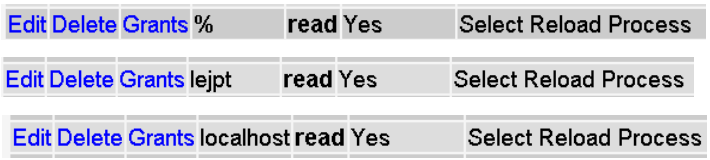
		nr	nume	tip	functia	domeniu
Edit	Delete	1	Eugen CULEA	referent	prof. dr.	physics
Edit	Delete	2	Mihai DAMIAN	referent	conf. dr.	engineering
Edit	Delete	3	Tiberiu LETIA	referent	prof. dr.	automatics
Edit	Delete	4	Horea NASCU	referent	prof. dr.	chemistry
Edit	Delete	5	Costica PAMFIL	referent	prof. dr.	agronomy
Edit	Delete	6	Lorentz JANTSCHI	chief redactor	sef lucr. dr.	informatics
Edit	Delete	7	Marcel DUDA	redactor	sef lucr. dr.	agronomy
Edit	Delete	8	Florica ALDEA	redactor	sef lucr. dr.	applied mathematics

Instalarea şi administrarea MySQL pe client

Se poate alege a se instala MyODBC-3.51.01 care după instalare se va regăsi în Control Panel/ODBC Data Sources (32bit); se va selecta şi se va configura pentru conectarea implicită pe serverul de date.

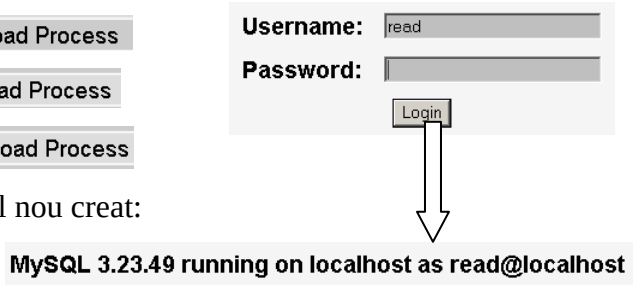


Pentru a permite accesul la citire în bazele de date de pe server, cu ajutorul lui phpMyAdmin se creează 3 utilizatori după modelul:

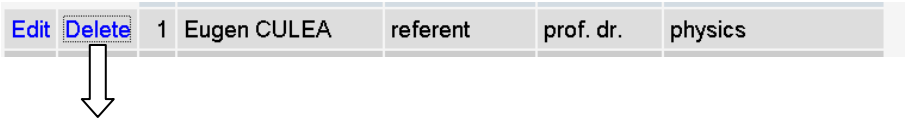


Edit	Delete	Grants	%	read	Yes	Select Reload Process
Edit	Delete	Grants	lejpt	read	Yes	Select Reload Process
Edit	Delete	Grants	localhost	read	Yes	Select Reload Process

Se testează accesul pentru utilizatorul nou creat:



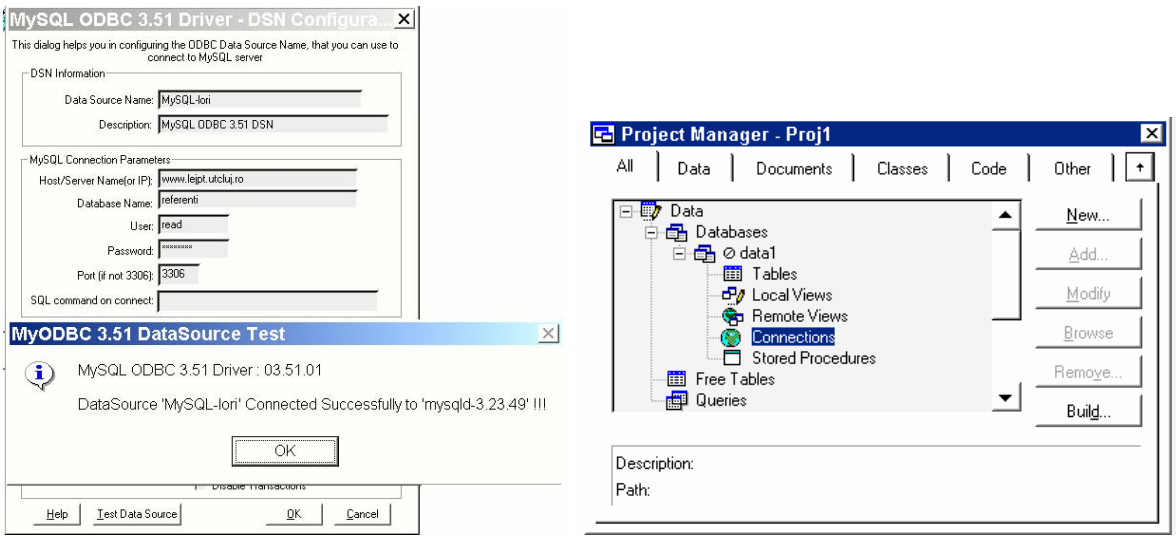
Tentativa de modificare a conținutului tabelelor este soldată eşecului:



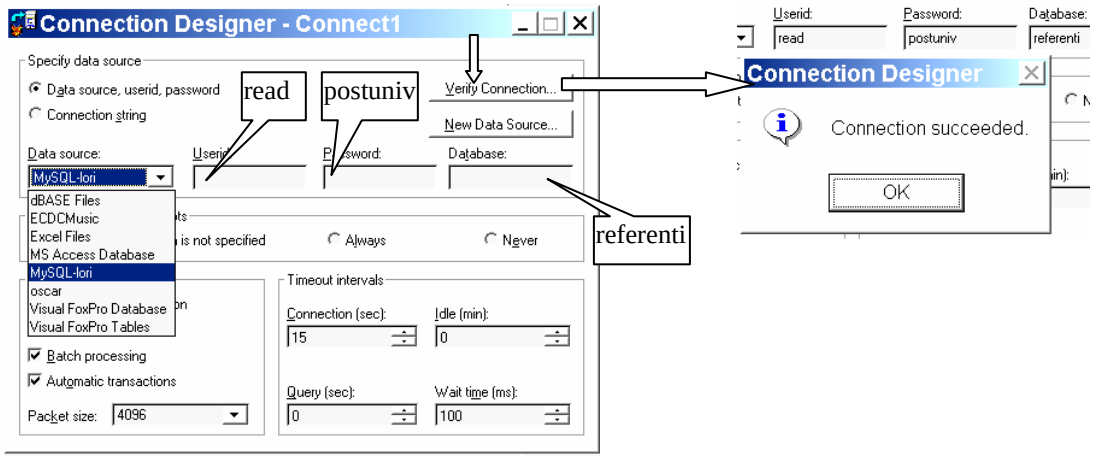
Edit	Delete	1	Eugen CULEA	referent	prof. dr.	physics

DELETE FROM `ljs` WHERE `nr` = '1' LIMIT 1
Access denied for user: 'read@localhost' to database 'referenti'

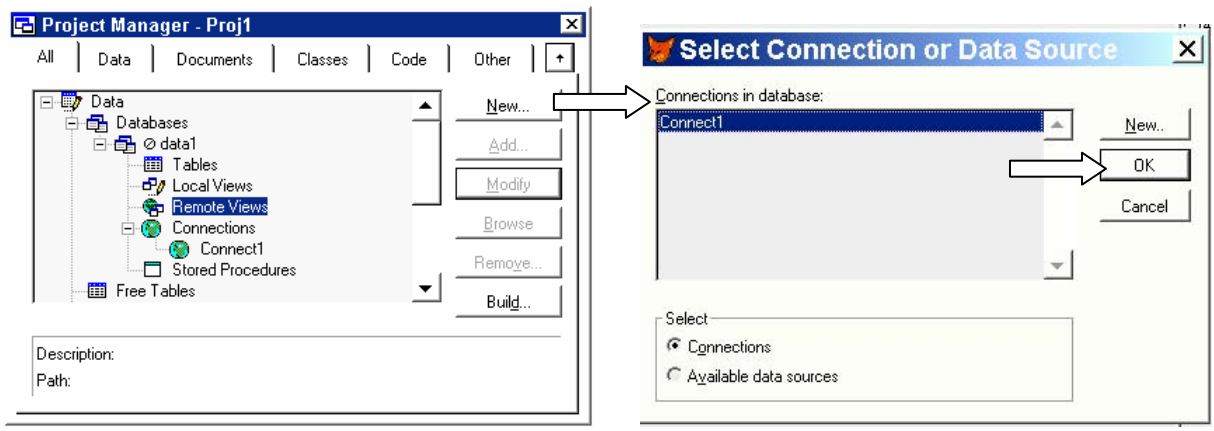
Se poate configura acum accesul de la distanță în baza de date, astfel încât în următoarea etapă se configurează MySQL pe client după modelul:



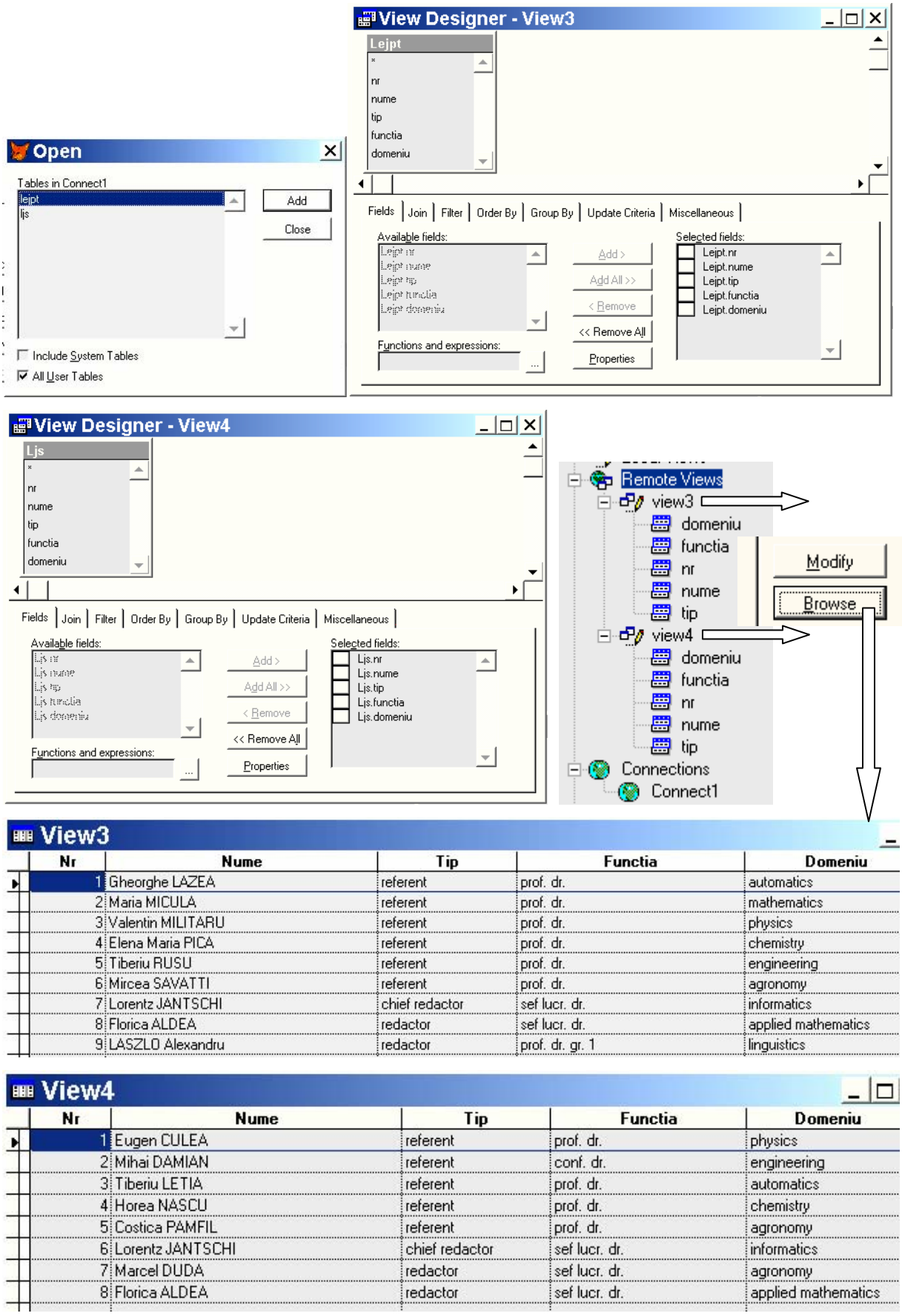
Se poate acum accesa baza de date şi din VFP. Se încarcă *Project Manager*, se deschide o bază de date sau se creează una nouă, se selectează Connections, apoi New... şi se urmează schema de mai jos:



Se salvează conexiunea în baza de date (connect1) şi se creează 2 vederi la distanță (Remote Views/New) din baza de date referenti:



Se poate crea o vedere care să conțină informațiile din tabelul ljs şi o vedere care să conțină informațiile din tabelul lejpt. Se vor include toate câmpurile din cele două tabele şi se vor salva acestea în baza de date:



View3

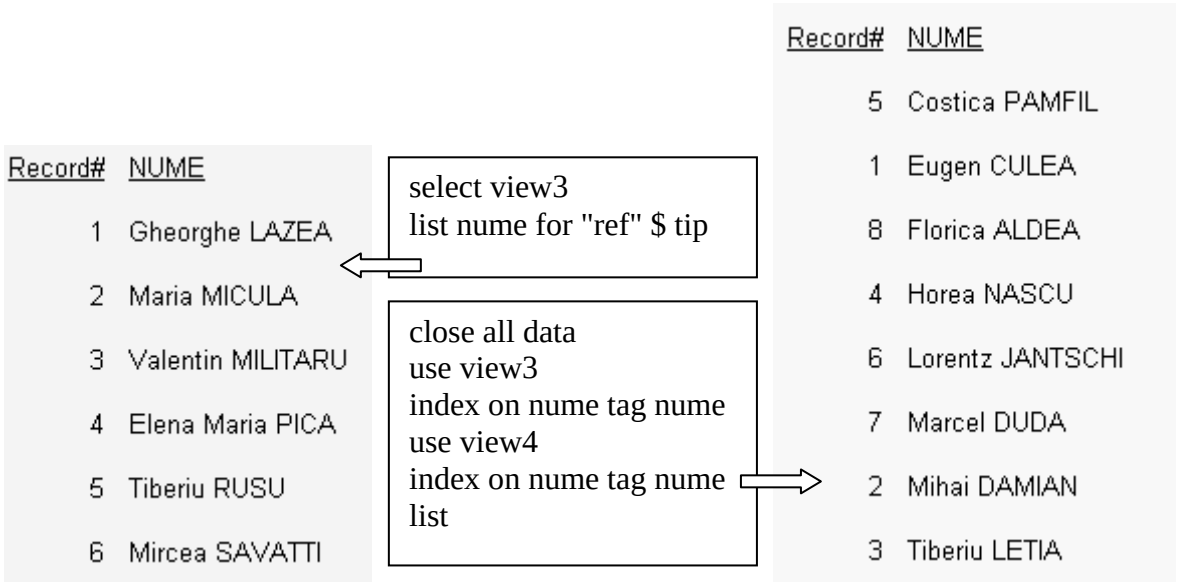
Nr	Nume	Tip	Functia	Domeniu
1	Gheorghe LAZEA	referent	prof. dr.	automatics
2	Maria MICULA	referent	prof. dr.	mathematics
3	Valentin MILITARU	referent	prof. dr.	physics
4	Elena Maria PICA	referent	prof. dr.	chemistry
5	Tiberiu RUSU	referent	prof. dr.	engineering
6	Mircea SAVATTI	referent	prof. dr.	agronomy
7	Lorentz JANTSCHI	chief redactor	sef lucr. dr.	informatics
8	Florica ALDEA	redactor	sef lucr. dr.	applied mathematics
9	LASZLO Alexandru	redactor	prof. dr. gr. 1	linguistics

View4

Nr	Nume	Tip	Functia	Domeniu
1	Eugen CULEA	referent	prof. dr.	physics
2	Mihai DAMIAN	referent	conf. dr.	engineering
3	Tiberiu LETIA	referent	prof. dr.	automatics
4	Horea NASCU	referent	prof. dr.	chemistry
5	Costica PAMFIL	referent	prof. dr.	agronomy
6	Lorentz JANTSCHI	chief redactor	sef lucr. dr.	informatics
7	Marcel DUDA	redactor	sef lucr. dr.	agronomy
8	Florica ALDEA	redactor	sef lucr. dr.	applied mathematics

Se pot executa vederile pe calea Remote Views/View3(4)/Browse. De asemenea, se pot realiza interogări, rapoarte şi formulare, acestea fiind tratate în continuare ca elemente aparţinătoare bazei de date locale.

Se pot da şi comenzi în fereastra de comenzi sau crea programe care să folosească aceste informații:




```
SELECT VIEW4.Nume; FROM DATA1!VIEW4 VIEW4;
   INNER JOIN DATA1!VIEW3 VIEW3 ON VIEW4.NUME = VIEW3.NUME;
   ORDER BY VIEW4.Nume to screen
```

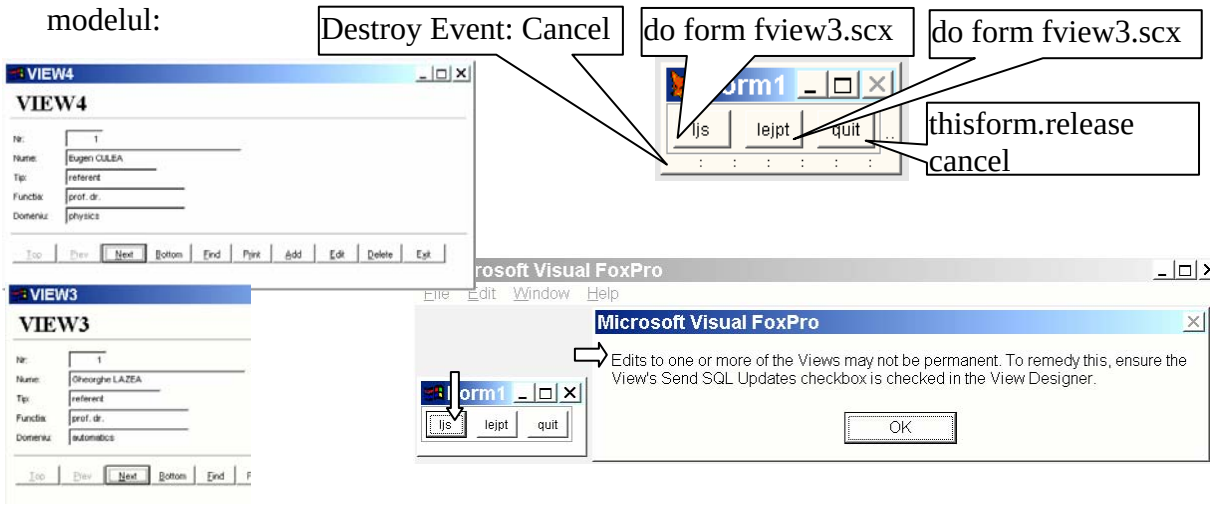
Se poate acum crea un executabil care să acceseze datele din conexiunea la distanță. Se urmează paşii:

1. Se creează un program principal pentru aplicație (Project Manager – proj1/Code/Programs/New...); fie acesta program1.prg;
2. Se introduc în acesta comenzile:

set default to GETDIR("","default for the application")

do form form1.scx

3. Se setează ca program principal al proiectului: Project Manager – proj1/Code/Programs/program1/Project/✓Set Main;
4. Se creează formularul form1 pe calea Project Manager – proj1/Forms/New Form după modelul:



32. Administrarea de la distanță a MyQSL/FreeBSD cu VFP/Win9.x

După instalarea MySQL pe server pentru a configura pentru acces partajat şi proteja serverul se poate folosi programul *mysql_setpermission*. Execuția acestui program necesită privilegii de administrator (su-2.05a#). În continuare configurarea serverului cu acest program se face prin intermediul unei interfețe de tip text:

Password for user to connect to MySQL:

##
Welcome to the permission setter 1.2 for MySQL.
##
What would you like to do:
1. Set password for a user.
2. Add a database + user privilege for that database.
- user can do all except all admin functions
3. Add user privilege for an existing database.
- user can do all except all admin functions
4. Add user privilege for an existing database.
- user can do all except all admin functions + no create/drop
5. Add user privilege for an existing database.
- user can do only selects (no update/delete/insert etc.)
0. exit this program
Make your choice [1,2,3,4,5,0]:

Prin această interfață se poate defini parola pentru superuser (root) şi se pot defini parolele şi drepturile de acces pentru toți utilizatorii serverului.

În continuare, administratorul poate configura şi exploata programatic serverul MySQL din interfața MVFP.

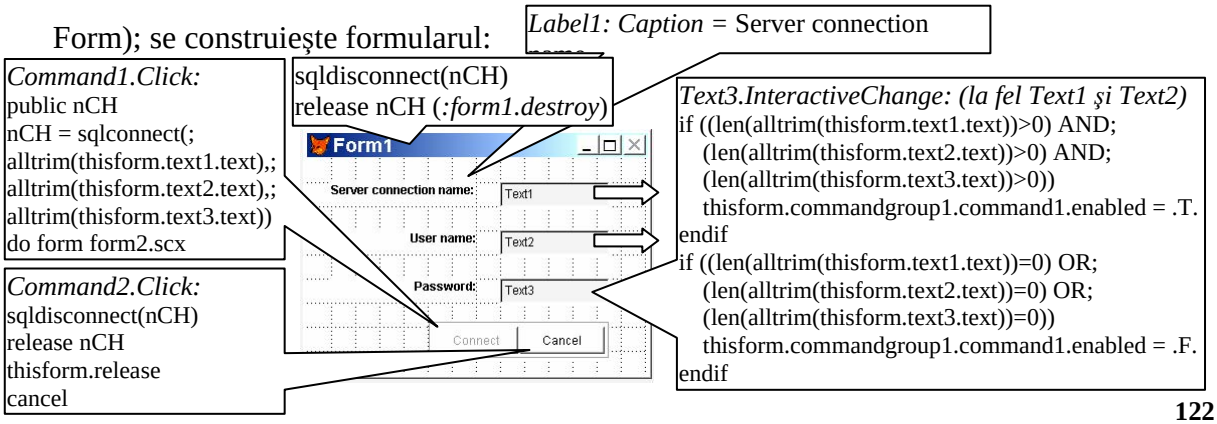
Conectarea la server se poate face cu comanda:

nConnectionHandle = SQLCONNECT('sqlremote','sa','secret')

unde *sqlremote* este numele conexiunii (*MySQL-lori*), *sa* este numele superuserului (*root*) iar *secret* este parola acestuia (********).

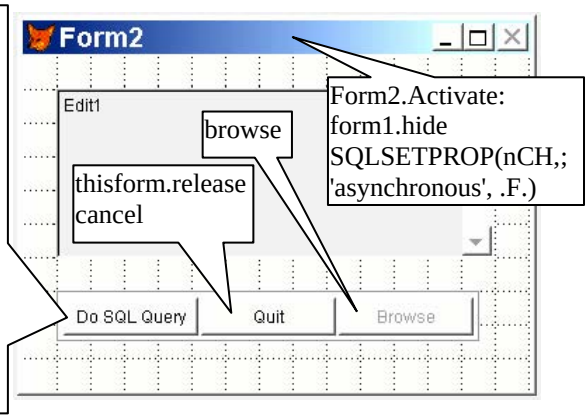
Pentru crearea unei interfețe grafice de administrare a serverului (*MySQL-lori*) în MVPF se pot urma paşii:

1. Se creează un nou proiect (New/Project/New File);
2. Se creează un nou formular (Project Manager – Proj1/Documents/Forms/New.../New Form); se construieşte formularul:



3. Se construieşte formularul form2:

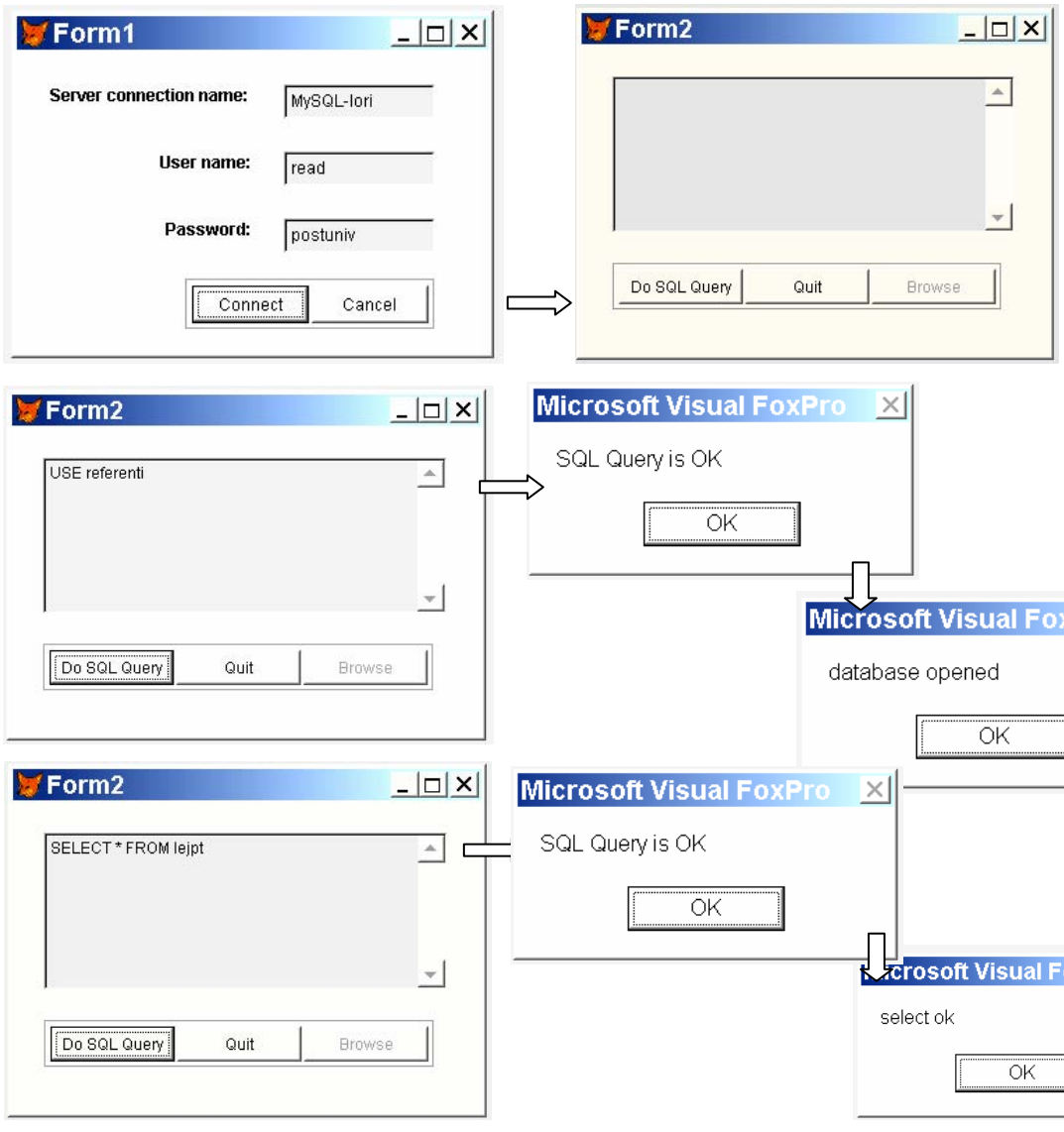
```
a = alltrim(thisform.edit1.text)
b = SQLEXEC(nCH, a, 'MyCursor')
if (b>0)
   messagebox("SQL Query is OK")
   if upper(substr(a,1,3))='USE'
messagebox("database opened")
thisform.commandgroup1.command3.enabled = .F.
   endif
   if upper(substr(a,1,6))='SELECT'
messagebox("select ok")
thisform.commandgroup1.command3.enabled = .T.
   endif
endif
```

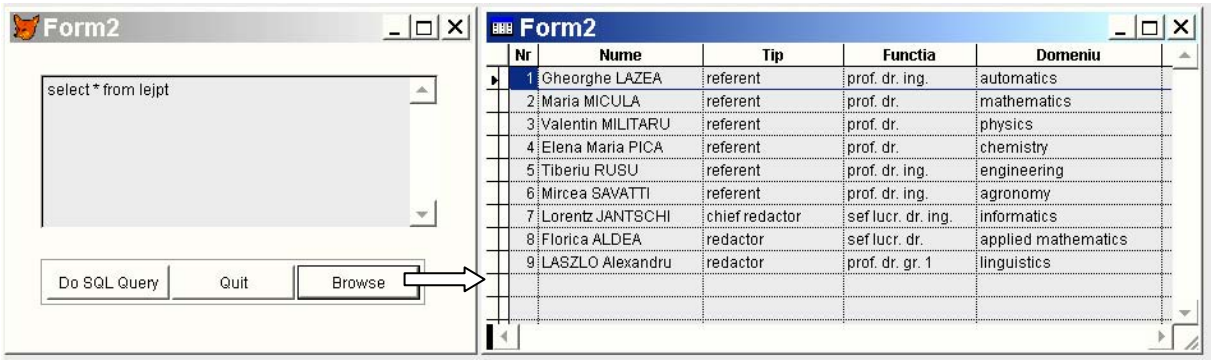


4. Se realizează programul principal (program1):

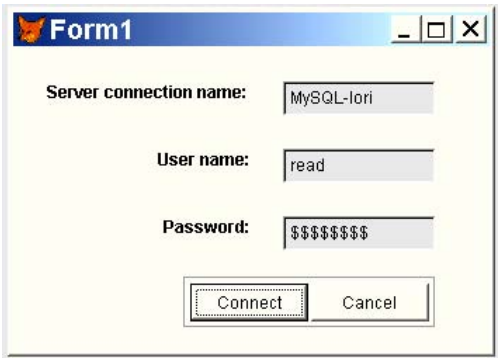
public nCH
do form form1.scx
read events

5. Execuția programului se face în modul următor (Program Manager – proj1/code/programs/program1/Run):





6. Se poate modifica formularul form1 astfel încât să nu afişeze parola la intrare. Se modifică proprietatea *PasswordChar* a casetei Text1: $;
7. Se execută din nou aplicația:



33. Comenzi SQL pentru accesul la un server de date

În aplicațiile client/server sunt utile următoarele comenzi VFP:

CURSORSETPROP(cProperty [, eExpression] [, cTableAlias | nWorkArea])

Exemplu 1.

Următorul exemplu demonstrează cum se poate deschide *optimistic* o tabelă cu ajutorul funcției CursorSetProp(). Funcția se poate folosi inclusiv pentru tabele locale. Se foloseşte tabela *institutii* din baza de date *universitati*. Variabila de mediu *MultiLocks* se va seta pe ON, ca o necesitate pentru accesul prin intermediul tampoanelor pe disk. Tabela *institutii* din baza de date *universitati* este deschisă, iar funcția CursorSetProp() este folosită pentru a seta modul de acces optimistic prin tampoane pe disk (*optimistic buffering mode*). Este afişat un mesaj informativ despre rezultatul operației.

```
CLOSE DATABASES
CLEAR
SET MULTILOCKS ON
OPEN DATABASE ('universitati')
USE institutii
lSuccess=CURSORSETPROP("Buffering", 5, "institutii")
IF lSuccess = .T.
  =MESSAGEBOX("Operation successful!",0,"Operation Status")
ELSE
  =MESSAGEBOX("Operation NOT successful!",0,"Operation Status")
ENDIF
```

Lista parametrilor poate accepta mai multe proprietăți urmate de valorile lor, aşa cum rezultă din următorul tabel:

proprietate (Property)	Valoarea (*eExpression*)
BatchUpdateCount (*)	Numărul de comenzi update trimise către sursa de date pentru tabelele tampon.
Buffering	1 – setează tamponul pentru înregistrări şi tabelă la Off; 2 – setează modul *pesimistic* pentru tamponul la înregistrări la On; 3 – setează modul *optimistic* pentru tamponul la înregistrări la On; 4 – setează modul *pesimistic* pentru tamponul la tabelă la On; 5 – setează modul *optimistic* pentru tamponul la tabelă la On;
CompareMemo	.T. face ca câmpurile *Memo*, *General* şi *Picture* să fie incluse în clauza *Where* pentru modificări; .F. – nu;
FetchAsNeeded	.F. – pentru vederi la distanță numărul înregistrărilor preluate e determinat de proprietatea *MaxRecord*; pentru vederile locale toate înregistrările sunt preluate; .T. – toate;
FetchMemo (*)	.T. – preia şi câmpurile memo; .F. – nu;
FetchSize (*)	Numărul de înregistrări citite progresiv de la distanță; implicit 100;

	-1 face ca să se preia întregul set;
KeyFieldList	Lista de câmpuri (specificate prin nume) care se doreşte a fi preluate în cursor, separate prin virgulă;
MaxRecords (*)	Numărul maxim de înregistrări preluate; -1: toate; 0: nici una (util pentru adăugări, când se doreşte execuția vederii, dar nu se doreşte citirea de date de la distanță ci doar scrierea);
Prepared	.T. atunci când se face doar o operație de pregătire a interogării (vezi aplicația de la capitolul 32, în care s-a folosit funcția SQLPrepare(), similară ca efect); se foloseşte apoi REQuery();
SendUpdates	.T. sau .F.
Tables	Lista tabelelor la distanță separate prin virgulă
UpdatableFieldList	Lista câmpurilor pentru Update (nume câmp în tabela la distanță, urmat de nume câmp în cursor);
UpdateNameList	Numele câmpurilor în cursor (pentru redenumiri);
UpdateType	1 – vechile date sunt modificate cu cele noi; 2 – ştergerea datelor vechi şi inserarea celor noi;
UseMemoSize (*)	Dimensiunea (în octeți) maximă a unui câmp de la care acesta când este preluat este stocat într-un câmp memo;
WhereType	Clauza Where pentru modificarea tabelelor la distanță; valori: 1 sau DB_KEY: doar câmpurile specificate în KeyFieldList; 2 sau DB_KEYANDUPDATABLE: câmpurile specificate în KeyFieldList şi cele modificabile; 3 sau DB_KEYANDMODIFIED: câmpurile specificate în KeyFieldList şi cele modificate; 4 sau DB_KEYANDTIMESTAMP: câmpurile specificate în KeyFieldList şi cele accesate;

(*) aceste proprietăți au ca destinație primără utilizarea lor la vederi la distanță; setarea acestora pentru vederile locale nu are efect.

