Prefaţă

Bazele de date constituie, astăzi, un univers fascinant. De la apariţia ei, când o bază de date era concepută ca un fişier, până în prezent, noţiunea de bază de date a suferit enorme schimbări în semantică. Pe de o parte, aceste schimbări sunt rodul îmbogăţirii cunoştinţelor teoretice referitoare la reprezentarea şi regăsirea datelor, iar pe de altă parte, realizarea a numeroase sisteme de gestiune a bazelor de date a făcut posibilă utilizarea pe scară largă în diverse domenii ale activităţii sociale a produselor program, având la bază aceste sisteme.

Primul pas în fundamentarea teoriei bazelor de date îl constituie definirea modelelor de date. Unul din cele mai importante modele de date îl constituie modelul relaţional. În acest model, datele sunt memorate sub formă de relaţii, un element al unei relaţii reprezentând un obiect al lumii reale.

Notăm câteva din avantajele modelului relaţional:

- datele sunt reprezentate simplu şi uşor de regăsit,
- posibilitatea utilizării limbajelor de nivel înalt pentru acces la date,
- realizarea integrităţii şi confidenţialităţii datelor,
- posibilitatea de realizare a unei mulţimi diverse de aplicaţii,
- existenţa unei metodologii de proiectare a bazelor de date.

Scopul principal al lucrarii il constituie familiarizarea cititorului cu elementele de baza ale modelului relational, precum si cu aspectele implementarii acestui model in sisteme de gestiune de baze de date de tip Foxpro. Esenţa modelului relaţional al bazelor de date îl constituie teoria dependenţelor. În fond, dependenţele pot fi considerate ca un limbaj pentru definirea semanticii bazelor de date. Această teorie a permis utilizarea logicii matematice în studiul dependenţelor. De fapt, acest lucru a fost urmărit, în special, în lucrarea de faţă. Restricţiile de tip dependenţe funcţionale şi multivaluate sunt tratate in capitolele XIV si XV. Pentru studiul dependenţelor funcţionale şi multivaluate este suficient calculul propoziţional (cap. XIV, XV).

Subliniem că pentru studiul bazelor de date deductive şi a programării logice se utilizează calculul cu predicate.

Materialul prezentat în această lucrare se bazează pe conţinutul cursurilor predate, mai mulţi ani, studenţilor Facultăţii de Informatică din Iaşi.

Autorul

CAPITOLUL I
ELEMENTE ALE MODELULUI RELAȚIONAL

Considerăm o mulțime nevidă, finită de simboluri, notată cu **U**. Numim elementele lui **U** atribute.

Fie **U** = $\{A_1, A_2, \ldots, A_n\}$. Fiecărui atribut A_i îi vom asocia o mulțime nevidă de valori notată dom(A_i), care va fi numită domeniul valorilor atributului A_i. Această mulțime dom (A_i) este numită şi mulțimea valorilor posibile pentru A_i, $1 \le i \le n$.

Vom numi uplu peste **U** o aplicație, notată φ, φ: $\mathbf{U} \to \bigcup_{1\le i\le n} \mathrm{dom}(A_i)$, astfel încât $\varphi(A_i) \in \mathrm{dom}(A_i)$, $1 \le i \le n$.

Pentru fiecare uplu φ peste **U** putem să-i asociem mulțimea $\{A_1:V_1, A_2 : V_2, \ldots, A_n : V_n\}$, unde $V_i = \varphi(A_i)$, $1 \le i \le n$.

Invers, dacă este dată mulțimea $\{A_1:V_1, A_2 : V_2, \ldots, A_n : V_n\}$, cu $V_i \in \mathrm{dom}(A_i)$, $1 \le i \le n$, atunci putem defini uplul φ peste **U** prin $\varphi(A_i) = V_i$, $1 \le i \le n$. Deci există o corespondență biunivocă între mulțimea de uple peste **U** şi mulțimea mulțimilor de forma considerată. Mai mult, dacă ordonăm mulțimea de atribute ale lui **U** sub forma: **U** = $(A_1, A_2, \ldots, A_n)$, atunci în locul mulțimii $\{A_1:V_1, A_2 : V_2, \ldots, A_n : V_n\}$ se poate considera numai vectorul $(V_1, V_2, \ldots, V_n)$. În sistemele de gestiune de baze de date (FOX, DBASE, etc.) se lucrează cu astfel de ordonări. Vom privi uplele atât ca aplicații, cât şi ca vectori, considerând **U** ordonată.

O relație peste **U**, notată r, este o mulțime de uple peste **U**. În cazul în care mulțimea de uple este vidă, spunem că relația este vidă. În cazul în care mulțimea de uple este finită, spunem că relația este finită.

Rezultă că într-o astfel de relație nu putem avea două uple identice.

Ca operații de actualizare a unei relații putem avea adăugarea de noi uple, ştergerea unor uple, modificarea de uple. În prelucrarea unei relații, mulțimea de uple variază în timp, ceea ce este constant în timp este structura sa, adică numele relației, împreună cu mulțimea de atribute. Să notăm această structură prin R(**U**) sau R[**U**], sau $R(A_1, A_2, \ldots, A_n)$ sau $R[A_1, A_2, \ldots, A_n]$, unde R este un simbol numit numele relației, iar **U** sau $\{A_1, A_2, \ldots, A_n\}$ este mulțimea de atribute corespunzătoare.

R(**U**) poartă denumirea şi de schemă de relație.

În cazul în care r este o relație finită peste **U** şi **U** se ordonează sub forma $(A_1, A_2, \ldots, A_n)$, atunci rezultă o reprezentare a lui r sub forma unui tablou:

	A_1	A_2	…	A_n
r	V_{11}	V_{12}	…	V_{1n}
	……….			
	V_{h1}	V_{h2}	…	V_{hn}

unde $(V_{i1}, V_{i2}, \ldots, V_{in})$ constituie un uplu din r, $1 \le i \le h$. Avem $V_{ij} \in \mathrm{dom}(A_j)$, $1 \le j \le n$, $1 \le i \le h$.

Din acest motiv, în majoritatea sistemelor de gestiune de baze de date, structurile de date ce memorează aceste relații se numesc tabele, iar uplele se numesc linii.

O schemă de baze de date, notată D, este o mulțime finită de scheme de relație: $D = \{R_1[U_1], \ldots, R_p[U_p]\}$, unde $R_j[U_j]$ este o schemă de relație, $1 \le j \le p$.

O bază de date peste schema de baze de date D, este o aplicație ce asociază fiecărei scheme de relație $R_j[U_j]$ o relație r_j peste U_j, $1 \le j \le p$. Dacă D se consideră ordonată sub forma ($R_1[U_1]$, …, $R_p[U_p]$), atunci baza de date este un vector de relații, notat $\bar{r}$ = (r_1, r_2, .., r_p).

Structura unei relații se numește partea sa intensională, iar uplele relației poartă denumirea de partea sa extensională.

Dăm în continuare câteva operații referitoare la relații:

a) Proiecția unei relații.

Fie t un uplu peste R[U] și X o submulțime a lui U. Proiecția lui t relativ la X, notată prin t[X] este restricția lui t (ca uplu) la submulțimea X. Aici t este considerat ca aplicație. Dacă X este mulțimea vidă, atunci t[X] il vom considera uplul vid. Dacă U este ordonată sub forma ($A_1,A_2,\ldots,A_n$), iar în această ordonare $X=(A_{i1},A_{i2},\ldots,A_{ik})$, cu $i_1<i_2<\ldots<i_k$ și $t=(V_1,V_2,\ldots,V_n)$, atunci $t[X]=(V_{i1},V_{i2},\ldots,V_{ik})$. Dacă r este o relație peste U, atunci proiecția lui r relativă la X, notată r[X] va fi definita prin: $r[X]=\{t[X] / t\in r\}$.

b) Reuniunea a două relații.

Fie r_1 și r_2 relații peste R[U]. Atunci $r_1\cup r_2$ notează reuniunea celor două și se definește ca:

$r_1\cup r_2=\{t$ / t uplu, $t\in r_1$ sau $t\in r_2\}$

c) Diferența a două relații.

Fie r_1 și r_2 relații peste R[U]. Diferența lor, notată r_1-r_2 se definește astfel:

$r_1-r_2=\{t$ / t uplu, $t\in r_1$ și $t\notin r_2\}$

d) Unire (join) oarecare.

Fie relațiile r_i definite peste $R_i[U_i]$, i=1,2 $U_1\cap U_2=\emptyset$.

Fie $A_{\alpha 1}, A_{\alpha 2},\ldots, A_{\alpha q}\in U_1$, $B_{\beta 1}, B_{\beta 2},\ldots, B_{\beta q}\in U_2$ și Θ_i un operator de comparare între dom($A_{\alpha i}$) și dom($B_{\beta i}$), $1\le i\le q$. Sintactic Θ_i este unul din operatorii =, <, >, <=, >=, <>. Deci Θ_i este o relație binară pe dom($A_{\alpha i}$)×dom($B_{\beta i}$). Rezultă că cele două domenii conțin valori comparabile prin Θ_i. Considerăm $\Theta=(A_{\alpha 1}\Theta_1 B_{\beta 1})\wedge\ldots\wedge(A_{\alpha q}\Theta_q B_{\beta q})$, unde semnul "$\wedge$" reprezintă conjuncția.

Join-ul oarecare între r_1 și r_2 prin expresia Θ se notează prin $r_1 \overset{*}{\underset{\Theta}{-}} r_2$ și se definește prin:

$$r_1 \overset{*}{\underset{\Theta}{-}} r_2=\{t / t \text{ uplu peste } U_1\cup U_2, t[U_i]\in r_i, i=1,2 \text{ și } t[A_{\alpha j}]\Theta_j t[B_{\beta j}], j=1,q\}$$

Se pot defini expresii de tip Θ mai generale decit cea de sus, folosind parantezele și operatorii de conjuncție și disjuncție ($\wedge$, respectiv $\vee$) astfel:

- o expresie elementară este de forma AΘB, unde A și B sunt atribute, iar Θ este operator de comparare.
- o expresie join se definește astfel:

1. Dacă e este o expresie elementară, atunci e și (e) sunt expresii join.
2. Dacă e_1 și e_2 sunt expresii join, atunci $e_1\wedge e_2$, $e_1\vee e_2$, $(e_1\wedge e_2)$, $(e_1\vee e_2)$ sunt expresii join.
3. Orice expresie join se obține numai prin regulile 1 și 2.

Dacă Θ este o expresie join, atunci se definește faptul că (t_1,t_2) satisface Θ în mod recursiv, $t_1\in r_1$, $t_2\in r_2$.

1. (t_1,t_2) satisface AΘB dacă $t_1[A]\Theta t_2[B]$.
2. (t_1,t_2) satisface $e_1\wedge e_2$ și $(e_1\wedge e_2)$ dacă (t_1,t_2) satisface e_1 și e_2.
3. (t_1,t_2) satisface $e_1\vee e_2$ și $(e_1\vee e_2)$, dacă (t_1,t_2) satisface e_1 sau e_2.

În acest caz joinul oarecare se definește prin:

$r_1 \underset{\Theta}{\overset{*}{—}} r_2=\{(t_1,t_2) / (t_1,t_2)$ uplu peste $U_1 \cup U_2$, $t_i \in r_i$, $i=1,2$ şi (t_1,t_2) satisface $\Theta\}$

e) Produsul cartezian.

Fie r_i relații definite peste $R_i[U_i]$, $i=1,2$ şi $U_1 \cap U_2=\emptyset$.

Produsul cartezian al relațiilor r_1 şi r_2 se noteaza prin $r_1 \times r_2$ şi se defineşte prin: $r_1 \times r_2=\{t$ / t uplu peste $U_1 \cup U_2$, $t[U_i] \in r_i$, $i=1,2\}$

Obs. Produsul cartezian este un join oarecare cu expresia join Θ=TRUE.

f) Unirea naturală (join natural)

Fie r_i relații peste $R_i[U_i]$, $i=1,2$. Se numeşte join natural sau unire a celor două relații, notat r_1*r_2 o relație peste $U_1 \cup U_2$ definită prin:

$r_1*r_2=\{t$ / t uplu peste $U_1 \cup U_2$, $t[U_i] \in r_i$, $i=1,2\}$

Fie R un nume pentru relația peste $U_1 \cup U_2$. Deci relația r_1*r_2 va fi considerată peste schema $R[U_1 \cup U_2]$. Considerăm un exemplu pentru operația de join natural. Fie R_1[ABCD], R_2[CDE] şi relațiile r_1 şi r_2 date astfel:

	A	B	C	D	
r_1	1	0	0	0	t_1
	1	1	0	0	t_2
	0	1	0	1	t_3
	0	0	0	1	t_4
	1	1	1	1	t_5

	C	D	E	
r_2	0	0	0	V_1
	1	1	1	V_2
	1	1	0	V_3
	1	0	0	V_4
	1	0	1	V_5

Atunci joinul natural r_1*r_2 va fi:

	A	B	C	D	E
	1	0	0	0	0
$r_1*r_2=$	1	1	0	0	0
	1	1	1	1	1
	1	1	1	1	0

Se observă că uplele t_3 şi t_4 din r_1 nu contribuie la construcția joinului, deoarece nu există uple $V_j \in r_2$, astfel încît $t_3[CD]=V_j[CD]$ şi nu există uple $V_j \in r_2$, astfel încît

$t_4[CD]=V_j[CD]$. La fel uplele V_4 şi V_5 din r_2 nu contribuie la formarea joinului, deoarece $V_4[CD]=V_5[CD]\notin r_1[CD]$.
Să remarcăm faptul că perechea (t_i,V_j), $t_i\in r_1$, $V_j\in r_2$ contribuie la formarea joinului, dacă $t_i[CD]=V_j[CD]$.
În acest caz, va rezulta un uplu în joinul natural, notat w, definit prin: $w[U_1]=t_i[U_1]$, $w[U_2-U_1]=V_j[U_2-U_1]$.

Se observă uşor următoarele relaţii:

1. $(r_1*r_2)[U_1]\subseteq r_1$, $(r_1*r_2)[U_2]\subseteq r_2$.
2. Dacă $r_1'=\{t_1 \,/\, t_1\in r_1, \exists t_2\in r_2, \text{a.î.}, t_1[U_1\cap U_2]=t_2[U_1\cap U_2]\}$
   şi $r_2'=\{t_2 \,/\, t_2\in r_2, \exists t_1\in r_1, \text{a.î.}, t_1[U_1\cap U_2]=t_2[U_1\cap U_2]\}$
   şi $r_1''=r_1-r_1'$, $r_2''=r_2-r_2'$, atunci
   $r_1*r_2= r_1'*r_2'$, $(r_1*r_2)[U_1]=r_1'$, $(r_1*r_2)[U_2]=r_2'$.
3. Dacă $\overline{r_1}\subseteq r_1$ şi $\overline{r_2}\subseteq r_2$ satisfac relaţia $\overline{r_1} * \overline{r_2} =r_1*r_2$, atunci $r_1'\subseteq\overline{r_1}$ şi $r_1'\subseteq\overline{r_2}$, adică relaţiile r_1', r_2' sunt minimale în r_1 respectiv r_2 cu proprietatea $r_1'* r_2' =r_1* r_2$. În cazul în care $U_1\cap U_2=\emptyset$, atunci este clar că joinul natural este tocmai produsul cartezian $r_1\times r_2$.

Operaţia de join natural se poate extinde la mai multe relaţii.
Fie r_i peste $R_i[U_i]$, $i=1,h$. Joinul natural al acestora se notează prin
$r_1*r_2* \dots *r_h$ sau $*\langle r_i,i=1,h\rangle$, sau $\quad * \langle r_i,i=1,h\rangle$ şi se defineşte prin:
$r_1*r_2* \dots *r_h=\{t \,/\, t \text{ uplu peste } U_1\cup\dots\cup U_h, \text{ a.î. } t[U_i]\in r_i, i=1,h\}$
Este clar că operaţia de join natural nu depinde de ordinea considerării relaţiilor r_i, $i=1,h$.
Dacă notăm cu $*_2$ operaţia de join natural între două relaţii şi cu $*_h$ operaţia de join natural între h relaţii, apare următoarea problemă:
<u>Problema 1</u>. Operaţia $*_h$ se poate exprima cu ajutorul operaţiei $*_2$?
Răspunsul la problema 1 este NU. Demonstraţi.

g) Selecţia.

Fie r o relaţie peste R[U].
Definim întâi expresia elementară de selecţie prin:

> $A\Theta B$ sau $A\Theta c$ sau $c\Theta B$, unde $A,B\in U$, iar c este o constantă, comparabilă cu elementele din dom(A) şi dom(B).

Dacă e_1 şi e_2 sunt expresii elementare de selecţie, atunci (e_1), $e_1\wedge e_2$, $e_1\vee e_2$, $(e_1\wedge e_2)$, $(e_1\vee e_2)$ este o expresie de selecţie. Orice expresie de selecţie se obţine numai prin regulile de mai sus. Fie E o expresie de selecţie.
Precizăm cazul în care uplul t satisface E.
Dacă $E\equiv A\Theta B$, atunci t satisface E, dacă $t[A]\Theta t[B]$.
Dacă $E\equiv A\Theta c$, atunci t satisface E, dacă $t[A]\Theta c$.
Dacă $E\equiv c\Theta B$, atunci t satisface E, dacă t satisface $c\Theta t[B]$.
Dacă $E\equiv e_1\wedge e_2$, atunci t satisface E, dacă t satisface e_1 şi e_2.
Dacă $E\equiv e_1\vee e_2$, atunci t satisface E, dacă t satisface e_1 sau t satisface e_2.
Fie F o expresie de selecţie.
Selecţia se notează prin $\sigma_F(\underline{r})$ şi se defineşte prin:
$\sigma_F(\underline{r})=\{t \,/\, t \text{ uplu peste } R[U], t \text{ satisface } F\}$.

h) Intersecţia relaţiilor.

Fie r_i relaţii peste $R_i[U_i]$, $i=1,2$ şi $U_1=U_2$.
Atunci intersecţia se defineşte prin:

$r_1 \cap r_2 = \{t \ / \ t$ uplu peste U_1, a.î. $t \in r_1$ şi $t \in r_2\}$

Se arată că o parte din operațiile cu relații se exprimă cu ajutorul altor operații. Mulțimea de operații: reuniune, diferență, produs cartezian, proiecție şi selecție se arată a fi minimală cu proprietatea că celelalte operații se exprimă cu ajutorul acestora.

CAPITOLUL II

Implementarea modelului relațional în SGBD-ul FOX

Fie dată schema R[U]. În sistemul Fox această schemă se reprezintă printr-o structură, notată S. Structura S conține pentru fiecare atribut din U următoarele elemente: nume, tip, lungime, număr de zecimale. Nume este numele atributului sau cîmpului şi este un identificator de cel mult 8 caractere, tipul specifică tipul cîmpului, lungimea este numărul maxim de caractere al valorii cîmpului, iar numărul de zecimale specifică numărul de zecimale pentru cîmpurile numerice.

Deci, structura S este formată dintr-o mulțime de cvadruple de forma ($nume_i$, tip_i, $lungime_i$, dec_i), $i=\overline{1,h}$ unde h este numărul de atribute din U.

Cîmpul tip_i are una din formele: C, N, F, D, L, M. Tipul C semnifică un cîmp de tip caracter, tipul N un cîmp de tip numeric, tipul F un cîmp de tip numeric, dar reprezentarea valorilor sale se face în virgulă mobilă (utilizînd mantisă şi exponent), tipul D un cîmp de tip dată calendaristică (formatul implicit al acesteia este ll/zz/aa, unde ll - este luna, zz - ziua din cadrul lunii, iar aa - ultimele două cifre ale anului), tipul L un cîmp de tip logic, tipul M un cîmp de tip MEMO.

Lungimea maximă a valorilor unui cîmp depinde de tip. Pentru cele de tip C lungimea maximă este de 254. Pentru cele de tip N sau F, lungimea maximă este de 20, în care se consideră şi semnul şi punctul zecimal. Pentru cele de tip D lungimea este 8, pentru cele de tip logic lungimea este 1, pentru cele de tip MEMO lungimea este 10. Precizăm ca valoarea unui cîmp MEMO este un pointer la un fişier ce conține un şir de caractere de lungime maximă 65535 (2^{16}-1). Deci vom putea lucra cu şiruri de caractere de lungime mai mare ca 254 şi mai mică decît 65535 utilizînd cîmpuri MEMO.

În FOX structura S se reprezintă ca un tabel de forma:

$cîmp_1$, tip_1, l_1, d_1

...

$cîmp_h$, tip_h, l_h, d_h

unde $cîmp_1$,…, $cîmp_h$ reprezintă atributele, tip_i – tipul, l_i – lungimile, iar d_i – numărul de zecimale pentru cazul cîmpurilor numerice.

Simbolului R îi corespunde un nume de fişier în sistemul de operare. Acesta va avea extensia DBF. Să notăm numele fişierului cu F_R. Fie r o relație peste R[U]. Atunci r se reprezintă în FOX sub forma unei mulțimi de înregistrări, cîte o înregistrare pentru fiecare linie a lui r. Dacă r are m linii, vom nota înregistrările sub forma $F=(I_1, I_2,…,I_m)$, unde I_j este înregistrarea corespunzătoare liniei j. În FOX înregistrările sunt ordonate şi pentru fiecare înregistrare la creare sau adăugare se atribuie un număr natural, numit numărul înregistrării respective. Dacă înregistrarea I este prima ce se crează, atunci ea primeşte numărul 1, deci se va nota I_1. Dacă I_k este înregistrarea de numărul cel mai mare, care este creată şi urmează o creare de o nouă înregistrare, atunci această nouă înregistrare va primi numărul k+1.

Dacă la un moment dat fişierul F are p înregistrări, atunci înregistrările lui F au numerele 1, 2, …,p.

Orice operație de actualizare a lui F se supune acestui principiu, adică drept rezultat al actualizării se obține o succesiune de înregistrări, numerotate de la 1 la p, unde p este numărul de înregistrări în acel moment. Desigur, numerele de înregistrare se modifică în urma operațiilor de actualizare, pentru a se păstra acest principiu.

Deci F_R conține două părți S - structura şi F - înregistrările, de aceea vom nota $F_R=(S, F)$.

Întotdeauna S≠∅. Putem avea însă F=∅, cînd relația este vidă la un moment dat.

Din punct de vedere fizic, înregistrările din F sînt plasate la adrese consecutive, adică dacă I_1 se plasează pe suport la adresa Q_F şi notăm cu Q_{Ij} adresa de memorare a înregistrării I_j (de număr j), cu L – lungimea unei înregistrări (care este constantă, nu depinde de înregistrare), atunci $Q_{Ij}=Q_F+L*(j-1)$.
Deci dispunerea înregistrărilor lui F este astfel:
Q_F

I_1
I_2
. . .
I_m
EOF

După ultima înregistrare se adaugă o marcă de sfîrşit de fişier, pe care o vom nota `EOF`. Aceasta va servi pentru citirea secvențială a înregistrărilor lui F. Formulele de mai sus permit accesul direct la o înregistrare, cunoscînd numărul ei. Dacă se dă numărul ei, de ex. j, se calculează Q_{Ij} după formula de sus şi se obține adresa acestei înregistrări.

Crearea în FOX a lui F_R

Există mai multe moduri de a crea un fişier DBF în FOX.
Vom da cîteva posibilități. Cea mai des utilizată este utilizarea comenzii `CREATE <numefişier>`. Această comandă poate să apară fie în fereastra de comandă, fie în cadrul unui program.
Execuția comenzii `CREATE <numefişier>` se realizează în două etape:
- În etapa întîia se defineşte structura, notată cu S.

Pentru aceasta SGBD-ul FOX afişează un antet de tablou de forma:

Name Type Width Dec
şi utilizatorul va specifica pentru fiecare cîmp cele patru caracteristici (nume, tip, lungime şi număr de zecimale pentru cîmpurile numerice). Terminarea specificării structurii se va face prin tasarea CTRL+W sau a butonului <OK>. Urmează afişarea mesajului INPUT DATA RECORDS NOW ? cu butoanele <YES> şi <NO>.
Dacă selectăm <NO>, atunci în acest moment mulțimea de înregistrări este vidă (F=∅), spunem că fişierul este vid, desigur vom avea memorată structura sa. Prin comenzi ulterioare ca APPEND, EDIT, CHANGE, etc. se vor introduce înregistrări în F. Dacă selectăm <YES>, atunci operația de creare continuă cu etapa a II-a şi anume definirea înregistrărilor.
Apare pe ecran macheta înregistrării, care are forma:

cîmp$_1$

l_1

cîmp$_2$

l_2

cîmp$_h$

l_h

unde cîmp$_i$ sînt cîmpurile fişierului, iar în dreapta fiecărui cîmp$_i$ apare o zonă pe ecran în altă culoare decît fondul ecranului, pe lungime l_i, unde se va afişa valoarea tastată de utilizator pentru acel cîmp.

Cursorul se deplasează cu fiecare caracter tastat şi afişat.

Dacă pentru cîmp$_i$ valoarea tastată este mai mică decît l_i, atunci utilizatorul trebuie să dea ENTER pentru ca acel cursor să treacă la cîmpul următor. În cazul în care pentru cîmp$_i$ valoarea tastată este de lungime egală cu l_i, atunci utilizatorul nu trebuie să dea ENTER, deoarece cursorul trece la cîmpul următor. Dacă există spaţiu pe ecran, pot fi afişate mai multe machete ale înregistrării pe un singur ecran. Terminarea etapei de definire a înregistrărilor lui F se realizează prin tastarea lui CTRL+W, ceeace înseamnă terminarea comenzii CREATE.

O altă modalitate de creare a unui fişier DBF este următoarea:

a) Se copie structura unui fişier existent folosind comanda COPY STRUCTURE EXTENDED într-un alt fişier F '.
b) Folosim CREATE FROM.

Comanda de la a) are forma:

```
COPY TO <fişier1> STRUCTURE EXTENDED
```

Fişierul bază de date activ este notat prin F_R=(S,F). Comanda copie structura S în <fişier$_1$>, astfel încît o înregistrare din <fişier$_1$> corespunde unui cîmp din S. Deci, <fişier$_1$> are 4 cîmpuri cu numele: FIELD_NAME, FIELD_TYPE, FIELD_LEN, FIELD_DEC. Numărul de înregistrări din <fişier$_1$> este egal cu numărul de cîmpuri din S.

Exemplu: Considerăm fişierul PERSONAL cu structura:

MARCA	N	5
NUME	C	10
PRENUME	C	12
SALARIU	N	7
NRCOPII	N	1
FUNCTIE	C	9
DATA_ANG	D	8

Prin secvenţa:

```
USE PERSONAL
COPY TO F1 STRUCTURE EXTENDED
USE F1
LIST
```

Se vor afişa înregistrările lui F1 astfel:

	FIELD_NAME	FIELD_TYPE	FIELD_LEN	FIELD_DEC
1	MARCA	N	5	0
2	NUME	C	10	
3	PRENUME	C	12	
4	SALARIU	N	7	0
5	NRCOPII	N	1	0
6	FUNCTIE	C	9	
7	DATA_ANG	D	8	

Deci F_1 conține structura fişierului PERSONAL, dar sub formă de înregistrări. Comanda `USE PERSONAL` realizează activarea fişierului PERSONAL în zona curentă care este zona de număr 1. `USE F1` activează fişierul F_1 în zona 1.
Comanda de la b) are forma:
`CREATE <fişier nou> FROM <fişier structură>`
unde `<fişier structură>` este cel prezent în comanda de la a) iar `<fişier nou>` va fi noul fişier creat care va avea structura existentă în `<fişier structură>`.

În exemplul anterior dacă vom da `CREATE PERSON1 FROM F1` atunci se crează fişierul PERSON1 cu structura care apare în F_1.

Această metodă este interesantă prin faptul că putem prin program să modificăm structura vechiului fişier, modificînd înregistrările din F_1. De exmplu, dorim ca lumgimea cîmpului SALARIU să fie de 8, lungimea lui NRCOPII să fie 2, iar numărul de zecimale pentru cîmpul MARCA să fie 1, atunci vom da:

```
USE F1          *activarea fişierului F1
GO 1            *poziţionarea pe înregistrarea 1
REPLACE FIELD_DEC WITH 1      *modificarea cîmpului
GO 4            *poziţionarea pe înregistrarea 4
REPLACE FIELD_LEN WITH 8      *modificarea valorii
GO 5            *poziţionarea pe înregistrarea 5
REPLACE FIELD_LEN WITH 2 *    modificarea valorii
CREATE PERSON2 FROM F1        *se   crează   PERSON2   *cu
structura modificată.
```

A treia posibilitate de creare a unui fişier F_R este dată de utilizarea comenzii `CREATE TABLE`, pe care o vom discuta mai tîrziu deoarece este legată de comenzi de tip SQL.

Actualizarea bazelor de date.

Pentru a lucra cu o bază de date (fişier tip DBF) ea trebuie în prealabil să fie deschisă sau activată. Acest lucru se realizează prin comanda `USE <nume fişier>` care presupune deschiderea bazei `<nume fişier>` în zona de lucru curentă. Un fişier tip DBF este numit şi tabelă.

Sistemul FOX pentru DOS permite utilizarea a 25 zone de lucru. Fără utilizarea comenzii `SELECT <număr>`, unde `<număr>` are o valoare între 1 şi 25, zona de lucru curentă este de număr 1. Folosind `SELECT n`, unde $n \in \{1, 2, .., 25\}$, noua zonă de lucru va deveni cea de număr n.

Vom vedea că comanda `USE <nume fişier> IN n` permite activarea bazei `<nume fişier>` în zona de lucru de număr n, fără schimbarea zonei de lucru curente. În fiecare zonă de lucru se poate activa un fişier. Mai mult acest fişier se poate activa în mai multe zone utilizînd clauza `AGAIN` în comanda `USE`. De exemplu, fişierul PERSONAL vrem să-l activăm în zonele 1 şi 2:

```
SELECT 1             *zona 1 este curentă
USE PERSONAL         *activarea lui PERSONAL în zona 1
SELECT 2             *zona 2 este curentă
USE PERSONAL AGAIN   *activarea lui PERSONAL şi în zona 2
```

Zona de lucru curentă are următoarea semnificaţie: orice comandă care nu conţine referiri la o anumită zonă, se va considera că lucrează asupra zonei curente, adică asupra fişierului activ în zona curentă.

Exemplu

```
SELECT 1
USE PERSONAL         *activarea lui PERSONAL în zona 1
SELECT 2
USE PRODUSE          *activarea lui PRODUSE în zona 2
LIST                 *se listează PRODUSE
SELECT 1
LIST                 *se listează PERSONAL
```

Dacă F_R =(S, F) este activ într-o anumită zonă, atunci putem avea situaţiile:

1) F este poziţionat pe o anumită înregistrare a sa, numită înregistrare curentă. (Există mai multe comenzi de poziţionare `GO`, `LOCATE`, `SKIP` etc.)
2) F este poziţionat pe `EOF`; spunem că în acest caz nu există înregistrare curentă a lui F.

Semnificaţia înregistrării curente este: în cazul în care o comandă nu conţine elemente ce definesc o altă înregistrare, atunci ea se referă la înregistrarea curentă. Exemplu:

```
USE PERSONAL
GO 2            *poziţionarea pe înregistrarea 2
DISPLAY         *afişează înregistrarea 2 (curentă)
```

Drept operaţii de actualizare avem:

1) Adăugări
2) Modificări
3) Ştergeri

Adăugările se pot realiza:

1.1. La sfîrşit;
1.2. În interior după o anumită înregistrare;
1.3. În interior înainte de o anumită înregistrare.

Adăugările la sfîrşit se pot realiza cu comenzile:

1.1.1. `APPEND`;
1.1.2. `APPEND BLANK`.

Comanda `APPEND` se execută astfel: se afişează macheta înregistrării, utilizatorul tastează un număr dorit de înregistrări pentru a fi adăugate în fişier. Terminarea comenzii se realizează cu `CTRL+W`.

Comanda `APPEND BLANK` adaugă la sfîrşitul fişierului o înregistrare, fără nici un cîmp completat (numită înregistrare vidă). Completarea cîmpurilor acesteia se poate realiza cu diverse comenzi: `REPLACE`, `BROWSE`, `EDIT`.

Exemplu. Dacă dorim să adăugăm în PERSONAL înregistrarea IONESCU ALA 1700000 2 MUNCITOR 05/12/70 atunci vom da secvenţa:

```
USE PERSONAL
APPEND BLANK
REPLACE MARCA WITH 125
REPLACE NUME WITH 'IONESCU'
REPLACE PRENUME WITH 'ALA'
REPLACE SALARIU WITH 1700000
REPLACE NRCOPII WITH 2
REPLACE FUNCŢIE WITH 'MUNCITOR'
REPLACE DATA_ANG WITH {05/12/70}
```

Pentru a adăuga în interior după o anumită înregistrare vom folosi:

1.2.1. `INSERT`

1.2.2. `INSERT BLANK`

În prealabil trebuie definită înregistrarea respectivă folosind comenzi de poziţionare (`GO`, `SKIP`, `LOCATE` etc.)

Fie I_j înregistrarea pe care se poziţionează fişierul (numită înregistrarea curentă). Atunci `INSERT` va afişa pe ecran macheta înregistrării şi utilizatorul poate specifica oricâte înregistrări, care vor fi incluse după înregistrarea I_j. Vom vedea că înregistrările se renumerotează în timpul acestei operaţii de adăugare. Terminarea comenzii se realizează cu CTRL+W. Comanda `INSERT BLANK` realizează adăugarea după I_j a unei înregistrări vide, care se va completa cu `REPLACE`, `BROWSE`, `EDIT`.

Adăugarea în interior înainte de o anumită înregistrare se realizează cu comenzile:

1.3.1. `INSERT BEFORE`

1.3.2. `INSERT BEFORE BLANK`

Ca şi în cazul lui 1.2., în prealabil trebuie definită înregistrarea respectivă prin comenzi de poziţionare. Fie I_j această înregistrare. Comanda `INSERT BEFORE` afişează macheta înregistrării, utilizatorul tastează cîte înregistrări doreşte să adauge, acestea vor fi adăugate înainte de I_j. Terminarea comenzii se realizează cu CTRL+W. Comanda `INSERT BEFORE BLANK` produce adăugarea unei înregistrări vide înainte de înregistrarea I_j. Completarea cîmpurilor acestei înregistrări se realizează ulterior acestei comenzi, utilizând comenzi de tipul `REPLACE`, `BROWSE`, `EDIT,etc`.

Să precizăm acum modul de renumerotare a înregistrărilor, precum şi înregistrarea curentă în cazul operaţiilor de adăugare. În primul rînd, în momentul în care se adaugă o înregistrare I', ea devine curentă.

Fie $F = (I_1, I_2, ..., I_m)$ la momentul t şi presupunem că adăugăm înregistrarea I' după I_j, atunci F devine F' (la momentul t + 1):

$F' = (I_1, I_2, ..., I_j, I', I_{j+1}, ..., I_m)$

Dacă notăm cu ORD_t funcţia de numerotare la momentul t şi cu ORD_{t+1} funcţia de numerotare la momentul t+1, atunci $ORD_t(I_l) = l,\ l = \overline{1,m}$

$ORD_{t+1}(I_l) = l,\ l = \overline{1,j}$, $ORD_{t+1}(I') = j + 1$

$ORD_{t+1}(I_l) = l + 1,\ l = \overline{j+1,m}$

Practic fişierul F' va conține noua înregistrare I' între I_j şi I_{j+1}. Fizic, înregistrările I_{j+1}, ..., I_m vor fi deplasate pentru a face loc înregistrării I' adăugate. Înregistrarea I' adăugată devine curentă, ea va avea numărul j+1. În cazul adăugării mai multor înregistrări procedeul este similar, înregistrările fiind adăugate una cîte una.

Modificarea înregistrărilor se realizează utilizînd mai multe comenzi, din care enumerăm REPLACE, BROWSE, EDIT, CHANGE. Dacă se utilizează REPLACE, atunci fişierul trebuie poziționat pe înregistrarea care se doreşte a fi modificată. Dacă vom dori să modificăm valoarea lui <cîmp$_i$> cu valoarea unei expresii <exp>, atunci vom da:

REPLACE <cîmp$_i$> WITH <exp>

Dacă utilizăm comanda BROWSE pentru modificare, atunci înregistrările sînt afişate pe ecran sub forma unui tabel, în care liniile sînt înregistrările, iar coloanele sînt cîmpurile. Deplasarea pe linii se realizează cu săgețile verticale, iar deplasarea în cadrul unei înregistrări cu săgețile orizontale. Cîmpul pe care se poziționează bara luminoasă poate fi modificat cu o nouă valoare dorită. Comenzile EDIT şi CHANGE afişează machetele înregistrărilor cu înregistrările respective; folosind săgețile verticale ne deplasăm înainte sau înapoi pe cîmpurile machetei.

Nu vom da aici sintaxa acestor comenzi şi posibilitățile de lucru ale acestora.

Ştergerea înregistrărilor se realizează în două etape:

3.1. Ştergerea logică

3.2. Ştergerea fizică.

Ştergerea logică se realizează prin DELETE, iar ştergerea fizică prin comanda PACK. Ştergerea logică înseamnă marcarea înregistrărilor pe care dorim să le eliminăm din fişier. O înregistrare marcată este vizualizată (cu comezile LIST, DISPLAY ALL) prin afişarea caracterului '*' ce precede înregistrarea respectivă afişată. După marcarea unor înregistrări în vederea eliminării lor din fişier, se poate renunța la marcarea unora (deci la eliminarea ulterioară a lor) folosind comanda RECALL care realizează aşa–numita operație de demarcare a înregistrărilor.

Sintaxa comenzii DELETE este:

DELETE <domeniu> FOR <condiție$_1$> WHILE <condiție$_2$>

în care una sau mai multe componente ale comenzii pot lipsi. Vom specifica posibilitatea absenței unei componente prin plasarea ei între paranteze pătrate. Deci vom scrie:

DELETE [<domeniu>] [FOR <condiție$_1$>] [WHILE <condiție$_2$>]

unde <condiție$_1$> şi <condiție$_2$> sînt expresii logice.

Componenta <domeniu> poate avea următoarele forme:

1. ALL
2. RECORD h
3. NEXT p
4. REST

Fie F = (I_1, I_2, ..., I_m) înregistrările unui fişier. Componentele comenzii DELETE sînt considerate în ordinea <domeniu>, WHILE, FOR. Să notăm prin F_1 rezultatul considerării componentei <domeniu>, apoi componenta WHILE are la intrare F_1 şi să notăm cu F_2 ieşirea. Componenta FOR va avea la intrare F_2 şi să notăm cu F_3 ieşirea acesteia. În cazul cînd <domeniu> are forma RECORD h, atunci h este o constantă sau o variabilă cu valori în {1, 2, .., m} şi reprezintă un număr de înregistrare. În cazul cînd <domeniu> are forma NEXT p sau REST, atunci F trebuie să fie poziționat (prin comenzi de poziționare) pe o anumită înregistrare, numită înregistrare curentă, pe care o vom nota I_c, unde c este numărul înregistrării curente.

Dacă <domeniu> este ALL atunci $F_1 = F$.

Dacă <domeniu> este RECORD h, atunci $F_1 = (I_h)$, deci înregistrarea de număr h, desigur cu condiția $h \in \{1, 2, .., m\}$. Dacă $h \notin \{1,2,.., m\}$, atunci $F_1 = \emptyset$.

În cazul cînd <domeniu> este NEXT p şi I_c este înregistrarea curentă, atunci $F_1 = (I_c, I_{c+1}, ..., I_{c+p-1})$, dacă $c+p-1 \le m$ şi $F_1 =(I_c, I_{c+1},..., I_m)$ în czul $c+p-1>m$.

În situația cînd <domeniu> este REST, atunci $F_1 =(I_c, I_{c+1},..., I_m)$.

Dacă, componenta <domeniu> lipseşte, atunci $F_1=(I_c)$ unde I_c este înregistrarea curentă. Dacă nu există înregistrare curentă $F_1 = \emptyset$.

Se observă că F_1 este formată din înregistrări consecutive din F (atunci cînd $F_1 \neq \emptyset$). Fie $F_1 =(I_q, I_{q+1},..., I_{q+s})$. Precizăm acum acțiunea componentei WHILE.

Dacă WHILE lipseşte, atunci $F_2 = F_1$.

Cînd componenta WHILE este prezentă, atunci avem următoarele situații:

a) condiție$_2$ (I_q) = .F.
b) condiție$_2$ (I_l) = .T. pentru $l = q, q+1,\ldots, q+s$
c) există j, $q \le j<q + s$, astfel încît:
   condiție$_2$ (I_l) = .T. pentru $l = q, q+1, ..., j$ şi
   condiție$_2$$(I_{j+1})$ = .F.

Am notat prin condiție$_2$$(I_l)$ valoarea de adevăr a expresiei logice <condiție$_2$> pentru înregistrarea I_l, ce se obține prin înlocuirea cîmpurilor ce apar în <condiție$_2$> cu valorile acestor cîmpuri pentru înregistrarea I_l.

În cazul a) $F_2=\emptyset$, în cazul b) $F_2 =(I_q, I_{q+1},..., I_{q+s})$ iar în cazul c) $F_2=(I_q, I_{q+1},..., I_j)$.

Se observă că dacă $F_2\neq\emptyset$, atunci F_2 este format din înregistrări consecutive (deci avînd numere consecutive) din F. Fie în acest caz $F_2 =(I_q, I_{q+1},..., I_j)$, $q \le j$.

Se ia în considerare, în continuare, componenta FOR. Dacă aceasta lipseşte, atunci $F_3=F_2$. Dacă este prezentă FOR <condiție$_1$> atunci se evaluează expresia logică <condiție$_1$> pentru toate înregistrările din F_2 . Se trec în F_3 numai acelea din F_2 pentru care <condiție$_1$> este adevărată. Deci $F_3 =\left(I_{\alpha_1}, I_{\alpha_2}, \ldots, I_{\alpha_k}\right)$ unde $q \le \alpha_1 < \alpha_2 < ...< \alpha_k \le j$ şi condiție$_1$$(I_{\alpha_1})$= .T., iar <condiție$_1$> pentru înregistrările din F_2 - F_3 este falsă.

Prin urmare rezultatul considerării celor trei componente este F_3. Comanda DELETE va marca logic toate înregistrările din F_3. Evident că vom putea da mai multe comenzi DELETE referitoare la un acelaşi fişier F.

Să dăm un exemplu de utilizare a comenzii DELETE cu diverse situații ale celor trei componente.

Fie fişierul PERSONAL cu structura:

MARCA, NUME, PRENUME, SALARIU, SECȚIE, NRCOPII, DATA_ANG şi avînd următoarele înregistrări:

	MARCA	NUME	PRENUME	SALARIU	SECȚIE	**NRCOPII**	DATA_ANG
1	100	A	A1	1000	1	0	05/20/70
2	200	A	A2	1100	1	1	07/15/72
3	300	B	B1	1150	1	2	08/20/71
4	400	B	B2	950	1	1	10/15/72
5	500	C	C1	1050	2	0	11/20/75

6	600	D	D1	850	2	1	03/18/71
7	700	E	E1	890	2	2	06/22/72
8	800	F	F1	920	2	1	09/25/72
9	900	G	G1	970	2	1	08/28/71
10	950	H	H1	1020	2	0	07/24/70

Să notăm cu F cele 10 înregistrări, $F = (I_1,...,I_{10})$.
Fie comanda `DELETE ALL WHILE SECTIE=1 FOR SALARIU>=1000`. (Componentele `WHILE` şi `FOR` pot fi specificate în orice ordine). Atunci
$F_1=F$, $F_2=(I_1, I_2, I_3, I_4)$, $F_3 =(I_1, I_2)$. Deci comanda va marca pentru ştergere înregistrările I_1 şi I_3. Demarcarea tuturor înregistrărilor se va realiza prin `RECALL ALL`.

Fie comanda `DELETE ALL WHILE SECTIE=2`.
Avem $F_1= F$, $F_2 =\emptyset$, $F_3=\emptyset$. Deci această comandă nu va marca nici o înregistrare.

Fie comanda `DELETE ALL FOR SALARIU<1000`. Ea va marca toate înregistrările cu SALARIU mai mic decît 1000. Deci $F_1=F$, $F_2 =F_1$, $F_3 =(I_4, I_6, I_7, I_8, I_9)$.

Dacă vom da:

```
GO 5
DELETE REST WHILE SECTIE=2 FOR MARCA<850
```

Atunci $I_c = I_5$, $F_1 =(I_5, \ldots, I_{10})$, $F_2=F_1$, $F_3=(I_5, I_6, I_7, I_8)$.
Fie acum următoarea secvență:

```
GO 6
DELETE NEXT 4 WHILE NRCOPII<>0;
     FOR YEAR(DATA_ANG) = 71
```

$I_c=I_6$, $F_1 =(I_6, \ldots, I_9)$, $F_2=F_1$, $F_3=(I_6, I_9)$.
Funcția `YEAR()` returnează anul din data calendaristică respectivă.
Ştergerea fizică se realizează cu comanda `PACK`, care va elimina înregistrările marcate din fişier şi evident va renumerota înregistrările. Comanda `RECALL` care demarchează înregistrările are aceeasi sintaxă ca şi comanda `DELETE`:

```
RECALL [<domeniu>] [FOR <condiție1>] [WHILE <condiție2>]
```

Dacă lipseşte componenta `<domeniu>`, atunci se consideră înregistrarea curentă (dacă există).

Fie F' succesiunea înregistrărilor marcate pentru ştergere şi F_3 înregistrările calculate de comanda `RECALL`, exact ca în cazul comenzii `DELETE`. Atunci, vor fi demarcate înregistrările din $F' \cap F_3$. Se pot specifica mai multe comenzi `RECALL` înainte de comanda `PACK`.
Tripleta `<domeniu>`, `FOR <condiție1>`, `WHILE <condiție2>` o vom întîlni şi la alte comenzi, de exemplu: `LOCATE`, `AVERAGE`, `CALCULATE`, `COUNT`, `DISPLAY`, `LIST`, `EDIT`, `CHANGE`, `EXPORT` etc.

