Articole

EVALUAREA LOGICĂ A CERERILOR ÎN SISTEMELE RELAȚIONALE

mat. Mircea Râureanu

Institutul de Cercetări în Informatică

REZUMAT

Lucrarea prezintă arhitectura și funcțiile unui sistem deductiv de evaluare a cererilor relaționale. În prima secțiune, sînt propuse soluții de surmontare a incompletitudinii algebrei relaționale în raport cu cererile recursive. Soluția teoretică a acestei probleme constă în introducerea conceptului de **Bază de Date Logică (Deductivă)**. În continuare, sînt definite noțiunile de finitudine a răspunsului la o cerere, calculabilitatea efectivă a unei cereri, etc. De asemenea, sînt specificate caracteristicile esențiale ale strategiilor de evaluare în contextul Bazelor de Date Deductive. În partea a doua, este prezentat sistemul **PYTHIA**, care este o încercare de cuplare slabă a unui mecanism referențial cu DMSB ORACLE. Limbajul obiect al sistemului este DATALOG. Sistemul este compus din trei subsisteme pentru construirea bazei de cunoștințe, achiziția de metode de evaluare și evaluarea cererilor. Baza de cunoștințe este structurată în clustere reprezentate printr-o structură specială de graf numită **graf de prelucrare**. Pentru evaluare se aplică o strategie ascendentă: **evaluare semi-naivă.**

Cuvinte cheie: clauze Horn, algebra relațională, baze de date deductive,cereri recursive, programare logică, evaluare seminaivă.

În ultima perioadă, s-a manifestat un puternic interes pentru combinarea tehnicilor și funcționalităților Programării Logice și Bazelor de Date [23], [26]. În acest proces, Programarea Logică are un rol decisiv, prefigurînd viitorul sistemelor de baze de date.

Integrarea celor două domenii este facilitată de rădacina lor comună: logica matematică.

Motivațiile principale, care au generat necesitatea apropierii celor două domenii, rezidă în [35]:

- performanța: cerința ca sistemele experte, care vor fi atît de utilizate în viitor, să poată prelucra volume foarte mari de fapte și reguli;
- controlul și partajarea cunoștiințelor: pentru bazele mari de cunoștințe se pun acut probleme de protecție și partajare;
- de integritate și îmbunătățirea securității cunoștințelor;
- noi funcționalități: DBMS-urile actuale nu pot prelucra obiecte complexe, închideri și metadate, necesare în aplicații complexe, cum ar fi CAD. O rezolvare naturală a acestor probleme este dată de tehnicile Programării Logice.

Există două sensuri de integrare a Programării Logice și Bazelor de Date [26]:

- extinderea Programării Logice prin posibilități de stocare și căutare, specifice bazelor de date (fig. 1);
- extinderea Bazelor de Date cu posibilități deductive(fig.2).

Dintre abordările discutate în literatură, opțiunea noastră se îndreaptă către abordarea eterogenă, în care sistemul rezultat este compus din două subsisteme slab cuplate, unul de Programare Logică, iar celălaltun DBMS.

1.PROBLEMA

În această secțiune vom aborda unele dificultăți care apar în utilizarea curentă a unor RDBMS.

Exemplu:

Să presupunem că într-o bază de date relațională există relația PARENT

P	S
a	b
b	c
c	d
.	.
.	.
.	.

Dacă întrebări de tipul:

"Care sînt copiii lui a?"

"Care este tatăl lui b?"

sînt ușor de exprimat în algebra rațională, nu același lucru se poate spune despre întrebări precum:

"Care sînt descendenții lui a?"

"Care sînt ancestorii lui e?"

"Există vreo relație de rudenie între a și f?"

De fapt, cererile asociate acestor întrebări nu pot fi exprimate în algebra relațională. Singura soluție ar fi recurgerea la facilitățile oferite de un limbaj de nivel înalt, din care vor fi apelate funcții de evaluare standard. De exemplu, pentru rezolvarea primelor două întrebări "dificile" se poate adopta următoarea strategie:

- se calculează o relație nouă, numită Ancestor (A,D), care leagă o persoană de toți descendenții săi. Procedura de calcul ar putea arăta astfel:

```
assign Parent to Ancestor;
while is-changed (Ancestor) do begin
   assign ancestor to Temp
   assign Temp x Parent to Temp
   assign Temp u Ancestor to Ancestor;
end;
```

S-a plecat de la observația că:

Ancestor:= Parent u Parent x Parent u Parent X Parent X...