Dicționar

pessimistic buffering: previne ca un utilizator în sistem multi-user să acceseze o anume înregistrare sau o tabelă când un alt utilizator face schimbări asupra ei; asigură cea mai mare securitate de mediu pentru modificarea înregistrărilor dar încetineşte operațiile cu utilizatorii;

optimistic buffering: o cale eficientă de a modifica înregistrări, deoarece blocarea acestora (lock) intervine doar atunci când acestea sunt scrise, şi face ca să se minimizeze timpul în care

un utilizator monopolizează sistemul într-un sistem multi-user; când se foloseşte blocarea la nivel de înregistrare sau tabelă în sistem multi-user, VFP aplică blocarea *optimistic*;

SQLCONNECT([DataSourceName, cUserID, cPassword | cConnectionName])

Exemplu 2.

```
STORE SQLCONNECT('MyFoxSQLNT', 'sa') TO gnConnHandle
IF gnConnHandle <= 0
  = MESSAGEBOX('Cannot make connection', 16, 'SQL Connect Error')
ELSE
  = MESSAGEBOX('Connection made', 48, 'SQL Connect Message)
  = SQLDISCONNECT(gnConnHandle)
ENDIF
```

SQLTABLES(nConnectionHandle [, cTableTypes] [, cCursorName])

Exemplul 3.

```
STORE SQLCONNECT('MyFoxSQLNT', 'sa') TO gnConnHandle
IF gnConnHandle < 0
  = MESSAGEBOX('Cannot make connection', 16, 'SQL Connect Error')
ELSE
  = MESSAGEBOX('Connection made', 48, 'SQL Connect Message')
  STORE SQLTABLES(gnConnHandle, 'TABLE', 'mycursor') TO nTables
  IF nTables = 1
    SELECT mycursor
    LIST
  ENDIF
ENDIF
```

SQLGETPROP(nConnectionHandle, cSetting)

Exemplul 4.

```
* Clear environment
CLOSE ALL
CLEAR ALL
CLEAR
* Display the Select Connection or Data Source dialog box
* to choose a connection
nHandle=SQLCONNECT()
* Test connection, report results
IF nHandle > 0
 cSource= SQLGETPROP(nHandle, "datasource")
 =MESSAGEBOX("Current Data Source = "+cSource,0,"Connection Results")
ELSE
 =MESSAGEBOX("Connection Error = " + ;
  ALLTRIM(STR(nHandle)),0,"Connection Results")
ENDIF
=SQLDISCONNECT(nHandle)
```

SQLSETPROP(nConnectionHandle, cSetting [, eExpression])

Exemplul 5.

```
* Clear environment
CLOSE ALL
CLEAR ALL
CLEAR
```

```
* Display the Select Connection or Datasource dialog box
* to choose a connection
nHandle=SQLCONNECT()
* Test connection, report results
IF nHandle > 0
  * Set PacketSize
  nSet=SQLSETPROP(nHandle, "PacketSize", 2048 )
  * Test setting and display results
  IF nSet > 0
    =MESSAGEBOX("PacketSize was set to 2048",0,"Connection Results")
  ELSE
    =MESSAGEBOX("Error setting PacketSize",0,"Connection Results")
  ENDIF
ELSE
  =MESSAGEBOX("No Connection",0,"Connection Results")
ENDIF
=SQLDISCONNECT(nHandle)
```

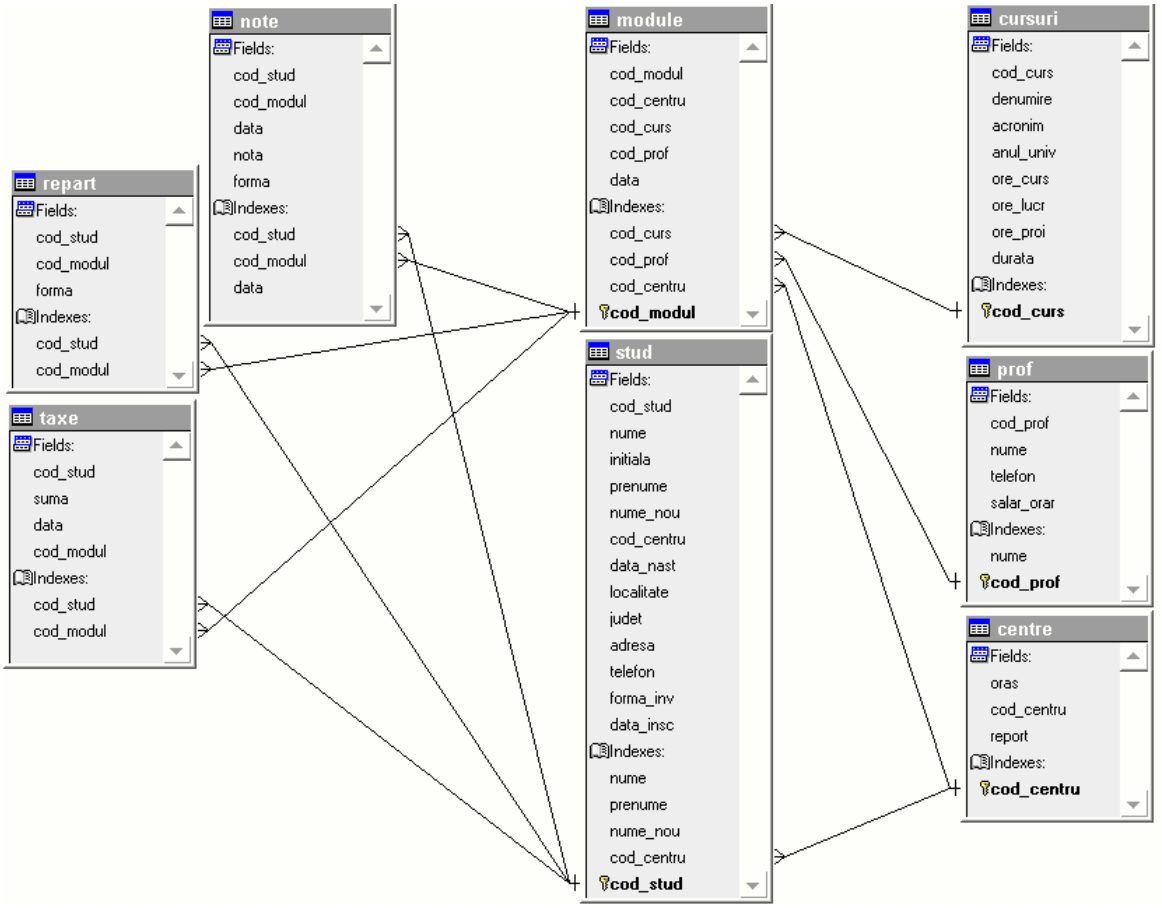
SQLMORERESULTS(nConnectionHandle)

Exemplul 6.

```
= SQLSETPROP(gnConnHandle, 'BatchMode', .F.)  && Individual result sets
= SQLEXEC(gnConnHandle, 'SELECT * FROM authors;
  SELECT * FROM titles')
= SQLMORERES(gnConnHandle)  && First result set
= SQLMORERES(gnConnHandle)  && Second result set
```

34. Aplicație exemplu

Se consideră baza de date cu cursanții Şcolii Academice Postuniversitare de Informatică Aplicată şi Programare cu următoarea structură:

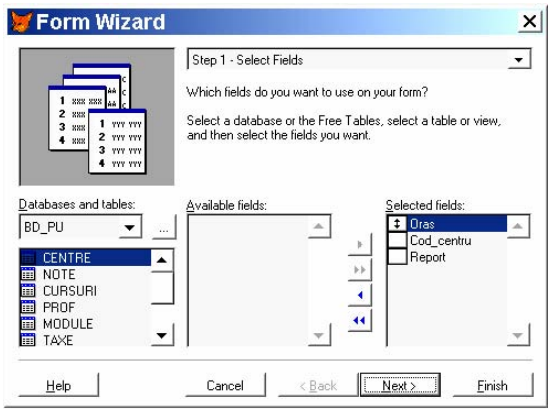


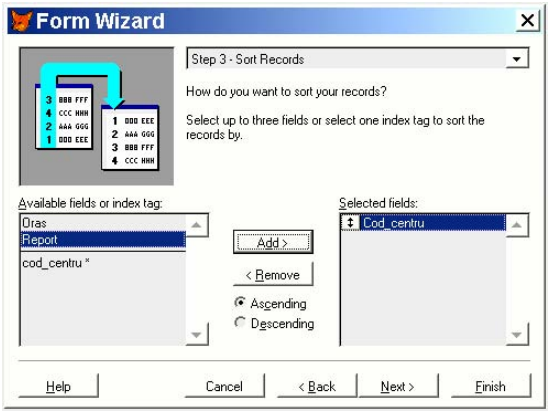
Obiectivul este crearea interfeței de exploatare a bazei de date. după crearea bazei de date, a tabelelor şi stabilirea relațiilor între tabele, conținutul directorului cu baza de date este:

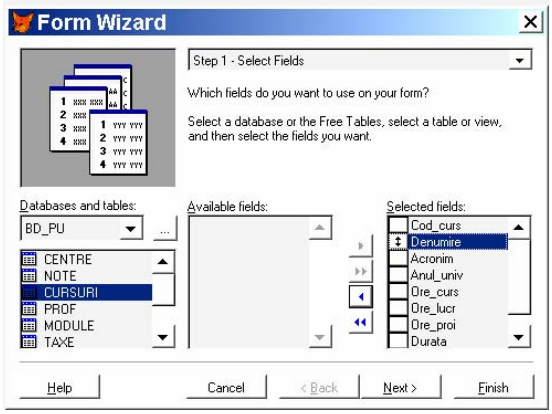
Name	Ext	Size
[..]		<DIR>
BD_PU	dbc	23,983
Bd_pu	dct	4,608
Bd_pu	dcx	10,240
centre	CDX	3,072
centre	dbf	752
cursuri	CDX	3,072
cursuri	dbf	2,354
Module	cdx	9,216
module	dbf	1,790
Note	cdx	26,624
note	dbf	40,342
Prof	cdx	6,144
prof	dbf	1,574
repart	CDX	16,896
repart	dbf	20,048
Stud	cdx	41,472
stud	dbf	140,554
stud	FPT	28,234
Taxe	cdx	35,328
taxe	dbf	118,628

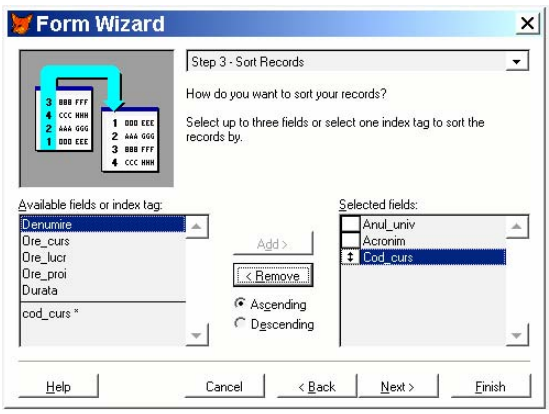
Se creează acum formularele şi rapoartele:

1. Formulare pentru adăugarea şi modificarea unui centru, şi curs; tabelele nu au chei străine deci formularele sunt pe baza unei singure tabele; se merge din meniu pe calea: New/Form/Wizard/Form Wizard; se generează astfel cele 3 formulare:

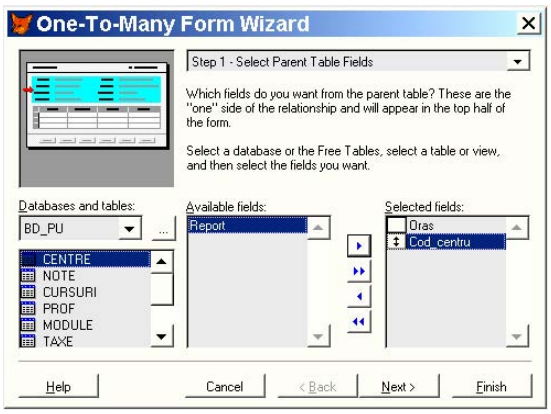


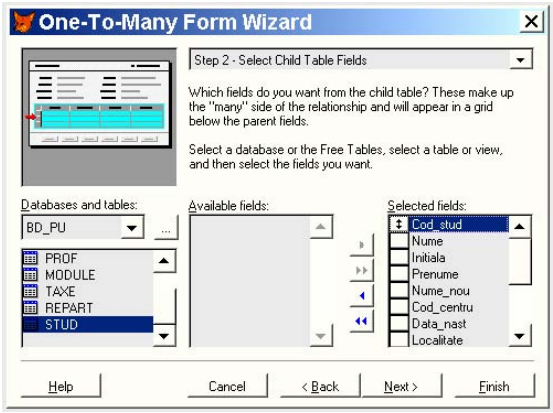


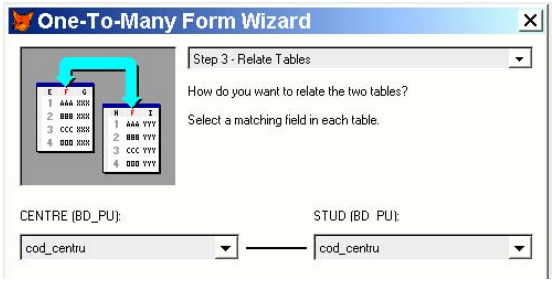


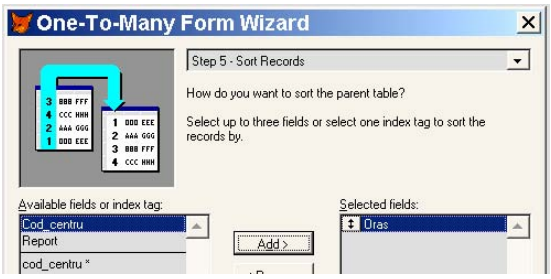


2. Se salvează aceste formulare în directorul cu baza de date. Se creează formulare pentru adăugarea/modificarea unui student; tabela are cheie străină din tabela centre; se foloseşte wizard-ul pentru aceasta: New/Form/Wizard/One-to-Many Form Wizard;

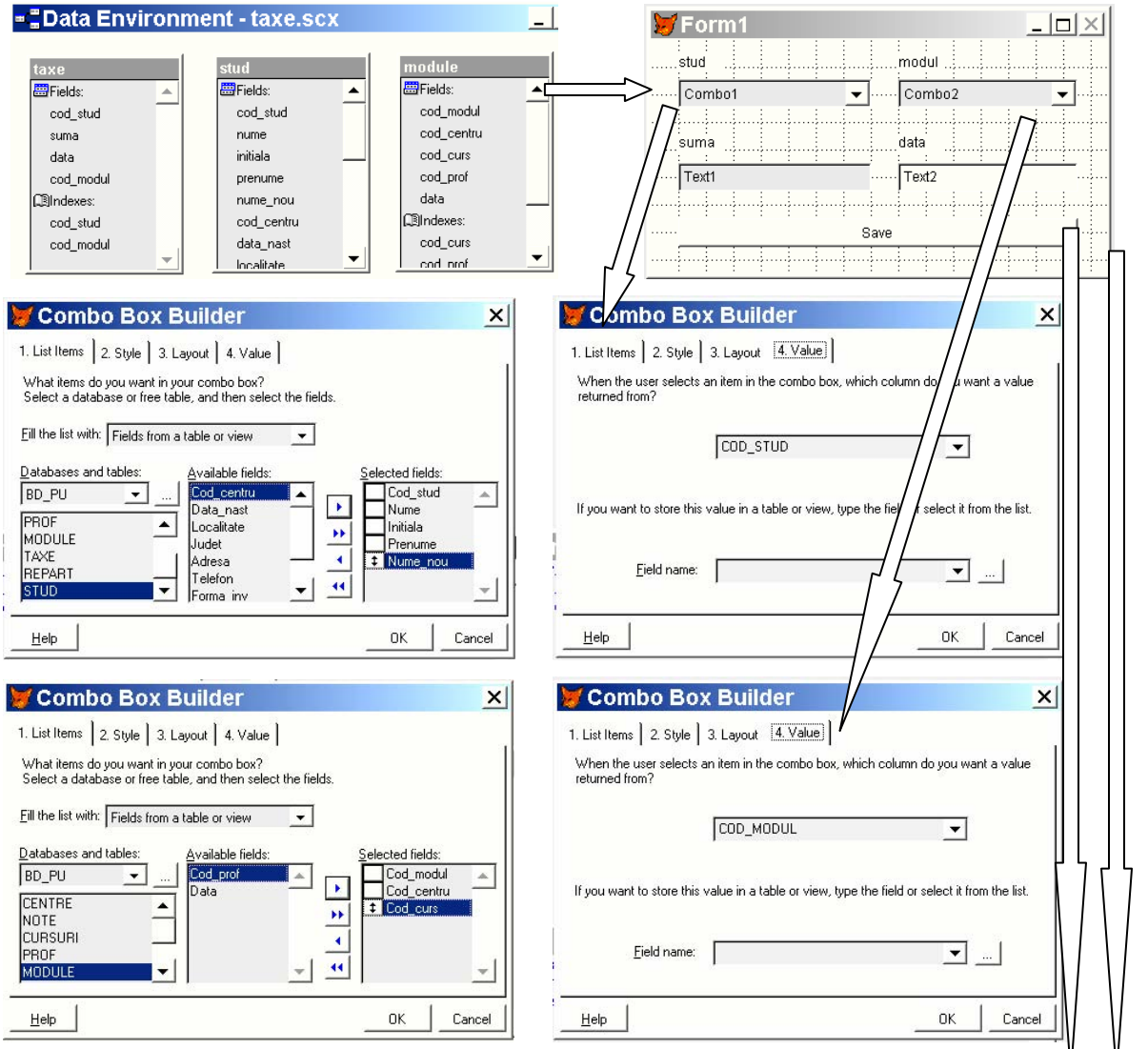








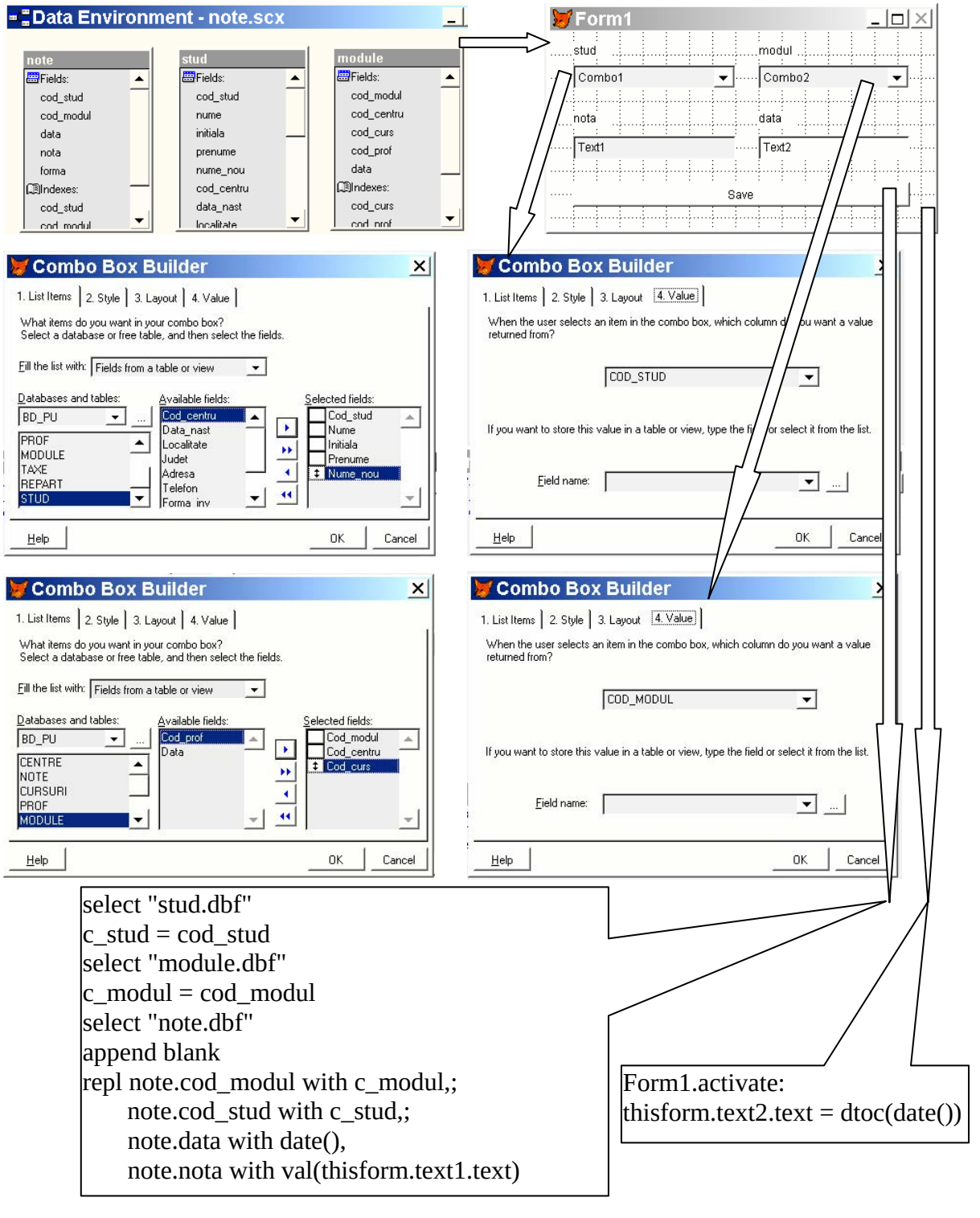
3. Tabelele note, repart şi taxe sunt tabele ce implementează relații (m,n) între tabelele module şi stud; adăugarea unei note presupune selecția studentului din tabela de studenți, selectarea modulului din tabela de module şi introducerea datei, formei şi notei; adăugarea unei repartiții presupune selecția studentului din tabela de studenți, selecția modulului din tabela de module şi introducerea formei; adăugarea unei taxe presupune selecția studentului din tabela stud, selecția modulului din tabela de module şi introducerea sumei şi datei; se merge pe calea New/Form/New File;




```
select "stud.dbf"
c_stud = cod_stud
select "module.dbf"
c_modul = cod_modul
select "taxe.dbf"
append blank
repl taxe.cod_modul with c_modul,;
     taxe.cod_stud with c_stud,;
     taxe.data with date(),
     taxe.suma with val(thisform.text1.text)
```

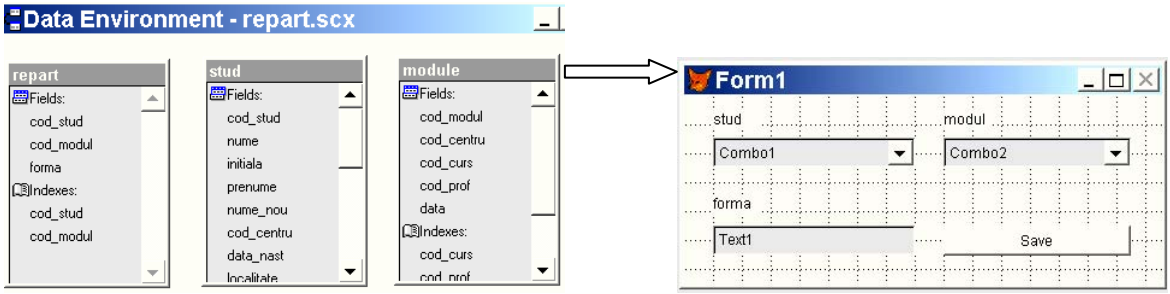
```
Form1.activate:
thisform.text2.text = dtoc(date())
```

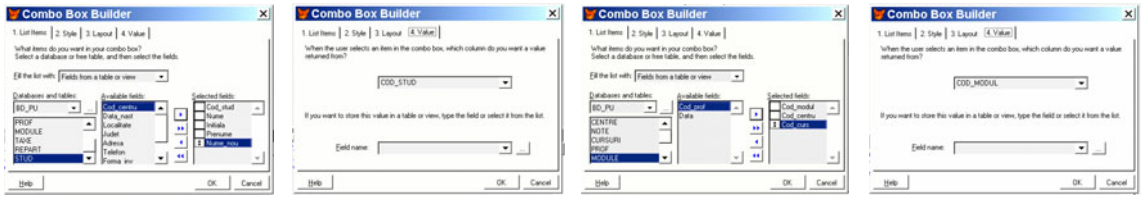
4. La fel se procedează şi pentru celelalte două formulare:




```
select "stud.dbf"
c_stud = cod_stud
select "module.dbf"
c_modul = cod_modul
select "note.dbf"
append blank
repl note.cod_modul with c_modul,;
     note.cod_stud with c_stud,;
     note.data with date(),
     note.nota with val(thisform.text1.text)
```

```
Form1.activate:
thisform.text2.text = dtoc(date())
```





Command1.Caption: Save;

Command1.Click:

select "stud.dbf"

c_stud = cod_stud

select "module.dbf"

c_modul = cod_modul

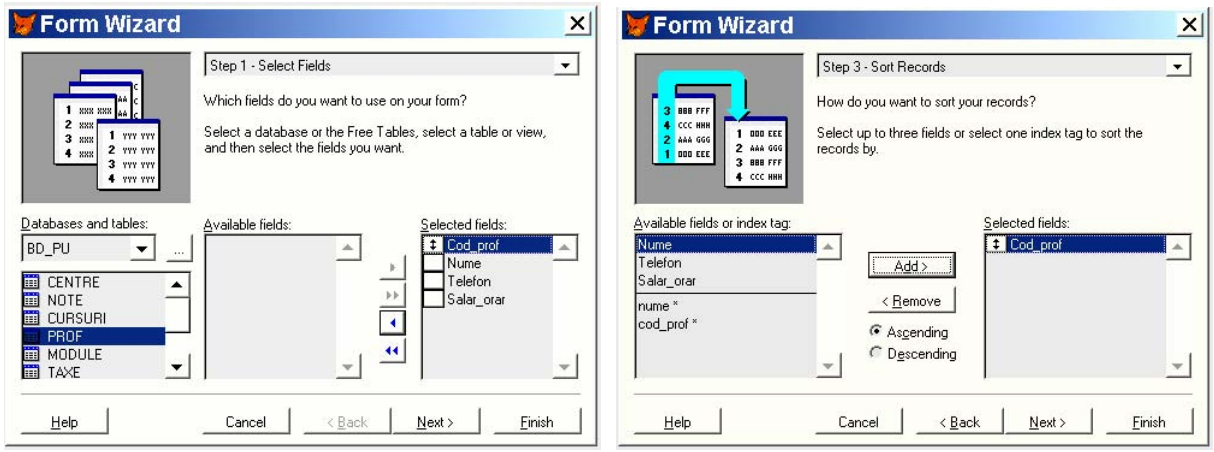
select "repart.dbf"

append blank

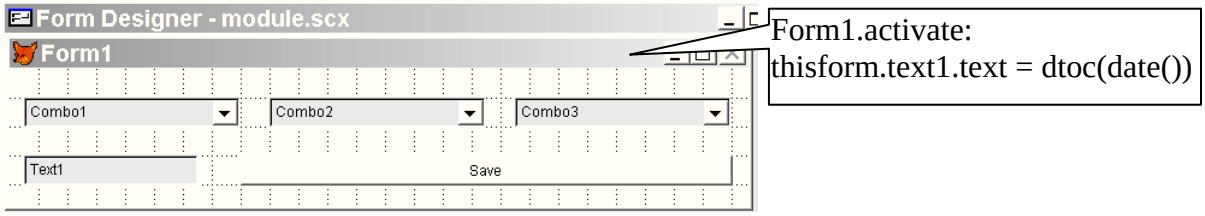
repl repart.cod_modul with c_modul, repart.cod_stud with c_stud,;

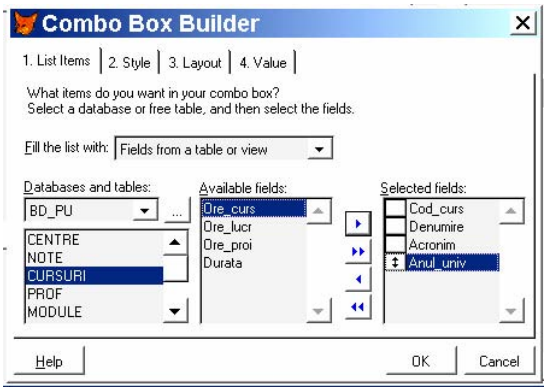
repart.forma with alltrim(thisform.text1.text)

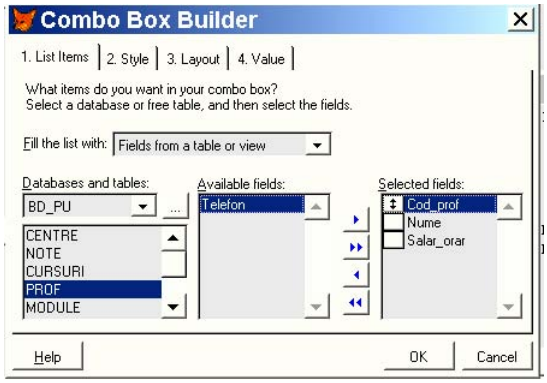
5. Formularul pentru introducerea datelor în tabelele prof şi module poate fi făcut cu ajutorul wizard-ului: New/Form/Wizard/One-to-Many Form Wizard, la fel ca la formularul stud, însă s-ar pierde o legătură; se poate face în forma anterioară, se crează un formular pentru prof:

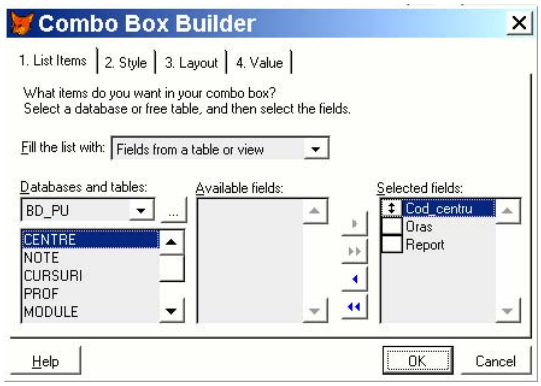


după care se creează un formular pentru modul:



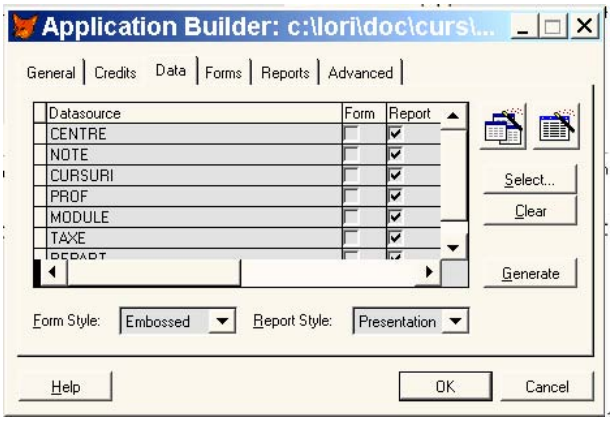


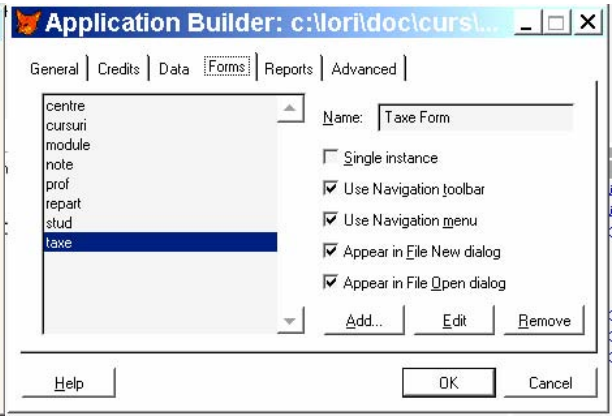


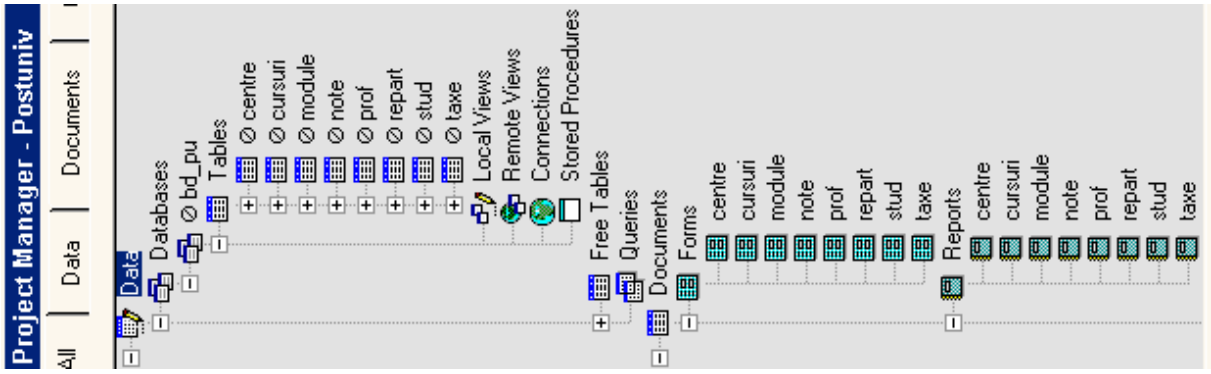



```
select "cursuri.dbf"
cod_c = cursuri.cod_curs
select "prof.dbf"
cod_p = prof.cod_prof
select "centre.dbf"
cod_n = centre.cod_centru
select module
append blank &&are functie autoincrement
replace module.cod_centru with cod_n,;
        module.cod_curs with cod_c,;
        module.cod_prof with cod_p,;
        module.data with date()
```

6. Se creează rapoartele cu ajutorul lui Report Wizard;
7. Se integrează aplicațiile într-n proiect cu ajutorul lui Application Wizard: New/Project/Wizard:







35. Documentarea aplicațiilor windows cu Microsoft HTML Help

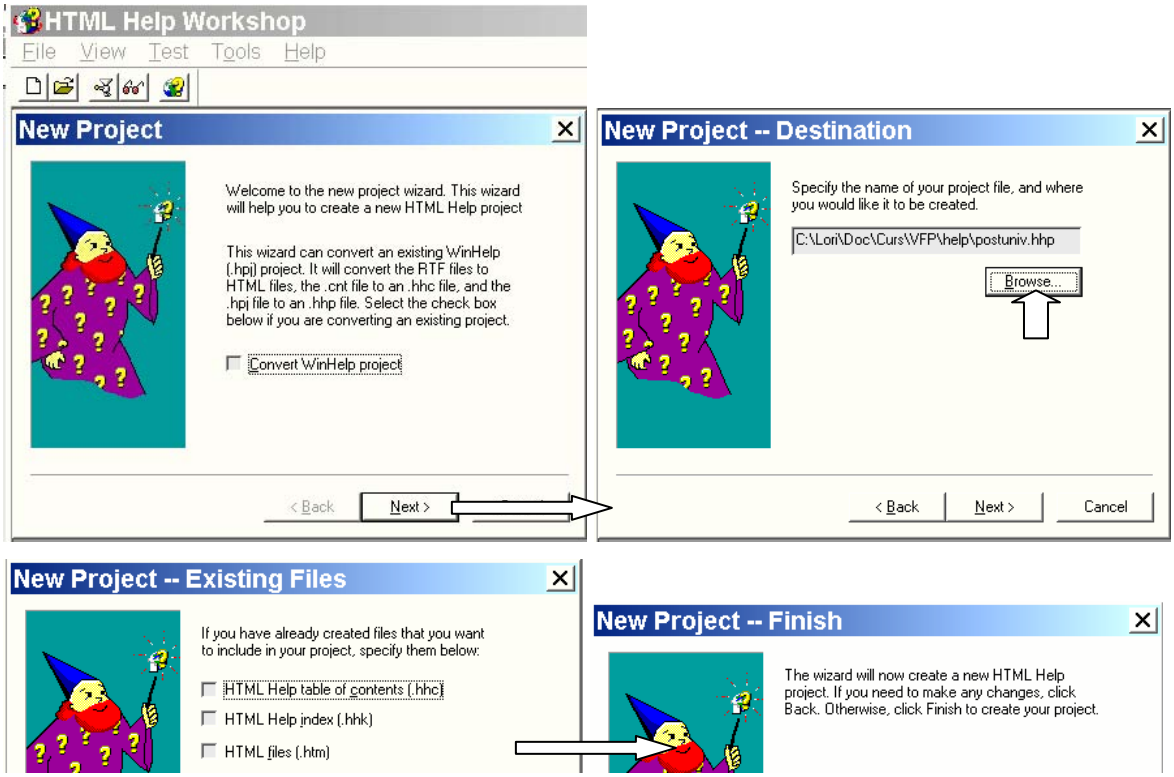
Microsoft® HTML Help constă dintr-un program de vizualizare online *Help Viewer* care foloseşte componentele lui *Microsoft Internet Explorer* pentru a afişa conținutul help-ului. Suportă HTML, ActiveX®, Java™, limbaje de scriptare (JScript®, and Microsoft Visual Basic® Scripting Edition) şi imagini în format HTML (.jpeg, .gif, şi .png).

HTML Help Workshop este instrumentul cu care se pot crea şi exploata proiectele help şi fişierele auxiliare.

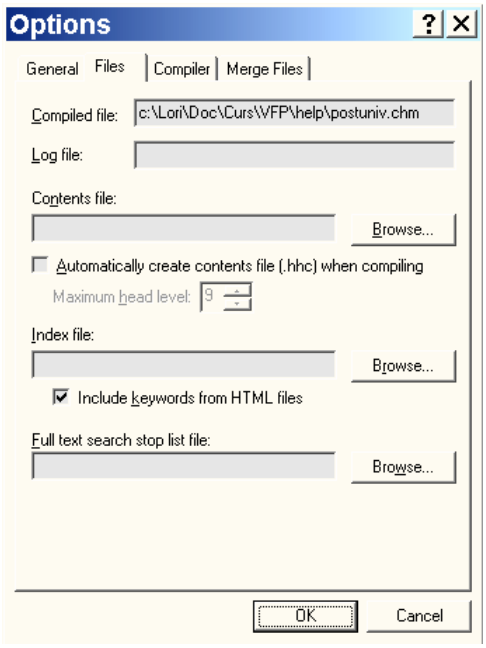
HTML Help project este un fişier text cu extensia *.hhp* care organizează toate elementele unui sistem help. El conține legăturile către toate topicele HTML (fişiere cu extensiile *.html* şi *.htm*) , imagini (*.jpeg, .gif, .png*), index (*.hhk*), şi cuprins (*.hhc*). *HTML Help Workshop* combină apoi toate aceste fişiere pentru a genera un singur fişier cu extensia *.chm*.