CAPITOLUL III

VARIABILE, TABLOURI, TRANSFERUL DATELOR DIN TABLOURI ÎN FIŞIERE ŞI INVERS

O variabilă simplă este un nume ales de utilizator pentru o anumită mărime de calculat. Tipul variabilei va fi determinat cu ocazia primei asignări a acesteia sub forma: `variabilă = expresie`

Variabilele cu un indice, numite tablouri unidimensionale se declară în program prin `DECLARE nume(expresie),` unde `nume` va fi numele tabloului, iar `expresie` trebuie să aibă valoarea întreagă şi va reprezenta numărul de elemente ale vectorului nume. Elementele acestui vector vor fi nume(1), ..., nume(h), unde h este valoarea expresiei.

Tablourile bidimensionale se vor declara prin `DECLARE nume(expresie`$_1$`, expresie`$_2$`).` Valorile expresiilor `expresie`$_1$, `expresie`$_2$ trebuie să fie întregi, fie acestea h şi k. Atunci comanda scrisă defineşte un tablou cu 2 dimensiuni, primul indice ia valori de la 1 la h, al doilea indice ia valori de la 1 la k.

Elementele tabloului se vor referi în program prin nume(i, j), unde $1 \le i \le h$, $1 \le j \le k$.

În locul cuvîntului rezervat `DECLARE` se poate utiliza cuvîntul `DIMENSION` cu acelaşi rezultat.

Într-o comandă `DECLARE` sau `DIMENSION` pot fi definite mai multe tablouri uni sau bidimensionale, separate prin virgulă. Înainte de iniţializarea elementului unui tablou, toate sînt considerate de tip logic cu valoarea .F. Este de reţinut că elementele unui tablou definit în program pot fi de tipuri diferite, spre deosebire de alte limbaje (TURBO PASCAL, C, FORTRAN etc.) unde toate elementele unui tablou trebuie să fie de acelaşi tip.

În cazul in care programul are proceduri sau funcţii, variabilele vor juca un rol important în transmiterea valorilor. Adăugarea de noi înregistrări la o bază cu valori preluate dintr-un tablou, se realizează cu comanda `APPEND FROM ARRAY` care are sintaxa generală:

```
APPEND FROM ARRAY <numet> [FOR condiţie]
[FIELDS c1, c2,..., ck]
```

unde `<numet>` este numele tabloului din care se preiau valorile. Fie cazul în care `<numet>` este un tablou unidimensional cu h componente. Fie n numărul de coloane ale fişierului activ în zona curentă. Se adaugă o singură înregistrare. Fie $cimp_i$, $1 \le i \le h$ cîmpurile fişierului bază de date activ.

În situaţia n=h, atunci pentru înregistrarea adăugată se vor asigna $cimp_i$=`<numet>`(i), $i = \overline{1,n}$.

În situaţia n<h, vom avea $cimp_i$=`<numet>`(i), $i = \overline{1,n}$, iar celelalte elemente ale tabloului `<numet>` nu vor fi utilizate.

În situaţia n > h, pentru înregistrarea adăugată se va atribui $cimp_i$=`<numet>`(i), $i = \overline{1,h}$, iar celelalte cîmpuri ale înregistrării adăugate rămîn necompletate.

În operaţia de asignare $cîmp_i$=`<numet>`(i), dacă lungimea maximă declarată pentru $cîmp_i$ este mai mică decît valoarea elementului `<numet>`(i), apare eroare de tip depăşire. Evident că trebuie ca tipul pentru $cîmp_i$ să fie acelaşi ca tipul pentru `<numet>`(i).

În cazul în care <numet> este un tablou bidimensional, să zicem cu p linii şi h coloane, atunci se adaugă la sfîrşit în fişier p înregistrări, cîte o înregistrare pentru fiecare linie din tablou. O linie a tabloului va produce valori pentru cîmpurile unei înregistrări ca şi în cazul unui tablou unidimensional (care este considerat ca o singură linie).

Dacă este prezentă componenta FOR <condiție>, atunci această <condiție> este construită cu cîmpuri ale fişierului activ, constante, variabile, funcții. Se adaugă numai înregistrările ce satisfac condiția din FOR. În cazul în care se utilizează componenta FIELDS c_1, c_2, ..., c_k, atunci c_i sînt cîmpuri ale fişierului activ şi pentru înregistrările adăugate se vor inițializa numai cîmpurile c_1, c_2,..., c_k, deci celelalte cîmpuri vor rămîne necompletate. Astfel, în acest caz asignările vor fi c_i=<numet>(i), $i=\overline{1,\ \min(t,\ h)}$ pentru tabloul <numet> unidimensional. Similar pentru tablourile bidimensionale.

Copierea înregistrărilor unei baze de date într-un tablou se realizează cu comanda COPY TO ARRAY ce are forma:

COPY TO ARRAY <numet> [FIELDS c_1, c_2, ..., c_k]
[<domeniu>][FOR <condiție$_1$>] [WHILE <condiție$_2$>]

Tripleta <domeniu>, FOR <condiție$_1$>, WHILE <condiție$_2$> defineşte acel F_3 care reprezintă mulțimea înregistrărilor ce va fi copiată în tabloul <numet>.

Fie dat F fişierul activ cu n cîmpuri: cîmp_1, ..., cîmp_n şi avînd m înregistrări, F = (I_1, I_2, ..., I_m). Dacă lipsesc componenele <domeniu>, FOR şi WHILE, atunci sînt considerate pentru transfer toate înregistrările din F.

Fie dat <numet> un tablou unidimensional cu h componente. Atunci se copie o singură înregistrare I_1 din F în <numet>.

Dacă n=h, atunci <numet>(i)=$\text{cimp}_i(I_1)$, $i = \overline{1,n}$.

Dacă n>h, atunci <numet>(i) = $\text{cimp}_i(I_1)$, $i = \overline{1,h}$, restul de n – h valori ale înregistrării I_1 nu sînt transferate.

În cazul cînd $n \le h$, atunci <nume>(i)=cimp_i (I_1), $i = \overline{1,n}$ celelalte h–n elemente ale tabloului rămîn cu vechile valori.

Dacă <numet> este un tablou bidimensional cu p linii şi h coloane, atunci se transferă datele sub forma: o înregistrare din F într-o linie din <numet>.

Dacă m=p , atunci se transferă cele m înregistrări ale lui F în cele m linii ale tabloului <numet>. Pentru o înregistrare transferată apar aceleaşi cazuri, ca pentru tabelul <numet> unidimensional (un tablou unidimensional este considerat format dintr-o singură linie şi numărul de coloane este egal cu numărul elementelor sale).

Dacă m<p, atunci rămîn p–m linii ale tabelului care nu sînt afectate de transformare (ultimele).

Dacă m>p, atunci rămîn ultimele m–p înregistrări din F, care nu se transferă în tablou.

Dacă se foloseşte componenta FIELDS c_1, c_2, ..., c_k, unde c_j sînt cîmpuri ale lui F, adică $c_j = \text{cîmp}_{\alpha_j}$, $j = \overline{1,k}$, atunci pentru o înregistrare I_i se consideră pentru transfer numai valorile sale penru cîmpurile c_j, $j = \overline{1,k}$, deci $\text{cimp}_{\alpha_1}(I_1), \ldots, \text{cimp}_{\alpha_k}(I_i)$. Situațiile sînt similare ca cele de mai sus, înlocuind n cu k.

Dacă este prezentă una sau mai multe din componentele <domeniu>, FOR <condiție$_1$>, WHILE <condiție$_2$>, atunci se calculează acel $F_3 = (I_{\beta_1}, I_{\beta_2}, ..., I_{\beta_q})$ ca în cazul comenzii DELETE. Pentru transfer se vor considera atunci numai înregistările din F_3. Apar aceleaşi situații ca în cazul celor de mai sus, înlocuind m cu q.

Comanda ignoră cîmpurile de tip memo.

Copierea unei singure înregistrări a unei baze de date într-un vector sau într-o mulțime de variabile de memorie se realizează cu comanda `SCATTER`, care are forma:

$$\mathtt{SCATTER\ [FIELDS}\ c_1,\ c_2,\ ...,\ c_k\mathtt{][MEMO]}\ \left\{\begin{array}{l} \mathtt{TO}\left\{\begin{array}{l} <\mathtt{numet}> \\ <\mathtt{numet}>\ \mathtt{BLANK} \end{array}\right\} \\ \mathtt{MEMVAR} \\ \mathtt{MEMVAR\ BLANK} \end{array}\right\}$$

unde `<numet>` este un vector (tablou unidimensional) creat anterior prin una din comenzile `DIMENSION`, `DECLARE`, `PUBLIC`.

În cazul cînd tabloul `<numet>` nu a fost creat anterior, atunci această comandă îl va crea.

Fişierul activ F în zona curentă trebuie să fie poziționat pe o anumită înregistrare, să zicem I_j (care se numeşte curentă). Fie cîmp$_1$,..., cîmp$_n$, cîmpurile lui F şi v_l valoarea lui cimp$_l$ pentru înregistrarea I_j, adică v_l = cimp$_l(I_j)$, $l = \overline{1,n}$. Fie h numărul de componente ale lui `<numet>`. Folosirea lui `TO <numet>` şi absența componentei `FIELDS`, înseamnă transferul valorilor v_1, v_2, ... în `<numet>`(1), ... Intervin aceleaşi trei situații între n şi h. Folosirea `TO <numet> BLANK` va determina faptul că elementele tabloului vor primi tipul valorilor transferate (v_j), dar valorile nu vor efectiv transferate.

Dacă se utilizează `MEMVAR`, atunci se crează n variabile de memorie cu acelaşi nume ca şi cîmpurile şi se transferă valorile v_j în variabilele cîmp$_j$, $j = \overline{1,n}$.

Pentru a se face referirea la variabila de memorie cîmp$_j$ vom utiliza în program `m.cimp`$_j$ sau `m`$\rightarrow$`cimp`$_j$.

Utilizarea `MEMVAR BLANK` înseamnă crearea aceloraşi variabile de memorie ca în cazul `MEMVAR`, avînd tipurile aceleaşi cu tipurile cîmpurilor respective, deosebirea provenind din faptul că valorile v_j nu se transferă în variabilele cimp$_j$, deci acestea rămîn neinițializate.

Cînd comanda conține componenta `FIELDS` c_1, c_2, ..., c_k, atunci se vor transfera numai valorile cîmpurilor c_l ale înregistrării curente I_j, $l = \overline{1,k}$. Cîmpurile c_l pot fi şi de tip MEMO.

Dacă lipseşte cuvîntul `MEMO` din comandă, atunci toate cîmpurile de tip memo sînt ignorate în acest transfer. Cînd este prezent `MEMO`, atunci şi cîmpurile de tip memo vor fi considerate în transfer. Evident, pentru aceste cîmpuri trebuie spațiu de memorie corespunzător. Dacă nu există memorie suficientă, atunci apare eroarea "Insufficient memory".

Operația inversă de transfer (din variabile de memorie sau vector în cîmpurile articolului curent) se realizează cu comanda `GATHER`, care are forma:

$$\texttt{GATHER} \begin{Bmatrix} \texttt{FROM < numet >} \\ \texttt{MEMVAR} \end{Bmatrix} \;\texttt{[FIELDS } c_1,\ c_2,\ \ldots,\ c_k\texttt{][MEMO]}$$

unde <numet> este un tablou unidimensional, ale cărui elemente conțin valorile de transferat.

Folosirea componentei MEMVAR înseamnă că există deja create variabilele de memorie cu același nume ca și cîmpurile fișierului activ (cu comanda SCATTER), acestea sînt modificate eventual, apoi noile valori se copie în cîmpurile înregistrării curente.

Dacă se dă componenta FIELDS, atunci se vor copia valorile variabilelor cu numele c_1, c_2, ..., c_k. Dacă lipsește MEMO, atunci cîmpurile de tip memo nu sînt considerate.

Dacă este prezent cuvîntul MEMO, atunci se vor atribui valori și pentru cîmpurile de tip memo din variabilele corespunzătoare.

Atribuirea de valori pentru o variabilă de memorie sau un element al unui tablou se realizează cu una din comenzile:

$$\texttt{STORE <expresie> TO} \begin{Bmatrix} \texttt{< variabila >} \\ \texttt{< element tablou >} \end{Bmatrix}$$

```
<variabilă> = <expresie>
<element tablou> = <expresie>
```

Se evaluează <expresie> și valoarea obținută se atribuie ca valoare pentru <variabilă> sau <element tablou>.

Se poate utiliza și o comandă de forma:

$\texttt{STORE <expresie> TO <var}_1\texttt{>, ..., <var}_h\texttt{>}$

în care $\texttt{<var}_i\texttt{>}$ sînt variabilele ce vor primi valoarea din <expresie>.

Despre variabilele publice și locale vom discuta cu ocazia procedurilor și funcțiilor.

Facem observația că la prima asignare a unei valori pentru o variabilă sau un element din tablou, va rezulta tipul acestuia ca fiind egal cu tipul expresiei ce a realizat asignarea. O atribuire ulterioară cu o valoare de alt tip decît cel inițial va genera o eroare.

CAPITOLUL IV

STRUCTURI DE CONTROL

În construcția programelor, în afară de comenzile obişnuite, intervin şi comenzi, numite structuri de control, ce implică executarea în general a mai multor comenzi funcție de o expresie logică.

a) **Structura IF** are două forme: IF complet şi IF incomplet.

Structura IF complet are forma:

```
IF <condiție>
     D1
ELSE
     D2
ENDIF
```

unde `<condiție>` este o expresie logică, formată cu cîmpuri ale fişierelor active, variabile, constante, funcții. D_1, D_2 sînt şiruri de comenzi, ce pot conține la rîndul lor, de asemenea, structuri de control.

Executarea acestei structuri se realizează astfel: dacă `<condiție>` este .T. atunci se execută comenzile din D_1, apoi se trece la comanda de după `ENDIF`, adică se termină executarea structurii. Dacă `<condiție>` este .F., atunci se execută comenzile din D_2, apoi se trece la comanda după `ENDIF`.

Dacă şirurile D_1 sau D_2 conțin la rîndul lor structuri de tip `IF`, atunci vom spune că avem structuri `IF` imbricate. Numărul de nivele de imbricare a structurilor nu este limitat.

Exemplu. Fie fişierul PERSONAL cu structura:
MARCA, NUME, PRENUME, SALARIU, SECTIE, NRCOPII, DATA_ANG în care cîmpul SECTIE este numeric şi are valorile 1, 2, 3 reprezentînd secțiile "INTRETINERE", "APROVIZIONARE", "DESFACERE" respectiv. Pentru înregistrarea 1 dorim să afişăm denumirea secției şi nu codul secției. Pentru aceasta vom da secvența:

```
USE PERSONAL
GO 1
IF SECTIE=1
     ?'INTRETINERE'
ELSE
     IF SECTIE = 2
          ?'APROVIZIONARE'
     ELSE
          IF SECTIE = 3
               ?'DESFACERE'
          ELSE
               ?'NUMAR SECTIE INCORECT'
          ENDIF
     ENDIF
ENDIF
```

Comanda `'?'` va realiza afişarea constantelor respective. În cazul în care valoarea cîmpului SECTIE pentru înregistrarea 1 este diferită de 1, 2, sau 3, atunci se afişează mesajul 'NUMAR SECTIE INCORECT'. Se observă perechile de cuvinte rezervate `IF`, `ENDIF`. Cuvîntul `ENDIF` se numeşte delimitator final al structurii, iar `IF` delimitator inițial al structurii.

Structura IF incompletă are forma:

```
IF <condiție>
      D
ENDIF
```

în care <condiție> este o expresie logică, iar D este un şir de comenzi, ce pot fi şi structuri de control.

Executarea acestei structuri se face astfel: se evaluează <condiție>; dacă ea este adevărată, atunci în continuare se execută comenzile din D, apoi se trece la comanda de după `ENDIF`, deci la terminarea execuției structurii. Dacă <condiție> este falsă, atunci se trece la comanda de după `ENDIF`.

Exemplu: aceeaşi afişare ca în exemplul anterior se poate realiza în mai multe structuri `IF` incomplete:

```
USE PERSONAL
GO 1
IF SECTIE=1
      ?'INTRETINERE'
ENDIF
IF SECTIE=2
      ?'APROVIZIONARE'
ENDIF
IF SECTIE=3
      ?'DESFACERE'
ENDIF
IF (SECTIE<1).OR.(SECTIE>3)
      ?'NUMAR SECTIE INCORECT'
ENDIF
```

b) **Structura DO WHILE** are forma generală:

```
DO WHILE <condiție>
      D
ENDDO
```

unde <condiție> este o expresie logică, iar D este un şir de comenzi, ce pot fi şi structuri de control.

Executarea acestei structuri se realizează astfel: se evaluează <condiție>, apoi dacă aceasta este .T. se execută comenzile din D şi se revine din nou la evaluarea <condiției>. În momentul în care <condiție> este .F. se trece la comanda de după `ENDDO`, adică la terminarea executării structurii. Rezultă că putem avea numai una din situațiile:

I) La prima evaluare <condiție> este .F.

II) Există un număr natural, j>0, astfel încît la evaluările de număr 1, 2, ..., j <condiție> este .T., iar la evaluarea de număr j+1, <condiție> este .F.

III) Pentru toate evaluările 1, 2, ... <condiție> este .T.

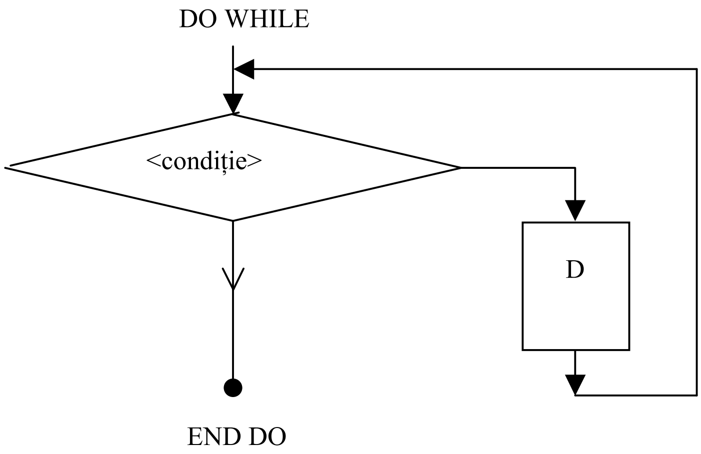
În situația I) comenzile din D nu se execută niciodată, în situația II) comenzile din D se execută de j ori, iar în situația III) comenzile din D se execută la nesfîrşit, spunem că în această situație programul buclează datorită neterminării executării acestei structuri.

Desigur, situația III) trebuie evitată de programator.

Rezultă că elementele din care este construită `<condiție>` (variabile, cîmpuri) variază odată cu succesiunea de evaluări pentru `<condiție>`. Utilizatorul va construi o astfel de structură cu siguranța ca la un moment dat să fie asigurat faptul că `<condiție>` va deveni falsă, după un număr finit de evaluări ale ei, pentru ca structura să se termine la un moment dat.

D se numește domeniul structurii. În D putem folosi comenzi speciale şi anume: `EXIT`, ce realizează saltul la comanda de după `ENDDO`, deci este o ieșire forțată din D pentru terminare, şi `LOOP` care întrerupe execuția lui D şi reia evaluarea `<condiției>`.

Deci, fără `EXIT` şi `LOOP` în D, executarea structurii se face:



Exemplu: Pentru un număr natural n să calculăm $\sum_{i=1}^{n} i^2$.

Presupunem că pentru citirea lui n utilizăm comanda `@SAY, GET`.

```
CLEAR
N=0
@1,1 SAY 'DATI NUMARUL NATURAL:' GET N
READ
I=1
S=0
DO WHILE I<=N
     S=S+I*I
     I=I+1
ENDDO
?'SUMA PATRATELOR PENTRU I=1 PINA LA:', N, 'ESTE =', S
```

În exemplul de mai sus, inițializarea lui I cu 1, apoi mărirea lui I cu o unitate în structura `DO WHILE` ne va asigura că după N+1 evaluări, condiția I<=N devine falsă, deci se asigură terminarea executării structurii.

În cazul în care N≤0, la prima evaluare condiția este falsă, deci S rămîne cu valoarea 0.

Evident, dacă N>0 şi în domeniu în loc de I=I+1 vom da I=I–1, deci scăderea valorii lui I cu 1, mereu, atunci condiția va fi adevărată mereu, deci programul buclează în interiorul acestei structuri.

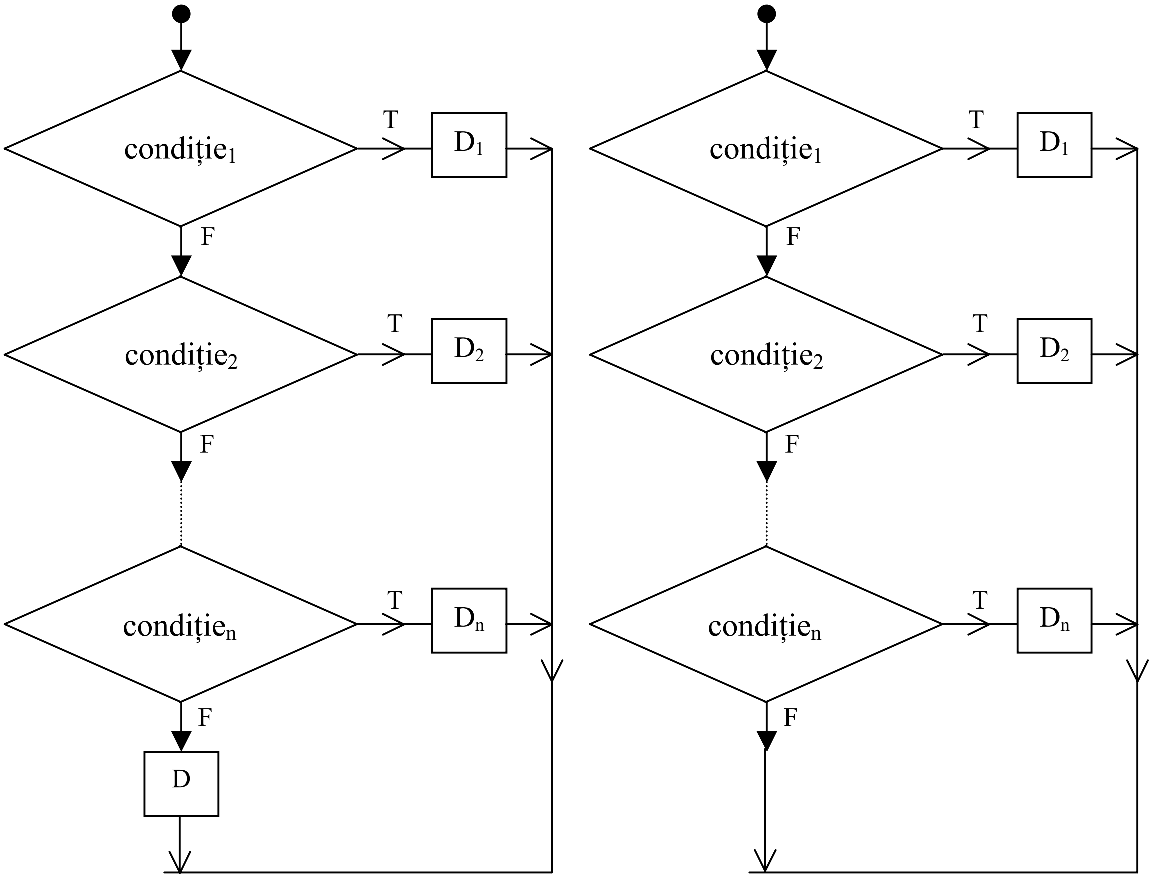
c) **Structura DO CASE**. Are forma generală:

```
DO CASE
     CASE <condiție1>
          D1
```

```
      CASE <condiţie2>
           D2
      .....
      CASE <condiţien>
           Dn
      [OTHERWISE
           D]
ENDCASE
```

în care `<condiţiei>`, $i=\overline{1,n}$ sînt expresii logice, D_i, $1 \le i \le n$ sînt şiruri de comenzi care nu pot conţine structuri `DO CASE`. D este de asemenea un şir de comenzi, ce nu poate conţine structuri `DO CASE`.

Executarea acestei structuri se realizează astfel: prima corespunde prezenţei lui `OTHERWISE`, iar a doua absenţei acestei părţi.



În cazul prezenţei lui `OTHERWISE D`, fie `<condiţiei>` prima care este .T. (deci `<condiţie1>` = .F.,..., <condiţie$_{i-1}$>=.F.); atunci se execută domeniul D_i şi se termină structura. Dacă toate condiţiile sînt false, atunci se execută D. Cînd lipseşte `OTHERWISE D`, atunci diferenţa faţă de precedenta constă în cazul cînd toate `<condiţiei>` sînt .F., în acest caz se termină executarea structurii.

Exemplul precedent realizat cu o structură `DO CASE`:

```
USE PERSONAL
GO 1
DO CASE
      CASE SECTIE=1
           ?'INTRETINERE'
      CASE SECTIE=2
```

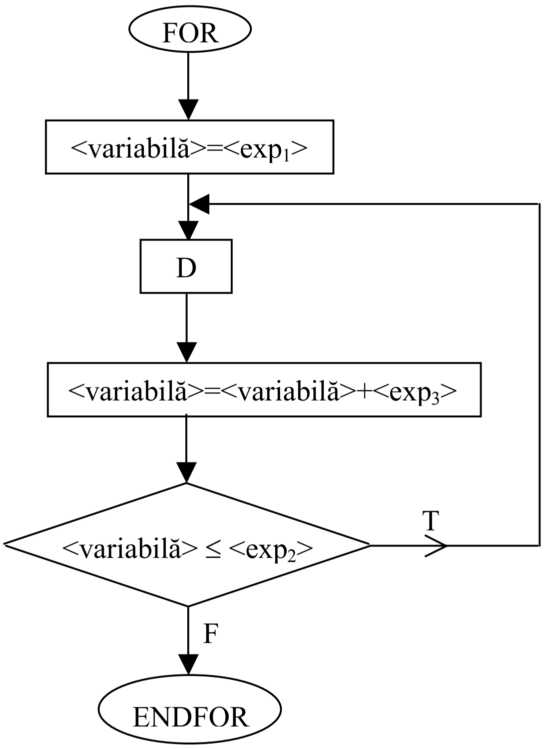
```
          ?'APROVIZIONARE'
     CASE SECTIE=3
          ?'DESFACERE'
     OTHERWISE
          ?'NUMAR SECTIE INCORECT'
ENDCASE
```

d) **Structura FOR**. Forma generală:

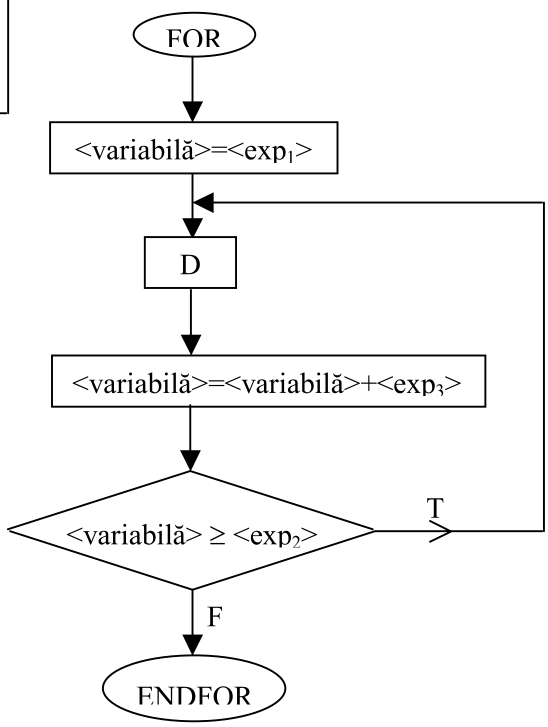
```
FOR <variabilă>=<exp1> TO <exp2> [STEP <exp3>]
     D
ENDFOR
```

unde `<variabilă>` este o variabilă de memorie numită contor. <exp$_i$> i=1,2,3 sunt expresii. <exp$_1$> dă valoarea inițială a variabilei, <exp$_2$> valoarea finală pentru variabilă, iar <exp$_3$> pasul de creştere a contorului. Dacă lipseşte componenta STEP <exp$_3$>, atunci pasul de creştere va fi 1. Dacă <exp$_3$> este pozitivă, atunci <exp$_1$>≤<exp$_2$>, iar dacă <exp$_3$> este negativă, trebuie să avem <exp$_1$>≥<exp$_2$>.

Pentru cazul <exp$_3$>>0, executarea structurii se face:



Pentru cazul cînd <exp$_3$> < 0, atunci avem:



Domeniul D al structurii poate conține structuri de control, inclusiv structuri `FOR`, precum şi comenzile `LOOP` şi `EXIT`. Comanda `LOOP` are ca efect reluarea executării domeniului D de la început, iar comanda `EXIT` realizează saltul după `ENDFOR`, deci terminarea forțată a structurii. În locul cuvîntului rezervat `ENDFOR` se poate folosi cuvîntul `NEXT`.
Exemplu. Fie tabloul TAB definit prin:
```
        DECLARE TAB(10, 20)
```
şi presupunem că primele M linii ($1 \leq M \leq 10$) şi primele N coloane ($1 \leq N \leq 20$) au primit valori prin diverse comenzi (de atribuire, de intrare-ieşire etc.).

Vrem să calculăm suma elementelor lui TAB pentru liniile 1,..., M şi coloanele 1, ..., N. Fie SUMA variabilă de lucru, ce va avea în final suma dorită.
```
SUMA=0
FOR I=1 TO M
        FOR J=1 TO N
                SUMA=SUMA+A(I, J)
        ENDFOR
ENDFOR
```
Dacă pentru aceeaşi matrice vrem să calculăm suma elementelor din fiecare linie şi s-o plasăm în LSUMA:
```
DECLARE LSUMA(10)
FOR I=1 TO M
        LSUMA(I)=0
        FOR J=1 TO N
                LSUMA(I)=LSUMA(I)+A(I, J)
        ENDFOR
ENDFOR
```
Expresiile <exp_1>, <exp_2>, <exp_3> trebuie să fie de acelaşi tip. <variabila> va căpăta tipul acestora.

e) **Structura SCAN**. Forma generală:

```
SCAN [<domeniu>][FOR <condiție1>][WHILE <condiție2>]
        D
ENDSCAN
```

unde <domeniu>, FOR <condiție$_1$>, WHILE <condiție$_2$> au aceeaşi sintaxă şi semnificație ca şi în cazul comenzii `DELETE` (ce realiza marcarea înregistrărilor pentru ştergere).
Fie F_R fişierul activ în zona curentă şi F înregistrările acestuia. $F = (I_1, I_2, ..., I_n)$. Dacă lipsesc componentele <domeniu>, `FOR`, `WHILE`, atunci se execută comenzile din D pe rînd pentru fiecare înregistrare din F, în ordinea $I_1, I_2, ..., I_n$. D se numeşte domeniul structurii `SCAN`. Dacă este prezentă una din cele trei componente, atunci se calculează ca şi în cazul comenzii `DELETE`, pe rînd vectorii F_1, F_2, F_3. Fie $F_3 = (I_{\alpha_1}, I_{\alpha_2}, ..., I_{\alpha_k})$. Atunci instrucțiunile din D se vor executa pentru fiecare înregistrare din F_3, în ordinea $I_{\alpha_1}, I_{\alpha_2}, ..., I_{\alpha_k}$. Dacă $F_3=\emptyset$, atunci nu se execută comenzile din D. Domeniul D poate conține comenzile `LOOP` şi `EXIT`. Comanda `LOOP` executată pentru înregistrarea I_j are rolul de a trece la executarea comenzilor de la începutul lui D, dar pentru înregistrarea următoare I_{j+1}. Comanda

`EXIT` realizează un salt în exteriorul structurii, adică după `ENDSCAN`, deci înregistrările următoare nu vor fi tratate.

Exemplu: să afişăm înregistrările fişierului PERSONAL pentru persoanele din Secţia 1:

```
USE PERSONAL
SCAN ALL FOR SECTIE=1
     ?MARCA, NUME, PRENUME, SALARIU, SECTIE
ENDSCAN
USE
```

f) **Structura TEXT**. Formatul general:

$$
\begin{array}{l}
\texttt{TEXT} \\
\quad \texttt{l}_1 \\
\quad \texttt{l}_2 \\
\quad \ldots \\
\quad \texttt{l}_n \\
\texttt{ENDTEXT}
\end{array}
$$

Liniile $\texttt{l}_1$, $\texttt{l}_2$,..., $\texttt{l}_n$ pot conţine text (succesiune de caractere), variabile de memorie, expresii, funcţii, sau orice combinaţii ale acestora. Dacă există comanda `SET TEXTMERGE OFF`, atunci liniile $\texttt{l}_i$, $i = \overline{1,n}$ sînt afişate exact în forma scrisă. Afişarea se face pe ecran dacă `SET CONSOLE ON`; la imprimantă, dacă există `SET PRINTER TO` sau SET PRINTER TO FILE urmată de SET PRINTER ON. Dacă `SET CONSOLE OFF` afişarea nu se face pe ecran. Dacă `SET TEXTMERGE ON`, atunci variabilele de memorie, expresiile, funcţiile sînt delimitate standard la început prin << iar la sfîrşit prin >> şi se afişează rezultatul evaluării, nu expresia respectivă. Utilizatorul poate defini delimitatorii proprii prin comanda:

$\texttt{SET TEXTMERGE DELIMITERS TO <expC}_1\texttt{>, <expC}_2\texttt{>}$

Primul este delimitatorul stînga iar al doilea delimitatorul dreapta. Dacă se specifică numai $\texttt{<expC}_1\texttt{>}$, atunci delimitatorul dreapta este acelaşi cu delimitatorul stînga şi egal cu $\texttt{<expC}_1\texttt{>}$. Comanda `SET TEXTMERGE` este:

$$
\texttt{SET TEXTMERGE} \begin{Bmatrix}\text{ON}\\ \text{OFF}\end{Bmatrix} \texttt{[TO <fişier>[ADDITIVE]]}
$$

$$
\quad \texttt{[WINDOW <numefer>]} \begin{Bmatrix}\text{SHOW}\\ \text{NOSHOW}\end{Bmatrix}
$$

`ON` implică evaluarea expresiilor, `OFF` – fără evaluare.

`TO <fişier>` implică plasarea rezultatului în `<fişier>`. Dacă se dă `ADDITIVE`, liniile $\texttt{l}_i$, $i = \overline{1,n}$ se adaugă în `<fişier>`. `WINDOW <numefer>` are ca efect afişarea liniilor $\texttt{l}_i$, $i = \overline{1,n}$ în fereastra specificată (definită cu `DEFINE WINDOW`). `SHOW` – afişarea pe ecran sau în fereastră a ieşirilor, `NOSHOW` – inhibă afişarea ieşirilor.

CAPITOLUL V
OPERAȚII DE INTRARE - IEŞIRE

SGBD-ul FOX are la dispoziție comenzi ce realizeză numai afişarea pe ecran sau tipărirea la imprimantă, comenzi numite de ieşire, precum şi comenzi ce afişează un mesaj şi aşteaptă un răspuns din partea utilizatorului, acestea se numesc comenzi de intrare–ieşire.

Să ne ocupăm întîi de cea de-a doua categorie.

a) **Comanda ACCEPT**. Forma generală:

`ACCEPT <expC> TO <variabilă>`

În care `<expC>` este o expresie de tip caracter, iar `<variabilă>` este o variabilă de memorie. Această variabilă este considerată de tip caracter. Comanda afişează valoarea `<expC>` pe ecran şi aşteaptă un şir de caractere ca răspuns din partea utilizatorului. Fie $c_1c_2...c_k$ acest răspuns. Atunci `<variabilă>` va primi ca valoare $c_1c_2...c_k$. În caz particular `<expC>` poate fi un literal şi atunci are forma '$d_1...d_h$', d_i – caracter. Răspunsul utilizatorului este terminat cu tasta `ENTER`. Dacă răspunsul utilizatorului este format numai din `ENTER`, atunci se spune că răspunsul este şirul vid şi `<variabilă>` va avea ca valoare şirul vid (un şir de caractere de lungime zero).

Exemplu: să presupunem că dorim să adăugăm o nouă înregistrare la sfîrşit în fişierul PERSONAL cu date citite cu comanda `ACCEPT`. Să precizăm tipurile cîmpurilor: MARCA N(5), NUME C(10), PRENUME C(12), SALARIU N(7), NRCOPII N(1), FUNCTIE C(9), DATA_ANG D.

Deoarece cu `ACCEPT` valoarea primită este memorată sub formă de şir de caractere, vom avea nevoie de o funcție ce converteşte un şir de caractere într-un număr (VAL) şi o funcție ce converteşte un şir într-o dată calendaristică (`CTOD`).

```
USE PERSONAL
ACCEPT 'DATI MARCA:' TO WMARCA
ACCEPT 'DATI NUMELE:' TO WNUME
ACCEPT 'DATI PRENUMELE:' TO WPRENUME
ACCEPT 'DATI SALARIUL:' TO WSALARIU
ACCEPT 'DATI NUMARUL DE COPII:' TO WNRCOPII
ACCEPT 'DATI FUNCTIA:' TO WFUNCTIE
ACCEPT 'DATI DATA ANGAJARII:' TO WDATA_ANG
APPEND BLANK
REPLACE MARCA WITH VAL(WMARCA)
REPLACE NUME WITH WNUME
REPLACE PRENUME WITH WPRENUME
REPLACE SALARIU WITH VAL(WSALARIU)
REPLACE NRCOPII WITH VAL(WNRCOPII)
REPLACE FUNCTIE WITH WFUNCTIE
REPLACE DATA_ANG WITH CTOD(WDATA_ANG)
DISPLAY
```

Expresia `VAL(WMARCA)` are ca valoare numărul dat de utilizator pentru cîmpul MARCA, sub forma unui şir de maximum 5 cifre. Funcția `CTOD` converteşte data angajării furnizată de utilizator sub forma {ll/zz/aa} într-o dată calendaristică, deoarece WDATA_ANG este de tip C, iar DATA_ANG este de tip D.

b) **Comanda INPUT**. Forma generală:

`INPUT <expC> TO <variabilă>`

unde `<expC>` este o expresie de tip caracter, iar `<variabilă>` este o variabilă de memorie. Deci sintaxa este aceeaşi ca a comenzii `ACCEPT`. Deosebirea va consta în modul de lucru. Ca şi `ACCEPT`, comanda `INPUT` afişează pe ecran valoarea expresiei `<expC>` (ce joacă rolul unui mesaj de avertizare a faptului că utilizatorul trebuie să tasteze o valoare). Dar răspunsul utilizatorului în acest caz trebuie să fie o expresie. Comanda evaluează acestă expresie şi valoarea rezultată o atribuie variabilei. Deci tipul variabilei va fi acelaşi cu tipul expresiei răspuns. Dacă expresia răspuns este o constantă, atunci dacă ea este de tip numeric, utilizatorul va da f_1, f_2, ..., f_k, unde f_i sînt cifre; dacă constanta este de tip C, utilizatorul va da 'c_1, ...c_k' unde c_i sînt caractere, iar dacă constanta este de tip D, utilizatorul va da {zz/ll/aa}.

Exemplu: dacă vrem să adăugăm la sfîrşit o înregistrare în fişierul PERSONAL, utilizînd comenzi `INPUT`, atunci;

```
USE PERSONAL
INPUT 'DATI MARCA' TO WMARCA
INPUT 'DATI NUMELE' TO WNUME
… similar ca la ACCEPT
APPEND BLANK
REPLACE MARCA WITH WMARCA
REPLACE NUME WITH WNUME
REPLACE PRENUME WITH WPRENUME
REPLACE SALARIU WITH WSALARIU
REPLACE NRCOPII WITH WNRCOPII
REPLACE FUNCTIE WITH WFUNCTIE
REPLACE DATA_ANG WITH WDATA_ANG
```

cu precizarea că pentru valorile de tip C, utilizatorul le va da între caracterele `','` (spre deosebire de `ACCEPT` unde nu se va da '), iar data angajării se va da de forma {ll/zz/aa}. Atunci WMARCA, WSALARIU, WNRCOPII vor fi de tip N, iar WDATA_ANG va fi de tip D. Prin urmare în acest caz nu sînt necesare funcțiile de conversie `VAL`, `CTOD`.

c) **Comanda WAIT**. Forma generală:

```
WAIT [<expC>] TO <variabilă> [WINDOW [NOWAIT]]
     [TIMEOUT <expN>] [CLEAR]
```

`<expC>` este o expresie de tip caracter, valoarea acesteia se va afişa pe ecran. Dacă `<expN>` lipseşte, atunci pe ecran se va fişa mesajul "Press any key to continue". Răspunsul utilizatorului poate fi `ENTER` sau c `ENTER`, unde c este un caracter. În primul caz răspunsul este şirul nul, iar `<variabilă>` va primi ca valoare şirul nul. În al doilea caz `<variabilă>` va primi ca valoare unicul caracter c. Folosirea cuvîntului `WINDOW` implică afişarea valorii pentru <expC> sau a mesajului standard "Press any key to continue" într-o fereastră sistem situată în colţul din dreapta sus al ecranului. Apăsarea unei taste duce la ştergerea mesajului. Dacă se foloseşte `NOWAIT`, atunci apăsarea unei taste nu va produce ştergerea mesajului de pe ecran, iar <variabilă> va conţine şirul nul. Folosirea componentei `TIMEOUT` <expN> înseamnă că se aşteaptă <expN> secunde pînă cînd utilizatorul tastează un răspuns. Folosirea cuvîntului `CLEAR` implică ştergerea ferestrei sistem.

d) **Comanda @ SAY/GET**. Forma generală:

@ l, c SAY <e_1> [PICTURE <expC1>] [FUNCTION <cod_1>]

$$\left[\begin{Bmatrix}\texttt{COLOR SCHEME}\langle\texttt{expN1}\rangle \\ \texttt{COLOR}\ \ c_1 / d_1, \ldots, c_h / d_h\end{Bmatrix}\right]$$

GET <variabilă>

[PICTURE <expC2>] [FUNCTION <cod_2>]

[DEFAULT <e_2>]

$$\left[\begin{Bmatrix}\texttt{ENABLE} \\ \texttt{DISABLE}\end{Bmatrix}\right]$$ [MESSAGE <expC3>] [OPEN] WINDOW <numefer>]

[RANGE <e_3>, <e_4>][SIZE <expN2>, <expN3>]

[VALID <expL1> [ERROR <expC4>]]

[WHEN <expL2>]

$$\left[\begin{Bmatrix}\texttt{COLOR SCHEME}\ \langle\texttt{expN4}\rangle \\ \texttt{COLOR}\ \ c_1 / d_1, \ldots, c_h / d_h\end{Bmatrix}\right]$$

l, c constituie coordonatele unui punct de pe ecran de unde începe afişarea valorii expresiei <e_1>. De obicei ecranul ese împărţit în 25 linii şi 80 coloane.

Liniile sînt numerotate de la 0 la 24, iar coloanele de la 0 la 79. Deci coordonatele colţului din stînga sus vor fi 0, 0 cele ale colţului din dreapta sus 0, 79, cele ale colţului din stînga jos 24, 0, iar cele ale colţului din dreapta jos 24, 79.

Componenta `SAY` realizează afişarea valorii expresiei <e_1> după anumite reguli date de `PICTURE` şi `FUNCTION`. Dacă avem `SET DEVICE TO SCREEN`, atunci valoarea lui <e_1> se afişează pe ecran, iar dacă avem `SET DEVICE TO PRINT`, atunci valoarea lui <e_1> este afişată la imprimantă. Dacă există `PICTURE` <expC1>, atunci <expC1> este un şir de caractere, numite caractere şablon, ce precizează anumite reguli de afişare. <expC1> poate conţine şi un cod de funcţie, notat cu f. În acest caz, valoarea <expC1> are forma @f c_1c_2 ...c_k, unde c_i sînt caractere şablon. Dacă este prezentă `FUNCTION` <cod_1>, atunci <cod_1> este o expresie ce are ca valoare un număr de coduri, deci de forma f_1f_2 ...f_k, unde f_i sînt coduri. Caracterele şablon şi codurile le vom discuta după ce discutăm celelalte componente.

Componenta `COLOR` defineşte modul de colorare.

O schemă de colorare se compune din 10 perechi de culori. O culoare este simbolizată prin una sau două litere rezultate din denumirea culorii în engleză: negru – N, albastru – B, verde – G, cyan – BG, roşu – R, magenta – BR, galben – GR, alb – W, invizibil – X.

Pentru o pereche de culori de forma c_i/d_i, culoarea c_i reprezintă culoarea de fond iar culoarea d_i este culoarea scrisului. După o culoare poate să apară caracterul + ce reprezintă intensificarea culorii, sau carcaterul * ce reprezintă pîlpîire. Există diverse scheme de colorare standard care sînt identificate printr-un număr de la 1 la 24.

Referitor la culori există comenzile: `SET COLOR`, `SET COLOR OFF`, `SET COLOR OF SCHEME`, `SET COLOR SET` şi `SET COLOR TO`. Acestea sînt explicate şi în cartea "FOXPRO – comenzi şi funcţii" de Lia Chiorean.

Componenta `GET` a comenzii se utilizează pentru primirea răspunsului utilizatorului. După un număr de comenzi `@ SAY/GET` este obligatorie utilizarea comenzii `READ` ce va realiza toate operaţiile de intrare/ieşire definite de comenzile `@SAY/GET`. Dacă componentele `PICTURE` şi `FUNCTION` prezente în partea `SAY` determină modul de afişare a valorii expresiei <e_1>, componentele `PICTURE` şi `FUNCTION` prezente în componenta `GET` sînt utilizate pentru data tastată de utilizator (data de intrare) ce va fi plasată ca valoare pentru <variabilă>. <variabilă> poate fi o variabilă de memorie, dar şi un cîmp al unei baze de date. În cazul în care este vorba de un cîmp şi comanda conţine `DEFAULT` <e_2> şi utilizatorul răspunde cu `ENTER`, atunci în cîmpul respectiv se va plasa valoarea expresiei <e_2>.

Componenta `ENABLE/DISABLE` permite sau împiedică modificarea valorii pentru variabilă.

Componenta `MESSAGE <expC3>` permite afişarea valorii <expC3>, care este un şir de caractere, pe ultima linie a ecranului, atunci cînd cursorul este plasat pe cîmpul de primire a valorii de răspuns corespunzătoare.

