METODOLOGII DE UTILIZARE A APARATULUI MATEMATIC ÎN PROCESUL DE PROIECTARE A BAZELOR DE DATE RELAȚIONALE.

Teodora VASCAN, UST, teodora_vascan@mail.ru

Rezumat. *Articolul prezintă o sinteză a fundamentului matematic necesar la proiectarea şi interogarea bazelor de date relaţionale. Sunt expuse unele metodologii de utilizare a algebrei relaţionale şi a calculului relaţional în procesul de proiectare a bazelor de date relaţionale.*

Abstract. *This article represents a synthesis of mathematical foundation necessary to design and querying relational databases. They exposed some methodology to use relational algebra and relational calculus in the process of designing relational databases.*

În teoria bazelor de date relaţionale se acentuiază prezenţa următoarelor proceduri:

1. de proiectare a bazelor de date (luând în considerare integritatea şi protecţia datelor);
2. de interogare şi actualizare a datelor;

3. de sincronizare a proceselor de acces la date în modul mulţi-utilizator.

Instrumentul teoretic pentru primele două proceduri îl constituie algebra relaţională şi calculul relaţional. **E. F. Codd**[3], a propus folosirea aparatului teoriei mulţimilor şi teoriei relaţiilor pentru prelucrarea bazelor de date [6]. El folosea pentru descrierea datelor din careva domeniu o formă specială de tabele bidimensionale, cunoscute în matematică sub numele de relaţie, şi o totalitate de reguli de lucru cu aceste tabele (algebra). Teoria normalizării propusă de el a fost un punct de plecare în crearea limbajelor de manipulare a datelor de tip relaţional. Printre acestea cele mai răspândite sunt *SQL* (limbaj structural al interogărilor) şi *QBE* (interogări după model), cu ajutorul cărora utilizatorul indică ce date sunt necesare de primit din baza de date, fără a concretiza procedura de căutare în baza de date. Algebra relaţională determină setul de operaţii (algebrice), care trebuie realizate de sistemul de gestiune a bazelor de date pentru a primi rezultatul interogării (cererii). Interacţionând cu sistemul în limbajul algebrei relaţionale, utilizatorul trebuie să ştie a manipula cu operaţiile algebrei relaţionale la construirea interogărilor [1]. Cu toate acestea de la utilizatorul-neprofesionist se cer careva cunoştinţe determinate din domeniul matematicii, şi lui i se alocă întrebările, legate de construirea unor aşa interogări în limbajul algebric, care ar necesita minim timp pentru realizare.

Calculul relaţional sau calculul relaţiilor este bazat pe compartimentul logicii matematice, care se numeşte *calculul predicatelor*. Calculul relaţional stă la baza abordării declarative a formulării interogărilor bazelor de date. Prin abordarea declarativă interogării bazei de date îi corespunde formula calculului relaţional. Răspuns la interogare va servi mulţimea obiectelor din domeniul de interpretare (în cazul nostru acest domeniu este baza de date), pe care este adevărată formula, ce corespunde interogării.

Cea mai generală formă de interogare în limbajul *SQL* prezintă prin sine o expresie algebrică ce include operaţii pe mulţimi, compusă din interogări elementare. În sistemele dezvoltate *SQL* sunt permise toate operaţiile asupra mulţimilor [3].

La baza unei baze de date relaţionale stau noţiunile de „relaţie” şi „legătură”.

Definiţie: Se numeşte **relaţie *r*** - submulţimea produsului cartezian. Câmpurile unei relaţii (tabele) pot fi aranjate în orice ordine. Pentru a stabili o oarecare ordine pentru o careva realizare concretă, vine noţiunea de „**schemă**” R – o mulţime ordonată a numelor atributelor $R(A_1, ..., A_n)$. Se spune schema R a relaţiei r sau $r(R)$. Oricărui nume A_i, i se pune în corespundere mulţimea D_i (domeniu sau $dom(Ai)$). Schema este o mulţime finită de tupluri $t \in r$, unde $t(A_i) = D_i$. O paradigmă a bazelor de date este cheia r a relaţiei R – submulţimea $\underline{K} = \{B_1,...,B_v\} \in R, m \le n$, cu restricţiile:

[3] **Edgar Frank Codd** (23 august 1923, Insula Portland Anglia – 18 aprilie 2003, Williams Island, Florida, SUA) a fost un informatician american de origine engleză care, lucrând pentru IBM, a inventat modelul relaţional pentru gestiunea bazelor de date, model care constituie baza teoretică a bazelor de date relaţionale. A adus şi alte contribuţii domeniului informaticii, dar modelul relaţional, o teorie generală şi de mare influenţă a gestiunii datelor, rămâne principala sa realizare.

1. pentru orice două tupluri t_1 şi t_2 există un aşa $B \in \underline{K}$, încât $t_1(B) \neq t_2(B)$;
2. $t_1(\underline{K}) \neq \underline{K}$;
3. nici o submulţime $\underline{K''} \subset \underline{K}$ nu are proprietăţile cheii.

Cheile enumerate în schema relaţională se numesc – **implicite**; în caz contrar – **ne-implicite**. Una din cheile subliniate se numeşte **primară**. Dacă cheia $\underline{K''} \subset \underline{K} \subset R$, atunci $\underline{K}$ - este **super-cheie** (cheie compusă). Există şi cazul când câteva mulţimi minime a atributelor determină funcţional toate atributele relaţiei. Aşa mulţimi se numesc chei **posibile**, deoarece oricare din ele poate fi aleasă în calitate de cheie compusă [1].

Exemplu: Fie relaţia *R(Oraş, Adresa, Cod_poştal).*

În mod evident atributele *Oraş, Adresa* $\rightarrow$ *Cod_poştal. Pe când Cod_poştal* $\rightarrow$ *Oraş* (deşi nu este o adresă). Ambele mulţimi pot fi chei posibile.

Definiţie: **Univers *U*** se numeşte mulţimea valorilor tuturor atributelor. Luînd în calcul cheile, schema *R* pe *U* este totalitatea relaţiilor $\{R_1, ..., R_t\}$ unde $R = \left\{S_i, \underline{K_i}; 1 \le i \le n\right\}, \quad \sum_{i=1}^{n} S_i = U$.

Definiţie: Atunci se numeşte **bază de date** *d* cu schema datelor *R* totalitatea relaţiilor $\{r_1, ..., r_n\}$, unde pentru orice schemă $R = \{S, \underline{K}\}$ există relaţii în *d*, ce sunt relaţii cu schema *S*, ce satisfac condiţiile oricărei chei.

În bazele de date relaţionale se operează cu **tabelele**. Cel mai simplu element este **tuplul** (înregistrarea).

La proiectarea bazei de date se formează o structură de tabele şi o schemă a relaţiilor, în timp ce utilizatorul poate avea nevoie de o structură complet diferită cu o schemă arbitrară (în limita universului existent) [2].

Pentru tabelele bazei de date sunt specifice următoarele acţiuni:

1) crearea;
2) actualizarea şi interogarea;
3) sincronizarea proceselor de acces.

În procesul de creare şi actualizare trebuie asigurată integritatea datelor, în timp ce în procesul interogării este necesar transformarea tabelelor presupunând asigurarea integrităţii.

Să începem cu descrierea operaţiilor de transformare a tabelelor.

Pentru a opera cu tabelele este necesar să fim în măsură de a le descrie formal.

Descrierea teoretică poate fi de două feluri:

1) predicate de rangul I;
2) folosirea regulilor algebrei relaţionale (*AR*) şi a calculului relaţional (*CR).*

Prima formă este specifică şi caracteristică pentru sistemele expert (intelectuale), de aceea vom descrie forma a doua.

Definiţie: **Algebră** – este o mulţime *A* cu operaţiile definite pe ea de forma: $f : A^n \to A$ unde *n* – dimensiune.

Definiţie: **Calcul** – este totalitatea regulilor de operare cu careva simboluri.