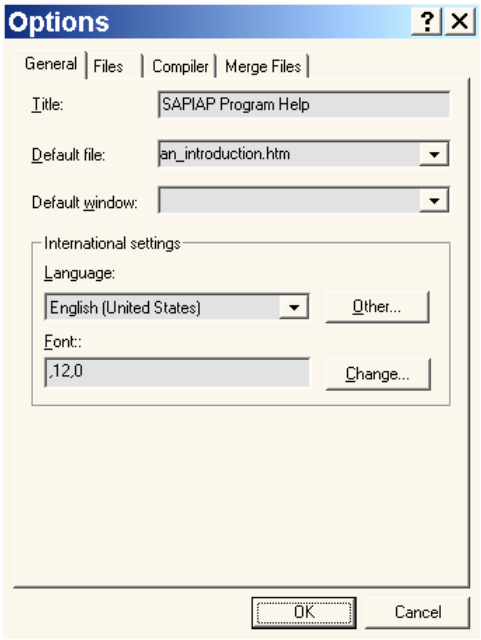
36. Crearea unui nou fişier help (.chm) cu HTML Help Workshop

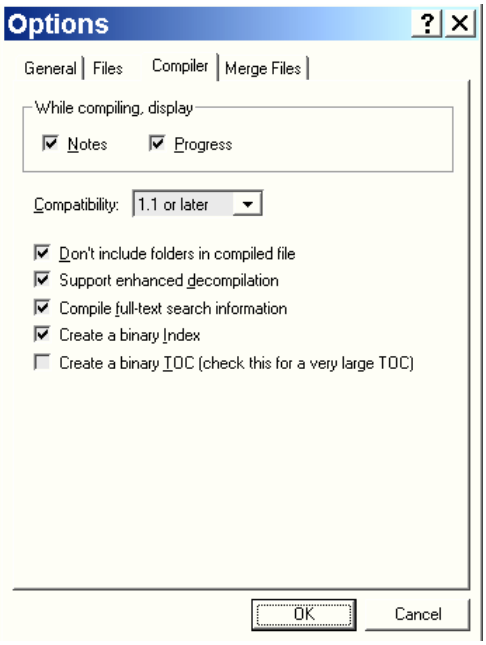
HTML Help Workshop este un produs free Microsoft şi poate fi descărcat de pe situl Microsoft. Se instalează în sistem (sunt necesare privilegii de administrator) după care se încarcă în execuţie pe calea *Start/Programs/HTML Help Workshop/HTML Help Workshop*. Se creează un nou help (File/New). Dacă au fost deja create fişiere componente ale proiectului, se pot include în el:

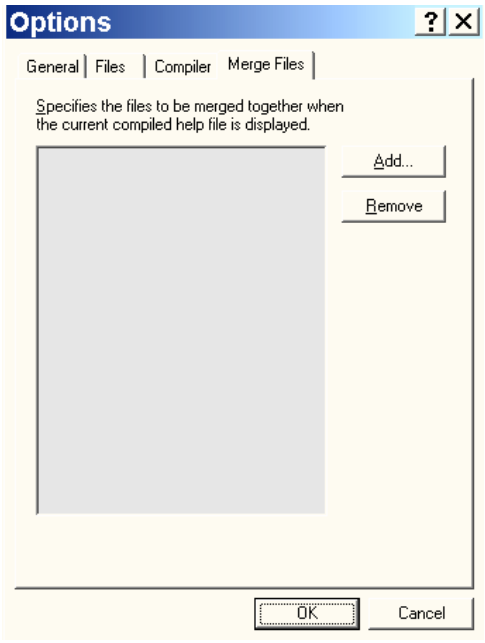


După generarea proiectului, se pot defini opţiunile acestuia, în care se includ topicele, aşa cum rezultă din figurile următoare.









Până în acest moment, conținutul fișierului *postuniv.hhp* este:

```
[OPTIONS]
Auto Index=Yes
Compatibility=1.1 or later
Compiled file=postuniv.chm
Default Font=,12,0
Default topic=data\an_introduction.htm
Display compile progress=Yes
Enhanced decompilation=Yes
Flat=Yes
Full-text search=Yes
Language=0x409 English (United States)
Title=SAPIAP Program Help
[FILES]
data\t_stud.htm
data\an_introduction.htm
f_centre.htm
f_cursuri.htm
f_module.htm
f_note.htm
f_prof.htm
f_stud.htm
r_centre.htm
r_cursuri.htm
r_module.htm
r_note.htm
r_prof.htm
r_stud.htm
t_centre.htm
t_cursuri.htm
t_module.htm
t_note.htm
t_prof.htm
a_documentation.htm
```

[INFOTYPES]

Fişierele *.htm sunt fişiere (D)HTML standard şi pot fi generate cu orice program creator de pagini web (Netscape Composer, Microsoft Word, FrontPage).

Iată conținutul fişierelor an_introduction.htm, a_documentation.htm şi f_centre.htm:

an_introduction.htm

```
<html><head><title>SAPIAP Help</title></head><body>
Baza de date a Scolii Academice Postuniversitare de Informatica Aplicata si Programare este
in urmatoarea structura:<BR>
<img src="bd_all.jpg"><BR>
<BR>Vezi si:<BR>
tabela <A HRef = "t_centre.htm">centre</A><BR>
tabela <A HRef = "t_stud.htm">studenti</A><BR>
tabela <A HRef = "t_cursuri.htm">cursuri</A><BR>
tabela <A HRef = "t_prof.htm">profesori</A><BR>
tabela <A HRef = "t_module.htm">module</A><BR>
tabela <A HRef = "t_note.htm">note</A><BR>
informatiile despre realizarea <A HRef = "a_documentation.htm">helpului</A>
</body></html>
```

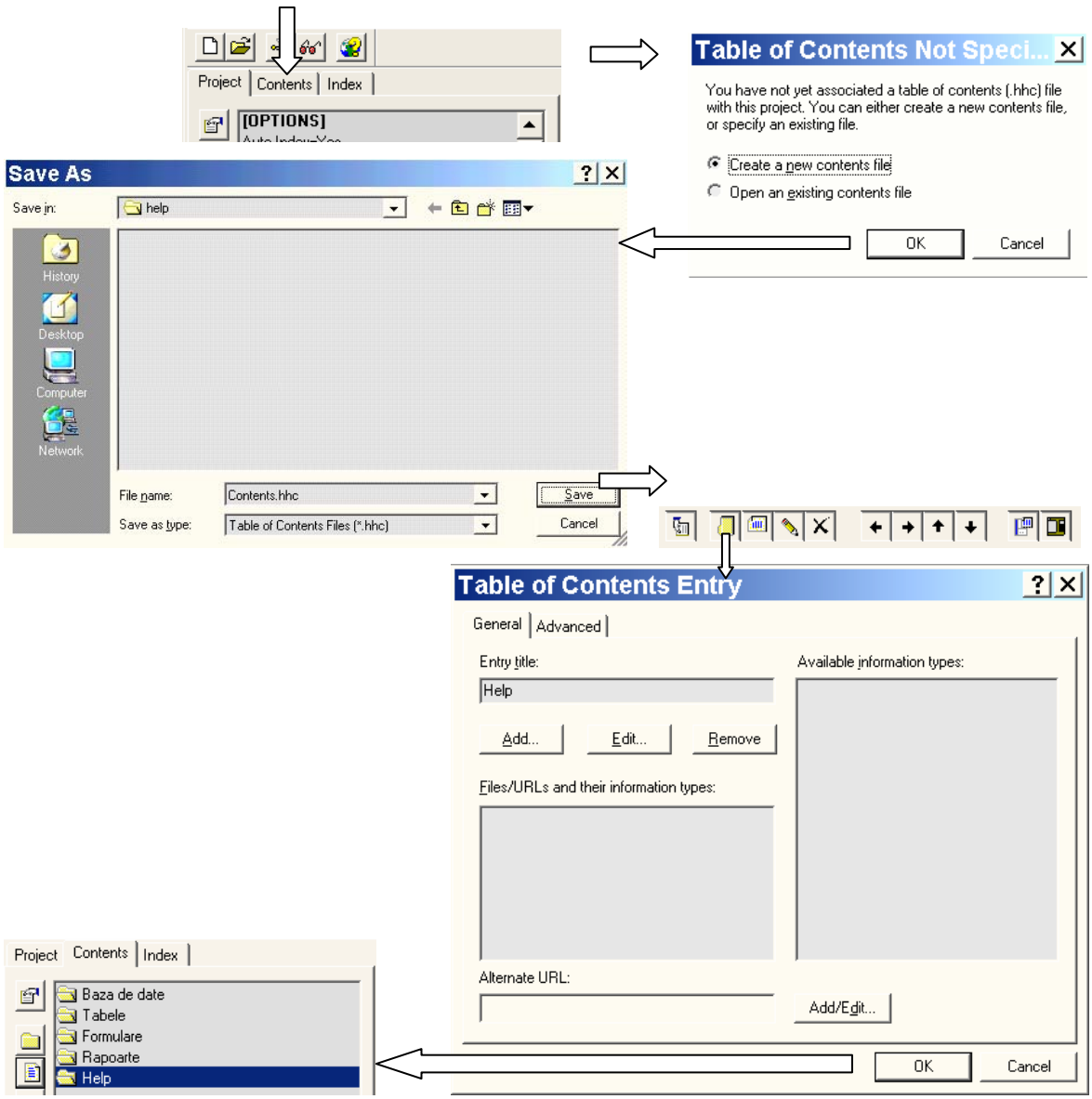
a_documentation.htm

```
<html><head><title>SAPIAP Help</title></head><body>
Fisierele incluse in topicul acestui help sunt:<BR>
<img src="z_doc.jpg"><BR>
</body></html>
```

f_centre.htm

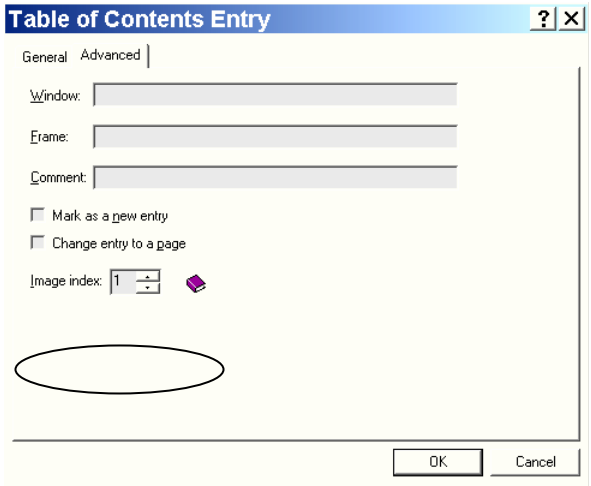
```
<html><head><title>Fromularul centre</title></head><body>
<BR>Formularul Centre ne permite sa prelucreaza informatiile din tabela de centre de
invatamant:<BR>
<img src="form_centre.jpg" align=center><BR>
<BR>Parcurgerea se realizeaza cu butoanele:<BR>
<ul><li>top: inceputul tabelei<li>prev: inregistrarea anterioara
<li>next: urmatoarea inregistrare<li>bottom: ultima inregistrare</ul>
<img src="find_centre.jpg" align=center><BR>
<BR>Cautarea informatiilor se face cu butonul Find:<BR>
<img src="result_centre.jpg" align=center><BR>
<BR>Tiparirea informatiilor se face cu butonul print:<BR>
<img src="result_centre.jpg" align=center><BR>
<BR>Adaugarea unui nou centru se face cu butonul Add:<BR>
<img src="add_centre.jpg" align=center><BR>
<BR>Editarea informatiilor despre un centru se face cu butonul Edit:<BR>
<img src="edit_centre.jpg" align=center><BR>
<BR>Stergerea unui centru se face cu butonul Delete iar iesirea din formular se face cu
butonul Exit.<BR>
Vezi si:<BR>
tabela <A HRef = "t_centre.htm">centre</A><BR>
raportul <A HRef = "r_centre.htm">centre</A><BR>
pagina de<A HRef = "an_introduction.htm">inceput</A><BR>
</body></html>
```

Se poate acum crea cuprinsul, începând cu repertoarul acestuia:



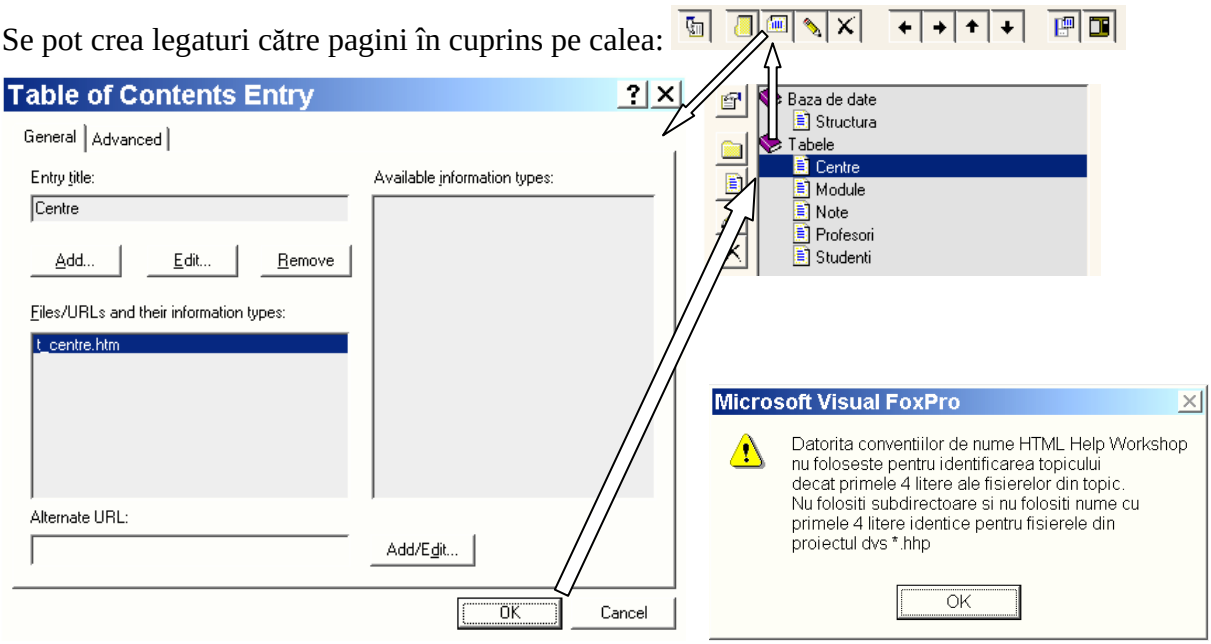
Se poate schimba iconița implicită asociată unei intrări pe calea:

Contents/Edit Selection/Table of Contents Entry/Advanced. Legenda este alăturată:

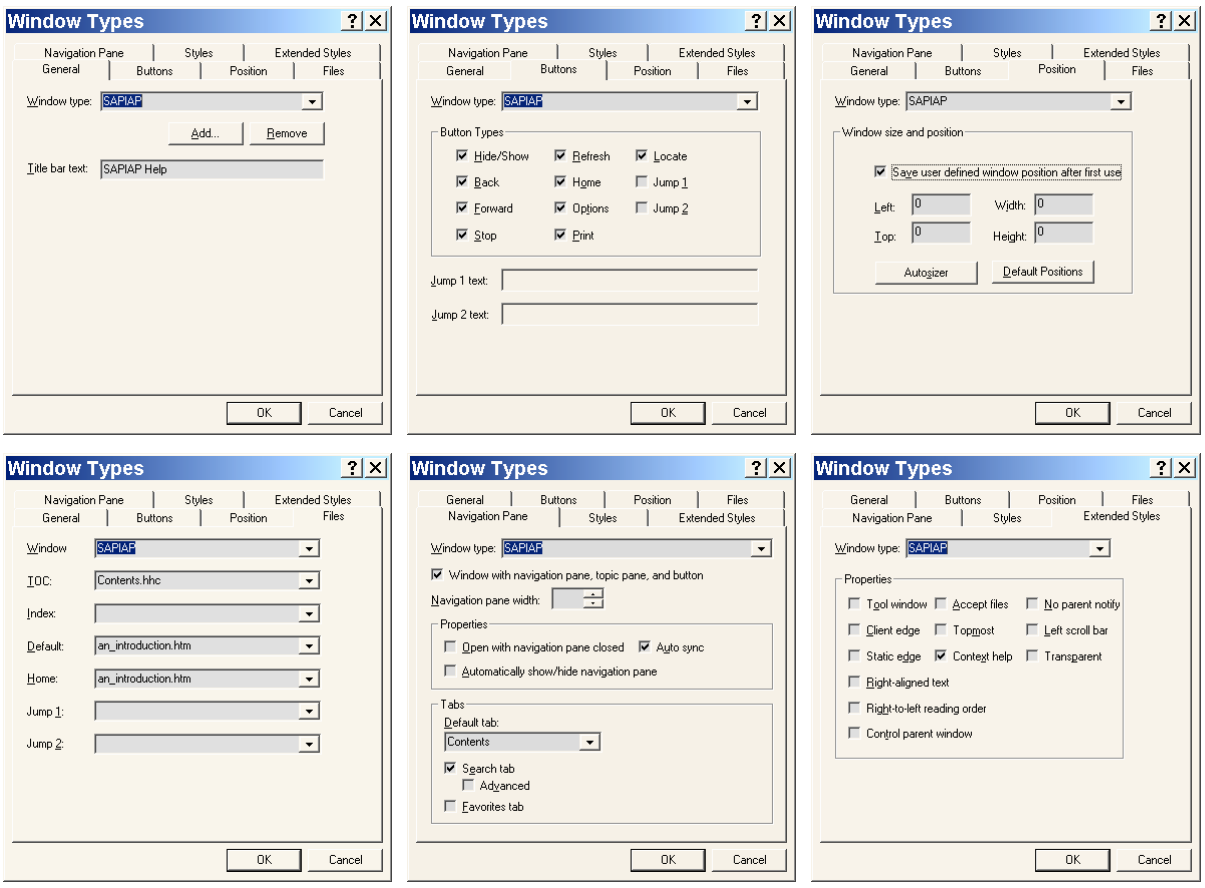


1		15		29	
2		16		30	
3		17		31	
4		18		32	
5		19		33	
6		20		34	
7		21		35	
8		22		36	
9		23		37	
10		24		38	
11		25		39	
12		26		40	
13		27		41	
14		28		42	

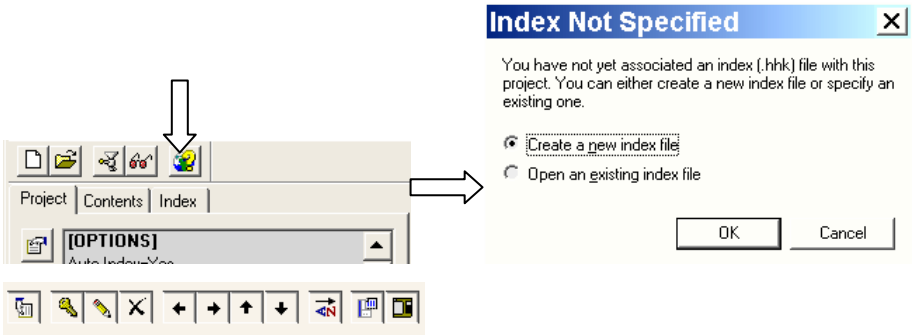
Se pot crea legaturi către pagini în cuprins pe calea:



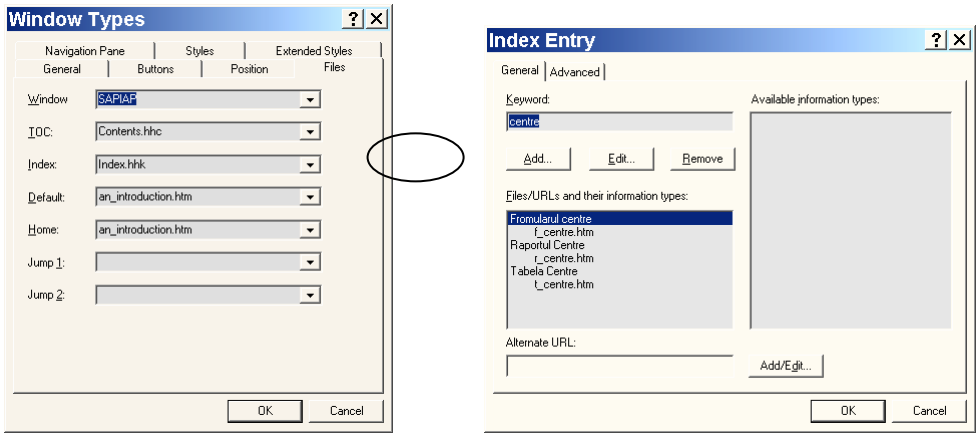
În mod implicit se generează o fereastră de help cu puține controale înglobate. Se poate însă crea o fereastră definită de utilizator. Se merge pe calea:



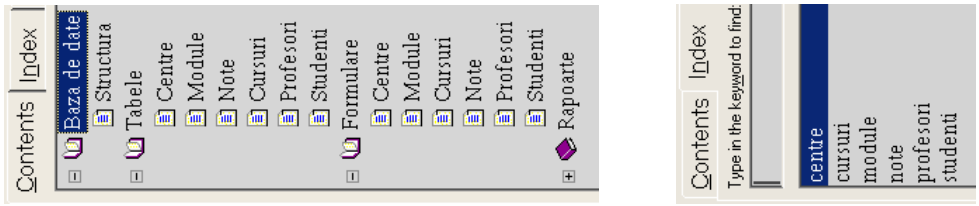
Se poate acum crea indexul:



Se adaugă opțiunea de index în Widow Types (*Add/Modify Window definitions*). Se adaugă apoi noi intrări în index pe aceeași cale ca pentru paginile din cuprins:



Compilarea helpului are ca rezultat fișierul postuniv.chm. Încărcarea acestuia în execuție duce la:



Se poate acum regenera aplicația cu *Application Builder* când se specifică fișierul help *postuniv.chm* în pagina *Advanced* a acestuia.

Lorentz JÄNTSCHI, Mădălina Ana VĂLEANU, Sorana Daniela BOLBOACĂ

37. Capitole speciale de baze de date
(securitate, coerență, restricții, tranzacții, concurență)

Alte tipuri de baze de date

O ***bază de date distribuită*** (BDD) este reprezentată de o mulțime de date corelate logic, mai exact o colecție de baze de date împreună cu colecția de relații dintre ele, baze de date care rezidă pe mașini diferite, ce lucrează în sisteme de operare de cele mai multe ori diferite, interconectate prin diferite rețele de comunicații. Corelarea (integrarea) logică constă în faptul că, din punct de vedere al utilizatorului, o BDD reprezintă o singură bază de date, o singură structură conceptuală globală reprezentând integrarea logică a mulțimilor de date. Astfel, repartiția fizică a datelor este transparentă pentru utilizator.

Din cele ce s-au menționat mai sus se poate trage concluzia că fiecare calculator împreună cu baza de date locală constituie *o stație* sau *un nod* al bazei de date distribuite, iar calculatoarele corespunzătoare acestor noduri sunt conectate între ele prin sistemul de comunicație al rețelei. În cadrul operațiilor obişnuite, cererile de acces ale aplicațiilor unei stații (unui nod) necesită numai accesul la baza de date locală. Aceste aplicații executate în întregime de către calculatorul stației (nodului) sunt numite *aplicații locale.* Ceea ce deosebește o mulțime oarecare de baze de date locale de o bază de date distribuită este existența unor aplicații care accesează date din mai multe stații. Aceste aplicații sunt numite *aplicații globale* sau *aplicații distribuite.*

Gestiunea unei BDD necesită ca în fiecare stație de lucru (în a cărei memorie externă rezidă o parte din baza de date distribuită) să existe:

- *sistemul de gestiune al bazelor de date locale (SGBDL),* care trebuie să fie un SGBD standard şi trebuie să cuprindă propriul dicționar pentru datele din baza de date locală;
- *componenta de comunicație (CC),* care trebuie să realizeze toate legăturile în cadrul rețelei de date locală ;
- *dicționarul global de date (DGD),* care trebuie să conțină informații despre baza de date distribuită referitoare la localizarea, structura, disponibilitatea şi modul de utilizare a datelor;
- *sistemul de gestiune a bazei de date distribuite (SGBDD),* care asigură interfața între baza de date distribuită şi utilizatorii acesteia;
- majoritatea stațiilor de lucru (nodurilor) au şi un *administrator al bazei de date distribuite.*

O ***bază de date distribuită omogen*** este compusă din baze de date locale administrate de acelaşi SGBD şi se obține, de obicei, prin diviziunea unei baze de date mari în baze de date

locale, SGBDD-ul care o gestionează fiind considerat, evident, omogen. La primirea unei cereri, stația de la care a fost lansată cererea devine stație coordonatoare şi este responsabilă pentru determinarea numelui entității şi pentru accesarea catalogului.

Componenta de gestiune a tranzacțiilor elaborează, de obicei, un plan global care include atât module de acces la distanță, cât şi module de acces local. Stațiile cooperante păstrează autonomia completă asupra propriilor date (locale).

Baza ***de date distribuită eterogenă*** se obține prin integrarea într-o bază de date unică a mai multor baze administrate de SGBD-uri diferite. Există două nivele de eterogenitate. O situație în care bazele de date sunt de acelaşi tip (de exemplu, relațional) dar sunt gestionate de SGBD-uri diferite şi alta în care bazele de date sunt de tipuri diferite (de exemplu, relațional, ierarhic, rețea) şi, evident, sunt gestionate de SGBD-uri diferite. Problema este deosebit de ambițioasă şi, evident, este dificil de rezolvat în totalitate.

Bazele de date federale (BDF), numite şi ***multibaze de date***, permit interoperabilitatea în baze de date autonome şi eterogene cu ajutorul unui limbaj multibază şi nu cu ajutorul unei scheme globale. Prin această abordare se permite asigurarea unei autonomii totale a diferitelor baze de date componente ale BDF şi, de asemenea, se oferă posibilitatea administrării, gestionării şi manipulării independente a acestor baze. Absența schemei locale implică tolerarea conflictelor semantice.

Percepția utilizatorului privind o baza de date federală nu este aceea de bază de date globală, ci aceea de mulțime de baze de date distincte interoperabile. Noțiunea de stație este înlocuită cu cea de bază de date, deoarece BD care fac parte dintr-o federație (BDF) nu se află obligatoriu pe stații diferite. Astfel, o arhitectură pe trei nivele a schemei unei baze de date federale (BDF) conține:

1) *nivelul privat* care conține schemele interne şi conceptuale ale fiecărei baze de date BDF;
2) *nivelul multibază (schema exportată)* descrie o bază de date existentă în federație ca pe un subansamblu al acesteia. Exisă, tot la acest nivel, o *schemă de dependențe interbaze* care face legătura între schemele exportate ale diferitelor baze de date din baza de date federală;
3) *Nivelul extern(schema externă)* permite manipularea datelor din diferite baze componente.

Bazele de date paralele (BDP) încearcă îmbunătățirea performanțelor, fiabilității şi costului gestiunii datelor, bazându-se pe paralelismul oferit de recentele arhitecturi multiprocesor. Există variantele:

- *arhitectura cu memorie comună,* în care fiecare procesor poate avea acces, în egală măsură, la fiecare modul de memorie centrală şi la fiecare unitate de disc, prin rețelele de interconexiune;

- *arhitectura cu memorie privată* este formată dintr-o mulțime de noduri care pot coopera datorită unei rețele de interconexiuni. Fiecare nod conține unul sau mai multe procesoare care au acces exclusiv la modulele memoriei interne şi la unitățile de discuri proprii nodului respective.

Limbajele de programare convenționale ce permit programarea obiectuală permit crearea, manipularea şi distrugerea obiectelor în memoria internă. Aceasta înseamnă că durata de viață a unui astfel de *obiect (tranzitoriu)* nu depăşeşte durata de viață a programului care l-a creat. O soluție în prelungirea vieții obiectelor este copierea lor în fişiere, pentru a putea fi citite şi manipulate ulterior, această abordare fiind evident o formă primitivă de persistență a obiectelor. Din păcate ea nu e transparentă, trebuind gestionată explicit, iar programarea operațiilor de gestionare este delicată, trebuind transformați pointerii (adrese virtuale) în adrese disc. În această situație, accesul într-un set de mai multe obiecte este neperformant, neajuns la care se adaugă şi problemelor clasice referitoare la gestionarea fişierelor. Abordarea cu ***baze de date de orientare obiect*** aduce o soluție nouă la problema gertiunii transparente a ***obiectelor persistente***, adică acelor obiecte care, stocate în baza de date, au o durată de viață evident mai mare decât programul prin care au fost create obiectele respective. Ideea de bază constă în a combina conceptele obiect cu concepte referitoare la bazele de date.

Prima motivație pentru bazele de date orientate obiect este nevoia de acces rapid la obiecte persistente prin programe scrise în limbaje cu facilități în programarea orientată pe obiecte.

A doua motivație este legată de noile cerințe din tehnologiile aplicațiilor complexe cum ar fi CAD (Computer Aided Design), CAM (Computer Aided Manufacturing), CASE (Computer Aided Software Engineering), CAE (Computer Aided Engineering), aplicații grafice, multimedia, ş.a.

Schema unei BDOO trebuie să cuprindă, deci, definițiile structurale (atribute şi tipuri) şi comportamentale (metode) ale obiectelor, precizând toate clasele definite pentru o aplicație. Definițiile claselor includ *moştenirea, relațiile de înrudire (supraclase-subclase) şi relațiile structurale dintre clase (asocierile).* Asocierile dintre clase sunt necesare pentru a cuprinde restricțiile de integritate referențială la nivel de clasă, o asociere între două clase fiind reprezentată natural prin două legături, una fiind inversa celeilalte. Schema bazei de date trebuie să specifice, deci, datele care ar putea fi materializate şi asocierile dintre ele, fiind definită de către proiectantul bazei utilizând conceptele modelului de date (în cazul obiect: clase, moştenire, etc.) Schema unei BDOO poate fi, astfel, modificată dinamic în funcție de necesitățile utilizatorilor, ceea ce presupune:

- *definirea unei taxonomii şi a unui model al schimbărilor,* taxonomia definind un set de schimbări semnificative ale schemei, iar modelul furnizează o bază pentru specificarea semanticilor schimbărilor schemei;
- *implementarea schimbării schemei.*

Se identifică două tipuri de schimbări în schema unei baze de date orientate obiect:

- schimbări referitoare la *structura ierarhiei de clase,* care includ adăugarea sau ştergerea unei clase şi modificarea relațiilor supraclasă-subclasă dintre două clase;
- schimbări referitoare la *modul de definire al unei clase,* acestea cuprinzând modificări ale atributelor şi metodelor definite pentru o anumită clasă (de exemplu, schimbarea numelui sau domeniului unui atribut, adăugarea, ştergerea unui atribut sau metode din clasa respectivă).

Deci, un SGBDOO este un SGBD care trebuie să suporte (gestioneze) conceptele de obiect, moştenire şi polimorfism. Termenul de obiect trebuie să reprezinte suportul pentru obiecte atomice potențial foarte mari(de exemplu imagini, sunete), pentru obiecte arbitrar de complexe şi pentru asigurarea identificării permanente a fiecărui obiect şi a metodelor asociate. Moştenirea multiplă şi trimiterea de mesaje sunt considerate funcționalități adiționale.

Sunt posibile mai multe abordări în construirea unui SGBDOO, în funcție de modul în care sunt combinate tehnologiile specifice bazelor de date şi tehnologii specifice limbajelor de programare orientate obiect:

- *abordarea bazată pe modelul relațional*, care presupune extensia unui SGBD relațional şi a limbajelor lui pentru a cuprinde conceptele obiect. Una din variantele extensiei unui SGBD relațional este dotarea acestuia cu un subsistem (translator) pentru memorare relațională aşa cum propune produsul OpenDB al firmei Hewlett Packard, derivat din produsul IRIS [Beech88]. O integrare mai puternică constă în ideea ca SGBD-ul relețional să suporte direct conceptele obiect, abordare folosită în prototipul Postgres. Un astfel de SGBDOO trebuie să ofere, în general, o interfață de tip SQL (extins) pentru accesarea obiectelor şi apelarea metodelor. Adică este necesară o extensie SQL care, pe lângă definirea şi manipularea doar a tabelelor relaționale, să aibă comenzi speciale pentru descrierea tipurilor abstracte şi a claselor. De asemenea, utilizatorul trebuie să aibă posibilitatea de a manipula clasele ca pe nişte relații, clasele putând avea ca atribute referințe de alte clase sau putând avea atribute complexe;
- *abordarea cu obiecte persistente*, care înseamnă extinderea unui mediu de programare obiectuală în vederea definirii şi manipulării obiectelor persistente. Într-o astfel de abordare, diferită de prima, se porneşte de la un limbaj de programare orientat obiect (cum

ar fi C++ sau Smalltalk) pentru a realiza definirea schemei unei BDO şi a programelor aplicație care să integreze apelurile la rutinele de gestiune a datelor, adică pentru a ajunge la un SGBDOO;

- *abordarea integrată*, în care SGBDOO foloseşte un model ce integrează concepte semantice şi obiect, adică nu se bazează pe nici un model deja existent, fie el relațional sau obiect. Un SGBDOO integrat poate fi independent de orice limbaj de programare, oferind o interfață multilimbaj, aşa cum lucrează sistemul O_2;
- *păstrarea unui limbaj compatibil SQL (extins), dar fără a folosi modelul relațional*, aşa cum propune UniSQL. Această ultimă abordare este diferită de abordarea bazată pe modelul relațional deoarece SGBDOO şi modelul lui consideră, pur şi simplu, conceptul de relație ca pe un caz particular de constructor.

Caracteristicile obligatorii ale unui SGBDO sunt:

1. *manipularea obiectelor atomice şi complexe (compuse).* Un *constructor* este o funcție asociată unei clase care permite crearea şi inițializarea unui obiect, iar un *constructor* este de asemenea o funcție asociată unei clase prin care se distruge un obiect. Un obiect complex este realizat din alte obiecte, adică o parte dintre atributele unui astfel de obiect pot fi ele însele obiecte (instanțe ale unor clase), un obiect complex (compus) având, deci, o structură ierarhică (o ierarhie de părți). Această noțiune s-a născut, deci, prin aplicarea de constructori asupra unor obiecte simple. Minimul de constructori pe care un SGBDOO trebuie să îi conțină sunt constructorii de seturi, liste şi tupluri (înregistrări). O altă condiție în cadrul modelului orientat pe obiecte este *ortogonalitatea*, aceasta însemnând că fiecare constructor trebuie să fie aplicabil fiecărui obiect.
2. *Identitatea obiectelor***.** Orice obiect trebuie să primească în momentul creării (prin intermediul sistemului) un *identificator*, adică o referință unică independentă de valorile atributelor lui.
3. *Încapsularea*. Ea derivă din tipurile de date abstracte şi s-a menținut în gestiunea BDOO din două motive: necesitatea de a diferenția specificarea unei operații de implementarea acesteia şi ideea de modularizare. Încapsularea sugerează două părți pentru orice obiect. *Partea de interfață* reprezentând specificarea unui set de operații care se pot aplica asupra unui obiect, fiind singura zonă vizibilă a obiectului. *Implementarea* se referă atât la componenta de date, cât şi la cea de procedură. Partea de date este, de fapt, reprezentarea (câmpurilor) obiectului, în timp ce partea de procedură descrie, cu ajutorul unui limbaj de programare, implementarea fiecărei operații. Implementarea este mascată (ascunsă) în cadrul obiectului.