Componenta `OPEN WINDOW <numefer>` permite editarea unui cîmp de tip memo, într-o fereastră de editare definită în prealabil cu `DEFINE WINDOW <numefer>`. Dacă se utilizează `OPEN WINDOW <numefer>` atunci se deschide automat fereastra cînd cursorul este poziţionat pe acel cîmp de tip memo. Dacă se specifică numai `WINDOW <numefer>` atunci poziţionarea cursorului pe cîmpul memo se face prin CTRL+HOME, CTRL+PgUp şi CTRL+PgDn. Ieşirea din editarea cîmpului memo se realizează prin CTRL+W, CTRL+END sau ESC.

Componenta `RANGE` se utilizează pentru cazurile în care <variabilă> este de tip C, D sau N şi defineşte un domeniu permis pentru valoarea tastată. Dacă notăm cu v valoarea tastată de utilizator, atunci $e_3 \le v \le e_4$. Dacă v nu aparţine domeniului specificat atunci apare un mesaj de eroare, care afişează domeniul. Se poate utiliza `RANGE` <e_3>, în acest caz se verifică $e_3 \le v$. Se poate utiliza forma RANGE , <e_4>, în acest caz se verifică $v \le e_4$.

Componenta `SIZE` specifică dimensiunea zonei de editare. <expN2> specifică numărul de linii de pe ecran, iar <expN3> dă numărul de coloane al zonei de editare. În absenţa componentei `SIZE` se consideră <expN2> = 1, iar numărul de coloane este determinat de lungimea valorii variabilei sau a cîmpului definit de <variabilă>, precum şi de clauza `PICTURE` dacă există. Anume:

Dacă nu există clauza PICTURE, atunci:

- dacă se editează un cîmp al unei baze de date de lungime l_1 şi fie <expN3> din SIZE de valoare l_2, atunci în situaţia $l_2>=l_1$, editarea se face pe o zonă de lungime l_1;
- Dacă se editează o variabilă de memorie de lungime l_1 şi fie l_2 ca mai sus şi $l_2>=l_1$, atunci editarea se face pe o zonă de lungime l_2. Dacă $l_2<l_1$, atunci editarea valorii variabilei se face pe lungimea l_2, dar prin defilarea caracterelor în zona de editare.

În situaţia cînd clauza PICTURE este prezentă avem:

- Fie l_3 dimensiunea specificată de PICTURE (numărul de caractere din <expC2>) şi l_2 lungimea dată de SIZE.

Dacă $l_3 \leq l_2$, atunci se editează numai primele l_3 caractere ale variabilei de memorie sau a cîmpului.

Dacă $l_3 > l_2$, se editează întreaga valoare de editat prin defilare în interiorul unei zone de lungime l_2.

Componenta VALID permite validarea valorii tastate de utilizator. Dacă <expL1> este .T. atunci valoarea tastată este acceptată drept valoare pentru <variabilă>. <expL1> poate conţine apeluri ale unor funcţii definite de utilizator (numite UDF), care sînt constituite într-un text sursă separat şi realizează aşa numitele operaţii de validare a valorii tastate. Cînd <expL1> este .F. se afişează un mesaj de eroare, procesul de răspuns continuă prin tastarea valorii corecte, tastînd în prealabil un spaţiu. Dacă utilizatorul doreşte în acest caz să se afişeze un mesaj propriu, va da acest mesaj ca valoare pentru <expC4> şi va folosi ERROR <expC4>.

Componenta WHEN <expL2> permite sau înhibă citirea unui cîmp folosind @SAY/GET. Dacă <expL2> = .T. atunci citirea este permisă, în caz contrar se trece la următorul cîmp definit de următoarea comandă @SAY/GET.

Comanda COLOR din SAY permite definirea culorilor pentru zonele de pe ecran, în care se afişează valoarea lui <e_1>, iar COLOR din GET permite definirea atributelor de culoare pentru zona de pe ecran unde se vizualizează răspunsul dat de utilizator.

Să precizăm acum codurile de funcţie:

A – admite numai caractere alfabetice;

B – cîmpul numeric va fi aliniat la stînga (în mod standard cîmpul numeric este aliniat la dreapta).

C – se afişează CR(credit) după un număr pozitiv (se poate utiliza numai cu date numerice şi în SAY).

D - înseamnă utilizarea formatului de dată curent definit prin SET DATE, pentru editarea datelor de tip D.

E – data calandaristică se editează în format european: zz/ll/aa.

I – valoarea este centrată în cîmpul de editare.

J – valoarea afişată (<e_1>) este aliniată la dreapta în cîmpul de afişare.

L – zerourile din faţa cifrelor sînt afişate în loc de spaţii.

T – suprimă spaţiile de la început şi de la sfîrşitul valorii de afişare.

X – afişează DB (debit) după numere negative. Se utilizează numai cu SAY.

Z – afişează spaţii dacă valoarea numerică este zero.

(-- plasează între paranteze rotunde valorile negative.

! – caracterele alfabetice sînt transformate în litere mari.

^ - afişează datele numerice în format ştiinţific.

$ - afişează simbolul monetar în faţa valorii numerice dacă SET CURRENCY LEFT şi după valoarea numerică, dacă SET CURRENCY RIGHT.

Să precizăm acum cîteva caractere şablon (ce apar în PICTURE).

A – corespunde unui caracter alfabetic.

L – permite numai date logice.

N – permite litere şi cifre.

X – permite orice caracter.

Y – permite numai valorile logice Y, y, N, n pe care le converteşte în litere mari.

9 – permite cifre (sau semn pentru cele numerice).

- permite cifre, spaţii şi semne.

! – converteşte litera mică respectivă în literă mare.

* - afişează asteriscuri în faţa valorii numerice.

. – afişează marca zecimală.

, - afişează caracterul ,

Există comenzi de forma @ l, c GET <variabila> ... care definesc aşa numite obiecte de control (comutatoare, butoane invizibile, butoane de pornire, butoane radio, crearea unui liste, crearea unui popup).

Să discutăm în continuare câteva comenzi de afişare.

a) **Comenzile ? şi ??**. Formatul general:

$\begin{Bmatrix} ? \\ ?? \end{Bmatrix}$ `<exp1>[PICTURE <expC1>][FUNCTION <expC2>]`

`[AT <expN>], <exp2> [PICTURE <expC3>] [FUNCTION <expC4>]...`

Comenzile realizează afişarea valorilor expresiilor `<exp1>`, `<exp2>`,... pe ecran sau la imprimantă. Comanda `?` realizează afişarea pe linia următoare liniei în care se găseşte cursorul, iar `??` realizează afişarea în poziţia curentă a cursorului. Dacă `SET PRINTER` este ON, rezultatul comenzilor se direcţionează şi spre imprimantă. Dacă `SET CONSOLE` este OFF şi `SET PRINTER` este ON, rezultatul se afişează numai la imprimantă. Dacă este prezentă componenta `PICTURE <expC1>`, atunci valoarea expresiei `<exp1>` se afişează conform şabloanelor din `<expC1>`. Dacă este prezentă componenta `FUNCTION <expC2>` atunci în afişarea valorii `<exp1>` se ţine cont de funcţiile din `<expC2>`. În afara codurilor de funcţie discutate mai sus (la comanda `@SAY/GET`) se poate folosi aici funcţia de forma V n ce permite afişarea pe verticală, n fiind numărul maxim de coloane utilizate pentru afişare. Componenta `AT<expN>` se foloseşte pentru a preciza coloana de unde să înceapă afişarea valorii `<exp1>`. Similar se pot utiliza aceleaşi componente pentru afişarea valorilor `<exp2>`, `<exp3>`, ...

b) **??? <expC>** Se evaluează `<expC>` (expresie de tip caracter) şi se trimite valoarea la imprimantă, fără modificarea capului de scriere.

c) **LIST** – afişează conţinutul unei baze de date. Format:

`LIST [FIELDS` $cimp_1$`, ...,` $cimp_n$`] <domeniu>`

```
[FOR <conditie1>]
[WHILE <conditie2>] [OFF]
[ {TO PRINTER       } ]  [NOCONSOLE]
[ {TO FILE <fisier> } ]
```

Comanda afişează informaţia continuu, fără oprire.

Se referă la baza de date activă în zona curentă. În absenţa componentei `FIELDS` se afişează valorile tuturor cîmpurilor. Prezenţa lui `FIELDS` implică afişarea valorilor numai pentru aceste cîmpuri.

Dacă lipseşte `<domeniu>` se consideră `ALL`.

Cînd una din cele trei componente este prezentă (sau mai multe), atunci se calculează mulţimile F_1, F_2, F_3, ca în cazul comenzii `DELETE`. În acest caz se vor afişa numai înregistrările din F_3.

Componenta `OFF` are rolul de a înhiba numărul articolului ce se afişează. Dacă se utilizează `TO PRINTER`, atunci ieşirea comenzii se dirijează spre imprimantă. Dacă se utilizează `TO FILE <fisier>`, atunci ieşirea va fi dirijată în `<fisier>`. Componenta `NOCONSOLE` împiedică afişarea pe ecran a ieşirii comenzii.

d) **LIST FILES** – Afişează informaţii despre fişiere. Format:

```
LIST FILES [ON <director>] [LIKE <expC>]
[ {TO PRINTER       } ]
[ {TO FILE <fisier> } ]
```

Absenţa argumentelor împiedică afişarea informaţiilor despre bazele de date din directorul curent. Componenta `ON <director>` pecizează discul şi directorul de unde dorim să aflăm informaţii despre respective. Utilizarea `LIKE <expC>` înseamnă că dorim informaţii despre anumite fişiere specificate de `<expC>`. Aici * înseamnă orice cuvînt, iar liniuţa de subliniere orice caracter. Ultima componentă are aceeaşi semnificaţie ca la c). Informaţiile obţinute se referă la numele fişierului, numărul înregistrărilor respective, data şi ora ultimei actualizări.

e) **LIST MEMORY**. Afişează conţinutul variabilelor de memorie şi al tablourilor. Formatul:

```
                          [ {TO PRINTER       } ]
LIST MEMORY [LIKE <expC>] [ {TO FILE <fisier> } ]  [NOCONSOLE]
```

Comanda afişează numele, tipul, conţinutul tuturor variabilelor de memorie şi al tablourilor existente, precum şi numărul de octeţi folosiţi. Se afişează şi definiţiile de meniu orizontal, meniu vertical, ferestre.

f) **LIST STATUS**. Afişează starea componentelor sistemului FOXPRO. Formatul genaral:

```
            [ {TO PRINTER       } ]
LIST STATUS [ {TO FILE <fisier> } ]  [NOCONSOLE]
```

Comanda afişează mai multe informaţii printre care: tabelele active, fişierele de index active, cheile de indexare, alias-urile pentru fişiere, relaţiile dintre bazele de date, cîmpurile memo active, fişierele de proceduri active, tipul procesorului, drumurile de căutare definite prin `PATH`, unitatea de disc curentă, zona de lucru curentă, setările definite prin comenzi `SET`.

g) **LIST STRUCTURE**. Afişează structura unei baze de date. Formatul general:

$$\texttt{LIST STRUCTURE}\ \left[\texttt{IN}\begin{Bmatrix}\langle\texttt{expN}\rangle\\ \langle\texttt{expC}\rangle\end{Bmatrix}\right]\ \left[\begin{Bmatrix}\texttt{TO PRINTER}\\ \texttt{TO FILE}\ \langle\texttt{fisier}\rangle\end{Bmatrix}\right]\ \texttt{[NOCONSOLE]}$$

Comanda afişează structura bazei de date, numărul de articole ale ei, precum şi data ultimei actualizări.

Dacă componenta `IN` lipseşte, atunci baza de date este cea activă în zona de lucru curentă. Dacă se foloseşte `IN <expN>` atunci baza de date este cea activă din zona de număr valoarea lui `<expN>`. Dacă se utilizează `IN <expC>`, atunci baza de date este cea activă cu numele alias (definit de `ALIAS`, sau cu aliasul standard al zonei) dat de `<expC>`.

h) **DISPLAY** are aceeaşi formă ca şi `LIST`. Deosebirea constă în faptul că absenţa componentei `<domeniu>` înseamnă aici înregistrarea curentă şi după umplerea unui ecran (la `DISPLAY ALL`) cu informaţii afişate, comanda se întrerupe, continuarea afişării realizîndu-se prin apăsarea unei taste.

CAPITOLUL VI
PROGRAME, PROCEDURI, FUNCȚII

Un program este o succesiune de comenzi în limbajul FOXPRO. Acest program este memorat fizic într-un fişier, numit fişier program, ce are extensia PRG. Crearea(construirea, editarea) unui program se poate realiza în mai multe moduri:

a) În fereastra de comandă vom da:

```
MODIFY COMMAND <nume>
```

unde <nume> va fi numele ce se atribuie noului program. SGBD-ul FOX utilizează primele 8 caractere ale acestui <nume>. Executarea comenzii MODIFY COMMAND implică apariția pe ecran a unei ferestre, numită fereastră de editare, în care utilizatorul va construi programul. Terminarea construirii programului se va face prin tastarea CTRL+W, ce va salva pe disc fişierul (cu extensia PRG) construit. Dacă dorim să anulăm operația de editare, vom tasta ESC.

b) Utilizăm meniul sistemului FOX. Din meniul vertical asociat acestuia selectăm componenta NEW şi în interior avînd posibilitățile Database, Program, File, Index, Report, Label, Screen, Menu, Query, Project. Caracterul . este poziționat pe prima (Database). Cu ajutorul mouse-lui dăm clic pe stînga pe Program şi atunci caracterul . va fi plasat aici. Avem apoi în fereastră butoanele <<OK>> şi <Cancel> şi selectăm butonul <<OK>>. Astfel se deschide aceeaşi fereastră de editare ca şi în cazul b). La terminare vom salva acest program, utilizînd componenta File şi Save as a meniului FOX.

Dacă dorim să modificăm un program şi folosim situația a) atunci vom da aceeaşi comandă, în fereastra de editare va apare programul pe care dorim să-l modificăm. În situația b) vom selecta componenta File, apoi din meniul vertical asociat componenta OPEN. Se afişează toate fişierele DBF şi PROGRAMELE din directorul curent şi vom selecta pe cel dorit.

În fereastra de comandă se poate specifica orice comandă, cu excepția structurilor de control, care sînt permise numai în interiorul programelor.

Programul va fi memorat într-un fişier cu extensia PRG, de aceea vom numi acest program fişier-program. O procedură este un text sursă de forma:

```
PROCEDURE <numepr>
     C1
     C2
     .
     .
     Ch
RETURN
```

unde <numepr> este un nume atribuit procedurii, deci un identificator. SGBD-ul FOX consideră numai primele 8 caractere din <numepr>, dacă acesta are o lungime mai mare ca 8. Comenzile C_1, C_2, …C_h constituie corpul procedurii.

O funcție este un text sursă ce are forma:

```
FUNCTION <numef>
     C1
     C2
     .
     .
     Ch
RETURN <expresie>
```

unde <numef> este un identificator, ce va constitui numele funcției, C1…..Ch constituie comenzi ce se execută la apelul funcției (mulțimea lor se numeşte corpul funcției). Funcția returnează o valoare şi anume valoarea expresiei din RETURN: <expresie>.

Atît pentru o procedură, cît şi pentru o funcție, putem avea parametri formali. Aceştia sînt definiți în comanda PARAMETERS care trebuie să fie plasată în procedură sau funcție imediat după PROCEDURE, respectiv după FUNCTION. Forma generală:

PARAMETERS f_1, f_2,…, f_h

unde f_i sînt parametrii formali ai procedurii sau ai funcției.

Parametrii f_i sunt fie variabile simple, fie tablouri cu un indice sau doi. În ultimul caz, desigur, f_i trebuie să apară într-o comandă DECLARE sau DIMENSION în cadrul aceluiaşi text al procedurii sau funcției. Apelul unei proceduri se realizează sub forma:

DO <numepr> WITH a_1, a_2, …,a_n

Unde <numepr> este numele procedurii, a_i sînt numiți parametri actuali, sau parametri de apel ai procedurii. Din punct de vedere sintactic, ei pot fi constante, variabile simple sau tablouri, expresii. Apelul unei funcții se realizează prin:

<numef> (a_1, a_2, …,a_n)

care este prezentă într-o expresie a programului apelant, în particular într-o comandă de asignare de forma <variabilă> = <numef> (a_1, a_2, … a_n).

Procedura este executată prin înlocuirea valorilor parametrilor formali cu acelea ale parametrilor actuali corespunzători. În mod similar funcția. Dar după executarea corpului unei funcții, rezultatul este o singură valoare, aceea din evaluarea expresiei din RETURN <expresie>. Această valoare va fi atribuită pentru <numef>(a_1,a_2, …a_n). În mod implicit, la apelul unei funcții se transmit valorile parametrilor actuali a_i către parametrii formali f_i, această situație se numeşte apel prin valoare, deoarece valoarea lui a_i este mutată ca valoare pentru f_i. În cazul apelurilor de proceduri, avem transmitere de valori atît din programul apelant spre procedură, cît şi din procedură către programul apelant. În mod implicit la apelul unei proceduri se transmit procedurii adresele parametrilor actuali, astfel încît se va identifica adresa lui f_i cu adresa lui a_i, procedura lucrînd astfel cu valorile parametrilor actuali, dar fără mutarea acestor valori din programul apelant către procedură. Un astfel de apel se numeşte apel prin referință (sau adresă).

Dacă a_i este o constantă sau expresie, atunci în apel se utilizează a_i drept valoare a parametrului respectiv f_i. Deci în acest caz are loc utilizarea de date de către procedură (dată furnizată de programul apelant). Dacă a_i este o variabilă atunci valoarea ei este transmisă programului apelant (procedurii) şi procedura poate modifica valoarea lui a_i, dar la revenire în programul apelant, a_i va avea valoarea modificată de procedură, deoarece procedura lucrează cu adresa variabilei a_i.

Evident că numărul parametrilor actuali trebuie să fie acelaşi cu numărul parametrilor formali şi a_i trebuie să aibă tipul acelaşi cu f_i.

Parametri actuali de apel al unor subprograme de tip procedură se pot transmite prin valoare dacă ei se includ între paranteze rotunde. Există comanda `SET UDFPARMS` care are următoarele forme:

```
1.SET UDFPARMS TO REFERENCE
2.SET UFDPARMS TO VALUE
```

Prima formă înseamnă transmiterea valorilor prin referință la apelul procedurilor, cît şi la apelul funcțiilor. A doua formă înseamnă transmiterea valorilor parametrilor actuali prin valoare.

Un tablou transmis prin valoare înseamnă transmiterea numai a primului element, pe cînd în cazul transmiterii prin referință, procedura sau funcția va putea lucra cu întregul tablou.

Există funcția `PARAMETERS()` ce returnează numărul de parametri actuali folosiți în cel mai recent apel realizat. Funcția este utilă atunci cînd o procedură sau funcție foloseşte un număr variabil de parametri formali. Dacă un program sursă conține un număr de proceduri, atunci programul se numeşte fişier de proceduri.

Un fişier de proceduri cu numele `<fp>` este activ dacă se dă comanda `SET PROCEDURE TO <fp>`. La un moment dat, există cel mult un fişier de proceduri activ. Închiderea unui fişier de proceduri care este activ se realizează prin `SET PROCEDURE TO`.

Un program (inclus într-un fişier program) are forma generală:

```
     C1
     C2
     .
     .
     Ck
PROCEDURE P1
     D1
RETURN
PROCEDURE P2
     D2
RETURN
     .
     .
PROCEDURE Ph

     Dh
RETURN
FUNCTION F1
```

```
      E1
RETURN <expr>
      .
      .
FUNCTION Fk
      Ek
RETURN <expr>
```

unde C_1, C_2,…, C_k sînt comenzi diferite de PROCEDURE sau FUNCTION. D_i sînt domeniile procedurilor, E_j – sînt domeniile funcţiilor. Funcţiile şi procedurile pot fi dispuse într-o ordine oarecare după comanda C_k.

Dacă F este un fişier program, atunci apelul acestuia se realizează prin DO F. Dacă P este o procedură fără parametri atunci apelul acesteia se realizează sub forma DO P. Rezultă că în cazul unui apel sub forma DO <nume> , situat într-un fişier program F, căutarea se realizează în următoarea ordine:

1) Se caută <nume> în fişierul curent F ca nume de procedură. Dacă există, îl compilează şi apoi execută saltul. Dacă nu există, se trece la 2.
2) Dacă există un fişier de proceduri FB activ, atunci se caută <nume> în FB. Dacă există, se face saltul la acesta. Altfel se trece la 3.
3) Fie fişierele program deschise succesiv prin apeluri, începînd din fereastra de comandă cu DO F_1:
   F_1, F_2, …F_m
   Deci în fereastra de comandă se va da DO F_1, iar în fişierul program F_i există comanda DO F_{i+1}, $1 \le i < m$. Atunci <nume> se caută pe rînd în F_m, F_{m-1},…, F_1. Dacă există, se apelează, realizîndu-se saltul, în caz contrar se trece la 4.
4) Se caută un fişier-program avînd numele = <nume>. Dacă există se compilează şi se realizează saltul la prima comandă din el. Altfel, apare o eroare.

Comanda de apel DO are forma generală:

DO <nume> [WITH a_1, a_2, …, a_h] [IN <fişier>]

Un fişier program poate conţine la rîndul său alte comenzi DO, nivelul admis de imbricare fiind 32. Executarea comenzilor unui fişier program contină pînă la întîlnirea unei comenzi RETURN, CANCEL, QUIT, DO sau pînă la sfîrşitul textului sursă.

O comandă RETURN restituie controlul programului apelant, CANCEL anulează execuţia curentă şi redă controlul SGBD-ului, QUIT trimite controlul sistemului de operare.

Utilizarea componentei IN <fişier> implică considerarea căutării în fişierul program cu numele <fişier>. Dacă <fişier> nu are extensie, atunci FOXPRO va căuta un fişier cu acest nume, în ordinea:

- extensia EXE – program executabil
- extensia APP – program aplicaţie
- extensia FXP – program versiune compilată
- extensia PRG – program sursă.

FOXPRO conţine un compilator, care traduce programul sursă într-un cod obiect, memorat într-un fişier cu extensia FXP. În momentul apelului, FOXPRO caută codul compilat şi dacă acesta există, el este apelat. Dacă nu, caută fişierul sursă, pe care îl compilează; codul obiect are acelaşi nume ca textul sursă. După compilare se apelează. Cînd se modifică programul sursă cu editorul FOXPRO, codul obiect vechi este distrus; la următoarea execuție se recompilează noul program. Dacă se utilizează un alt editor, codul obiect existent nu este distrus, în momentul execuției programului se utilizează vechea versiune a codului obiect. Dacă se utilizează comanda

```
SET DEVELOPMENT ON
```

atunci vechea versiune a codului obiect este ştearsă şi noul text al programului este recompilat. Dacă `SET DEVELOPMENT OFF`, atunci se utilizează vechea versiune a codului obiect.

Există comanda `CLEAR PROGRAM` ce şterge din buffer programele compilate.

Să considerăm un exemplu de program ce conține o procedură. Fie fişierul PERSONAL, utilizat în exemplele anterioare. Dorim să calculăm pentru fiecare salariu impozitul respectiv. Să presupunem că grila de impozitare este următoarea:

De la	Pînă la	Impozit fix	Procent din ce depăşeşte "De la"
0	500000	0	1.00%
500001	1000000	5000	2.00%
1000001	1500000	20000	3.00%
1500001	2000000	40000	3.50%
2000001	3000000	50000	4.00%
Peste 3000000		100000	4.50%

Dacă SAL este o valoare ce se impozitează atunci să notăm cu IMPOZIT, variabila ce va conține impozitul. Să notăm procedura cu CALCULIMP, ce va avea ca parametri SAL şi IMPOZIT. Să citim secvențial fişierul PERSONAL şi pentru fiecare valoare a cîmpului SALARIU vom apela procedura CALCULIMP, ce calculează impozitul. Vom afişa MARCA, NUME, PRENUME, SALARIU, IMP (impozitul) şi REST = SALARIU – IMP.

```
CLEAR
SET TALK OFF
USE PERSONAL
GO TOP
DO WHILE.NOT.EOF( )
     IMP = 0
     DO CALCULIMP WITH SALARIU, IMP
?          MARCA, NUME, PRENUME, SALARIU,;
           IMP, SALARIU - IMP
     SKIP
ENDDO
USE
RETURN
PROCEDURE CALCULIMP
PARAMETERS SAL, IMPOZIT
IF SAL <= 500000
     IMPOZIT = SAL/100
ENDIF
IF (SAL > 500000).AND.SAL <= 1000000)
     IMPOZIT = 5000 + (SAL -; 500000)*2/100
ENDIF
```

```
IF (SAL > 1000000).AND.(SAL <= 1500000)
     IMPOZIT = 20000 + (SAL –; 1000000)*3/100
ENDIF
IF (SAL > 1500000).AND.(SAL<= 2000000)
     IMPOZIT = 40000 + (SAL –; 1500000)*3.5/100
ENDIF
IF( SAL> 2000000).AND.(SAL <= 3000000)
     IMPOZIT = 50000 + (SAL - 2000000)*; 4/100
ENDIF
IF (SAL > 3000000)
     IMPOZIT   =   100000   +   (SAL   –   3000000)*;
4.5/100
ENDIF
RETURN
*SFIRSIT PROGRAM
```

Dacă includem grila de impozitare într-un tablou TAB(N, 4), unde N este numărul de linii, atunci procedura se poate simplifica: (Fie N = 30)

```
IF SAL > TAB( 30, 4)
IMPOZIT = TAB( 30, 3) + ;
     (SAL - TAB( 30, 1)) * TAB(30,4)/100
ENDIF
FOR I = 1 TO 29
     IF   (SAL   >   =   TAB(I,1)).AND.(SAL   <=;
TAB(I,2))
          IMPOZIT = TAB( I, 3) + (SAL –; TAB(
I, 1)) * TAB( I, 4)/100
     ENDIF
ENDFOR
```

CAPITOLUL VII
COMENZI DE POZITIONARE SI CAUTARE

Poziţionarea într-un fişier se realizează prin comenzile `GO` şi `SKIP`. Comanda `GO` are formele:

$$\texttt{1) GO <expN1>}\ \left[\texttt{IN}\left\{\begin{array}{l}\texttt{< expN2 >}\\ \texttt{< expC >}\end{array}\right\}\right]$$

```
2) GO TOP
3) GO BOTTOM
```

În cazurile 2) şi 3) poate fi acelaşi IN ca la 1).

Comanda 1) realizează poziţionarea fişierului pe înregistrarea de număr `<expN1>`, dacă există.

Comanda 2) realizează poziţionarea pe prima înregistrare, iar 3) poziţionează fişierul pe ultima înregistrare.

În toate cazurile, înregistrarea pe care se poziţionează fişierul devine curentă. Atunci cînd se utilizează componenta `IN <expN2>` poziţionarea se realizează pe fişierul activ din zona de număr `<expN2>`, iar cînd se utilizează `<expC>`, atunci poziţionarea se realizează în fişierul activ cu aliasul `<expC>`.

În legătură cu aceste comenzi, există funcţiile `RECNO()` ce are ca valoare numărul înregistrării curente, `RECCOUNT()` ce are ca valoare numărul total de înregistrări din fişierul activ, `EOF()` ce are valoarea .T. dacă fişierul este poziţionat pe EOF şi .F. dacă fişierul este poziţionat pe o anumită înregistrare din fişier, `BOF()` ce are valoarea .T. dacă pointerul de poziţionare se află înaintea primei înregistrări, .F. dacă pointerul este plasat pe o înregistrare.

Dacă dorim consultarea secvenţială a fişierului PERSONAL, afişînd toate înregistrările, vom da:

```
USE PERSONAL
GO TOP
DO WHILE.NOT.EOF()
     DISPLAY
     SKIP
ENDDO
```

Comanda SKIP realizează deplasarea în cadrul fişierului activ. Forma generală:

$$\texttt{SKIP [<expN1>]}\ \left[\texttt{IN}\left\{\begin{array}{l}\texttt{< expN2 >}\\ \texttt{< expC >}\end{array}\right\}\right]$$

Comanda `SKIP` fără componente realizează deplasarea pe articolul următor. `SKIP <expN1>` realizează deplasarea cu valoarea `<expN1>`. Dacă aceasta este pozitivă, deplasarea se face în jos, dacă este negativă, în sus, înţelegînd sus- spre primul articol.

Dacă dorim să realizăm deplasarea în fişierul activ situat în altă zonă (de număr `<expN2>`) vom da `SKIP <expN1> IN <expN2>`. Dacă fişierul activ din altă zonă are aliasul `<expC>`, atunci vom da:

`SKIP <expN1> IN <expC>`.

Căutarea secvențială într-un fişier se realizează cu comanda LOCATE. Căutările multiple le vom realiza cu LOCATE şi CONTINUE. Forma generală pentru LOCATE:

LOCATE [<domeniu>] FOR <condiție$_1$> [WHILE <condiție$_2$>]

Componentele <domeniu>, FOR, WHILE au aceeaşi formă ca în cazul comenzii DELETE.

Dacă F este fişierul de înregistrări F =(I_1, I_2, …,I_n), atunci se calculează F_1, F_2, F_3, exact ca în cazul comenzii DELETE. Fie F_3 =($I_{\alpha1}$,.$I_{\alpha2}$,… $I_{\alpha k}$). Atunci LOCATE poziționează fişierul pe $I_{\alpha1}$ (prima din F_3). O comandă CONTINUE îl va poziționa pe $I_{\alpha2}$, etc. Dacă F este poziționat pe $I_{\alpha k}$, atunci executarea unei comenzi CONTINUE va poziționa F pe EOF, deci EOF()=.T. Dacă F_3 este vid, atunci funcția FOUND()=.F., altfel FOUND()=.T. Orice execuție a lui LOCATE ce realizează poziționarea pe o înregistrare va da EOF()=.F.

Exemplu. Să considerăm fişierul PERSONAL de mai sus şi să presupunem că pot exista mai multe persoane cu acelaşi nume.

În cazul în care nu există nici o persoană cu un anumit nume, vom afişa mesajul 'NUME INEXISTENT'. Vom utiliza variabila WNUME şi o vom citi cu SAY/GET.

```
CLEAR
SET TALK OFF
WNUME = SPACE(10)
@1,1 SAY 'DATI NUMELE PERSOANEI:' GET WNUME
READ
USE PERSONAL
LOCATE ALL FOR NUME = WNUME
IF FOUND()
     DO WHILE FOUND()
          ?MARCA, NUME, PRENUME, SALARIU
          CONTINUE
     ENDDO
ELSE
     ?'NUME INEXISTENT:', WNUME
ENDIF
USE
```

CAPITOLUL VIII

MENIURI VERTICALE

Un meniu vertical are următoarea formă:

opțiune$_1$
opțiune$_2$
…
opțiune$_h$

unde opțiune$_i$ se numeşte componentă a meniului vertical. Fie l_1, c_1 coordonatele colțului din stînga sus al meniului şi l_2 ,c_2 coordonatele colțului din dreapta jos al meniului.

Declararea meniului se realizează cu comanda `DEFINE POPUP`, iar declararea fiecărei componente cu comanda `DEFINE BAR`. Vom da comanda `DEFINE POPUP` cu cele mai importante componente ale sale.

```
DEFINE POPUP <nume popup> From l1, c1  [TO l2, c2 ]
[IN [WINDOW] <nume fer>] [IN SCREEN]
[FOOTER <expc1 >] [MARGIN] [MARK <expc2 >]
[MESSAGE <expc3 >] [MOVER] [MULTI]
[ { PROMPT  FIELD  < expresie  >       } ]
[ { PROMPT  FILES[LIKE   < sablon  >]  } ]
[ { PROMPT  STRUCTURE                  } ]
[RELATIVE] [SCROLL] [SHADOW] [TITLE <expc4>]
[ { COLOR SCHEME < expN >       } ]
[ { COLOR C1/D1,C2/D2,...Ch/Dh  } ]
```

Referitor la natura componentelor unui meniu vertical, putem avea una din situațiile:

1) Componentele sînt definite de utilizator prin comenzi `DEFINE BAR` (cîte una pentru fiecare componentă). În acest caz lipseşte componenta `PROMPT` din comanda `DEFINE POPUP`.
2) Componentele sînt valori ale unor cîmpuri ale unei baze de date, sau valori ale unei expresii formate din cîmpuri ale unei baze de date. În acest caz comanda `DEFINE POPUP` are componenta `PROMPT FIELD <expresie>`, în care `<expresie>` poate fi un cîmp sau o expresie formată din cîmpuri ale unei baze de date, fie din zona curentă, fie din altă zonă. În acest caz nu sînt necesare comenzi `DEFINE BAR`.
3) Componentele meniului sînt fişiere dintr-un anumit director situat pe un anumit disc. În acest caz comanda `DEFINE POPUP` conține `PROMPT FILES`…. Dacă lipseşte `LIKE` se vor lua în considerare toate fişierele din directorul curent. Dacă există `LIKE`, atunci în <şablon> se poate specifica unitatea, directorul, precum şi o mulțime de fişiere, utilizînd caracterele şablon: * (pentru orice cuvînt), ? (pentru un caracter). În meniu vor apare drept opțiuni acest nume de fişiere, definite de `LIKE`.
4) Componentele meniului vertical sunt cîmpurile bazei de date active din zona curentă. În acest caz vom da `PROMPT STRUCTURE`.

În situațiile 2), 3), 4), desigur nu trebuie să dăm comenzi de tip DEFINE BAR. (l1 , c1) – reprezintă coordonatele colțului din stînga sus al meniului, (l2 ,c2) ale celui din dreapta jos. Dacă lipsește componenta TO l2, c2, atunci numărul de linii pentru meniu se calculează după numărul componentelor şi este limitat la numărul de linii ale ferestrei active în care se afişează meniul (sau ale ecranului, cînd nu există fereastră activă). Numărul de coloane se calculează după lungimea cea mai mare a componentelor şi desigur este limitată la dimensiunea ferestrei active (respectiv a ecranului). În cazul în care numărul de componente este mai mare decît numărul de linii ale ecranului, atunci componentele vor defila în interiorul meniului cu ajutorul săgeților ↑,↓. Meniul vertical se activează cu:

```
ACTIVATE POPUP <nume popup>
```

Dacă se utilizează IN SCREEN, atunci meniul se activează pe ecran, iar dacă se utilizează IN WINDOW <numefer> el se activează în fereastra cu numele precizat aici (fereastra în prealabil trebuie definită cu DEFINE WINDOW <nume fer>). Utilizarea părții FOOTER <expc1> implică afişarea valorii expresiei de tip caracter <expc1> în partea de jos a meniului vertical. Utilizarea cuvîntului MARGIN implică lăsarea unei margini (spațiu) suplimentar în stînga şi în dreapta fiecărei componente. Clauza MARK specifică prin <expc2> un caracter de marcaj, ce va fi plasat înaintea fiecărei opțiuni a meniului vertical. Caracterul de marcaj implicit este ◊ (small diamond). Pentru a se realiza marcarea, vom utiliza în prealabil SET MARK OFF. Dacă valoarea expresiei <expc2> este formată din mai multe caractere, atunci se consideră primul caracter drept caracter de marcaj.

Clauza MESSAGE precizează prin <expc3> un mesaj, care se va afişa centrat pe linia specificată de SET MESSAGE, în momentul activării meniului vertical. Dacă pentru o opțiune a meniului vertical se asociază un nou meniu vertical, atunci în dreapta opțiunii respective apare o săgeată. Clauza MOVER permite rearanjarea opțiunilor meniului. Dacă opțiunea este prevăzută cu săgeată şi cursorul este poziționat pe ea, atunci ea poate fi mutată cu CTRL/PgUp şi CTRL/PgDn.

Clauza MULTI permite alegerea mai multor opțiuni dintr-un popup. Opțiunea se va selecta cu caracterul de marcaj. Alegerea mai multor opțiuni se realizează ținînd apăsată tasta SHIFT şi atunci cînd bara este pe opțiune se tastează ENTER sau Spațiu. Nu se pot face alegeri multiple într-un popup definit cu PROMPT.

Clauza RELATIVE controlează ordinea opțiunilor din popup. Fără această clauză, opțiunea este plasată în meniu după numărul său (ce apare în DEFINE BAR <n>) şi se rezervă spații pentru opțiunile cu numere lipsă.

Utilizarea clauzei RELATIVE, implică plasarea opțiunilor în popup în ordinea definirii lor (fără legătură cu numerele lor). În acest caz se pot utiliza clauzele AFTER şi BEFORE în comenzile DEFINE BAR, pentru a plasa opțiunile funcție de unele existente.

Clauza SCROLL plasează în dreapta popup-ului o bară de defilare. Aceasta se afişează numai dacă în meniu nu încap toate opțiunile definite. Clauza SHADOW plasează o umbră în spatele popup-ului, dacă SET SHADOW ON. Clauza TITLE specifică prin <expc4> un şir de caractere ce va fi plasat ca titlu şi afişat deasupra meniului. Clauza COLOR setează culorile (pentru un popup numărul implicit al schemei de culoare este 2).

Definirea unei opțiuni (componente) într-un meniu vertical se realizează cu comanda DEFINE BAR, ce are forma:

```
DEFINE BAR < expN1 > OF < numepopup > PROMPT < expC1 >
[{BEFORE < expN2 >
  AFTER < expN2    }] [MARK < expC3 >][MESSAGE < expC4 >]
[SKIP [FOR < expL >]] [{COLOR SCHEME < expN3 >
                        COLOR c1/d1,.../ ,ch/dh}]
```

<expN1> este numărul opțiunii, <expC1> este prompterul opțiunii ce apare în popup, BEFORE <expN2> permite în cazul clauzei RELATIVE din DEFINE POPUP de a plasa această componentă înainte de componenta cu numărul <expN2>, iar AFTER <expN2> plasarea după. MARK este similară cu MARK din DEFINE POPUP, dar aceasta de aici are prioritate asupra celei din DEFINE POPUP. Dacă se dă SKIP sau SKIP FOR <expL>, unde <expL> are valoarea .T., atunci opțiunea este dezactivată (nu va putea fi selectată). Dacă se dă SKIP FOR <expL> şi <expL> este .F., atunci opțiunea este activată.

Dezactivarea se realizează şi prin plasarea caracterului “\” ca prim caracter în <expC1>. Clauzele MESSAGE şi COLOR au acelaşi rol ca şi la DEFINE POPUP.

Activarea meniului vertical se realizează cu comanda

```
ACTIVATE POPUP <nume popup>
```

iar dezactivarea meniului activ se face cu comanda

```
DEACTIVATE POPUP
```

Putem asocia o procedură unui meniu vertical, rutină ce va fi lansată odată cu activarea popup-lui.

Comanda de asociere:

```
ON SELECTION POPUP {< numepopup >
                    ALL          } [<comandă>]
```

<comandă> este de obicei de forma DO <nume procedură>. Folosirea lui ALL înseamnă că la activarea tuturor meniurilor verticale se va executa <comandă>. Această <comandă> poate fi şi o comandă simplă.

Deci <comandă> se va executa pentru orice selectare a oricărei opțiuni a meniului vertical activ. La un moment dat există un singur meniu vertical activ. Dacă se dă ON SELECTION POPUP fără <comanda> atunci se dezasociază comanda veche de meniul (sau meniurile) respectiv.

Putem asocia o comandă pentru fiecare opțiune a unui meniu vertical, utilizînd ON SELECTION BAR:

ON SELECTION BAR <expN> OF <nume popup> [<comanda>],

unde <expN> are ca valoare numărul opțiunii.

Cînd se selectează opțiunea respectivă (plasînd bara luminoasă pe ea şi tastînd ENTER) atunci se va executa <comanda>, dacă aceasta apare în ON SELECTION BAR. Omiterea părții <comanda> implică neasocierea acestei opțiuni a meniului cu nici o comandă. Desigur, <comanda> poate avea forma DO <nume procedură>

sau poate fi o comandă sinplă. Dacă unei astfel de linii (opțiuni) din meniul vertical i se ataşează un alt meniu vertical, atunci <comanda> poate fi:

ACTIVATE POPUP <nume popup$_1$>

Comanda ON SELECTION BAR trebuie să fie plasată în textul sursă între DEFINE POPUP şi ACTIVATE POPUP.

CAPITOLUL IX
MENIURI ORIZONTALE

Definirea unui meniu orizontal se realizează cu comanda `DEFINE MENU` ce are forma generală:

```
DEFINE MENU <numemeniu> [BAR [AT LINE <expN1>]]
[{WINDOW < numefer >}] [MARK < expC1 >]
[{IN SCREEN         }]
[MESSAGE <expc2>] [NOMARGIN]
      [{COLOR SCHEME < expN2 >}]
      [{COLOR c1/d1,...,ch/dh }]
```

Un meniu orizontal are forma unei succesiuni de elemente (numite pad-uri) dispuse, de obicei, orizontal.

Pad-urile se definesc cu comanda `DEFINE PAD`. Fiecărui pad i se asociază o acțiune printr-o comandă de forma `ON PAD` sau `ON SELECTION PAD`. `<numemeniu>` este numele atribuit de utilizator acestui meniu orizontal. Dacă se utilizează clauza `BAR`, atunci meniul respectiv se comportă ca meniul linie al sistemului FOX. Dacă numărul elementelor este mai mare decît dimensiunea ecranului, atunci acestea defilează cu ajutorul săgeților orizontale. Dacă se precizează `AT LINE <expN1>`, atunci linia meniului se afişează pe linia de număr `<expN1>` de pe ecran.

Clauza `NOMARGIN` şterge spațiile din stînga şi din dreapta elementelor. Celelalte clauze acționează ca şi în cazul comenzii `DEFINE POPUP`.

Crearea unui element al meniului orizontal se realizează cu comanda `DEFINE PAD` ce are forma:

```
DEFINE PAD <numepad> OF <numemeniu>
PROMPT <expc1> [AT l, c]
[{BEFORE < numepad >}] [MARK < expc3 >]
[{AFTER < numepad > }]
[SKIP [FOR<expL>]] [MESSAGE <expC4>]
[{COLOR SCHEME < expN >}]
[{COLOR c1/d1,...,ch/dh}]
```

Componenta `AT l, c` permite precizarea coordonatelor de început ale pad-ului respectiv. Dacă în `DEFINE MENU` s-a dat `BAR`, atunci clauza `AT` nu mai este necesară. Celelalte clauze au semnificația de la `DEFINE BAR`. La alegerea unui element (prin plasarea barei luminoase pe el şi tastarea lui ENTER) se poate activa un meniu vertical, un alt meniu orizontal sau se poate executa o rutină (program sau procedură). Activarea meniului se realizează prin comanda : `ACTIVATE MENU` ce are forma generală:

```
ACTIVATE MENU <numemeniu> [NOWAIT][PAD <numepad>]
```

Folosirea clauzei NOWAIT implică continuarea execuției programului (cu linia următoare lui ACTIVATE MENU) după activarea meniului respectiv. Dacă se utilizează PAD <numepad> atunci la activarea meniului, bara luminoasă este plasată pe această opțiune, altfel ea se plasează pe primul pad al meniului. Dezactivarea meniului activ se realizează cu comanda DEACTIVATE MENU.

Asignarea unei acțiuni pentru un meniu orizontal se realizează cu comanda ON SELECTION MENU, de forma:

$$\texttt{ON SELECTION MENU} \left\{\begin{array}{l}\texttt{< numemeniu >} \\ \texttt{ALL}\end{array}\right\} [\texttt{< comanda >}]$$

Dacă se dă <numemeniu> <comanda>, atunci această <comanda> va fi executată la activarea meniului respectiv, indiferent de elementul selectat al meniului. Dacă se dă ALL <comanda>, atunci <comanda> se va executa la activarea tuturor meniurilor, <comanda> poate fi DO <numeprogram>, DO <nume procedură> sau o comandă diferită de acestea. Absența clauzei <comanda> determină neatribuirea pentru acest meniu a niciunei acțiuni. Pentru un pad se poate atribui o procedură sau program cu comanda ON SELECTION PAD, sau un alt meniu orizontal sau vertical cu comanda ON PAD.

ON SELECTION PAD <numepad> OF <nume meniu> [<comandă>]

Se asignează pentru <numepad> o acțiune definită de <comandă>, care de obicei este DO <nume program>, sau DO <nume procedură>. ON SELECTION PAD trebuie plasată între DEFINE MENU şi ACTIVATE MENU. Dacă se dă fără <comandă>, atunci pad-ului respectiv nu i se asociază nici o acțiune. Comanda ON PAD are forma:

$$\texttt{ON PAD <numepad> OF <numemeniu>} \left[\left\{\begin{array}{l}\texttt{ACTIVATE POPUP < mv >} \\ \texttt{ACTIVATE MENU < mo >}\end{array}\right\}\right]$$

<mv> este numele unui meniu vertical iar <mo> este numele unui meniu orizontal.