Calculul relaţional în esenţa sa este mai simplu, dar folosirea lui este limitată de următoarele circumstanţe:

1) multe probleme algoritmice încă nu sunt rezolvate şi nu pot fi rezolvate în limitele calculului relaţional;
2) complexitatea combinatorică a schemelor relaţionale nu se deschide prin calculul relaţional;
3) practic nu este posibil – în limitele calculului relaţional – de demonstrat caracterul complet al operaţiilor de transformare efectuate.

Toate aceste circumstanţe sunt posibile de realizat cu ajutorul algebrei relaţionale, implementarea software a căreia sunt limbajele **procedurale** de programare.

Fie *C1(L)* – mulţimea tuturor formulelor închise ale sistemului *L*. Dacă formula $\varphi \in C_1(L)$, se spune că modelul M satisface $\varphi(\varphi = M)$ dacă φ este adevărat pe *M*.

Fie $y \in C_1(L)$. Formula Ψ se numeşte consecinţă a lui Y (ce derivă din Y, dacă din $\psi = M$, rezultă *Y=M* pentru orice model *M*).

Definiţie: Orice relaţie construită corect cu ajutorul operatorilor acceptaţi de sistem se numeşte **expresie algebraică**.

Fie *U* –univers (mulţimea atributelor), D – mulţimea domeniilor, dom – funcţia completă din $U(dom : U \to D)$, $R = \{R_i, i = 1, p\}$ - mulţimea schemelor de relaţii, $d = \{r_i, i = 1, p\}$- mulţimea tuturor relaţiilor $r_i(R_i), \theta = \{\neq, =, \leq, \geq, >, <\}$ - mulţimea relaţiilor binare (condiţii pe domenele din *D*), *O* – mulţimea operatorilor (operaţii), ce folosesc atribute din *U* şi relaţii din θ.

Definiţie: Se numeşte **algebră relaţională** pe *U, D, dom, R, d, Q* tuplul $B = \{U, D, dom, R, d, \theta, o\}$.

În algebra relaţională se evidenţiază următoarele operaţii: proiecţia care se notează P, selecţia (*S*), joncţiunea (*J*), reuniunea (*U*), diferenţa (*DF*), împărţirea, intersecţia, produsul cartezian (*CP*). Fie relaţiile *R(A,B,C)* şi *P(D,E,F)*. Reuniunea, intersecţia şi diferenţa are loc asupra relaţiilor cu aceiaşi aritate [4], [5].

1. **Definiţie**: **Reuniunea** a două relaţii *R* şi *P* este mulţimea tuplurilor aparţinând fie lui *R* fie lui *P*.

   Operaţia de reuniune *U (R,P)* fără repetarea liniilor:

R(A,B,C)	$\cup$	***P(D,E,F)***	=	***Q(A,B,C)***
a b c		m n o		a b c
d e f		g h i		d e f
g h i				g h i

j k l	j k l
	m n o

În *SQL* reuniunea se poate exprima folosind operatorul *UNION*:

```
SELECT A,B,C
FROM R
UNION
SELECT D,E,F
FROM P;
```

2. **Definiţie**: **Diferenţa** a două relaţii *R* şi *P* este mulţimea tuplurilor care aparţin *R*, dar nu aparţin lui *P*.

Diferenţa *(DF(R,P))* – din *R* se elimină rândurile care sunt în *P*.

R(A,B,C)	***DF***	***P(D,E,F)***	**=**	***Q(A,B,C)***
a b c		m n o		a b c
d e f		g h i		d e f
g h i				j k l
j k l				

În *SQL* diferenţa se poate exprima folosind operatorul *MINUS* [3]:

```
SELECT A, B,C
FROM R
MINUS
SELECT D, E, F
FROM P;
```

În plus diferenţa poate fi simulată şi prin operatorul *NOT EXISTS*. De exemplu, comanda *SQL* de mai sus este echivalentă cu următoarea:

```
SELECT A, B,C
FROM R
WHERE NOT EXISTS
    (SELECT *
    FROM P
   WHERE R.A=P.D AND R.B=P.E AND R.C=P.F);
```

Observaţie: Pentru simplitate în comanda *SQL* de mai sus s-a presupus că nici un atribut din relaţiile *R* sau *P* nu poate avea valoarea *NULL*.