4. *Ierarhii de clase sau tipuri (moştenire simplă).* O clasă poate fi specializarea unei alte clase şi, prin urmare, o moşteneşte. Moştenirea reduce eforturile de programare. Există patru modalități de a moşteni: *prin substituție* (dacă asupra unui obiect din subtip se pot executa mai multe operații decât asupra unui obiect din supratip), *prin incluziune* (dacă orice obiect din subtip este şi obiect din supratip), *prin restricție* (caz particular al incluziunii în care obiectele subtipului sunt toate din supratip care satisfac o restricție specificată) şi *prin specializare* (obiectele subtipului aparțin şi supratipului, dar conțin un surplus de informații specifice).
5. *Polimorfismul.* Adică, la primirea unui mesaj, stabilirea metodei care se aplică se face în mod dinamic, în funcție de clasa obiectului în cauză. Astfel, obiecte diferite (instanțe ale unor clase diferite) pot fi adresate uniform (pot primi aceleaşi mesaje), dar manifestă comportamente diferite. Aceasta asigură manipularea simplă şi coerentă a seturilor eterogene de obiecte.
6. *Uşurința interogării.* Un obiect poate fi reperat utilizând valorile atributelor sale, legăturile cu alte obiecte sau metodele aplicate obiectului respectiv.
7. *Persistența obiectelor.* Obiectele au durata de viață mai mare decât programul care le-a creat.
8. *Concurența acceselor*. O BDOO poate fi partajată simultan de către tranzacții care o consultă şi tranzacții care o modifică.
9. *Fiabilitatea obiectelor*. Obiectele trebuie să poată fi restaurate în catul unui incident, adică să poată fi aduse la starea pe care au avut-o înaintea manifestării incidentului.

Dintre caracteristicile opționale ale unui SGBDOO menționăm:

1. *Moştenirea multiplă.* O clasă (subclassă) poate fi specializarea directă a două sau mai multe supraclase şi, deci, să moştenească proprietățile acestora.
2. *Versiuni ale obiectelor***.** Plecând de la un anumit obiect, prin modificări succesive ale schemei, pot fi obținute mai multe versiuni ale obiectului. Acestea generează un graf al versiunilor unui obiect care trebuie să fie gestionat de sistem.
3. *Distribuția obiectelor.* Aceasta presupune gestiunea obiectelor dintr-o BDOO în stații de lucru diferite (fragmente BDOO locale).
4. *Modelarea tranzacțiilor evoluate*. Ideea constă în acceptarea tranzacțiilor imbricate care pot fi descompuse în subtranzacții.

Nu există deocamdată o părere comună privind arhitectura unui SGBDOO, mai mult, arhitectura unui astfel de sistem poate fi o problemă de perspectivă, fiind dependentă de modul în care este văzut SGBDOO de către cercetător, ca utilizator final sau ca administrator de sistem. Totuşi, arhitectura unui SGBDOO trebuie să cuprindă trei componente majore:

1. *gestionarul de obiecte (Obiect Manager)* care asigură interfața dintre procesele (prelucrările) externe şi SGBDOO. Deci, funcțiile gestionarului de obiecte constau în:
   a. *funcții de prelucrare a mesajelor*, incluzând editarea de legături în momentul lansării în execuție şi verificarea tipului, ca şi translatarea cererilor;
   b. *facilități de definire şi modificare a schemei* bazei de date, incluzând definiri de clase noi sau corectate în ierarhii sau rețele de clase existente;
2. *serverul de obiecte* care este responsabil cu asigurarea serviciilor de bază corespunzătoare oricărui SGBD, cum ar fi: gestiunea tranzacțiilor şi gestiunea stocului de obiecte. Deci, serverul de obiecte realizează recuperarea (refacerea), inserția, ştergerea şi actualizarea obiectelor stocate în stocul rezident de obiecte. Un astfel de server poate manipula, de unul singur, tranzacții transmise de la mai mulți gestionari de obiecte. Deci, funcțiile unui server de obiecte sunt:
   a. *gestiunea tranzacțiilor*, incluzând controlul concurențial, gestiunea zonelor "buffer" şi servicii de refacere;
   b. *gestiunea fizică a stocului de obiecte*, incluzând plasarea obiectelor şi implementarea metodelor de acces;
   c. eventual *servicii de arhivare şi de asigurare a rezervelor (dublurilor);*
3. *stocul resident de obiecte* sau, mai exact, baza de date orientată obiect (BDOO) însăşi.

Bazele de date funcționale au apărut odată cu definirea modelului funcțional al datelor şi cu utilizarea unor limbaje fundamentate pe principiile programării funcționale ca limbaje de definire şi manipulare a datelor den bazele de date.

Considerând modelul funcțional, o bază de date funcțională (BDF) este definită ca un ansamblu de funcții memorate. Funcțiile sunt utilizate atât pentru definirea atributelor de entitate, deci pentru descrierea entităților, cât şi pentru exprimarea asocierilor (relațiilor) dintre entități.

Într-o BDF, atributele de entitate sunt considerate drept funcții care mapează entitatea pe anumite domenii. De exemplu, atributul NUME al unei entități PERSOANA este văzut ca o funcție care mapează PERSOANA pe domeniul şirurilor de caractere.

Legăturile dintre entități sunt reprezentate, în general, de două funcții. Considerând, de exemplu, entitățile MUNCITOR şi SECȚIE, una dintre funcții aplicată asupra entității MUNCITOR returnează o entitate SECȚIE (secția în care lucrează muncitorul respectiv), iar a doua funcție aplicată entității SECȚIE returnează o entitate MUNCITOR (muncitorii care lucrează în secția respectivă) . Din acest motiv, în cadrul BDF funcțiile nu trebuie să fie definite într-o singură direcție, impunându-se prezența unui mecanism explicit pentru *inversarea funcțiilor*. În plus este necesar şi un mecanism pentru *compunerea funcțiilor* prin

care să se asigure navigarea în cadrul modelului funcțional. Aceste două mecanisme stau la baza operațiilor de căpătâi folosite în cadrul BDF.

Prin intermediul ***bazelor de date inteligente (BDI)*** s-a urmărit realizarea unei gestionări naturale a datelor, adică a unei memorări, accesări şi utilizări fireşti şi facile a datelor, în scopul satisfacerii depline a cerințelor informaționale a utilizatorilor, deci pentru îmbunătățirea procesului decizional economic. Se poate afirma, deci, că BDI au apărut ca rezultat al cercetărilor în vederea ameliorării şi perfecționării tehnologiei bazelor de date.

Este de reținut faptul că interesul pentru BDI s-a manifestat, pe lângă domeniul bazelor de date, ți în domeniul inteligenței artificiale, în legătură cu eforturile de realizare a bazelor de cunoştințe. Pentru a soluționa problema gestionării şi utilizării unor volume mari de cunoştințe s-a ajuns la soluția naturală conform căreia se organizează aceste cunoştințe sub forma unor baze de cunoştințe, după modelul organizării detelor în baze de date. Cu toate că bazele de cunoştințe prezintă caracteristici structurale şi funcționale distincte de cele ale bazelor de date s-a constatat că cercetările din domeniul bazelor de cunoştințe şi cele din domeniul bazelor de date inteligente (BDI) se susțin reciproc, fiind posibilă o trecere de la tehnologia bazelor de date la cea a bazelor de cunoştințe prin intermediul BDI. Uneori, BDI sunt considerate drept baze de cunoştințe într-o formă "primară" sau chiar baze de cunoştințe propriu-zise.

Caracteristicile BDI precum şi soluțiile de realizare efectivă variază extrem de mult, ceea ce creează dificultăți în însăşi definirea bazelor de date inteligente (BDI), printre puținele realizări practice semnificative, se numără ***bazele de date deductive (BDDe)***.

BDI trebuie să permită tratarea unor mari cantități de date, provenind din diferite surse, sub diferite forme de reprezentare (texte, imagini, sunet, etc.). Aceasta înseamnă că BDI cere utilizarea *tehnologiilor multimedia*, aceasta presupunând disponibilități cât mai mari în domeniul suportului extern de date, forma digitizată a imaginilor şi sunetelor ocupând foarte mult spațiu de memorare. Pe lângă aceste disponibilități, tehnologiile multimedia necesită şi dispozitive speciale pentru introducerea/extragerea datelor (cititor optic de caractere, scanner, plotter, sintetizator de voce, etc.).

Integritate şi coerență

Un aspect important al bazelor de date relaționale este ***integritatea***.

Restricția de integritate este o asertiune care trebuie să fie verificată de către date la momente de timp determinate.

Baza de date coerentă este baza de date pentru care mulțimea restricțiilor de integritate (explicite sau implicite) este respectată de către datele bazei.

Există mai mute tipuri de restricții de integritate. Pentru a ilustra aceste tipuri diferite de restricții vom utiliza baza de date relațională din figura 1. Această bază de date modelează sumar un magazin clasic care stochează, vinde si cumpără produse.

Mai întâi, o restricție de integritate poate viza unicitatea datei. Definirea unui tip elernentar de dată permite adesea exprimarea unor asemenea restrictii, de exemplu (vezi figura 1):

- restrictii ale domeniului de variație (de exemplu data cantitate cumpărată este un număr intreg).
- restricții asupra domeniului de valori (de exemplu data cantitate vândută trebuie să fie cuprinsă intre 1 şi 1000).

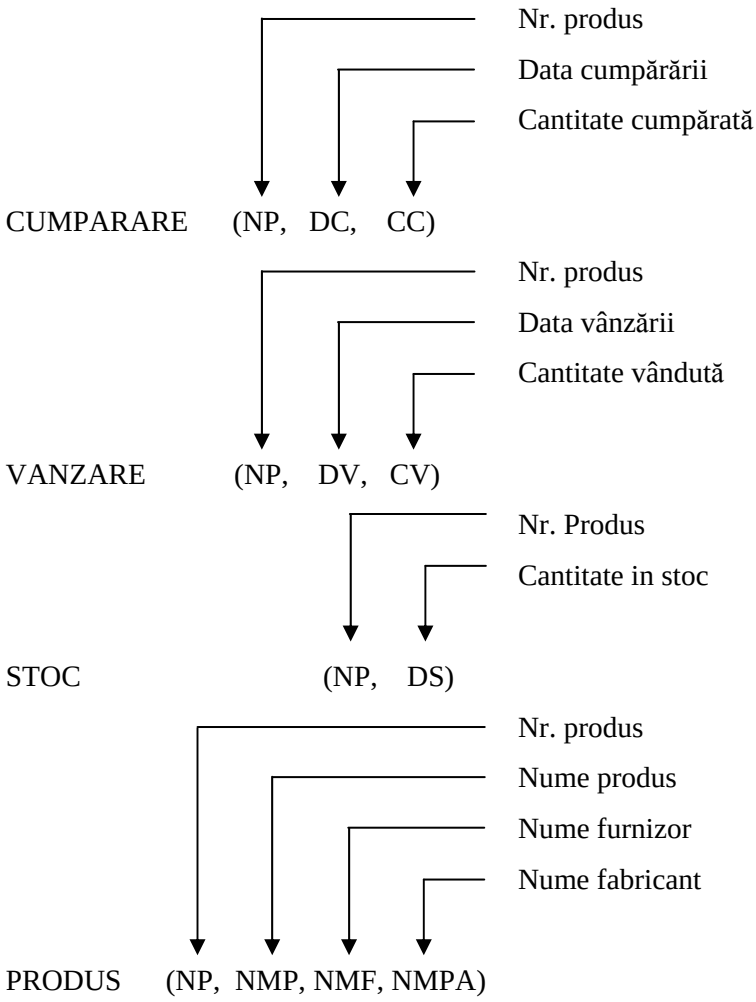


Figura 1. Exemple de scheme relaționale

O restricție de integritate poate de asemenea să acționeze asupra mai multor date. Asemenea restricții pot fi restricții de dependență între date:

- *dependențe functionale*, cum ar fi numărul produsului determină numele său (NP → NMP).
- *dependente multivoce*, ca de exemplu: număr produs determină multiplu numele furnizorului independent de numele fabricantului, la rândul sau multideterminat de

numărul produsului (NP →→ NMF şi NP →→ NMFA).

- *dependente de incluziune* cum ar fi de exemplu faptul că orice tuplu al relației STOC trebuie să corespundă unui tuplu in relatia PRODUS; se poate încă vorbi de dependențe *existențiale* sau *referențiale* in cazul particular de incluziune a cheilor de relații.
- *restricții aritmetice* care trebuie să fie verificate pentru anumite date; de exemplu, pentru fiecare produs, cantitatea în stoc (CS) trebuie să rămană egală cu suma cantităților cumpărate minus suma cantităților vândute (CS = ∑CC - ∑CV). Un caz particular al unor asemenea restricții sunt restricțiile de *duplicare* cum at fi A = B, care permit generarea de copii multiple ale unei date.
- un alt tip important de restricții de integritate este constituit de anumite *valori invariante* existente în bază. De exemplu, suma capitalurilor conținute intr-o bază de date bancară trebuie să rămână constantă în timpul executiei unui transfer.
- *restrictii temporale de integritate,* când anumite restricții de integritate pot fi dependente de timp, actionând de exemplu la sfârsitul unei săptămâni sau a unei luni.

Este important de notat că toate restricțiile de integritate (cu excepția restricțiilor temporale) cărora li se supune o bază de date pot fi exprimate cu ajutorul logicii predicatelor de ordinul intâi.

Păstrarea integrității datelor este mult mai dificilă în bazele de date distribuite eterogene, decât într-o bază de date omogenă. Bazele de date distribuite omogene posedă un puternic control central care este identic cu schema SGBD. Dacă nodurile din rețeaua distribuită sunt eterogene, pot apărea câteva probleme care pot amenința integritatea datelor distribuite. Aceste pot fi:

- Inconsistențe între restricțiile de integritate locale;
- Dificultăți în specificarea restricțiilor de integritate globale;
- Inconsistențe între restricțiile de integritate locale şi globale.

Restricțiile de integritate locale diferă în baze de date distribuite eterogene. Inconsistențele pot cauza probleme, în mod special interogările complexe care se bazează pe mai mult de o bază de date. Dezvoltarea unor restricții de integritate globale, pot elimina conflictele între bazele de date individuale. Totuşi, acestea nu sunt întotdeauna simplu ded implementat. Restricțiile de integritate globale, pe de altă parte sunt separate în organizațiile individuale. Nu este întotdeauna practic să schimbi structura organizației pentru a face baza de date distribuită consistentă. În cele din urmă aceasta conduce la inconsistențe între constrângerile locale şi globale. Conflictele depind de nivelul de control central. Dacă există un control global puternic, restricțiile de integritate globale au prioritate. Dacă controlul central este slab, restricțiile de integritate locale înving.

Controlul concurent ajută la asigurarea integrității datelor. Acest control regularizează modul în datele sunt utilizate atunci când mai mult de un utilizator accesează aceleaşi date. Într-un sistem distribuit, de cele mai multe ori, nu doar un singur SGBD controlează accesul la date. Dacă controlul concurenței nu este integrat într-un singur sistem distribuit, pot să apară diferite probleme. Astfel, există trei surse posibile de probleme de acces concurent la date. Acestea sunt:

- *Actualizări pierdute* –o actualizare terminată cu succes este ştearsă din neatenție de către un alt utilizator;
- *Tranzacții nesincronizate care violează restricțiile de integritate*;
- *Datele citite sunt incorecte deoarece sunt obținute în timpul unei actualizări*.

Fiecare dintre aceste trei probleme pot fi reduse sau înlăturate prin implementarea unei scheme adecvate de blocare (pe durata unei blocări doar un singur subiect are acces la o entitate dată) sau a unei metode de acces *timestamp* (primesc prioritatea prima dată subiecții care au cerut prima dată acest lucru).

Probleme mai deosebite apar în cadrul sistemelor SGBD cu niveluri multiple de acces. Într-un asemenea sistem, utilizatorii sunt restricționați la accesul complet la date. Politicile de acces restricționează accesul utilizatorilor doar la anumite date. Politicile de acces pentru sisteme cu niveluri multiple de securitate sunt de două tipuri: deschise sau închise. *O politică de acces deschisă*, presupune un sistem în care toate datele sunt considerate accesibile dacă nu accesul la un element de dată particulară nu este în mod expres interzis. Într-un sistem în care avem o *politică de acces închisă*, accesul la toate datele este în mod implicit interzis dacă utilizatorul nu are privilegii de acces pentru datele respective.

Securitatea bazelor de date distribuite

Aspecte generale ale securității bazelor de date

Pe lângă multiplele avantaje ale folosirii bazelor de date distribuite, acestea au şi câteva dezavantaje: proiectarea şi controlul sunt mai complexe iar securitatea este mai scăzută datorită faptului că datele sunt păstrate pe mai multe mașini.

Bazele de date distribuite întâmpină toate problemele de securitate pe care le întâmpină bazele de date centralizate şi în plus de acestea mai au şi altele în plus. Ne vom opri puțin asupra elementelor de securitate comune tuturor bazelor de date şi pe urmă vom analiza cele specifice bazelor de date distribuite.

O bază de date securizată trebuie să satisfacă următoarele cerințe:

1. Trebuie să posede o integritate fizică (protecție împotriva pierderii datelor cauzate de căderi de tensiune sau dezastre naturale);

2. Trebuie să asigure integritatea logică (protecția structurii logice a bazei de date);
3. Datele trebuie să fie disponibile când este nevoie de ele;
4. Baza de date trebuie să aibă un sistem de auditare;
5. Trebuie să fie asigurată integritatea datelor;
6. Accesul trebuie să se realizeze pe nivele în funcție de tipul datelor conținute;
7. Baza de date trebuie să permită autentificarea utilizatorilor la intrarea în sistem;
8. Datele sensibile trebuie să fie protejate împotriva accesului neautorizat.

Scopul securizării unei baze de date este de a se asigura faptul că datele stocate în baza de date sunt protejate împotriva accesului neautorizat, modificări neautorizate şi actualizări nepermise. Aceasta poate fi realizată utilizând controlul accesului, controlul concurent, actualizări utilizând proceduri care realizează actualizarea în două faze (aceasta rezolvă problema integrității bazei de date apărute în urma unei căderi de tensiune în timpul unei tranzacții).

Nivelul de restricționare a accesului depinde de sensibilitatea datelor şi de gradul în care dezvoltatorul aderă la principiul celui mai mic privilegiu (accesul este limitat doar la articolele necesare rezolvării sarcinilor atribuite). În mod tipic, un *grilaj* (lattice) este păstrat în SGBD (Sistemul de Gestiune al Bazei de Date), şi care stochează privilegiile de acces ale utilizatorilor individuali. Astfel, când un utilizator se loghează, interfața obține de la SGBD privilegiul de securitate specific fiecărui utilizator.

Nivelurile de acces pot fi modelate prin satisfacerea uneia sau mai multora dintre următoarele criterii:

1. *Disponibilitatea datelor* –indisponibilitatea datelor este o cauză comună a blocării unui element de date particular de către un alt subiect, care forțează subiectul care face cererea să aştepte într-o coadă;
2. *Acceptarea accesului* – numai utilizatorii autorizați pot vizualiza sau modifica datele. Într-un sistem cu un singur nivel de securitate, acest lucru este relativ uşor de implementat. Dacă utilizatorul este neautorizat, sistemul de operare nu-i acordă accesul în sistem. Pe un sistem cu nivele multiple de securitate, controlul accesului este mult mai dificil de implementat, deoarece SGBD trebuie să impună diferențiat privilegiile de acces ale utilizatorilor;
3. *Siguranța autenticității* –Aceasta include restricția accesului la orele normale de lucru pentru a se asigura că utilizatorul înregistrat este cel care pretinde că este. De asemenea, aceasta include şi analiza activităților utilizatorului în sistem, aceasta reducând probabilitatea unor atacuri.

În dezvoltarea bazelor de date distribuite, una dintre primele întrebări care necesită un răspuns este unde să se aloce drepturile de acces. Există astfel trei posibilități de alocare a drepturilor de acces.

- *Utilizatorilor le este acordat acces în sistem de la site-ul lor de acasă*. Această strategie este uşor de implementat, nefiind cu mult mai dificilă de implementat decât o strategie cu acces centralizat. Succesul acestei strategii depinde de încrederea comunicației dintre diferite site-uri (site-ul aflat la distanță trebuie să primească toate informațiile necesare). Deoarece multe site-uri diferite pot să aloce drepturi de acces, probabilitatea accesurilor neautorizate creşte. O dată ce un singur site a fost compromis, întregul sistem este compromis. Dacă fiecare site menține controlul accesului pentru toți utilizatorii, impactul compromiterii unui singur site este redus.
- *Utilizatorilor le este acordat acces în sistem de la un site aflat la distanță.* Această strategie asigură o securitate ridicată, dar are câteva dezavantaje. Probabil, cel mai important dezavantaj este acela că sunt necesare procese adiționale pe palierele superioare. De asemenea, operațiunile care au loc necesită participarea câtorva site-uri. În plus de aceasta, menținerea corectă a copiilor tabelelor este mult mai scumpă şi mult mai predispunsă la erori.
- *Acordarea privilegiilor cu drepturi de acces se face centralizat în anumite noduri ale rețelei denumite servere de politici (policy servers).* Aceste servere sunt aranjate în rețea. Când un server de politică primeşte o cerere de acces toți membrii din rețea determină dacă accesul utilizatorului este autorizat.

Tranzacții

Atunci când unei baze de date i se aplică o prelucrare, în general aceasta trece prin diferite stări tranzitorii în timpul cărora anumite restricții de integritate nu mai sunt verificate. Pentru a izola unitățile de prelucrare care respectă coerenta bazelor de date, se introduce noțiunea de tranzacție.

Tranzacția este unitatea de prelucrare secvențială, executată in contul unui utilizator care aplicată unei baze de date coerentă, restituie o bază de date coerentă.

Restricțiile de integritate sunt întotdeauna invarianți pentru tranzacții. Un sistem de baze de date execută in general o mulțime de tranzacții simultan pentru diverşi utilizatori. Problemele de concurență apar din cauza interferențelor care pot surveni între aceste tranzacții diverse asupra datelor partajate.

Datele sunt deci actualizate prin unități de prelucrare care respectă coerența bazei de

date, numite tranzacții. O tranzacție este compusă dintr-o unitate de prelucrare mai fină numită acțiune.

Acțiunea este o comandă invizibilă executată de către sistem, în cadrul unei tranzacții.

Exemplu: citirea unei date in memorie (READ x $\rightarrow x_1$), sau scrierea in bază (WRITE $x_1 \rightarrow$ x) sunt in general acțiuni.

Probleme de concurență

Un prim exemplu bine cunoscut de probleme ridicate la execuția simultană a tranzacțiilor este pierderea de operații. Cazul cel mai tipic este pierderea unei actualizări. Această problemă rezultă din faptul că o tranzacție poate, în timpul execuției unei acțiuni a_i să citească un obiect într-un tampon care-i aparține, apoi să modifice acest obiect (sau simplu să-l rescrie) in timpul unei acțiuni a_j (j > i), pornind de la valori citite anterior, in timp ce o actualizare a_k a acestui obiect a fost realizată printr-o altă tranzacție între acțiunile a_i si a_j. În acest caz, actualizarea a_k este pierdută. O asemenea situație este prezentată in figura 2. O problemă similară este confuzia de identitate care poate surveni când o tranzacție conservă o referință la o dată distrusă printr-o altă tranzacție.

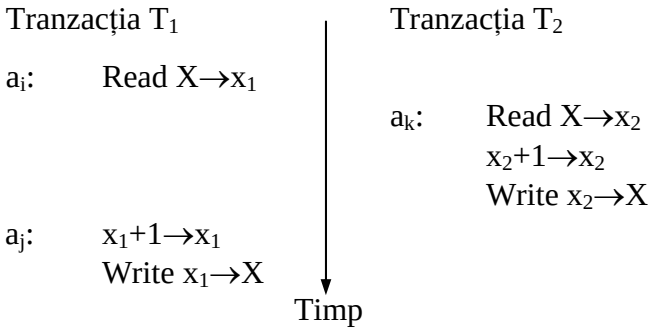


Figura 2. Exemplu de pierdere de operație

Un al doilea tip de problemă este ridicat de existenta restricțiilor de integritate. Această problemă, numită *inconsistență,* survine de fiecare dată când o tranzacție observă sau modifică greşit o stare tranzitorie a bazei de date, fiind caracterizată de faptul că o restricție de integritate nu este verificată. Un prim exemplu de inconsistență este prezentat în figura 3. A şi B sunt două date care trebuie să verifice restricția A=B. Tranzacția T_2 observă pe A după ce ea a fost modificată prin T_1 si pe B înainte ca ea să fi fost modificată prin T_1. Astfel T_2 imprimă o valoare pentru A diferită de cea a lui B. Un al doilea exemplu mai grav deoarece el provoacă distrugerea bazei de date este prezentat in figura 4. Aici, T actualizează o stare tranzitorie incoerentă caracterizată prin A=B. Inconsistențele pot fi produse de asemenea prin tranzacții datorită nereproductibilității citirilor. Într-adevăr, o tranzacție poate citi de două ori acceşi dată şi poate găsi două valori diferite datorită faptului că data a fost modificată printr-o tranzacție concurentă între două citiri.

Fig. 5 ilustrează aceeaşi situație: T_1 imprimă două valori diferite pentru aceeaşi dată A.

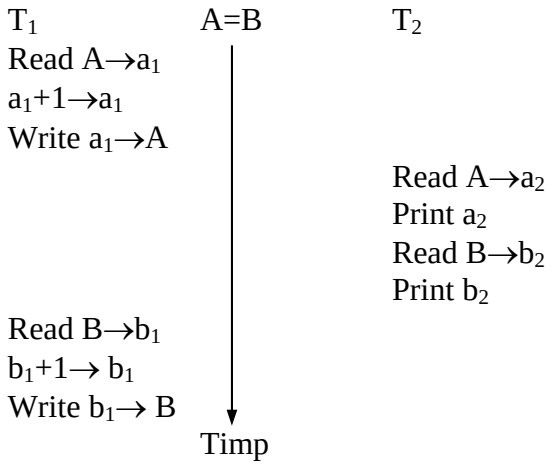


Figura 3. Exemplu de observare a inconsistențelor

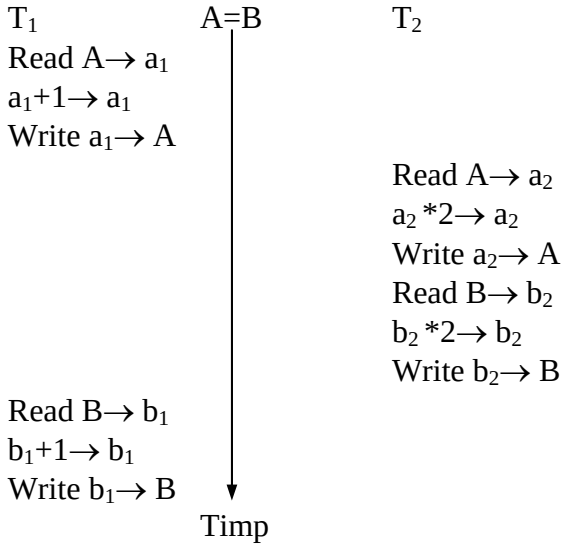


Figura 4. Exemplu de introducere de inconsistențe într-o bază de date

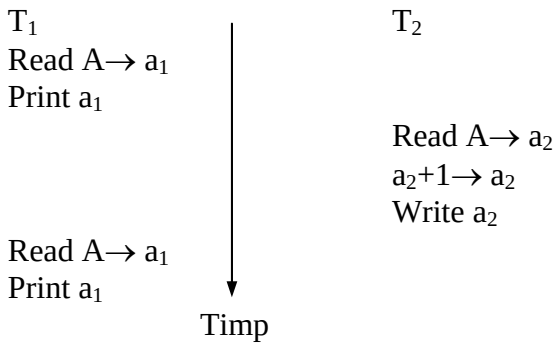


Figura 5. Nereproductibilitatea citirilor

Tranzacții în baze de date distribuite

Datele într-un sistem de baze de date distribuite sunt stocate pe mai multe maşini şi datele de pe fiecare maşină sunt gestionate de un SGBD care poate exista independent de celelalte maşini. Concepția clasică despre un sistem de baze de date distribuite este aceea că

sistemul ar trebui să facă impactul distribuției datelor transparent pentru utilizator. În particular, următoarele proprietăți sunt considerate a fi de dorit:

- ***independența datelor distribuite***: Utilizatorii ar trebui să poată interoga baza de date fără să spună unde sunt localizate tabelele, relațiile referite sau fragmente de relații. Acest principiu este o extindere naturală a independentei logice şi fizice a datelor. Mai mult, interogări care se propagă asupra mai multor maşini ar trebui să fie optimizate în mod sistematic într-o manieră bazată pe cost, luând în considerare costurile datorate comunicării şi diferențele dintre costurile calculelor locale.
- ***atomicitatea tranzacțiilor distribuite***: Utilizatorii ar trebui să poată să scrie tranzacții care accesează şi actualizează datele pe mai multe maşini ca şi când ar fi scris tranzacții asupra datelor locale. În particular, efectele tranzacțiilor distribuite pe mai multe maşini ar trebui să continue să fie atomice; cu alte cuvinte, toate modificările rămân valabile dacă tranzacția a făcut COMMIT şi nici una nu rămâne dacă ea a fost nereuşită.

Deşi proprietățile de mai sus sunt de dorit, în anumite situații, de exemplu, când maşinile sunt conectate de o rețea înceată pe o distanță mare, aceste proprietăți nu sunt uşor de obținut. Când maşinile sunt complet distribuite, aceste proprietăți nici nu mai sunt de dorit.

Dacă datele sunt distribuite, dar toate serverele rulează acelaşi SGBD, avem un sistem de baze de date distribuite omogen. Dacă pe calculatoare diferite rulează SGBD-uri diferite, autonome şi sunt conectate împreună pentru a accesa datele de pe mai multe maşini, vom avea un sistem de baze de date distribuite heterogen sau un sistem multi-bază de date.

Pentru construcția sistemelor heterogene trebuie să avem standarde general acceptate pentru gateway-uri. Gateway-urile sunt pachete soft care acceptă cereri (într-un format SQL), le trimite la SGBD-ul local şi apoi returnează răspunsul la client (într-un format standard). Accesând servere de date prin gateway-uri, diferențele dintre ele (capabilități, formatul datelor, etc.) sunt mascate şi diferitele servere dintr-un mediu distribuit sunt legate la un grad înalt.

Există, în principiu, două tipuri de arhitecturi pentru bazele de date distribuite.
Prima arhitectură este reprezentată de sistemele client-server. Un sistem client-server are unul sau mai multe procese client şi unul sau mai multe procese server, iar clientul poate trimite o interogare la oricare proces server. Clienții sunt responsabili pentru interfața cu utilizatorul, iar serverele întrețin datele şi execută tranzacții. De aceea un proces client poate rula pe un calculator personal şi poate trimite datele la un server funcțional pe un mainframe.

A doua arhitectură de sisteme de baze de date distribuite este formată din mai multe servere care colaborează între ele. În această arhitectură, fiecare server este capabil să execute tranzacții asupra datelor locale şi prin colaborare ele pot executa şi tranzacții care să se întindă

pe mai multe maşini. Când un server primeşte o interogare care necesită accesul la datele de pe alte servere, generează subquery-uri care să fie executate de celelalte servere şi pune rezultatele împreună pentru a da un răspuns la interogarea originală.

Managementul tranzacțiilor distribuite

Într-un SGBD distribuit, o anumită tranzacție este pornită pe o anumită maşină, dar ea poate să acceseze datele de pe altă maşină. În cele ce urmează, vom numi subtranzacție, partea din activitatea unei tranzacții care se referă numai la datele de pe un calculator. Când o tranzacție este pornită pe un calculator, managerul de tranzacții de aici împarte tranzacția într-o colecție de una sau mai multe subtranzacții care se execută pe maşini diferite, le trimite spre execuție managerilor de tranzacții de pe celelalte maşini şi coordonează activitatea lor. Ne propunem ca în continuare să răspundem la următoarele întrebări/aspecte: [HOFFER,2002]

- Controlul concurenței distribuite: Cum pot fi gestionate blocările de obiecte stocate pe alte maşini decât cea locală? Cum pot ti detectate deadlock-urile într-o bază de date distribuită?
- Refacerea datelor în baze de date distribuite: Atomicitatea tranzacției trebuie să fie asigurată- când o tranzacție face *commit*, toate acțiunile sale de pe toate maşinile pe care se execută tranzacția trebuie să fie persistente. Analog, dacă o tranzacție face *abort* (este nereuşită), nici una din acțiunile sale nu trebuie să fie persistente.

Controlul concurenței distribuite

Într-un mediu concurent, momentul în care se obține un blocaj asupra unui obiect şi momentul în care se renunță la el (are loc deblocarea) este determinat de protocolul de control a concurenței. Managementul blocărilor poate fi distribuit pe mai multe maşini în multe feluri:

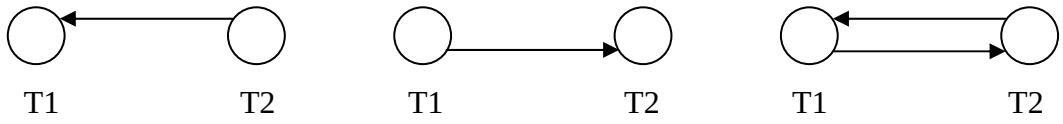
- Centralizat: o singură maşină este responsabilă de tratarea cererilor de blocare/deblocare pentru toate obiectele.
- Copie primară: o copie a fiecărui obiect este desemnată ca fiind copia primară. Toate cererile de blocare/deblocare a unei copii a acestui obiect sunt tratate de managerul de blocaje (lock manager) de pe maşina unde este stocată copia primară a obiectului (indiferent de locul unde se află copia care se vrea blocată).
- Complet distribuit: cererile de blocare sau deblocare a unei copii a unui obiect stocat pe o anumită maşină sunt tratate de managerul de blocaje de pe maşina unde este stocată copia.

Schema centralizată este vulnerabilă la căderea maşinii care controlează blocajele. Schema cu copie primară evită această problemă, dar în general, citirea unui obiect, în acest caz, înseamnă comunicarea cu două maşini: maşina unde este salvată copia primară a obiectului şi maşina unde se află copia care se citeşte. Această problemă este evitată în schema complet

distribuită deoarece blocarea se face pe maşina unde este stocată copia care se citeşte. Cu toate acestea, la scriere, trebuiesc setate blocaje pe toate maşinile unde sunt modificate copii în cazul schemei complet distribuite, pe când la celelalte două scheme blocajul e necesar numai la o maşină.

Deadlock distribuit

Un aspect care necesită atenție sporită când se foloseşte schema cu copie primară sau cea complet distribuită este detecția deadlockurilor. În orice SGBD deadlock-urile trebuie detectate şi rezolvate (renunțând la anumite tranzacții care au cauzat deadlock- ul). Fiecare maşină menține un graf local cu obiectele care aşteaptă (depind) de alte obiecte şi, bineînțeles, un ciclu în acest graf indică un deadlock. Oricum, un deadlock se poate întâmpla şi dacă nu există nici un ciclu într-un graf local. Să presupunem, de exemplu, că avem două maşini, A şi B, amândouă conținând copii ale obiectelor 01 şi 02 şi că folosim în tranzacții tehnica "read-any, write-a1l". Tranzacția T1, care vrea să citească 01 şi să scrie 02, obține un blocaj pe obiectul 01 şi altul pe 02 pe maşina A şi cere un blocaj pe obiectul 02 la maşina B. Între timp, tranzacția T2 care vrea să citească 02 şi să scrie 01 obține o blocare pe obiectul 02 şi una pe 01 la maşina A. După cum se vede în figura de mai jos, tranzacția T1 aşteaptă după T2 la maşina B, iar T2 aşteaptă după T1 la maşina A; deci avem un deadlock care nu poate fi detectat bazându-ne numai pe grafe de aşteptare locale.



Maşina A Maşina B Graf de aşteptare global

Pentru a detecta astfel de deadlock-uri globale trebuie să folosim un algoritm de detecție a deadlock-urilor distribuite. Vom descrie trei astfel de algoritmi.