Dacă se dă ACTIVATE POPUP <mv> atunci la selectarea pad-ului respectiv se activează meniul vertical <mv>. Pentru ACTIVATE MENU <mo> se va activa meniul orizontal <mo>. Dacă lipseşte această clauză, atunci la activarea pad-ului respectiv nu are loc nici o acțiune. Referitoare la meniuri există funcțiile:

a) BAR() – returnează numărul componentei dintr-un meniu vertical, selectate ultima dată.
b) PROMPT() - returnează numele (promptul) unei componente a unui meniu vertical sau orizontal.
c) POPUP() - returnează numele popup-ului activ sub forma unui şir de caractere, cu litere mari. Dacă nu există niciun popup activ se returnează şirul nul.
d) MENU() - returnează numele meniului orizontal activ sub forma unui şir de caractere, cu litere mari. Dacă nu există meniu orizontal activ, se returnează şirul nul.
e) PRMBAR()– ce are forma PRMBAR(<expC>, <expN>) returnează numele unei componente a unui meniu vertical cu numele <expC>, componentă ce are numărul <expN>. Meniul vertical nu trebuie să fie activ. Dacă programul nu conține specificarea unei taste de apel (\<) sau caracterul \ de dezactivare, atunci funcția returnează textul parametrului fără aceste caractere.

f) `PRMPAD(<expC1>, <expC2>)`, unde `<expC1>` are ca valoare numele meniului orizontal iar `<expC2>` are ca valoare numele unui pad al meniului orizontal respectiv. Funcția returnează textul specificat prin clauza `PROMPT` a comenzii `DEFINE PAD` respective. Meniul nu trebuie să fie activ.

g) `MRKBAR(<expC>, <expN>)` unde `<expC>` dă numele unui popup, iar `<expN>` dă numărul unei linii din popup. Funcția returnează .T. dacă linia respectivă a fost marcată.

h) `MRKPAD(<expC1>, <expC2>)` unde `<expC1>` conține numele unui meniu orizontal iar `<expC2>` numele unui pad din acest meniu. Funcția returnează .T. dacă padul respectiv este marcat.

i) `SKPBAR(<expC>, <expN>)`, unde `<expC>` dă numele unui popup, iar `<expN>` numărul unei componente din el. Funcția returnează .T. dacă componenta respectivă a fost dezactivată cu comanda `SET SKIP OF BAR <expN> OF <expC>`. Dacă opțiunea este activă funcția returnează .F.

j) `SKPPAD(<expC1>, <expC2>)` unde `<expC1>` specifică numele unui meniu orizontal, iar `<expC2>` specifică numele unui pad al acestuia. Funcția returnează .T., dacă pad-ul respectiv a fost dezactivat cu comanda `SET SKIP OF PAD <numepad> OF <numemeniu>`.

CAPITOLUL X
INDEXAREA BAZELOR DE DATE

Structura indecşilor. Fie F un fişier avînd înregistrările (I_1, I_2, …, I_n). Fie structura sa FS = (cîmp$_1$, cîmp$_2$, … , cîmp$_h$). Pentru construcţia unui index se consideră o expresie de indexare pe care o notăm cu E. Această expresie poate conţine cîmpurile lui F, variabile, constante, dar poate conţine şi cîmpuri ale altor fişiere active. În particular E poate fi formată dintr-un singur cîmp din FS. Fie înregistrarea $I_j = (v_{j1}, v_{j2}, \dots, v_{jh})$, unde v_{ji} este valoarea lui cîmp$_i$ pentru înregistrarea I_j. Să notăm prin $E(I_j)$ valoarea obţinută prin înlocuirea cîmpului cîmp$_i$ din expresia E cu v_{ji} , deci evaluarea lui E pentru înregistrarea I_j. Atunci cînd componentele expresiei E sînt de tipuri diferite, trebuie utilizate funcţii de conversie. În privinţa valorilor expresiei E pentru înregistrările I_j putem avea situaţiile:

a) pentru orice $I_j \neq I_e$ avem $E(I_j) \neq E(I_e)$.
b) există $I_e \neq I_e$ astfel încît $E(I_j) = E(I_e)$.

Considerăm în continuare situaţia a).

Indecşii pot fi creaţi considerînd valorile $E(I_j)$ ordonate crescător sau descrescător. Ne vom ocupa de cazul crescător. La construcţie se ordonează aceste valori şi fie şirul ordonat rezultat:

$$E(I_{\alpha 1}) < E(I_{\alpha 2}) < \dots < E(I_{\alpha n})$$

unde $I_{\alpha 1}, I_{\alpha 2}, \dots, I_{\alpha n}$ sînt toate înregistrările lui F, eventual într-o altă ordine, deci ($\alpha 1$, $\alpha 2$,…, αn) este o permutare pentru (1, 2,… , n). În situaţia a) se spune că E constituie cheie unică pentru F. Indexul construit pentru F şi expresia E, notat INDEX(F,E) are următoarele componente:

INDEX(F, E) = $(\alpha 1, E(I_{\alpha 1}))\ (\alpha 2, E(I_{\alpha 2}))\ \dots\ (\alpha n, E(I_{\alpha n}))$, αi este numărul înregistrării $I_{\alpha i}$.

Exemplu: fie PRODUSE un fişier cu structura CODP (codul unui produs), DENP (denumirea produsului), CANTP(cantitatea), PRETP (preţul unitar al produsului). Fie următorul conţinut al fişierului:

	CODP	DENP	CANTP	PRETP
I_1	100	P1	100	20.50
I_2	150	P2	200	50.75
I_3	175	P3	300	75.00
I_4	125	P4	150	90.00
I_5	120	P6	250	60.00
I_6	160	P6	125	27.00

Considerăm drept expresie E de indexare: E=CODP.

Atunci valorile expresiei E pentru înregistrările I_1, … ,I_6 vor fi :

$$\begin{matrix} (100, & 150, & 175, & 125, & 120, & 160) \\ I_1 & I_2 & I_3 & I_4 & I_5 & I_6 \end{matrix}$$

Ordonarea lor crescătoare va fi:

$$\begin{matrix} (100, & 120, & 125, & 150, & 160, & 175) \\ I_1 & I_5 & I_4 & I_2 & I_6 & I_3 \end{matrix}$$

Deci INDEX (F, CODP) = (1,100) (5,120) (4,125) (2,150) (6,160) (3,175)

Căutarea utilizînd indexul construit înseamnă specificarea de către utilizator a unei valori V_0, numită cheie de căutare. Apare întrebarea: există în F o înregistrare I_j, astfel încît $V_0 = E(I_j)$? Vom vedea că pentru căutare avem comenzile `SEEK` şi `FIND`, iar rezultatul căutării este dat de funcția `FOUND()` ce are valoarea .T. dacă există I_j, .F. altfel. În exemplul anterior, dacă $V_0 = 120$, se caută în index şi se obține perechea (5,120), deci înregistrarea de număr 5 este cea căutată, fişierul F poziționîndu-se pe I_5 (ce devine curentă) şi funcția `FOUND() = .T.`

Dacă ne interesează un produs cu codul 140, atunci $V_0 = 140$ şi căutarea în INDEX(F, CODP) este cu insucces, deci funcția `FOUND() =.F.`

Considerăm acum situația b), deci pot exista înregistrări diferite $I_j \neq I_e$, astfel încît $E(I_j) = E(I_e)$. În acest caz, construcția indexului se poate face în 2 moduri:

b1) se includ în index toate valorile expresiei de indexare,

b2) se includ în index numai valorile distincte ale expresiei de indexare.

Cazul b1) apare atunci cînd în comanda `INDEX` nu există cuvîntul rezervat `UNIQUE`, sau în program există `SET UNIQUE OFF`. Cazul b2) apare aunci cînd în comada `INDEX` este dat cuvîntul `UNIQUE` sau în program avem `SET UNIQUE ON`.

În situația b1) considerînd ordonarea crescătoare a valorilor expresiei E pentru înregistrările I_1, I_2, .. I_n, atunci fie aceste valori în ordine crescătoare:

$E(I_{\alpha 1}) \leq E(I_{\alpha 2}) \leq \ldots \leq E(I_{\alpha n})$

cu (α1, α2,…, αn) o permutare a lui (1, 2,…, n). Indexul are aceeaşi formă ca şi în cazul a) cu deosebirea că pot exista valori egale ale componentelor de pe locul doi.

$INDEX(F,E) = (\alpha 1, E(I_{\alpha 1}))\ (\alpha 2, E(I_{\alpha 2}))\ \ldots\ (\alpha n, E(I_{\alpha n}))$

Exemplu: fie fişierul PERSONAL cu structura:

	MARCA	NUME	PRENUME	SALARIU
I_1	100	ALEXA	ANA	1000000
I_2	200	ALEXA	ION	1500000
I_3	250	IONESCU	VASILE	1250000
I_4	270	IONESCU	CECILIA	1600000
I_5	290	POPESCU	DAN	1650000
I_6	300	POPESCU	ELENA	1700000

Dacă dorim să realizăm căutarea în acces direct în fişierul PERSONAL după NUME, atunci expresia de indexare va fi E = NUME. În acest caz valorile lui E sînt:

ALEXA,	ALEXA,	IONESCU,	IONESCU,	POPESCU,	POPESCU
I_1	I_2	I_3	I_4	I_5	I_6

Ele sînt chiar ordonate crescător. În situația b1) indexul va avea conținutul:

INDEX(F, NUME) = (1, ALEXA) (2, ALEXA) (3, IONESCU) (4, IONESCU) (5, POPESCU) (6, POPESCU)

Căutarea unei înregistrări cu un anumit nume va furniza o singură înregistrare cu acel nume, dacă există. De exemplu, fie V_0 = 'ALEXA'. Algoritmul de căutare ne va da înregistrarea cu nr. 1. Orice căutare ulterioară cu V_0 = 'ALEXA' va da acelaşi rezultat. În situația b2) (cînd se includ în index numai valorile distincte ale expresiei de indexare) se ordonează crescător valorile expresiei E pentru I_1, I_2, … I_n, fie acestea:

$E(I_{\alpha 1}) \leq E(I_{\alpha 2}) \leq \ldots \leq E(I_{\alpha n})$

Din acest şir se aleg numai cele distincte, fie:

$E(I_{\beta 1}) < E(I_{\beta 2}) < \ldots < E(I_{\beta t})$

$I_{\beta j}$ are numărul de înregistrare βj. În acest caz indexul are forma:

$$INDEX(F,E) = (\beta 1, E(I_{\beta 1}))(\beta_2, E(I_{\beta 2}))\ldots(\beta t, E(I_{\beta t})).$$

Fie V_0 – valoarea de căutare. Algoritmul de căutare va cerceta întîi indexul INDEX(F, E). Dacă există i, $1 \le i \le t$ astfel încît $V_0 = I_{\beta i}$, atunci rezultatul este înregistrarea $I_{\beta i}$, ce devine curentă, funcţia `FOUND()=.T.`, `EOF() = .F.` În F pot exista şi alte înregistrări cu acelaşi V_0. În prealabil, dacă se ordonează F după expresia E, atunci după căutarea directă cu V_0 se revine la căutarea secvenţială (prin `SET ORDER TO`) şi se citesc celelalte înregistrări cu acelaşi V_0 (secvenţial).

Construcţia indecşilor. Un index se poate construi pornind de la definirea structurii sau utilizînd comanda `INDEX`. Discutăm întîi comanda `INDEX`. Forma generală:

$$\texttt{INDEX ON <expresie>}\left\{\begin{array}{l}\texttt{TO < fisier index >}\\ \texttt{TAG < numetag > [OF < fisier.CDX >]}\end{array}\right\}$$

$$\texttt{[FOR <expresie logică>][COMPACT]}\left[\left\{\begin{array}{l}\texttt{ASCENDING}\\ \texttt{DESCENDING}\end{array}\right\}\right]\texttt{[UNIQUE]}$$

`[ADDITIVE]`

Un index creat de către `INDEX` poate fi memorat singur într-un fişier al sistemului de operare `<fişier index>` cu extensia IDX, sau poate fi inclus împreună cu mai mulţi indecşi într-un acelaşi fişier al sistemului de operare numit `<fişier.CDX>` avînd extensia CDX. În ultimul caz, indexul se numeşte tag sau reper, numele este `<numetag>`. Fişierul ce conţine aceste repere (tag-uri) se numeşte fişier multiindex. Dacă dorim să creăm un index în mod descrescător, atunci obligatoriu acesta trebuie creat ca tag. Componenta `<expresie>` defineşte expresia de indexare, notată cu E. În general E se construieşte din cîmpuri ale bazei de date active, constante, variabile de memorie. Dacă în E există elemente de tip diferit, atunci se utilizează funcţiile de conversie. Componenta `TO <fişier index>` specifică fişierul cu extensia IDX al sistemului de operare, ce va memora indexul construit, conform expresiei E. `TAG <numetag>` precizează numele tag-ului în care se va memora indexul. `OF <fişier.CDX>` precizează fişierul multiindex (cu extensia CDX), ce va conţine tagul `<numetag>`. Există două tipuri de fişiere multiindex: de exploatare (sau al structurii) şi al utilizatorului. Fişierul multiindex de exploatare se defineşte odată cu crearea structurii fişierului bază de date.

Dacă pentru $cîmp_i$ se doreşte construirea unui index, atunci pe linia de definire a atributului $cîmp_i$ se aduce cursorul în faţa numelui cîmpului şi tastînd spaţii se vor vizualiza pe rînd caracterele ↑, ↓ şi spaţiu. În prima situaţie, se va crea un index crescător cu valorile acestui cîmp (la definirea înregistrărilor), în a doua situaţie se va crea un index descrescător cu valorile acestui cîmp, iar în a treia situaţie nu se creează index pentru $cîmp_i$. Toţi indecşii creaţi astfel vor fi memoraţi sub formă de tag-uri (cu acelaşi nume cu cîmpul respectiv – $nume_i$) într-un fişier multiindex cu numele F.CDX, unde F este numele fişierului de date. Dacă pentru nici un cîmp nu se defineşte crearea de index, atunci F.CDX nu se creează. Dacă nu se precizează componenta OF <fişier.CDX>, atunci tag-ul respectiv va fi introdus în F.CDX.

Cînd se precizează componenta `TAG <nume reper> OF <fişier.CDX>` şi `<fişier.CDX>` nu există, atunci el se creează. Odată cu activarea fişierului F.DBF se activează şi fişierul F.CDX şi deci şi reperele din el.

Fişierele de index sau multiindex ale utilizatorului trebuie activate cu comenzi de forma: USE, SET INDEX, SET ORDER.

Dacă dorim să construim un index numai pentru o parte din înregistrările fişierului vom da componenta FOR <expresie logică>, indexul se va construi pentru înregistrările ce fac această expresie .T. Cuvîntul COMPACT implică memorarea indecşilor într-o formă compactă. ASCENDING, DESCENDING – precizează modul în care se ordonează valorile expresiei de indexare. UNIQUE determină includerea în index numai a perechilor (j, v_j) pentru toate valorile distincte ale expresiei de indexare. În absența cuvîntului ADDITIVE, toți indecşii activi, cu excepția celor din F.CDX, devin inactivi. Utilizînd ADDITIVE indecşii care au fost activi rămîn activi.

Stările indecşilor. **Comenzile USE, SET INDEX, SET ORDER.** Fie F fişierul bază de date activ în zona curentă. Pentru F se pot defini atît indecşi incluşi în fişierul multiindex de exploatare (odată cu definirea structurii), cît şi indecşi construiți cu comanda INDEX. Fie IN(F, E) un index construit pentru F, cu expresia de indexare E. Indexul IN(F, E) se numeşte activ la un moment dat pentru F, dacă orice actualizare a lui F (adăugări, ştergeri, modificări) conduce la actualizarea indexului, în sensul să reflecte noul conținut al lui F. Indexul IN(F, E) se numeşte principal (master) la un moment dat, dacă el este activ în acel moment şi dacă el este utilizat pentru accesul la F (prin comenzile SEEK, FIND sau funcția SEEK).

Dacă V_0 este valoarea de căutare, atunci se caută perechea (j, v_j) din index, astfel că $v_j = V_0$. În cazul că există, rezultatul căutării este înregistrarea de număr j. Spunem că indexul după care se face căutarea este index principal în acest moment.

Un index IN(F,E) se numeşte inactiv la un moment dat, dacă el este construit şi orice actualizare a lui F nu antrenează actualizarea indexului. Reperele din F.CDX sînt active pe toată perioada cît F este activ.

Comanda USE activează (sau dezactivează) un fişier bază de date, activează o mulțime de indecşi şi defineşte indexul principal.

Comanda SET INDEX activează o mulțime de indecşi, defineşte eventual un nou index principal.

Comanda SET ORDER defineşte noul index principal.

Sintaxa comenzii USE:

```
USE [{< fisier > | ?}] [IN <zona>][AGAIN][INDEX <lista fisierelor index>]
    [ORDER {< expresie > | < fisierindex > | TAG < numetag > [OF < fisier.CDX >]}]
    [{ASCENDING | DESCENDING}] [ALIAS <numea>][EXCLUSIVE][NOUPDATE
```

În sistemul de gestiune FOXPRO –DOS se poate lucra cu 25 zone pentru fişiere, la un moment dat într-o zonă poate fi activ un singur fişier bază de date. Zonele se numerotează 1, 2, …, 25. Primele 10 zone se identifică prin literele A, B, C,…, J iar zonele 11 – 25 prin W11, … W25. Aceşti identificatori se numesc alias–uri standard. In sistemul de gestiune Visual Foxpro numarul de zone de lucru este de 32767. Prezența componentei ALIAS <numea> implică un alt nume alias pentru fişierul deschis cu acest `USE`. `<fişier>` este numele fişierului bază de date ce va fi activat în acest moment. Dacă se utilizează `USE ?`, atunci sistemul FOXPRO inițiază un dialog prin care se solicită utilizatorului numele fişierului (se afişează un meniu). Componenta `IN <zona>`, serveşte pentru activarea fişierului respectiv în zona de număr prezent în componentă.

Evident `<zona>` aparține mulțimii {1, … , 25}. Comanda `USE` fără componente duce la dezactivarea fişierului ce a fost activ în zona curentă, iar `USE IN <zonă>` dezactivează fişierul activ din `<zonă>`. Dacă lipseşte componenta `IN <zona>`, atunci deschiderea (sau închiderea) se referă la zona curentă. Dacă a existat un fişier activ, atunci acest `USE` îl dezactivează. Componenta `AGAIN` activează un fişier, ce este deja activ în altă zonă de lucru. Fişierele de index construite pentru fişierul bază de date respectiv sînt disponibile şi în noua zonă de lucru. Componenta `INDEX` precizează mulțimea de indecşi care vor deveni activi din acest moment. Lista are forma $f_1, f_2, \dots, f_h$, unde f_i sînt fişierele index sau multiindex (deci au extensiile IDX sau CDX). Deoarece se activează obligatoriu indecşii din F.CDX, atunci acest F.CDX nu trebuie plasat în listă. În absența componentei `ORDER`, dacă f_1 este de tip CDX, atunci spunem că nu există index principal şi accesul la F se realizează în ordine naturală (în ordinea crescătoare a numărului de înregistrare). Dacă lipseşte `ORDER` şi f_1 este de tip IDX, atunci indexul din f_1 devine principal.

Componenta `ORDER` defineşte indexul principal (master), deci indexul folosit în consultarea lui F. `<expresie>` trebuie să aibă valori numerice întregi. Se ordonează indecşii din fişierele f_1, f_2, …f_h astfel: întîi se consideră indecşii din fişierele de tip IDX apoi indecşii din F.CDX (dacă există), apoi indecşii din fişierele de tip CDX din listă, în ordinea lor din listă. Fie numerele obținute 1, 2, …, n. Fie k valoarea pentru `<expresie>`. În situația $k \in \{1, 2, \dots, n\}$, noul index principal va fi cel de număr k. În situația $k = 0$, nu vom avea index principal şi consultarea se va face în ordine naturală. În situația $k < 0$ sau $k > n$, va apare o eroare (INDEX NOT FOUND).

Cînd în `USE` nu se specifică nici componenta `INDEX`, nici `ORDER`, atunci consultarea lui F se realizează în ordine naturală. Folosirea `ORDER 0` sau `ORDER` înseamnă că fişierele de index din `INDEX` sînt active, dar consulterea lui F se realizează în ordinea naturală.

Folosirea `ORDER <fişier index>` înseamnă că indexul plasat în `<fişier index>` va deveni principal. `ORDER TAG <numetag> OF <fişier.CDX>` înseamnă că reperul `<numetag>` din `<fişier.CDX>` va deveni principal. Cînd nu se specifică `OF <fişier.CDX>`, atunci se caută `<numetag>` întîi în F.CDX, apoi în celelalte fişiere multiindex (deci de tip CDX). Folosirea lui `ASCENDING` implică furnizarea înregistrărilor în ordinea crescătoare a expresiei de indexare E, indiferent de modul de construcție al indexului, iar `DESCENDING` înseamnă furnizarea înregistrărilor în ordine descrescătoare. Evident indexul respectiv trebuie să fie principal. Cînd nu se specifică unul din aceste cuvinte, furnizarea înregistrărilor se realizează în ordinea în care au fost create. `EXCLUSIVE` se foloseşte în sistemul FOXPRO pentru rețea şi înseamnă că `<fişier>` va fi folosit numai de programul curent. Acest acces exclusiv durează pînă la închiderea lui F. `NOUPDATE` interzice modificarea structurii bazei de date şi a înregistrărilor sale.

Sintaxa comenzii `SET INDEX`:

$$\texttt{SET INDEX TO}\begin{bmatrix}\{f_1,\ f_2,\ \ldots,\ f_k\}\\ ?\end{bmatrix}$$

$$\left[\texttt{ORDER}\left\{\begin{array}{l}\texttt{< expresie >}\\ \texttt{< fisier index >}\\ \texttt{TAG < nume tag > [of < fisier.CDX.]}\end{array}\right\}\right]$$

$$\left[\left\{\begin{array}{l}\texttt{ASCENDING}\\ \texttt{DESCENDING}\end{array}\right\}\right]\quad\texttt{[ADDITIVE]}$$

Forma `SET INDEX TO` implică închiderea tuturor indecşilor activi asociaţi lui F, cu excepţia celor din F.CDX. f_i , $1 \le i \le k$ sînt fişiere de tip IDX sau CDX. Dacă f_1 este de tip IDX, atunci acesta devine principal, dacă lipseşte componenta `ORDER`. Dacă lipseşte `ORDER` şi f_1 este de tip CDX, atunci nu va exista index principal, deci consultarea lui F se realizează în ordine naturală. Folosirea caracterului `?` implică afişarea unui meniu din care utilizatorul alege un fişier IDX sau CDX. `ORDER` defineşte indexul principal, exact ca în cazul comenzii `USE`. Cuvintele `ASCENDING`, `DESCENDING` au acelaşi rol ca în comanda `USE`. Dacă se utilizează `ADDITIVE`, indecşii ce au fost activi rămîn activi, altfel vor deveni activi numai cei din f_1, f_2, … f_k.

Sintaxa comenzii `SET ORDER`:

$$\texttt{SET ORDER TO}\left[\left\{\begin{array}{l}\texttt{< expresie >}\\ \texttt{< fisier index >}\\ \texttt{TAG < nume tag > [OF < fisier.CDX >]}\end{array}\right\}\right]$$

$$\left[\texttt{IN}\left\{\begin{array}{l}\texttt{< expN >}\\ \texttt{< expC >}\end{array}\right\}\right]\quad\left[\left\{\begin{array}{l}\texttt{ASCENDING}\\ \texttt{DESCENDING}\end{array}\right\}\right]$$

Comanda defineşte indexul principal. Primul argument are aceeaşi semnificaţie ca în cazul comenzii `USE` şi `SET INDEX`. Componenta `IN` defineşte o zonă ce conţine un fişier bază de date F' şi pentru care se va defini noul index principal. `<expN>` este un număr din {1, …, 25} ce reprezintă o zonă de lucru, iar `<expC>` are ca valoare un alias (standard sau definit de `ALIAS`). Absenţa componentei `IN` implică definirea indexului principal pentru fişierul activ în zona curentă.

Căutarea utilizînd indexul o realizăm prin `FIND`, `SEEK` şi funcţia `SEEK()`. Comanda `FIND` are forma:

$$\texttt{FIND}\left\{\begin{array}{l}C_1C_2\texttt{....}C_k\\ \texttt{'}C_1C_2\texttt{.....}C_k\texttt{'}\\ \texttt{\& < variabila >}\end{array}\right\}$$

În primul caz valoarea de căutare V_0 este $C_1 \ldots C_k$, în al doilea caz, de asemenea $V_0=C_1\ldots C_k$. În al treilea caz $V_0=d_1..d_k$, unde $d_1..d_k$ este valoarea pentru `<variabila>`. Această variabilă trebuie să fie de tip caracter. Comanda `SEEK` are forma: `SEEK <expresie>`. În această situație `<expresie>` dă valoarea V_0. Funcția `SEEK` are forma $\texttt{SEEK}\left(< \texttt{expresie} > \left[, \begin{Bmatrix} < \texttt{expN} > \\ < \texttt{expC} > \end{Bmatrix} \right] \right)$ unde `<expresie>` dă valoarea de căutare V_0, iar `<expN>` dă zona în care se găseşte fişierul unde se va face căutarea cu index. `<expC>` dă aliasul fişierului unde se face căutarea. Funcția returnează .T., dacă s-a găsit articolul, astfel .F .

CAPITOLUL XI
RELAȚII ÎNTRE TABELE

Considerăm două tabele: PERSONAL şi VINZARI.
Fie PERSONAL cu structura:

MARCA (N, 5)
NUME (C, 10)
PRENUME (C, 10)
SALARIU (N, 7)

Fie VINZARI cu structura:

MARCAV (N, 5) - marca vînzătorului
PROD (C, 5) - codul produsului vîndut
CANT (N, 5) - cantitatea vîndută
PRET (N, 3) - prețul unitar al produsului vîndut

Presupunem că valorile cîmpului MARCA din PERSONAL identifică persoana respectivă (se spune că MARCA este cheie unică a fişierului). În VÎNZĂRI pot exista mai multe înregistrări cu aceeaşi valoare pentru MARCAV (deci vînzări realizate de aceeaşi persoană).

Să considerăm cîte o instanță pentru fiecare tabelă:

PERSONAL

	MARCA	NUME	PRENUME	SALARIU
I_1	100	ALEXA	ION	1000000
I_2	200	BARBU	VASILE	1200000
I_3	300	COSTEA	ANA	1100000
I_4	400	DARIE	LIVIU	1500000

VINZARI

	MARCAV	PROD	CANT	PREȚ
J_1	100	P1	50	2
J_2	100	P3	25	1
J_3	200	P2	30	3
J_4	400	P1	60	1
J_5	400	P2	80	2
J_6	400	P4	40	1

Presupunem că în orice moment al exploatării celor două tabele avem păstrată consistența datelor, adică pentru orice valoare a cîmpului MARCAV din VINZARI există o înregistrare în PERSONAL cu aceeaşi marcă. Notînd cu MARCAV(J_e) valoarea cîmpului MARCAV pentru înregistrarea J_e şi cu MARCA(I_j) valoarea cîmpului MARCA pentru înregistrarea I_j, atunci consistența de mai sus se exprimă sub forma:

$$(\forall J_e)[J_e \in \text{VÎNZĂRI} \Rightarrow (\exists! Ij \in \text{PERSONAL})((\text{MARCAV}(J_e) = \text{MARCA}(I_j))]$$

unde $\exists!$ notează existența unui unic element.

Dacă lăsăm deoparte această proprietate de consistență, atunci unei înregistrări J_e din VINZARI poate să nu-i corespundă nici o înregistrare I_j din PERSONAL cu proprietatea specificată. Vom exprima acest caz prin formula :

$$(\forall J_e)[J_e \in VINZARI \Rightarrow (\theta I_j)((I_j \in PERSONAL) \wedge (MARCAV(J_e) = MARCA(I_j)))]$$

unde θ înseamnă fie ∃! (există unic) fie ¬∃ (deci nu există).

În acest ultim caz putem stabili o funcție, notată prin:

$$\varphi_{MARCAV,\ MARCA}: VINZARI \rightarrow PERSONAL \cup \{\underline{EOF}\}$$

definită prin:

$\varphi_{MARCAV,MARCA}(J_e) = I_j$ dacă există I_j cu proprietatea $MARCAV(J_e)=MARCA(I_j)$, și $\varphi_{MARCAV,\ MARCA}(J_e) = \underline{EOF}$ dacă nu există I_j ca mai sus. EOF aici este un simbol ce va reprezenta sfîrşitul de fişier din PERSONAL.

Putem stabili o relație între VINZARI şi PERSONAL (care se numeşte în acest caz de tip 1 – 1) prin perechea MARCAV şi MARCA. Tabela PERSONAL trebuie să fie indexată după MARCA. Cîmpul MARCAV se numeşte expresie de relaționare între cele două. Tabela VINZARI se numeşte părinte, iar PERSONAL se numeşte fiu. Realizarea prin program a acestei relații se realizează astfel (fie VINZARI deschis în zona 1 şi PERSONAL în zona 2):

```
USE PERSONAL IN 2
SELECT 2
INDEX ON MARCA TO IMARCA
SELECT 1
USE VINZARI
SET RELATION TO MARCAV INTO 2
```

Rezultatul definirii unei astfel de relații este următorul:

Poziționarea tabelei VINZARI pe înregistrarea J_e, va avea ca rezultat poziționarea tabelei fiu (PERSONAL) pe înregistrarea I_j, cu proprietatea MARCAV(Je) = $MARCA(I_j)$ (cînd există) sau poziționarea lui PERSONAL pe EOF (cînd nu există I_j). Să realizăm un program ce afişează pentru fiecare vînzare din VINZARI, în plus față de cîmpurile respective din VINZARI, şi NUME, PRENUME şi SALARIU.

După definirea relației de mai sus, vom da:

```
CLEAR
GO TOP
DO WHILE NOT EOF(1)
     IF EOF(2)
          X = REPL('', 10)
          Y = REPL('', 10)
          Z = 0
     ELSE
          X = B→NUME
          Y = B→PRENUME
          Z = B→SALARIU
     ENDIF
     ?MARCAV, X, Y, Z, PROD, CANT, PRET
     SKIP
ENDDO
CLOSE ALL
```

În cazul general o tabelă părinte poate fi legat prin relații de tip 1 - 1 cu mai multe tabele fiu.

Fie F_1 tabela părinte şi F_2^h, $h = \overline{1, p}$ tabelele fiu. Fie E_2^h expresia de indexare pentru F_2^h şi E_1^h expresia de relaţionare între F_1 şi F_2^h. Spunem că avem o relaţie de tip 1 –1 între F_1 şi F_2^h definită prin perechea (E_1^h, E_2^h) şi vom nota $F_1 \xrightarrow{(E^h_1, E^h_2)} F_2^h$, h= 1,p. Să presupunem că F_1 îl activăm în zona 1, iar F_2^h în zona h+1, h= 1,p. Atunci secvenţa care defineşte aceste relaţii 1-1 va fi:

```
SELE 1
USE F1
SELE 2
USE F2^1
INDEX ON E2^1 TO I1
SELE 3
USE F2^2
INDEX ON E2^2 TO I2
.
.
.
SELE p+1
USE F2^p
INDEX ON E2^p TO Ip
SELE 1
SET RELATION TO E1^1 INTO 2,…, E1^p INTO p+1
```

Să notăm $F_1=(I_1, \ldots , I_n)$, iar $F_2^h =(J_1^h, \ldots J^h_{mh})$.

Definirea relaţiilor de mai sus înseamnă

$$F_1 \xrightarrow{(E^h_1, E^h_2)} F_2^h, \; h = \overline{1, p}$$

Efectul acestor relaţii se materializează prin:

Poziţionarea lui F_1 pe I_j înseamnă poziţionarea lui F_2^h pe J_e^h, dacă J_e^h există cu proprietatea $E_1^h(I_j)=E_2^h(J_e^h)$ şi poziţionarea lui F_2^h pe EOF, dacă nu există J_e^h cu proprietatea menţionată. Acest lucru se realizează pentru fiecare $h = \overline{1, p}$.

Exemplu: În afară de tabelele PERSONAL şi VINZARI de mai sus să considerăm şi PRODUSE cu structura:

CODP(C, 5) - codul produsului

DENP(C, 10) - denumirea produsului

UM(C, 3) - unitatea de măsură

şi cu presupunerea CODP – cheie unică. Atunci, în afară de relaţia 1–1 între VINZARI şi PERSONAL definită prin (MARCAV, MARCA) vom putea defini şi o relaţie 1-1 între VÎNZĂRI şi PRODUSE folosind perechea (PROD, CODP).

Problemă: Să se realizeze un program (folosind relaţiile specificate mai sus) care pentru fiecare vînzare din VINZARI, afişează în afară de MARCAV, PROD, CANT, PRET şi numele, prenumele, salariul vînzătorului respectiv, precum şi denumirea şi unitatea de măsură a produsului vîndut.

Forma generală a comenzii `SET RELATION` este

$$\texttt{SET RELATION TO <expr1>INTO} \begin{Bmatrix} \texttt{< expN1 >} \\ \texttt{< expC1 >} \end{Bmatrix}, \ldots..,$$

$$\texttt{<exprp> INTO} \begin{Bmatrix} \texttt{< expNp >} \\ \texttt{< expCp >} \end{Bmatrix} \texttt{[ADDITIVE]}$$

`<expr1>,…, <exprp>` sînt expresii de relaționare, `<expN1>,…, <expNp>` sînt expresii numerice ce au ca valori numere de zone de lucru, `<expC1>,…, <expCp>` sînt expresii de tip caracter ce au ca valori numele alias ale unor fişiere (fie aliasul standard, fie aliasul definit prin comanda `USE…ALIAS`).

Dacă F_1 este tabela activă în zona curentă, iar $F_2^{p'}$ este tabela activă în zona de număr `<expN`$_{p'}$`>` sau care are alias-ul `<expC`$_{p'}$`>`, atunci se atabileşte o relație de tip 1 – 1 între F_1 şi $F_2^{p'}$, $p'=\overline{1,\ p}$. Expresia de relaționare este `<expr`$_{p'}$`>` şi tabela $F_2^{p'}$ trebuie să fie indexată. Dacă lipseşte `ADDITIVE`, atunci toate relațiile de tip 1 – 1 definite pentru F_1 ca fişier părinte sînt anulate şi rămîn numai cele definite de această comandă `SET RELATION`. În prezența cuvîntului `ADDITIVE` se păstrează şi vechile relații definite pentru F_1 ca părinte. Comanda:

```
SET RELATION TO
```

anulează toate relațiile definite pentru F_1 ca tabelă părinte.

Se pot crea relații de tip 1-n între tabele, adică pentru o înregistrare I_j a tabelei părinte să existe în general mai multe înregistrări în tabela fiu corespunzătoare.

Considerăm tabelele PERSONAL şi VINZARI de mai sus şi dorim să realizăm o relație de tip 1 – n între PERSONAL şi VINZARI. Practic ne interesează ca pentru fiecare persoană din PERSONAL să afişăm toate vînzările realizate de ea (dacă nu are nici o vînzare să afişăm spații).

Pentru a realiza o relație de tip 1 – n va trebui întîi să definim o relație de tip 1 – 1 folosind comanda `SET RELATION`, apoi vom da comanda `SET SKIP TO`.

În exemplul nostru, fie PERSONAL în zona 1 şi VINZARI în zona 2. Secvența ce defineşte relația 1 – n între cele două va fi:

```
SELECT 2
USE VINZARI
INDEX ON MARCAV TO IM
SELECT 1
USE PERSONAL
SET RELATION TO MARCA INTO 2
SET SKIP TO B
*B este alias standard al zonei a doua,
*unde este activ fişierul VINZARI.
```

Definirea acestei relații înseamnă următoarele:

Dacă tabela PERSONAL este poziționată pe o înregistrare să zicem I_1, atunci tabela VINZARI se poziționează pe prima corespunzătoare lui I_1, adică J_1. O comandă `SKIP` realizată pe PERSONAL nu trece la înregistrarea următoare din PERSONAL ci poziționează VINZARI pe J_2. O nouă comandă `SKIP` pe PERSONAL va poziționa VINZARI pe J_3 etc. Dacă pentru I_j din PERSONAL nu există înregistrarea respectivă J_e din VINZARI, atunci VINZARI se poziționează pe EOF.

Ex.: să afişăm pentru fiecare persoană vînzările realizate (dacă nu are vînzări, vom afişa spații). În afară de secvența de definire a relației 1 – n (de mai sus) vom da:

```
GO TOP
DO WHILE.NOT.EOF (1)
     IF EOF(2)
          X = SPACE (5)
          Y = 0
          Z = 0
     ELSE
          X = B→PROD
```

```
        Y = B→CANT
        Z = B→PRET
    ENDIF
    ?MARCA, NUME, PRENUME, SALARIU, X, Y, Z
    SKIP
ENDDO
CLEAR ALL
```

Forma generală a comenzii `SET SKIP TO` este

```
SET SKIP TO [<alias1>, <alias2>, …, <aliasp>]
```

Dacă F_1 este tabela activă în zona curentă, iar `<aliasj>` este nume alias pentru F_j, $j=\overline{1, p}$, atunci comanda stabileşte o relaţie de tip 1 – n între F_1 şi F_j. Evident între F_1 şi F_j trebuie să existe în prealabil definită o relaţie de tip 1 – 1 şi desigur F_j trebuie să fie indexată. Desigur, în acest caz expresia de indexare pentru F_j nu trebuie să fie cheie unică. Comanda în forma `SET SKIP TO` anulează relaţiile 1 – n existente ale lui F_1 pe poziţie de părinte şi deci rămîn aceste relaţii în forma 1 – 1.

Comanda

$$\texttt{SET RELATION OFF INTO} \begin{Bmatrix} \texttt{< expN >} \\ \texttt{< expC >} \end{Bmatrix}$$

anulează relaţia de tip 1- 1 între F_1 şi F_2, unde F_1 este tabela activă în zona curentă, iar F_2 este cea activă în zona de număr `<expN>` sau care are aliasul `<expC>`.
Se pot stabili relaţii de forma $F_1 \to F_2 \to \ldots \to F_h$ atît de tip 1 – 1 cît şi de tip 1 – n.

CAPITOLUL XII
OBIECTE DE CONTROL

Dacă selectăm opțiunea `Open` a submeniului `File` din meniul sistem FOX, atunci pe ecran apare fereastra sistem cu numele `OPEN` şi cu structura:

```
Open
                         Lista
[…]                      ↙                    Drive       [      C      ]
                                                                         ↖
[TC]                                                                       Liste ascunse
FOXHELP.DBF                                                              ↙
FOXUSER.DBF                               Directory   [ FOXPRO25    ]
PERSONAL.DBF
VINZARI.DBF
[ ] ALL Files                                          <<Open>>    ← Declanşator
 ↖                                                                    implicit
  Comutator                                             <New>
                                                                   ↖
[ DATABASE      ]   Type                                             Declanşatoare
                    ↖Lista ascunsa                     <Cancel>    ↙
```

În această fereastră sînt reprezentate nişte elemente numite obiecte de control. Din acestea unul singur poate fi selectat la un moment dat. Folosind tasta TAB sau SHIFT+TAB ne deplasăm cu bara luminoasă pe obiectul dorit. Atunci tastînd ENTER îl vom selecta. Folosind mouse-ul, deplasăm cursorul pe obiectul respectiv şi acționăm butonul stîng al mouse-ului.

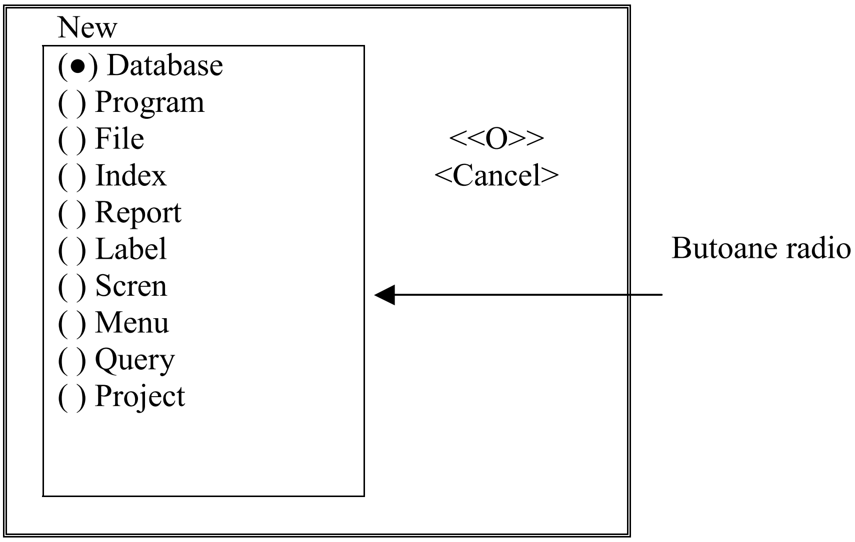
Sub Open avem obiectul numit "listă" ce afişează o mulțime de fişiere sau directoare ce constitue conținutul directorului curent. Deplasarea în listă se realizează cu tastele ↑, ↓ iar alegerea cu ENTER. O listă ascunsă este un tip special de listă în care afişarea elementelor listei se realizează numai după selectarea listei respective. Avem mai sus 3 liste ascunse: Type, Drive, Directory. Aceste cuvinte se numesc prompterii listelor respective.

Lista ascunsă are laturile jos şi dreapta duble. Selectarea unui element al listei ascunse se realizează prin poziționarea barei pe elementul dorit cu tastele ↑ , ↓ şi tastînd ENTER.

Un comutator este un obiect ce are 2 stări: activat, dezactivat. Starea activat este reprezentată prin X , iar starea dezactivat prin spațiu. Obiectul este reprezentat prin [ ]. Folosind tastatura, activarea sau dezactivarea comutatorului se realizează selectînd comutatorul şi dînd ENTER sau spațiu, starea comutatorului schimbîndu-se în cealaltă. Folosind mouse-ul, îl poziționăm pe comutator şi acționăm butonul stîng al mouse-ului. În fereastra `OPEN` de sus avem comutatorul cu prompterul "All Files" ce este dezactivat.

Inițierea unei acțiuni proiectate de utilizator se face cu ajutorul declanşatoarelor sau butoanelor. Reprezentarea declanşatoarelor se realizează cu ajutorul parantezelor unghiulare deschise şi închise: < , >.

În fereastra `OPEN` de mai sus există 3 declanşatoare, numite Open, New, Cancel. Selectarea unui declanşator se realizează prin poziţionarea barei pe el (prin TAB sau SHIFT+TAB) şi tastînd ENTER sau SPACE. Există declanşatoare implicite (cum ar fi <<Open>>) şi simple (<New>, <Cancel>). Un declanşator implicit se poate acţiona şi prin CTRL+ENTER, fără acţionarea lui. Acţionarea lui Open va avea ca efect deschiderea fişierului selectat din Listă. Acţionarea lui New va avea ca efect deschiderea ferestrei de editare a unui nou program dacă în Type s-a selectat Program etc. Acţionarea lui Cancel implică închiderea acestei ferestre. Dacă selectăm opţiunea `New` a submeniului `File`, atunci pe ecran va apare fereastra `New`:



La un moment dat se poate activa un singur buton radio, plasînd ● pe el. Activarea se face prin selectarea lui cu TAB sau SHIFT+TAB şi tastarea lui ENTER sau SPACE.

Utilizatorul are posibilitatea să-şi definească în programe aceste obiecte de control, utilizînd comanda `@l,c GET` fără componenta `SAY` folosită pentru operaţii de intrare ieşire. În această comandă se pot utiliza componentele `FUNCTION` şi `PICTURE` ca şi în cazul comenzii `@l,c SAY…GET`.

De data aceasta codul de funcţie (primul din `FUNCTION`) defineşte tipul obiectului de control. <u>Comutatoarele</u> sînt definite specificînd în `FUNCTION` *C $c_1 \ldots c_k$, unde $c_1 \ldots c_k$ este un text ce va fi afişat lîngă comutator, numit prompt-ul său. Aceiaşi acţiune are loc dacă în `PICTURE` dăm @*C $c_1 \ldots c_k$.

Comanda `GET` pentru crearea comutatorului are forma:

$$\texttt{@l,c GET} \begin{Bmatrix} \texttt{< var >} \\ \texttt{< cimp >} \end{Bmatrix} \begin{Bmatrix} \texttt{FUNCTION < expC1 >} \\ \texttt{PICTURE < expC2 >} \end{Bmatrix}$$

$$\texttt{[DEFAULT <expr>] [SIZE <expN2>, <expN3>]}$$

$$\left[\begin{Bmatrix} \texttt{ENABLE} \\ \texttt{DISABLE} \end{Bmatrix}\right] \texttt{[MESSAGE <expC5>}$$

$$\texttt{[VALID<expL1>] [WHEN<expL2>]}$$

$$\begin{Bmatrix} \texttt{COLOR SCHEME < expN5 >} \\ \texttt{COLOR c1/d1....ch/dh} \end{Bmatrix}$$

l,c reprezintă punctul de unde va începe afişarea comutatorului. Alegerea făcută de utilizator (activat sau dezactivat) va fi memorată în variabila `<var>` sau în cîmpul definit de <cimp> al unui fişier activ poziţionat pe o înregistrare. Dacă tipul <var> sau <cimp> este logic, atunci în cazul activării, plasarea lui X între [ ], va însemna valoarea .T., în caz contrar .F. Dacă tipul este numeric, în cazul activării comutatorului se plasează în <var> sau <cimp> o valoare nenulă, altfel valoarea nulă.

Trebuie să existe o comandă `READ` ce realizează efectiv operaţia de creare a unui comutator. În `FUNCTION` sau `PICTURE` se pot preciza încă două coduri de funcţie şi anume: `T` însemnînd că selectarea comutatorului termină comanda `READ` curentă, iar codul de funcţie `N` înseamnă că selectarea comutatorului nu va termina comanda `READ` curentă.

Interzicerea accesului la comutator se poate face prin componenta `FUNCTION '*C \\ C`$_1$`..C`$_h$`'`.

Componenta `MESSAGE <expC5>` implică afişarea mesajului <expC5> pe ultima linie a ecranului, la activarea comutatorului. Celelalte componente au fost discutate cu ocazia comenzii `@SAY… GET`.

O <u>listă</u> se defineşte specificînd `FUNCTION '&'` sau `PICTURE '@&'`. În plus se pot folosi codurile de funcţie `T` ce înseamnă că `READ` se termină la selectarea acestui obiect (deci celelalte `@GET` ce urmeză pînă la `READ` vor fi ignorate), iar `N` nu implică terminarea execuţiei lui `READ` curent.

Pentru definirea obiectului de tip listă, comanda are forma:

$$\texttt{@l,c GET}\left\{\begin{array}{l}\texttt{< var >}\\ \texttt{< cimp >}\end{array}\right\}\texttt{FROM}\left\{\begin{array}{l}\texttt{< tablou > [RANGE l1, l2]}\\ \texttt{POPUP < numepopup >}\end{array}\right\}$$

$$\left\{\begin{array}{l}\texttt{FUNCTION < expC1 >}\\ \texttt{PICTURE < expC2 >}\end{array}\right\}\texttt{[DEFAULT …] …}$$

cu restul componentelor ca la comutatoare.

Selecţia elementului din listă va fi memorată în <var> sau <cimp>. Acestea trebuie să fie de tip numeric sau caracter. Dacă este de tip N, atunci se va memora poziţia din listă a elementului selectat, dacă este de tip C se va memora chiar elementul listei. Elementele listei se preiau fie dintr-un tablou, fie dintr-un meniu vertical (popup).