și că acest proces se oprește;

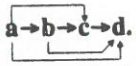
- se exprimă cererile respective, (de ex. în SQL), asupra relației Ancestor:

```
SELECT D FROM Ancestor
WHERE A = 'a';
SELECT A FROM Ancestor
WHERE D='f';
```

Pentru a înțelege mai bine calculul relației Ancestor vom asocia relației Parent un graf orientat. Astfel, unui tuplu Parent (x,y) îi va corespunde un arc X→Y.
Atunci, graful asociat relației Parent este

a→b→c→d,

iar mulțimea descendenților va fi dată de mulțimea tuturor drumurilor din graf, și anume



Relația corespunzătoare acestui graf este Ancestor și este închiderea tranzitivă a lui Parent.
Acest exemplu arată că, în algebra rațională standard, nu pot fi exprimate închideri tranzitive de relații. De aceea, pentru a suplini această deficiență trebuie construite proceduri în limbajele de programare de nivel înalt.

2. CAUZA DIFICULTĂȚILOR

Cauza profundă a insuficienței limbajelor relaționale în raport cu închiderea tranzitivă constă în faptul că, algebra relațională nu permite exprimarea recursivității. În fig.3, este ilustrată situarea calculului relațional (care este echivalent cu algebra relațională cu funcții agregat [12], față de logica în formă clauzală. Rezolvarea problemei exprimării recursivității în limbajele logice de interogare este fundamentală pentru realizarea integrării sistemelor de Programare Logică și de Baze de Date.

3.UNELE SOLUȚII ALE PROBLEMEI

În literatura de specialitate sînt propuse mai multe soluții, printre care mai importante par a fi următoarele :

- extinderea algebrei relaționale cu operatori de calcul al recursivității;
- utilizarea limbajelor logice existente, ca limbaje de interogare pentru RDBMS;
- crearea de limbaje logice dedicate, fundamentate pe conceptul de Baze de Date Logice.

În continuare, vom analiza succint unele soluții-candidat din fiecare categorie.

3.1. Operatorul ALFA

R. Agrawal, urmînd o idee a lui Zloof [36], propune introducerea în algebra relațională extinsă cu operatori de agregare, a unui operator de închidere tranzitivă, numit alfa [1], [2].
Dispunînd de acest operator, denumit CLOSURE în SQL, cererile din exemplul anterior pot fi exprimate astfel:

```
SELECT D FROM
CLOSURE <Parent)
WHERE A ="a".
```

și

```
SELECT A FROM
CLOSURE <Parent)
WHERE D ="F"
```

Relația CLOSURE (Parent (P,S)) are o formă specifică, și anume pentru Parent

P	S
a	b
b	c
c	d

CLOSURE

P	S	Delta
a	b	<a,b>
b	c	<b,c>
c	d	<c,d>
a	c	<a,b>, <b,c>
b	d	<b,c>, <c,d>
a	d	<a,b>, <b,c>, <c,d>

Atributul special Delta indică tuplele intermediare de calcul, și nu este disponibil decît pentru operația de proiectare.
În plus, este introdus un operator de formare de agregate, care asociaza unei relații oarecare R și unei funcții agregat f, o relație nouă, are același număr de tuple, ca R, și un atribut adițional, care conține valorile lui f pentru fiecare tuplu.
În literatură se găsesc și alte propuneri similare, din care amintim:

- închiderea relațională [22];
- închiderea tranzitivă generalizată [13];
- recursivitatea de traversare [29].

În concluzie, cu toate că extensiile propuse adaugă facilități deductive RDBMS-urilor actuale, problema recursitivității este, de fapt, ocolită. Mai grav, în aceste condiții integrarea dintre Programarea Logică și DBMS rămîne un obiectiv îndepărtat.

3.2. PROLOG ca limbaj logic de interogare

O serie de articole [35], [10], [30], propun limbajul PROLOG ca limbaj logic de interogare a unei RDBMS. Într-adevăr, limbajul PROLOG poate fi considerat un limbaj relațional orientat pe domeniu a cărui putere depășește pe aceea a calculului relațional. Aceasta rezultă din faptul că PROLOG, ca limbaj logic pur, este echivalent cu subsetul clauzelor HORN generale al Logicii de Ordinul Întîi. De aceea, prin intermediul recursivitîții, este posibil calculul unor cereri care nu pot fi exprimate în calculul relațional.
În PROLOG, calculul relației ANCESTOR se exprimă astfel:
Ancestor <x,y): -Parent <x,y>.
Ancestor <x>: -Parent <x, z), Ancestor <z,y>.
O cerere tipică poate fi : Ancestor ('a', x) care va furniza lista tuturor descendenților lui 'a'.
Evidenta similaritate între o cerere PROLOG și o cerere exprimată în calculul relațional orientat pe