- Primul algoritm este centralizat şi consistă în trimiterea periodică a tuturor grafelor de aşteptare locale la o maşină care este responsabilă cu detecția deadlock-utilor globale. La această maşină, graful de aşteptare global este generat prin combinarea tuturor grafelor de aşteptare locale. Mulțimea nodurilor este reuniunea nodurilor din grafele locale şi există câte o muchie de la un nod la altul dacă există o muchie de acest fel într-unul dintre grafele locale.
- Al doilea algoritm este ierarhic şi grupează maşinile din mediul distribuit într-o ierarhie. Fiecare nod din ierarhie construieşte un graf de aşteptare pentru descoperirea deadlock-urilor care implică numai maşinile conținute în subarborele acestui nod. Astfel, toate maşinile trimit periodic graful de aşteptare local la maşina responsabilă pentru construirea grafului de aşteptare global. Acest algoritm se bazează pe observația că deadlock-urile se

întâmplă în mod frecvent între maşinile vecine. Toate deadlock-urile vor fi în final detectate, doar că unele mai repede decât altele.

- Al treilea algoritm operează astfel: dacă execuția unei tranzacții durează mai mult decât o valoare de time-out, atunci ea este terminată (aborted). Deşi acest algoritm poate cauza multe restartări care nu sunt neapărat necesare, încărcarea datorată detecției deadlock-urilor este mică şi în baze de date distribuite heterogene în care maşinile nu pot să schimbe grafe de aşteptare ea este singura opțiune.

O observație în ce priveşte detecția deadlock-urilor distribuite: întârzieri în propagarea informațiilor locale poate determina algoritmul de detectare a deadlock-urilor să identifice deadlock-uri care de fapt nu există (aşa numitele *phantom deadlocks*) şi să provoace abort-uri nenecesare.

Refacerea datelor în urma tranzacțiilor eşuate

Recuperarea datelor într-un SGBD distribuit comportă o complexitate mai mare decât în cazul SGBD-urilor centralizate datorită următoarelor motive:

- pot apărea noi tipuri de defecte (căderea rețelei, căderea unei stații *remote* pe care se execută o subtranzacție etc.) [FRANKLIN,1996]
- fie toate subtranzacțiile unei tranzacții fac *commit*, fie nici una şi această proprietate trebuie garantată în ciuda căderii unei legături sau a unui host. Această proprietate este garantată prin folosirea unui *protocol de commit*.

Atât într-un SGBD centralizat cât şi într-unul distribuit, anumite acțiuni trebuiesc efectuate într-o execuție normală pentru a se obține informațiile necesare pentru recuperarea datelor în cazul unei erori. Pe lângă aceste informații, într-un SGBD distribuit se mai țin fişiere de *„loguri”* la fiecare maşină din sistem. Cel mai folosit protocol de *commit* este *Two-Phase Commit (2PC).* O variantă a acestuia, numită *2PC with presumed abort* a fost adoptată ca standard industrial.

Two-Phase Commit Protocol (2PC)

În timpul unei execuții normale, fiecare maşină din sistem menține „loguri” ale acțiunilor/subtranzacțiilor care au loc. Pe lângă aceasta, este urmat un *protocol de commit* pentru a garanta faptul că toate subtranzacțiile unei tranzacții date fie fac *commit*, fie *abort* în mod uniform. Managerul de tranzacții de pe stația de unde este originară tranzacția se numeşte *coordonator* pentru toate tranzacțiile; managerii de tranzacții de pe maşinile unde se execută subtranzacții se numesc *subordonați*. Protocolul 2PC presupune un schimb de mesaje între hosturi şi scrierea de „loguri”. Când utilizatorul decide să realizeze tranzacția, comanda *commit* este trimisă la coordonatorul tranzacției. Aceasta inițializează protocolul 2PC:

1. Coordonatorul trimite un mesaj *prepare* la fiecare subordonat.
2. Când un subordonat primeşte mesajul *prepare,* decide ce să facă cu subtranzacția: să realizeze *commit* sau *abort*. Scrie apoi un *log abort* sau un *log prepare* (în cazul în care a decis *commit*) şi apoi trimite un mesaj *yes* sau *no* la coordonator.
3. Dacă coordonatorul primeşte răspunsul *yes* de la toți subordonații, scrie un *log commit* şi apoi trimite mesajul *commit* la toți subordonații. Dacă, în schimb, primeşte cel putin un răspuns *no* sau nu primeşte răspuns de la unii subordonați într-un interval de timp (time-out), el va scrie un *log abort* şi apoi trimite mesajul *abort* la toți subordonații.
4. Când un subordonat primeşte un mesaj *abort*, scrie un *log abort*, trimite un mesaj *ack* la coordonator şi termină subtranzacția (*abort*). Când un subordonat primeşte mesajul *commit,* scrie un *log commit,* trimite coordonatorului un mesaj *ack* şi face *commit* la subtranzacție.
5. Când un coordonator primeşte mesaje *ack* de la toți subordonații, scrie un log se sfârşit pentru tranzacție.

Numele *two-phase commit* reflectă faptul că două runde de mesaje sunt schimbate: prima dată are loc o fază de pregătire (votare) a *commit*-ului, apoi o fază de terminare, amândouă fiind inițiate de coordonator. Principiul de bază este ca oricare dintre managerii de tranzacții implicați (coordonator şi subordonați) pot termina (*abort*) unilateral tranzacția, dar pentru *commit* trebuie acordul tuturor. Înainte de a trimite un mesaj conform protocolului 2PC se scrie un log cu decizia respectivă tocmai pentru a garanta faptul că decizia supraviețuieşte chiar dacă stația suferă vreo defecțiune fatală. O tranzacție este oficial încheiată (commited) în momentul în care log-ul *commit* al coordonatorului este scris pe suport. Alte erori/esecuri care mai pot apărea după acest moment nu mai afectează tranzacția; este irevocabil "comisă". Înregistrările de log conțin tipul înregistrării, id-ul tranzacției şi identitatea coordonatorului.

Refacerea datelor (recovery) după o defecțiune gravă

Când o stație revine on-line după o defecțiune gravă/restart, SGBD-ul invocă un proces de refacere a datelor care citeşte logurile şi procesează toate tranzacțiile care executau protocolul de *commit* când a apărut defecțiunea. Managerul de tranzacții (Transaction Manager) al acestei stații ar fi putut fi coordonatorul pentru unele din aceste tranzacții şi subordonatul pentru altele. Procesul de refacere a datelor urmăreşte liniile:

- dacă avem un log de *commit* sau *abort* pentru tranzacția T, starea ei este clar; mai executăm odată tranzacția, respectiv refacem datele dinaintea tranzacției. Dacă stația curentă este coordonatorul (lucru care poate fi determinat din logurile *commit* şi *abort*), trebuie să retrimitem în mod periodic - deoarece pot exista alte legături sau stații căzute -

un mesaj *commit* sau *abort* la fiecare subordonat până când primim un *ack;* după ce am primit un *ack*, scriem un log de sfârşit de tranzacție pentru T.

- dacă avem un log *prepare* pentru tranzacția T, dar nu avem log de *commit* sau de *abort*, înseamnă că stația este un subordonat şi coordonatorul îl aflăm din logul *prepare*. Trebuie, apoi, să contactăm în mod repetat pe coordonatorul tranzacției ca să determinăm starea tranzacției T. După ce coordonatorul răspunde fie cu *commit*, fie cu *abort*, o să scriem un log corespunzător, re-executăm tranzacția sau refacem datele dinaintea tranzacției - în funcție de răspunsul dinainte - şi apoi scriem logul de sfârşit de tranzacție pentru T.
- dacă nu avem în loguri nici *prepare*, nici *commit*, nici *abort* pentru tranzacția T, în mod sigur T nu a trecut de prima fază a protocolului 2PC până în momentul în care s-a produs defecțiunea; deci putem să facem *abort* şi să refacem datele, iar apoi să scriem un log de sfârşit de tranzacție. În cazul acesta nu avem cum să determinăm dacă stația este coordonator sau subordonat pentru T. Oricum, dacă stația aceasta este coordonator, sete posibil să fi trimis un mesaj prepare înainte de defecțiune şi, dacă este aşa, alte stații ar fi putut să voteze *yes*. Dacă un asemenea subordonat contactează procesul de refacere a datelor de pe stația curentă, acum ştim că stația curentă este un coordonator pentru T şi, fiindcă nu este nici un log de *abort* sau *commit*, răspunsul către subordonat trebuie să fie "*abort* T".

Se poate observa că dacă coordonatorul pentru tranzacția T eşuează, subordonații care au votat *yes* nu pot decide ce să facă cu tranzacția T, *commit* sau *abort*, până când coordonatorul revine on-line - în acest caz se spune că T este blocată. În principiu, stațiile subordonate active pot comunica între ele şi dacă cel puțin una dintre ele conține un log de *abort* sau *commit* pentru tranzacția T, starea lui T devine global cunoscut. Ca să comunice unul cu celalalt, toți subordonații trebuie să cunoască identitatea celorlalți subordonați la momentul în care au primit mesajul *prepare.* Cu toate acestea, protocolul 2PC rămâne vulnerabil la defecțiunea coordonatorului în timpul procesului de refacere a datelor, deoarece chiar dacă toți subordonații au votat *yes*, coordonatorul – care are şi el un vot – ar fi putut decide să facă *abort* la tranzacția T şi această decizie nu poate fi determinată până când coordonatorul nu revine on-line.

Am discutat mai sus modul în care o stație reface datele după o defecțiune fatală, dar ce ar trebui să facă o stație implicată într-un protocol de *commit* dacă se pierde legătura cu o altă stație cu care comunică? Dacă stația curentă este coordonator ar trebui sa facă *abort* la tranzacție. Dacă însă, stația curentă este un subordonat şi încă nu a răspuns la mesajul *prepare* al coordonatorului atunci trebuie să facă *abort* la tranzacție. Dacă este un subordonat şi a votat *yes*, atunci nu poate, în mod unilateral, să facă *abort* la tranzacție şi nici nu poate face *commit*

- stația este blocată; ea trebuie să contacteze periodic pe coordonator până când primeşte un răspuns. Căderile de linii de comunicație sunt văzute de stațiile active ca şi defecțiuni ale stațiilor cu care comunică şi, deci, soluțiile de mai sus se aplică şi în cazul acesta.

Three-Phase Commit

Un protocol de *commit* numit *three-phase commit (3PC)* poate evita blocarea chiar dacă stația coordonator suferă o defecțiune fatală în timpul refacerii datelor (recovery). Ideea de bază este că atunci când coordonatorul trimite un mesaj *prepare* şi primeşte un vot *yes* de la toți subordonații, el trimite la toate stațiile un mesaj *precommit,* nu un mesaj *commit.* Când un număr suficient de *ack* au fost recepționați, coordonatorul scrie un *log commit* şi trimite un mesaj *commit* la toți subordonații. În 3PC coordonatorul amână efectiv decizia de a face *commit* până când este sigur că destule stații ştiu despre decizia de a face *commit*; dacă coordonatorul se defectează de mai multe ori, aceste stații pot comunica una cu cealaltă şi pot detecta faptul că tranzacția trebuie "commisa" -sau *abort* dacă nici una dintre ele n-a primit un mesaj *precommit* - fără să mai aştepte până coordonatorul revine on-line.

Protocolul 3PC impune un cost adițional semnificativ în timpul execuției normale şi cere ca căderile de linii de comunicații să nu ducă la o partiționare a rețelei (atunci când unele stații nu se mai pot conecta la altele prin nici o cale) pentru ca să asigure eliberarea de blocaje. Din acest motiv, nu prea este folosit în practică.

Algoritmi de control al concurenței bazelor de date distribuite

Integritatea bazelor de date distribuite poate fi afectată cel mai des în cazul unui acces concurent la date. Dacă cererile de acces la baza de date sunt de tipul „regăsire”, atunci secvențializarea accesului la mediul de memorare este suficientă pentru a nu mai fi nevoie de alte precauții. Situația se schimbă atunci când se fac modificări asupra datelor, şi anume când mai mulți utilizatori accesează aceiaşi resursă pentru o „actualizare”. Un exemplu tipic este sistemul de rezervare a locurilor, unde mai mulți agenți fac rezervări simultan şi deci modifică permanent lista locurilor libere şi lista rezervărilor. În lipsa unui control adecvat al cererilor de acces există pericolul ca acelaşi loc să fie rezervat de mai multe ori. Acest lucru este posibil deoarece în intervalul de timp de la acceptarea unei anumite cereri de rezervare pe un loc anume şi până la rezolvarea acesteia (eliminarea locului cu pricina din lista de locuri libere) ar putea fi acceptate şi rezolvate alte cereri de rezervare pentru acelaşi loc. Deci două procese care citesc şi modifică valoarea aceluiaşi obiect ar putea interacționa în mod nedorit atunci când sunt executate simultan. Pentru evitarea acestor interacțiuni este necesară impunerea unor restricții asupra execuției concurente a proceselor care execută operații de citire şi de modificare a datelor.

Desigur, calea cea mai simplă pentru a realiza acest lucru, este şi cea mai ineficientă, deoarece minimizează eficiența sistemului. Aceasta este de a executa tranzacțiile în mod independent, una după cealaltă. Nivelul de concurență (numărul de tranzacții executate concurent), este unul dintre cei mai importanți parametrii pentru caracterizarea unui sistem de baze de date, şi de aceea mecanismele de control al concurenței urmăresc realizarea unui compromis între menținerea consistenței bazei de date şi obținerea unui nivel de concurență cât mai ridicat.

Anomalii de interferență în bazele de date distribuite

Două sau mai multe tranzacții asupra unei BDD poate conduce la apariția unor stări inconsistente ale bazei de date dacă acestea nu sunt controlate, şi la apariția unor rezultate inconsistente.

Dacă avem două tranzacții T_i şi T_j spunem că sunt *susceptibile de interferență* dacă rezultatul execuției concurente a acestora poate fi diferit de rezultatul execuției seriale. Acestea pot apărea atunci când T_i şi T_j efectuează operații asupra unor date comune. Spunem că două tranzacții T_i şi T_j sunt *conflictuale,* dacă ele sunt efectuate în mod concurent. Anomaliile de interferență pot fi:

- *Anomalii de actualizare pierdute;*
- *Anomalii de citire improprie;*
- *Anomalii de citire neproductibilă.*

Anomalii de actualizare pierdută

Acest tip de anomalie corespunde unui conflict de tip scriere – scriere. Ea constă în faptul că rezultatul actualizării efectuate de o tranzacție se pierde ca urmare a reactualizării aceleiaşi date de către o altă tranzacție, fără ca reactualizarea să fie influențată de rezultatul primei actualizări.

Exemplu: avem două tranzacții T_1 şi T_2 şi următoarea secvență de execuție concurentă:

T_1	T_2
	Citeşte Data;
Citeşte Data;	
Data = Data + 25;	
Scrie Data;	
	Data = Data + 10;
	Scrie Data;

În urma acestor instrucțiuni variabila *Data* ar trebui mărită cu 25 + 10 = 35, dar la final se observă că această variabilă este mărită cu 10.

Anomalii de citire improprie

Acest tip de anomalie corespunde unui conflict de tip citire-scriere şi apare atunci când o tranzacție surprinde o stare temporar inconsistentă a bazei de date.

Exemplu: avem următoarea secvență de instrucțiuni concurente pentru două tranzacții T_1 şi T_2 :

T_1	T_2
Citeşte Data1;	
Data1 = Data1 - 25;	
Scrie Data1;	
	Citeşte Data1;
	Citeşte Data2;
	Data3 = Data1 + Data2;
	Scrie Data3;
Citeşte Data2;	
Data2 = Data2+25;	
Scrie Data2;	

În această secvență, valoarea sumei Data1+Data2 este aceiaşi înainte şi după execuția de mai sus. Se doreşte a se reține în Data3 valoarea acestei sume, dar datorită interferenței valoarea Data3 este cu 25 mai mică decât cea reală.

Anomalii de citire nereproductibilă

Corespunde unui conflict de tip citire-scriere şi apare atunci când aceeaşi tranzacție găseşte valori diferite la citiri repetate ale aceleiaşi date.

Exemplu: avem următoarea execuție concurentă a două tranzacții T_1 şi T_2 :

T_1	T_2
Citeşte Data1;	
Data2 = Data1;	
Citeşte Data2;	
	Citeşte Data1;
	Data1=Data1+25;
	Scrie Data1;
Citeşte Data1;	
Data3 = Data1;	
Scrie Data3;	

În urma execuției tranzacției T_1 valorile rezultate pentru *Data2* şi *Data3* ar trebui să fie egale, însă ele sunt diferite datorită interferenței cu tranzacția T_1.

Serializabilitatea în bazele de date distribuite

Prin execuția concurentă a mai multor tranzacții se pot obține rezultate diferite față de situația în care fiecare tranzacție este executată independent. În mod firesc se consideră corect rezultatul obținut prin execuția independentă a tranzacțiilor. O astfel de execuție o numim *execuție serială*. O *planificare* a unui set de tranzacții este o ordine de execuție a unor paşi elementari (LOCK, READ, WRITE) ai tranzacției setului. O planificare spunem că este *serială* dacă toți paşii oricărei tranzacții apar în poziții consecutive ale planificării. O asemenea planificare determină o execuție serială, fără interferențe a tranzacțiilor. O

planificare se numeşte *serializabilă* dacă şi numai dacă efectul ei este echivalent cu cel al unei planificări seriale.

Exemplu:

Dacă avem două tranzacții T_1 şi T_2 definite prin secvențele de operații:

T_1: READ A; A=A-10;WRITE A; READ B; B=B+10;WRITE B;

T_2: READ B; B=B-20;WRITE B; READ C; C=C+20;WRITE C.

Orice planificare serială a tranzacțiilor celor două are proprietatea că suma A+B+C rămâne nemodificată. În tabelul de mai jos, prezentăm trei planificări diferite ale tranzacțiilor T_1 şi T_2 :serială, serializabilă şi neserealizabilă:

Serială		*Serializabilă*		*Neserializabilă*	
T_1	T_2	T_1	T_2	T_1	T_2
READ A		READ A		READ A	
A=A-10			READ B	A=A-10	
WRITE A		A=A-10			READ B
READ B			B=B-20	WRITE A	
B=B+10		WRITE A			B=B-20
WRITE B			WRITE B	READ B	
	READ B	READ B			WRITE B
	B=B-20		READ C	B=B+10	
	WRITE B	B=B+10			READ C
Serială		*Serializabilă*		*Neserealizabilă*	
	READ C		C=C+20	WRITE B	
	C=C+20	WRITE B			C=C+20
	WRITE C		WRITE C		WRITE C

Tabel 1 – Planificări pentru tranzacțiile T_1 şi T_2

După execuția planificării neserealizabile, suma A+B+C este mărită cu 20 ca urmare a pierderii actualizării lui B (WRITE B) din tranzacția T_2 .

Serializibilitatea în baze de date distribuite nereplicate

În cazul *bazelor de date distribuite, nereplicate,* ordinea de execuție a tranzacțiilor în fiecare nod este reprezentată printr-o *planificare locală*. Dacă baza de date este nereplicată şi planificările locale ale tuturor nodurilor sunt serializabile, atunci *planificarea globală*, rezultată din reuniunea tuturor planificărilor locale, este la rândul ei serializabilă atât timp cât planificările locale prevăd aceeaşi ordine de serializare pentru tranzacții comune.

Serializibilitatea în baze de date distribuite replicate

Într-o bază de date distribuită replicată (în care pentru aceiaşi dată apar mai multe copii în noduri diferite) este posibil ca planificările locale să fie seriale şi cu toate acestea consistența mutuală a datelor să fie compromisă.

Consistența mutuală a tuturor copiilor datelor replicate presupune faptul că, cel puțin din punctul de vedere al utilizatorului, aceste copii sunt permanent identice. Pentru aceasta, nu

este suficient ca planificările locale nodurilor să fie serializabile, ci este necesar ca planificarea globală să fie serializabilă. Trebuie realizate, astfel următoarele condiții:

1. Fiecare planificare locală trebuie să fie serializabilă;
2. Oricare două operații conflictuale trebuie să fie în aceeaşi ordine relativă în toate planificările locale care apar împreună. – garantează faptul că ordinea de serializare a două tranzacții este aceeaşi în toate nodurile în care acestea se execută împreună.

În cazul bazelor de date replicate, apare şi problema suplimentară a menținerii consistenței mutuale a copiilor aceleiaşi date. Aceasta înseamnă că orice actualizare a unei copii trebuie propagată şi asupra celorlalte în aşa fel încât rezultatele să fie identice. Asigurarea acestor condiții este garantată prin *protocoale de control al replicilor*.

În esență un protocol de control al replicilor acționează în modul descris în cele ce urmează. Dacă avem o dată x şi x_1, x_2, …, x_n copii ale acesteia. Data x va fi referită ca *dată logică*, iar copiile sale vor fi referite ca *date fizice*. Utilizatorii vor lansa operații de acces (citire sau scriere) referindu-se la data logică x. Protocolul de control al replicilor are rolul de a transforma orice operație de acces la o dată logică în operații echivalente asupra datelor fizice. Modul concret de realizare a acestei transformări depinde de particularitățile fiecărui protocol de control al replicilor.

Astfel, pentru bazele de date distribuite replicate este necesară satisfacerea a două criterii de corectitudine:

- *Controlul replicilor* – din punct de vedere al utilizatorilor, setul de copii ale unei date logice se comportă ca şi când ar fi o singură copie a datelor respective
- *Controlul concurenței* – prin care efectul execuției concurente a mai multor tranzacții este echivalent cu efectul execuției seriale ale aceloraşi tranzacții.

Algoritmi de blocare distribuită

Gestiunea operațiilor de blocare este realizată printr-o componentă a SGBD numită *lock manager*. În cazul bazelor de date distribuite replicate această gestiune este mai complexă deoarece presupune transformarea cererilor de blocare lansate de către utilizator asupra unor date logice în operații de blocare a datelor fizice. Astfel, fiecare nod al unei baze de date are câte un look manager care gestionează operațiile de blocare locale, dar care în plus, posedă şi facilități de cooperare şi cooperare cu look manager-urile locale din celelalte noduri în vederea implementării protocolului de control al replicilor. Dacă o dată are mai multe copii fizice distribuite în diferite noduri, translatarea unei cereri de blocare a acesteia în operații de blocare a datelor fizice se poate face după mai multe protocoale de control al replicilor. Câte dintre aceste protocoale sunt prezentate mai jos.

a) Protocolul Write-Locks-All_Read-Lock-One

Acest protocol, este definit prin următoarele reguli impuse tranzacțiilor:

1. Pentru ca o tranzacție să obțină o blocare pentru citire (RLOCK(x)) asupra unei date logice x, este suficient să obțină o blocare pentru citire (RLOCK(x_i)) asupra unei copii oarecare x_i a lui x.
2. Pentru ca o tranzacție să obțină blocarea pentru scriere (WLOCK(x)) a unei date logice x, este necesar să obțină blocarea pentru scriere (WLOCK(x_i)) a tuturor copiilor x_i a lui x.

Efectul combinat al regulilor de mai sus, împreună cu regulile locale din fiecare nod este că nu pot exista două tranzacții care să blocheze simultan aceiaşi dată logică *x*, dacă cel puțin una dintre blocări este pentru scriere.

Acest protocol necesită 2n mesaje de control şi n mesaje de date pentru scriere, respectiv 2 mesaje de control şi unul de date pentru citire.

b)Protocolul de blocare majoritară

Este un protocol, care este definit prin următoarele reguli impuse tranzacțiilor:

1. Pentru ca o tranzacție să obțină o blocare pentru citire (RLOCK(x)) asupra unei date logice *x*, este necesar să obțină blocarea pentru citire (RLOCK(x_i)) a majorității copiilor x_i ale lui *x*.
2. pentru ca o tranzacție să obțină blocarea pentru citire (WLOCK(x)) a unei date logice *x*, este necesar să obțină blocarea pentru scriere (WLOCK(x_i)) a majorității copiilor x_i ale lui *x*.

Protocolul de blocare majoritară asigură excluderea mutuală la nivelul datei logice *x*, deoarece nu pot exista două tranzacții care să blocheze simultan data logică *x*, dacă una dintre blocări este pentru scriere.

Acest protocol necesită cel puțin *n+1* mesaje de control şi *n* mesaje de date pentru o operațiune de scriere. Pentru citire sunt necesare tot *n+1* mesaje de control dar numai un singur mesaj de date.

c) Protocolul nodului central

Acest protocol este cel mai simplu dintr-o clasă de protocoale a căror caracteristică comună este aceea că responsabilitatea acceptării sau respingerii unei cereri de blocare pentru o dată logică *x* revine unui singur nod al rețelei distribuite, indiferent de numărul copiilor existente pentru *x*. Sarcina respingerii sau acceptării unei cereri de blocare revine unui singur nod din rețea desemnat ca *nod central*. În sistem există deci un singur look manager care este plasat în nodul central.

Procedura pentru o operație de scriere sau citire a unei date logice *x* este următoarea:

1. Nodul central primeşte o cerere de blocare (pentru citire sau scriere) a datei logice *x*;

2.

a - Dacă cererea de blocare primită nu poate fi satisfăcută, atunci nodul central trimite un mesaj de control corespunzător către nodul care a lansat cererea;

b – Dacă cererea de blocare a fost satisfăcută, şi operația este de citire, atunci nodul central trimite un mesaj de control către unul din nodurile în care există o copie a lui *x*, mesaj prin care se comandă transferarea valorii lui *x* către nodul care a lansat cererea. Dacă operația este de scriere, atunci nodul central trimite un mesaj de control către nodul solicitant prin care celui din urmă i se acordă dreptul de a actualiza toate copiile datei logice *x*.

3. Dacă operația este de citire, atunci valoarea citită x este transmisă printr-un mesaj de date la nodul solicitant. Dacă operația este de scriere, atunci nodul solicitant transmite câte un mesaj de date (cu noua valoarea a lui x) către fiecare nod în care există o copie a datei logice *x*. De regulă, la acestea se mai adaugă un mesaj de control, de la nodul solicitant către nodul central, pentru deblocarea datei logice x.

În cazul acestui protocol sunt necesare 2 mesaje de control pentru operația de citire şi 3 mesaje de control pentru operația de scriere. Dezavantajul este că în cazul unui trafic de rețea intens, nodul central devine suprasolicitat.

d) Protocolul nodului primar

Acest protocol, este asemănător cu protocolul nodului central cu deosebirea că pentru fiecare dată logică *x* se desemnează un *nod primar* care joacă rolul nodului central pentru data respectivă. În plus, de regulă, în nodul primar al unei date logice există şi o copie a acesteia. Deci, în principiu, în fiecare nod al rețelei există un loog manager responsabil pentru gestionarea cererilor de blocare a unei părți a datelor logice din sistem. O cerere de blocare a unei date logice *x*, va trebui deci adresată nodului primar asociat lui *x*.

Acest protocol, necesită 2 mesaje de control pentru o operație de scriere şi unul singur pentru citire.

e) Protocolul copiilor primare

Protocolul copiilor primare se bazează pe folosirea unor operatori de citire şi de scriere care funcționează ca privilegii pe care le pot obține nodurile rețelei în contul tranzacțiilor care solicită accesul la date.

Într-un moment de timp determinat, pentru o dată logică *x*, poate exista în sistem doar un singur operator de scriere. Dacă nu există nici un operator de scriere, atunci pot exista oricâți operatori de citire pentru *x*. Dacă un nod deține un operator de scriere pentru *x*, atunci acesta poate accepta o cerere de blocare pentru scrierea lui *x* din partea unei tranzacții care se

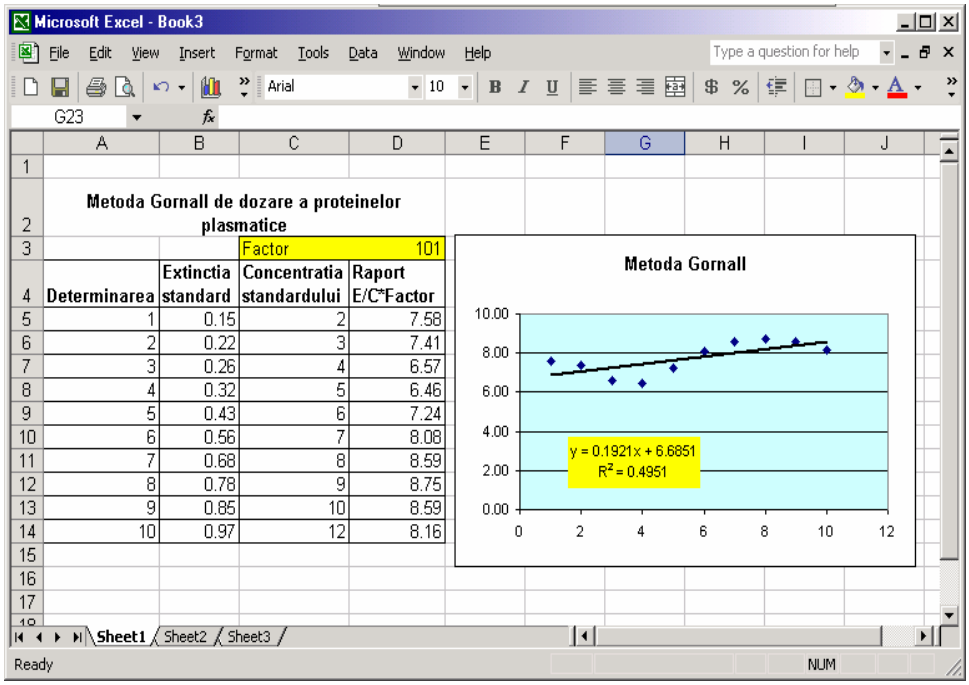
execută în acest nod. Un nod care deține doar un operator de citire pentru *x* poate accepta o cerere de blocare pentru citirea lui *x*, dar nu și pentru scrierea lui.

Dezavantajul acestui protocol este numărul mare de mesaje. Astfel, pentru o operație de citire sau de scriere sunt necesare *3n* mesaje de control.

38. Soluția Microsoft Office: Excel & Access

Excel: tabele şi grafice

1. Deschideți aplicația Microsoft Excel (calea *Start/Programs/Microsoft Excel*) şi realizați un tabel cu aspectul de mai jos:



2. Calculați utilizând formula:

   $\frac{E}{C} \times factor$ = *extincție/concentație*factor* valorile din coloana D;

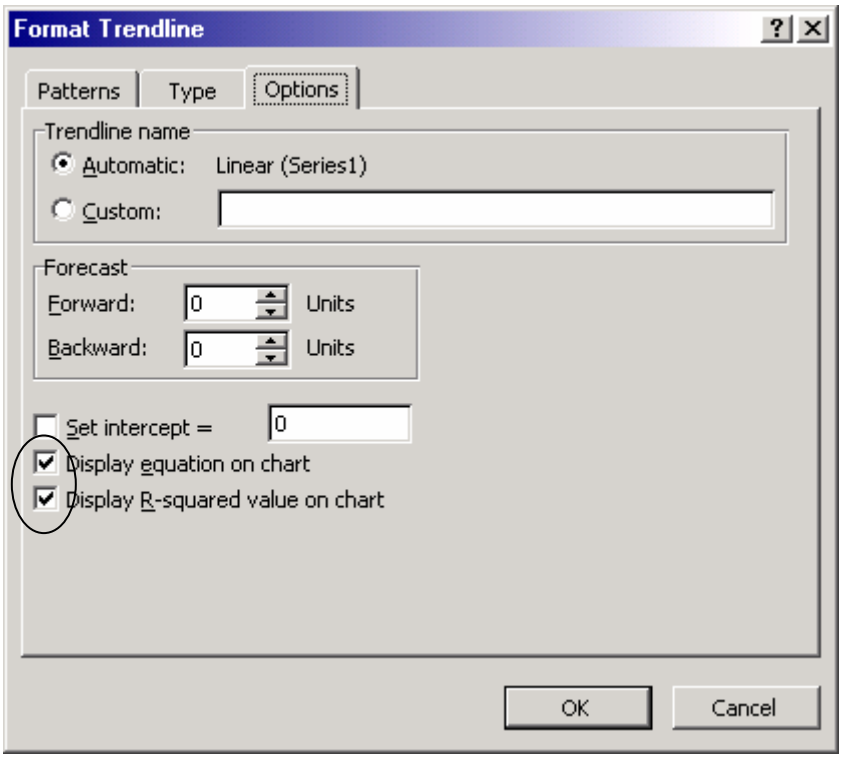
3. Reprezentați grafic legătura dintre concentrație şi raport;

4. Salvați fişierul.

Instrucțiuni:

1. După ce introduceți conținutul unei celule, validați cu [Enter] sau [Tab];
2. Pentru a edita conținutul unei celule folosiți tasta [F2] sau dublu click;
3. Pentru a realiza un grafic selectați valorile pe care doriți să le reprezentați şi apoi apasati butonul pentru Chart Wizard: 
4. Pentru a introduce o formulă într-o celulă începeți întotdeauna cu = şi adresați celulele prin numele celulei (litera care desemnează coloana şi cifra care indică linia); în cazul nostru formula este *extincție/concentație*factor* ceea ce se traduce prin introducerea în celula D5 a formulei: =B5/C5*D$3;
5. Pentru a copia formula pentru celelalte celule (D6...D14) selectați domeniul D5:D14 şi folosiți din meniu opțiunea *Edit-Fill-Down* (semnul $ în D$3 specifică faptul că valoarea rândului 3 va fi invariabilă la copiere);

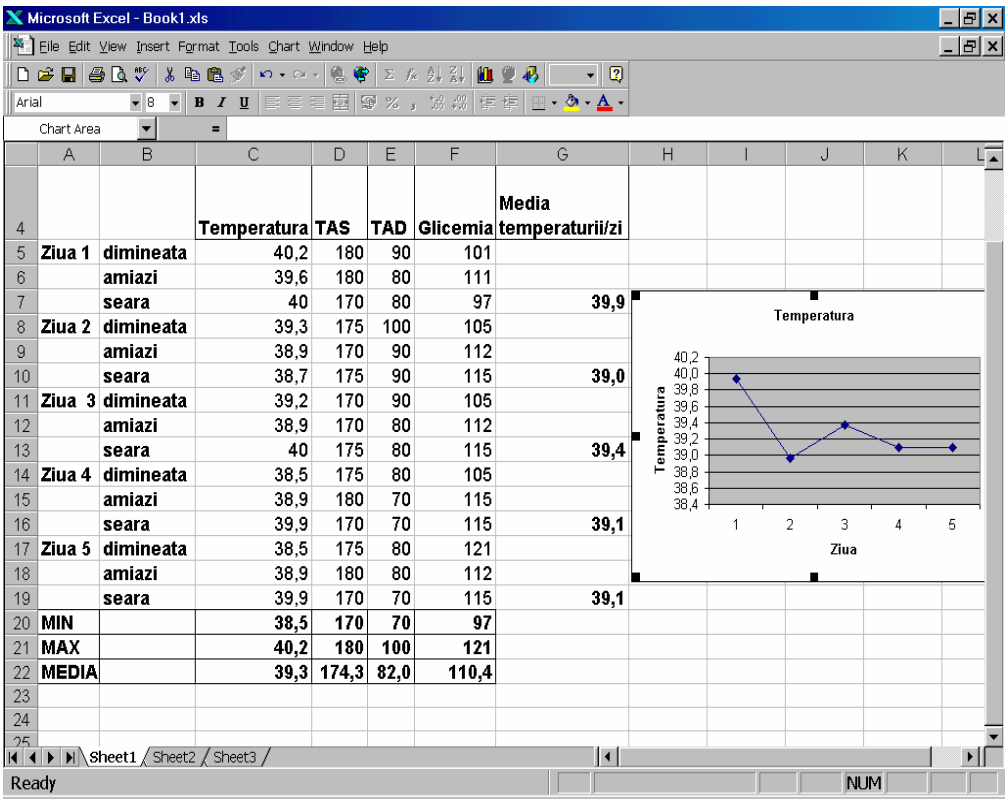
6. Selectați domeniul D5:D14 şi accesați cu butonul drept al mouse-ului meniul de context *FormatCells/Number*, iar la *Category* selectați *Number* pentru a schimba numărul de zecimale cu care se afişează valoarea obținută (veți alege 2 zecimale);
7. Selectați în grafic un punct şi accesați cu butonul drept al mouse-ului meniul de context *AddTrendline* pentru a trasa dreapta de regresie. În fereastra care se deschide, la *Type* selectați dreapta de regresie liniară, iar la *Options* alegeți să se tipărească pe ecran ecuația dreptei şi valoarea coeficientului de determinare R^2:



Coeficientul de determinare se apropie de valoarea 1 atunci când între cele două şiruri de valori există o relație de dependență liniară puternică ($0.5 < r^2 \leq 1$) şi de 0 atunci când nu există o relație de dependență liniară între cele două şiruri de valori ($0 \leq r^2 < 0.0625$).