Cînd se construieşte lista dintr-un tablou, atunci

- dacă tabloul este unidimensional, elementele tabloului se plasează în ordinea indicilor în listă.
- dacă tabloul este bidimensional, atunci se plasează în listă numai prima coloana a tabloului.

Dacă dorim să preluăm alte elemente ale tabloului bidimensional pentru a le include în listă, atunci vom folosi `RANGE` l_1, l_2. Fie tabloul TAB(m, h). Elementele acestuia se memorează astfel: TAB(1, 1), TAB(1, 2),…, TAB(1, h), TAB(2,1),…, TAB(2, h),…, TAB(m, 1), TAB(m, 2),…, TAB(m, h).

În această numerotare avem şirul 1, 2,…., m*h. Dacă folosim `RANGE` l_1, l_2 cu $l_1, l_2 \in \{1, 2, \dots m*h\}$, $l_1 \le l_2$, atunci vor fi incluse în lista creată toate elementele tabloului cu numerele cuprinse între l_1 şi l_2 inclusiv. Desigur că pentru preluarea tuturor elementelor tabloului vom folosi `RANGE 1, m*h`.

<u>Lista ascunsă</u> se defineşte prin `PICTURE '@^'` sau `FUNCTION '^'`. Se pot folosi aceleaşi coduri de funcţie T şi N ca la celelalte obiecte. Comanda de definire a listei ascunse este aceeaşi ca la listă, deosebirea constă în `PICTURE` sau `FUNCTION` şi în absenţa clauzei `FROM POPUP < nume popup>`. Mai mult, `PICTURE` poate

avea forma `'@^o1;o2;..;om'` Atunci `o1, o2,…, om` vor fi elementele listei ascunse. Similar pentru `FUNCTION '^ o1;o2;…;om'`.

În `PICTURE` sau `FUNCTION` se pot utiliza şi codurile T, N.

Declanşatoarele (sau butoanele de pornire) au `PICTURE '@* c1..cn'` sau `FUNCTION '* c1…cn'` unde `c1….cn` este textul butonului.

Sintaxa comenzii este aceeaşi ca la comutatoare. Dacă se crează mai multe butoane, atunci textele lor se separă prin `';'` Astfel în `FUNCTION` putem da `FUNCTION '* den1;den2;….denh'` unde den_i sunt denumirile butoanelor ce se crează. În afara codurilor de funcţie T şi N ca celelalte obiecte de control se pot utiliza codurile H – şir orizontal de butoane, V – şir vertical de butoane.

Butoanele invizibile se construiesc cu `FUNCTION '*I` $c_1..c_n$`'` sau `PICTURE '@*I` $c_1…c_n$`'` folosite în aceeaşi comandă `@l,c GET` ca pentru butoanele de pornire. Şi aici se pot utiliza codurile de funcţii T, N, H, V ca la butoanele de pornire. Dezactivarea unui buton invizibil se realizează prin plasarea caracterelor `\\` în faţa numelui său (`c1..cn`). Pot fi create mai multe butoane invizibile prin separarea acestora cu caracterul `';'` ca la butoane de pornire.

Butoanele radio se definesc cu `FUNCTION '*R c1…cn'` sau `PICTURE '@*R c1…cn'` utilizate în aceeaşi comandă `@l,c GET` de la comutatoare. Şirul `c1…cn` va fi numele butonului (promptul său). Activarea butonului (ca şi a celorlalte obiecte de control) se face cu `READ`.

În `FUNCTION` sau `PICTURE` se pot utiliza şi codurile de funcţie T, N ca în cazul celorlalte obiecte de control. La fel H şi V pentru dispunerea pe orizontală sau verticală. Se pot crea mai multe butoane radio folosind `';'` pentru terminarea numelui unuia, deci

`FUNCTION '*R den1;den2;…denh'` sau

`PICTURE '@*R den1;den2;…denh'`.

Putem crea mai multe obiecte de control cu comenzi `@l,c GET`, toate urmate de un READ. Evident în program utilizăm variabila `<var>` sau cîmpul `<cimp>` din `GET` pentru a testa valoarea acesteia şi a executa procedura ataşată. Mai există posibilitatea definirii unei regiuni de editare text cu ajutorul comenzii `@l,c EDIT`. Sintaxa comenzii este aceeaşi ca la `@l,c GET`, cu deosebire că în locul cuvîntului `GET` se utilizează `EDIT`. `<expN1>` din `SIZE` specifică numărul de linii ale zonei de editare, `<expN2>` numărul de coloane ale acesteia.

CAPITOLUL XIII
PROBLEME

1. Se consideră tabelele:

a) CARTI cu structura:
COTA (C, 5) – cota cărții
AUTOR (C, 10) – autorul
TITLUL (C, 12) – titlul cărții
NRP (N, 3) – numărul de permis al celui ce a împrumutat cartea
DATAI (D) – data împrumutului.

b) CITITORI cu structura:
NRP (N, 3) – număr permis
NUME(C, 10) – nume cititor

Să se creeze un meniu orizontal MO cu opțiunile OPERATII şi IESIRE. Pentru acestea se vor ataşa meniurile verticale V1, V2, respectiv. Meniul V1 va avea opțiunile LISTA, R1, R2, iar V2 va avea opțiunile IESIRE FOX şi IESIRE DOS. Să se creeze procedurile necesare pentru fiecare opțiune. Procedura LISTA va citi cu comanda `@SAY, GET` un nume de fişier (din cele două) şi va lista acest fişier cîte 6 înregistrări pe ecran. Procedura R1 afişează pentru fiecare carte numele cititorului. Apoi va afişa cititorii restanțieri cu mai mult de 3 zile. Procedura R2 citeşte cu `@SAY,GET` un nume de cititor şi afişează toate cărțile împrumutate de acel cititor în ordine crescătoare a numelor autorilor. Procedura asociată opțiunii IESIRE FOX va realiza ieşirea în fereastra de comandă FOX, iar cea asociată lui IESIRE DOS va realiza ieşirea în sistemul de operare DOS. Cîmpul COTA este cheie unică pentru CARTI, fiecare din NRP şi NUME sînt chei unice pentru CITITORI.

2) Se consideră tabelele:

a) PERSONAL cu structura:
MARCA (N, 3) – marca persoanei
NUME (C, 10) – numele persoanei
SALARIU (N, 7)
MARCASEF (N, 3) – marca şefului
CODSECTIE (N, 1) – codul secției

b) SECTIE cu structura:
CODSECTIE (N, 1) – codul secției
DENS (C, 10) – denumirea secției

Să se realizeze un program care citeşte cu `@SAY, GET` un nume de fişier (din cele două) şi listează acest fişier, cîte 7 înregistrări pe un ecran. Apoi, după citirea unui nume de persoană cu `ACCEPT` sau `INPUT`, listează toți şefii ierarhic superiori ai acestuia. (persoana fără şef are MARCASEF=0). Apoi programul citeşte cu `INPUT` o denumire de secție şi afişează toate persoanele din aceeaşi secție şi suma totală a salariilor acestora.

3) Se consideră tabelele

a) DEPUNERI cu structura
NRCONT (N, 5) – numărul contului
SUMA (N, 7) – suma depusă
DATA (D) – data depunerii.

b) CONTURI cu structura

NRCONT (N, 5) – numărul contului
NUME(C, 10) – numele depunătorului

O persoană poate avea mai multe conturi şi pentru fiecare cont, mai multe depuneri.

Să se realizeze un program care:

- citeşte două date calendaristice şi afişează toate depunerile realizate în intervalul respectiv;
- pentru fiecare depunător afişează toate depunerile sale şi totalul acestor sume.

4. Se consideră tabelele:

a) URMCONTR (urmăriri contracte) cu structura:
CODC (N, 3) – cod contract
CODP (N, 4) – cod produs
CANTR (N, 6) – cantitatea realizată la produsul CODP pentru contractul CODC
CANTC (N, 7) – cantitatea cerută (contractată pentru perechea (CODC,CODP)).

b) LIVRĂRI (livrări la contracte) cu structura:
CODC (N, 3) – cod contract
CODP (N, 4) – cod produs
CANTL (N, 5) – cantitate livrată

Să se construiască un program care:

- actualizează fişierul URMCONTR cu LIVRĂRI;
- pentru fiecare contract să se afişeze toate produsele pentru care CANTR = CANTC şi numărul acestor produse;
- afişează contractele onorate integral.

5) Se consideră tabelele:

a) CONTRACTE cu structura:
CODC (N, 3) – codul contractului
CODP (N, 4) – codul produsului
CANT (N, 6) – cantitatea contractată

b) PRODUSE cu structura:
CODP (N, 4) – codul produsului
DENP (C, 10) – denumire produs
PRET (N, 2) – preț unitar.

Să se realizeze un program care:

- pentru fiecare contract calculează valoarea totală a contractului şi afişează această valoare împreună cu codul contractului;
- citeşte cu `@SAY, GET` o denumire de produs şi calculează pentru acest produs cantitatea totală prevăzută în toate contractele şi afişează toate contractele în care apare acel produs.

CODP este cheie unică în PRODUSE. Se pot utiliza indecşi.

6. Se consideră tabelele:

a) CONTRACT cu structura:
CODC (N, 3) – cod contract
CODB (N, 2) – cod beneficiar
CODF (N, 2) – cod furnizor

CODP (N, 4) – cod produs
CANT (N, 5) – cantitate
PRET (N, 2) –preț
b) SOCIET (societăți) cu structura:
CODS (N, 2) – codul societății ce poate fi beneficiar sau furnizor
DENS (C, 10) – denumire societate

Să se realizeze un program care:

- citeşte un cod de societate şi caută în SOCIET denumirea respectivă utilizînd un index, apoi modifică această denumire;
- calculează valoarea tuturor contractelor;
- pentru fiecare contract afişează codul beneficiarului, denumirea beneficiarului, codul furnizorului, denumirea furnizorului;
- citeşte o denumire de societate, listează toate înregistrările din CONTRACT pentru care acea societate este beneficiară şi calculează, pentru fiecare contract al acestui beneficiar, valoarea totală.

7. Se consideră tabelele:
a) AUTO cu structura:
MARCA (N, 5) – marca autoturismului
NUME(C, 10) – numele proprietarului
NRCIRC (C, 5) – număr înmatriculare
b) REPARATII
NRCIRC (C, 5) – număr de înmatriculare
DATAR (D) – data reparației
SUMA (N, 5) – suma plătită pentru reparație

Se presupune că un proprietar are o singură maşină, ce poate avea mai multe reparații.

Să se realizeze un program care:

- pentru fiecare nume de proprietar afişează suma totală plătită pentru reparații;
- citeşte un nume de proprietar şi afişează pentru acesta suma plătită în fiecare lună pentru reparațiile făcute, precum şi suma totală;

8. Se consideră tabelele:
a) FOND (fond de cărți) cu structura:
COTA (C, 4) – cota cărții
AUTOR (C, 10) – autorul
TITLUL (C,10) – titlul cărții
IND (N, 1) ce are valoare 1 dacă acea carte e împumutată şi 0 altfel.
b) ÎMPRUMUT cu structura:
COTA (C, 10) – cota cărții împrumutate
NUME(C, 20) – numele cititorului
DATAI (D) – data împrumutului

Să se construiască un program care:

- citeşte o cotă. Dacă cota există în FOND şi nu există în IMPRUMUT va afişa o fişă de împrumut, citind şi numele unui cititor. Actualizează apoi FOND şi IMPRUMUT. Dacă cota citită există în FOND şi există şi în IMPRUMUT se va afişa cartea şi persoana ce a împrumutat-o. Dacă nu există nici în FOND, nici în IMPRUMUT se va afişa un mesaj;
- citeşte un autor şi afişează toate cărțile acestuia.

9. Se consideră tabelele:
a) DEPOZITE cu structura:
CODD (N, 3) – cod depozit
CODP (N, 4) – cod produs
CANTD (N, 5) – cantitatea existentă
b) VINZARI (la produsele din depozite) cu structura:
CODPV (N, 4) – codul produsului vîndut
CODDV (N, 3) – codul depozitului
CANT (N, 4) – cantitatea vîndută
PRET (N, 2) – preț unitar

Să se realizeze un program care:

- pentru fiecare depozit calculează valoarea totală vîndută pentru produsele din acel depozit;
- pentru fiecare depozit şi produs calculează valoarea totală vîndută la acel produs, din acel depozit;
- actualizează DEPOZITE cu vînzările din VINZARI.

CAPITOLUL XIV
DEPENDENŢE FUNCŢIONALE

1 Definiţie. Proprietăţi. Sisteme de reguli de inferenţă.

Fie $X, Y \subseteq U$. Vom nota sintactic o dependenţă funcţională prin $X \to Y$. Semantic, vom spune că o relaţie r peste U satisface dependenţa funcţională $X \to Y$ dacă: $(\forall t_1, t_2)(t_1, t_2 \in r)[t_1[X] = t_2[X] \Rightarrow t_1[Y] = t_2[Y]]$. Dacă $X = \emptyset$, atunci spunem că r satisface $\emptyset \to Y$ dacă $(\forall t_1, t_2)(t_1, t_2 \in r)[t_1[Y] = t_2[Y]]$, altfel spus, $r[Y]$ constă dintr-un singur element. Dacă $Y = \emptyset$, atunci considerăm că orice relaţie r peste U satisface dependenţa funcţională $X \to \emptyset$. Dacă r satisface dependenţa funcţională $X \to Y$, atunci există o funcţie $\varphi : r[X] \to r[Y]$ definită prin: $\varphi(t) = t'[Y]$, unde $t' \in r$ şi $t'[X] = t \in r[X]$.

Dacă r satisface $X \to Y$ se mai spune că X determină funcţional pe Y în r.

Proprietăţi ale dependenţelor funcţionale:

FD1. (Reflexivitate) Dacă $Y \subseteq X$, atunci r satisface $X \to Y$ pentru orice relaţie r peste U.

FD2 (Extensie) Dacă r satisface $X \to Y$ şi $Z \subseteq W$, atunci r satisface $XW \to YZ$.

FD3 (Tranzitivitate) Dacă r satisface $X \to Y$ şi $Y \to Z$, atunci r satisface $X \to Z$.

FD4 (Pseudotranzitivitate) Dacă r satisface $X \to Y$ şi $YW \to Z$, atunci r satisface $XW \to Z$.

FD5 (Uniune) Dacă r satisface $X \to Y$ şi $X \to Z$, atunci r satisface $X \to YZ$.

FD6 (Descompunere) Dacă r satisface $X \to YZ$, atunci r satisface $X \to Y$ şi $X \to Z$.

FD7 (Proiectabilitate) Dacă r peste U satisface $X \to Y$ şi $X \subset Z \subseteq U$, atunci $r[Z]$ satisface $X \to Y \cap Z$.

FD8 (Proiectabilitate inversă) Dacă $X \to Y$ este satisfăcută de o proiecţie a lui r, atunci $X \to Y$ este satisfăcută de r.

Proprietăţile enunţate rezultă imediat aplicând definiţia satisfacerii unei dependenţe funcţionale de o relaţie r.

Fie Σ o mulţime de dependenţe funcţionale peste U. Spunem că $X \to Y$ este consecinţă din Σ, dacă orice relaţie ce satisface toate dependenţele lui Σ va satisface şi $X \to Y$. Vom nota această situaţie prin $\Sigma \models X \to Y$.

Deci $\Sigma \models X \to Y$ dacă pentru $(\forall r)[\forall \sigma \in \Sigma$, r satisface $\sigma \rightsquigarrow r$ satisface $X \to Y\}$.

Fie $\Sigma^* = \{X \to Y | \Sigma \models X \to Y\}$. Fie Σ_1 o mulţime de dependenţe funcţionale. Spunem că Σ_1 constituie o acoperire pentru Σ^*, dacă $\Sigma_1^* = \Sigma^*$.

Propoziţia 1.1 *Pentru orice mulţime Σ de dependenţe funcţionale există o acoperire Σ_1 pentru Σ^*, astfel încât toate dependenţele din Σ_1 sunt de forma $X \to A$, A fiind un atribut din U.*

Demonstraţie. Pentru fiecare $X \to Y \in \Sigma$, cu $Y = B_1 B_2 \ldots B_h$, considerăm $X \to B_j$ incluse în Σ_1, $j = \overline{1,h}$. După proprietăţile FD5 şi FD6 avem: r satisface $X \to Y$ dacă şi numai dacă r satisface $X \to B_j$, $j = \overline{1,h}$. După aceleaşi proprietăţi şi din modul de definire a lui Σ_1 avem: r satisface σ, $\forall \sigma \in \Sigma$ dacă şi numai dacă r satisface σ_1, $\forall \sigma_1 \in \Sigma_1$. Aceasta din urmă conduce la $\Sigma \models X \to Y$ dacă şi numai dacă $\Sigma_1 \models X \to Y$, ceea ce înseamnă $\Sigma^* = \Sigma_1^*$. $\square$

Propoziţia 1.2 $\Sigma \models X \to Y$ *dacă şi numai dacă* $\Sigma \models X \to B_j$ *pentru* $j = \overline{1,h}$, *unde* $Y = B_1 \ldots B_h$.

Justificarea propoziţiei rezultă imediat din FD5 şi FD6. Din cele două propoziţii de mai sus, rezultă că studiul dependenţelor funcţionale, din punct de vedere al relaţiei de "consecinţă", se reduce la studiul acestei "consecinţe" pe mulţimea dependenţelor funcţionale în care membrul doi are un singur atribut.

În continuare vom considera reguli formale de deducere a noi dependenţe funcţionale, pornind de la o mulţime dată Σ.

Fie $\mathcal{R}$ o mulţime de reguli formale de deducere pentru dependenţe funcţionale şi Σ o mulţime de dependenţe funcţionale. Spunem că $X \to Y$ are o demonstraţie în Σ utilizând regulile $\mathcal{R}$ şi vom nota $\Sigma|\!\frac{\ }{\mathcal{R}} X \to Y$, dacă există şirul $\sigma_1, \sigma_2, \ldots, \sigma_n$, astfel încât:

a) $\sigma_n = X \to Y$ şi

b) pentru orice $i = \overline{1,n}$, $\sigma_i \in \Sigma$ sau există o regulă din $\mathcal{R}$ de forma $\frac{\sigma_{j_1}, \sigma_{j_2}, \ldots, \sigma_{j_k}}{\sigma_i}$, unde $j_1, j_2, \ldots, j_k < i$ (adică σ_i se obţine utilizând o regulă din $\mathcal{R}$, cu premizele existente în şir înainte de σ_i). Corespunzător proprietăţilor FD1–FD6 se pot defini reguli formale de deducere a dependenţelor funcţionale:

$$\text{FD1f}: \frac{Y \subseteq X}{X \to Y} \qquad \text{FD4f}: \frac{X \to Y,\ YW \to Z}{XW \to Z}$$

$$\text{FD2f}: \frac{X \to Y,\ Z \subseteq W}{XW \to YZ} \qquad \text{FD5f}: \frac{X \to Y,\ X \to Z}{X \to YZ}$$

$$\text{FD3f}: \frac{X \to Y,\ Y \to Z}{X \to Z} \qquad \text{FD6f}: \frac{X \to YZ}{X \to Y},\ \frac{X \to YZ}{X \to Z}$$

Armstrong [2] a definit următoarele reguli de inferenţă (numite *axiomele lui Armstrong*):

$$\text{A1}: \quad \frac{}{A_1 \ldots A_m \to A_i},\ i = \overline{1,m}$$

$$\text{A2}: \quad \frac{A_1 \ldots A_m \to B_1 \ldots B_r}{A_1 \ldots A_m \to B_j}, j = \overline{1,r}, \frac{A_1 \ldots A_m \to B_j, j = \overline{1,r}}{A_1 \ldots A_m \to B_1 \ldots B_r}$$

$$\text{A3}: \quad \frac{A_1 \ldots A_m \to B_1 \ldots B_r,\ B_1 \ldots B_r \to C_1 \ldots C_p}{A_1 \ldots A_m \to C_1 \ldots C_p}$$

unde A_i, B_j, C_k sunt atribute.

Observaţia 1.1 *Regula A3 este în fond FD3f (tranzitivitatea).*

Propoziţia 1.3 *Regulile FD4f, FD5f, FD6f se exprimă cu ajutorul regulilor FD1f, FD2f, FD3f.*

În adevăr, fie $X \to Y$ şi $YW \to Z$ date. Aplicăm FD2f pentru $X \to Y$ şi $W \subseteq W$ şi obţinem $XW \to YW$. Aceasta din urmă şi $YW \to Z$ şi FD3f conduc la $XW \to Z$.

Fie date $X \to Y$ şi $X \to Z$. Aplicând FD2f pentru $X \to Y$ şi $X \subseteq X$ obţinem $X \to XY$; de asemenea aplicăm FD2f pentru $X \to Z$ şi $Y \subseteq Y$ şi obţinem $XY \to YZ$. Prin tranzitivitate din $X \to XY$ şi $XY \to YZ$ obţinem $X \to YZ$.
Fie $X \to YZ$. După FD1f obţinem $YZ \to Y$ şi $YZ \to Z$, aplicând FD3f se obţine $X \to Y$ şi $X \to Z$.
Să notăm prin $\Sigma^+_{\mathcal{R}} = \{X \to Y | \Sigma|\overline{_{\mathcal{R}}} X \to Y\}$.
Fie $\mathcal{R}_1 = \{$FD1f, FD2f, FD3f$\}$, $\mathcal{R}_2 = \mathcal{R}_1 \cup \{$FD4f, FD5f, FD6f$\}$ $\mathcal{R}_A = \{A_1, A_2, A_3\}$.

Observaţia 1.2 $\Sigma^+_{\mathcal{R}_1} = \Sigma^+_{\mathcal{R}_2}$ *având în vedere propoziţia 1.3.*

Propoziţia 1.4 *Regulile din $\mathcal{R}_1$ se exprimă prin cele din $\mathcal{R}_A$ şi invers.*

Demonstraţie. Fie $X = A_1 \ldots A_m$ şi $Y = A_{i_1} \ldots A_{i_k}$, $\{i_1, \ldots, i_k\} \subseteq \{1, 2, \ldots, m\}$. Aplicând regula A1 obţinem
$X \to A_{i_1}, \ldots,$ $X \to A_{i_k}$, apoi considerând A2 partea a II-a obţinem $X \to A_{i_1} \ldots A_{i_k}$, adică $X \to Y$. Deci FD1f se exprimă prin regulile din $\mathcal{R}_A$.

Fie $X \to Y$ şi $Z \subseteq W$ date. Evidenţiem atributele din fiecare:
$X = A_1 \ldots A_m$, $Y = B_1 \ldots B_p$, $W = C_1 \ldots C_q$, $Z = C_{i_1} \ldots C_{i_k}$ cu $\{i_1, \ldots, i_k\} \subseteq \{1, 2, \ldots, q\}$.
Din $X \to Y$ prin A2 partea a I-a obţinem $X \to B_j$, $j = \overline{1, p}$.
Prin A1 şi A2 obţinem $XW \to X$. Din $X \to B_j$ şi $XW \to X$ obţinem $XW \to B_j$, $j = \overline{1, p}$ utilizând A3. Prin A1 obţinem $XW \to C_{i_l}$, $l = \overline{1, k}$. Aplicând A2 partea a II-a pentru $XW \to B_j$, $j = \overline{1, p}$ şi $XW \to C_{i_l}$, $l = \overline{1, k}$ obţinem $XW \to YZ$.
Regula FD3f este exact A3.
Regula A1 se exprimă numai prin FD1f. Regula A2 partea a I-a se exprimă aplicând regula FD6f de $r - 1$ ori, iar FD6f se exprimă cu ajutorul celor din $\mathcal{R}_1$, după propoziţia 1.3. $\square$

Dacă $\frac{\sigma_{j_1}, \ldots, \sigma_{j_h}}{\sigma_i} \in \mathcal{R}_1$ şi se exprimă cu ajutorul regulilor lui R_2, notăm prin $trans_{\mathcal{R}_2}(\sigma_{j_1}, \ldots, \sigma_{j_h}, \sigma_i)$ şirul de dependenţe ce se obţine pornind de la premizele $\sigma_{j_1} \ldots \sigma_{j_h}$ şi aplicând regulile lui R_2 pentru obţinerea lui σ_i.

Propoziţia 1.5 *Fie $\mathcal{R}'_1$ şi $\mathcal{R}'_2$ două mulţimi de reguli, astfel încât $\mathcal{R}'_1$ se exprimă prin $\mathcal{R}'_2$ şi invers. Atunci $\Sigma^+_{\mathcal{R}'_1} = \Sigma^+_{\mathcal{R}'_2}$ pentru orice mulţime Σ de dependenţe funcţionale.*

Demonstraţie. Fie $X \to Y \in \Sigma^+_{\mathcal{R}'_1}$. Să arătăm că

$$X \to Y \in \Sigma^+_{\mathcal{R}'_2} \tag{$*$}$$

Există şirul $\sigma_1, \sigma_2, \ldots, \sigma_n = X \to Y$, astfel încât pentru orice i, $i = \overline{1,n}$ avem:

a) $\sigma_i \in \Sigma$ sau

b) există $\alpha = \frac{\sigma_{j_1}, \ldots, \sigma_{j_h}}{\sigma_i} \in \mathcal{R}'_1$ cu $j_1, j_2, \ldots, j_h < i$

Demonstraţia o realizăm prin inducţie după n.

Dacă $n = 1$, atunci putem avea:

c) $\sigma_1 \in \Sigma$ şi deci $\sigma_1 \in \Sigma^+_{\mathcal{R}'_2}$ sau

d) $\frac{}{\sigma_1} \in \mathcal{R}'_1$

În cazul d) şirul $trans_{\mathcal{R}'_2}(\ ,\sigma_1)$ constituie o demonstraţie pentru σ_1 în Σ, utilizând $\mathcal{R}'_2$, adică $\sigma_1 \in \Sigma^+_{\mathcal{R}'_2}$.

Presupunem afirmaţia $(*)$ valabilă în cazul în care $X \to Y$ are demonstraţii în Σ utilizând reguli din $\mathcal{R}'_1$ şi lungimea demonstraţiei este mai mică sau egală cu n.

Fie acum $X \to Y \in \Sigma^+_{\mathcal{R}'_1}$ cu lungimea demonstraţiei egală cu $n+1$. Există şirul $\sigma_1, \sigma_2, \ldots, \sigma_n, \sigma_{n+1} = X \to Y$, astfel încât pentru orice i, $1 \le i \le n+1$, avem:

a1) $\sigma_i \in \Sigma$ sau

b1) există $\alpha = \frac{\sigma_{j_1}, \ldots \sigma_{j_h}}{\sigma_i} \in \mathcal{R}'_1$ cu $j_1, \ldots, j_h < i$

Dacă $\sigma_{n+1} \in \Sigma$, atunci $\sigma_{n+1} = X \to Y \in \Sigma^+_{\mathcal{R}'_2}$.

Dacă σ_{n+1} se obţine prin b1), atunci există $\frac{\sigma_{j_1}, \ldots \sigma_{j_h}}{\sigma_{n+1}} \in \mathcal{R}'_1$ cu $j_1, \ldots, j_h < n+1$.

După ipoteza inducţiei avem: $\sigma_{j_1}, \ldots, \sigma_{j_h} \in \Sigma^+_{\mathcal{R}'_2}$.

Fie $dem_{\mathcal{R}'_2}(\sigma_{j_i})$ o demonstraţie pentru σ_{j_i} în Σ, utilizând regulile lui $\mathcal{R}'_2$, $i = \overline{1,h}$. Atunci şirul:

$$dem_{\mathcal{R}'_2}(\sigma_{j_1}), \ldots, dem_{\mathcal{R}'_2}(\sigma_{j_h}),\ trans_{\mathcal{R}'_2}(\sigma_{j_1}, \ldots, \sigma_{j_h}, \sigma_{n+1})$$

constituie o demonstraţie pentru σ_{n+1} în Σ, utilizând regulile din $\mathcal{R}'_2$. Deci $\sigma_{n+1} \in \Sigma^+_{\mathcal{R}'_2}$.
Relaţia $(*)$ înseamnă $\Sigma^+_{\mathcal{R}'_1} \subseteq \Sigma^+_{\mathcal{R}'_2}$. Raţionamentul fiind simetric, rezultă şi incluziunea inversă, deci egalitatea dorită. $\square$

Consecinţa 1.1 $\Sigma^+_{\mathcal{R}_1} = \Sigma^+_{\mathcal{R}_A}$.

Fie $X \subseteq U$ şi $\mathcal{R}$ o mulţime de reguli de inferenţă. Să notăm prin $X^+_{\mathcal{R}} = \{A | \Sigma |\overline{_{\mathcal{R}}} X \to A\}$

Lema 1.1 $\Sigma|\overline{_{\mathcal{R}_1}} X \to Y$ *dacă şi numai dacă* $Y \subseteq X^+_{\mathcal{R}_1}$.

Demonstraţie. Fie $Y = A_1 A_2 \ldots A_m$, $A_i \in U$, $i = \overline{1,m}$. Presupunem că $Y \subseteq X^+_{\mathcal{R}_1}$. Atunci $A_i \in X^+_{\mathcal{R}_1}$, $i = \overline{1,m}$, deci $\Sigma|\overline{_{\mathcal{R}_1}} X \to A_i$, $i = \overline{1,m}$. Aplicând regula A2 partea a II-a (A2 se exprimă cu ajutorul regulilor din $\mathcal{R}_1$) obţinem: $\Sigma|\overline{_{\mathcal{R}_1}} X \to Y$.

Invers, dacă $\Sigma|\overline{_{\mathcal{R}_1}} X \to Y$, atunci aplicând regula A2, partea I-a (A2 se exprimă cu ajutorul regulilor din $\mathcal{R}_1$) se obţine: $\Sigma|\overline{_{\mathcal{R}_1}} X \to A_i$, $i = \overline{1,m}$, ceea ce înseamnă $A_i \in X^+_{\mathcal{R}_1}$, $i = \overline{1,m}$, deci $Y \subseteq X^+_{\mathcal{R}_1}$. $\square$

Lema 1.2 *Fie* Σ *o mulţime de dependenţe funcţionale şi* $\sigma : X \to Y$ *o dependenţă funcţională astfel încât* $\Sigma|\not\overline{_{\mathcal{R}_1}} X \to Y$. *Atunci există o relaţie* r_σ *ce satisface toate dependenţele din* Σ *şi* r_σ *nu satisface* $X \to Y$.

Demonstraţie. Avem $X^+_{\mathcal{R}_1} \subsetneq U$, căci în caz contrar, $X^+_{\mathcal{R}_1} = U$ şi utilizând lema 1.1 şi faptul că $Y \subseteq U$, obţinem $\Sigma|\overline{_{\mathcal{R}_1}} X \to Y$, deci contradicţie.
Definim relaţia $r_\sigma = \begin{pmatrix} t_1 \\ t_2 \end{pmatrix}$, unde t_1 este uplul conţinînd 1 pentru toate atributele din U, iar t_2 este definit prin: $t_2[A] = 1$ dacă $A \in X^+_{\mathcal{R}_1}$ şi 0 altfel.
Deoarece $X^+_{\mathcal{R}_1} \subsetneq U$, rezultă $t_1 \neq t_2$.

1) Arătăm că r_σ satisface orice dependenţă funcţională $V \to W \in \Sigma$. Presupunem contrariul, deci r_σ nu satisface $V \to W$. Atunci $V \subseteq X^+_{\mathcal{R}_1}$, căci dacă $V \not\subseteq X^+_{\mathcal{R}_1}$, există $B \in V$ şi $B \notin X^+_{\mathcal{R}_1}$, ceea ce înseamnă $t_1[B] \neq t_2[B]$, deci $t_1[V] \neq t_2[V]$, de unde rezultă r_σ satisface $V \to W$ (contradicţie).
Pe de altă parte, avem: $W \not\subseteq X^+_{\mathcal{R}_1}$, căci dacă $W \subseteq X^+_{\mathcal{R}_1}$, atunci $t_1[W] = t_2[W]$ (după definiţia lui r_σ), ceea ce ar însemna că r_σ satisface $V \to W$ (contradicţie).
Din $W \not\subseteq X^+_{\mathcal{R}_1}$ rezultă că există $A \in W$, $A \notin X^+_{\mathcal{R}_1}$. Deci $t_2[A] = 0$ şi $t_1[A] = 1$. Relaţia $V \subseteq X^+_{\mathcal{R}_1}$ conduce la $\Sigma|_{\overline{\mathcal{R}_1}} X \to V$, după lema 1.1. Dar $V \to W \in \Sigma$, aplicând tranzitivitatea rezultă $\Sigma|_{\overline{\mathcal{R}_1}} X \to W$, ceea ce înseamnă $W \subseteq X^+_{\mathcal{R}_1}$ contradicţie.

2) Arătăm că r_σ nu satisface $X \to Y$.
Presupunem că r_σ satisface $X \to Y$. Aceasta înseamnă că $t_1[X] = t_2[X]$ implică $t_1[Y] = t_2[Y]$. Deoarece $X \subseteq X^+_{\mathcal{R}_1}$ şi $t_1[A] = t_2[A], \forall A \in X^+_{\mathcal{R}_1}$, rezultă $t_1[X] = t_2[X]$, deci avem $t_1[Y] = t_2[Y]$, ceea ce înseamnă după definiţia lui r_σ că $Y \subseteq X^+_{\mathcal{R}_1}$ şi după lema 1.1, obţinem $\Sigma|_{\overline{\mathcal{R}_1}} X \to Y$, deci o contradicţie. $\square$

Teorema 1.1 (Armstrong). *Fie Σ o mulţime de dependenţe funcţionale. Atunci există o relaţie r_0 ce satisface exact elementele lui $\Sigma^+_{\mathcal{R}_1}$, adică:*
1) r_0 satisface τ, $\forall \tau \in \Sigma^+_{\mathcal{R}_1}$ şi
2) r_0 nu satisface γ, $\forall \gamma \notin \Sigma^+_{\mathcal{R}_1}$.

Demonstraţie. Deoarece U este finită, rezultă că Σ este finită şi $\mathcal{P}(U) \times \mathcal{P}(U) - \Sigma^+_{\mathcal{R}_1}$ este finită. ($\mathcal{P}(U)$ este mulţimea părţilor lui U; pentru orice dependenţă funcţională $X \to Y$, există o unică pereche $(X, Y) \in \mathcal{P}(U) \times \mathcal{P}(U)$).

Fie $\sigma_1, \sigma_2, \ldots, \sigma_k$ dependenţele care nu aparţin mulţimii $\Sigma^+_{\mathcal{R}_1}$. Pentru $\sigma \notin \Sigma^+_{\mathcal{R}_1}$ din lema anterioară am construit $r_\sigma = \begin{pmatrix} t_1 \\ t_2 \end{pmatrix}$ unde t_1 şi t_2 aveau valorile 0 şi 1.

Vom considera valori diferite pentru $\sigma_i \neq \sigma_j$ şi anume:
În r_{σ_i} vom considera valorile $2i - 2$ şi $2i - 1$, $i = \overline{1, k}$. Desigur r_{σ_i} le vom construi ca în lema precedentă:

$$r_{\sigma_i} = \begin{pmatrix} t_1^i \\ t_2^i \end{pmatrix}$$

$t_1^i[A] = 2i - 1, \ \forall A \in U$, iar

$t_2^i[A] = \begin{cases} 2i-1 & \text{pentru } A \in X^+_{i,\mathcal{R}_1} \\ 2i-2 & \text{altfel.} \end{cases}$

$(\sigma_i : X_i \to Y_i)$.

Definim r_0 ca fiind reuniunea relaţiilor r_{σ_i}, $i = \overline{1,k}$. În continuare pentru fiecare atribut A ce satisface $\Sigma|\overline{_{\mathcal{R}_1}}\emptyset \to A$ (echivalent cu $A \in \emptyset^+_{\mathcal{R}_1}$) înlocuim toate valorile din coloana atributului A în r_0 cu o aceeaşi valoare, notată v_A. Vom lua pentru astfel de atribute A şi A', $v_A \neq v_{A'}$ şi $v_A \geq 2k$.

Vom arăta că r_0, astfel construit, este o relaţie Armstrong pentru Σ.

a) Arătăm că r_0 satisface orice $X \to Y \in \Sigma$

Fie $A \in Y$ oarecare. Este suficient să arătăm că r_0 satisface $X \to A$ (aplicând uniunea va rezulta că r_0 satisface $X \to Y$). Distingem două cazuri:

I $X \subseteq \emptyset^+_{\mathcal{R}_1}$. Avem $\Sigma|\overline{_{\mathcal{R}_1}}\emptyset \to X$ şi împreună cu $X \to Y \in \Sigma$, obţinem $\Sigma|\overline{_{\mathcal{R}_1}}\emptyset \to Y$ (prin tranzitivitate), ceea ce înseamnă $Y \subseteq \emptyset^+_{\mathcal{R}_1}$ (lema 1.1). $A \in Y$ şi $Y \subseteq \emptyset^+_{\mathcal{R}_1}$ implică $A \in \emptyset^+_{\mathcal{R}_1}$ şi din construcţia lui r_0 toate valorile în coloana A sunt v_A, ceea ce denotă faptul că r_0 satisface $X \to A$.

II $X \not\subseteq \emptyset^+_{\mathcal{R}_1}$. Rezultă că $\exists B \in X$, $B \notin \emptyset^+_{\mathcal{R}_1}$. Fie $t_1, t_2 \in r_0$ cu proprietatea $t_1[X] = t_2[X]$. Trebuie să arătăm că $t_1[A] = t_2[A]$. Deoarece $B \in X$ şi $B \notin \emptyset^+_{\mathcal{R}_1}$, rezultă că în această situaţie coloana lui B conţine 2k valori distincte, deci t_1 şi t_2 nu conţin aceleaşi valori. Cum $t_1[X] = t_2[X]$, rezultă că t_1 şi t_2 provin din aceeaşi relaţie r_{σ_i}. (Eventual anumite coloane A din t_1 şi t_2 au suferit modificarea cu valori v_A). După lema anterioară t_1 şi t_2 iniţiale din r_{σ_i} au satisfăcut Σ, deci şi $X \to A$. Noile t_1 şi t_2 obţinute prin identificarea valorilor din anumite coloane, vor satisface, de asemenea, $X \to A$, deci $t_1[A] = t_2[A]$.

b) Arătăm că r_0 nu satisface σ, $\forall \sigma \notin \Sigma^+_{\mathcal{R}_1}$ (echivalent cu r_0 nu satisface σ_i, $i = \overline{1,k}$).

După lema anterioară ştim că r_{σ_i} nu satisface σ_i.

$r_{\sigma_i} = \begin{pmatrix} t_1^i \\ t_2^i \end{pmatrix}$

$t_1^i[A] = 2i-1$, $\forall A \in U$

$t_2^i[A] = \begin{cases} 2i-1 \text{ dacă } A \in X^+_{i,\mathcal{R}_1} \text{ şi} \\ 2i-2 \text{ altfel.} \end{cases}$

$\sigma_i : X_i \to Y_i$

Deoarece $X_i \subseteq X^+_{i,\mathcal{R}_1}$ înseamnă că $\exists B_i \in Y_i$, $B_i \notin X^+_{i,\mathcal{R}_1}$, astfel încât $t^i_1[B_i] \neq t^i_2[B_i]$. Vom arăta că nu toate coloanele de acest tip B_i din Y suferă identificarea datorată apartenenţei atributului respectiv la $\emptyset^+_{\mathcal{R}_1}$.

Presupunem că pentru orice $B_i \in Y_i$, $B_i \notin X^+_{i,\mathcal{R}_1}$ avem $B_i \in \emptyset^+_{\mathcal{R}_1}$. Atunci $\Sigma|\overline{_{\mathcal{R}_1}}\emptyset \to B_i$, aplicând uniunea obţinem: $\Sigma|\overline{_{\mathcal{R}_1}}\emptyset \to Y_i - X^+_{i,\mathcal{R}_1}$. Aceasta împreună cu $\Sigma|\overline{_{\mathcal{R}_1}}X_i \to \emptyset$ şi tranzitivitatea ne dau $\Sigma|\overline{_{\mathcal{R}_1}}X_i \to Y_i - X^+_{i,\mathcal{R}_1}$. Dar avem $\Sigma|\overline{_{\mathcal{R}_1}}X_i \to X^+_{i,\mathcal{R}_1}$, aplicând uniunea pentru toate A din $X^+_{i,\mathcal{R}_1} = \{A|\Sigma|\overline{_{\mathcal{R}_1}}X_i \to A\}$. Aplicând încă o dată uniunea obţinem $\Sigma|\overline{_{\mathcal{R}_1}}X_i \to Y_i$, adică $\Sigma|\overline{_{\mathcal{R}_1}}\sigma_i$, contradicţie. Rezultă că în urma identificării $\exists B_i \in Y_i$, astfel încât $t^i_1[B_i] \neq t^i_2[B_i]$. Având $t^i_1[X_i] = t^i_2[X_i]$, rezultă că r_0 nu satisface σ_i. $\square$

2 Studiul dependenţelor funcţionale utilizând calculul propoziţional

Acest studiu a fost realizat de Fagin în [24]. Pentru fiecare atribut $A \in U$ se asociază o variabilă propoziţională notată a. Corespunzător dependenţei funcţionale $A_1 \ldots A_m \to B_1 \ldots B_p$ asociem implicaţia logică $a_1 \ldots a_m \Rightarrow b_1 \ldots b_p$, unde $a_1 \ldots a_m$ înseamnă conjuncţia logică a variabilelor $a_1, a_2, \ldots, a_m$; similar $b_1 \ldots b_p$. Semnul "$\Rightarrow$" reprezintă implicaţia logică.

Vom nota valorile de adevăr din calculul propoziţional prin 1 (adevărat) şi 0 (fals). O asignare o notăm prin δ şi este o funcţie $\delta : Var \to \{0, 1\}$, unde Var este mulţimea variabilelor propoziţionale. Funcţia δ se extinde la formule în general (şi la implicaţii în particular) prin

$\delta(p_1 \wedge p_2) = \delta(p_1) \wedge \delta(p_2)$, p_1 şi p_2 fiind formule,
$\delta(p \vee p_2) = \delta(p_1) \vee \delta(p_2)$,
$\delta(\neg p) = \neg\delta(p)$,

unde conjuncţia, disjuncţia şi negaţia în $\{0, 1\}$ sunt definite în modul cunoscut.

Rezultă că $\delta(a_1 \ldots a_m \Rightarrow b_1 \ldots b_p) = 1$ dacă şi numai dacă

1. $\exists i,\ 1 \leq i \leq m$, astfel încât $\delta(a_i) = 0$ sau
2. $\forall i,\ i = \overline{1, m}$, $\delta(a_i) = 1$ şi $\forall j,\ j = \overline{1, p},\ \delta(b_j) = 1$.

În calculul propoziţional există noţiunea de consecinţă logică a unei formule α dintr-o mulţime de formule F. Se notează prin $F \mathrel{|\overline{\underline{\ \text{c.l.}\ }}} \alpha$, dacă $(\forall\delta)[(\forall f \in F,\ \delta(f) = 1) \rightsquigarrow \delta(\alpha) = 1]$; δ notează o asignare a variabilelor propoziţionale.

Pentru o dependenţă σ vom nota prin $\overline{\sigma}$ implicaţia ataşată, iar pentru o mulţime de dependenţe funcţionale Σ, vom nota prin $\overline{\Sigma} = \{\overline{\sigma} | \sigma \in \Sigma\}$.

Exemplul 2.1 *Fie* $\Sigma = \{AB \to C,\ AC \to D,\ BD \to E\}$. *Atunci* $\overline{\Sigma} = \{ab \Rightarrow c,\ ac \Rightarrow d,\ bd \Rightarrow e\}$.

Vom stabili o legătură între noţiunea de "consecinţă" definită în domeniul dependenţelor funcţionale şi noţiunea de "consecinţă logică" definită în domeniul calculului propoziţional.

Fie $\sigma : AB \to E$ şi Σ din exemplul anterior. Se pune întrebarea dacă $\Sigma \models \sigma$? Pentru aceasta, fie r o relaţie ce satisface toate elementele lui Σ şi fie $t_1, t_2 \in r$ astfel încât $t_1[AB] = t_2[AB]$. Din r satisface $AB \to C$ rezultă $t_1[C] = t_2[C]$, deci $t_1[AC] = t_2[AC]$. Din faptul că r satisface $AC \to D$ rezultă $t_1[D] = t_2[D]$, deci $t_1[BD] = t_2[BD]$. Din faptul că r satisface $BD \to E$ rezultă că $t_1[E] = t_2[E]$, deci r satisface $AB \to E$. În concluzie $\Sigma \models AB \to E$.

Vrem să vedem acum dacă $\overline{\Sigma} \models_{\overline{\text{c.l.}}} \overline{\sigma}$? Pentru aceasta vom putea proceda astfel: construim tabela cu cele 2^5 asignări ale variabilelor propoziţionale a, b, c, d, e şi vom calcula valoarea de asignare pentru toate elementele lui $\overline{\Sigma}$ şi pentru $\overline{\sigma}$ şi dacă în fiecare caz, în care toate elementele lui $\overline{\Sigma}$ sunt 1, rezultă şi $\overline{\sigma}$ este 1, atunci vom avea $\overline{\Sigma} \models_{\overline{\text{c.l.}}} \overline{\sigma}$. Dar putem reduce acest calcul considerând asignările δ astfel:

1. Dacă $\delta(a) = 0$ sau $\delta(b) = 0$, atunci $\delta(\overline{\sigma}) = 1$.

2. Dacă $\delta(a) = \delta(b) = 1$, atunci deoarece δ trebuie să satisfacă $ab \Rightarrow c$, rezultă $\delta(c) = 1$. Cum δ trebuie să satisfacă $ac \Rightarrow d$, rezultă $\delta(d) = 1$ şi la fel δ trebuie să satisfacă $bd \Rightarrow e$, rezultă $\delta(e) = 1$, deci $\delta(ab \Rightarrow e) = 1$, adică $\delta(\overline{\sigma}) = 1$.

Vom stabili echivalenţa între "consecinţă" din universul dependenţelor funcţionale şi "consecinţă logică" din calculul propoziţional.