domeniu, ascunde profundele diferențe dintre modelele lor de execuție. Spre deosebire de cele mai multe RDBMS, scopurile PROLOG sînt executate în ordinea strictă a specificării lor (back ward chaining cu backtracking). Această ordine este numită navigație fizică [35]. Prin contrast RDBMS-urile sînt nenavigaționale, ordinea de execuție a cererilor fiind în sarcina componentei de optimitizare a sistemului.
Altă dificultate importantă o constituie faptul că PROLOG operează pe un singur tuplu la un moment dat, în timp ce RDBMS-urile lucrează pe relații. Aceasta duce la intensificarea fluxului de date între cele două sisteme și, deci, la ineficiență în calculul unei cereri. O deficiență fundamentală a limbajului PROLOG constă în incompletitudinea metodei sale. Pentru ilustrare, să luăm exemplul anterior în care a fost schimbată ordinea clauzelor, iar în corpul primei clauze au fost inversate predicatele.:
Ancestor <x,y):- Ancestor (z,y>, Parent <x,z).
Ancestor <x,y):- Parent (x,y >.
Evaluarea acestui sistem nu se termină, întrucît pentru primul predicat din prima clauză, se va genera un arbore de rezoluție infinit.

3.3. Bazele de date logice

Soluția cea mai promițătoare constă în construirea unui fundament logic pentru bazele de date.
Reiter [28], a arătat că Bazele de date Relaționale pot fi văzute ca teorii ale Logicii de Ordinul Întîi (punctul de vedere al teoriei demonstrației) în contextul celor trei binecunoscute presupuneri: a închiderii domeniului, a numerelor unice și a lumii închise.
În [14], o Bază de Date Logică (sau Deductivă) este definită prin:

- o mulțime de fapte elementare (tuple) și o mulțime de reguli (clauze Horn fără simboluri funcționale);
- o mulțime de constrîngeri de integritate (clauze Horn pure);
- o meta-regulă : negația ca eșec la căutare.

Faptele elementare și regulile constituie Baza de Date Extensională și, respectiv, baza de Date Intensională.

3.3.1. Reprezentarea unei baze de date logice

Pentru a descrie sugestiv un set de reguli au fost concepute diferite formalisme grafice, cum ar fi: Predicate connection Graphs [20], System Graphs [18], [19], Rule/Goal Graphs [32], [20].
Dintre aceste reprezentări ne vom concentra atenția pe ultima ca fiind cea mai bogată în informații.
Un graf regulă/scop reprezintă relațiile dintre reguli și predicate sub forma unui graf cu două tipuri de noduri:

- noduri-scop: corespund predicatelor dintr-o regulă;
- noduri-regulă: corespund regulilor.

Arcele corespund implicațiilor logice dintre predicate.

Față de celelalte reprezentări grafice, grafurile regulă/scop conțin predicate ornamentate. Un ornament al unui predicat n-ar este o secvență de lungime n de "b" și "f" (b = bound, f = free). De fapt, un predicat ornament se obține prin propagarea constantelor în graful regulă/scop.
Exemplu:
Fie Baza de Date Intensională:

r1: p1:-p3, p4.
r2: p2:-p4, p5.
r3: p3:-p6, p4, p3.
r4: p4:-p5, p3.
r5: p3:-p6.
r6: p5:-p5, p7.
r7: p5:-p6.
r8: p7:-p8, p9.

Acesteia îi corespunde graful regulă/scop următor:

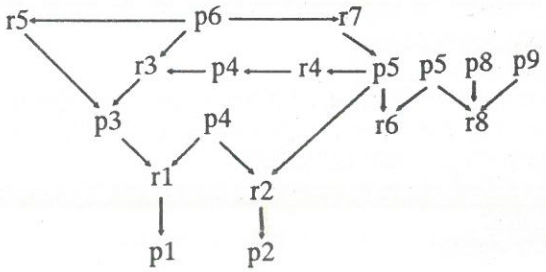


Fig.1

3.3.2. Recursivitatea

Măsura, în care un sistem deductiv rezolvă problema evaluarii predicatelor recursive, este criteriul esențial după care, acel sistem este apreciat. În această secțiune vom introduce unele notiuni legate de conceptul de recursivitate, util pentru înțelegerea caracteristicilor sistemului PYTHIA. Fie p și q două predicate. Se spune că p derivă pe q, notat p→q, dacă p apare în corpul unei reguli al cărei cap este q. Vom defini – →+ ca fiind închiderea tranzitivă a relației →.
Un predicat este recursiv dacă p→+p.
Două predicate p și q sînt mutual recursive dacă p→+q, q →+p
Se poate arăta ușor că relația → este o relație de echivalență pe mulțimea predicatelor recursive pe care o partiționează în clase disjuncte.
O regulă r: pi -p1, p2,..,pn este recursivă dacă există cel puțin un predicat pi, care este mutual recursiv cu p.
O regulă r este liniar recursivă dacă există pi unic în corpul său, care este mutual recursiv cu p.
Din motive practice este foarte utilă luarea în considerare a anumitor forme canonice de sisteme de reguli. Conceperea de strategii de evaluare, specifice acestor forme are rațiuni puternice, întrucît, în multe cazuri reale, este posibilă aducerea unui sistem de reguli la forma canonică.

4.5. Verificarea finitudinii răspunsurilor

Sistemele de reguli acceptate de sistem fac parte din categoria regulilor bottom-up evaluabile.
În cazul predicatelor aritmetice evaluabile, condiția de evaluabilitate cere ca variabilele acestora să fie instanțiate înainte ca orice evaluare să aibă loc. O singură excepție este permisă, și anume în cazul predicatului de egalitate de forma x=<expresie>, nu este necesar ca variabila x să fie legată (dacă nu apare în <expresie>).
În aceste condiții, DATALOG avînd proprietatea de completitudine, cererile pot fi calculate efectiv, iar răspunsurile sînt relații finite.
Probleme deosebite privind calculabilitatea efectivă și finitudinea vor fi abordate în versiunile ulterioare ale sistemului care vor extinde limbajul DATALOG, în sensul admiterii de simboluri funcționale în termeni. În acest caz, problema deciderii calculabilității efective pentru clauze HORN cu predicate de comparație este nedecidabilă, chiar dacă nu este implicată recursivitatea.

4.6. Arhitectura sistemului

Sistemul PYTHIA este compus din trei subsisteme slab cuplate între ele prin Baza de Cunoștințe și Baza de Metode, și anume:

- subsistemul de construire a Bazei de Cunoștințe;
- subsistemul de achiziție de metode;
- subsistemul de evaluare a cererilor.

Subsistemul de construire a Bazei de Cunoștințe are următoarele funcții:

- analiza sintactică a sistemelor de reguli: în această fază se verifică corectitudinea sintatică și evaluabilitatea bottom-up ale regulilor. În urma analizei sintactice, rezultă o formă intermediară specifică;
- construirea grafului de prelucrare contractat: această funcție se realizează în trei pași, și anume:
- transformarea formei intermediare anterior create în graf de prelucrare;
- "purificarea" predicatelor din graf, adică, împărțirea predicatelor în două clase: intensionale pure și extensionale pure. La acest pas, sînt introduse 0 predicate fictive intensionale;
- contractarea grafului de prelucrare prin punerea în evidență a clicilor recursive.
- canonizarea clicilor recursive: în această fază, se încearcă aducerea CC-nodurilor la una din formele canonice de clustere acceptate de sistem. Apoi, toate nodurile-scop ale grafului de prelucrare contractat, recursive sau nu, se etichetează conform următoarei tipologii:

a. noduri nerecursive:
   1. extensionale: <E>
   2. intensionale: <I>

b. noduri recursive:
   - canonice:
     1. închidere tranzitivă: <TC>
     2. liniar recursive: <LR>
     3. biliniar recursive:
   - necanonice: <NC>

Procesul de canonizare este compus din mai multe subprocese, și anume:

- simplificarea: dacă un predicat intensional intermediar S dintr-un cluster nu este implicat decît în cel mult un ciclu, el poate fi eliminat prin rezoluție simplă. Un caz interesant este simplificarea unor reguli mutual recursive în reguli non-mutual recursive;
- detectarea subexpresiilor comune: dacă există o subexpresie comună unor reguli diferite, care definesc același predicat, în anumite condiții subexpresia comună poate fi partajată;
- detectarea regulilor multiple de închidere tranzitivă;
- detectarea și tratarea recursivității la nivele multiple: dacă R și S sînt două predicate recursive, și R→+S, iar S→+R atunci S se află la un nivel de recursivitate mai coborît decît R [16]. Atunci, S poate fi evaluat înaintea lui R.