3. **Definiţie**: **Intersecţia** a două relaţii *R* şi *P* este mulţimea tuplurilor care aparţin atât lui *R* cât şi lui *P*.

Intersecţia $R \cap P$ - elementele comune ale ambelor mulţimi:

R(A,B,C)	∩	***P(D,E,F)***	**=**	***Q(A,B,C)***
a b c		m n o		g h i

d e f g h i

g h i

j k l

În *SQL* intersecţia se poate exprima folosind operatorul *INTERSECT*:

```
SELECT A, B,C
  FROM R
    INTERSECT
      SELECT D, E, F
        FROM P;
```

În plus intersecţia poate fi simulată şi prin operatorul *EXISTS*. De exemplu, comanda *SQL* de mai sus este echivalentă cu următoarea:

```
SELECT A, B,C
FROM R
WHERE EXISTS
      (SELECT *
      FROM P
      WHERE R.A=P.D AND R.B=P.E AND R.C=P.F);
```

În cazul când operatorii *UNION*, *DIFFERENCE* sau *INTERSECT* se aplică unor relaţii care sunt obţinute prin selecţie din aceiaşi relaţie, atunci aceştia pot fi simulaţi prin aplicarea operatorilor logici corespunzători (*OR*, *AND NOT*, *AND*) asupra condiţiilor de selecţie. De exemplu, următoarele comenzi sunt echivalente:

```
SELECT A, B
FROM R
   WHERE A=`x1`
     MINUS
SELECT A, B
  FROM R
 WHERE B=`y1`;
SELECT A, B
  FROM R
 WHERE A=`x1` AND NOT B=`y1`;
```

4. **Definiţie**: **Produsul cartezian** al două relaţii *R* şi *P* este mulţimea tuturor tuplurilor care se obţin prin concatenarea unui tuplu din *R* cu un tuplu din *P*. Prin urmare, dacă aritatea relaţiei *R* este *m*, iar aritatea relaţiei *P* este *n*, atunci produsul cartezian dintre *R* şi *P* va avea aritatea *m+n*, notaţiile folosite de obicei pentru acest operator sunt: *R×P*, *PRODUCT (R,P)*, *TIMES(R,P)*, *CP(R,P)*.

   Produsul cartezian *CP(R,P):*

R(A, B, C)	***P(D, E, F)***	***CP(R, P)***
a b c	m n o	a b c m n o
d e f	g h i	d e f m n o
g h i		g h i m n o
j k l		j k l m n o
		a b c g h i
		d e f g h i
		g h i g h i
		j k l g h i

Produsul cartezian va fi exprimat în *SQL* printr-o comandă *SELECT* pe mai multe tabele fără clauza *WHERE*:

*SELECT **

FROM R, P

5. **Operatorul PROJECT (proiecţia) –** acesta este un operator care are ca parametri un atribut sau mai multe atribute ale unor relaţii şi care elimină din relaţie toate celelalte atribute, producând o submulţime ”pe verticală” a acesteia. Datorită faptului că suprimarea unor atribute poate avea ca efect apariţia unor tupluri duplicate, acestea vor fi eliminate din relaţia rezultantă deoarece, prin definiţie, o relaţie nu poate conţine tupluri cu valori identice. Notaţiile folosite de obicei pentru acest operator sunt: $\pi_x(R)$ şi *PROJECT(R;X)* unde *R* reprezintă relaţia, iar *X* este atributul sau mulţimea de atribute care constituie parametrii proiecţiei.