Teorema 2.1 (de echivalenţă). *Fie Σ o mulţime de dependenţe funcţionale şi σ o dependenţă funcţională. Fie $\overline{\Sigma}$ mulţimea de implicaţii corespunzătoare lui Σ şi $\overline{\sigma}$ implicaţia asociată lui σ. Atunci avem: σ este consecinţă a lui Σ dacă şi numai dacă $\overline{\sigma}$ este consecinţă logică a lui $\overline{\Sigma}$ ($\Sigma \models \sigma \ \ iff \ \ \overline{\Sigma} \models_{\overline{c.l.}} \overline{\sigma}$).*[1]

[1] iff = if and only if (engl.) dacă şi numai dacă

Conform acestei teoreme problema deciderii dacă o dependenţă funcţională este consecinţă a unei mulţimi de dependenţe se transformă într-o problemă de a decide dacă o implicaţie este consecinţă logică a unei mulţimi de implicaţii.

Pentru această ultimă problemă de decizie există un algoritm eficient, datorat lui Chang.

După propoziţiile 1.1, 1.2, putem presupune că Σ are dependenţele cu un singur atribut în partea dreaptă şi la fel σ are în partea dreaptă un singur atribut. **Algoritmul lui Chang:**

1. Se consideră cuvinte formate peste $\overline{U} \cup \{\sim, *\}$ astfel:

   pentru $a_1 \dots a_m \Rightarrow b \in \overline{\Sigma}$ se consideră cuvântul

   $$\sim a_1 * \sim a_2 * \dots \sim a_m * b$$

   Dacă $U = \{A_1, \dots, A_n\}$, atunci $\overline{U} = \{a_1, \dots, a_n\}$.

   Să notăm prin $\mathcal{S}$ mulţimea de cuvinte obţinute.

2. Dacă $\overline{\sigma}$ are forma: $c_1 \dots c_k \Rightarrow d$, atunci adăugăm la $\mathcal{S}$ următoarele $k+1$ cuvinte:

   c_1
   c_2
   $\vdots$
   c_k
   $\sim d$

   Numim o variabilă a_i "atom" şi $\sim a_i$ "atom negat".

3. Algoritmul caută un atom X, astfel încât X este un cuvânt în $\mathcal{S}$ şi există un cuvânt ce începe cu $\sim X$. Dacă există un astfel de atom, se selectează unul arbitrar şi se şterge $\sim X$ din fiecare şir ce începe cu $\sim X$ (eventual şi $*$ dacă acest caracter $*$ urmează în şir).

4. Se continuă pasul 3) atât timp cât există un astfel de atom. Este clar că algoritmul se opreşte întotdeauna şi există 2 situaţii:

   (a) s-a obţinut şirul vid, notat cu λ sau

   (b) nu există nici un atom X ce satisface cele 2 condiţii şi nu s-a obţinut şirul vid.

Dacă s-a obţinut (a) atunci $\overline{\sigma}$ este consecinţă logică a lui $\overline{\Sigma}$, altfel $\overline{\sigma}$ nu este consecinţă logică a lui $\overline{\Sigma}$.

Exemplul 2.2 *Fie* $\overline{\Sigma} = \{ab \Rightarrow c,\ ac \Rightarrow d,\ bd \Rightarrow e\}$ *şi* $\overline{\sigma} : ab \Rightarrow e$. *Atunci* $\mathcal{S} = \{\sim a* \sim b * c,\ \ \sim a* \sim c * d,\ \ \sim b* \sim d * e,\ a,\ b,\ \sim e\}$. *Putem alege* $X = a$, *atunci* $\mathcal{S}$ *devine*

$$\mathcal{S} = \{\sim b * c,\ \ \sim c * d,\ \ \sim b* \sim d * e,\ a,\ b,\ \sim e\}$$

În continuare putem lua $X = b$ *şi* $\mathcal{S}$ *devine*

$$\mathcal{S} = \{c,\ \ \sim c * d,\ \ \sim d * e,\ a,\ b,\ \sim e\}$$

Fie acum $X = c$, $\mathcal{S}$ *devine:*

$$\mathcal{S} = \{c,\ d, \sim d * e,\ a,\ b,\ \sim e\}$$

Luând acum $X = d$ *obţinem:*

$$\mathcal{S} = \{c,\ d,\ e,\ a,\ b, \sim e\},$$

Pentru $X = e$, *rezultă:*

$$S = \{c,\ d,\ e,\ a,\ b, \lambda\},\ \ deci\ \overline{\Sigma} \models_{\overline{c.l.}} \overline{\sigma}$$

Teorema 2.2 (de completitudine a dependenţelor). *Fie* Σ *o mulţime de dependenţe funcţionale,* σ *o dependenţă funcţională. Atunci avem* σ *este o consecinţă a lui* Σ *dacă şi numai dacă* σ *are o demonstraţie în* Σ *utilizând regulile de inferenţă* A_1, A_2, A_3 *(axiomele lui Armstrong).*

În notaţii, teorema se exprimă prin:

$$\Sigma \models \sigma\ \ iff\ \ \Sigma \vdash_{\overline{\mathcal{R}_A}} \sigma.$$

Demonstraţie. Regulile de inferenţă A_1, A_2, A_3 au forma:

(A1) $\dfrac{}{A_1 \dots A_m \to A_i},\ 1 \le i \le m$

(A2) $\dfrac{A_1 \dots A_m \to B_1 \dots B_r}{A_1 \dots A_m \to B_j}, j = \overline{1, r},\ \dfrac{A_1 \dots A_m \to B_j,\ j = \overline{1, r}}{A_1 \dots A_m \to B_1 \dots B_r}$

$$\text{(A3)}\ \frac{A_1 \dots A_m \to B_1 \dots B_r,\ B_1 \dots B_r \to C_1 \dots C_p}{A_1 \dots A_m \to C_1 \dots C_p}$$

Fapt 1: Dacă $\mathcal{R}$ este un sistem de reguli de inferenţă valide, şi dacă $\Sigma|\!\frac{\ \ }{\mathcal{R}}\sigma$ atunci $\Sigma \models \sigma$. O regulă $\frac{\alpha_1, \alpha_2, \dots, \alpha_k}{\alpha}$ se numeşte validă dacă$\{\alpha_1, \dots, \alpha_k\} \models \alpha$.

Justificarea faptului rezultă prin inducţie asupra lungimii demonstraţiei. Fie $\sigma_1, \sigma_2, \dots, \sigma_h = \sigma$ o demonstraţie pentru σ în Σ utilizând regulile $\mathcal{R}$. Dacă $h = 1$, atunci $\sigma_1 = \sigma$ poate fi în una din situaţiile:

a) $\sigma_1 = \sigma \in \Sigma$ şi deci $\Sigma \models \sigma$, sau

b) există o regulă de forma $\frac{}{\sigma} \in \mathcal{R}$; regula fiind validă, rezultă σ dependenţă trivială (deci satisfăcută de orice relaţie), de unde $\Sigma \models \sigma$.

Presupunem Fapt 1 adevărat pentru demonstraţii de lungime mai mică sau egală cu h. Fie o demonstraţie a lui σ de lungime $h+1 : \sigma_1, \sigma_2, \dots, \sigma_h, \sigma_{h+1} = \sigma$. Pentru σ avem 2 situaţii:

c) $\sigma \in \Sigma$, deci $\Sigma \models \sigma$ sau

d) $\exists \frac{\sigma_{i_1}, \dots, \sigma_{i_k}}{\sigma} \in \mathcal{R},\ i_1, \dots, i_k \leq h$.

În cazul d) după ipoteza inducţiei avem: $\Sigma \models \sigma_{i_1}, \dots, \Sigma \models \sigma_{i_k}$ şi deoarece regula este validă, avem $\{\sigma_{i_1}, \dots, \sigma_{i_k}\} \models \sigma$. Rezultă atunci $\Sigma \models \sigma$.

Regulile (A1) (A2) (A3) sunt valide, din proprietăţile dependenţelor funcţionale. Deci putem aplica Fapt 1, ceea ce implică:

$$\text{Dacă } \Sigma|\!\frac{\ \ }{\mathcal{R}_A}\sigma, \text{ atunci } \Sigma \models \sigma.$$

Invers: presupunem $\Sigma \models \sigma$. Considerăm $\Sigma^+_{\mathcal{R}_A}$ (închiderea lui Σ referitoare la regulile de inferenţă $\mathcal{R}_A$). Amintim că $\Sigma^+_{\mathcal{R}_A} = \{\sigma_1 | \Sigma|\!\frac{\ \ }{\mathcal{R}_A}\sigma_1\}$. Trebuie să arătăm că $\sigma \in \Sigma^+_{\mathcal{R}_A}$.

Avem: $\Sigma \subseteq \Sigma^+_{\mathcal{R}_A}$. După teorema lui Armstrong (Teorema 1.1) există o relaţie r_0 ce satisface exact elementele din $\Sigma^+_{\mathcal{R}_1}$, unde $\mathcal{R}_1 = \{\text{FD1f, FD2f, FD3f}\}$. După consecinţa 1.1, avem $\Sigma^+_{\mathcal{R}_1} = \Sigma^+_{\mathcal{R}_A}$. Deci relaţia r_0 satisface exact elementele lui $\Sigma^+_{\mathcal{R}_A}$. Deoarece $\Sigma \subseteq \Sigma^+_{\mathcal{R}_A}$, rezultă că r_0 satisface toate dependenţele din Σ. Având $\Sigma \models \sigma$ obţinem că r_0 satisface σ. Deoarece r_0 satisface exact elementele lui $\Sigma^+_{\mathcal{R}_A}$, obţinem că $\sigma \in \Sigma^+_{\mathcal{R}_A}$, deci $\Sigma|\!\frac{\ \ }{\mathcal{R}_A}\sigma$. $\square$

Considerăm în continuare o mulţime de reguli de inferenţă pentru formule ale calculului propoziţional (similare cu regulile A1,A2, A3).

(A1′) $\dfrac{}{a_1 \ldots a_m \Rightarrow a_i}$, $1 \le i \le m$

(A2′) $\dfrac{a_1 \ldots a_m \Rightarrow b_1 \ldots b_r}{a_1 \ldots a_m \Rightarrow b_j}$, $j = \overline{1,r}$, $\dfrac{a_1 \ldots a_m \Rightarrow b_j,\ j = \overline{1,r}}{a_1 \ldots a_m \Rightarrow b_1 \ldots b_r}$

(A3′) $\dfrac{a_1 \ldots a_m \Rightarrow b_1 \ldots b_r,\ b_1 \ldots b_r \Rightarrow c_1 \ldots c_p}{a_1 \ldots a_m \Rightarrow c_1 \ldots c_p}$

Teorema 2.3 (de completitudine implicaţională). *Fie $\overline{\Sigma}$ o mulţime de implicaţii asociate dependenţelor funcţionale din Σ, $\overline{\sigma}$ implicaţia asociată dependenţei funcţionale σ. Atunci $\overline{\sigma}$ este consecinţă logică a lui $\overline{\Sigma}$ dacă şi numai dacă $\overline{\sigma}$ are o demonstraţie în $\overline{\Sigma}$, utilizând regulile de inferenţă (A1′), (A2′), (A3′).*

Demonstraţie. Regulile (A1′), (A2′), (A3′) considerate ca reguli în calculul propoziţional sunt valide, adică orice asignare care face adevărate premizele regulii, face adevărată şi concluzia regulii. Procedând ca în demonstraţia teoremei de completitudine a dependenţelor obţinem că dacă $\overline{\Sigma}\vert\!\underline{\quad}_{\mathrm{A1',A2',A3'}}\overline{\sigma}$, atunci $\overline{\Sigma} \models_{\overline{\mathrm{c.l.}}} \overline{\sigma}$.

Invers, fie $\overline{\Sigma} \models_{\overline{\mathrm{c.l.}}} \overline{\sigma}$. Va trebui să arătăm că $\overline{\Sigma}\vert\!\underline{\quad}_{\mathrm{A1',A2',A3'}}\overline{\sigma}$. Fie $\overline{\sigma} : a_1 \ldots a_m \Rightarrow d_1 \ldots d_h$. Să notăm prin:

$$PROVE = \{e \vert \overline{\Sigma}\vert\!\underline{\quad}_{\mathrm{A1',A2',A3'}} a_1 \ldots a_m \Rightarrow e\}$$

Aplicând regula (A1′) obţinem $a_i \in PROVE$, $1 \le i \le m$. Dorim să arătăm că $d_j \in PROVE$, pentru orice $j = \overline{1,h}$. Presupunem contrariul, deci $\exists j \in \{1, 2, \ldots, h\}$, astfel încât $d_j \notin PROVE$, adică $\overline{\Sigma}\vert\!\not\underline{\quad}_{\mathrm{A1',A2',A3'}} a_1 \ldots a_m \Rightarrow d_j$. Considerăm următoarea asignare:

$$\delta_0(x) = \begin{cases} 1 & \text{dacă } x \in PROVE \\ 0 & \text{altfel} \end{cases}$$

Deoarece $a_i \in PROVE$, rezultă că $\delta_0(a_i) = 1$, orice $i = \overline{1,m}$. $d_j \notin PROVE$ implică $\delta_0(d_j) = 0$, deci $\delta_0(\overline{\sigma}) = 0$. Arătăm că δ_0 satisface toate elementele din $\overline{\Sigma}$. Fie $b_1 \ldots b_r \Rightarrow c_1 \ldots c_p \in \overline{\Sigma}$. Se disting două situaţii:

1. $b_i \in PROVE$ pentru $i = \overline{1,r}$; aceasta înseamnă că

$$\overline{\Sigma}\vert\!\underline{\quad}_{\mathrm{A1',A2',A3'}} a_1 \ldots a_m \Rightarrow b_i,\ i = \overline{1,r}$$

Aplicând (A2$'$) partea a II-a obţinem:

$$\overline{\Sigma}|\overline{_{\text{A1}',\text{A2}',\text{A3}'}} a_1 \dots a_m \Rightarrow b_1 \dots b_r.$$

Această relaţie împreună cu $b_1 \dots b_r \Rightarrow c_1 \dots c_p \in \overline{\Sigma}$ şi (A3$'$) conduc la $\overline{\Sigma}|\overline{_{\text{A1}',\text{A2}',\text{A3}'}} a_1 \dots a_m \Rightarrow c_1 \dots c_p$. De aici, aplicând (A2$'$) partea a II-a obţinem:

$$\overline{\Sigma}|\overline{_{\text{A1}',\text{A2}',\text{A3}'}} a_1 \dots a_m \Rightarrow c_k, \quad 1 \leq k \leq p,$$

ceea ce înseamnă că $c_k \in PROVE$, orice k, $1 \leq k \leq p$ şi după definiţia lui δ_0 obţinem $\delta_0(c_k) = 1$, $\forall k$, $1 \leq k \leq p$, deci $\delta_0(c_1 \dots c_p) = 1$, ceea ce înseamnă $\delta_0(b_1 \dots b_r \Rightarrow c_1 \dots c_p) = 1$.

2. Există $i \in \{1, 2, \dots, r\}$, astfel încât $b_i \notin PROVE$. Aceasta înseamnă că $\delta_0(b_i) = 0$, deci $\delta_0(b_1 \dots b_r) = 0$, de unde $\delta_0(b_1 \dots b_r \Rightarrow c_1 \dots c_p) = 1$. Asignarea δ_0 construită mai sus satisface condiţia $\delta_0(\overline{\gamma}) = 1$, orice $\overline{\gamma} \in \overline{\Sigma}$, dar $\delta_0(\overline{\sigma}) = 0$. Acest lucru contrazice ipoteza $\overline{\Sigma} \models_{\overline{\text{c.l.}}} \overline{\sigma}$. Deci toate variabilele $d_j \in PROVE$, $j = \overline{1, h}$. De aici:

$$\overline{\Sigma}|\overline{_{\text{A1}',\text{A2}',\text{A3}'}} a_1 \dots a_m \Rightarrow d_j, \quad j = \overline{1, h}$$

şi aplicând A2$'$ se obţine $\overline{\Sigma}|\overline{_{\text{A1}',\text{A2}',\text{A3}'}} a_1 \dots a_m \Rightarrow d_1 \dots d_h$, adică $\overline{\Sigma}|\overline{_{\text{A1}',\text{A2}',\text{A3}'}} \overline{\sigma}$.
□

Suntem acum în măsură să demonstrăm teorema 2.1 (de echivalenţă): Fie Σ o mulţime de dependenţe şi σ o dependenţă funcţională. Fie $\overline{\Sigma}$ mulţimea implicaţiilor corespunzătoare lui Σ şi $\overline{\sigma}$ implicaţia corespunzătoare lui σ. Atunci: σ este consecinţă a lui Σ dacă şi numai dacă $\overline{\sigma}$ este consecinţă logică a lui $\overline{\Sigma}$.

Demonstraţie. După teorema de completitudine a dependenţelor avem: $\Sigma \models \sigma \;\; iff \;\; \Sigma|\overline{_{\mathcal{R}_A}} \sigma$.

Orice demonstraţie a lui σ în Σ, utilizând regulile $\mathcal{R}_A$ se poate transforma sintactic într-o demonstraţie a lui $\overline{\sigma}$ în $\overline{\Sigma}$ utilizând regulile A'_1, A'_2, A'_3 înlocuind atributele prin variabilele propoziţionale respective, semnul " $\rightarrow$ " prin " $\Rightarrow$ " şi invers. Deci avem:

$\Sigma|\overline{_{\mathcal{R}_A}} \sigma \;\; iff \;\; \overline{\Sigma}|\overline{_{A'_1,A'_2,A'_3}} \overline{\sigma}$. După teorema de completitudine implicaţională avem:

$\overline{\Sigma}|\overline{_{A'_1,A'_2,A'_3}} \overline{\sigma} \;\; iff \;\; \overline{\Sigma} \models_{\overline{\text{c.l.}}} \overline{\sigma}$. Astfel avem:

$\Sigma \models \sigma \;\; iff \;\; \overline{\Sigma} \models_{\overline{\text{c.l.}}} \overline{\sigma}.$ □

În continuare vom da o demonstraţie semantică a teoremei de echivalenţă, utilizând noţiunea de consecinţă pe mulţimea relaţiilor cu 2 uple.

Spunem că σ este consecinţă a lui Σ pe mulţimea relaţiilor cu 2 uple şi vom nota $\Sigma \models_{\overline{\text{2-uple}}} \sigma$, dacă orice relaţie cu 2 uple ce satisface toate elementele lui Σ, va satisface, de asemenea, şi pe σ.

Vom da întâi o lemă, numită semantică, ce leagă relaţiile cu 2 uple şi asignările:

Lema 2.1 *Fie δ o asignare şi $r = \begin{pmatrix} t_1 \\ t_2 \end{pmatrix}$ o relaţie cu două uple, astfel încât pentru orice atribut $A \in U$ avem: $t_1[A] = t_2[A] \;\; iff \;\; \delta(a) = 1$. (a este variabila propoziţională corespunzătoare atributului A). Atunci avem:*

r satisface σ iff $\delta(\overline{\sigma}) = 1$, pentru orice σ dependenţă funcţională.

Demonstraţie. Presupunem că r satisface $\sigma : A_1 \ldots A_m \to B_1 \ldots B_r$, să arătăm că $\delta(\overline{\sigma}) = 1$. Distingem următoarele cazuri:

I) $t_1[A_1 \ldots A_m] = t_2[A_1 \ldots A_m]$. Rezultă atunci $t_1[B_1 \ldots B_r]$ $= t_2[B_1 \ldots B_r]$, deci $t_1[B_j] = t_2[B_j]$, pentru toţi $j = \overline{1,r}$, de unde $\delta(b_j) = 1, j = \overline{1,r}$, deci $\delta(b_1 \ldots b_r) = 1$, ceea ce implică $\delta(a_1 \ldots a_m \Rightarrow b_1 \ldots b_r) = 1$.

II) $\exists i,\; i \in \{1 \ldots m\},\; t_1[A_i] \neq t_2[A_i]$. Atunci $\delta(a_i) = 0$, de unde $\delta(a_1 \ldots a_m \Rightarrow b_1 \ldots b_r) = 1$.

Afirmaţia inversă rezultă similar. □

Teorema 2.4 (teorema de echivalenţă semantică).

Următoarele afirmaţii sunt echivalente:

1. σ este o consecinţă a lui Σ.

2. $\overline{\sigma}$ este o consecinţă logică a lui $\overline{\Sigma}$.

3. σ este o consecinţă a lui Σ pe mulţimea relaţiilor cu 2 uple.

Demonstraţie. Este clar că 1 implică 3. Să arătăm că 3 implică 1. Presupunem contrariul, deci σ este o consecinţă a lui Σ pe mulţimea relaţiilor cu 2 uple şi σ nu este consecinţă a lui Σ. Atunci există o relaţie r ce satisface toate elementele lui Σ şi nu satisface σ. Fie $\sigma : X \to Y$, atunci există 2 uple $t_1, t_2 \in r$, astfel încât $t_1[X] = t_2[X]$ şi $t_1[Y] \neq t_2[Y]$. Atunci

$r_1 = \begin{pmatrix} t_1 \\ t_2 \end{pmatrix}$ satisface Σ, deoarece r_1 este o subrelaţie a lui r ce satisface Σ. Avem r_1 nu satisface σ. Aceasta înseamnă că σ nu este consecinţă a lui Σ pe mulţimea relaţiilor cu 2 uple, deci contradicţie. Am arătat deci că 1 şi 3 sunt echivalente. Arătăm acum că 2 şi 3 sunt echivalente.

Să justificăm întâi că 3 implică 2. Presupunem contrariul. Deci σ este o consecinţă a lui Σ pe mulţimea relaţiilor cu 2 uple şi $\overline{\sigma}$ nu este consecinţă logică a lui $\overline{\Sigma}$. Atunci există o asignare δ, care satisface toate elementele lui $\overline{\Sigma}$ şi δ nu satisface $\overline{\sigma}$. Definim o relaţie $r_1 = \begin{pmatrix} t_1 \\ t_2 \end{pmatrix}$ cu 2 uple, unde t_1 ia valoarea 1 pentru orice atribut, iar $t_2[A] = 1$ dacă $\delta(a) = 1$, altfel $t_2[A] = 0$. Rezultă că asignarea δ şi relaţia r_1 sunt în condiţiile stabilite de lema semantică. Deoarece $\delta(\overline{\sigma}) = 0$, după lema semantică obţinem r_1 nu satisface σ. Avem $\delta(\overline{\gamma}) = 1$ pentru orice $\overline{\gamma} \in \overline{\Sigma}$. Aplicând din nou lema semantică obţinem: r_1 satisface γ, pentru orice $\gamma \in \Sigma$. Această afirmaţie împreună cu afirmaţia că r_1 nu satisface σ contrazice faptul că σ este o consecinţă a lui Σ pe mulţimea relaţiilor cu 2 uple.

Să arătăm acum că 2 implică 3. Presupunem contrariul. Avem $\overline{\sigma}$ este o consecinţă logică a lui $\overline{\Sigma}$ şi σ nu este o consecinţă a lui Σ pe mulţimea relaţiilor cu 2 uple. Rezultă atunci că există r_1 o relaţie cu 2 uple, fie acestea t_1, t_2, ce satisface toate elementele lui Σ şi nu satisface σ. Definim o asignare δ astfel:

$\delta(a) = 1$ iff $t_1[A] = t_2[A]$. Obţinem astfel că δ şi r_1 sunt în condiţiile lemei semantice. Deoarece r_1 satisface orice $\gamma \in \Sigma$, după lema semantică obţinem: $\delta(\overline{\gamma}) = 1$. Deoarece r_1 nu satisface σ, din nou aplicând lema semantică obţinem: $\delta(\overline{\sigma}) = 0$. Acestea contrazic faptul că $\overline{\sigma}$ este o consecinţă logică a lui $\overline{\Sigma}$. Astfel am arătat că 2 este echivalent cu 3. Deci 1, 2 şi 3 sunt echivalente. Echivalenţa afirmaţiilor 1 şi 2 constituie tocmai teorema de echivalenţă.

$\square$

Deoarece pentru formule din calculul propozitional ştim ce înseamnă faptul că o formulă $\overline{\alpha} \vee \overline{\beta}$ este consecinţă logică a unei mulţimi de formule $\overline{\Sigma}$ (orice asignare care face adevărate elementele lui $\overline{\Sigma}$, face adevărată formula $\overline{\alpha} \vee \overline{\beta}$, adică face adevărată pe $\overline{\alpha}$ sau pe $\overline{\beta}$) este natural să extindem noţiunea de consecinţă pentru expresii de dependenţe de forma $\alpha \vee \beta$:

$\alpha \vee \beta$ este consecinţă din Σ, notat $\Sigma \models \alpha \vee \beta$ dacă orice relaţie ce satisface toate elementele lui Σ va satisface pe α sau pe β.

Astfel, este natural să ne întrebăm dacă teorema de echivalenţă mai este valabilă în acest caz.

Întrebare: Este adevărată teorema "Fie date Σ o mulţime de dependenţe funcţionale, α şi β două dependenţe funcţionale. Atunci $\alpha \vee \beta$ este consecinţă a lui Σ iff $\overline{\alpha} \vee \overline{\beta}$ este consecinţă logică a lui $\overline{\Sigma}$ "?

Vom arăta printr-un contraexemplu că teorema este falsă:

Fie $\Sigma = \{A \to A\}$, $U = \{A, B\}$, $\alpha : A \to B$, $\beta : B \to A$. Considerăm $r = \begin{pmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \end{pmatrix}$ peste U. Avem r satisface Σ, dar r nu satisface α şi nici β. Deci $\Sigma \not\models \alpha \vee \beta$. Vom arăta că $\overline{\Sigma} \models_{\text{c.l.}} \overline{\alpha} \vee \overline{\beta}$. $\overline{\alpha} : a \Rightarrow b, \overline{\beta} : b \Rightarrow a$. Fie δ o asignare ce satisface $\overline{\Sigma} = \{a \Rightarrow a\}$, deci δ este oarecare. Dacă $\delta(a) = 0$, atunci $\delta(\overline{\alpha}) = 1$, deci $\delta(\overline{\alpha} \vee \overline{\beta}) = 1$, iar dacă $\delta(a) = 1$, atunci $\delta(\overline{\beta}) = 1$, deci $\delta(\overline{\alpha} \vee \overline{\beta}) = 1$. Cu ajutorul aceluiaşi exemplu demonstrăm teorema:

Teorema 2.5 *Există o mulţime $\overline{\Sigma}$ de implicaţii şi o pereche $\overline{\alpha}, \overline{\beta}$ de implicaţii, astfel încât avem îndeplinite:*
a) $\overline{\alpha}$ nu este consecinţă logică din $\overline{\Sigma}$ şi
b) $\overline{\beta}$ nu este consecinţă logică din $\overline{\Sigma}$ şi
c) $\overline{\alpha} \vee \overline{\beta}$ este consecinţă logică din $\overline{\Sigma}$.

Demonstraţie. Considerăm $\overline{\Sigma} = \{a \Rightarrow a\}$, $\overline{\alpha} : a \Rightarrow b$, $\overline{\beta} : b \Rightarrow a$ Am văzut că $\overline{\Sigma} \models_{\text{c.l.}} \overline{\alpha} \vee \overline{\beta}$ deci c) este îndeplinită. Pentru a) consider δ_1 asignare, astfel încât $\delta_1(a) = 1$ şi $\delta_1(b) = 0$. Atunci δ_1 satisface $\overline{\Sigma}$, dar $\delta_1(\overline{\alpha}) = 0$. Pentru b) consider δ_2 asignare, astfel încât $\delta_2(b) = 1$ şi $\delta_2(a) = 0$. Rezultă δ_2 satisface $\overline{\Sigma}$, dar $\delta_2(\overline{\beta}) = 0$. □

Spre deosebire de implicaţii, pentru dependenţe funcţionale avem următorul rezultat:

Teorema 2.6 *Este imposibil să existe o mulţime Σ de dependenţe funcţionale şi o pereche α, β de dependenţe funcţionale, astfel ca să avem:*
a) α nu este consecinţă a lui Σ şi
b) β nu este consecinţă a lui Σ şi
c) $\alpha \vee \beta$ este consecinţă a lui Σ.

Demonstraţie. Presupunem că există Σ, α şi β astfel încât a), b) şi c) să fie satisfăcute. Fie $\Sigma^+_{\mathcal{R}_1}$ închiderea lui Σ cu privire la regulile de reflexivitate, extensie şi tranzitivitate. Se ştie că $\Sigma^+_{\mathcal{R}_1} = \Sigma^+_{\mathcal{R}_A}$, unde $\mathcal{R}_A$ sunt regulile A1, A2, A3 ale lui Armstrong. După Teorema 1.1 (Armstrong) există o relaţie r_0 ce satisface exact elementele lui $\Sigma^+_{\mathcal{R}_1}$. Deoarece c) este îndeplinită, rezultă că r_0 satisface α sau r_0 satisface β. Fie r_0 satisface α. Deoarece r_0 este relaţie Armstrong pentru Σ, rezultă $\alpha \in \Sigma^+_{\mathcal{R}_1}$, deci $\Sigma \vert\!\frac{}{\mathcal{R}_1} \alpha$, ceea ce este echivalent cu $\Sigma \vert\!\frac{}{\mathcal{R}_A} \alpha$. După teorema de completitudine a dependenţelor, obţinem $\Sigma \models \alpha$, ceea ce contrazice a). Dacă r_0 satisface β, rezultă similar ca mai sus, $\Sigma \models \beta$, ceea ce contrazice b). $\square$

Vom da în continuare două aplicaţii ale teoremei de echivalenţă la teoremele Delobel-Casey [17].

Fie $\sigma : A_1 \ldots A_m \rightarrow B_1 \ldots B_r$ o dependenţă funcţională. Definim prima transformată Delobel-Casey pentru σ şi notată $f_1(\sigma)$, ca fiind funcţia booleană:

$$f_1(\sigma) = a_1 \ldots a_m b'_1 + \ldots + a_1 \ldots a_m b'_r$$

unde a_i este variabila corespunzătoare lui $A_i, 1 \leq i \leq m, b_j$ este variabila corespunzătoare lui $B_j, j = \overline{1, r}, b'_j$ este negaţia lui $b_j, +$ este suma booleană, iar $a_1 \ldots a_m b'_j$ înseamnă conjuncţia booleană între $a_1, \ldots, a_m, b'_j$.

Dacă Σ este o mulţime de dependenţe funcţionale, atunci prima transformată Delobel-Casey pentru Σ este:

$$f_1(\Sigma) = f_1(\sigma_1) + \ldots + f_1(\sigma_h), \text{ unde } \Sigma = \{\sigma, \ldots, \sigma_h\}.$$

Exemplul 2.3 *Fie* $\Sigma = \{AB \rightarrow C, BC \rightarrow D, CE \rightarrow AB\}$. $f_1(\Sigma) = abc' + bcd' + cea' + ceb'$.

Este clar că funcţia booleană $f_1(\sigma)$ ia valoarea 1 pentru asignarea δ iff $\delta(a_i) = 1, i = \overline{1, m}$ şi $\exists j, j \in \{1, \ldots, r\}$ astfel încât $\delta(b_j) = 0$.

Teorema 2.7 (Delobel-Casey) [18]. *Fie Σ_1 şi Σ_2 două mulţimi de dependenţe şi $f_1(\Sigma_1)$, respectiv $f_1(\Sigma_2)$, prima transformată asociată lui Σ_1 şi Σ_2. Atunci Σ_1 este f-echivalentă cu Σ_2 dacă şi numai dacă $f_1(\Sigma_1)$ este b-echivalentă cu $f_1(\Sigma_2)$.*

Demonstraţie. Numim Σ_1 f-echivalentă cu Σ_2 dacă $\Sigma_1^* = \Sigma_2^*$ (adică $\Sigma_1 \models \sigma \;\; iff \;\; \Sigma_2 \models \sigma$).

Funcţiile booleene f şi g sunt b-echivalente, dacă pentru orice asignare δ avem $\delta(f) = 1 \;\; iff \;\; \delta(g) = 1$. Aceasta înseamnă că f şi g sunt egale ca funcţii. Notăm f-echivalenţa între Σ_1 şi Σ_2 prin $\Sigma_1 \underset{f}{\equiv} \Sigma_2$,iar b-echivalenţa între f şi g prin $f = g$.

Fie $\Sigma = \{\sigma_1 \dots \sigma_h\}$. Arătăm că formula $\overline{\sigma_1} \dots \overline{\sigma_h}$ (conjuncţia implicaţiilor $\overline{\sigma_1}, \dots, \overline{\sigma_h}$) este echivalentă cu $f_1(\Sigma)'$. Aceasta înseamnă că pentru orice asignare δ, avem:

$$[\delta(\overline{\sigma_i}) = 1 \text{ orice } i = \overline{1,h}] \;\; iff \;\; \delta(f_1(\Sigma)) = 0$$

În adevăr, $\delta(f_1(\Sigma)) = 0 \;\; iff \;\; \delta(f_1(\sigma_i)) = 0,\ i = \overline{1,h}$
$\delta(f_1(\sigma_i)) = 0 \;\; iff \;\; \delta(a_1 \dots a_m b_j') = 0, j = \overline{1,r}$, unde $\sigma_i : A_1 \dots A_m \to B_1 \dots B_r$.

Dacă $\exists k,\ k \in \{1 \dots m\}$, astfel încât $\delta(a_k) = 0$, atunci $\delta(a_1 \dots a_m b_j') = 0, j = \overline{1,r}$ şi $\delta(\overline{\sigma_i}) = 1$, deci $\delta(f_1(\sigma_i)) = 0 \;\; iff \;\; \delta(\overline{\sigma_i}) = 1$($i$ oarecare, $i = \overline{1,h}$), de unde $\delta(f_1(\Sigma)) = 0 \;\; iff \;\; [\delta(\overline{\sigma_i}) = 1,$ orice $i = \overline{1,h}]$.

Dacă $\delta(a_k) = 1$, pentru $\forall k = \overline{1,m}$ şi fie $\delta(a_1 \dots a_m b_j') = 0, j = \overline{1,r}$, atunci $\delta(b_j') = 0, j = \overline{1,r}$, deci $\delta(b_j) = 1, j = \overline{1,r}, \delta(b_1 \dots b_r) = 1$, adică $\delta(\overline{\sigma_i}) = 1$. Obţinem, deci, şi în acest caz:

$$\delta(f_1(\sigma_i)) = 0 \;\; iff \;\; \delta(\overline{\sigma_i}) = 1.$$

Fie acum $\Sigma_1 \underset{f}{\equiv} \Sigma_2$. Rezultă $\Sigma_1 \models \sigma$ iff $\Sigma_2 \models \sigma$. De aici obţinem că $\overline{\sigma_1} \dots \overline{\sigma_h}$ este echivalentă cu $\overline{\beta_1} \dots \overline{\beta_t}$, unde

$$\Sigma_1 = \{\sigma_1 \dots \sigma_h\}, \text{ iar } \Sigma_2 = \{\beta_1 \dots \beta_t\}.$$

Dar formula $\overline{\sigma_1} \dots \overline{\sigma_h}$ este echivalentă cu $f_1(\Sigma_1)'$, iar $\overline{\beta_1} \dots \overline{\beta_t}$ este echivalentă cu $f_1(\Sigma_2)'$. Deci $f_1(\Sigma_1)' = f_1(\Sigma_2)'$, de unde $f_1(\Sigma_1) = f_1(\Sigma_2)$. Dacă $f_1(\Sigma_1) = f_1(\Sigma_2)$, raţionând în ordine inversă obţinem $\Sigma_1 \underset{f}{\equiv} \Sigma_2$. □

Exemplul 2.4 *Considerăm U formată din atributele Profesor (notat pe scurt P), Oră (H), An (Y), Sală (N), Denumirea cursului (T), având dependenţele funcţionale* $P \to T$, $PH \to Y$, $PH \to N$, $HN \to P$, $HN \to Y$, $HY \to P$, $HY \to N$. *Fie* Σ_1 *mulţimea acestor dependenţe. Funcţia booleană* $f_1(\Sigma_1)$ *este:* $f_1(\Sigma_1) = pt' + phy' + phn' + hnp' + hny' + hyp' + hyn'$. *Utilizând hărţile Karnaugh se poate arăta că această funcţie este echivalentă cu funcţia g care are expresia* $g = pt' + hyt' + hytn' + pthn' + nhty' + nythp' + nht'$. *Expresia g corespunde la prima transformată Delobel-Casey pentru* *mulţimea Σ_2, unde $\Sigma_2 = \{P \to T,\ HY \to T,\ HYT \to N,\ PTH \to N,\ NHT \to Y,\ NYTH \to P,\ NH \to T\}$. Deci $g = f_1(\Sigma_2)$. Deoarece $f_1(\Sigma_1) = f_1(\Sigma_2)$, după teorema 2.7 obţinem că $\Sigma_1 \underset{f}{\equiv} \Sigma_2$, adică Σ_1 şi Σ_2 sunt f-echivalente, ceea ce se exprimă şi prin $\Sigma_1^* = \Sigma_2^*$. Rezultă de aici o metodă pentru obţinerea acoperirilor unei mulţimi de dependenţe funcţionale. Reamintim că Σ_2 se numeşte acoperire pentru Σ_1 dacă $\Sigma_2^* = \Sigma_1^*$.*

Vom trece acum la cea de a doua teoremă Delobel-Casey.

Definiţia 2.1 *Mulţimea de atribute $K \subseteq U$ se numeşte supracheie pentru Σ, dacă $\Sigma \models K \to U$. Dacă $U = D_1 \ldots D_n$, atunci transformata a doua Delobel- Casey, notată $f_2(\Sigma)$ este definită prin:*

$$f_2(\Sigma) = d_1 \ldots d_n + f_1(\Sigma).$$

Exemplul 2.5 *Fie $U = \{A, B, C, D\}$ şi $\Sigma = \{AB \to CD, C \to A\}$. Atunci $f_2(\Sigma) = abcd + abc' + abd' + ca'$.*

Teorema 2.8 (a doua Delobel-Casey) [18]. *Următoarele afirmaţii sunt echivalente:*

a) $K = K_1 \ldots K_m$ este o supracheie pentru Σ.

b) $k_1 \ldots k_m$ implică logic $f_2(\Sigma)$.

Demonstraţie. Afirmaţia b) înseamnă că pentru orice asignare δ, pentru care $\delta(k_1 \ldots k_m) = 1$ avem $\delta(f_2(\Sigma)) = 1$.

Fie a) adevărată. Avem deci $\Sigma \models K_1 \ldots K_m \to U$. După teorema de echivalenţă obţinem $\overline{\Sigma} \models_{\text{c.l.}} k_1 \ldots k_m \Rightarrow d_1 \ldots d_n$. Dacă $\Sigma = \{\sigma_1, \ldots \sigma_k\}$, atunci ultima afirmaţie este echivalentă cu $\overline{\sigma_1} \ldots \overline{\sigma_k} \models_{\text{c.l.}} k_1 \ldots k_m \Rightarrow d_1 \ldots d_n$, care este echivalentă (din proprietăţile consecinţei logice) cu $k_1 \ldots k_m \models_{\text{c.l.}} \overline{\sigma_1} \ldots \overline{\sigma_k} \Rightarrow d_1 \ldots d_n$. Dar $\overline{\sigma_1} \ldots \overline{\sigma_k} \Rightarrow d_1 \ldots d_n$ este echivalentă logic cu $d_1 \ldots d_n \vee (\overline{\sigma_1} \ldots \overline{\sigma_k})'$. Obţinem astfel: $k_1 \ldots k_m \models_{\text{c.l.}} d_1 \ldots d_n \vee (\overline{\sigma_1} \ldots \overline{\sigma_k})'$. Am văzut că $\overline{\sigma_1} \ldots \overline{\sigma_k}$ este echivalentă cu $f_1(\Sigma)'$, deci $(\overline{\sigma_1} \ldots \overline{\sigma_k})'$ este echivalentă cu $f_1(\Sigma)$. Obţinem $k_1 \ldots k_m$ implică logic $d_1 \ldots d_n + f_1(\Sigma) = f_2(\Sigma)$, adică b). Urmărind acelaşi raţionament în ordine inversă, obţinem b) implică a). $\square$

Referinţe bibliografice: [3], [14], [15], [17], [18], [19], [24], [30], [33], [39], [45], [51], [52]

CAPITOLUL XV
DEPENDENŢE MULTIVALUATE

Fie $X, Y \subseteq U$. O dependenţă multivaluată este notată sintactic prin $X \rightarrow\!\!\!\rightarrow Y$. Vom da două definiţii pentru satisfacerea unei dependenţe multivaluate de către o relaţie r peste U.

Definiţia 0.1 *Relaţia r peste U satisface dependenţa multivaluată $X \rightarrow\!\!\!\rightarrow Y$, dacă pentru orice două uple $t_1, t_2 \in r$ şi $t_1[X] = t_2[X]$, există uplele t_3 şi t_4 din r, astfel încât:*
(i) $t_3[X] = t_1[X]$, $t_3[Y] = t_1[Y]$şi $t_3[Z] = t_2[Z]$;
(ii) $t_4[X] = t_2[X]$, $t_4[Y] = t_2[Y]$ şi $t_4[Z] = t_1[Z]$, unde $Z = U - XY$.

În definiţia 0.1 este suficient să cerem existenţa lui t_3 sau t_4, celălalt rezultă considerând uplele în ordinea t_2,t_1.

Pentru $t \in r$ avem $t[X] \in r[X]$. Notăm prin $F_Y(t[X]) = \{t'[Y]/t' \in r, t'[X] = t[X]\}$. Aceasta se numeşte mulţimea Y-valorilor asociate lui $t[X]$.

Exemplul 0.1 *Fie relaţia r dată astfel:*

A	B	C	D
a_1	b_1	c_1	d_1
a_1	b_2	c_2	d_2
a_1	b_1	c_1	d_2
a_1	b_2	c_2	d_1
a_2	b_3	c_1	d_1
a_2	b_3	c_1	d_2

Se verifică faptul că r satisface $A \rightarrow\!\!\!\rightarrow BC$ conform definiţiei 0.1.
Pentru $t \in r$, definim $M_Y(t[XZ]) = \{t'[Y]/t' \in r, t'[XZ] = t[XZ]\}$.

Definiţia 0.2 *Relaţia r peste U satisface dependenţa multivaluată* $X \twoheadrightarrow Y$, *dacă pentru orice* $t_1, t_2 \in r$, *dacă* $t_1[X] = t_2[X]$ *atunci* $M_Y(t_1[XZ]) = M_Y(t_2[XZ])$.

Propoziţia 0.1 *Definiţiile 0.1, 0.2 sunt echivalente.*

Demonstraţie. Fie r ce satisface $X \twoheadrightarrow Y$ după definiţia 0.1 şi fie $t_1, t_2 \in r$, astfel încât $t_1[X] = t_2[X]$. Fie $t_1'[Y] \in M_Y(t_1[XZ])$. Să arătăm că $t_1'[Y] \in M_Y(t_2[XZ])$. Avem $t_1' \in r$ şi $t_1'[XZ] = t_1[XZ]$. Rezultă $t_1'[X] = t_1[X] = t_2[X]$. Deoarece r satisface $X \twoheadrightarrow Y$ după definiţia 0.1 şi $t_1'[X] = t_2[X]$, există $t_3 \in r$, astfel încât: $t_3[X] = t_1'[X]$, $t_3[Y] = t_1'[Y]$ şi $t_3[Z] = t_2[Z]$. De aici $t_3[XZ] = t_2[XZ]$, de unde $t_3[Y] \in M_Y(t_2[XZ])$. Deoarece $t_1'[Y] = t_3[Y]$, obţinem $t_1'[Y] \in M_Y(t_2[XZ])$. Raţionamentul fiind simetric rezultă şi invers, adică $t_2'[Y] \in M_Y(t_2[XZ])$ implică $t_2'[Y] \in M_Y(t_1[XZ])$. Am arătat $M_Y(t_1[XZ]) = M_Y(t_2[XZ])$, adică r satisface $X \twoheadrightarrow Y$ după definiţia 1.2. Presupunem acum că r satisface $X \twoheadrightarrow Y$ după definiţia 1.2 şi fie $t_1, t_2 \in r$, astfel încât $t_1[X] = t_2[X]$. Din $M_Y(t_1[XZ]) = M_Y(t_2[XZ])$ şi faptul că $t_1[Y] \in M_Y(t_1[XZ])$ obţinem $t_1[Y] \in M_Y(t_2[XZ])$. Deci $\exists t_2' \in r$, astfel încât $t_1[Y] = t_2'[Y]$ şi $t_2'[XZ] = t_2[XZ]$. Pentru $t_2' \in r$ avem: $t_2'[X] = t_2[X] = t_1[X]$, $t_2'[Y] = t_1[Y]$ şi $t_2'[Z] = t_2[Z]$. Similar obţinem $t_1' \in r$, astfel încât: $t_1'[X] = t_2[X]$, $t_1'[Y] = t_2[Y]$ şi $t_1'[Z] = t_1[Z]$. Am arătat că r satisface $X \twoheadrightarrow Y$ după definiţia 0.1. $\square$

Observaţia 0.1 *Dacă r satisface dependenţa funcţională* $X \to Y$, *atunci pentru orice* $t \in r$, *avem* $M_Y(t[XZ]) = \{t[Y]\}$.

Observaţia 0.2 *Dacă r satisface dependenţa funcţională* $X \to Y$, *atunci r satisface dependenţa multivaluată* $X \twoheadrightarrow Y$.

Observaţia 0.3 *Dacă r satisface dependenţa multivaluată* $X \twoheadrightarrow Y$, *atunci putem defini o funcţie* $\psi : r[X] \to \mathcal{P}(r[Y])$, *prin* $\psi(t[X]) = M_Y(t[XZ])$, $\forall t \in r$. *Când r satisface* $X \to Y$, *atunci* $\psi : r[X] \to r[Y]$.

Proprietăţi ale dependenţelor multivaluate:

Propoziţia 0.2 *MVD0* **(Complementariere)**. *Fie* $X, Y, Z \subseteq U$, *astfel încât* $XYZ = U$ *şi* $Y \cap Z \subseteq X$. *Dacă r satisface* $X \twoheadrightarrow Y$, *atunci r satisface* $X \twoheadrightarrow Z$.

