RELAȚII ȘI BAZE DE DATE

mat. Brînzaru Alina Monica

Institutul de Cercetări în Informatică

Rezumat: Lucrarea își propune prezentarea într-o formă structurată a unor concepte utilizate în lucrul cu baze de date.Se propune și un model de a extinde operațiile algebrei relaționale la operații cu mulțimi fuzzy, având în vedere că mulțimea atributelor asociate unei tabele poate fi considerată a fi o mulțime fuzzy.

Cuvinte cheie: bază de date, sistem de gestiune a unei baze de date, schemă de baze de date, tabelă a unei scheme de baze de date, model relațional, model relațional de date fuzzy.

Necesitatea manipulării eficiente a unor mari cantități de date a generat un interes în dezvoltarea unui mod special de sisteme de programare cunoscute ca sisteme de gestiune a bazelor de date - S.G.B.D. (D.B.M.S.-Database Management System). Pentru a obține eficacitate în gestionarea bazelor de date și pentru a permite utilizatorului să conceptualizeze organizarea bazelor de date, datele trebuie structurate după un anume model conceptual de reprezentare. În mod curent, cel mai important model conceptual este modelul relațional, care a apărut în articolele lui Codd în anii 1970.

Într-un sistem de baze de date relațional, datele sunt structurate în tabele, care constituie o reprezentare convenabilă a relațiilor finite. Dacă nu apar alte specificații, se va presupune, în această secțiune, că toate relațiile sunt finite.

Fie $\{D_{i1},D_{i2},...,D_{in}\}$ o colecție de n mulțimi finite.

Orice relație n-ară $\rho\subseteq\prod\{D_i \mid i\in\{i_1,i_2,...,i_n\}\}$, $\rho=\{(x_1,x_2,...x_n),...,(z_1,z_2,...,z_n)\}$ poate fi reprezentată grafic printr-o tabelă τ: numele acestei tabele este un cuvânt m dintr-un alfabet oarecare V. Presupunem că V conține cel puțin literele mici și mari ale alfabetului latin și simboluri speciale ca: $\cup,\cap,—,x,[],|\ |,—$.

În plus, considerăm în V orice simboluri cu indici de care avem nevoie într-un anume context.

Considerăm V-alfabet de bază.

Indicii $i_1, i_2,...,i_n$ ai n-mulțimilor (mulțimi n-are) implicate în produsul cartezian sunt atributele tabelei . Considerăm că aceşti indici sunt cuvinte în V.

În cuprinsul expunerii, vom cere ca atributele oricărei tabele să fie două câte două distincte.

Mulțimea notată D_{im} este domeniul atributului i_m, $\leq m\leq n$. Notăm $D_{im}=Dom(i_m)$.

Definiție: O schemă de baze de date este o pereche $S=(I, \ )$ unde $I\subseteq V^*$ este mulțimea atributelor i $=\{D_i \mid i\in I\}$ este o colecție de mulțimi indexate după mulțimea I.

Mulțimea D_i este domeniul atributului i.

Pe parcursul lucrului cu o bază de date, pot adăuga sau elimina atribute din schemă.

Definiție: O tabelă a unei scheme de baze de date $S=(I, \ )$ este un triplet $\tau=(m,H,\rho)$ unde $m\in V^*$ este numele tabelei, $H=\{i_1,i_2,...,i_n\}\subseteq I$ este schema tabelei sau capul tabelei și ρ este o relație, $\rho\subseteq\prod_{i\in H} D_i$

În acord cu notațiile practicate în teoria bazelor de date, notăm schema tabelei τ prin $sch(\tau)=i_1i_2...i_n$. De asemenea, dacă $\tau=(m,H,\rho)$, ne referim la $sch(\tau)$ ca schema relației ρ, și folosim notația $sch(\rho)$ în loc de $sch(\tau)$.

O tabelă τ poate fi privită ca o mulțime de șiruri având propriul nume, a cărei structură este descrisă prin același cap.

Dacă $\tau=(m,H,\rho)$, relația ρ va fi notată ca $\rho(\tau)$.

Fie $S=(I, \ )$ o schemă de baze de date.

Definiție: O bază de date relațională este o colecție finită D de tabele, $D=\{\tau_1,...,\tau_p\}$ peste schema S. Schema S a bazei de date D este notată prin sch(D).

Exemplu: Considerăm baza de date a departamentului unui colegiu D_{dept} peste schema:

S=({C#,CNUME,ANSTUDIU,NRSALA, PNUME,PBIROU , SNUME,. SCOD, GRAD},

$\{D_{C\#},D_{CNUME},D_{PONDERI},D_{NRSALA},D_{PNUME},D_{PBIROU},D_{SNUME},D_{SCOD},D_{GRAD}\}$).

Baza de date se compune din tabelele prezentate mai jos. Am reprezentat o fotografie a datelor actuale, conținute de baze de date; conținutul actual al bazei de date variind în timp, datorită ștergerilor, inserțiilor sau a actualizărilor efectuate asupra înregistrărilor.

CURSURI

C#	CNUME	ANSTUDIU	NRSALA
110	Bazele informaticii	4	200
210	Structuri de date	4	209
240	Proiectarea algoritmilor	3	403
310	Teoria grafurilor	3	315
320	Teoria corpurilor	3	402
440	Sisteme de operare	3	317

PROFESORI

PNUME	C#	PBIROU
POPA	110	S556
ICHIM	240	S173
MIHAI	210	S220
SOARE	310	S556
MANEA	440	S230
ION	310	S041

STUDENŢI

C#	SNUME	SCOD	SERIE B.I.
110	Popescu Maria	2341	AF30
110	Ionescu Claudia	5641	CR68
110	Georgescu Irina	9008	AR21
110	Cosma Adrian	2341	FR43
110	Albu Sorin	4430	BE41
210	Mihailescu Stefan	2345	CU50
210	Olteanu Paul	7655	DH65
310	Badea Ionut	0988	AT70
310	Cornea Adrian	5448	BM53

Fie τ_1,τ_2,τ_3 tabelele numite CURSURI, PROFESORI, STUDENŢI respectiv. Avem:

$sch(\tau_1)$=C#CNUME ANSTUDIU NRSALA ,

$sch(\tau_2)$=PNUME C# BIROU.

$sch(\tau_3)$=C# SNUME SCOD SERIE B.I.

Desigur, $D_{c\#}$=Dom(C#) se compune din numerele de curs ale tuturor cursurilor predate la acel colegiu. În acelaşi fel, D_{pnume}=Dom(PNUME) conţine numele tuturor profesorilor afiliaţi la colegiu.

Fiecare tabelă are un nume unic. Introducem o operaţie ce permite crearea unui alias al tabelei cu o posibilă redenumire a atributelor; în acord cu unicitatea numelor, presupunem că la fiecare moment al creării unei noi tabele sau al definirii unui alias al unei tabele avem un nume nou, diferit de toate numele utiiizate până în prezent.

Fie $\tau=(m,H,\rho)$ o tabelă, unde $sch(\tau)=H$,$H=i_1...i_n$.

Presupunem că H' este o altă mulţime de atribute şi $H=j_1...j_n$ astfel ca $Dom(i_p)=Dom(j_p)$ pentru $1\le p\le n$.

Definiţie: Tabela $\tau'=(m',H',\rho)$ este obţinută prin redenumire din tabela $\tau=(m,H,\rho)$ dacă ele reprezintă aceeaşi relaţie ρ , în timp ce atributul j_p înlocuieşte i_p în τ' pentru $1\le p\le n$.

Indicăm definiţia tabelei τ' prin $m'(j_1...j_n):= m$.

Dacă H=H', tabela τ' este obţinută schimbând numele tabelei τ cu τ'.Definiţia lui τ' este indicată în acest caz prin m':= m.

Exemplu: Considerăm tabela numită PROFESORI1, unde

PROFESORI1(TNUME CNR SALA) :=PROFESORI.

Tabela PROFESORI1 va fi obţinută din tabela numită PROFESORI din exemplul 1.