Excel: formule

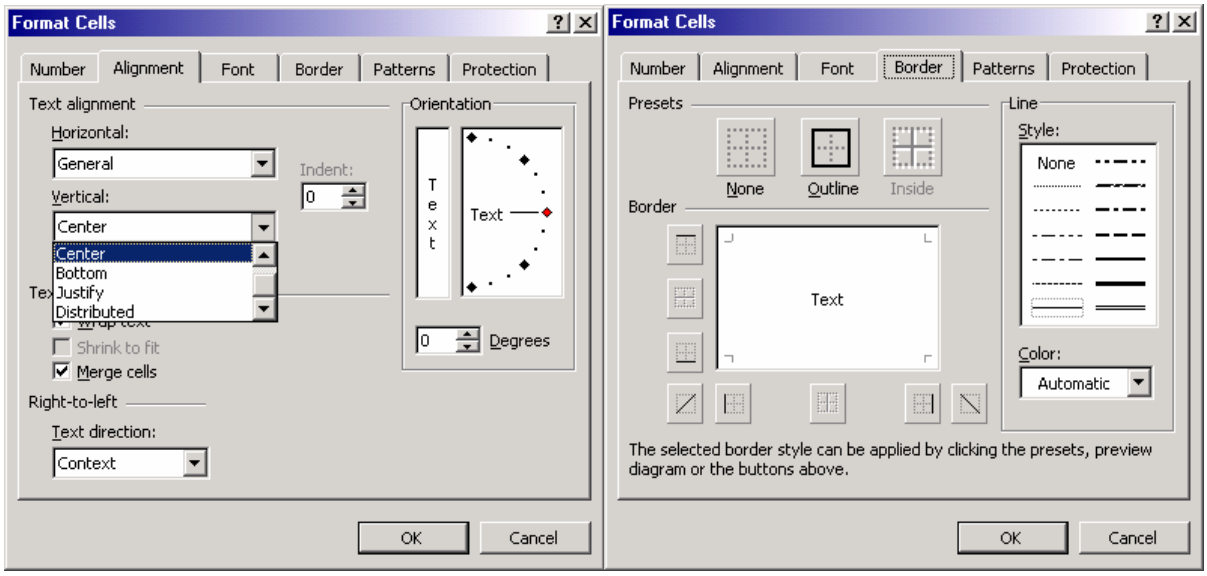
1. Deschideți aplicația Microsoft Excel (calea *Start/Programs/Microsoft Excel*) şi realizați un tabel cu aspectul de mai jos.
2. Determinați valorile minime, maxime şi media temperaturilor, TAS, TAD şi glicemiei.
3. Reprezentați grafic evoluția mediei zilnice a temperaturii.
4. Salvați fişierul.



	A	B	C	D	E	F	G
4			Temperatura	TAS	TAD	Glicemia	Media temperaturii/zi
5	Ziua 1	dimineata	40,2	180	90	101	
6		amiazi	39,6	180	80	111	
7		seara	40	170	80	97	39,9
8	Ziua 2	dimineata	39,3	175	100	105	
9		amiazi	38,9	170	90	112	
10		seara	38,7	175	90	115	39,0
11	Ziua 3	dimineata	39,2	170	90	105	
12		amiazi	38,9	170	80	112	
13		seara	40	175	80	115	39,4
14	Ziua 4	dimineata	38,5	175	80	105	
15		amiazi	38,9	180	70	115	
16		seara	39,9	170	70	115	39,1
17	Ziua 5	dimineata	38,5	175	80	121	
18		amiazi	38,9	180	80	112	
19		seara	39,9	170	70	115	39,1
20	MIN		38,5	170	70	97	
21	MAX		40,2	180	100	121	
22	MEDIA		39,3	174,3	82,0	110,4	
23							
24							

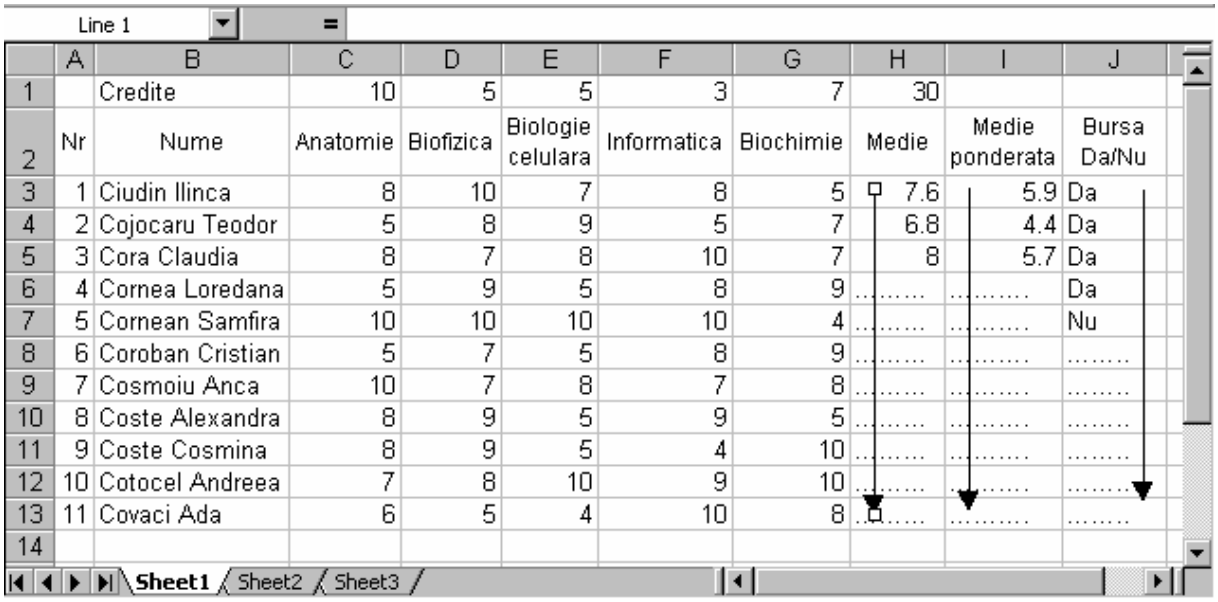
Instrucțiuni:

1. După ce introduceți conținutul unei celule validați cu *Enter* sau *Tab*.
2. Pentru a corecta conținutul unei celule folosiți tasta *F2* sau *Dublu click*.
3. Pentru a introduce o funcție selectați: *Insert-Function*, alegeți *MIN*, *MAX* respectiv *AVERAGE* după care alegeți domeniul de la C5 la C19 (de exemplu), sau scrieți C5:C19.
4. Pentru a micşora numărul de zecimale selectați celula cu valoarea pentru care doriți micşorarea numărului de zecimale şi din *Format-Cells-Number* alegeți opțiunea *Number* iar la caseta cu zecimale scrieți numărul de zecimale dorite. Pentru altă variantă mai rapidă apăsați pe butonul: 
5. Pentru reunirea mai multor celule într-una singura selectați *Format-Cells-Alignment*, alegeți opțiunea *Merge cells*.
6. Pentru alinierea textului în centrul celulei selectați textul dorit iar apoi alegeți din meniu comanda *Format-Cells-Alignment*, alegeți opțiunea *Center* din casuța derulantă.
7. Pentru spargerea textului dintr-o celulă pe mai multe rânduri selectați textul dorit iar apoi alegeți din meniu comanda *Format-Cells-Alignment***,** alegeți opțiunea *Wrap text*.
8. Pentru haşurarea conturului celulei selectați *Format-Cells-Border*, alegeți tipul de linie dorită şi apoi modul în care doriți haşurarea celulei, de pe butoane.
9. Pentru a realiza un grafic selectați valorile pe care doriți să le reprezentați grafic, în cazul nostru G7:G19, şi apoi căutați butonul Chart Wizard şi selectați tipul de grafic dorit, etc.



Excel: Formule

Deschideți aplicația Microsoft Excel (butonul ***Start–Programs–Microsoft Excel***) si realizați un tabel cu aspectul de mai jos:



	A	B	C	D	E	F	G	H	I	J
1		Credite	10	5	5	3	7	30		
2	Nr	Nume	Anatomie	Biofizica	Biologie celulara	Informatica	Biochimie	Medie	Medie ponderata	Bursa Da/Nu
3	1	Ciudin Ilinca	8	10	7	8	5	7.6	5.9	Da
4	2	Cojocaru Teodor	5	8	9	5	7	6.8	4.4	Da
5	3	Cora Claudia	8	7	8	10	7	8	5.7	Da
6	4	Cornea Loredana	5	9	5	8	9	.....	.....	Da
7	5	Cornean Samfira	10	10	10	10	4	.....	.....	Nu
8	6	Coroban Cristian	5	7	5	8	9	.....	.....	.....
9	7	Cosmoiu Anca	10	7	8	7	8	.....	.....	.....
10	8	Coste Alexandra	8	9	5	9	5	.....	.....	.....
11	9	Coste Cosmina	8	9	5	4	10	.....	.....	.....
12	10	Cotocel Andreea	7	8	10	9	10	.....	.....	.....
13	11	Covaci Ada	6	5	4	10	8	.....	.....	.....
14										

Sheet1 Sheet2 Sheet3

În coloana H să se calculeze media aritmetică a notelor pentru fiecare student. In coloana I să se calculeze pentru fiecare student media aritmetică ponderată pentru fiecare materie cu numărul de credite (aflate în linia 1 a tabelului). In coloana J să se verifice pentru fiecare student dacă îşi păstrează bursa sau nu după cum îşi ia toate examenele cu note ≥5 sau nu. Salvați fişierul.

Instrucțiuni:

1. Pentru *a micşora numărul de zecimale* apăsați pe butonul: 
2. Completați tabelul cu formulele corecte în conformitate cu formulele de mai jos:
   1. Pe coloana cu Media pentru celula H3 aveți formula: =(C3+D3+E3+F3+G3)/5

2. Pe coloana cu Media ponderată pentru celula I3 aveți formula:

   =((C$1*C3)+(D$1*D3)+(E$1*E3)+(F$1*F3)+(G$1*G3))/H$1

3. Pe coloana cu Bursă Da/Nu pentru celula J3 aveți formula:

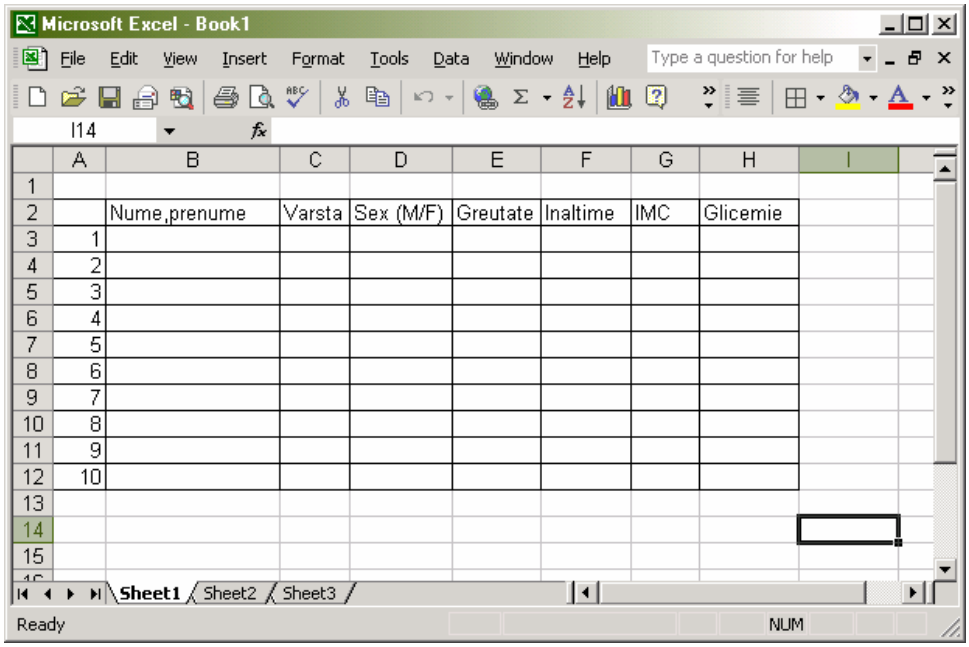
   =IF(OR(C3<5,D3<5,E3<5,F3<5,G3<5),"Nu","Da")

4. Pentru coloanele Medie, Medie ponderată, Bursă formula se poate copia astfel : se selectează coloana şi se alege opțiunea din meniu *Edit/Fill Down*:

3. Pentru formatare selectați domeniul dorit. Alegeți din meniu comanda *Format/Cells/Alignment* şi selectați opțiunea *Center* din caseta derulantă pentru aliniere în centrul celulei**,** selectați opțiunea *Merge text* pentru reuniunea mai multor celule, selectați opțiunea *Wrap text* pentru spargerea textului pe mai multe rânduri.
4. Pentru formatare selectați domeniul dorit. Alegeți din meniu comanda *Format/Cells/Border* şi selectați tipul de linie dorită pentru încadrarea celulei sau domeniului selectat, apoi apăsați butonul pentru tipul de încadrare dorită.

Excel: Sortare

Realizați un tabel cu aspectul de mai jos în care introduceți date pentru 10 pacienți:



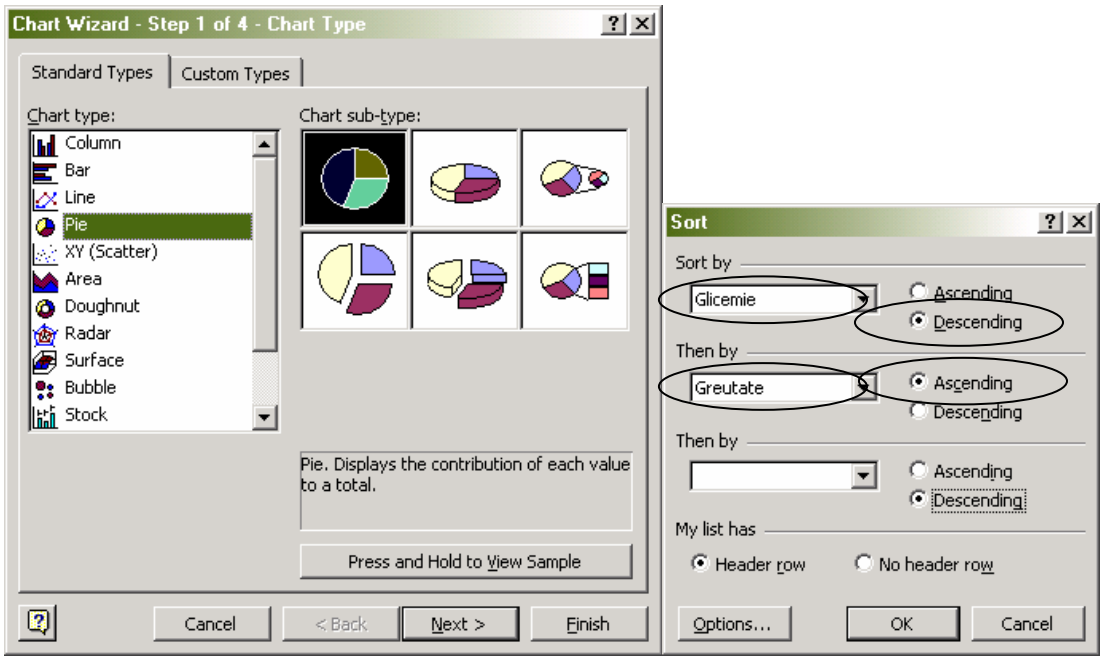
Cerințe:

1. Să se completeze coloana lipsă după formula: IMC = Greutate(kg)/Inaltime(m)^2
2. Să se realizeze un grafic cu distribuția pe sexe a pacienților.
3. Să se ordoneze descrescător tabelul după coloana Glicemie (ca prima cheie) şi apoi după greutate (a doua cheie).

4. Realizați o reprezentare grafică care să exprime legătura dintre IMC şi vârstă. Plasați pe grafic şi dreapta de regresie şi coeficientul de determinare precum şi titlu de grafic şi titluri pe axe.

Instrucțiuni:

1. Pentru a introduce o formulă în celula respectivă (de exemplu în celula G3) scrieți =E3/(F3^2).
2. Pentru a micşora numărul de zecimale selectați celula cu valoarea pentru care doriți micşorarea numărului de zecimale şi din *Format/Cells/Number* alegeți opțiunea *Number* iar la caseta cu zecimale scrieți numărul de zecimale dorite.
3. Pentru a realiza graficul alegeți butonul Chart Wizard şi selectați tipul Pie:



4. Pentru a sorta datele din tabel după coloana H şi E selectați tot tabelul apoi alegeți din meniu comanda *Data/Sort* şi alegeți ca primă cheie de sortare Glicemia descrescător şi apoi după greutate tot descrescător.
5. Realizați o reprezentare grafică de tip scatter cu datele corespunzătoare din coloanele C şi G. Graficul respectiv poate fi completat şi cu dreapta de regresie liniară şi coeficientul de determinare.

Access: gestiunea datelor

În continuare se va ilustra realizarea unei colecții de date medicale precum şi unele aplicații cu aceste date, folosind sistemul de gestiune al bazelor de date Access.

Problemă propusă. Să se realizeze o aplicație care să gestioneze activitatea unui cabinet medical, luând în considerare două categorii de date, de identificare a pacientului şi date privind consultația folosind Microsoft Access.

Paşii de lucru:

- analiza problemei;
- realizarea diagramei entitate – relație;
- implementarea aplicației în Microsoft Access.

Analiza problemei

Specificația sugerează folosirea următoarelor entități:

Pacient, identificat prin:	Nume
	Data naşterii
	Sex
Consultație, identificată prin:	Pacientul care este consultat
	Data la care se face consultul
	Diagnosticul în urma consultului

Observație: entitățile stabilite în urma analizei trebuie să acopere aplicațiile care urmează să fie realizate cu ajutorul bazei de date.

Realizarea diagramei entitate - relație

Pacient la o anumită dată> Consultație

Se presupune pentru simplitate că un pacient nu este consultat într-o zi de două ori. Diagrama entitate - relație este utilă din următoarele puncte de vedere:

- diagrama ne dă numărul de tabele pe care le va conține baza de date – fiecărei entități îi va corespunde un tabel;
- diagrama ne arată care este relația dintre tabele.

Deci, pe baza diagramei entitate – relație, baza noastră de date va conține două tabele, tabela cu datele referitoare la pacienți, pe care o vom denumi *Pacienti* şi tabela cu datele referitoare la consultații, pe care o vom denumi *Consultatii*.

În cadrul aplicației noastre, fiecărui pacient îi va fi asociat un identificator intern, un *cod personal*. Acest cod trebuie să fie absolut transparent utilizatorului, utilizatorul nici nu trebuie să știe de prezența lui. Vom numi acest identificator *Id_pacient*.

Tabela *Pacienti* va avea patru câmpuri: *Id_pacient*, *nume*, *data_nasterii* şi *sex* iar tabela *Consultatii* trei: *Id_pacient*, *Data_consultatiei*, *Diagnostic*.

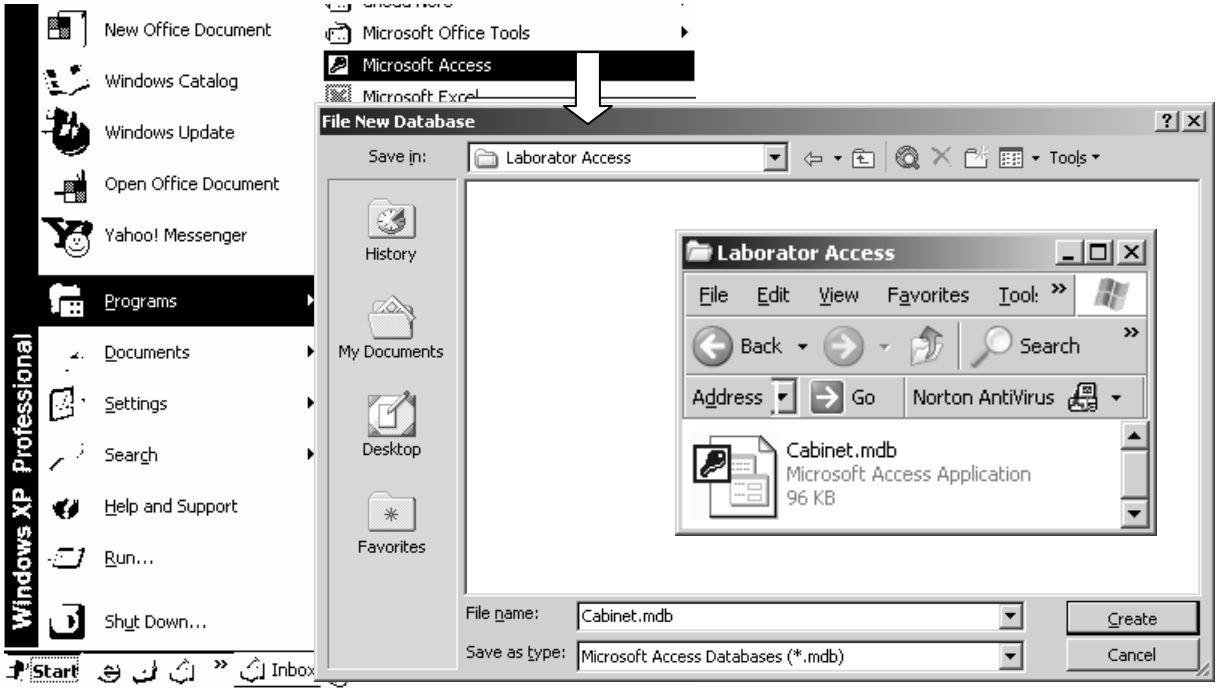
Cheia primară este elementul care definește unicitatea şi consistența datelor într-o bază de date.

Pentru baza noastră de date cheia primară a tabelului *Pacienti* este formată din câmpul *IdPacient* iar pentru tabelul *Consultatii* este formată din câmpurile *NumPac* şi *Data_consultatiei*.

Implementarea aplicației

1. Se deschide aplicația Microsoft Access (calea *Start/Programs/Microsoft Access*);

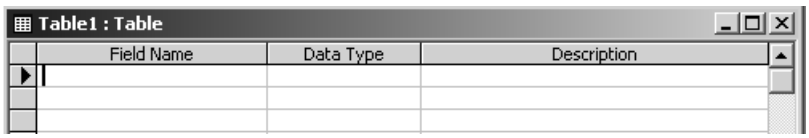
2. Se crează un nou fişier de date; După cum se observă, numele fişierului bază de date este “Cabinet” , el având extensia MDB. Directorul în care se va salva acest fişier este directorul propriu localizat pe server. Identificarea unui fişier Microsoft Access este foarte uşoară;



Crearea tabelelor

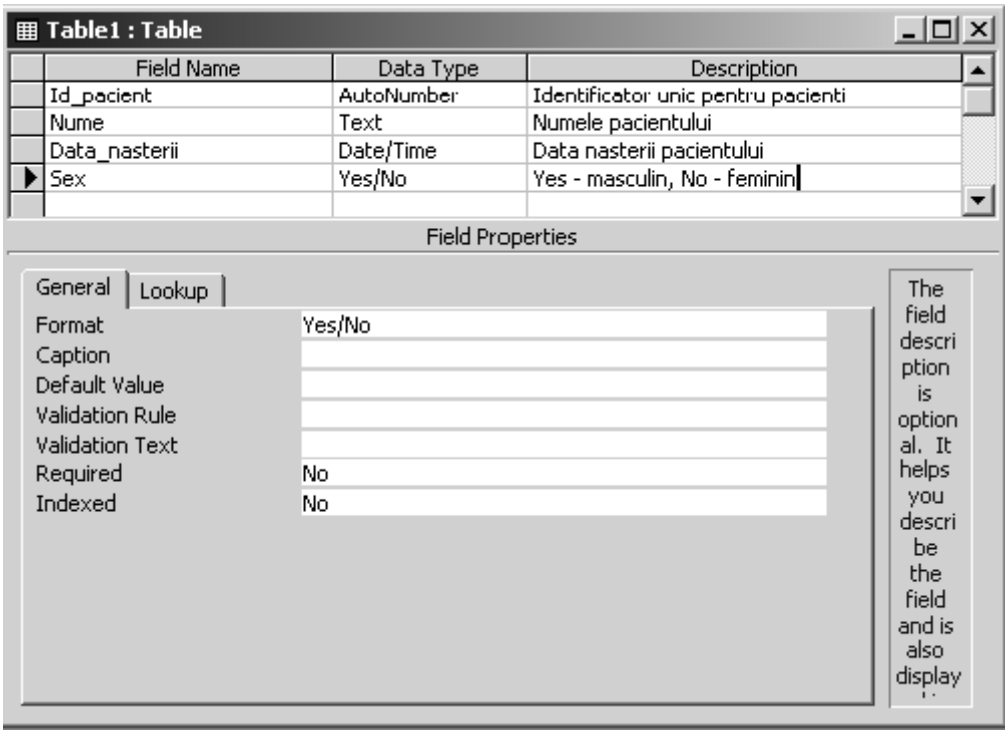
3. Pentru crearea tabelelor vom folosi opțiunea *Create table in design view* (Fereastra *Tables* din meniul *Microsoft Access*)

4. Vom crea pe rând tabelele *Pacienti* şi *Consultatii* (utilizând *Table Design*):

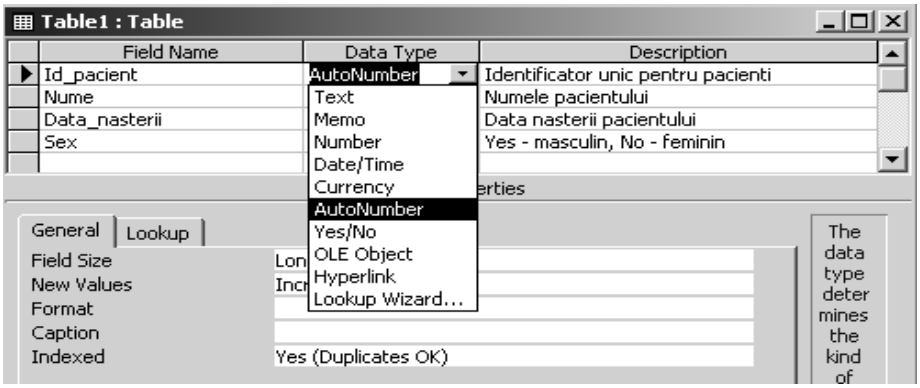


Precum se observă în figura de mai sus, în mod implementare, se introduc numele, tipul şi descrierea fiecărui câmp al tabelei (atenție: numele câmpului şi tipul acestuia sunt obligatorii!)

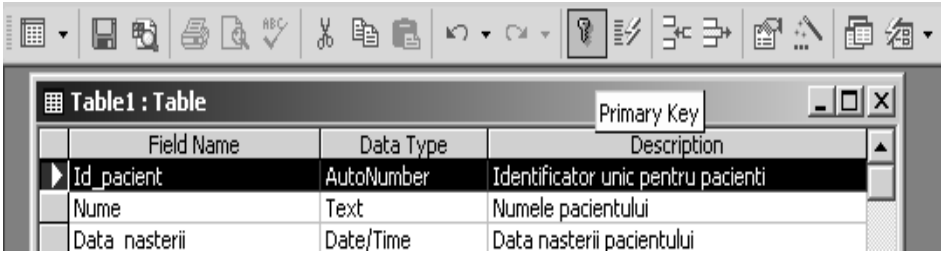
5. Structura tabelului *Pacienti*:



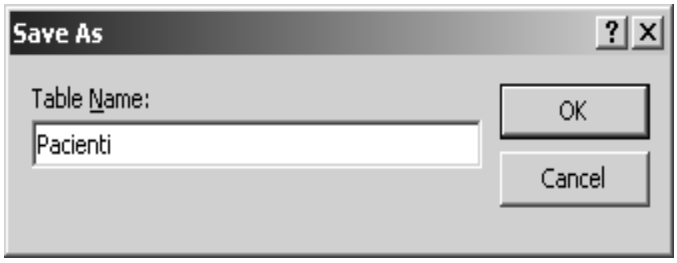
Tipul câmpului poate fi unul din cele afişate în figura de mai jos:



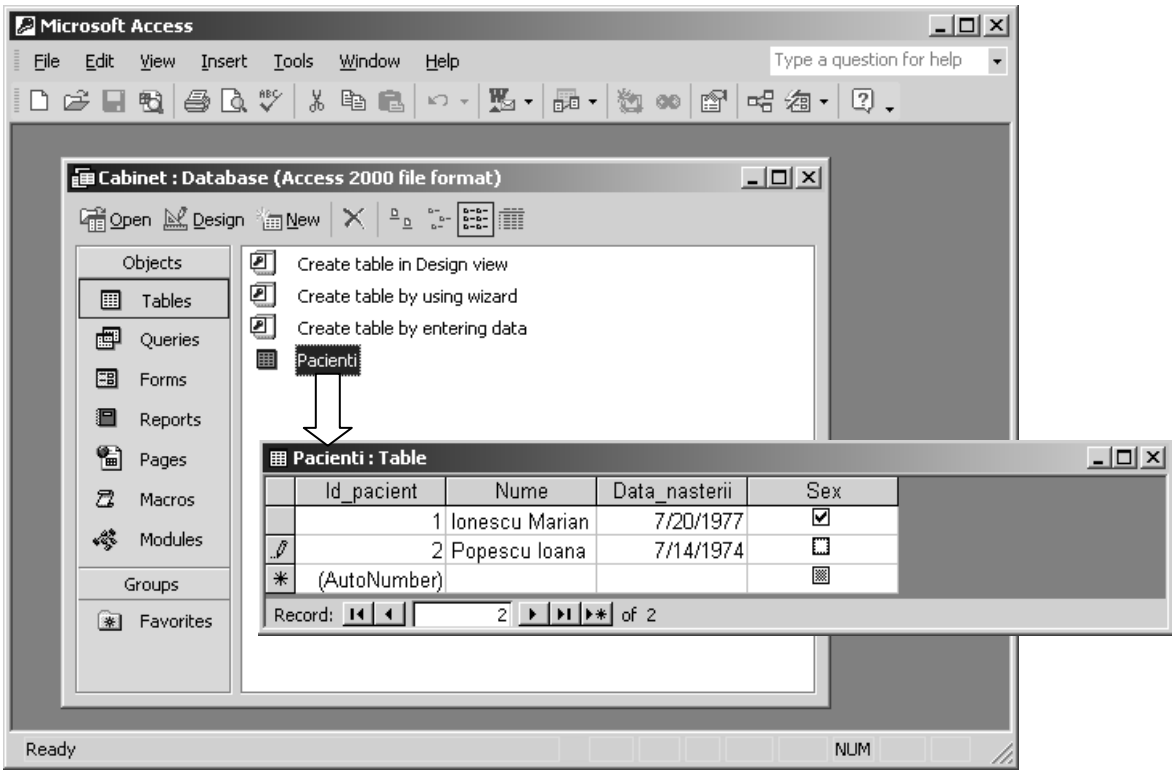
6. Pentru alegerea cheii primare a tabelului *Pacienti* se selectează câmpul care o constituie, *Id_pacient*, şi se face clic pe pictograma specifică din bara de instrumente:



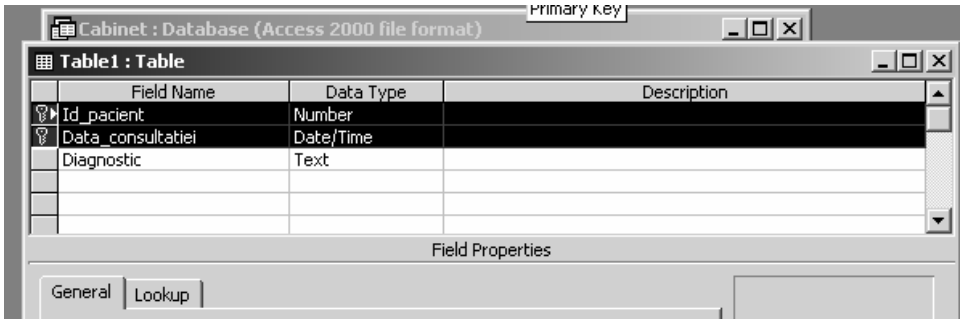
7. Pentru a putea introduce date, tabela trebuie salvată:



8. Observăm că în fereastra tabelelor a apărut o entitate nouă numită *Pacienti*. Introducerea datelor în tabela pacienților se face deschizând această tabelă:



9. Analog se creează tabela *Consultatii*:

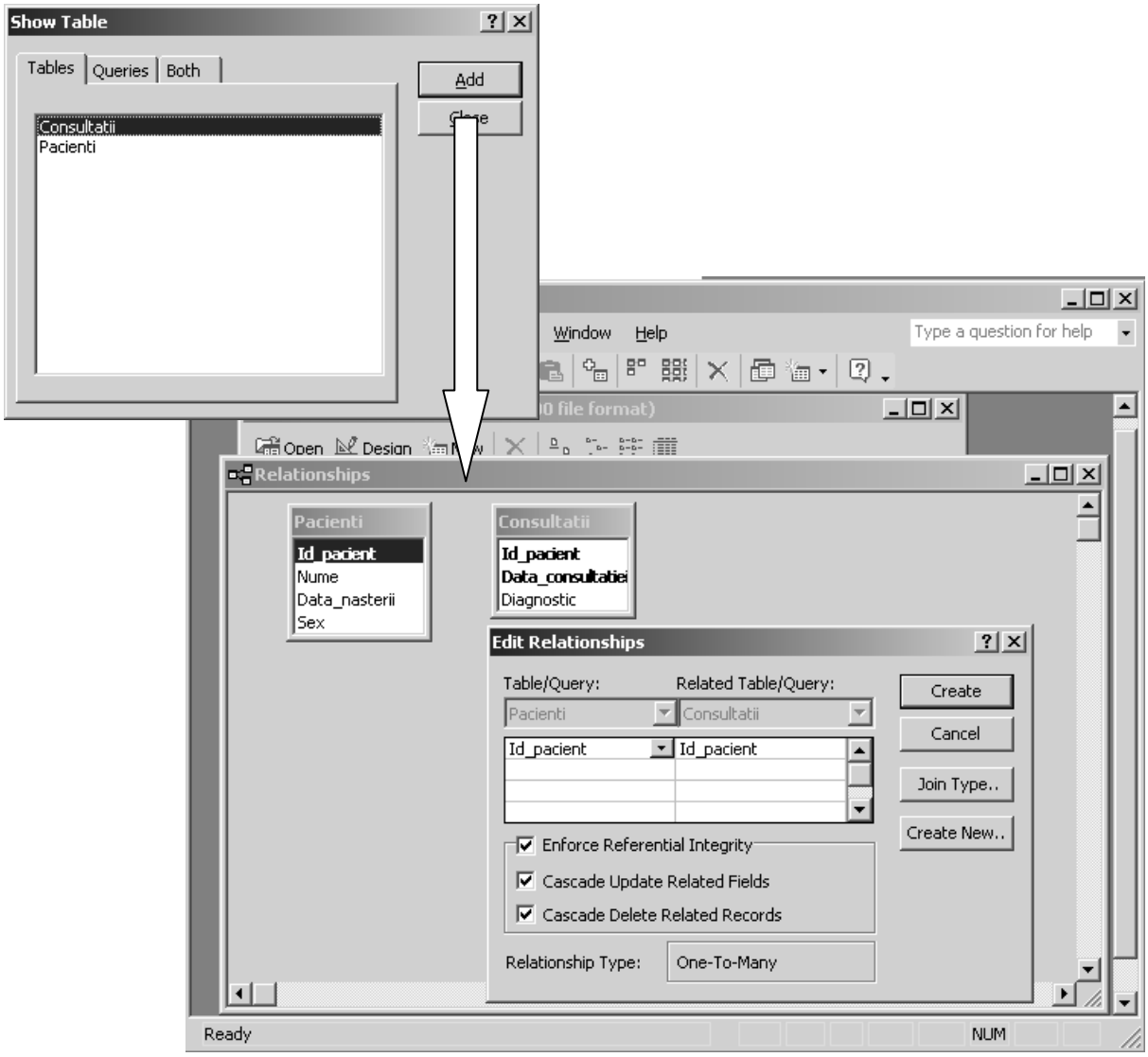


Atenție: Cheia primară a acestei tabele este o expresie formată din două câmpuri!