MVD1 (**Reflexivitate**). *Dacă* $Y \subseteq X$, *atunci orice relaţie* r *satisface* $X \twoheadrightarrow Y$.

MVD2 (**Extensie**). *Fie* $Z \subseteq W$ *şi* r *satisface* $X \twoheadrightarrow Y$. *Atunci* r *satisface* $XW \twoheadrightarrow YZ$.

MVD3 (**Tranzitivitate**). *Dacă* r *satisface* $X \twoheadrightarrow Y$ *şi* $Y \twoheadrightarrow Z$, *atunci* r *satisface* $X \twoheadrightarrow Z - Y$.

MVD4 (**Pseudotranzitivitate**). *Dacă* r *satisface* $X \twoheadrightarrow Y$ *şi* $YW \twoheadrightarrow Z$, *atunci* r *satisface* $XW \twoheadrightarrow Z - YW$.

MVD5 (**Uniune**). *Dacă* r *satisface* $X \twoheadrightarrow Y$ *şi* $X \twoheadrightarrow Z$, *atunci* r *satisface* $X \twoheadrightarrow YZ$.

MVD6 (**Descompunere**). *Dacă* r *satisface* $X \twoheadrightarrow Y$ *şi* $X \twoheadrightarrow Z$, *atunci* r *satisface* $X \twoheadrightarrow Y \cap Z$,$X \twoheadrightarrow Y - Z$,$X \twoheadrightarrow Z - Y$.

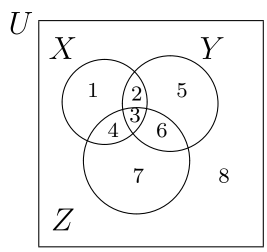
Deoarece vom lucra cu mulţimi de dependenţe, ce pot fi funcţionale sau multivaluate, vom avea nevoie de aşa numitele proprietăţi mixte.

FD-MVD1. Dacă r satisface $X \to Y$, atunci r satisface $X \twoheadrightarrow Y$.

FD-MVD2. Dacă r satisface $X \twoheadrightarrow Z$ şi $Y \to Z'$, cu $Z' \subseteq Z$ şi $Y \cap Z = \emptyset$, atunci r satisface $X \to Z'$.

FD-MVD3. Dacă r satisface $X \twoheadrightarrow Y$ şi $XY \to Z$, atunci r satisface $X \to Z - Y$.

Demonstrarea lui **MVD0**:
Pentru $X, Y, Z \subseteq U$ vom considera, în general, următoarea diagramă:



U reprezintă întregul pătrat, $X = \{1,2,3,4\}$, $Y = \{2,3,5,6\}$, $Z = \{3,4,6,7\}$, $U - XYZ = \{8\}$. În condiţiile proprietăţii MVD0 ($XYZ = U$ şi $Y \cap Z \subseteq X$), avem $8 = \emptyset$ şi $6 = \emptyset$. Fie $T_1 = U - XY = \{7\}$, $T_2 = U - XZ = \{5\}$. Presupunem că r satisface $X \twoheadrightarrow Y$. Aceasta înseamnă că pentru orice $t, t' \in r$ cu $t[X] = t'[X]$, există t_3 şi $t_4 \in r$, astfel încât $t[X] = t_3[X]$, $t[Y] = t_3[Y]$, $t'[T_1] = t_3[T_1]$ şi $t_4[X] = t'[X]$, $t_4[Y] = t'[Y]$, $t_4[T_1] = t[T_1]$. Să notăm prin t_i respectiv t'_i, proiecţia lui t respectiv t', pe domeniul i. Atunci pentru t avem:

$t \rightarrow ((t_1, t_2, t_3, t_4), (t_2, t_3, t_5), (t_7))$

$t' \rightarrow ((\underbrace{t'_1, t'_2, t'_3, t'_4}_{X}), (\underbrace{t'_2, t'_3, t'_5}_{Y})(\underbrace{\ t'_7\ }_{T_1}))$

$t \rightarrow ((t_1, t_2, t_3, t_4), (t_3, t_4, t_7), (t_5))$

$t' \rightarrow ((\underbrace{t'_1, t'_2, t'_3, t'_4}_{X}), (\underbrace{t'_3, t'_4, t'_7}_{Z}), (\underbrace{\ t'_5\ }_{T_2}))$

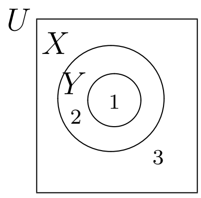
Din $t[X] = t'[X]$ rezultă $t_i = t'_i$, $i = \overline{1,4}$. Aplicăm faptul că r satisface $X \twoheadrightarrow Y$. Rezultă că există $t'' \in r$, astfel încât $t'' \rightarrow ((\underbrace{t_1, t_2, t_3, t_4}_{X}), (\underbrace{t_2, t_3, t_5}_{Y}), (\underbrace{\ t'_7\ }_{T_1}))$.

Acest t'' proiectat pe X, Z, T_2 dă:

$$t'' \rightarrow ((t_1, t_2, t_3, t_4), (t_3, t_4, t'_7), (t_5)) = ((t'_1, t'_2, t'_3, t'_4), (t'_3, t'_4, t'_7),$$
$$(t_5))$$

Avem deci satisfăcută definiţia 0.1 pentru t' şi t. Y şi Z intervin simetric în MVD0, deci rezultă şi invers: dacă r satisface $X \twoheadrightarrow Z$, atunci r satisface $X \twoheadrightarrow Y$.

Demonstrarea proprietăţii **MVD1**:



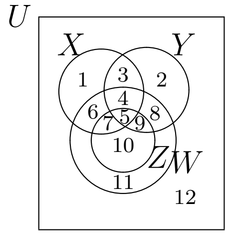
$Y = \{1\}$, $X = \{1,2\}$

$Z = U - XY = \{3\}$.

Fie t, $t' \in r$, astfel încât $t[X] = t'[X]$, adică $t_i = t'_i$, $i = 1, 2$.

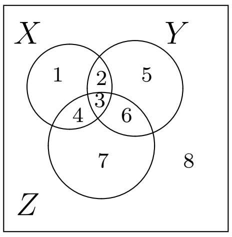
$t \to ((t_1, t_2), (t_1), (t_3))$
$t' \to (\underbrace{(t'_1, t'_2)}_{X}, \underbrace{(\ t'_1\ )}_{Y}, \underbrace{(\ t'_3\ )}_{Z})$
considerăm $t'' = t'$. Avem:
$t'' \to ((t'_1, t'_2), (t'_1), (t'_3)) = (\underbrace{(t_1, t_2)}_{X}, \underbrace{(\ t_1\ )}_{Y}, \underbrace{(\ t'_3\ )}_{Z})$

Demonstrarea proprietăţii **MVD2**:
Avem $Z \subseteq W$ şi r satisface $X \twoheadrightarrow Y$. Arătăm că r satisface $XW \twoheadrightarrow YZ$.



$X = \{1, 3, 4, 5, 6, 7\}$
$Y = \{2, 3, 4, 5, 8, 9\}$
$W = \{4, 5, 6, 7, 8, 9, 10, 11\}$
$Z = \{5, 7, 9, 10\}$
$T_1 = U - XY = \{10, 11, 12\}$
$T_2 = U - XWYZ = \{12\}$.
Fie t, $t' \in r$, astfel încât $t[XW] = t'[XW]$, adică $t_i = t'_i$, $i = 1, 3, \overline{4 - 11}$.
$t \to ((t_1, t_3, t_4 - t_{11}), (t_2 - t_5, t_7 - t_{10}), (t_{12}))$
$t' \to (\underbrace{(t'_1, t'_3, t'_4 - t'_{11})}_{XW}, \underbrace{(t'_2 - t'_5, t'_7 - t'_{10})}_{YZ}, \underbrace{(\ t'_{12}\ )}_{T_2})$,
unde $t_i - t_j$ notează toate componentele începând cu t_i şi terminând cu t_j, $(i < j)$.
Proiectăm acum t şi t' pe tripleta (X, Y, T_1):
$t \to ((t_1, t_3, t_4 - t_7), (t_2 - t_5, t_8, t_9), (t_{10} - t_{12}))$
$t' \to ((t'_1, t'_3, t'_4 - t'_7), (t'_2 - t'_5, t'_8, t'_9), (t'_{10} - t'_{12}))$.
Deoarece avem $t[X] = t'[X]$ şi r satisface $X \twoheadrightarrow Y$, rezultă că există $t'' \in r$, astfel încât: $t'' \to ((t_1, t_3, t_4 - t_7), (t_2 - t_5, t_8, t_9), (t'_{10} - t'_{12}))$ pe X, Y, T_1.
Proiectăm acest t'' pe XW, YZ, T_2, obţinem: $t'' \to ((t_1, t_3, t_4 - t_9, t'_{10}, t'_{11}), (t_2 - t_5, t_7 - t_9, t'_{10}), (t'_{12})) = ((t_1, t_3, t_4 - t_{11}), (t_2 - t_5, t_7 - t_{10}), (t'_{12}))$, deoarece $t_i = t'_i, i = 1, 3, \overline{4 - 11}$. Pentru t şi t' am obţinut t'' ce satisface definiţia 0.1.

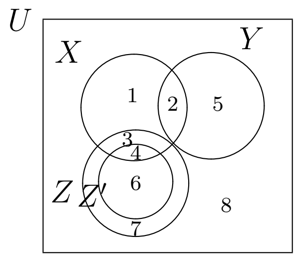
Demonstrarea proprietăţii **MVD3**:
Dacă r satisface $X \twoheadrightarrow Y$ şi $Y \twoheadrightarrow Z$, atunci r satisface $X \twoheadrightarrow Z - Y$.



$X = \{1, 2, 3, 4\}$
$Y = \{2, 3, 5, 6\}$
$Z = \{3, 4, 6, 7\}$,
$Z - Y = \{4, 7\}$
$T_1 = U - XY = \{7, 8\}$
$T_2 = U - YZ = \{1, 8\}$
$T_3 = U - X(Z - Y) = \{5, 6, 8\}$
Fie $t, t' \in r$, astfel încât $t[X] = t'[X]$, adică
$t_i = t'_i, i = \overline{1, 4}$
$t \rightarrow ((t_1 - t_4), (t_4, t_7), (t_5, t_6, t_8)$
$t' \rightarrow ((t'_1 - t'_4), (t'_4, t'_7), (t'_5, t'_6, t'_8))$ pe $X, Z - Y, T_3$
Considerăm t şi t' proiectate pe X, Y, T_1:
$t \rightarrow ((t_1 - t_4), (t_2, t_3, t_5, t_6), (t_7, t_8))$
$t' \rightarrow ((t'_1 - t'_4), (t'_2, t'_3, t'_5, t'_6), (t'_7, t'_8))$
Deoarece r satisface $X \twoheadrightarrow Y$, rezultă că există $t'' \in r$, astfel încât: $t'' \rightarrow ((t_1 - t_4), (t_2, t_3, t_5, t_6), (t'_7, t'_8))$ pe X, Y, T_1. Considerăm acum t şi t'' pe Y, Z, T_2.
$t \rightarrow ((t_2, t_3, t_5, t_6),\ (t_2, t_4, t_6, t_7),\ (t_1, t_8))$
$t'' \rightarrow ((t_2, t_3, t_5, t_6),\ (t_2, t_4, t_6, t'_7),\ (t_1, t'_8))$
r satisface $Y \twoheadrightarrow Z$. Pentru t'' şi t există $t''' \in r$, astfel încât $t''' \rightarrow ((t_2, t_3, t_5, t_6),\ (t_2, t_4, t_6, t'_7),(t_1, t_8))$. Considerând t''' proiectat pe $X, Z - Y$ şi T_3 obţinem:
$t''' \rightarrow ((t_1 - t_4),(t_4, t'_7),(t_5, t_6, t_8)) = ((t'_1 - t'_4),(t'_4, t'_7),(t_5, t_6, t_8))$
În concluzie, pentru t' şi $t \in r$ cu $t'[X] = t[X]$ am găsit t''' ce satisface definiţia 0.1.

Am arătat **FD-MVD1**. Să arătăm acum **FD-MVD2**:

Fie r ce satisface $X \twoheadrightarrow Z$ şi $Y \to Z'$, cu $Z' \subseteq Z$ şi $Y \cap Z = \emptyset$. Să arătăm că r satisface $X \to Z'$.



$X = \{1, 2, 3, 4\}$
$Y = \{2, 5\}$
$Z = \{3, 4, 6, 7\}$
$Z' = \{4, 6\}$
$T_1 = U - XZ = \{5, 8\}$.

Fie t,$t' \in r$, astfel încât : $t[X] = t'[X]$, deci $t_i = t'_i$, $i = \overline{1,4}$. Să arătăm că $t[Z'] = t'[Z']$, adică $t_6 = t'_6$.
Considerăm proiecţiile uplelor t şi t' pe X, Z, T_1 :

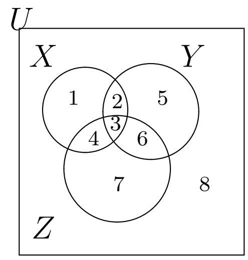
$t \to ((t_1 - t_4),(t_3, t_4, t_6, t_7),(t_5, t_8))$
$t' \to ((t'_1 - t'_4),(t'_3, t'_4, t'_6, t'_7),(t'_5, t'_8))$

Deoarece r satisface $X \twoheadrightarrow Z$, există $t'' \in r$, astfel încât proiecţiile lui t'' pe X, Z, T_1 sunt :

$t'' \to ((t_1 - t_4),\ (t_3, t_4, t_6, t_7),\ (t'_5, t'_8))$

Avem $t''_2 = t_2 = t'_2$ şi $t''_5 = t'_5$, deci $t''[Y] = t'[Y]$. Deoarece r satisface $Y \to Z'$, obţinem $t''[Z'] = t'[Z']$, adică $t''_6 = t'_6$, dar $t''_6 = t_6$. Obţinem deci $t_6 = t'_6$.

Să arătăm acum **FD-MVD3**:
Presupunem că r satisface $X \twoheadrightarrow Y$ şi $XY \to Z$. Să arătăm că r satisface $X \to Z - Y$.



$X = \{1, 2, 3, 4\}$
$Y = \{2, 3, 5, 6\}$
$Z = \{3, 4, 6, 7\}$
$Z - Y = \{4, 7\}$
$T_1 = U - XY = \{7, 8\}$.

Fie t, $t' \in r$, astfel încât $t[X] = t'[X]$, adică $t_i = t'_i$, $i = \overline{1,4}$. Să arătăm că $t[Z - Y] = t'[Z - Y]$, adică $t_7 = t'_7$. (Avem $t_4 = t'_4$).

Să proiectăm t şi t' pe X, Y şi T_1:
$t \to ((t_1 - t_4), (t_2, t_3, t_5, t_6), (t_7, t_8))$
$t' \to ((t'_1 - t'_4), (t'_2, t'_3, t'_5, t'_6), (t'_7, t'_8))$

Deoarece r satisface $X \twoheadrightarrow Y$, există $t'' \in r$, astfel încât proiecţiile lui t'' pe X, Y şi T_1 sunt:
$t'' \to ((t_1 - t_4), (t_2, t_3, t_5, t_6), (t'_7, t'_8))$

Avem $t''_i = t_i$, $i = \overline{1,6}$. Deoarece r satisface $XY \to Z$, rezultă $t''[Z] = t[Z]$, de unde $t''_7 = t_7$. Dar $t''_7 = t'_7$, de unde $t'_7 = t_7$. $\square$

Pentru fiecare proprietate a dependenţelor multivaluate asociem o regulă formală prin aceeaşi metodă ca la dependenţele funcţionale:

MVD0f: $\dfrac{XYZ = U \text{ şi } Y \cap Z \subseteq X,\ X \twoheadrightarrow Y}{X \twoheadrightarrow Z}$

MVD1f: $\dfrac{Y \subseteq X}{X \twoheadrightarrow Y}$

MVD2f: $\dfrac{Z \subseteq W,\ X \twoheadrightarrow Y}{XW \twoheadrightarrow YZ}$

MVD3f: $\dfrac{X \twoheadrightarrow Y,\ Y \twoheadrightarrow Z}{X \twoheadrightarrow Z - Y}$

MVD4f: $\dfrac{X \twoheadrightarrow Y,\ YW \twoheadrightarrow Z}{XW \twoheadrightarrow Z - YW}$

MVD5f: $\dfrac{X \twoheadrightarrow Y,\ X \twoheadrightarrow Z}{X \twoheadrightarrow YZ}$

MVD6f: $\dfrac{X \twoheadrightarrow Y,\ X \twoheadrightarrow Z}{X \twoheadrightarrow Y \cap Z,\ X \twoheadrightarrow Y - Z,\ X \twoheadrightarrow Z - Y}$

FD-MVD1f: $\dfrac{X \to Y}{X \twoheadrightarrow Y}$

FD-MVD2f: $\dfrac{X \twoheadrightarrow Z,\ Y \to Z',\ Z' \subseteq Z,\ Y \cap Z = \emptyset}{X \to Z'}$

FD-MVD3f: $\dfrac{X \rightarrow\rightarrow Y,\ XY \rightarrow Z}{X \rightarrow Z - Y}$.

Propoziţia 0.3 *Regulile de inferenţă enunţate mai sus sunt valide.*

Demonstraţie. Rezultă imediat din propoziţia 0.2. □

Propoziţia 0.4 *Fie $\mathcal{R}$ o mulţime de reguli valide şi γ o regulă $\dfrac{\alpha_1, \ldots, \alpha_k}{\beta}$, astfel încât $\{\alpha_1, \ldots, \alpha_k\}|_{\overline{\mathcal{R}}}\beta$, atunci şi regula γ este validă.*

Afirmaţia rezultă uşor prin inducţie după lungimea demonstraţiei în $\{\alpha_1, \ldots, \alpha_k\}$ utilizând $\mathcal{R}$. Faptul că $\{\alpha_1, \ldots, \alpha_k\}|_{\overline{\mathcal{R}}}\beta$ îl vom numi: "regula γ se exprimă cu ajutorul regulilor de inferenţă din $\mathcal{R}$". În continuare, vom considera în afară de regulile de inferenţă de mai sus şi regulile de inferenţă FD1f, FD2f, FD3f pentru dependenţele funcţionale.

Propoziţia 0.5 *Fie $\mathcal{R}_{FM}$ = { FD1f-FD3f, MVD0f-MVD3f, FD-MVD1f-FD-MVD3f }. Avem:*

FD-MVD3f se exprimă prin celelalte reguli din $\mathcal{R}_{FM}$ şi FD-MVD2f se exprimă prin celelalte reguli din $\mathcal{R}_{FM}$.

Demonstraţie. Fie $\alpha_1 : X \rightarrow\rightarrow Y$ şi $\alpha_2 : XY \rightarrow Z$. Aplicăm la prima MVD0f obţinem $\alpha_3 : X \rightarrow\rightarrow U - XY$. Din $XY \rightarrow Z$ şi $Z \rightarrow Z - XY$ (obţinută din FD1f) prin FD3f rezultă $\alpha_4 : XY \rightarrow Z - XY$. Deoarece $Z - XY \subseteq U - XY$ şi $XY \cap (U - XY) = \emptyset$, putem aplica FD-MVD2f pentru α_3 şi α_4 şi obţinem: $\alpha_5 : X \rightarrow Z - XY$. După FD1f avem $\alpha_6 : X \rightarrow X \cap Z - Y$. Aplicând FD5f care se exprimă cu ajutorul regulilor FD1f-FD3f (Propoziţia 1.3 Cap. II) rezultă $\alpha_7 : X \rightarrow Z - Y$, adică FD-MVD3f se exprimă prin celelalte reguli din $\mathcal{R}_{FM}$.

Fie date $\alpha_1 : X \rightarrow\rightarrow Z$, $\alpha_2 : Y \rightarrow Z'$ cu condiţiile $Z' \subseteq Z$ şi $Y \cap Z = \emptyset$. Aplicând MVD0f lui α_1 obţinem: $\alpha_3 : X \rightarrow\rightarrow U - XZ$. Deoarece $Y \subseteq X(U - XZ)$ prin FD1f obţinem: $\alpha_4 : X(U - XZ) \rightarrow Y$. Aplicând FD3f pentru α_4 şi α_2 se obţine $\alpha_5 : X(U - XZ) \rightarrow Z'$. Putem aplica regula FD-MVD3f pentru α_3 şi α_5, ceea ce conduce la $\alpha_6 : X \rightarrow Z' - (U - XZ)$. Dar $Z' - (U - XZ) = Z'$. Deci $\alpha_6 : X \rightarrow Z'$. În concluzie, regula FD-MVD2f se exprimă prin celelalte reguli din $\mathcal{R}_{FM}$. □

Propoziţia 0.6 *Regulile MVD4f-MVD6f se exprimă cu ajutorul regulilor MVD0f-MVD3f.*

Demonstraţie. Fie $\alpha_1 : X \twoheadrightarrow Y$ şi $\alpha_2 : YW \twoheadrightarrow Z$. Aplicăm pentru α_1 şi $W \subseteq W$ regula MVD2f şi obţinem: $\alpha_3 : XW \twoheadrightarrow YW$. Pentru α_3 şi α_2 aplicăm MVD3f şi se obţine: $\alpha_4 : XW \twoheadrightarrow Z - YW$. Deci $\{\alpha_1, \alpha_2\} |\overline{_{\{\text{MVD2f, MVD3f}\}}} \alpha_4$ (regula pentru pseudotranzitivitate). Considerăm acum MVD5f. Fie $\alpha_1 : X \twoheadrightarrow Y$ şi $\alpha_2 : X \twoheadrightarrow Z$. Din $\alpha_1, Z \subseteq Z$ şi MVD2f se obţine $\alpha_3 : XZ \twoheadrightarrow YZ$. Aplicând lui α_3 regula MVD0f se obţine $\alpha_4 : XZ \twoheadrightarrow (U - XYZ)$. Din $\alpha_2, X \subseteq X$ şi MVD2f rezultă $\alpha_5 : X \twoheadrightarrow XZ$. Din α_5, α_4 şi regula MVD3f rezultă $\alpha_6 : X \twoheadrightarrow U - XYZ$. Aplicând pentru α_6 regula MVD0f rezultă $\alpha_7 : X \twoheadrightarrow YZ$.

Fie $\alpha_1 : X \twoheadrightarrow Y$ şi $\alpha_2 : X \twoheadrightarrow Z$. Din $X \twoheadrightarrow X$ (reflexivitate) şi α_1 prin MVD3f obţinem $\alpha_3 : X \twoheadrightarrow Y - X$. Aplicând lui α_3 regula MVD0f obţinem $\alpha_4 : X \twoheadrightarrow U - XY$. Din $X \twoheadrightarrow X$ şi $X \twoheadrightarrow Z$ prin MVD3f obţinem: $\alpha_5 : X \twoheadrightarrow Z - X$. Aplicând acesteia MVD0f obţinem: $\alpha_6 : X \twoheadrightarrow U - XZ$. Aplicând acum MVD4f (reuniunea) pentru α_4 şi α_6 obţinem: $\alpha_7 : X \twoheadrightarrow (U - XY)(U - XZ)$. Aplicăm acum MVD0f pentru α_7, ceea ce dă: $\alpha_8 : X \twoheadrightarrow Y \cap Z - X$. Prin MVD1f avem $\alpha_9 : X \twoheadrightarrow X \cap Y \cap Z$. Prin reuniune (MVD4f) din α_8 şi α_9 se obţine $\alpha_{10} : X \twoheadrightarrow Y \cap Z$.

Astfel, pornind de la α_1 şi α_2 şi aplicând regulile MVD0f-MVD3f se obţine α_{10}. Să arătăm acum cea de a doua parte a lui MVD6f. Pornim de la $\alpha_1 : X \twoheadrightarrow Y$ şi $\alpha_2 : X \twoheadrightarrow Z$. Aplicând MVD5f se obţine $X \twoheadrightarrow YZ$. De aici, prin MVD0f, se obţine:

$\alpha_3 : X \twoheadrightarrow U - XYZ$. Aplicăm MVD5f pentru α_2 şi α_3 ceea ce dă : $\alpha_4 : X \twoheadrightarrow Z(U - XYZ)$. Pentru α_4 aplicăm MVD0f, ceea ce conduce la $\alpha_5 : X \twoheadrightarrow Y - XZ$. Prin reflexivitate avem $\alpha_6 : X \twoheadrightarrow X \cap Y - Z$. Prin MVD5f din α_5 şi α_6 se obţine: $\alpha_7 : X \twoheadrightarrow Y - Z$. În mod similar, se obţine $X \twoheadrightarrow Z - Y$ (schimbând în fond Y cu Z peste tot). □

Teorema 0.1 *Fie Σ o mulţime de dependenţe funcţionale sau multivaluate şi X o submulţime de atribute. Atunci există o partiţie a lui $U - X$ notată prin $\{Y_1, \ldots, Y_k\}$, astfel încât pentru $Z \subseteq U - X$ avem $\Sigma |\overline{_{\mathcal{R}_{FM}}} X \twoheadrightarrow Z$ iff Z este reuniunea unui număr de mulţimi din partiţia $\{Y_1, \ldots, Y_k\}$.*

Demonstraţie. Construim partiţia notată P, astfel: iniţial considerăm

în P numai $U - X$. Fie P obţinută la un moment dat având elementele $W_1, \ldots, W_n$.
Presupunem că $\Sigma|\overline{_{\mathcal{R}_{FM}}} X \twoheadrightarrow W_i$, pentru orice $i = \overline{1,n}$ (iniţial $\Sigma \vdash X \twoheadrightarrow U - X$ după MVD0f şi MVD1f).

Fie $\Sigma|\overline{_{\mathcal{R}_{FM}}} X \twoheadrightarrow Z$ şi $Z \subseteq U - X$ şi Z nu este reuniune de mulţimi W_i. Deoarece P este partiţie pentru $U - X$, rezultă că există W_i din P, astfel încât $W_i \cap Z \neq \emptyset$ şi $W_i - Z \neq \emptyset$. Pentru fiecare astfel de W_i din P înlocuim în P pe W_i cu $W_i \cap Z$ şi $W_i - Z$. Deoarece $\Sigma|\overline{_{\mathcal{R}_{FM}}} X \twoheadrightarrow Z$ şi după ipoteza inducţiei $\Sigma|\overline{_{\mathcal{R}_{FM}}} X \twoheadrightarrow W_i$, aplicând MVD6f se obţine $\Sigma|\overline{_{\mathcal{R}_{FM}}} X \twoheadrightarrow W_i \cap Z$ şi $\Sigma|\overline{_{\mathcal{R}_{FM}}} X \twoheadrightarrow W_i - Z$.

Altfel spus, noua partiţie satisface aceeaşi proprietate ca vechea partiţie. Deoarece U este finită şi mulţimea dependenţelor funcţionale sau multivaluate este finită, rezultă că algoritmul de mai sus este finit. (Numărul partiţiilor lui $U - X$ este de asemenea finit).

Fie $P = \{Y_1, \ldots, Y_k\}$ partiţia finală obţinută. Rezultă prin inducţie după paşii folosiţi în construcţia lui P că $\Sigma|\overline{_{\mathcal{R}_{FM}}} X \twoheadrightarrow Y_i$, $i = \overline{1,k}$. Dacă $Z \subseteq U - X$ este reuniune de Y_i, adică $Z = Y_{i_1} \cup \ldots \cup Y_{i_h}$, aplicând MVD5f se obţine $\Sigma|\overline{_{\mathcal{R}_{FM}}} X \twoheadrightarrow Z$.

Invers. Pentru $Z \subseteq U - X$, dacă $\Sigma|\overline{_{\mathcal{R}_{FM}}} X \twoheadrightarrow Z$, atunci Z este reuniune de Y_i, pentru că altfel s-ar putea rafina partiţia P, ceea ce este o contradicţie.
$\square$

Definiţia 0.3 *Pentru Σ o mulţime de dependenţe funcţionale sau multivaluate şi X o submulţime de atribute, numim baza de dependenţă pentru X cu privire la Σ, partiţia $B(\Sigma, X) = \{\{A_1\} \ldots \{A_h\}, Y_1, \ldots, Y_k\}$, unde $X = A_1 \ldots A_h$, iar $Y_1 \ldots Y_k$ este partiţia construită în teorema 0.1*

Observaţia 0.4 *Avem $\Sigma|\overline{_{\mathcal{R}_{FM}}} X \twoheadrightarrow Z$ iff Z este o reuniune de elemente din partiţia $B(\Sigma, X)$.*

În adevăr, dacă Z este reuniune de elemente din $B(\Sigma, X)$, atunci fie $Z = A_{i_1} \ldots A_{i_t} \cup Y_{j_1} \cup \ldots \cup Y_{j_l}$. Avem : $\Sigma|\overline{_{\mathcal{R}_{FM}}} X \twoheadrightarrow A_{i_p}$, $p = \overline{1,t}$ după MVD1f şi $\Sigma|\overline{_{\mathcal{R}_{FM}}} X \twoheadrightarrow Y_{j_q}$, $q = \overline{1,l}$ după teorema 0.1. Aplicând acestora MVD5f se obţine $\Sigma|\overline{_{\mathcal{R}_{FM}}} X \twoheadrightarrow Z$.

Invers, presupunem că avem $\Sigma|\overline{_{\mathcal{R}_{FM}}} X \twoheadrightarrow Z$. Fie $Z = X_1 \cup Z_1$, unde $X_1 \subseteq X$ şi $Z_1 \subseteq U - X$. ($X_1 \cap Z_1 = \emptyset$). După MVD6f rezultă $\Sigma|\overline{_{\mathcal{R}_{FM}}} X \twoheadrightarrow Z_1$. De aici după teorema 0.1, $Z_1 = Y_{j_1} \cup \ldots \cup Y_{j_l}$. Dacă

$X_1 = A_{i_1} \ldots A_{i_t}$, atunci Z este reuniunea elementelor $A_{i_1} \ldots A_{i_t}$, $Y_{j_1} \ldots Y_{j_l}$ din $B(\Sigma, X)$.

Observaţia 0.5 *Fie* $X^*_\Sigma = \{A | \Sigma|_{\overline{\mathcal{R}_{FM}}} X \to A\}$. *Atunci pentru orice* $A \in X^*_\Sigma$ *avem* $\{A\} \in B(\Sigma, X)$.

În adevăr, pentru $A \in X^*_\Sigma$ după FD-MVD1f, obţinem $\Sigma|_{\overline{\mathcal{R}_{FM}}} X \twoheadrightarrow A$ şi aplicând teorema 0.1, rezultă $\{A\} \in B(\Sigma, X)$.

1 Studiul dependenţelor funcţionale şi multivaluate utilizând calculul propoziţional

Pentru fiecare atribut $A \in U$ asociem o variabilă propoziţională notată a. Pentru o dependenţă funcţională $\alpha : A_1 \ldots A_m \to B_1 \ldots B_h$ asociem formula (numită implicaţie) $\overline{\alpha} : a_1 \ldots a_m \Rightarrow b_1 \ldots b_h$ (ca în cazul studiului dependenţelor funcţionale, utilizând calculul propoziţional, capitolul II, §2). Dacă $\alpha : A_1 \ldots A_m \twoheadrightarrow B_1 \ldots B_h$ este o dependenţă multivaluată şi $U - A_1 \ldots A_m B_1 \ldots B_h = C_1 \ldots C_p$, atunci formula asociată lui α va fi $\overline{\alpha} : a_1 \ldots a_m \Rightarrow b_1 \ldots b_h + c_1 \ldots c_p$. Semnul $+$ notează disjuncţia logică. Dacă $\alpha : X \to Y$, atunci $\overline{\alpha}$ se notează şi prin $x \Rightarrow y$. Dacă $\alpha : X \twoheadrightarrow Y$, atunci $\overline{\alpha}$ se notează şi prin $x \Rrightarrow y$.

Mulţimile X, Y şi $Z = U - XY$ pot fi vide. Facem convenţia că pentru mulţimea vidă $\emptyset$, formula asociată are valoarea true (conjuncţia unei mulţimi vide de variabile propoziţionale este true).

Propoziţia 1.1 $x \Rrightarrow y$ *este true iff* $x \Rrightarrow y - x$ *este true.*

Demonstraţie. Dacă $X = A_1 \ldots A_m$, $Y = B_1 \ldots B_h$ şi $Z = U - XY = C_1 \ldots C_p$, atunci $x \Rrightarrow y$ este $a_1 \ldots a_m \Rightarrow b_1 \ldots b_h + c_1 \ldots c_p$. Fie $Y - X = B_1 \ldots B_t$ şi $Y \cap X = B_{t+1} \ldots B_h$. Dacă δ este o asignare, astfel încât $\delta(x \Rrightarrow y) = true$, atunci putem avea:

a) există i, $i \in \{1, 2, \ldots, m\}$, astfel încât $\delta(a_i) = false$.
În acest caz $\delta(x \Rightarrow y - x) = true$.

b) $\forall i$, $i \in \{1, 2, \ldots m\}$, $\delta(a_i) = true$. De aici $\delta(a_1 \ldots a_m) = true$.

Rezultă $\delta(b_1 \ldots b_h) = true$ sau $\delta(c_1 \ldots c_p) = true$.

b_1) $\delta(b_1 \ldots b_h) = true$ implică $\delta(b_1 \ldots b_t) = true$, deci $\delta(x \Rightarrow (y - x) + c_1 \ldots c_p) = true$.

b_2) $\delta(c_1 \ldots c_p) = true$ atunci $\delta(x \Rightarrow\!\!\!\Rightarrow y - x) = true$.

Invers rezultă similar. $\square$

Pentru simplitatea scrierii vom considera valoarea de adevăr true notată prin 1, iar false prin 0.

Ca şi în cazul dependenţelor funcţionale, intenţia noastră este de a stabili o legătură între noţiunea de consecinţă din domeniul dependenţelor funcţionale şi multivaluate şi noţiunea de consecinţă logică din calculul propoziţional.

Exemplul 1.1 *Fie* $U = \{A, B, C, D\}$ *şi* $\Sigma = \{A \rightarrow\!\!\rightarrow B, C \rightarrow B\}$ *şi* $\sigma : A \rightarrow B$. *Atunci* $\overline{\sigma} : a \Rightarrow b$, $\overline{\Sigma} = \{a \Rightarrow b + cd,\ c \Rightarrow b\}$.

Arătăm că $\Sigma \models \sigma$. *În adevăr, fie* r *o relaţie ce satisface dependenţele* $A \rightarrow\!\!\rightarrow B$ *şi* $C \rightarrow B$. *Să arătăm că* r *satisface* $A \rightarrow B$. *Fie* $t_1, t_2 \in r$ *cu* $t_1[A] = t_2[A]$ *şi fie* $t_1 = (a_1, b_1, c_1, d_1)$ *şi* $t_2 = (a_2, b_2, c_2, d_2)$. *Deoarece* r *satisface* $A \rightarrow\!\!\rightarrow B$, *rezultă că există* $t_3, t_4 \in r$, *astfel încât* $t_3 = (a_1, b_1, c_2, d_2)$ *şi* $t_4 = (a_1, b_2, c_1, d_1)$. *Deoarece* $t_1, t_4 \in r$ *şi* r *satisface* $C \rightarrow B$, *rezultă că* $b_1 = b_2$, *adică* $t_1[B] = t_2[B]$, *deci* r *satisface* $A \rightarrow B$. *Arătăm acum că* $\overline{\Sigma} \models_{c.l.} \overline{\sigma}$. *Fie* δ *o asignare, astfel încât* $\delta(a \Rightarrow b + cd) = 1$ *şi* $\delta(c \Rightarrow b) = 1$. *Să arătăm că* $\delta(a \Rightarrow b) = 1$. *Dacă* $\delta(a) = 0$, *atunci am terminat. Dacă* $\delta(a) = 1$, *atunci* $\delta(b + cd) = 1$, *deci* $\delta(b) = 1$ *sau* $\delta(cd) = 1$; *în cazul* $\delta(b) = 1$ *am terminat. În cazul* $\delta(cd) = 1$ *rezultă* $\delta(c) = 1$ *şi cu* $\delta(c \Rightarrow b) = 1$ *se obţine* $\delta(b) = 1$, *deci iarăşi* $\delta(a \Rightarrow b) = 1$.

În continuare, vom considera o legătură între relaţii cu 2 uple şi asignări de variabile propoziţionale.

Definiţia 1.1 *Fie* T *o relaţie cu 2 uple* t_1 *şi* t_2 *şi* $X \rightarrow\!\!\rightarrow Y$ *o dependenţă multivaluată. Spunem că* $X \rightarrow\!\!\rightarrow Y$ *este îndeplinită activ în* T, *dacă este îndeplinită în* T *(*T *satisface* $X \rightarrow\!\!\rightarrow Y$*) şi* $t_1[X] = t_2[X]$. *(Vom spune şi:* T *satisface activ* $X \rightarrow\!\!\rightarrow Y$*).*

Observaţia 1.1 *Există două situaţii în care o dependenţă multivaluată* $X \rightarrow\!\!\rightarrow Y$ *este îndeplinită de* T:

1) Cele două uple nu coincid pe X, *sau*

2) $X \rightarrow\!\!\rightarrow Y$ *este îndeplinită activ în* T.

Lema 1.1 *Fie* $U = XYZ$ *şi* T *o relaţie cu 2 uple. Atunci dependenţa multivaluată* $X \to\to Y$ *este îndeplinită activ în* T*, dacă şi numai dacă:*
1) Cele două uple din T *concordă pe* X *şi*
2) Cele două uple din T *concordă pe* Y *sau concordă pe* Z*.*

Demonstraţie. Dacă 1) şi 2) sunt satisfăcute, atunci $X \to\to Y$ este îndeplinită activ în T. Invers, fie $X \to\to Y$ îndeplinită activ în T. Avem în primul rînd 1). Fie cele două uple din T: $t = (x, y, z), t' = (x', y', z')$ cu $x = x'$. Deoarece T satisface $X \to\to Y$ rezultă că $t_3 = (x, y, z')$ şi $t_4 = (x', y', z) \in T$. Dacă $y \neq y'$ şi $z \neq z'$, atunci ar rezulta că T are 4 uple distincte, deci contradicţie. Prin urmare $y = y'$ sau $z = z'$. $\square$

Lema 1.2 *Fie* T *şi* T' *relaţii cu două uple peste* U *satisfăcând condiţia: dacă cele două uple din* T *concordă într-o coloană, atunci cele două uple ale lui* T' *concordă în aceeaşi coloană. Atunci orice dependenţă multivaluată ce este îndeplinită activ în* T *este, de asemenea, îndeplinită activ în* T'*.*

Demonstraţie. Fie $X \to\to Y$ îndeplinită activ în T. Fie $Z = U - XY$. Deoarece $X \to\to Y$ este echivalentă cu $X \to\to Y - X$ putem presupune că X, Y şi Z sunt disjuncte. Deoarece $X \to\to Y$ este îndeplinită activ în T, atunci cele două uple din T concordă pe X, de unde rezultă că cele două uple ale lui T' concordă pe X. După lema 1.1, cele două uple din T concordă pe Y sau pe Z, deci cele două uple din T' concordă pe Y sau Z. De aici, aplicând din nou lema 1.1, se obţine faptul că $X \to\to Y$ este îndeplinită activ în T'. $\square$

Lema 1.3 *Fie* r *o relaţie cu două uple* t_1 *şi* t_2 *şi asignarea* δ*, definită prin* $\delta(a) = 1$ *dacă* $t_1[A] = t_2[A]$ *şi* $\delta(a) = 0$ *în caz contrar. Atunci* r *satisface dependenţa* σ *(funcţională sau multivaluată) dacă şi numai dacă* $\delta(\overline{\sigma}) = 1$*, unde* $\overline{\sigma}$ *este formula asociată dependenţei* σ*.*

Demonstraţie. Presupunem că $\delta(\overline{\sigma}) = 1$. Să arătăm că r satisface σ.

Cazul 1. σ este o dependenţă funcţională: $X \to Y$, atunci $\overline{\sigma} : x \Rightarrow y$. Dacă $\delta(x) = 0$, atunci există a ce apare în x, astfel încât $\delta(a) = 0$, de unde $t_1[A] \neq t_2[A]$, deci $t_1[X] \neq t_2[X]$. Aceasta înseamnă că r satisface $X \to Y$.

Dacă $\delta(x) = 1$, atunci $\delta(y) = 1$, deci $t_1[X] = t_2[X]$ şi $t_1[Y] = t_2[Y]$, ceea ce denotă faptul că r satisface $X \to Y$.

Cazul 2. σ este o dependenţă multivaluată : $X \twoheadrightarrow Y$. Atunci $\overline{\sigma} : x \Rightarrow y+z$, unde z are variabile corespunzătoare lui Z, $Z = U - XY$. Dacă $\delta(x) = 0$, atunci $t_1[X] \neq t_2[X]$, deci r satisface $X \twoheadrightarrow Y$. Dacă $\delta(x) = 1$, atunci $\delta(y) = 1$ sau $\delta(z) = 1$. Dacă $\delta(y) = 1$, atunci $t_1[Y] = t_2[Y]$. Aceasta împreună cu $t_1[X] = t_2[X]$ conduc la faptul că r satisface $X \twoheadrightarrow Y$ (lema 1.1). Dacă $\delta(z) = 1$, atunci $t_1[Z] = t_2[Z]$. De aici utilizând aceeaşi lema 1.1, obţinem: r satisface $X \twoheadrightarrow Y$.

Demonstraţia lemei în sens invers este similară cu cea de sus. $\square$

Vom enunţa în continuare teorema de echivalenţă:

Teorema 1.1 Teorema de echivalenţă. *Fie Σ o mulţime de dependenţe funcţionale sau multivaluate şi σ o dependenţă funcţională sau multivaluată. Următoarele afirmaţii sunt echivalente:*

a) σ este o consecinţă a lui Σ.

b) σ este o consecinţă a lui Σ pe mulţimea relaţiilor cu 2 uple.

c) $\overline{\sigma}$ este consecinţă logică a lui $\overline{\Sigma}$.

Vom da întâi o demonstraţie sintactică a teoremei de echivalenţă. Pentru aceasta vom considera reguli de inferenţă pentru implicaţii ce se construiesc pornind de la regulile de inferenţă din $\mathcal{R}_{FM} = \{$FD1f-FD3f, MVD0f-MVD3f, FD-MVD1f-FD- M
În capitolul II am considerat regulile de inferenţă A1′, A2′, A3′ pentru implicaţii din calculul propoziţional, reguli ce corespund axiomelor lui Armstrong. Enunţăm regulile de inferenţă asociate celor din $\mathcal{R}_{FM}$:

FD1′: $\dfrac{y \subseteq x}{x \Rightarrow y}$

$y \subseteq x$ notează faptul că orice variabilă ce apare în y, apare de asemenea în x.

FD2′: $\dfrac{x \Rightarrow y,\ z \subseteq w}{xw \Rightarrow yz}$

FD3′: $\dfrac{x \Rightarrow y,\ y \Rightarrow z}{x \Rightarrow z}$

MVD0′: $\dfrac{xyz = u,\ y \cap z \subseteq x, x \twoheadRightarrow y}{x \twoheadRightarrow z}$, u este conjuncţia tuturor variabilelor asociate lui U.

MVD1′: $\dfrac{y \subseteq x}{x \twoheadRightarrow y}$

MVD2′: $\dfrac{z \subseteq w,\ x \Rightarrow\!\!\!\Rightarrow y}{xw \Rightarrow\!\!\!\Rightarrow yz}$

MVD3′: $\dfrac{x \Rightarrow\!\!\!\Rightarrow y,\ y \Rightarrow\!\!\!\Rightarrow z}{x \Rightarrow\!\!\!\Rightarrow z - y}$

FD-MVD1′: $\dfrac{x \Rightarrow y}{x \Rightarrow\!\!\!\Rightarrow y}$

FD-MVD2′: $\dfrac{x \Rightarrow\!\!\!\Rightarrow z,\ y \Rightarrow z',\ z' \subseteq z,\ y \cap z = \emptyset}{x \Rightarrow z'}$

FD-MVD3′: $\dfrac{x \Rightarrow\!\!\!\Rightarrow y,\ xy \Rightarrow z}{x \Rightarrow z - y}$

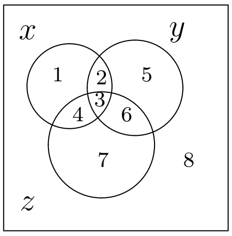
Observaţia 1.2 *Deoarece sistemul de reguli {A1′, A2′, A3′} este valid şi {A1′, A2′, A3′} este echivalent cu {FD1′,FD2′,FD3′} (în virtutea faptului că {A1, A2, A3} este echivalent cu {FD1f, FD2f, FD3f} şi* $\Sigma|\overline{_{\{FD1f\text{-}FD3f\}}}\,\sigma$ *iff* $\Sigma|\overline{_{\{A_1,A_2,A_3\}}}\,\sigma$*) rezultă că FD1′, FD2′, FD3′ sunt valide. Această afirmaţie rezultă desigur şi direct.*

Observaţia 1.3 *În virtutea propoziţiei 0.5 ne vom dispensa de una din regulile FD-MVD2′ sau FD-MVD3′. Vom renunţa la ultima.*

Lema 1.4 *Regulile de inferenţă FD1′-FD3′, MVD0′-MVD3′, FD-MVD1′, FD-MVD2′ sunt valide. Fie* $\mathcal{R}'_{FM}$ *mulţimea acestor reguli.*

Demonstraţie. Primele 3 sunt valide după observaţia 1.2.

MVD0′ este validă: Considerăm reprezentarea mulţimilor de variabile din $x, y, z, u - xy, u - xz$:

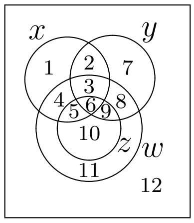


Din ipotezele respective se obţine: $6 = \emptyset$ şi $8 = \emptyset$.
Fie $t_1 = u - xy$ şi $t_2 = u - xz$.
Să notăm prin $\bar{i}$ conjuncţia variabilelor din domeniul i, $i = \overline{1,5}, 7$.