PROFESORI1

TNUME	CNR	SALA
AVRAM	110	S556
MARIN	240	S173
IORDACHE	210	S220
PREDA	310	S556
POPA	440	S230
ION	310	S041

Codd a introdus o mulţime de operaţii ce se efectuează asupra relaţiilor, cunoscute sub numele de algebră relaţională. Mai târziu, vom defini aceste operaţii şi vom explora relaţiile (legăturile) diferite ce există între acestea. Mai exact, considerăm o mulţime de 8 operaţii: 4 operaţii de "teoria mulţimilor", adică operaţii care utilizează faptul că relaţiile sunt mulţimi de tuple (reuniune, intersecţie, diferenţă, produs) şi 4 operaţii speciale (selecţie, proiecţie, uniune-join şi diviziune). Nu toate operaţiile pe care le introducem sunt independente. Este posibil să exprimăm toate operaţiile algebrei relaţionale peste o mulţime de 5 operaţii (uniune, diferenţă, produs, selecţie, proiecţie).

Operaţiile algebrei relaţionale generează noi relaţii. Conform cu unicitatea numelor pentru tabele, presupunem că la fiecare moment al introducerii unei relaţii dăm un nume nou tabelei reprezentând această relaţie; tabela al cărei nume este m va fi notată prin $\tau(m)$.

Operaţiile unare (selecţia şi proiecţia) au prioritate faţă de toate celelalte operaţii. Nici o altă regulă de prioritate nu este specificată pentru operaţiile algebrei relaţionale.

Definiţie: Relaţiile ρ şi θ sunt compatibile dacă ele sunt submulţimi ale aceluiaşi produs cartezian $\prod_{i\in I} D_i$

Acest concept ne permite să introducem câteva operaţii de teoria mulţimilor. Fie ρ, θ două relaţii compatibile.

Definiție: Reuniunea (intersecția, diferența) relațiilor ρ, θ este relația $\rho \cup \theta$ ($\rho \cap \theta$, $\rho - \theta$ respectiv).

Dacă $\tau=(m,H,\rho)$, $\tau'=(m',H,\theta)$ sunt tabele reprezentând relațiile ρ și θ, atunci relația $\rho\omega\theta$ va fi reprezentată prin tabela $(m\omega m', H, \rho\omega\theta)$, unde $\omega \in \{\cap, \cup, -\}$.

Observăm că reuniunea (intersecția,diferența) a două relații compatibile este compatibilă cu ρ și θ.

Pentru o tabelă $\tau=(m,I,\rho)$, unde $I=i_1...i_n$, considerăm atributele $m.i_1,...,m.i_n$; ne referim la aceste atribute ca atribute m-limitate ale lui I.

Fie ρ, θ două relații reprezentate prin două tabele distincte (m,I,ρ) și (m',J,θ) respectiv, unde $I=i_1...i_p$ și $J=j_1...j_q$.

Considerăm schemele $I^m=m.i_1,...,m.i_p$, $J^{m'}=m'.j_1...m'.j_p$ și observăm că $I^m \cap J^{m'}=\emptyset$ în cazul $m \neq m'$ chiar dacă I și J nu sunt necesar disjuncte.

Presupunem, de asemenea, că $Dom(m.i_k)=Dom(i_k)$ și $Dom(m'.j_l)=Dom(j_l)$ pentru $1 \leq k \leq p$ și $1 \leq l \leq q$, respectiv.

Definiție: Produsul relațiilor ρ, θ este relația $\rho x\theta$, unde $sch(\rho x\theta)=I^m \cup J^{m'}$ și $w \in \rho x\theta$ dacă există $r \in \rho$ și $t \in \theta$ astfel ca $w(m.i_k)=r(i_k)$ și $w(m'.j_h)=t(j_h)$ pentru $i_k \in I, j_h \in J$.

Tabela reprezentând relația $\rho x\theta$ este $\tau=(mxm', I^m \cup J^{m'}, \rho x\theta)$.

Fie $\rho \subseteq \prod_{i \in I} D_i$ o relație pe schema I și presupunem că J este o submulțime a atributelor sale, $J \subseteq \{i_1,...,i_n\}$. Considerăm $c_j: J \to I$ dată de $c_j(i)=i$ pentru toți $i \in J$.

Definiție: Proiecția tuplului $t \in \rho$ este tuplul $Z: J \to \bigcup_{i \in J} D_i$ definit de $Z=t\,c_j$. Noul tuplu va fi notat cu $t[J]$.

Proiecția relației ρ pe mulțimea atributelor J este relația $\rho[J]=\{t[J] \mid t \in \rho\}$.

Dacă $\tau=(m,I,\rho)$ este tabela reprezentând ρ, atunci relația $\rho[j]$ va fi reprezentată prin tabela $\tau'=(m[j],J,\rho[j])$.

Exemplu: Considerăm relația STUDENȚI.

Dorim să eliminăm numele studenților și să considerăm numai numerele cursurilor, codurile studenților și respectiv valorile seriei B.I.

Această operație poate fi realizată prin proiecție:

STUDENȚI [C# SCOD SERIE B.I.].

Proiecția este descrisă în tabela următoare:

C#	SCOD	SERIE B.I.
110	2341	AF30
110	5641	CR68
110	9008	AR21
110	2341	FR43
110	4430	BE41
210	2345	CU50
210	7655	DH65
310	0988	AT70
310	5448	BM53

Fie ρ o relație reprezentată de $\tau=(m,I,\rho)$ și să presupunem că ζ este o relație astfel ca $sch(\zeta)=H$, unde $H=h_1h_2...h_p \subseteq I$.

Definiție: ζ-selecția unei relații ρ este relația:

$sel_\zeta(\rho)=\{t \mid t \in \rho, t[H] \in \zeta\}$

Această relație va fi reprezentată prin $(m_\zeta, I, sel_\zeta(\rho))$.

Considerăm un atribut $i \in I$ și fie $a \in Dom(i)$. Fie ζ o relație având schema $\{i\}$ și să presupunem că $\zeta=\{a\}$. Rezultatul selecției $sel_\zeta(\rho)$ este mulțimea tuturor tuplelor lui ρ care au a-i-a componentă egală cu a, $\{t \mid t \in \rho$ și $t(i)=a\}$.

Această selecție specială va fi notată prin $sel_{i=a}(m)$.

Exemplu: Se presupune că este necesar să determinăm cursurile având valoarea 4 pe câmpul "ANSTUDIU". Aplicăm selecția. Rezultatul selecției $sel_{PONDERI=4}$ (CURSURI) este relația reprezentată prin tabela următoare:

C#	CNUME	PONDERI	NRSALA
110	Bazele informaticii	4	W1100
210	Structuri de date	4	M3210

$sel_{ANSTUDIU=4}$(CURSURI)

Fie ρ o relație astfel ca i,j sunt două atribute pentru care $Dom(i)=Dom(j)=D$. Dacă α este o relație pe D ($sch(\alpha)=ij$), atunci notăm selecția $sel_\alpha(\rho)$ prin $sel_{i\alpha j}(\rho)$.

Exemplu: Presupunem că avem o tabelă ce înregistrează dimensiunea tablourilor unei colecții. Tablourile sunt referite prin numerele lor de inventar INV#, iar câmpurile HD,VD reprezintă dimensiunea orizontală, respectiv verticală.

COLECȚIE

INV#	HD	VD
1221	30	50
1278	40	70
3411	50	40
4670	120	80
5182	80	60

Selecția $sel_{HD>VD}$(COLECTIE) va avea ca rezultat tabela ce conține lucrările a căror dimensiune orizontală este mai mare ca dimensiune verticală:

$sel_{HD>VD}$(COLECȚIE)

INV#	HD	VD
3411	50	40
4670	120	80
5182	80	60

Teoremă: Compunerea selecțiilor este comutativă.

Mai precis, pentru $H,K \subseteq I=sch(\rho)$ avem:

(1) $sel_\zeta(sel_\varepsilon(\rho))=sel_\varepsilon(sel_\zeta(\rho))$ pentru orice relații ζ,ε cu $sch(\zeta)=H$ și $sch(\varepsilon)=K$.