R(A,B,C)	$\pi_{B,C}(R)$	=	***Q(A,B,C)***
a b c			b c
d b c			d k
a d k			

În *SQL* proiecţia fără dubluri se obţine folosind comanda *SELECT* cu specificaţia *DISTINCT*, altfel se obţine proiecţia cu dubluri:

SELECT DISTINCT B, C

FROM R;

6. **Operatorul SELECT –**este un operator unar care este utilizat pentru extragerea tuturor tuplurilor dintr-o relaţie care satisfac o condiţie specificată, producând astfel o submulţime „pe orizontală" a relaţiei. Condiţia este o expresie logică ce poate conţine nume de atribute, constante, operatori logici (*AND, NOT, OR*), operatori de comparaţie (<, =, >, <=, >=, !=). Notaţiile folosite de obicei pentru acest operator sunt $\sigma_F(R)$ sau *SELECT (R,F)*, unde *R* – reprezintă relaţia, iar $F(A_i, \theta,\ constanta)$ – relaţia rezultantă cu aritatea *n*; A_i –atributul relaţiei *R*; θ sau m – condiţie logică $(<, >, =, <>, \cap, \cup)$:

R(A,B,C)	$\sigma_{A=a}(R)$	=	***Q(A,B,C)***
a b c			a b c
d b c			a d k
a d k			

În *SQL* selecţia se obţine folosind comanda *SELECT* cu clauza *WHERE*:

```
SELECT *
  FROM R
 WHERE  A=`x1` OR  B=`y1`;
```

Combinarea selecţiei cu proiecţia fără dubluri se face în modul următor:

```
SELECT DISTINCT C, A
  FROM R
WHERE A=`x1` OR B=`y1`;
```

Exemplu:

A	B	C
x1	y1	z1
x1	y1	z2
x1	y2	z2
x2	y2	z1

→

A	B	C
x1	y1	z1
x1	y1	z2
x1	y2	z2

→

C	A
z1	x1
z2	x1

Combinarea selecţiei cu proiecţia cu dubluri se face în modul următor:

```
SELECT  C, A
  FROM R
 WHERE  A=`x1` OR  B=`y1`;
```

A	B	C
x1	y1	z1
x1	y1	z2

→

A	B	C
x1	y1	z1
x1	y1	z2

→

C	A
z1	x1
z2	x1
z2	x1

x1	y2	z2		x1	y2	z2
x2	y2	z1				

7. Operatorul de **compunere** (joncţiune) permite regăsirea informaţiei din mai multe relaţii corelate. Compunerea este o operaţie binară care are ca rezultat o nouă relaţie în care fiecare tuplu este o combinaţie a unui tuplu din prima relaţie cu un tuplu din a doua relaţie.

Compunerea $J_{AmB} = Q = \sigma_{AmB}$:

R(A, B, C)	***P(D, E)***	***Q(A, B, C, D, E)*** *pentru B≠D*
a b c	d c	a b c d c
d b c	b a	d b c d c
a d k	b b	a d k b c
		a d k b b

8. Dacă câmpurile ce se compară, numele cărora ar fi bine să fie acelaşi, în relaţia rezultantă se numără o singură dată, atunci se vorbeşte despre **compunerea naturală** *NJ*.

Definiţie: **Compunerea naturală** este o operaţie binară comutativă care combină tupluri din două relaţii *R, P*, cu condiţia că atributele comune să aibă valori identice. În cazul compunerii naturale atributele specificate trebuie să aibă acelaşi nume.

De obicei, compunerea naturală se notează prin $R \lhd\rhd P$ sau *NJ(RP)*.

$NJ(R,P) = \pi_{1,2,...,m}(S_1,...,S_n)(R \times P)$, unde $S_i = (A_i^R = A_i^P; i = 1,n)$, A_i - lista atributelor ce coincid în relaţia iniţială; *1, ..., m* - lista ordonată a tuturor componentelor produsului cartezian $R \times P$, cu excepţia $A_1^P,...,A_n^P$.