Formula $x \twoheadrightarrow y$ se scrie astfel: $\overline{1}\ \overline{2}\ \overline{3}\ \overline{4} \ \Rightarrow \overline{2}\ \overline{3}\ \overline{5} + \overline{7}$, iar $x \twoheadrightarrow z$ devine $\overline{1}\ \overline{2}\ \overline{3}\ \overline{4} \Rightarrow \overline{3}\ \overline{4}\ \overline{7} + \overline{5}$.
Fie δ o asignare astfel încât $\delta(x \twoheadrightarrow y) = 1$.
Dacă $\delta(\overline{1}\ \overline{2}\ \overline{3}\ \overline{4}) = 0$, atunci $\delta(x \twoheadrightarrow z) = 1$.
Dacă $\delta(\overline{1}\ \overline{2}\ \overline{3}\ \overline{4}) = 1$, atunci $\delta(\overline{2}\ \overline{3}\ \overline{5}) = 1$ sau $\delta(\overline{7}) = 1$.
Când avem $\delta(\overline{2}\ \overline{3}\ \overline{5}) = 1$, atunci $\delta(\overline{5}) = 1$, deci $\delta(x \twoheadrightarrow z) = 1$.
Când avem $\delta(\overline{7}) = 1$, atunci $\delta(\overline{3}\ \overline{4}\ \overline{7}) = 1$, deci $\delta(x \twoheadrightarrow z) = 1$.

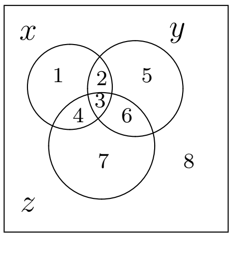
MVD1′ este validă: Fie $z = u - xy = u - x$, atunci $x \twoheadrightarrow y$ notează formula $x \Rightarrow y + z$, care este tautologie ($y \subseteq x$).

MVD2′ este validă: Ca mai sus, reprezentăm mulţimile respective de variabile (notate cu litere mici, la fel ca şi conjuncţia lor).



Fie $\overline{i}$ conjuncţia variabilelor din domeniul i, $i = \overline{1,12}$. Fie $t_1 = u - xy$ şi $t_2 = u - xwyz$. Atunci $x \twoheadrightarrow y$ notează $\overline{1}\ \overline{2}\ \overline{3}\ \overline{4}\ \overline{5}\ \overline{6} \twoheadrightarrow \overline{2}\ \overline{3}\ \overline{6}\ \overline{7}\ \overline{8}\ \overline{9} + \overline{10}\ \overline{11}\ \overline{12}$. Formula $xw \twoheadrightarrow yz$ devine: $\overline{1}\ \overline{2}\ \overline{3}\ \overline{4}\ \overline{5}\ \overline{6}\ \overline{8}\ \overline{9}\ \overline{10}\ \overline{11} \Rightarrow \overline{2}\ \overline{3}\ \overline{6}\ \overline{7}\ \overline{8}\ \overline{9}\ \overline{5}\ \overline{10} + \overline{12}$. Fie δ o asignare, astfel încât $\delta(x \twoheadrightarrow y) = 1$. Dacă există $i \in \overline{1,11} - \{7\}$, astfel încât $\delta(\overline{i}) = 0$, atunci $\delta(xw \twoheadrightarrow yz) = 1$. Dacă $\delta(\overline{i}) = 1$, pentru $i = \overline{1,11}$, $i \neq 7$, atunci deoarece $\delta(x \twoheadrightarrow y) = 1$, rezultă $\delta(\overline{2}\ \overline{3}\ \overline{6}\ \overline{7}\ \overline{8}\ \overline{9}) = 1$ sau $\delta(\overline{10}\ \overline{11}\ \overline{12}) = 1$. În ambele cazuri, se obţine $\delta(xw \twoheadrightarrow yz) = 1$.

MVD3′ este validă: Fie reprezentarea mulţimilor de variabile din $x, y, z, u - xy, u - yz, u - x(z - y)$:



Notăm cu $\overline{i}$ conjuncţia variabilelor din domeniul $i, i = \overline{1,8}$. Formula $x \Rightarrow\!\!\!\Rightarrow y$ devine:$\overline{1}\ \overline{2}\ \overline{3}\ \overline{4}\ \Rightarrow \overline{2}\ \overline{3}\ \overline{5}\ \overline{6} + \overline{7}\ \overline{8}$.
Formula $y \Rightarrow\!\!\!\Rightarrow z$ se scrie astfel: $\overline{2}\ \overline{3}\ \overline{5}\ \overline{6} \Rightarrow \overline{3}\ \overline{4}\ \overline{6}\ \overline{7} + \overline{1}\ \overline{8}$. Formula $x \Rightarrow\!\!\!\Rightarrow z - y$ devine: $\overline{1}\ \overline{2}\ \overline{3}\ \overline{4} \Rightarrow \overline{4}\ \overline{7} + \overline{5}\ \overline{6}\ \overline{8}$.
Fie o asignare δ ce satisface ipotezele: $\delta(x \Rightarrow\!\!\!\Rightarrow y) = 1$ şi $\delta(y \Rightarrow\!\!\!\Rightarrow z) = 1$. Să arătăm că $\delta(x \Rightarrow\!\!\!\Rightarrow z - y) = 1$. Dacă $\exists i \in \{1, 2, 3, 4\}$, astfel încât $\delta(\overline{i}) = 0$, atunci avem concluzia dorită.
Dacă $\delta(\overline{i}) = 1$, $i = \overline{1,4}$, atunci deoarece δ satisface $x \Rightarrow\!\!\!\Rightarrow y$, avem:

a) $\delta(\overline{2}\ \overline{3}\ \overline{5}\ \overline{6}) = 1$ sau

b) $\delta(\overline{7}\ \overline{8}) = 1$

În cazul a), deoarece δ satisface $y \Rightarrow\!\!\!\Rightarrow z$ avem:

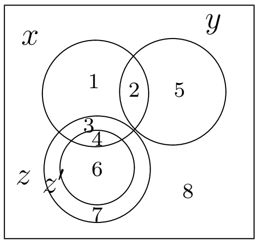
a1) $\delta(\overline{3}\ \overline{4}\ \overline{6}\ \overline{7}) = 1$ sau

a2) $\delta(\overline{1}\ \overline{8}) = 1$

În cazul a1), deoarece $\delta(\overline{4}\ \overline{7}) = 1$ se obţine $\delta(x \Rightarrow\!\!\!\Rightarrow z - y) = 1$.
În cazul a2) avem $\delta(\overline{5}\ \overline{6}\ \overline{8}) = 1$, deci $\delta(x \Rightarrow\!\!\!\Rightarrow z - y) = 1$.
Fie acum b) adevărată. De aici $\delta(\overline{7}) = 1$ şi deoarece $\delta(\overline{4}) = 1$, se obţine $\delta(\overline{4}\ \overline{7}) = 1$, adică $\delta(x \Rightarrow\!\!\!\Rightarrow z - y) = 1$.

FD-MVD1′ este validă: (imediat).

FD-MVD2′ este validă:



Utilizând aceleaşi notaţii ca mai sus obţinem:

$x \Rightarrow\!\!\!\Rightarrow z : \overline{1}\ \overline{2}\ \overline{3}\ \overline{4} \Rightarrow \overline{3}\ \overline{4}\ \overline{6}\ \overline{7} + \overline{5}\ \overline{8}$

$y \Rightarrow z' : \overline{2}\ \overline{5} \Rightarrow \overline{4}\ \overline{6}$

$x \Rightarrow z' : \overline{1}\ \overline{2}\ \overline{3}\ \overline{4} \Rightarrow \overline{4}\ \overline{6}$

Fie o asignare δ, astfel încât $\delta(x \Rightarrow\Rightarrow z) = 1$ şi $\delta(y \Rightarrow z') = 1$. Trebuie să arătăm că $\delta(x \Rightarrow z') = 1$. Dacă există $i \in \{1, 2, 3, 4\}$, astfel încât $\delta(\overline{i}) = 0$, atunci am terminat. Dacă $\delta(\overline{i}) = 1$, pentru $i = \overline{1,4}$, atunci deoarece δ satisface $x \Rightarrow\Rightarrow z$, avem $\delta(\overline{3}\ \overline{4}\ \overline{6}\ \overline{7}) = 1$ sau $\delta(\overline{5}\ \overline{8}) = 1$. În prima situaţie avem $\delta(\overline{4}\ \overline{6}) = 1$, deci $\delta(x \Rightarrow z') = 1$. În situaţia $\delta(\overline{5}\ \overline{8}) = 1$, avem $\delta(\overline{2}\ \overline{5}) = 1$ şi deoarece δ satisface $y \Rightarrow z'$, rezultă $\delta(\overline{4}\ \overline{6}) = 1$, deci $\delta(x \Rightarrow z') = 1$. $\square$

Observaţia 1.4 *Fie $\overline{\Sigma}$ o mulţime de formule din calculul propoziţional şi $\overline{\Sigma}^+$ mulţimea formulelor ce pot fi derivate din $\overline{\Sigma}$, utilizând regulile de inferenţă FD1′-FD3′, MVD0′- MVD3′, FD-MVD1′, FD-MVD2′. Atunci după lema 1.4 rezultă că $\overline{\Sigma} \models_{\overline{c.l.}} \overline{\Sigma}^+$, adică formulele din $\overline{\Sigma}^+$ sunt consecinţe logice ale formulelor din $\overline{\Sigma}$.*

Lema 1.5 *Fie $\overline{\Sigma}$ o mulţime de formule asociate mulţimii Σ de dependenţe funcţionale sau multivaluate şi X o mulţime de atribute. Fie $X^+ = \{A|\Sigma|_{\overline{\mathcal{R}_{FM}}} X \rightarrow A\}$. Fie $B(\Sigma, X)$ baza de dependenţă pentru X cu privire la Σ şi $W \in B(\Sigma, X)$, astfel încât $W \cap X^+ = \emptyset$. Considerăm asignarea δ_0 definită astfel: $\delta_0(a) = 0$ iff $A \in W$. Atunci avem: $\delta_0(\overline{\sigma}) = 1$ pentru $\forall \overline{\sigma} \in \overline{\Sigma}$.*

Demonstraţie. Fie $\overline{\sigma} : y \Rrightarrow z$, deci σ este dependenţa multivaluată $Y \rightarrow\rightarrow Z$; $\overline{\sigma} \in \overline{\Sigma}$. Să arătăm că $\delta_0(\overline{\sigma}) = 1$. Dacă $\delta_0(y) = 0$, atunci rezultă $\delta_0(\overline{\sigma}) = 1$. Fie deci situaţia când $\delta_0(y)$=1. După definiţia lui δ_0, obţinem: $Y \cap W = \emptyset$. Fie $V_1, V_2, \ldots, V_k$ toate elementele partiţiei $B(\Sigma, X)$, ce au intersecţie nevidă cu Y. Fie $Y' = V_1 \cup \ldots \cup V_k$. Rezultă că $Y \subseteq Y'$ şi $V_j \cap W = \emptyset$ pentru orice $j = \overline{1, k}$ (altfel am avea pentru un j, $W = V_j$ şi $Y \cap V_j \neq \emptyset$ implică $Y \cap W \neq \emptyset$ absurd). Obţinem astfel $Y' \cap W = \emptyset$ şi după definiţia lui δ_0 rezultă $\delta_0(y') = 1$. Aplicând extensia (MVD2f) pentru $Y \rightarrow\rightarrow Z$ se obţine: $\Sigma|_{\overline{\mathcal{R}_{FM}}} YY' \rightarrow\rightarrow Z$, deci $\Sigma|_{\overline{\mathcal{R}_{FM}}} Y' \rightarrow\rightarrow Z$. Deoarece Y' este reuniune de elemente din $B(\Sigma, X)$ după observaţia 0.4, avem $\Sigma|_{\overline{\mathcal{R}_{FM}}} X \rightarrow\rightarrow Y'$. Aplicând tranzitivitatea (MVD3f) pentru $X \rightarrow\rightarrow Y'$ şi $Y' \rightarrow\rightarrow Z$ se obţine: $\Sigma|_{\overline{\mathcal{R}_{FM}}} X \rightarrow\rightarrow Z - Y'$. Aplicând din nou observaţia 0.4 rezultă că $Z - Y'$ este reuniune de elemente din $B(\Sigma, X)$. Deoarece $W \in B(\Sigma, X)$, putem avea numai una din situaţiile: a) $W \subseteq Z - Y'$, sau b) $W \cap (Z - Y') = \emptyset$. În situaţia a) avem $W \subseteq Z$, deoarece $Y' \cap W = \emptyset$. Atunci, dacă $T_1 = U - YZ$, obţinem $W \cap T_1 = \emptyset$. După definiţia asignării δ_0, rezultă că $\delta_0(t_1) = 1$. Dar $y \Rrightarrow z$ înseamnă $y \Rightarrow z + t_1$, de unde $\delta_0(y \Rrightarrow z) = 1$.

În situaţia b) rezultă $W \cap Z = \emptyset$ (deoarece $W \cap Y' = \emptyset$). După definiţia lui δ_0 vom avea $\delta_0(z) = 1$, deci $\delta_0(y \Rightarrow z + t_1) = 1$,adică $\delta_0(\overline{\sigma}) = 1$. $(T_1 = U - YZ)$.

Fie acum $\overline{\sigma} \in \overline{\Sigma}$ o formulă asociată unei dependenţe funcţionale, deci $\overline{\sigma} : y \Rightarrow z$. Dacă $\delta_0(y) = 0$, atunci $\delta_0(\overline{\sigma}) = 1$. Fie deci $\delta_0(y) = 1$. Trebuie să arătăm că $\delta_0(z) = 1$. Fie Y' definită ca în cazul anterior. Pentru $Y \to Z \in \Sigma$, aplicând FD-MVD1f, obţinem $\Sigma \vdash_{\mathcal{R}_{FM}} Y \twoheadrightarrow Z$. Pentru $Y \twoheadrightarrow Z$ aplicând extensia (MVD2f) se obţine: $\Sigma \vdash_{\mathcal{R}_{FM}} YY' \twoheadrightarrow Z$, adică $\Sigma \vdash_{\mathcal{R}_{FM}} Y' \twoheadrightarrow Z$ (deoarece $Y \subseteq Y'$). Ca mai sus, se obţine $\Sigma \vdash_{\mathcal{R}_{FM}} X \twoheadrightarrow Y'$. Aplicând tranzitivitatea, rezultă $\Sigma \vdash_{\mathcal{R}_{FM}} X \twoheadrightarrow Z - Y'$. Din $Y \to Z$, aplicând descompunerea (ce se exprimă prin FD1f-FD3f), obţinem $\Sigma \vdash_{\mathcal{R}_{FM}} Y \to Z - Y'$. Deoarece $Y \cap (Z - Y') = \emptyset$, putem aplica regula FD-MVD2f pentru $X \twoheadrightarrow Z - Y'$ şi $Y \to Z - Y'$, ceea ce ne dă: $\Sigma \vdash_{\mathcal{R}_{FM}} X \to Z - Y'$, ceea ce înseamnă după definiţia lui X^+ că $Z - Y' \subseteq X^+$.

După definiţia lui δ_0, rezultă că $\delta_0(b) = 1$, pentru orice $B \in X^+$, deci $\delta_0(z - y') = 1$. Deoarece $\delta_0(y') = 1$, se obţine $\delta_0(z) = 1$, adică $\delta_0(y \Rightarrow z) = 1$.

$\square$

Teorema 1.2 (Teorema de completitudine pentru formule). *Fie $\overline{\Sigma}$ mulţimea de formule asociate mulţimii Σ de dependenţe funcţionale sau multivaluate şi $\overline{\sigma}$ formula asociată dependenţei σ (funcţională sau multivaluată). Atunci $\overline{\sigma}$ este consecinţă logică a lui $\overline{\Sigma}$ dacă şi numai dacă $\overline{\sigma}$ poate fi demonstrată în $\overline{\Sigma}$ utilizând regulile de inferenţă $\mathcal{R}'_{FM}$. Pe scurt:*

$$\overline{\Sigma} \models_{c.l.} \overline{\sigma} \;\; iff \;\; \overline{\Sigma} \vdash_{\mathcal{R}'_{FM}} \overline{\sigma}.$$

Demonstraţie. Dacă $\overline{\Sigma} \vdash_{\mathcal{R}'_{FM}} \overline{\sigma}$, atunci $\overline{\Sigma} \models_{\text{c.l.}} \overline{\sigma}$ după observaţia 1.4. Fie acum $\overline{\Sigma} \models_{\text{c.l.}} \overline{\sigma}$ şi să demonstrăm că $\overline{\Sigma} \vdash_{\mathcal{R}'_{FM}} \overline{\sigma}$. Vom presupune contrariul: deci $\overline{\Sigma} \not\vdash_{\mathcal{R}'_{FM}} \overline{\sigma}$. Vom distinge cazurile când σ este funcţională sau multivaluată.

a) $\sigma : X \to Y$, deci $\overline{\sigma} : x \Rightarrow y$. Deoarece $\overline{\Sigma} \not\vdash_{\mathcal{R}'_{FM}} \overline{\sigma}$, rezultă că există B în Y, astfel încât $\overline{\Sigma} \not\vdash_{\mathcal{R}'_{FM}} x \Rightarrow b$ (în caz contrar, aplicând reuniunea, s-ar obţine $\overline{\Sigma} \vdash_{\mathcal{R}'_{FM}} x \Rightarrow y$. Avem atunci $B \notin X^+$. Baza de dependenţă $B(\Sigma, X)$, fiind o partiţie a lui U, rezultă că există W din $B(\Sigma, X)$, astfel încât $B \in W$. Rezultă $W \cap X^+ = \emptyset$ (se utilizează şi observaţia 0.4). Definim asignarea δ_0 prin $\delta_0(a) = 0 \;\; iff \;\; A \in W$. După lema 1.5, toate formulele lui $\overline{\Sigma}$ sunt adevărate pentru δ_0, dar $\delta_0(b) = 0$ şi $\delta_0(x) = 1$, deci $\delta_0(x \Rightarrow y) = 0$, ceea ce contrazice ipoteza: $\overline{\Sigma} \models_{\text{c.l.}} \overline{\sigma}$.

b)$\sigma : X \twoheadrightarrow Y$, deci $\sigma : x \Rrightarrow y$. Deoarece $\overline{\Sigma} \not\vdash_{\mathcal{R}'_{FM}} \overline{\sigma}$, avem $\Sigma \not\vdash_{\mathcal{R}_{FM}} \sigma$. Ţinând cont de observaţia 0.4, obţinem că există $W \in B(\Sigma, X)$, astfel încât

$W \cap Y \neq \emptyset$ şi $W \not\subseteq Y$ (altfel, Y ar fi reuniune de elemente din baza de dependenţă a lui X relativ la Σ). Deoarece $W \not\subseteq Y$ rezultă că W are cel puţin două elemente. Deoarece fiecare atribut din X^+ formează o clasă în $B(\Sigma, X)$ obţinem că $W \cap X^+ = \emptyset$.
Definim asignarea δ_0 prin: $\delta_0(a) = 0 \;\; iff \;\; A \in W$. După lema 1.5 obţinem $\delta_0(\overline{\alpha}) = 1$ pentru orice $\overline{\alpha} \in \overline{\Sigma}$. Pe de altă parte, $\delta_0(y) = 0$ pentru că $W \cap Y \neq \emptyset$, $\delta_0(x) = 1$ pentru că $W \cap X^+ = \emptyset$. Fie $T = U - XY$. Rezultă că există $C \in T$, $C \in W$, deci $\delta_0(c) = 0$, de unde $\delta_0(t) = 0$. Astfel, $\delta_0(x \Rightarrow y + t) = 0$, adică $\delta_0(\overline{\sigma}) = 0$, ceea ce contrazice ipoteza: $\overline{\Sigma} \models_{\overline{\text{c.l.}}} \overline{\sigma}$. □

În continuare vom da demonstraţia sintactică a teoremei de echivalenţă.

Teorema 1.3 (Teorema de echivalenţă). *Fie Σ o mulţime de dependenţe funcţionale sau multivaluate şi σ o dependenţă funcţională sau multivaluată. Atunci sunt echivalente următoarele afirmaţii:*

a) σ este o consecinţă a lui Σ.
b) σ este o consecinţă a lui Σ pe domeniul relaţiilor cu 2 uple.
c) $\overline{\sigma}$ este o consecinţă logică a lui $\overline{\Sigma}$.

Demonstraţie.
a) implică b) rezultă imediat.
Arătăm că **b) implică c).** Fie b) adevărată şi presupunem c) falsă. Atunci există o asignare δ, astfel încât $\delta(\overline{\alpha}) = 1$ pentru orice $\overline{\alpha} \in \overline{\Sigma}$ şi $\delta(\overline{\sigma}) = 0$.
Considerăm relaţia r cu 2 uple t_1 şi t_2 definite astfel: $t_1[A] = 1$ pentru orice $A \in U, t_2[A] = 1 \;\; iff \;\; \delta(a) = 1$. După lema 2.5 obţinem: r satisface α, orice $\alpha \in \Sigma$ şi r nu satisface σ, ceea ce contrazice b).
Arătăm acum: **c) implică a).** Fie $\overline{\sigma}$ consecinţă logică a lui $\overline{\Sigma}$. După teorema 1.1 rezultă că $\overline{\Sigma}\vdash_{\overline{\mathcal{R}'_{FM}}}\overline{\sigma}$, de unde $\Sigma\vdash_{\overline{\mathcal{R}_{FM}}}\sigma$. Avem atunci $\Sigma \models \sigma$, deoarece regulile din $\mathcal{R}_{FM}$ sunt valide. □

Ca un rezultat important, ce reiese din teoremele 1.2 şi 1.3, vom demonstra teorema de completitudine a regulilor de inferenţă pentru dependenţe funcţionale şi multivaluate ($\mathcal{R}_{FM} = \{$FD1f-FD3f, MVD0f-MVD3f, FD-MVD1f–FD-MVD3f$\}$). Am constatat că regula FD-MVD3f se exprimă prin celelalte, la fel FD-MVD2f se exprimă prin celelalte.

Teorema 1.4 *Fie Σ o mulţime de dependenţe funcţionale sau multivaluate şi σ o dependenţă funcţională sau multivaluată. Atunci următoarele afirmaţii sunt echivalente:*

a) σ este o consecinţă a lui Σ.

b) σ este demonstrabilă în Σ utilizând regulile de inferenţă din $\mathcal{R}_{FM}$.

Adică $\Sigma \models \sigma \;\; iff \;\; \Sigma \mid\!\frac{}{\mathcal{R}_{FM}} \sigma$.

Demonstraţie. Fie $\mathcal{R}'_{FM}$ toate regulile din $\mathcal{R}_{FM}$ cu excepţia regulii FD-MVD3f. Avem: $\Sigma \mid\!\frac{}{\mathcal{R}_{FM}} \sigma$ iff $\Sigma \mid\!\frac{}{\mathcal{R}'_{FM}} \sigma$. De asemenea avem: $\Sigma \mid\!\frac{}{\mathcal{R}'_{FM}} \sigma$ iff $\overline{\Sigma} \models_{\overline{\text{c.l.}}} \overline{\sigma}$ (teorema 1.1) şi $\overline{\Sigma} \models_{\overline{\text{c.l.}}} \overline{\sigma}$ iff $\Sigma \models \sigma$ (teorema 1.3). $\square$

În continuare, dorim să demonstrăm semantic teorema de echivalenţă (teorema 1.3). Pentru aceasta avem nevoie de un rezultat preliminar, care pentru o relaţie ce satisface o mulţime dată de dependenţe Σ (funcţionale sau multivaluate) şi nu satisface o dependenţă dată σ, construieşte o subrelaţie a ei formată din 2 uple, subrelaţie ce satisface aceleaşi proprietăţi.

Lema 1.6 (Lema subrelaţiei cu 2 uple). *Fie r o relaţie, Σ o mulţime de dependenţe (funcţionale sau multivaluate) şi σ o singură dependenţă. Presupunem că r satisface toate elementele lui Σ şi nu satisface σ. Atunci există o subrelaţie T în r cu 2 uple, ce satisface toate elementele lui Σ şi nu satisface σ.*

Demonstraţie. Vom considera situaţiile când σ este o dependenţă funcţională sau multivaluată.

I) Dependenţa σ este funcţională. Fie $\sigma : X \to Y$. Avem r nu satisface $\sigma \;\; iff$ există $A \in Y$, astfel încât r nu satisface $X \to A$. În continuare vom putea considera σ de forma $X \to A$. Există atunci două uple în r, fie acestea t_1 şi t_2, astfel încât $t_1[X] = t_2[X]$ şi $t_1[A] \neq t_2[A]$. Considerăm mulţimea M_2 a tuturor relaţiilor cu 2 uple din r, astfel încât acestea nu satisfac $X \to A$. După cele de mai sus, rezultă că $M_2 \neq \emptyset$. Din M_2 considerăm acea subrelaţie din r cu 2 uple ce îndeplineşte activ un număr maxim de dependenţe multivaluate. Fie aceasta notată cu T. Deoarece numărul de dependenţe multivaluate este finit, rezultă că T există. Dorim să arătăm că T satisface toate elementele lui Σ. Fie $\alpha \in \Sigma$; dacă α este dependenţă funcţională, atunci faptul că r satisface Σ implică r satisface α; rezultă că orice subrelaţie din r satisface α, adică T satisface α. Fie $\alpha \in \Sigma$,

α dependenţă multivaluată, $\alpha : U_1 \rightarrow\rightarrow V$. Presupunem că T nu satisface $U_1 \rightarrow\rightarrow V$. Deoarece o relaţie satisface $U_1 \rightarrow\rightarrow V$ dacă şi numai dacă ea satisface $U_1 \rightarrow\rightarrow V - U_1$, rezultă că putem considera U_1 şi V disjuncte. Fie $W = U - U_1V$. Obţinem de aici: $W \neq \emptyset$, căci altfel am avea: $U_1 \rightarrow\rightarrow W$ satisfăcută de orice relaţie şi după complementariere se obţine $U_1 \rightarrow\rightarrow V$ satisfăcută de orice relaţie, deci de T (contradicţie). Fie T formată din uplele t_1 şi t_2. Deoarece T nu satisface $U_1 \rightarrow\rightarrow V$, rezultă că $t_1[U_1] = t_2[U_1]$. Fie (u, v, w) tripletul obţinut prin proiecţia lui t_1 pe U_1, V respectiv W; similar fie (u, v', w') proiecţia lui t_2 pe U_1, V, W. Rezultă $v \neq v'$ şi $w \neq w'$, pentru că altfel am avea faptul că T satisface activ $U_1 \rightarrow\rightarrow V$, deci T satisface $U_1 \rightarrow\rightarrow V$. Deoarece T nu satisface $X \rightarrow A$, avem $t_1[A] \neq t_2[A]$, deci $A \in V$ sau $A \in W$. Fie $A \in V$ (în mod similar se tratează cazul $A \in W$).

Fie T' relaţia din 2 uple constând din $(u, v, w) = t_1$ şi $(u, v', w) = t'_2$. Din faptul că $t_1, t_2 \in r$ şi r satisface $U_1 \rightarrow\rightarrow V$ rezultă că $t'_2 \in r$, deci T' este o subrelaţie în r cu 2 uple. Deoarece t_1 şi t_2 concordă pe X, rezultă că t_1 şi t'_2 concordă, de asemenea, pe X. Din $A \in V$ şi $t_1[A] \neq t_2[A]$, obţinem $t_1[A] \neq t'_2[A]$. Astfel $T' \in M_2$. Mai mult, după lema 1.1, T' satisface activ $U_1 \rightarrow\rightarrow V$. După lema 1.2, T' satisface activ orice dependenţă multivaluată satisfăcută de T. Deoarece T nu satisface activ $U_1 \rightarrow\rightarrow V$ (altfel ar satisface $U_1 \rightarrow\rightarrow V$), rezultă că T' satisface activ un număr mai mare de dependenţe multivaluate decât T şi $T' \in M_2$, ceea ce contrazice alegerea lui T. Contradicţia obţinută justifică faptul că T satisface $U_1 \rightarrow\rightarrow V$, dependenţă multivaluată oarecare din Σ.

II) Dependenţa σ este multivaluată. Fie $\sigma : X \rightarrow\rightarrow Y$. Ca şi mai sus, putem presupune că: $X \cap Y = \emptyset$. Fie $Z = U - XY$. Spunem că perechea de uple (x, y, z), (x, y', z') constituie martori ai eşecului dependenţei $X \rightarrow\rightarrow Y$ într-o relaţie dată r, dacă ele aparţin lui r şi (x, y', z) sau (x, y, z') nu aparţine lui r. Astfel, r nu satisface $X \rightarrow\rightarrow Y$ dacă şi numai dacă r conţine 2- uple ce sunt martori ai eşecului dependenţei $X \rightarrow\rightarrow Y$ în r. Deoarece prin ipoteză avem faptul că r nu satisface σ, există două uple ce constituie martorii eşecului lui σ în r, fie acestea $t_1 = (x, y, z)$ şi $t_2 = (x, y', z')$. Avem deci $t_1, t_2 \in r$ şi $t_3 = (x, y', z) \notin r$ sau $t_4 = (x, y, z') \notin r$. Fie M_2 mulţimea tuturor subrelaţiilor lui r formate din 2 uple, ce constituie martorii eşecului lui σ în r. Considerăm T din M_2 ce satisface activ un număr maxim de dependenţe multivaluate. ($M_2 \neq \emptyset$ şi numărul de dependenţe multivaluate este finit). Arătăm că T satisface toate dependenţele din Σ. Fie $T = \{t_1, t_2\}, t_1 = (x, y, z), t_2 = (x, y', z')$. Fie $\alpha \in \Sigma$. Dacă α este dependenţă funcţională, deoarece r satisface Σ, rezultă r satisface α, deci orice subrelaţie din r satisface α, de unde se obţine că T satisface α. Fie acum α dependenţă multivaluată din Σ, $\alpha : U_1 \twoheadrightarrow V$. Ca şi mai sus, putem presupune că $U_1 \cap V = \emptyset$ şi $V \neq \emptyset$ şi $W = U - U_1V \neq \emptyset$. Fie T formată din t_1 şi t_2.

Presupunem că T nu satisface α. Rezultă atunci $t_1[U_1] = t_2[U_1]$. Fie $\overline{V} = \{C|C \in V, t_1[C] \neq t_2[C]\}$ şi $\overline{W} = \{D|D \in W, t_1[D] \neq t_2[D]\}$. Deci $\overline{V}$ este formată din toate coloanele lui V pentru care t_1 şi t_2 diferă; similar $\overline{W}$. Deoarece T nu satisface $U_1 \twoheadrightarrow V$ rezultă că $\overline{V} \neq \emptyset$ şi $\overline{W} \neq \emptyset$, pentru că altfel T ar satisface activ $U_1 \twoheadrightarrow V$ (lema 1.1) şi deci ar satisface $U_1 \twoheadrightarrow V$. Considerăm proiecţiile lui t_1 pe U_1, V, W ca fiind (u, v, w), iar proiecţiile lui t_2 pe aceleaşi: (u, v', w'). Fie T_1 relaţia cu 2 uple formată din t_1 şi (u, v', w) şi T_2 relaţia cu 2 uple formată din t_1 şi (u, v, w'). Deoarece t_1 şi $t_2 \in r$ şi r satisface α, rezultă T_1, T_2 subrelaţii în r. Deoarece $v \neq v'$ şi $w \neq w'$, rezultă că T_1 şi T_2 au efectiv câte 2 uple. După lema 1.1, avem: T_1 şi T_2 satisfac activ $U_1 \twoheadrightarrow V$. După lema 1.2, orice dependenţă multivaluată satisfăcută activ de T, este, de asemenea, satisfăcută activ de T_1 şi de T_2. În privinţa relaţiilor T_1, T_2 şi a dependenţei $\sigma : X \twoheadrightarrow Y$ avem 2 situaţii:

a) T_1 sau T_2 nu satisface σ, sau

b) T_1 şi T_2 satisfac σ.

În situaţia a), fie T_1 nu satisface σ. Atunci uplele lui T_1 constituie martori ai eşecului lui σ în $r(T_1 \subseteq r)$. După definiţia mulţimii M_2, rezultă $T_1 \in M_2$. Avem: T nu satisface $U_1 \twoheadrightarrow V$ şi T_1 satisface $U_1 \twoheadrightarrow V$. În plus, orice dependenţă multivaluată satisfăcută activ de T este, de asemenea, satisfăcută activ de T_1. Rezultă că $T_1 \in M_2$ şi satisface mai multe dependenţe multivaluate decât T, ceea ce este o contradicţie conform definirii lui T. În mod similar, dacă T_2 nu satisface σ, rezultă o contradicţie.

Fie acum situaţia b): T_1 şi T_2 satisfac σ. Deoarece uplele lui T_1 coincid pe X şi la fel uplele lui T_2, rezultă că T_1 şi T_2 satisfac activ σ. Aplicând lema 1.1, obţinem: cele două uple ale lui T_1 concordă pe Y sau pe Z. Dacă ele concordă pe Y, atunci $\overline{V} \subseteq Z$,pentru că $\overline{V}$ conţine exact coloanele în care cele două uple ale lui T_1 nu concordă. Dacă uplele lui T_1 concordă pe Z, atunci $\overline{V} \subseteq Y$.

Similar, doarece T_2 satisface activ σ, obţinem: $\overline{W} \subseteq Y$ sau $\overline{W} \subseteq Z$.

Astfel distingem 4 cazuri:

1) $\overline{V} \subseteq Y$ şi $\overline{W} \subseteq Y$

2) $\overline{V} \subseteq Y$ şi $\overline{W} \subseteq Z$

3) $\overline{V} \subseteq Z$ şi $\overline{W} \subseteq Y$

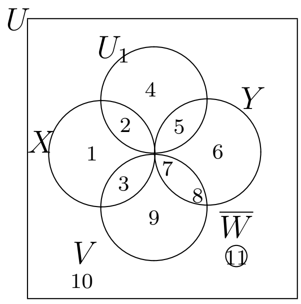
4) $\overline{V} \subseteq Z$ şi $\overline{W} \subseteq Z$

Mulţimea $\overline{V}\ \overline{W}$ reprezintă toate coloanele în care cele două uple din T diferă.

În cazul 1), rezultă că cele două uple din T concordă pe Z, deci T satisface $X \to\to Y$, ceea ce este o contradicţie.

Similar cazul 4) conduce la o contradicţie.

Fie cazul 2): $\overline{V} \subseteq Y$ şi $\overline{W} \subseteq Z$. Uplele t_1 şi t_2 ale lui T diferă numai pe $\overline{V}\ \overline{W}$, în rest ele coincid. Să reprezentăm mulţimile de atribute $U_1, V, W, X, Y, Z, \overline{V}, \overline{W}$ astfel:



Avem: $X = \{1, 2, 3\}$
$Y = \{5, 6, 7, 8\}$
$Z = \{4, 9, 10, 11\}$
$U_1 = \{2, 4, 5\}$
$V = \{3, 7, 8, 9\}$
$W = \{1, 6, 10, 11\}$
$\overline{V} = \{8\}$
$\overline{W} = \{11\}$

Pentru t_i, $i = 1, 2$ să notăm prin t_{ij} proiecţia lui t_i pe domeniul j, $j = \overline{1, 11}$. Avem $t_{1j} = t_{2j}$ pentru $j \in \{1, 2, \ldots, 11\} - \{8, 11\}$ şi $t_{1,8} \neq t_{2,8}$, $t_{1,11} \neq t_{2,11}$. Fie $t_3 = (x, y', z)$ şi $t_4 = (u, v', w)$. Amintim faptul că $t_1 = (x, y, z)$ şi $t_2 = (x, y', z')$. Rezultă că $t_3[11] = t_1[11]$, $t_3[8] = t_2[8]$ şi $t_4[11] = t_1[11]$, $t_4[8] = t_2[8]$. Pentru celelalte domenii, t_3 şi t_4 coincid: $t_3[j] = t_4[j]$, $j \in \{1, 2, \ldots, 11\} - \{8, 11\}$, deoarece acestea coincid cu t_{1j} sau t_{2j} care sunt egale. Obţinem astfel $(x, y', z) = (u, v', w)$. În mod similar se obţine $(x, y, z') = (u, v, w')$. Dar (u, v', w) şi $(u, v, w') \in r$, ceea ce implică $(x, y', z) \in r$ şi $(x, y, z') \in r$, ceea ce este o contradicţie.

În concluzie, T, subrelaţie aleasă din r, satisface orice dependenţă α din Σ. □

Vom da în continuare demonstraţia semantică a teoremei de echivalenţă:

Teorema 1.5 (Teorema de echivalenţă). *Următoarele afirmaţii sunt echivalente:*

a) σ este o consecinţă a lui Σ;

b) σ este o consecinţă a lui Σ pe mulţimea relaţiilor cu două uple;

c) $\overline{\sigma}$ este o consecinţă logică a lui $\overline{\Sigma}$.

Demonstraţie. Aplicând lema 1.6 obţinem că a) şi b) sunt echivalente. Raţionând ca în demonstraţia teoremei 1.3, obţinem b) implică c) şi c) implică b), ceea ce înseamnă b) echivalent cu c), deci a), b), c) sunt echivalente. □

Referinţe bibliografice: [4], [7], [9], [19], [23], [28], [32], [39], [45], [47], [49], [51], [52], [54]

References

[1] Aho, A.V., Beeri, C., Ullman, J.: *The theory of joins in relational databases*, ACM Trans. on Database Systems, 4(1979) 297-314.

[2] Armstrong, W. W.: *Dependency structures of database relationships*, Proc. IFIP 74, North-Holland, Amsterdam, 1974, 580-583.

[3] Armstrong, W. W., Delobel, C.: *Decompositions and functional dependencies in relations*, ACM Trans. Database Systems 5,4 (1980), 404-430.

[4] Beeri, C., Fagin, R., Howard, J. H.: *A complete axiomatization for functional and multivalued dependencies in database relations*, Proc. ACM SIGMOD Conf. on Management of Data, 1977, Toronto, 47-61.

[5] Beeri, C., Vardi, M. Y.: *A proof procedure for data dependencies*, JACM 31(1984), 718-741.

[6] Beeri, C., Bernstein, P.: *Computational Problems Related to the Design of Normal Form Relational Schemas*, ACM Trans. on Database Systems, vol. 4, nr. 1, 1979.

[7] Beeri, C.: *On the membership problem for functional and multivalued dependencies in relational databases*, ACM Trans. Database Systems, 5, 3(1980), 241-259.

[8] Bernstein, P. A., Chiu, D. W.: *Using semi-joins to solve relational queries*, JACM 28(1981), 25-40.

[9] Biskup, J.: *On the complementation rule for multivalued dependencies in database relations*, Acta Informatica, 10, 3(1978), 297-305.

[10] Biskup, J.: *Inferrences of multivalued dependencies in fixed and undetermined universes*, Theoretical Comp. Sc., 10, 1(1980), 93-105.

[11] Buszkowski, W., Orlowska, E.: *On the logic of database dependencies*, Bull. Polish Academy of Sciences, vol. 34, 5-6, 345-354.

[12] Casanova, M. A., Fagin, R., Papadimitriou, C.: *Inclusion dependencies and their interaction with functional dependencies*, J. Computer and System Sciences 28(1984), 29-59.

[13] Chang, C.L., Lee, R.C.T.: *Symbolic Logic and mechanical theorem proving*, Academic Press, New York, 1973.

[14] Codd, E.F.: *A relational model for large shared data banks*, Comm. ACM 13, 6(1970), 377-387.

[15] Date, C.J.: *An Introduction to Database Systems*, Addison-Wesley, Reading, Mass, 1977.

[16] P. De Bra, Paredaens, J.: *Conditional dependencies for horizontal decompositions*, Lecture Notes in Comp. Sc. 154, Springer-Verlag, 1983, 67-82.

[17] Delobel, C., Casey, R.G.: *Decompositions of a database and the theory of Boolean switching functions*, IBM J.Res. Dev. 17, 5(1973), 374-386.

[18] Delobel, C., Adiba, M.: *Relational database systems*, North-Holland, Amsterdam, 1985.

[19] Fagin, R., Vardi, M.: *The theory of Data dependencies-A survey*, Proc. of Symposia in Applied Mathematics, vol.34, 1986, 19-71.

[20] Fagin, R.: *A normal form for relational database that is based on domains and keys*, ACM Trans. on Database Systems 6(1981), 387-415.

[21] Fagin, R.: *Armstrong databases*, Proc. 7^{th} IBM Symp. on Math. Foundations of Comp. Sc., Kanagawa, Japan, May, 1982.

[22] Fagin, R.: *Horn Clauses and database dependencies*, JACM, 29(1982), 952-985.

[23] Fagin, R.: *Multivalued dependencies and a new normal form for relational databases*, ACM. Trans. Database Systems 2, 3(1977), 262-278.

[24] Fagin, R. : *Functional dependencies in a relational database and propositional logic* , IBM J. Res. Dev. 21, 6(1977), 534-544.

[25] Fagin, R. : *Normal forms and relational database operators*, Proc. ACM-SIGMOD Int. Conf. on Management of Data, Boston, Mass. May 1979, 153-160.

[26] Felea, V. : *On the Family of Conditional Implicational Dependencies* ,in Fundamenta Informaticae, vol.24/3, 95, pag. 909-919.

[27] Felea, V. : *On the Family of Conditional Generalized Dependencies* , în Analele Univ. Iasi, vol 8,1999.

[28] Galil, Z. : *An almost linear-time algorithm for computing a dependency basis in a relational database*, JACM 29(1982) 96-102.

[29] Gallaire, N., Minker, J., Nicolas, J.M. : *Logic and databases, a deductive aproach*, Computing Surveys, 16, iunie 1984, 153-185.

[30] Ginsburg, S., Zaiddan, S.M. : *Properties of functional dependency families*, JACM 29(1982), 678-698.

[31] Grant, J., Jacobs, B. E. : *On the family of generalized dependency constraints*, JACM 29(82), 986-997.

[32] Hagihara, K., Ito, M., Taniguchi, K., Kasami, T.: *Decision problems for multivalued dependencies in relational databases*, SIAM J. Computing 8(1979), 247-264.

[33] Honeyman, P. : *Testing satisfaction of functional dependencies*, JACM 29(1982), 668-677.

[34] Hull, R. : *Finitely specificable implicational dependency families*, JACM 31(1984), 210-226.

[35] Jacobs, B. : *On database logic*, JACM, 29,2, 1982, 310-332.

[36] Lewis, H. : *Complexity results for classes of quantificational formulas*, J. Computer and S. Sc. 21(1980) 317-353.

[37] Liu, L., Demers, A.: *An algorithm for testing lossless join property in relational databases*, Inf. Processing Letters ,11(1980),73-76.

[38] Lungu, I., Muşat, N., Roşca, I., Sabău, Gh.:*Baze de date relaţionale, utilizarea lui SQL-PLUS*, Editura ALL, Bucureşti 1992.

[39] Maier, D.: *The Theory of Relational Databases, Computer Science Press*, Rockville, Maryland, 1983.

[40] Maier, D., Mendelzon, A.O., Sadri, F., Ullman, J.D.: *Adequacy of decompositions of relational databases*,J.Computer and Systems Science, 21(1980),368-379.

[41] Maier, D., Mendelzon, A., Sagiv, Y.:*Testing implication of data dependencies*, ACM Trans on Database Systems, 4(1979), 455- 469.

[42] Mitchell, J.C.:*Inference rules for functional and inclusion dependencies*, Proc. 2nd ACM SIGACT-SIGMOD Symp on Principles of Database Systems, 1983, Atlanta, 58-69.

[43] Mitchell, J.C.:*The implication problem for functional and inclusion dependencies*, Information and Control 56(1983), 154- 173.

[44] Papadimitriou, C.:*The theory of database concurency control*, Computer Science Press, Rockville(MD), 1986.

[45] Rissanen, J.:*Theory of relations for databases - a tutorial survey*. Proc. 7th Symp. on Math. Found. of Comp. Sc., Lecture Notes in CS, 64, Springer-Verlag, 1978, 536-551.

[46] Sadri, F., Ullman, J.D.:*Template dependencies:A large class of dependencies in relational databases and their complete aximatization*, JACM 29(1981), 363-372.

[47] Sagiv, Y.:*An algorithm for inferring multivalued dependencies with an application to propositional logic*, JACM 27(1980), 250-262.

[48] Sagiv, Y., Walecka, S.: *Subset dependencies and a completeness result for a subclass of embedded multivalued dependencies*, JACM 29(1982), 103-117.

[49] Sagiv, Y., Delobel, C., D. Stott Parker, Fagin, R.: *An equivalence Between Relational Database Dependencies and a Fragment of Propositional Logic*, J. of Ass. for Computing Mach. vol. 28, nr. 3, 1981, 435-453.

[50] Spyratos, N.:*A homomorphism theorem for database mappings*, Inf.Proc.Letters, 15,11, oct. 1982, 91-96.

[51] Thalheim, B.: *Dependencies in Relational Databases*, Teubner-Texte zur Mathematik, Band 126, B.G. Teubner Verlagsgesellschaft Stuttgart-Leipzig, 1991.

[52] Ullman, J.D.: *Principles of Database Systems, Computer Science Press*, Rockville, Maryland(1982).

[53] Vardi, M.Y.: *The implication and finite implication problems for typed template dependencies*, J. Computer and System Sc. 28(1984), 3-28.

[54] Vardi, M.Y.:*Inferring multivalued dependencies from functional and join dependencies*, Acta Informatica, 19(1983), 305-324.

[55] Yannakakis, M., Papadimitriou C.: *Algebraic dependencies*,J. Computer and System Sciences 25(1982), 3-41.

[56] Lia Chiorean:*FOXPRO. Comenzi si functii.* Editura Microinformatica, 1994.

[57] Maria Boldea, Ioan Boldea:*Programarea in FOXPRO pentru Windows*, Editura didactica si pedagogica, Bucuresti, 1999.

[58] Gabriel Dima, Mihai Dima:*Foxpro*, Editura Teora, 1993.

[59] Gabriel Dima, Mihai Dima:*Foxpro 2.6 sub Windows*, Editura Teora, Bucuresti, 1996.

[60] Gabriel Dima, Mihai Dima:*Microsoft Visual FoxPro 7.0*, Editura Teora, Bucuresti, 2002.

[61] Ion Lungu s.a.*Sistemul Foxpro - Prezentare si aplicatii*, Editura ALL Bucuresti, 1993.

CUPRINS