In plus, selecția este distributivă peste reuniune, intersecție și diferență, adică:

$sel_\zeta(\rho \omega \theta)=sel_\zeta(\rho) \omega\ sel_\zeta(\theta)$ unde $\omega \in \{\cap,\cup,-\}$.

Demonstrație: Comutativitatea selecției rezultă din faptul că ambele părți ale ecuației (1) sunt egale cu $\{t \mid t \in \rho, t[H] \in \zeta$ și $t[K] \in \varepsilon\}$.

Demonstrăm distributivitatea selecției asupra diferenței.

Fie $t \in sel_\zeta(\rho - \theta)$. Avem $t \in \rho - \theta$ și $t[H] \in \zeta$. De aceea, $t \in sel_\zeta(\rho)$ și $t \notin sel_\zeta(\theta)$; deci, $t \in sel_\zeta(\rho) - sel_\zeta(\theta)$. Deducem $sel_\zeta(\rho - \theta) \subseteq sel_\zeta(\rho) - sel_\zeta(\theta)$.

Reciproc, fie $t \in sel_\zeta(\rho) - sel_\zeta(\theta)$. Aceasta implică $t \in \rho$ și $t[H] \in \zeta$. Din $t \notin sel_\zeta(\theta)$, rezultă $t \notin \theta$; deci, $t \in \rho - \theta$. Am obținut $t \in sel_\zeta(\rho - \theta)$ și astfel avem incluziunea inversă: $sel_\zeta(\rho - \theta) \supseteq sel_\zeta(\rho) - sel_\zeta(\theta)$, adică egalitatea de care aveam nevoie.

Din $sel_\zeta(sel_\varepsilon(\rho))=sel_\varepsilon(sel_\zeta(\rho))$, notăm valoarea comună a acestor relații prin $sel_{\zeta \text{ and } \varepsilon}(\rho)$.

Exemplu: Considerăm relația COLLECTION introdusă în exemplul 6. Expresia

$sel_{INV\#>2000 \text{ and } HD>VD}$(COLLECTION) desemnează tablourile al căror număr de inventar e >2000 și a căror dimensiune orizontală e > decât dimensiunea verticală.

Fie ρ,θ două relații $\rho \subseteq \prod_{i \in I} D_i$ și $\theta \subseteq \prod_{i \in I} D_i$ reprezentate prin tabelele $\tau=(m,I,\rho)$ și $\tau'=(m',J,\theta)$ respectiv. Presupunem că $I \cap J=K$ și $I \cup J=L$.

Definiție: Operația de uniune (join) poate fi aplicată tuplurilor $t \in \rho$, $s \in \theta$ dacă $t(i)=s(i)$ pentru toți $i \in K$. Join (uniunea, compunerea, asocierea) este tuplul $z \in \prod_{i \in L} Di$, unde

$$z(i)=\begin{cases} t(i) \ daca \ i \in I \\ s(i) \ daca \ i \in J \end{cases}$$

Join-ul relațiilor ρ și θ relația $\rho \bowtie \theta$ peste schema $I \cup J$ ce se compune din toate asocierile tuplurilor compozabile din două relații.

Tabela reprezentând $\rho \bowtie \theta$ este $(m \bowtie m', I \cup J, \rho \bowtie \theta)$.

In termenii proiecției, t,s sunt compozabile dacă $t[k]=s[k]$. Uniunea lor z va satisface atât $z[I]=t[I]$, cât și $z[J]=s[J]$.

Propoziție Uniunea a două relații compatibile coincide cu intersecția lor.

Demonstrație Fie $\rho,\sigma \subseteq \prod_{i \in I} D_i$ două relații compatibile peste schema I și fie $z \in \rho \bowtie \sigma$. Atunci există două tupluri compozabile $t \in \rho, s \in \sigma$ astfel încât $z(i)=t(i)$ și, de asemenea, $z(i)=t(i)$ pentru toți $i \in I$, din definiția uniunii. Aceasta conduce imediat la $z=t=s \in \rho \cap \sigma$.

Reciproc, dacă $u \in \rho \cap \sigma$, atunci u este compozabilă cu ea însăși, și aceasta dă $u \in \rho \bowtie \sigma$.

Uniunea are un număr de proprietăți utile. Menționăm, în continuare, câteva dintre acestea.

Teoremă Operația de join este asociativă, comutativă și idempotentă, adică:

$$\rho \bowtie (\mu \bowtie \theta)=(\rho \bowtie \mu) \bowtie \theta,$$
$$\rho \bowtie \mu=\mu \bowtie \rho, \rho \bowtie \rho=\rho$$

pentru orice relații ρ,μ,θ.

Uniunea este distributivă în raport cu reuniunea și intersecția, adică:

$$\rho \bowtie (\mu \cup \theta)=(\rho \bowtie \mu) \cup (\rho \bowtie \theta)$$
$$\rho \bowtie (\mu \cap \theta)=(\rho \bowtie \mu) \cap (\rho \bowtie \theta)$$

Demonstrație: Vom demonstra asociativitatea compunerii. Celelalte proprietăți se vor demonstra analog.

Înainte de a prezenta argumentația asociativității, observăm că, dacă ρ_1, ρ_2 sunt două relații peste schemele I_1, I_2 respectiv, $t_1 \in \rho_1$ și $t_2 \in \rho_2$ respectiv și $t_1[P]=t_2[P]$ pentru $P \subseteq I_1 \cap I_2$, atunci avem $t_1[Q]=t_2[Q]$ pentru orice $Q \subseteq P$.

De asemenea, $t_1[R_1 \cup R_2]=t_2[R_1 \cup R_2]$ dacă și numai dacă $t_1[R_1]=t_2[R_1]$ și $t_1[R_2]=t_2[R_2]$.

Fie ρ,μ,θ 3 relații peste schemele I,J,K respectiv. Fie $u \in \rho \bowtie (\mu \bowtie \theta)$. Atunci, există două tuple compozabile t,s astfel ca $t \in \rho$ și $s \in \mu \bowtie$

şi u[I]=t[I],u[J∪K]=t[J∪K]. Din observaţia anterioară, avem, de asemenea, u[J]=s[J] şi u[K]=s[K]. Deoarece t, s sunt compozabile deducem t[I∩(J∪K)]=s[I∩(J∪K)], ceea ce implică t[I∩J]=s[I∩J] şi t[I∩K]=s[I∩K].

Din $s\in\mu\bowtie\theta$, există $z\in\mu$ şi $w\in\theta$ astfel încât s[J]=z[J],s[K]=w[K] şi z,w sunt compozabile, adică z[J∩K]=w[J∩K].

Tuplurile t, z sunt compozabile din:

t[I∩J]=u[I∩J]=s[I∩J]=z[I∩J].

Uniunea lor este un tuplu $y\in\rho\bowtie\mu$, şi avem y[I]=t, y[J]=z şi $y\in\rho\bowtie\mu$.

Deoarece y[(I∪J)∩K]=w[(I∪J)∩K], deducem că tuplurile y şi w sunt compozabile. Într-adevăr, y[I∩K]=t[I∩K]=u[I∩K]şi w[I∩K]=s[I∩K]=u[I∩K]; în acelaşi fel, y[J∩K]=z[J∩K]=s[J∩K]=u[J∩K] şi aceasta dă:

y[(I∪J)∩K]=y[(I∩K)∪(J∩K)]=w[(I∩K)∪(J∩K)]=w[(I∪J)∩K]

Fie v uniunea lui y şi w, $v\in(\rho\bowtie\mu)\bowtie\theta$.

Avem:

v[I]=y[I]=t[I]=u[I],

v[J]=y[J]=z[J]=s[J]=u[J], şi

v[K]=w[K]=s[K]=u[K].

De aceea, v[I∪J∪K]=u[I∪J∪K]; deci v=u, ceea ce arată că $u\in(\rho\bowtie\mu)\bowtie\theta$. Am demonstrat incluziunea $\rho\bowtie(\mu\bowtie\theta)\subseteq(\rho\bowtie\mu)\bowtie\theta$. Un argument similar poate fi folosit pentru a demonstra incuziunea invers N.