R(A, B, C)	***P(D, A, B)***	***Q(A, B, C, D)***
a b c	d d d	a b c a
d b c	a a b	d b c d
a d k	c c d	d b a d
d b a		

Următorul exemplu ilustrează realizarea *compunerii naturale* în *SQL*:

```
SELECT DISTINCT R.A, R.B, R.C
  FROM R, P
 WHERE R.B=P.B
       AND R.C=P.C;
```

9. **Diviziunea** (împărţirea) este o operaţie binară care se aplică asupra a două relaţii *R* şi *P*, astfel încât mulţimea atributelor lui *R* include mulţimea atributelor lui *P*. Dacă *R* este o relaţie cu aritatea *m*, iar *P* este o relaţie cu aritatea *n*, unde $m > n$, atunci diviziunea lui *R* la *P* este mulţimea tuplurilor de dimensiune *m-n* la care, adăugând orice tuplu din *P*, se obţine un tuplu din S. Notaţiile utilizate cel mai des sunt: $R \div P$, *DIVISION(R,P), DIVIDER (R,P)*.

Fie avem:

(X,Y)=(A,B)	***Y=(B)***	***X=(A)***
1 a	y_1=e	x_1=1
1 b		3
1 c		4
1 d		5
1 e		
1 f	y_2=b	x_2=1
2 a	d	4
2 b		
3 c	y_3=a	x_3=1
3 e	b	
4 b	c	
4 d	d	
4 e	e	
5 e	f	

Diviziunea este o operaţie derivată care se exprimă cu ajutorul diferenţei, produsului cartezian şi proiecţiei: $R \div P = R_1 - R_2$ unde $R_1 = \pi_X(R), R_2 = \pi_X((R_1 \times P) - R)$, iar *X* este mulţimea atributelor lui *R* care nu există în *P*.

Pentru a ilustra exprimarea operatorului *DIVISION* vom considera relatiile **curs_student** şi **curs_fundamental** de mai jos:

Curs_student

cod_student	**curs**
S1	matematica
S1	fizica
S1	mecanica

Curs_funfamental

curs
matematica
fizica

Curs_student÷curs_fundamental

cod_student
S1
S4

S2	matematica
S2	informatica
S3	fizica
S4	matematica
S4	fizica

Atunci relaţia **curs_student ÷ curs_fundamental** poate fi definită prin următoarea întrebare: *care sunt studenţii care urmează toate cursurile fundamentale*? Alternativ, această relaţie poate fi definită prin întrebarea: *care sunt studenţii pentru care nu există curs fundamental care să nu fie urmat de aceştia*? Utilizând a doua formulare, rezultă că operatorul *DIVISION* poate fi simulat în *SQL* prin 2 operatori *NOT EXISTS*:

```
SELECT DISTINCT cod_student
FROM curs_student cs1
   WHERE NOT EXISTS
        (SELECT *
        FROM curs_fundamental cf
         WHERE NOT EXISTS
              (SELECT *
              FROM curs_student cs2
               WHERE cf.curs = cs2.curs
              AND cs1.cod_student = cs2.cod_student));
```

Cel mai des sunt folosite operaţiile de selecţie *(S)*, proiecţie *(P)* şi joncţiunea *(J)*, numite *SPJ*-operaţii.

Bibliografie:

1. **V. Cotelea**, *Algebra relaţională şi limbajul SQL*, Chişinău: Vizual Design, 2013, 284 p.
2. **R. Elmasri, S. B. Navathe**, *Fundamentals of Database Systems*, Edition 6, Addison Wesley Pub Co Inc, 2010, ISBN 0136086209, 9780136086208, Page 145 – 186.
3. **M. Fotache** - *Proiectarea bazelor de date. Normalizare şi postnormalizare. Implementări SQL şi Oracle*, Ed. Polirom, 2005.

Resurse Internet:

4. http://www.nyu.edu/classes/jcf/CSCI-GA.2433-001_sp15/slides/session5/RelationalAlgebra-RelationalCalculus-SQL.pdf
5. http://www.ccs.neu.edu/home/kathleen/classes/cs3200/4-RAAndRC.pdf
6. https://en.wikipedia.org/wiki/Edgar_F._Codd