10. Vom crea legătura dintre tabele. Fiecărui pacient îi corespunde una sau mai multe consultații. Pentru ca baza noastră de date să fie consistentă fiecărui pacient trebuie să îi corespundă cel puțin o consultație. Legătura dintre cele două tabele va fi între câmpurile *Id_pacient* al tabelei *Pacienti* şi *Id_pacient* din tabela *Consultatii*. Această legătură este de

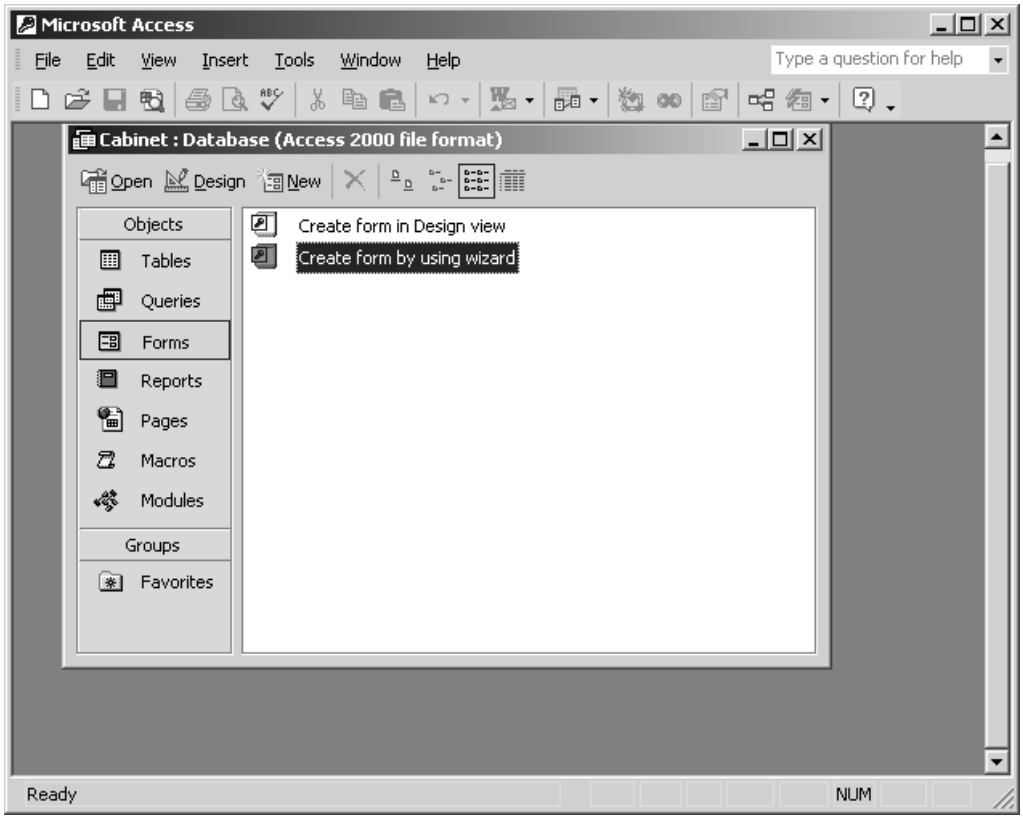
tipul *unul la mai mulți* (*one-to-many*). Realizarea relațiilor se face selectând pictograma din bara de instrumente:

11. Vom adăuga ambele tabele (figura 14) şi vom creia relația utilizând mouse-ul (*drag and drop*):



Precum se observă în figură, se selectează *Enforce Referential Integrity*. Această opțiune se referă la integritatea relației din bazei de date. Se asigură faptul că, în momentul în care o înregistrare în tabela părinte (*Pacienti*) se modifică, această modificare se va regăsi şi în tabela fiu (*Consultatii*). La fel, la ştergerea unei înregistrări din tabela *Pacienti*, se vor şterge înregistrările corespunzătoare şi din tabela *Consultatii*.

12. Odată creată relația trebuie salvată.
13. Vom crea un formular pentru introducerea datelor. Pentru aceasta vom deschide fereastra corespunzătoare formelor şi vom selecta opțiunea *Create form by using wizard*:



14. Vom urma paşii specifici:

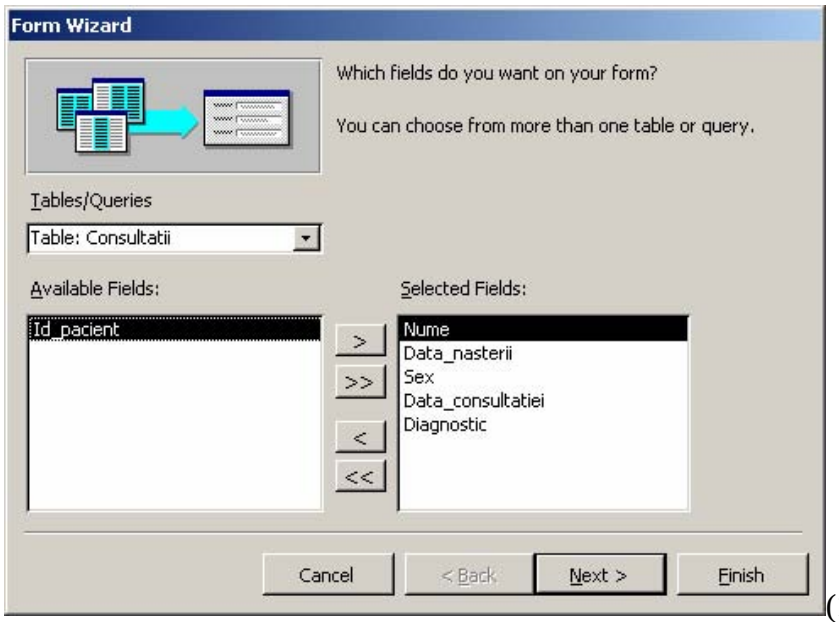
(1) Selectarea câmpurilor conținute de formular;

(2) Selectarea tipului de formular;

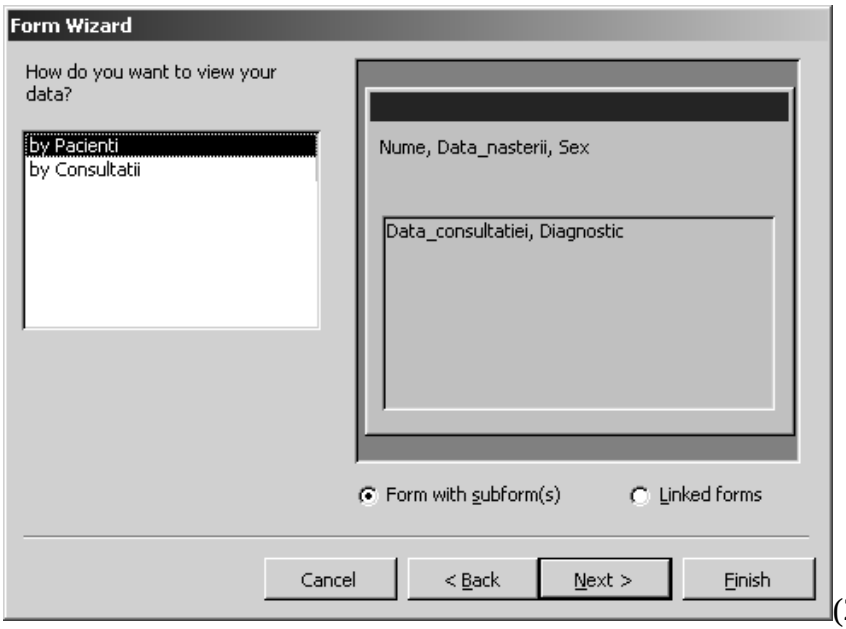
(3) Selectarea tipului subformularului în care vom introduce consultațiile;

(4) Selectarea tipului de obiecte folosite;

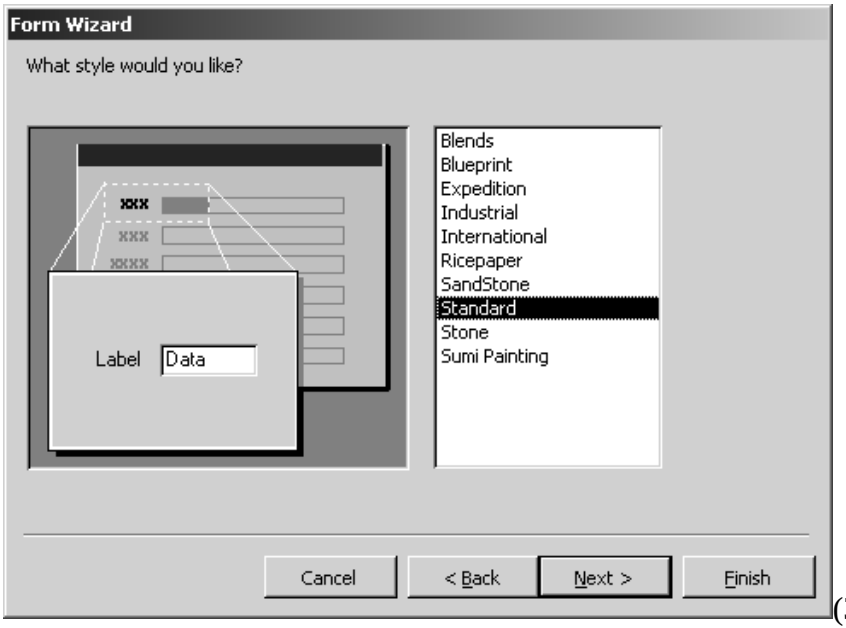
(5) Selectarea numelui formularului şi subformularului;



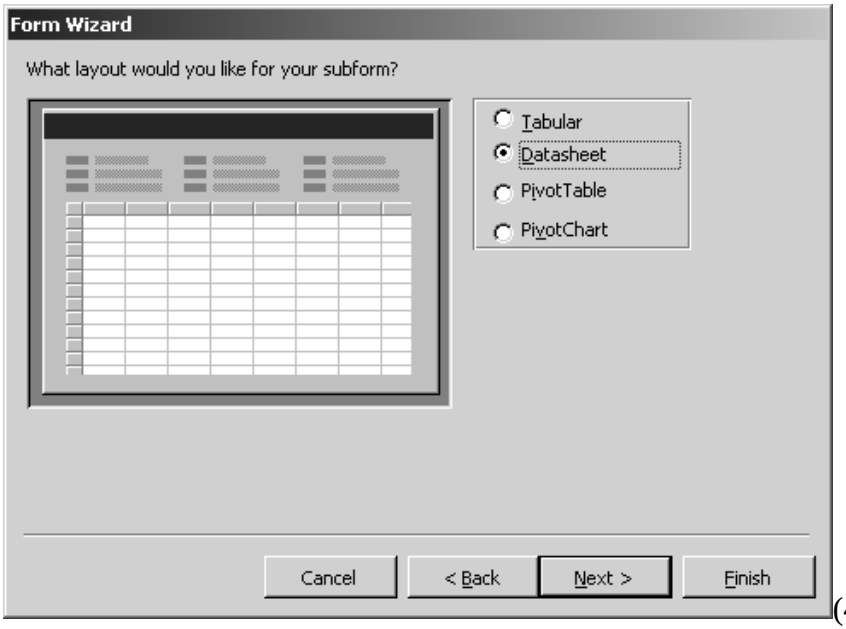
(1)


Form Wizard
How do you want to view your data?
by Pacienti
by Consultatii
Nume, Data_nasterii, Sex
Data_consultatiei, Diagnostic
Form with subform(s)
Linked forms
Cancel
< Back
Next >
Finish

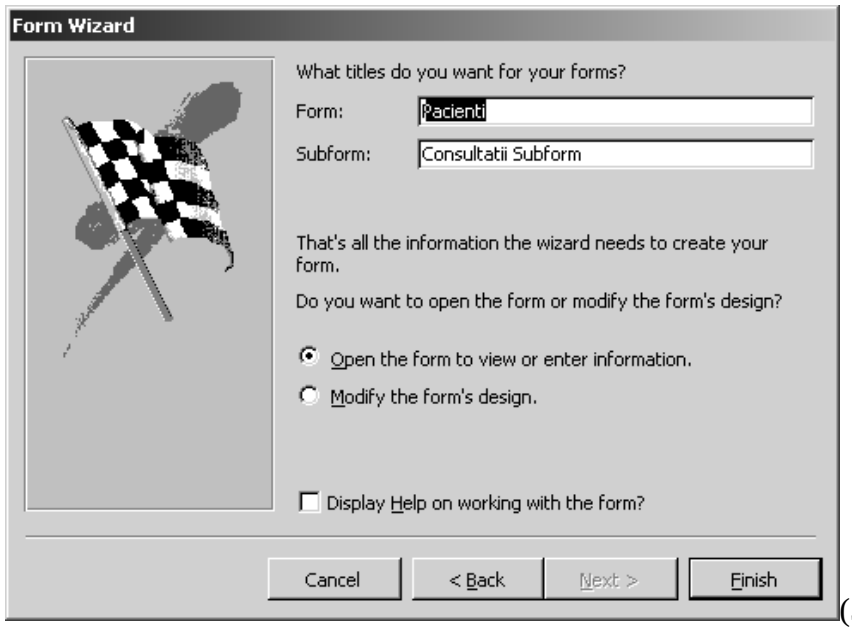
(2)


Form Wizard
What style would you like?
Label
Data
Blends
Blueprint
Expedition
Industrial
International
Ricepaper
SandStone
Standard
Stone
Sumi Painting
Cancel
< Back
Next >
Finish

(3)

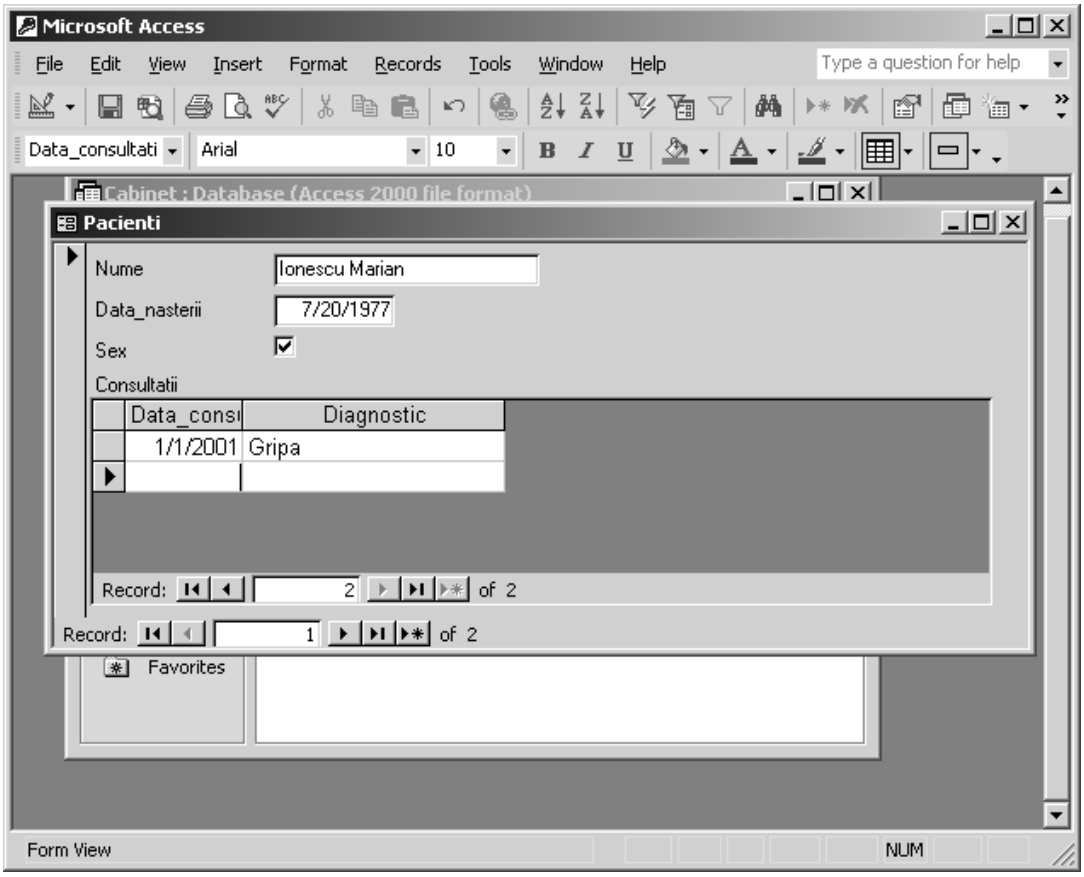

Form Wizard
What layout would you like for your subform?
Tabular
Datasheet
PivotTable
PivotChart
Cancel
< Back
Next >
Finish

(4)



(5)

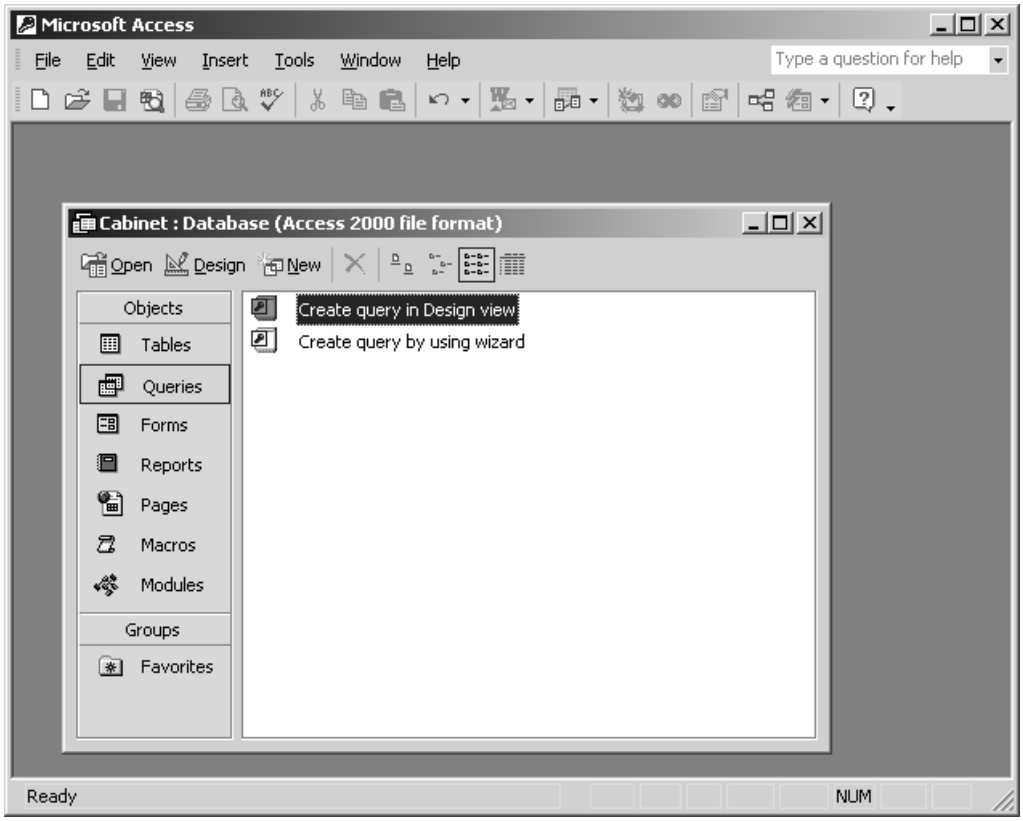
15. Forma noastră este compusă din două părți: antet (datele de identificare a pacientului) şi partea de consultații.



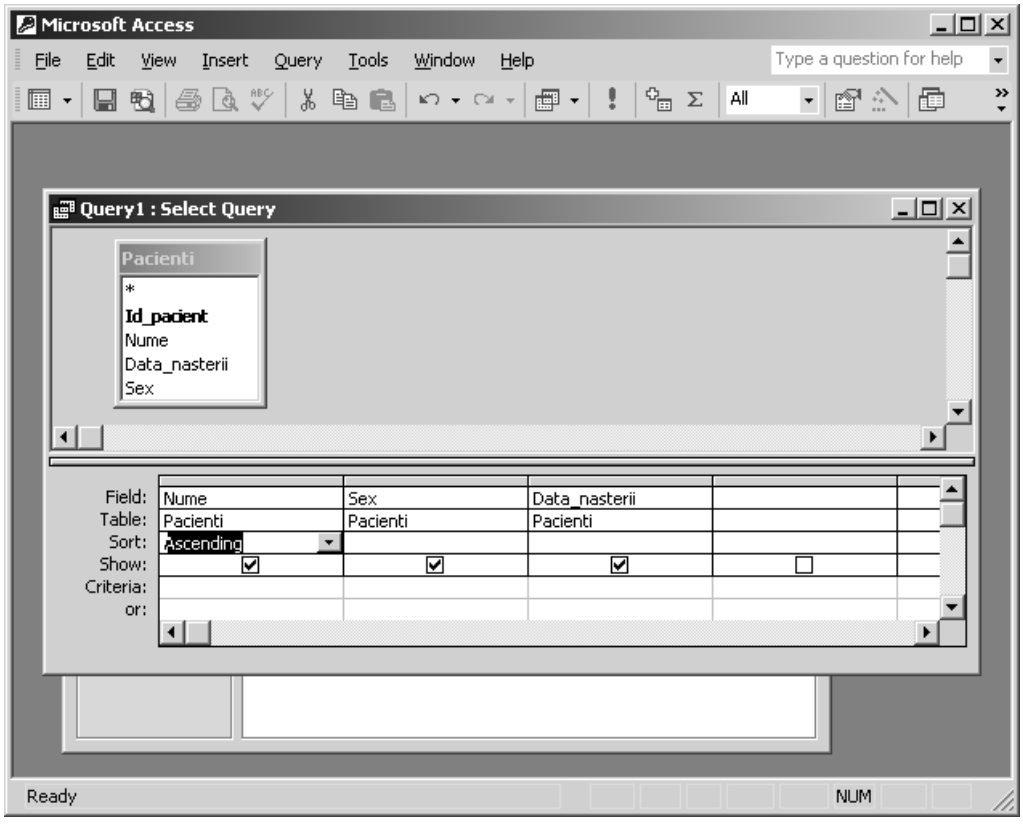
Navigarea precum şi introducerea de noi date se face cu ajutorul navigatorului ce conține, în ordine, butoane de *salt la prima înregistrare*, *înregistrarea anterioară*, *înregistrarea curentă*, *înregistrarea următoare*, *ultima înregistrare*, *adăugare*:

Ca interogări şi rapoarte ne interesează lista pacienților, precum şi lista consultațiilor făcute după o anumită dată.

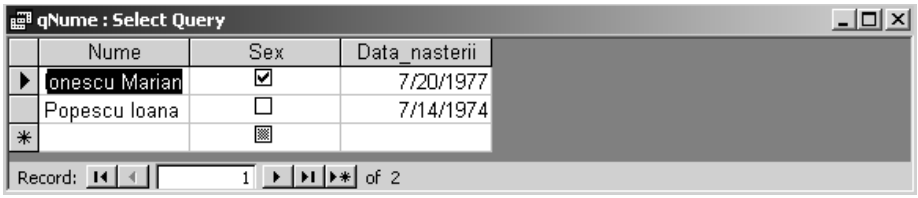
16. Vom realiza interogările utilizând instrumentele oferite de fereastra *Query*:



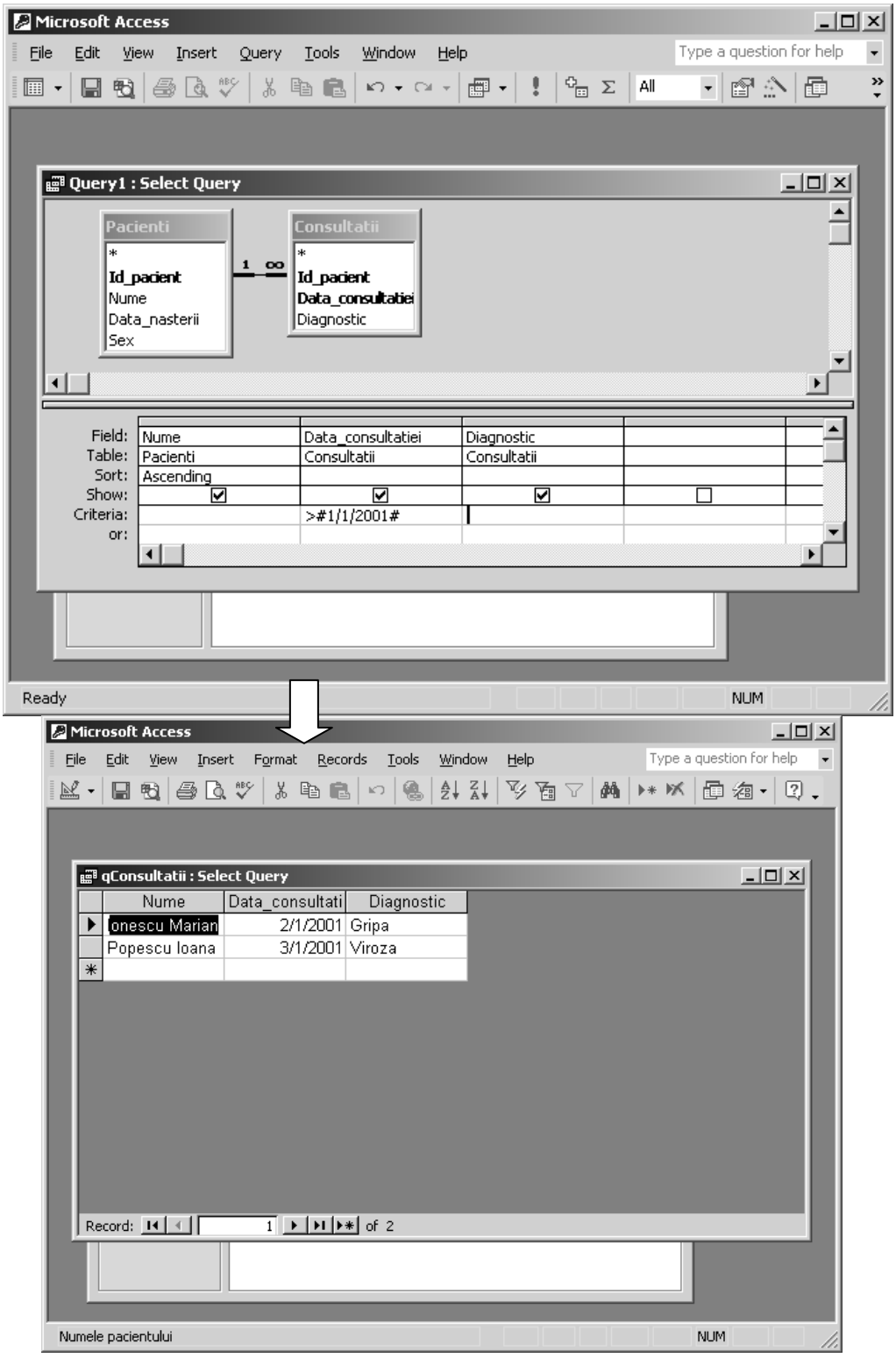
Pentru realizarea listei pacienților vom alege din tabela *Pacienti* câmpurile *nume* (sortat alfabetic), *sex* şi *data nasterii*:



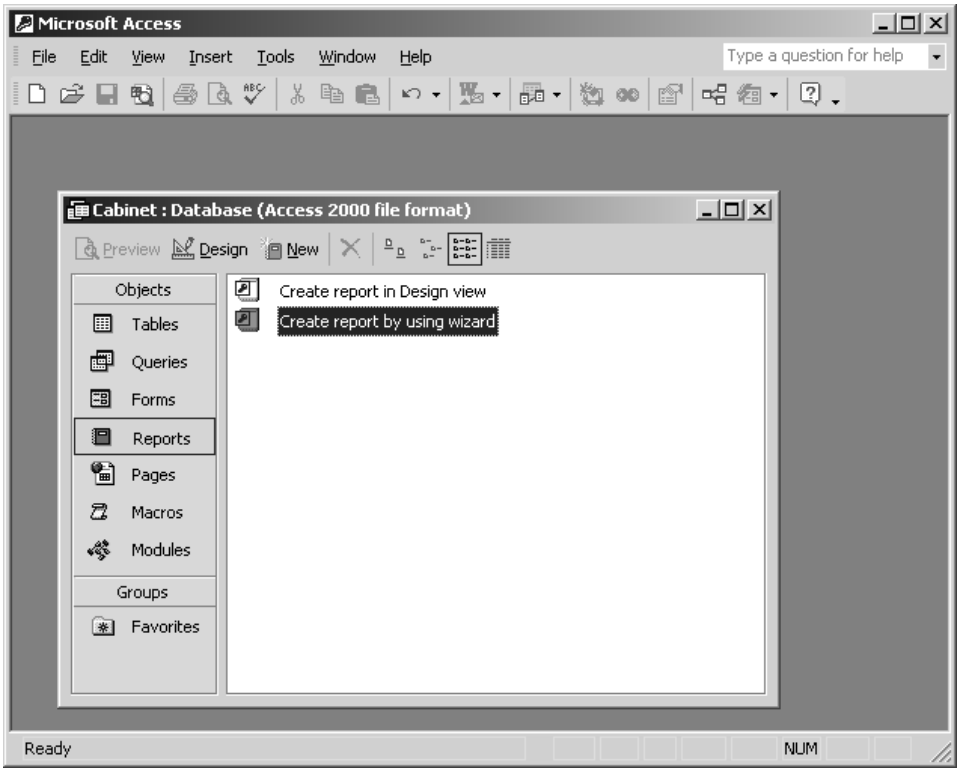
17. După salvare, interogarea poate fi executată ducând la rezultatul de mai jos:



18. Analog se realizează lista consultațiilor efectuate după o anumită dată. Vom da exemplu lista consultațiilor de la data 1.1.2001:



19. Rapoartele vor fi realizate utilizând interogările anterioare, în fereastra *Reports*:



20. Vom realiza raportul corespunzător listei de consultații, urmând pașii:

(1) Alegerea câmpurilor ce vor fi conținute în raport;

(2) Modul de prezentare a informației în raport;

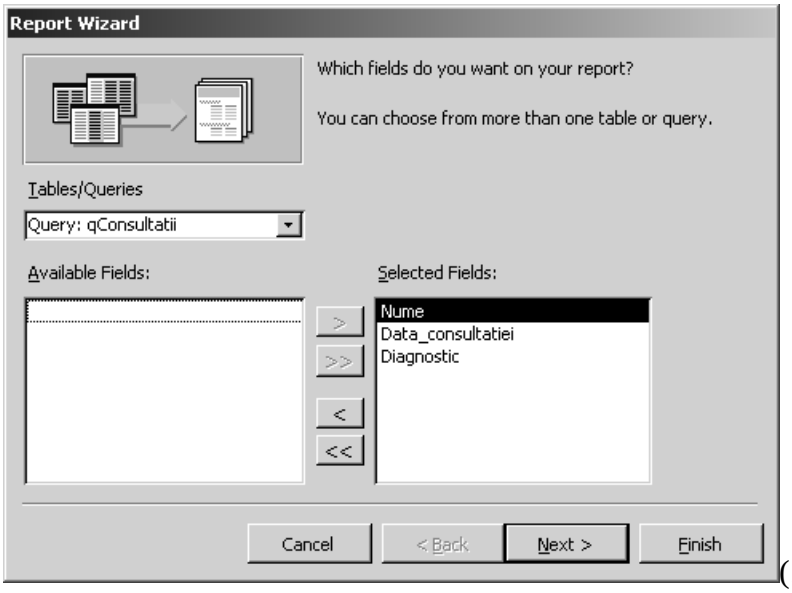
(3) Modul de grupare a informației;

(4) Modul de sortare a informației;

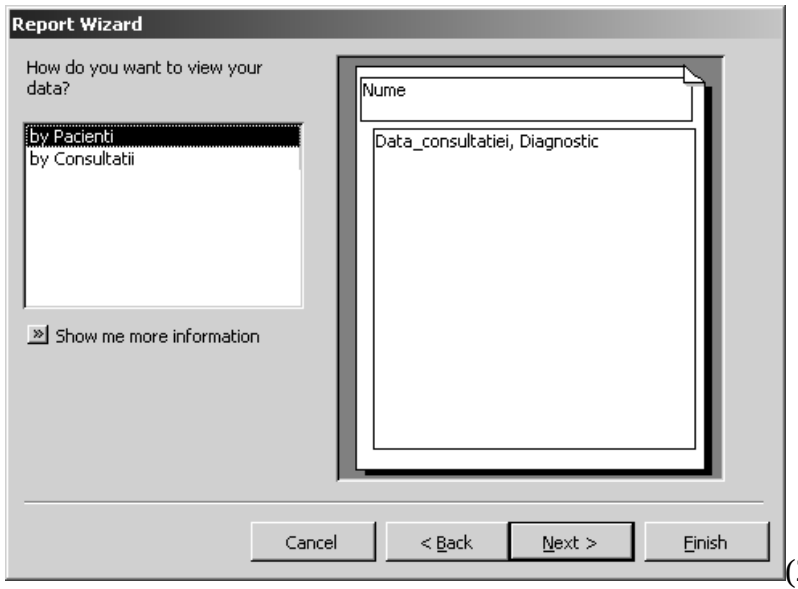
(5) Tipul raportului;

(6) Stilul titlului;

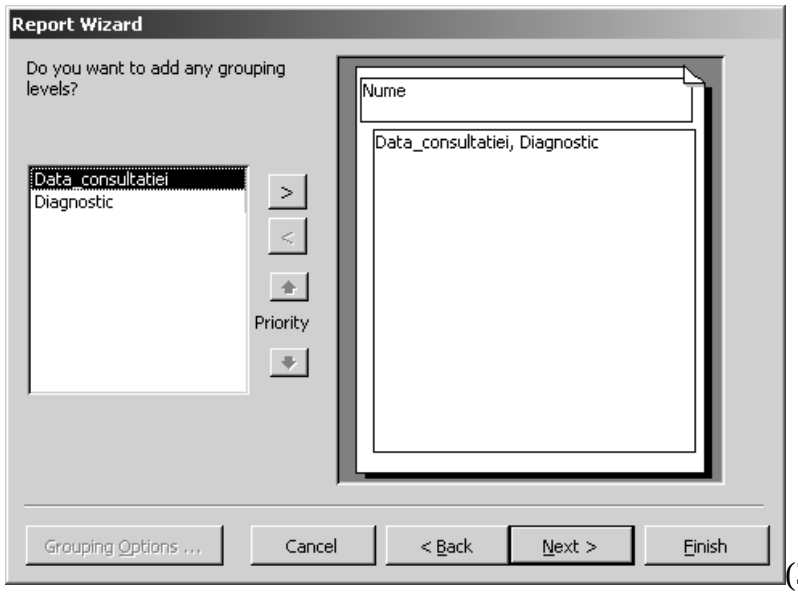
(7) Numele raportului;



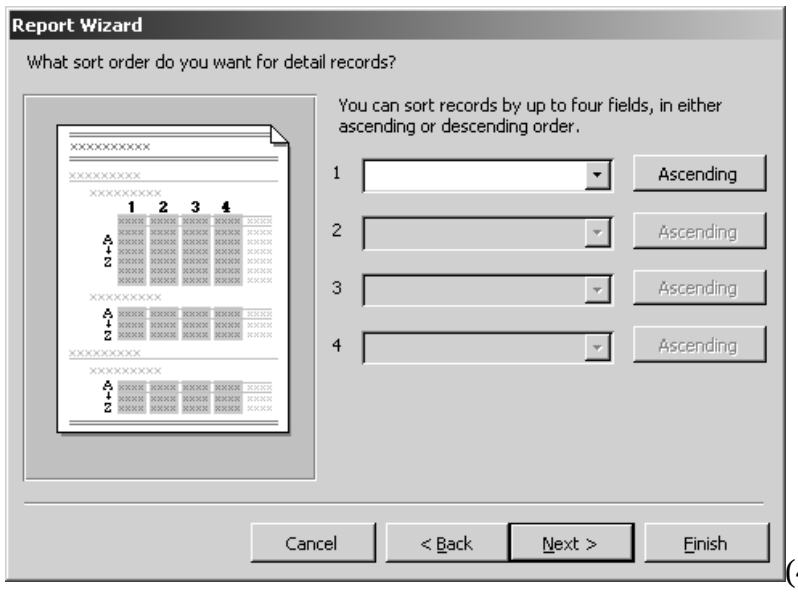
(1)