Fie ρ,θ două relaţii astfel ca sch(θ)⊆sch(ρ). Presupunem că cele două relaţii sunt reprezentate prin tabelele τ=(m,I∪J,ρ) şi τ'=(m',J,θ), respectiv, unde I∩J=∅, $I=i_1\ldots i_n$ şi $J=j_1\ldots j_m$.

Definiţie: Diviziunea (câtul) relaţiilor ρ şi θ este relaţia μ=ρ:—θ,unde sch(μ)=I şi t∈μ dacă $\{t\}\bowtie\theta\subseteq\rho$.

Relaţia μ este reprezentată prin tabela (m:—m',I,μ).

Pentru ca un tuplu t să fie membru în ρ:—θ, t trebuie să fie compozabil cu toate tuplele lui θ, şi rezultatul acestor compuneri trebuie să aparţină relaţiei ρ.

Fie D o bază de date relaţională.

Definiţie: Un calcul (computation) pentru o relaţie ρ în algebra relaţională este un şir de nume de tabele $m_1,\ldots,m_n$ astfel încât există un şir de tabele $\{\tau_i\mid\tau_i=(m_i,I_i,\rho_i),1\le i\le n\}$ şi pentru orice i, $1\le i\le n$, avem $\rho_i\in D$ sau ρ_i este obţinută dintr-un subşir de predecesori ai săi în şir prin redenumire sau prin aplicarea uneia din operaţiile algebrei relaţionale.

Propoziţie: Uniunea a două relaţii poate fi calculată în algebra relaţională folosind operaţiile de produs, selecţie şi proiecţie.

Demonstraţie: Fie D o bază de date relaţională şi fie ρ,σ două relaţii reprezentate prin tabelele (r,I,ρ) şi (s,J,σ), respectiv, unde I∩J=K, $I=i_1\ldots i_p k_1\ldots k_m$, $J=k_1\ldots k_m j_1\ldots j_q$ şi $K=k_1\ldots k_m$.

Considerăm mulţimile $K^r=r.k_1\ldots r.k_m$ şi $K^s=s.k_1\ldots s.k_m$.

Fie ζ relaţia pe $\prod_{k\in K}r_{\cup K}sDom(k)$ definită prin:

$\zeta=\{t\mid t\in\prod_{k\in K}r_{\cup K}s$ Dom(k) şi $t(r.k_n)=t(s.k_n)$ pentru $k_n\in K\}$

Considerăm următorul calcul în algebra relaţională:

r,s, m:=r*s, u:=sel_ζ(m)[r.i_1,....,r.i_p,r.

k_1,...,r.k_m,s.j_1...,s.j_q],

v(i_1,...,i_p,k_1,...,k_m,j_1,...,j_q):=u.

Pentru $t_r=(t_r(i_1),\ldots,t_r(i_p),t_r(k_1),\ldots,t_r(k_m))\in r$ şi $t_s=(t_s(k_1),\ldots,t_s(k_m),t_s(j_1),\ldots,t_s(j_q))\in$ s,tabela m va conţine tuplul: $(t_r(i_1),\ldots,t_r(i_p),t_r(k_1),\ldots,t_r(k_m),t_s(k_1),\ldots,t_s(k_m),t_s(j_1),\ldots,t_s(j_q))$.

Un astfel de tuplu va fi reţinut de selecţia sel_ζ dacă şi numai dacă $t_r(k_1)=t_s(k_1),\ldots,t_r(k_m)=t_s(k_m)$ şi aceasta este exact condiţia de compozabilitate pentru tuplele t_r şi t_s.

Limbajul standard de interogare SQL calculează uniunea a două relaţii în modul descris mai sus.

O discuţie asupra semanticii operaţiilor de restabilire în SQL folosind algebra relaţională poate fi găsită în finalul acestei secţiuni.

Teoremă: Toate operaţiile algebrei relaţionale pot fi exprimate prin 5 operaţii de bază: reuniune, diferenţă, produs, proiecţie şi selecţie.

Algebra relaţională este utilizată ca un standard cu care evaluăm orice sistem de baze de date relaţional.

Un sistem de baze de date este numit complet relaţional, dacă el este capabil să realizeze toate operaţiile algebrei relaţionale. Ultima teoremă arată că pentru a demonstra completitudinea relaţională este suficient să arătăm că cele cinci operaţii fundamentale pot fi implementate.

Un sistem de baze de date este numit complet relaţional dacă el este capabil să realizeze toate operaţiile algebrei relaţionale.Ultima teoremă arată că pentru a demonstra completitudinea relaţională este suficient să arătăm că cele cinci operaţii fundamentale pot fi implementate.

SQL-STRUCTURED QUERY LANGUAGE este un limbaj de manipulare a datelor dezvoltat inițial de IBM pentru un prototip de DBMS numit "System R". IBM și alții au produs un număr de sisteme care suportă SQL, și este de așteptat ca SQL să devină limbajul relațional standard.

Considerăm câteva interogări aplicate bazei de date D_{dept} și calculul care rezolvă aceste interogări.

Exemplu: Query (Interogare) Să se determine numele profesorilor care predau studentului Albert cursuri de 4 ore.

Informațiile cerute de acest query pot fi găsite în trei tabele:

CURSURI (pentru C# și numărul de ore), PROFESORI (pentru asocierea între cursuri și profesori) și STUDENTI (pentru asocierea între cursuri și studenți)

Folosim următorul calcul:

CURSURI;

PROFESORI;

STUDENȚI;

T1:=CURSURI ⋈ PROFESORI ⋈ STUDENȚI;

$T2:=sel_{SNUME='Albert',CREDITS=4}(T1)$;

T3:=T2 [PNUME]

Scopul calculului T1 este de a grupa (prin operația de uniune -join) toate informațiile cerute de query; astfel, în T2, numai tuplele ce cuprind studentul Albert sunt reținute.

Eventual, în T3, proiectăm aceste tuple pe numele profesorilor, și obținem răspunsul dorit ca o tabelă.

Aceeași interogare poate fi rezolvată prin aplicarea produsului; această soluție alternativă prezintă un interes special din punct de vedere al limbajului standard de interogare SQL; poate fi obținută prin următorul calcul:

CURSURI;

PROFESORI;

STUDENȚI;

T1:=CURSURI X PROFESORI X STUDENȚI;

$T2:=sel_{CURSURI.C\#=PROFESORI.C\# \text{ and } PROFESORI.C\#=STUDENTI.C\#}(T1)$;

$T3:=sel_{STUDENTI.SNUME='Albert' \text{ and } CURSURI.ANSTUDIU=4}(T2)$;

T4:=T3 [PROFESORI.PNUME];

Formarea produsului unei relații cu ea însăși este necesară în situațiile ce presupun compararea membrilor aceleiași relații. Această construcție presupune considerarea unor precauții speciale; într-adevăr, am presupus că atributele fiecărei tabele sunt perechi distincte. Dacă $\tau=(m,H,\rho)$ este o tabelă reprezentând relația ρ cu $sch(\rho)=i_1...i_n$ atunci, dacă vom încerca să calculăm $\nu=\rho x\rho$, obținem o tabelă a cărei schemă va fi $m.i_1,...,m.i_n$ $m.i_1,...,m.i_n$, ceea ce nu poate fi compatibil cu presupunerea anterioară.

Soluția pentru această problemă este de a defini un alias $\tau'=(m',I'\rho)$ al tabelei $\tau=(m,I,\rho)$ unde $I\cap I'=\varnothing$, și atunci formăm produsul mxm'.

Considerăm această tehnică în următoarele două exemple:

Exemplu:

Determinați toate perechile de cursuri predate de aceiași profesori.

Considerăm următorul calcul:

PROFESORI;

TA:=PROFESORI [PNUME,C#];

TB:=TA;

T1:=TA X TB;

$T2:=SEL_{TA.C\#=TB.C\#}(T2)$;

T3:=T2 [TA.PNUME,TB.PNUME];

$T4:=sel_{TA.PNUME<TB.PNUME}(T3)$;

Definiția tabelei TA reține acea parte a tabelei PROFESORI care este necesară pentru a rezolva interogarea (query-ul); următoarea, TB, un alias pentru TA, este creată și ea permite calculul produsului exprimat în pasul al treilea.

În T2, noi păstrăm numai acele tuple care se referă la același curs; numele profesorilor sunt selectate în T3. Ultimul pas este utilizat pentru a elimina perechile redundante.

De exemplu, dacă profesorii POPA și ICHIM predau același curs, obținem răspunsul în 4 perechi: (POPA, POPA), (POPA, ICHIM). (ICHIM, POPA) și (ICHIM, ICHIM).

Exemplu: Query: Determinați studenții care urmează toate cursurile predate de profesorul SOARE.

Determinăm inițial toate cursurile predate de profesorul SOARE. După această operație extragem, prin proiecție, numele studenților și cursurile urmate de aceștia din tabela STUDENTI. În final, rezultatul dorit va fi obținut utilizând diviziunea. Aceste operații sunt realizate prin următorul program:

PROFESORI;

$T1:=sel_{PNUME=SMITH}(PROFESORI)$;

T2:=T1 [C#];

STUDENȚI;

T3:=STUDENȚI [C#, SNUME];

T4:=T3:-T2;

SQL este limbajul de interogare standard pentru sisteme de baze de date relaționale. Interogările sunt exprimate prin construcții numite expresii SELECT. Răspunsul returnat de o interogare este o tabelă, și vom prezenta efectul unor astfel de interogări în termenii algebrei relaționale.

SQL conține facilități de definire a datelor ce permit crearea sau eliminarea tabelelor și a altor obiecte înrudite. Utilizatorul poate indica anumite domenii pentru atributele tabelelor alese dintr-o listă de domenii care este specifică pentru fiecare implementare a limbajului SQL. Menționăm mai jos câteva din domeniile disponibile în SQL pentru sistemul INGRES.

vchar(n) - Mulțimea șirurilor ce conțin cel mult 2000 de caractere.
c(n) - Mulțimea șirurilor ce conțin cel mult 255 de caractere ASCII.
float4 - Mulțimea numerelor reale, în simplă precizie.
float8 - Mulțimea numerelor reale, în dublă precizie.
integer1 - Mulțimea numerelor întregi, memorate pe 1 byte.
integer2 - Mulțimea numerelor întregi, memorate pe 2 bytes.

sau smallint
integer4 - Mulțimea numerelor întregi memorate pe 4 bytes.
sau integer

O instrucțiune cum ar fi:

CREATE TABLE CURSURI (C# **INTEGER2,** CNUME **VCHAR(40),**

PONDERI **INTEGER1**, NRSALA **VCHAR(5))**

creează tabela (CURSURI, C# CNUME ANSTUDIU NRSALA, ∅), adică creează o tabelă goală unde utilizatorul poate începe inserarea tuplelor relației. Avem $D_{C\#}$=INTEGER2, D_{CNUME}=VCHAR(40), $D_{ANSTUDIU}$=INTEGER1 și D_{NRSALA}=VCHAR(5)

Pentru a crea tabela $\tau=(m,i_1,...,i_n,\varnothing)$ scriem instrucțiunea:

CREATE TABLE $m(i_1\ d_1,...,i_n\ d_n)$, unde d_k este domeniul pentru i_k pentru $1\le k\le n$.

Forma generală a unei expresii SELECT este:

SELECT A FROM T WHERE C.

Aici, T este o listă $l_1, l_2,...,l_k$ unde l_j poate fi sau un nume de tabelă m_j sau poate defini aliasul $\hat{}m_j$ al tabelei m_j. În ultimul caz, l_j este $l_j=m_j\ \hat{}m_j$,unde cele două nume sunt separate prin spațiu, nu prin virgulă.

A este o listă de atribute modificate numită lista țintă și C este o listă a condițiilor așa cum au fost considerate ca indici ai operației de selecție.

Prezența listei condițiilor C este opțională. Aceasta înseamnă că:

SELECT A FROM T este, de asemenea, o expresie SELECT corectă.

Dacă numai o tabelă este conținută într-o expresie **SELECT** și este necesar să considerăm toate atributele sale în lista țintă, putem înlocui A cu *.

Execuția unei expresii SQL constă, cel puțin din punct de vedere conceptual, din execuția a trei pași:

1. Se calculează produsul $m=l_1x...xl_k$; dacă $l_j=m_j\hat{}m_j$, considerăm în produs aliasul $\hat{}m_j$ în locul tabelei m_j.
2. Se apelează selecția specificată prin C.
3. Se face proiecția pe câmpurile conținute de lista A.

Exemplu: Pentru interogarea discutată în exemplul 8, folosim următoarea expresie SELECT pentru a implementa a doua soluție discutată în acel

SELECT PNUME **FROM** CURSURI, PROFESORI, STUDENTI
WHERE CURSURI.C#=PROFESORI. C# **AND**
PROFESORI.C#=STUDENȚI.C# **AND**
STUDENȚI. SNUME='Albert' **AND**
CURSURI.ANSTUDIU=4

exemplu:

Exemplu: Interogarea din exemplul 9 este rezolvată prin următoarea expresie SELECT:

SELECT PA.C#,PB.C# **FROM** PROFESORI PA, PROFESORI PB
WHERE PA.C#=PB.C#
AND PA.PNUME<PB.PNUME.

Observăm că două aliasuri ale tabelei PROFESORI, PA și PB sunt create în această expresie. Se compară toate liniile noii tabele, și numai acele linii ale produsului PAxPB conținând aceleași numere de curs și nume diferite ale profesorilor (în ordine alfabetică) sunt reținute.

Expresiile SELECT pot apare ca subexpresii SELECT în condiția C a unei alte expresii SELECT. Presupunem că o expresie SELECT închisă între paranteze returnează o mulțime de valori mai degrabă decât o tabelă.

Exemplu: Considerăm următoarea subexpresie SELECT:

...(**SELECT** SNUME **FROM** STUDENȚI **WHERE** C#=110).

Rezultatul acestei expresii va fi mulțimea numelor tuturor studenților înscriși la cursul 110.

Mulțimile generate de subinterogări pot fi implicate în selecții. O condiție ca:

v **IN (SELECT**...) va fi satisfăcută de acele tuple a căror componentă corespunzătoare atributului v va fi un membru al mulțimii generate prin subselecție.

Exemplu: Această tehnică este ilustrată de următoarea expresie:

```
SELECT SNUME FROM STUDENTI
                WHERE C# IN
                (SELECT C# FROM PROFESORI
                WHERE PNUME='Ionescu')
```

Subexpresia SELECT generează mulțimea cursurilor predate de Profesor Ionescu; o linie a tabelei STUDENȚI satisface condiția selecției dacă componenta sa C# este un membru al mulțimii numerelor de curs predate de Profesor Jones, ceea ce înseamnă că expresia principală SELECT va returna numele studenților care au obținut un grad în unul din cursurile predate de profesorul Ionescu.

Subexpresiile SELECT pot fi făcute dependente de o parte a informației transmisă prin apelul unei expresii SELECT.

Exemplu: Această caracteristică a limbajului SQL este ilustrată prin altă soluție a interogării considerată în exemplul anterior.

```
SELECT SNUME FROM STUDENTI
                WHERE 'POPA' IN
(SELECT PNUME FROM PROFESORI
WHERE C#=STUDENȚI.C#).
```

Pentru fiecare student al cărui nume apare în tabela STUDENȚI, calculăm în subexpresia SELECT mulțimea numelor profesorilor care predau cursuri la care studentul este înscris. Numerele cursurilor sunt transmise subexpresiei SELECT de interogarea principală prin parametrul STUDENTI.C# al subexpresiei SELECT.