Report Wizard
How do you want to view your data?
by Pacienti
by Consultatii
Show me more information
Nume
Data_consultatiei, Diagnostic
Cancel
< Back
Next >
Finish

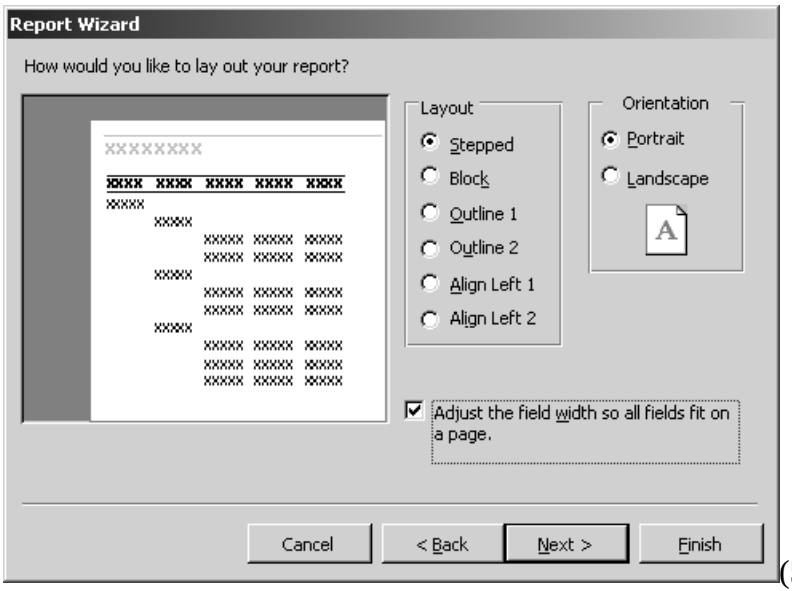
(2)


Report Wizard
Do you want to add any grouping levels?
Data_consultatiei
Diagnostic
Priority
Nume
Data_consultatiei, Diagnostic
Grouping Options ...
Cancel
< Back
Next >
Finish

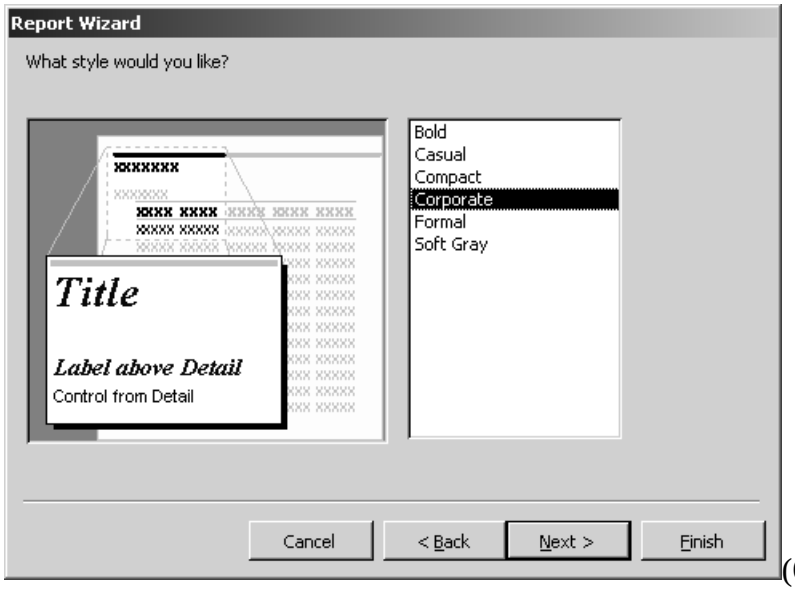
(3)


Report Wizard
What sort order do you want for detail records?
You can sort records by up to four fields, in either ascending or descending order.
1
2
3
4
Ascending
Cancel
< Back
Next >
Finish

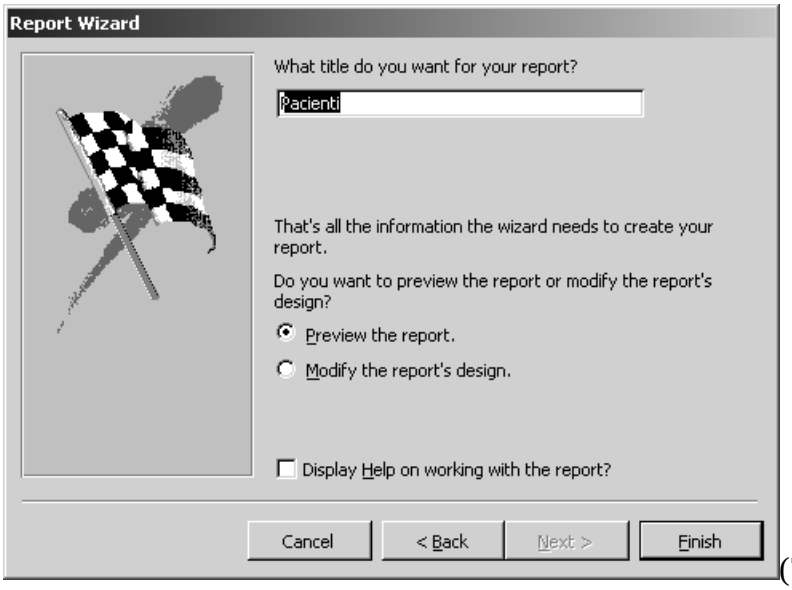
(4)


Report Wizard
How would you like to lay out your report?
Layout
Stepped
Block
Outline 1
Outline 2
Align Left 1
Align Left 2
Orientation
Portrait
Landscape
Adjust the field width so all fields fit on a page.
Cancel
< Back
Next >
Finish

(5)

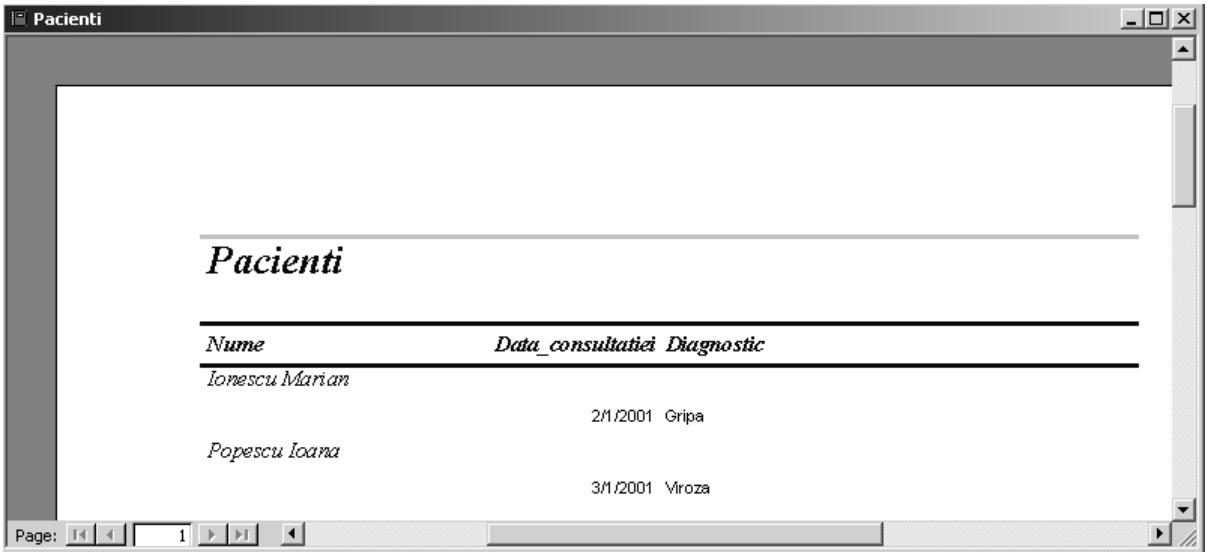

Report Wizard
What style would you like?
Bold
Casual
Compact
Corporate
Formal
Soft Gray
Title
Label above Detail
Control from Detail
Cancel
< Back
Next >
Finish

(6)


Report Wizard
What title do you want for your report?
Pacienti
That's all the information the wizard needs to create your report.
Do you want to preview the report or modify the report's design?
Preview the report.
Modify the report's design.
Display Help on working with the report?
Cancel
< Back
Next >
Finish

(7)

21. Se obține următorul rezultat:

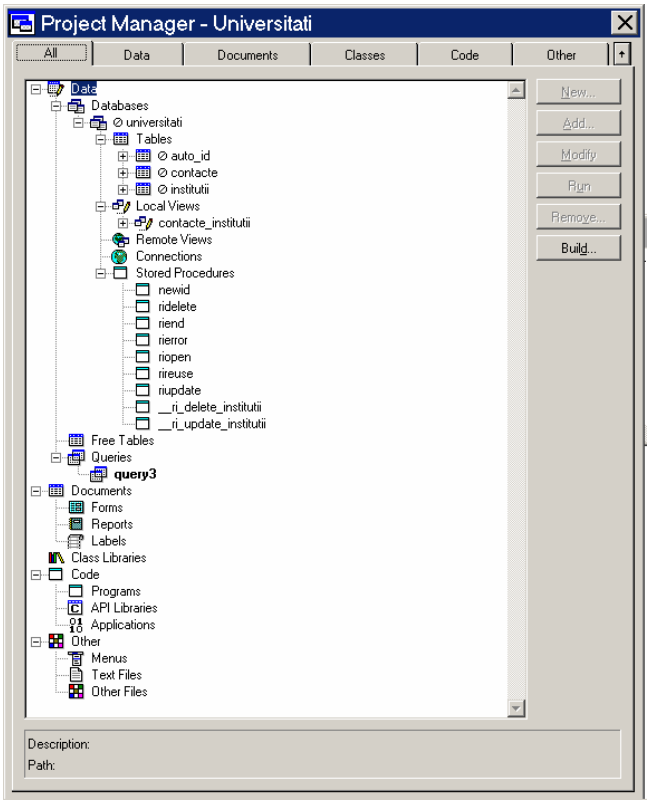


Access: probleme propuse

1. Se cere realizarea unei baze de date Access care să gestioneze activitatea unui grup de practică medicală, cuprinzând tabelele: pacienți, consultații, personal. Proiectați structura tabelelor şi a relațiile dintre ele încât să conțină datele necesare aplicației. Elaborați un raport care să furnizeze lista pacienților unui medic din grupul de practică.
2. Se cere realizarea unei baze de date Access care să gestioneze activitarea unui cabinet de medic de familie cuprinzând tabelele: Pacienți, Consultații, Medicamente compensate. Proiectați structura tabelelor şi a relațiile dintre ele încât să conțină datele necesare aplicației. Elaborați un raport care să furnizeze lista medicamentelor compensate prescrise de medicul de familie într-o anumită perioadă.
3. Se cere realizarea unei aplicații care să gestioneze activitatea unei farmacii implicând două tabele: medicamente, furnizori. Proiectați structura tabelelor şi a relațiile dintre ele încât să conțină datele necesare aplicației. Elaborați un raport care să furnizeze lista medicamentelor compensate furnizate de un furnizor dat într-o anumită perioadă.
4. Se cere realizarea unei aplicații care să gestioneze mişcarea consumabilelor într-o instituție spitalicească implicând tabelele: Consumabile, Utilizatori. Proiectați structura tabelelor şi a relațiile dintre ele încât să conțină datele necesare aplicației. Elaborați un raport care să furnizeze lista consumabilelor folosite de un utilizator dat într-o anumită perioadă.

39. Probleme propuse

A. Integrarea aplicațiilor şi Project Manager



Efectuați următorii paşi şi identificați elementele acestora:

1. Crearea unui proiect (universități);
2. Crearea unei baze de date (universități);
3. Crearea tabelelor bazei de date (instituții, contacte);
4. Introducerea datelor în tabele;
5. Modificarea tabelelor: adăugarea câmpului nr în ambele tabele;
6. Indexarea primară (instituții) şi regulară (contacte);
7. Stabilirea relației între tabele (JOIN);
8. Introducerea condițiilor de validare a datelor la nivel de câmp;
9. Introducerea condițiilor de validare a datelor la nivel de înregistrare;
10. Definirea de relații de integritate referențială;
11. Vizualizarea codului de integritate referențială generat (stored procedures);
12. Crearea unei interogări (contacte_institutii);
13. Vizualizarea codului SQL;
14. Selectarea destinației şi vizualizarea interogării (browse, table, cursor);
15. Modificarea tabelei contacte (adăugarea câmpului şi indexului poz);

16. Crearea unui grafic pe baza interogării şi câpurilor numerice nr şi poz (count(poz)/grupul nr);
17. Crearea structurii bazei de date ŞAPIAP;
18. Crearea unei vederi locale (persoanele de contact şi instituții);
19. Update Criteria şi Send SQL Updates;
20. Crearea vederii după instituții şi apoi persoane de contact;
21. Stocarea de proceduri (stored procedures) în baza de date; procedura pentru câpuri autoincrement;
22. Wizard-ul şi crearea de vederi; vedere parametrizată (persoane de contact pentru o instituție);
23. Fereastra de comenzi (command window);
24. Zone de lucru (data session);
25. Comenzi;
26. Expresii;
27. Funcții;

B. Utilizarea ferestrei de comenzi (command window) şi crearea de programe (New/Program ...)

Efectuați următorii paşi şi identificați elementele acestora:

1. Exerciții demonstrative cu comenzile pe şiruri de caractere;
2. Vizualizarea valorii returnate de *Expression Builder*

USE Contacte

GETEXPR 'Introdu conditia de localizare ' TO gcTemp;

? gcTemp

3. Crearea unui program (de exemplu cel anterior); următoarele comenzi sunt utile:

- pentru a crea un program: *New/Program,New File* sau comanda *modify command*;
- pentru a salva un program: *File/Save*;
- pentru a deschide un program: *File/Open/File type: program/Open* sau *modi comm <nume_program>*; se poate face şi *modi comm ?* când se activează o fereastră de dialog;
- pentru a executa un program: *Program/Do...* sau cu comanda *do <nume_program>*;

4. Exerciții cu comenzile browse, locate, replace;
5. Program cu replace;
6. Exemple cu variabile; exemplul 1 corect:

- *local gdDate*
- *STORE DATE( ) TO gdDate*

7. Exerciții cu matrici
8. Exerciții cu IF, CASE, SCAN, FOR, DO WHILE;
9. Exemplu cu o procedură într-un program:

```
Do myproc with "aaa"
PROCEDURE myproc( cString )
  ? "myproc" + cString
ENDPROC
```

10. Exemplu cu o funcție într-un program:

```
? plus2saptamani(date())
FUNCTION plus2saptamani
PARAMETERS dDate
  RETURN dDate + 14
ENDFUNC
```

11. Exemplu de program cu o procedură cu 3 parametrii

```
GETEXPR 'Introdu o expresie: '  TO gcTemp
DO procedura WITH DATE(), gcTemp, 10
PROCEDURE procedura( dData, cSir, nOriTipar )
  FOR nCnt = 1 to nOriTipar
    ? DTOC(dData) + " " + cSir + " " + STR(nCnt)
  ENDFOR
ENDPROC
```

12. *Report Wizard*; raportul persoanelor de contact din baza de date *universitati*;
13. *Report Designer*; realizarea unui raport cu totaluri pe categorii;
14. *Quick Report*; raport dintr-o vedere; varianta 1 şi varianta 2;
15. *Labels*; etichete simple şi etichete din vedere cu parametru;
16. Generarea unei interogări din program folosind macrosubstituția;
17. *Form Wizard*; formular pentru parcurgerea şi modificarea datelor pentru instituțiile existente şi pentru introducerea datelor pentru o nouă instituție în baza de date *universitati.dbc*; execuția formularului;
18. Crearea unui raport pentru imprimarea informațiilor din formular; raportul *institutii.frx*;
19. Modificarea structurii tabelei *institutii.dbf*;
20. Modificarea formularului *institutii.scx*; execuția formularului; adăugarea unei instituții;
21. Tipărirea informațiilor din formular cu ajutorul raportului institutii.frx;
22. *One-to-many Form Wizard*; Formular pentru parcurgerea şi modificarea datelor simultan pentru instituțiile şi persoanele de contact din baza de date *universitati.dbc*;

23. *Form Builder*; formular pentru modificarea unei persoane de contact;
24. *Form Designer*; crearea formularului din aplicația 4;
25. Execuția formularului;
26. Setarea atributelor la *Tab Order* şi *Forms/Maximum design area*;

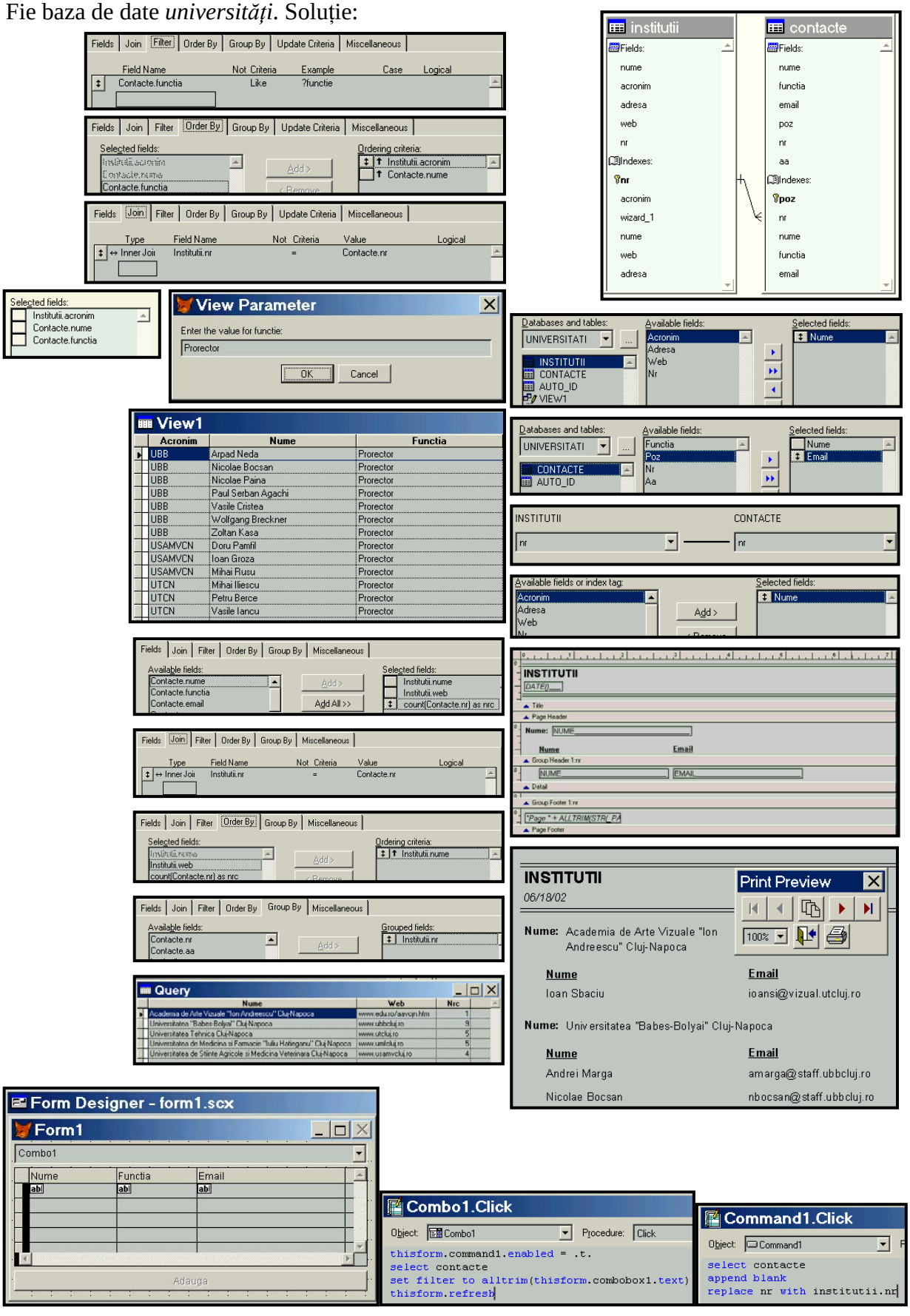
40. Test de cunoştințe

Se consideră o bază de date care conține două tabele relatate pe baza unei relații de tipul 1 la n (One to Many Relationship). Tabela cu cheia primară a relației se va numi tabela părinte iar tabela cu cheia străină a relației se va numi tabela fiu.

1. Să se realizeze o vedere cu parametru care să conțină un câmp de identificare din tabela părinte (de preferință o cheie candidată) şi cel puțin două câmpuri din tabela fiu în la care căutarea să se facă într-un câmp din tabela fiu după o valoare de tip caracter. Ordonarea să se facă după valorile câmpului din tabela părinte şi apoi după valorile unui câmp din tabela fiu.
2. Să se realizeze o interogare pe baza tabelei părinte şi tabelei fiu care să conțină două câmpuri din tabela părinte şi un câmp care să conțină numărul înregistrărilor relatate pe baza relației din tabela fiu pentru fiecare înregistrare din tabela părinte. Să se ordoneze înregistrările după un câmp din tabela părinte. Să se denumească câmpul ce conține numărul înregistrărilor din tabela fiu nrc.
3. Să se realizeze un raport care să conțină un câmp din tabela părinte şi două câmpuri din tabela fiu mai puțin cheile străine şi primare şi să se grupeze informațiile după cheia primară din tabela părinte.
4. Să se realizeze un formular care să permită introducerea unei înregistrări în tabela fiu pe baza unei selecții a înregistrării relatate în tabela părinte cu ajutorul unei liste combo.
5. Să se realizeze un meniu pentru o aplicație.
6. Întrebări:
    a. Ce este o cheie primară şi o cheie străină;
    b. Ce este o tabelă părinte şi o tabelă fiu;
    c. Ce este un index primar şi ce este un index regular;
    d. Caracterizați o tabelă, o vedere, o interogare şi un cursor;
    e. Cum implementați o relație m la n într-o bază de date;
    f. Care sunt etapele unei conectări la un server de baze de date;
    g. Care este diferența între un meniu sistem şi un meniu contextual;
    h. Cum se poate realiza un help pentru o aplicație;

41. Model de soluție pentru test

Fie baza de date *universități*. Soluție:



III. Index de cuvinte cheie

IV. Bibliografie

[Lucrări de referință în domeniu]

1. Donald Knuth, The art of computer programming. Volume 1: Fundamental algorithms. Third Edition, Addison-Wesley, ISBN 0-201-89683-4, 1997, 650 p.
2. Donald Knuth, The art of computer programming. Volume 2: Seminumerical Algorithms. Third Edition, ISBN 0-201-89684-2, 1997, 762 p.
3. Donald Knuth, The art of computer programming. Volume 3: Sorting and Searching. Second Edition, Addison-Wesley, ISBN 0-201-89685-0, 1998, 780 p.
4. Marcus Egger, Advanced Object Oriented Programming With Visual Foxpro 6.0, Hentzenwerke Publishing, ISBN 0-96550-938-9, 1999, 416 p.
5. Nigel McFarlane, Rapid Application Development with Mozilla, Prentice Hall, Pearson Education, ISBN 0-13-142343-6, 2003, 800 p.
6. Stefan Stanczyk, Bob Champion, Theory and Practice of Relational Databases, Taylor & Francis, ISBN 0-415-24701-2, 2001, 272 p.
7. Terry Halpin, Information Modeling and Relational Databases: From Conceptual Analysis to Logical Design, Academic Press, ISBN 1-55860-672-6, 2001, 761 p.

[Lucrări ale autorilor]

1. Carmen Elena STOENOIU, Lorentz JÄNTSCHI, Sorana Daniela BOLBOACĂ, Computer-Based Testing in Physical Chemistry Topic, Third Humboldt Conference on Computational Chemistry, June 24-28, Conference Proceedings, ISBN 954-323-199-0 then 978-954-323-199-7, p. 94, Bulgaria, 2006, Varna
2. Delia Maria GLIGOR, Lorentz JÄNTSCHI, Periodic System of Elements Database and Its Applications, Oradea University Annals, Chemistry Fascicle, Oradea Univeristy Press, Oradea, Issue 12, 180-194, 2005, ISSN 1224-7626
3. Elena ZAHARIEVA-STOYANOVA, Lorentz JÄNTSCHI, Application of Software Data Dependency Detection Algorithm in Superscalar Computer Architecture, International Conference on Computer Systems and Technologies (e-Learning), June 19-20, work published in volume CompSysTech'2003 (ISBN 954-9641-33-3) at p. II.61-II.66, Bulgaria, 2003, Sofia
4. Elena ZAHARIEVA-STOYANOVA, Lorentz JÄNTSCHI, Detection of Software Data Dependency in Superscalar Computer Architecture Execution, CSCS14 International Conference, July 2-5, work published in volume II "CSCS-15 14-th International

Conference on Control Systems and Computer Science" (ISBN 973-8449-17-0, ISBN 973-8449-19-7) at p. 351-356, Romania, 2003, Bucureşti

5. Horea Iustin NAŞCU, Lorentz JÄNTSCHI, Multiple Choice Examination System 1. Database Design and Implementation for General Chemistry, Leonardo Journal of Sciences, AcademicDirect, Internet, Issue 5, 18-33, 2004, ISSN 1583-0233
6. Horea Iustin NAŞCU, Lorentz JÄNTSCHI, Multiple Choice Examination System 2. Online Quizzes for General Chemistry, Leonardo Electronic Journal of Practices and Technologies, AcademicDirect, Internet, Issue 5, 26-36, 2004, ISSN 1583-1078
7. Lorentz JÄNTSCHI, Dana AVRAM, Internet, Local Databases and Browsers, International Conference on Quality Control, Automation and Robotics, May 23-25, work published in volume "AQTR Theta 13" (ISBN 973-9357-11-1) at p. 516-521, Romania, 2002, Cluj-Napoca
8. Lorentz JÄNTSCHI, Delphi Client - Server Implementation of Multiple Linear Regression Findings: a QSAR/QSPR Application, Applied Medical Informatics, SRIMA, Cluj-Napoca, Issue 15, 48-55, 2004, ISSN 1224-5593
9. Lorentz JÄNTSCHI, Free Software Development. 1. Fitting Statistical Regressions, Leonardo Journal of Sciences, AcademicDirect, Internet, Issue 1, 31-52, 2002, ISSN 1583-0233
10. Lorentz JÄNTSCHI, I386-Based Computer Architecture and Elementary Data Operations, Leonardo Journal of Sciences, AcademicDirect, Internet, Issue 3, 9-23, 2003, ISSN 1583-0233
11. Lorentz JÄNTSCHI, Installing and Testing a Server Operating System, Leonardo Electronic Journal of Practices and Technologies, AcademicDirect, Internet, Issue 3, 1-30, 2003, ISSN 1583-1078
12. Lorentz JÄNTSCHI, Mariana MARCU, Sorana Daniela BOLBOACĂ, SQL Application for Secondary School Leaving Examination, UNITECH'03 International Scientific Conference, November 21-22, work published in volume I "ISC UNITECH'03 GABROVO Proceedings" (ISBN 954-683-167-0) at p. 258-262, Bulgaria, 2003, Gabrovo
13. Lorentz JÄNTSCHI, Mihaela Ligia UNGUREŞAN, Parallel processing of data. C++ Applications, Oradea University Annals, Mathematics Fascicle, Oradea University Press, Oradea, VIII, 105-112, 2001, ISSN 1221-1265
14. Lorentz JÄNTSCHI, Sorana BOLBOACĂ, Organizing Guidelines Models and Clinical Practice Guidelines, 11th International Symposium for Health Information Management Research, July 14-16, Proceedings, ISBN 0-7703-9016-1, p. 328-338, Nova Scotia, CA, 2006, Halifax

15. Lorentz JÄNTSCHI, Sorana Daniela BOLBOACĂ, Computer Aided System for Student's Knowledge Assessment, The 10th World Multi-Conference on Systemics, Cybernetics and Informatics, July 16-19, Proceedings, ISBN 980-6560-65-5 (Collection) && ISBN 980-6560-65-3 (Volume 1), p. 97-101, Florida, U.S.A, 2006, Orlando
16. Lorentz JÄNTSCHI, Sorana Daniela BOLBOACĂ, Installation and Configuration Issues about FreeBSD Operating System, A&QT-R 2004 (THETA 14) 2004 IEEE-TTTC - International Conference on Automation, Quality and Testing, Robotics, May 13-15, volume "II" (ISBN 973-713-047-2) at p. 251-256, Romania, 2004, Cluj-Napoca
17. Lorentz JÄNTSCHI, SQL by Example. 1. Application for High School Bachelor Examination, Leonardo Electronic Journal of Practices and Technologies, AcademicDirect, Internet, Issue 2, 20-36, 2003, ISSN 1583-1078
18. Mădălina VĂLEANU, Building Biological Databases, Applied Medical Informatics, Srima, Cluj-Napoca, Nr. 3-4/2002, pag. 67-72, 2002, ISSN 1224-5593
19. Mădălina VĂLEANU, Global Interity Preservation in Medical Distributed Databases, Applied Medical Informatics, Srima, Cluj-Napoca, Nr. 3-4/2003, pag. 36-45, 2003, ISSN 1224-5593
20. Mădălina VĂLEANU, Integrity Aspects in Database, Applied Medical Informatics, Srima, Cluj-Napoca, Nr. 3-4/2004, p. 9-15, 2004, ISSN 1224-5593
21. Mădălina VĂLEANU, Integrity in Distributed Databases, Applied Medical Informatics, Srima, Cluj-Napoca, Nr. 1-2/2003, pag. 3-8, 2003, ISSN 1224-5593
22. Monica ŞTEFU, Mihaela Ligia UNGUREŞAN, Lorentz JÄNTSCHI, Free Software Development. 2. Chemical Database Management, Leonardo Electronic Journal of Practices and Technologies, AcademicDirect, Internet, Issue 1, 69-76, 2002, ISSN 1583-1078
23. Sorana BOLBOACĂ, Horațiu COLOSI, Tudor DRUGAN, Andrei ACHIMAŞ, Ştefan ŢIGAN, Elements of Medical Informatics and Biostatistics, SRIMA Publishing House, Cluj-Napoca, Romania, 211 pages, 2003, ISBN 973-85285-0-X.
24. Sorana BOLBOACĂ, Lorentz JÄNTSCHI, Computer-Assisted Training and Evaluation System in Evidence-Based Medicine, 11th International Symposium for Health Information Management Research, July 14-16, Proceedings, ISBN 0-7703-9016-1, p. 220-226, Nova Scotia, CA, 2006, Halifax
25. Sorana BOLBOACĂ, The Computer Revolution in Neurobiology, Applied Medical Informatics, SRIMA, Cluj-Napoca, Issue 11, 32-8, 2002, ISSN 1224-5593
26. Sorana Daniela BOLBOACĂ, Lorentz JÄNTSCHI, Andrei ACHIMAŞ CADARIU, SQL by Example. 2. PHP and MySQL Web Application based on Tanner-Whitehouse

Standard, Leonardo Electronic Journal of Practices and Technologies, AcademicDirect, Internet, Issue 2, 37-52, 2003, ISSN 1583-1078

27. Sorana Daniela BOLBOACĂ, Lorentz JÄNTSCHI, Andrei ACHIMAŞ CADARIU, Relational Information in Medicine: A Challenge, Roentgenologia & Radiologia, Bulgarian Association of Radiology, Sofia, Issue XLIV(1), 22-25, 2005, ISSN 0486-400X
28. Sorana Daniela BOLBOACĂ, Lorentz JÄNTSCHI, Andrei ACHIMAŞ CADARIU, PHP and MySQL Medical Application Based on Tanner Whitehouse Standard, UNITECH'03 International Scientific Conference, November 21-22, work published in volume I "ISC UNITECH'03 GABROVO Proceedings" (ISBN 954-683-167-0) at p. 304-308, Bulgaria, 2003, Gabrovo
29. Sorana Daniela BOLBOACĂ, Lorentz JÄNTSCHI, Assessment of an Computer Based Curriculum in Evidence-Based Medicine, The 10th World Multi-Conference on Systemics, Cybernetics and Informatics, July 16-19, e-Proceedings, ISBN 980-6560-92-2 (CD), paper #3 e-KCC (THEME: Other theme or topic in the domain of KCC 2006), 5 p., Florida, U.S.A, 2006, Orlando
30. Sorana Daniela BOLBOACĂ, Lorentz JÄNTSCHI, Carmencita DENEŞ, Andrei ACHIMAŞ CADARIU, Skeletal Maturity Assessment Client-Server Application, Roentgenologia & Radiologia, Bulgarian Association of Radiology, Sofia, Issue XLIV(3), 189-193, 2005, ISSN 0486-400X
31. Tudor DRUGAN, Cosmina BONDOR, Sorana BOLBOACĂ, Tudor CĂLINICI, Horaţiu COLOSI, Ramona GĂLĂTUŞ, Dan ISTRATE, Mădălina VĂLEANU, Andrei ACHIMAŞ, Ştefan ŢIGAN, Practical Applications of Medical Informatics and Statistics (in Romanian), ALMA MATER Publishing House, Cluj-Napoca, Romania, 198 pages, 2002, ISBN 973-85354-5-X.
32. Tudor DRUGAN, Sorana BOLBOACĂ, Horaţiu COLOSI, Ramona GĂLĂTUŞ, Tudor CĂLINICI, Dan ISTRATE, Cosmina BONDOR, Mădălina VĂLEANU, Andrei ACHIMAŞ, Ştefan ŢIGAN, Applied Medical Informatics (in Romanina), SRIMA Publishing House, Cluj-Napoca, Romania, 204 pages, 2003, ISBN 973-8296-09-9.
33. Tudor DRUGAN, Sorana BOLBOACĂ, Tudor CĂLINICI, Dan ISTRATE, Horaţiu COLOSI, Ramona GĂLĂTUŞ, Cosmina BONDOR, Mădălina VĂLEANU, Andrei ACHIMAŞ, Ştefan ŢIGAN, Applications of Medical Informatics and Biostatisctics (in Romanian), SRIMA Publishing House, Cluj-Napoca, Romania, 193 pages, 2004, ISBN 973-85285-3-4.

V. Abstract

The work *Rapid Programming of Relational Databases Applications* is the result of didactic experience of over ten years in field of creating and exploiting relational databases, of implementing of applications dedicated to fields of: scientific research (databases for chemistry and medicine), archiving (databases with scientific publications), management (materials and money) and never the less, education.

The work is addressed to which ones which want to acquire skills and abilities to use the applications dedicated to databases, and especially of applications from *Rapid Application Development (RAD)* category. From this category of applications, most popular is FoxPro; this is also the application which are discussed most deep in the work.

From the programming style point of view, the work offer the *Microsoft* solution, being discussed three applications dedicated to databases provided by *Microsoft* trust. In the increasing order of programming environment provided, these are: *Excel*, *Access*, and *Visual FoxPro*. Related to the versions of discussed applications, these are neither the last ones from the market, neither the first ones. Regarding strictly to Visual FoxPro, discussion are made exemplifying with applications for version 6 (release year 1998), discussion being then perfectly valid for versions 5 and 7.

The axis of didactical exposition of work is directed from *problem* to *solution*, passing through *mathematical model*, *algorithm*, and *implementation*.

The fundamental theoretical knowledge of databases are not skipped from scientific exposition of work; contrary, are deeply discussed the problems of storage at physical as well as logical level, the problems of consistency and integrity are exemplified through the work, and fundamental problems of security, coherency, restrictions and transactions are treated (because of its apart specificity) near to the end of the work (section *Special Chapters of Databases*).

The work has a profound forming character. The abilities which work wants to create are: projecting, implementation, normalizing and assuring of referential integrity of a relational database and implementation of information management applications; the use of controls provided by a visual programming environment for rapid application development; realizing of client-server complex for management of distributed databases. Never the less, are placed the software engineering, the approaching technique for proposed problem from top to down and from bottom to up (*top-down* and *bottom-up*) by using of well known Latin dictum *divide et impera* (*divide and overrule*).

VI. Contents

AcademicDirect
ISBN 973-86211-5-1
ISBN13 980-973-86211-5-1

Academic Pres
ISBN 973-86211-5-1
ISBN13 980-973-86211-5-1