Exemplu: Presupunem că alegem numele unui student din tabela STUDENȚI și notăm numele acestui student prin STUDENȚI.SNUME. Pentru a determina cursurile la care acest student este înscris, vom opera pe o altă copie STUDENȚIA a tabelei STUDENȚI. Următoarea subexpresie SELECT:

```
(SELECT C# FROM STUDENTI STUDENTIA
                WHERE STUDENȚIA.
                SNUME=STUDENȚI.SNUME)
```

depinde de STUDENTI.SNUME. Soluția acestei interogări este dată de

```
SELECT SNUME FROM STUDENȚI
                WHERE C# IN
                (SELECT C# FROM STUDENȚI
                STUDENȚIA
                WHERE STUDENȚIA.SNUME=
                STUDENȚI.SNUME).
```

Subexpresiile SELECT pot fi testate în SQL dacă sunt sau nu vide prin

NOT EXISTS(SELECT...) sau prin

EXISTS (SELECT...)

Astfel de teste sunt utile în implementarea în SQL a diferitelor operații ale algebrei relaționale.

Exemplu: Determinați numele studenților care nu frecventează nici un curs predat de Profesor POPA. În algebra relațională, putem rezolva această problemă prin următorul calcul:

STUDENȚI;

T1:=STUDENȚI[SNUME];

$T2:=sel_{PNUME='POPA'}(PROFESORI)$;

T3:=STUDENȚI $\bowtie$ T2;

T4:=T3[SNUME];

T5:=T1-T4;

În SQL, această succesiune de operații poate fi realizată impunând condiția ca mulțimea cursurilor urmate de un student cu profesorul POPA să fie vidă:

```
SELECT SNUME FROM STUDENTI
                WHERE NOT EXISTS
                (SELECT SNUME FROM
                STUDENTI STUDENTIA,
                PROFESORI WHERE
                STUDENTIA.SNUME=
                STUDENTI.SNUME
                ANDSTUDENTIA.C#=
                PROFESORI.C# AND
                PROFESORI.PNUME='POPA').
```

Două expresii SQL care returnează aceeași listă scop pot fi implicate într-o operație UNION.

Exemplu: Presupunem că este necesar să determinăm cursurile din anul 4 de studiu sau sunt predate de Profesor SOARE. Aceasta poate fi rezolvată în SQL prin următoarea construcție:

```
SELECT C# FROM CURSURI
        WHERE ANSTUDIU=4
UNION
SELECT C# FROM PROFESORI
        WHERE PNUME='SOARE'.
```

SQL este capabil să realizeze inserția, ștergerea sau reactualizarea tabelelor.Pentru a realiza o inserție într-o tabelă $\tau=(m,i_1,...,i_n,\rho)$ poate fi folosită următoarea comandă INSERT:

INSERT INTO $m(i_1,i_2,....,i_n)$ **VALORILE** $(v_1,v_2,...v_n)$.

Aceasta inserează în tabela m a cărei schemă este $i_1 i_2 ... i_n$ n-tuplul $(v_1,...,v_n)$.

Exemplu: Pentru a insera 4-tuplul (330, BAZE DE DATE, 3,W2310) în tabela CURSURI scriem:

INSERT INTO CURSURI
(C#,CNUME, ANSTUDIU, NRSALA)

VALUES (330,'BAZE DE DATE', 3,'W2310').

Tuplele pot fi adăugate la o tabelă începând cu datele deja existente. Această operație poate fi realizată printr-o operație INSERT având următoarea formă:

INSERT INTO $m(i_1,...,i_n)$

SELECT ...,

unde SELECT extrage date dintr-o tabelă existentă.

Exemplu: Putem crea o tabelă, ce va conține cursurile cu valoarea ponderilor 3 prin

CREATE TABLE C3CR (C# **INTEGER2**,CNUME **VCHAR(40)**
NRSALA **VCHAR(5)).**

Astfel, tripletele corespunzătoare tuturor cursurilor urmate de studenții anului 3 existente în tabela CURSURI pot fi adăugate tabelei C3CR prin:

INSERT INTO C3CR(C#,CNUME,NRSALA)
SELECT C#,CNUME,NRSALA
FROM CURSURI
WHERE ANSTUDIU=3.

Datele pot fi eliminate dintr-o tabelă m utilizând o instrucțiune DELETE. Forma generală a unei astfel de instrucțiuni este:

DELETE FROM m **WHERE C,**

unde C este o condiție ce determină tuplele ce vor fi eliminate. Prezența condiției este opțională; cu alte cuvinte, putem avea o ștergere de forma:

DELETE FROM m.

Efectul acestei instrucțiuni este eliminarea tuturor liniilor tabelei m.

Exemplu: Următorul DELETE poate fi utilizat pentru a șterge tuplele care se referă la studentul al cărui cod este 9008 din tabela STUDENȚI:

DELETE FROM STUDENTI **WHERE** SNUME=9008.

Exemplu: Pentru a elimina din tabela CURSURI toate cursurile predate de profesorii POPA sau SOARE, putem realiza următoarea operație DELETE:

DELETE FROM CURSURI **WHERE**
C# **IN (SELECT C# FROM** PROFESORI
WHERE PNUME='POPA' **OR**
PNUME='SOARE').

Comanda SQL UPDATE modifică valorile în tabelele existente.

Forma generală a unei comenzi UPDATE realizată pe o tabelă $\tau = (m,H,\rho)$ este:

UPDATE m
SET $i_1=e_1,...,i_n=e_n$
WHERE C,

unde C este o condiție opțională, $H=i_1...i_n$ și $e_1,...e_n$ sunt valori sau expresii ce vor fi asignate componentelor corespunzând tuplelor lui m care satisfac condiția C.

Exemplu: Pentru a crește cu 1 valoarea anului de studiu al cursurilor cu numerele de cod ce depășesc nivelul 300, putem executa următorul UPDATE:

UPDATE CURSURI
SET PONDERI=PONDERI+1
WHERE C#>300.

Exemplu: Pentru a schimba gradele tuturor studenților ce urmează un curs cu profesorul SOARE la valoarea A, putem realiza următorul update:

UPDATE STUDENTI
SET STUDENTI='A'
WHERE C# **IN**
(**SELECT** C# **FROM** PROFESORI
WHERE
PNUME='SOARE').

Legătura între SQL și algebra relațională este exprimată prin următoarea teoremă.

Teoremă: Fie $\tau'=(m',G,\rho)$ și $\tau''=(m'',H,\sigma)$ două tabele și presupunem că tabela $\tau=(m,K,\theta)$ este construită pornind de la mulțimea tabelelor $\{\tau',\tau''\}$ prin aplicarea uneia din operațiile algebrei relaționale sau prin redenumire. În fiecare caz, aceeași tabelă poate fi construită în SQL.

Demonstrație: Este necesar să considerăm șase cazuri ce corespund redenumirii fiecăreia din cele cinci operații ale algebrei relaționale.

1. Presupunem că τ este obținută din τ' prin redenumire, $K=i_1...i_n$ și $G=j_1...j_n$. În acest caz, creăm τ și apoi inserăm liniile lui τ' în τ:

CREATE TABLE $m_{(i1\ d1,...,indn)}$;
INSERT INTO $m_{(i1,...,in)}$
SELECT $j_1,...,j_n$ **FROM** m'.

2. Dacă τ este creată din τ' și τ'' prin uniune uniune, atunci tabelele τ și τ'' trebuie să fie compatibile, adică, $G=H=K=i_1...i_n$. După crearea tabelei τ, putem completa această tabelă cu tuplele

care aparţin atât lui τ', cât şi lui τ''utilizând următoarele instrucţiuni INSERT:

INSERT INTO $m(i_1,...,i_n)$
SELECT $i_1,...,i_n$ **FROM** m';
INSERT INTO $m(i_1,...,i_n)$
SELECT $i_1,...,i_n$ **FROM** m''.

3. Pentru cazul când τ este construită din τ''şi τ' prin diferenţă, trebuie să avem compatibilitate între tabelele τ' şi τ''. După crearea tabelei τ, utilizăm două instrucţiuni SQL: o inserţie urmată de o ştergere:

INSERT INTO $m(i_1,...,i_n)$
SELECT $i_1,...,i_n$ **FROM** m';
DELETE FROM m **WHERE**
EXISTS (SELECT * **FROM** m''
WHERE i_1=m''.i_1 **AND**
AND i_n=m''.i_n).

4. Să presupunem acum că τ este creată din τ' şi τ'' prin produs. Dacă $G=g_1...g_m$ şi $H=h_1...h_n$, atunci $K=m'.g_1...m'.g_m m''.h_1...m''.h_n$. În acest caz, după crearea tabelei τ, folosim următoarea instrucţiune INSERT:

INSERT INTO $m(m'.g_1...m'.g_m m''.h_1...m''.h_n)$
SELECT $m'.g_1...m'.g_m m''.h_1...m''.h_n$
FROM m',m''.

5. Când τ este obţinută din τ' prin proiecţie, avem $K \subseteq G$. Dacă $G=i_1...i_n$ şi $K=j_{p1}..j_{pr}$, atunci creăm tabela τ şi apoi folosim următorul INSERT:

INSERT INTO $m(i_{p1},...,i_{pr})$
SELECT $i_{p1}...i_{pr}$ **FROM**
m'.

6. În final, dacă τ este obţinută din τ' printr-o selecţie specificată prin condiţia C, atunci putem utiliza inserţia:

INSERT INTO $m(i_1,...,i_n)$
SELECT $i_1,i_2,...,i_n$ **FROM** m'
WHERE C.

SQL, ca numeroase alte limbaje de interogare relaţionale, oferă posibilitatea utilizării unei mulţimi de funcţii predefinite. Printre cele mai importante sunt sum, avg, max, min, count care calculează suma, media, maximul şi minimul unei mulţimi de valori sau numără tuplele unei mulţimi de relaţii, respectiv.

Aceste funcţii pot fi utilizate în expresii SELECT.

Exemplu: Determinăm numărul cursurilor pe care le urmează studentul cu numărul 2341; putem folosi următoarea expresie:

SELECT COUNT(C#) **FROM** STUDENŢI
WHERE C#='2341'.

Considerăm relaţia ρ pe schema R şi fie $K \subseteq R$. Definim echivalenţa θ_k pe ρ (adică, mulţimea n-tuplelor care formează relaţia ρ) ca $(u,v) \in \theta_k$ dacă u[K]=v[K]. Clasele de echivalenţă ale mulţimii ρ/θ_k sunt grupurile determinate de atributele lui K. Adică, grupul unui tuplu t constă din toate tuplele care au aceleaşi componente pe atributele lui K conţinute de t. SQL poate partiţiona o relaţie utilizând o parte din atributele acesteia şi apoi să calculeze o caracteristică pentru fiecare din aceste clase cu ajutorul opţiunii **GROUP BY** în expresia **SELECT**. Dacă opţiunea **WHERE** este utilizată, atunci opţiunea **GROUP BY** poate fi utilizată după opţiunea **WHERE**. În această situaţie, mulţimea cât va fi calculată după apelul selecţie specificată prin **WHERE**.

Exemplu: Presupunem că este necesar să determinăm numărul cursurilor absolvite de studenţii în baze de date. Grupăm tuplele tabelei în clasele de echivalenţă ale relaţiei θ_{SNUME}. Aceasta poate fi realizată utilizând opţiunea GROUP BY a unei expresii SELECT:

SELECT SNUME, COUNT(C#) **FROM** STUDENŢI
WHERE STUDENTI!='F'
GROUP BY SNUME.

Dacă tabela STUDENTIB are conţinutul:

STUDENTIB

C#	STUDENTN	STUDENT#	SERIE
110	POPESCU IOANA	2341	A
110	IONESCU MARIA	5641	D
110	SZABO MAGDA	9008	B
110	ABAGIU SORINA	2341	F
210	GEORGESCU DANA	4430	F
210	ARVAY TUNDE	2345	A
210	PREDA DECEBAL	7655	A
310	POPA STEFAN	0988	A
310	POPESCU ION	5448	A

atunci, din punct de vedere conceptual, tabela este împărţită în următoarele grupe după eliminarea acelor tupluri ce au componenta SERIE egală cu F.

STUDENTIB

C#	STUDENTN	STUDENT#	GRADE
110	POPESCU IOANA	2341	A
110	IONESCU MARIA	5641	D
110	SZABO MAGDA	9008	B
310	ARVAY TUNDE	0988	A
210	PREDA DECEBAL	2345	A
210	POPA STEFAN	7655	A
310	POPESCU ION	5448	A

Ca rezultat, obţinem următoarea tabelă.

STUDENTN

POPESCU IOANA	1
IONESCU MARIA	1
POPA STEFAN	2
PREDA DECEBAL	3

SQL cere ca interogările să returneze valori simple pentru fiecare grup.

Spre exemplu, expresia SQL

SELECT STUDENTN.C# **FROM** STUDENTIB

WHERE STUDENTIB!='F'

GROUP BY STUDENTN

nu este admisă deoarece răspunsul va returna o mulțime de numere de curs în loc de cardinalitatea acelei mulțimi.

După realizarea unei grupări prin utilizarea expresiei GROUP BY, putem selecta anumite grupuri utilizând opțiunea HAVING. Dacă este necesar să obținem numai studenții care au absolvit cel puțin două cursuri, putem utiliza următoarea expresie SELECT:

SELECT STUDENTN,COUNT(C#) **FROM** STUDENTIB
WHERE STUIDENTIB!='F'
GROUP BY STUDENTN
HAVING COUNT(C#) >=2.

Ca rezultat, avem următoarea tabelă:

STUDENTN

SZABO MAGDA	2
GEORGESCU DANA	3

Un atribut aparține unui anumit domeniu.Considerăm o colecție de mulțimi finite.

Fie $\{D_{i1},D_{i2},...,D_{in}\}$ n mulțimi finite.
$\rho \subseteq \prod\{D_{ij} | i \in \{i_1,i_2,...,i_n\}\}, \rho=\{(x_1,...,x_n),...,(z_1,...,z_n)\}$
ρ poate fi reprezentată printr-o tabelă.
$i_1,...,i_n$ sunt atributele tabelei τ.

Presupunem $i_1,i_2,...,i_n \in V^*$(un atribut este un cuvânt în V).

La un anumit moment, nu se cunoaște valoarea unui atribut.

Exemplu: Fie atributele asociate la trei domenii:

ORAS, JUDEȚ, POPULAȚIE.

Relația este: Capitale de județ cu populația respectivă.

Definiție: {DEVA, Hunedoara, 312 405}

Definiție: F(ORAȘ)=DEVA, F(JUDEȚ)= Hunedoara, F(POPULAȚIE)=312405

CAPITALE-JUDEȚ(ORAȘ, JUDEȚ, POPULAȚIE)
entitate — atribute asociate

La un moment dat, nu se cunoaște populația Hunedoarei sau nu știu populația din județ. Este necesar să introduc o valoare convențională în relație care să reprezinte un atribut necunoscut sau neaplicabil și aceasta este valoarea null.

Este necesară o aritmetică și o logică nouă care să cuprindă acest element.

and	t	f	null
t	t	f	null
f	f	f	f
null	null	f	null

or	t	f	null
t	t	t	t
f	t	f	null
null	t	null	null

x	¬x
t	f
f	t
null	null

Rezultatul aplicării operatorilor +,--,*,/,≤,= este null când unul din operanzi este null.

null=null este null.

Prin utilizarea elementului 'null', apare o noțiune de 'vag'-fuzzy.

Noțiunea de mulțime fuzzy a fost introdusă de L.A. Zadeh în anul 1965. Mulțimile fuzzy (și conceptele fuzzy în general) au apărut din necesitatea de a descrie noțiunile imprecise, vagi (în traducere, "fuzzy" înseamnă neclar, estompat, vag, imprecis). De exemplu, atribute cum ar fi "înalt", "scund","bogat", "sărac","mic" sau "mare" sunt imprecise. Ele sunt modelate utilizând conceptele fuzzy.

Apare necesitatea definirii unei mulțimi fuzzy prin descrierea unor "grade de apartenență". Există diferite posibilități de a alege mulțimea valorilor gradelor de apartenență. O posibilitate este de a alege mulțimea [0,1] sau, mai general, o latice cu prim și ultim element. În acest caz, vom considera că grad de apartenență 0 înseamnă neapartenența, iar gradul de apartenență 1 apartenența totală. De exemplu, este greu să vorbim de "mulțimea oamenilor scunzi" deoarece noțiunea "scund" face parte din clasa noțiunilor imprecise. Se observă însă că în funcție de înălțime, putem asocia fiecărui om un "grad de apartenență " la această mulțime. Un adult de 1m va fi considerat scund în mod sigur (deci gradul de apartenență va putea fi considerat 1) ,iar cineva de 2m nu este scund în mod cert (deci, în acest caz, gradul de apartenență va putea fi considerat 0). Celor cu înălțimi intermediare li se vor asocia corespunzător grade de apartenență la această mulțime.

Definiție: Se numește mulțime fuzzy în X o aplicație F:X--->[0,1].

Observație: Noțiunea de mulțime fuzzy poate fi extinsă dacă admitem ca funcția caracteristică să ia valori într-o latice sau într-o algebră Boole.

Operații cu mulțimi fuzzy

Reamintim că operatiile uzuale din teoria clasică a mulțimilor se extind la mulțimi fuzzy.

Considerăm M,N două mulțimi fuzzy, f funcția caracteristică a mulțimii M și g funcția caracteristică a mulțimii N.

Incluziunea

$M \subset N$ şi $f \leq g$ (unde $<$ este relaţia de ordine punctuală între funcţii $f \leq g$ ($\forall x \in X\ f(x) < g(x)$)).

Reuniunea

$M \cup N$ este mulţimea fuzzy cu funcţia caracteristică $h: X \to [0,1]$, $h(x) = \max(f(x), g(x))\ \forall x \in X$.

Intersecţia

$M \cap N$ este mulţimea fuzzy cu funcţia caracteristică $h: X \to [0,1]$ $h(x) = \min(f(x), g(x))\ \forall x \in X$.

Complementara unei mulţimi fuzzy

CM, complementara mulţimii M este mulţimea fuzzy a cărei funcţie caracteristică este

$h(x) = f(x)' = 1 - f(x)$.

Mulţimea atributelor asociate unei relaţii poate fi considerată ca o multime fuzzy.

Se poate verifica fără dificultăţi că rezultatele prezentate mai sus îşi păstrează valabilitatea considerând această "extensie fuzzy a unei baze de date."

Vom considera o variantă de definire a unui model de date relaţional fuzzy.

În cele ce urmează, vom presupune că U_i este o mulţime finită, $F(U_i)$ o clasă a tuturor submulţimilor fuzzy ale lui U_i, $\prod_{i=1}^{n} = U_1 * U_2 * \ldots * U_n$, $\prod_{i=1}^{n} F(U_i) = F(U_1) * F(U_2) * \ldots * F(U_n)$.

Definiţie: O relaţie $\sim R$ este o submulţime fuzzy a mulţimii $\prod_{i=1}^{n} U_i$, definită prin $\mu_{\sim R}$ adică fiind dată funcţia $\mu_{\sim R}: \prod_{i=1}^{n} \to [0,1]$, vom considera relaţia fuzzy corespunzătoare lui R: $\sim R = \{(u, \mu_{\sim R}(u)); u = (u_1, u_2, \ldots, u_n) \in \prod_{i=1}^{n} U_i\}$

Observăm că $R = \{u; \mu_{\sim R}(u) > 0,\ u \in \prod_{i=1}^{n} U_i\} = \text{supp} \sim R = \prod_{i=1}^{n} U_i$. $\sim R$ este numită relaţia fuzzy corespunzătoare lui R, notată ca $\sim R = (R, \mu_{\sim R})$ unde $\mu_{\sim R}$ este numit gradul de apartenenţă al lui u în R. Dacă pentru fiecare $u \in R$, $\mu_{\sim R}(u) = 1$, atunci $\sim R = R$.

Vom defini operaţiile modelului de date fuzzy relaţional, utilizând definiţiile operaţiilor asupra submulţimilor fuzzy.

Dacă $\sim R = (R, \mu_{\sim R})$ şi $\sim S = (S, \mu_{\sim S})$, avem:

(1) $\sim R \cup \sim S = (R \cup S, \mu_{\sim R} V \mu_{\sim S})$,
unde $(\mu_{\sim R} V \mu_{\sim S})(u) = \max(\mu_{\sim R}(u), \mu_{\sim S}(u)), u \in \prod_{i=1}^{n} U_i$;

(2) $\sim R \cap \sim S = (R \cap S, \mu_{\sim R} \Lambda \mu_{\sim S})$,
unde $(\mu_{\sim R} \Lambda \mu_{\sim S})(u) = \min(\mu_{\sim R}(u), \mu_{\sim S}(u)), u \in \prod_{i=1}^{n} U_i$;

(3) $R - S = (T, \mu_{\sim T})$ unde $T = (R - S) \cup \{u; \mu_{\sim R}(u) - \mu_{\sim S}(u) > 0,$
$u \in R \cap S\}$,
$\mu_{\sim T}(u) = \max(\mu_{\sim R}(u) - \mu_{\sim S}(u), 0)$;

(4) Pentru orice $0 < \lambda \leq 1$,

(5) $(\sim R)_\lambda = \{u;\ \mu_{\sim R}(u) \geq \lambda, u \in \prod_{i=1}^{k} U_i\ \} = R_\lambda \subset \prod_{i=1}^{n} U_i$,
$R = \cup_{0 \leq \lambda \leq 1} R_\lambda$.

(6) Proiecţia, pentru orice $A = \prod_{i=1}^{k} U_{ij}$.
$\sim R[A] = (R[A], \mu_{\sim R[A]}) = \{u[A], \mu_{\sim R}(u)); u \in R\}$, unde
$u[A] = (u_{i1}, u_{i2}, \ldots, u_{ik}) \in A$.

(7) Uniune (join). Fie $\sim R$ o relaţie fuzzy în $\prod_{j=1}^{k1} U_{ij}$, $\sim S$ o relaţie fuzzy în $\prod_{m=1}^{k2} U_{jm}$, atunci putem defini:

$$\sim R \underset{F}{\bowtie} \sim S = (R \underset{F}{\bowtie} S,\ \mu_{\sim R} \underset{F}{\bowtie} \sim S),$$ un

$$R \underset{F}{\bowtie} S = \{rs; r \in R, s \in S, F(r,s) = 1\}$$

$$\mu_{\sim R \bowtie \sim S}(rs) = \min(\mu_{\sim R}(r), \mu_{\sim S}(s))$$

$r = (u_{i1}, u_{i2}, \ldots, u_{ik1})$ şi $s = (u_{j1}, u_{j2}, \ldots, u_{jk2})$
şi F este o funcţie booleană reprezentând condiţia din uniune.

(8) selecţia $R[F] = \{(u, \mu_{\sim R(u)};\ u \in R;\ F(u, \mu \sim R(u) = 1\}$, unde F este o funcţie booleană ce reprezintă condiţia selecţiei şi F poate conţine $\mu \sim R$.

Bibliografie:

1. **FEJER, P. A., SIMOVICI, D. A.:** Mathematical Foundations of Computer Science. Volume I: Sets, Relations and Induction, Springer Verlag, Berlin, 1990.
2. **OZKARAHAN, E.:** Database Management Concepts, Design and Practice Prentice-Hall International,Inc.
3. **GALOCSI :** Conception de bases de données. În: Informatique, Ed. E. Dunod, 1991.
4. **ULLMAN, J.D.:** Principles of Database and Knowledge-base Systems. În: Computer Science Press, 1988.
5. **LU, JIE:** A model for fuzzy relational database. În: Collected Works on Fuzzy Sets and Systems, Human Science and Technology Publisher, 1991.