În afara transformărilor anterioare, subsistemul de canonizare are în vedere și unele transformări logice tipice, cum ar fi eliminarea tautologiilor.
Subsistemul de evaluare a cererilor implementează o strategie de evaluare bottom-up a arborelui de prelucrare canonizat, numită evaluarea semi-naivă. În faza inițială, cererea este analizată, iar constantele sînt propagate prin arborele de prelucrare ornamentînd nodurile acestuia. Propagarea constantelor poate face ca anumite ramuri ale arborelui de prelucrare să devină inaccesibile datorită detectării unor inconsistențe de unificare. Evaluarea propriu-zisă este specifică tipului de nod de evaluat. În cazul nodurilor nerecursive, evaluarea se reduce la transmiterea cererilor asociate către RDBMS ORACLE. Pentru nodurile recursive canonice, evaluarea constă în apelul unor proceduri built-in specifice. În sfîrșit, pentru nodurile recursive necanonice, se vor utiliza metodele din Baza de Metode. Deocamdată, Baza de Metode a sistemului PYTHIA cuprinde metoda Evaluării Semi-naive și metoda generalizată a Mulțimilor Magice [8].
Subsistemul de achiziție de metode are drept scop facilitarea integrării de noi metode și proceduri în Baza de Metode. În versiunile ulterioare ale sistemului PYTHIA, subsistemul de achiziție se va transforma într-un subsistem complex de achiziție și configurare.

.4.7. Dezvoltări ulterioare

Sistemul PYTHIA este în curs de realizare pe un IBM-PC like-computer, limbajul de implementare fiind Borland C++.
În viitor, eforturile de dezvoltare a sistemului PYTHIA se vor concentra în trei direcții principale:

- perfecționarea interioară a sistemului, prin achiziția de metode de evaluare complexe ([32], [27]);

- universalizarea limbajului DATALOG, prin introducerea negației stratificate [31] și a simbolurilor funcționale;
- introducerea unei componente de evaluare top-down (de tip PROLOG). Activarea acesteia va fi decisă de algoritmul lui Ullman-Van Gelder ([34], [33]), care este un algoritm semipolinominal în raport cu aritatea predicatelor de evaluat.

Concluzii

Lucrarea prezintă arhitectura și funcțiile unui sistem deductiv de evaluare a cererilor relaționale. În prima parte, sînt abordate soluții de rezolvare ale unor dificultăți de exprimare ale cererilor în algebra relațională. În acest context, sînt analizate unele propuneri legate, atît de extinderea algebrei relaționale, cît și de utilizarea limbajului PROLOG ca limbaj de interogare relațional. Soluția teoretică a problemei abordării logice a bazelor de date o constituie conceptul de Baze de Date Logice. În lucrare,sînt prezentate succint o serie de noțiuni legate de acest concept, cum ar fi recursivitatea, finitudinea și calculabilitatea efectivă a cererilor. De asemenea, sînt trecute în revistă caracteristicile esențiale ale strategiilor de evaluare a cererilor în contextul Bazelor de Date Logice. În partea a doua, este prezentat sistemul PYTHIA, care reprezintă o tentativă de a cupla slab un mecanism inferențial eficient cu DBMS-ul Oracle. Limbajul obiect al sistemului este DATALOG semantic echivalent cu Logica Clauzelor Horn pure. Sistemul este compus din trei subsisteme care au funcții de construire a Bazei de Cunoștințe, de achiziție de metode de evaluare și, respectiv, de evaluare a cererilor. Baza de Cunoștințe este structurată în structuri speciale de reguli, numite clustere. Baza de metode cuprinde o serie de proceduri built-in de evaluare a clusterelor canonice, precum și două metode complexe: evaluarea semi-naivă și metoda generalizată a mulțimilor magice. Acestea permit abordarea eficientă a cererilor recursive generale. Subsistemul de evaluare urmează o strategie bottom-up de parcurgere a unui arbore de prelucrare, asociat clusterului de interogat, în cursul căreia RDBMS-ul ORACLE acționează ca un server executînd cererile SQL generate.
În viitor, cercetările de dezvoltare se vor concentra în direcțiile măririi expresivității limbajului DATALOG și a integrării în sistem a unor noi strategii de evaluare, inclusiv de tip PROLOG.

BIBLIOGRAFIE

1. AGRAWAL, R.: Alpha: an Extension of Relational Algebra Express a Class of Recursive Queries. În: Proc. of Third Int. Con. on Data Eng., 1987, pp.37-45.
2. AGRAWAL, R.: Moving Selections into Linear Least Fixpoint. (Preprint).
3. BANCILHON, F., MAIER, D., SAGIV, Y., ULLMAN, J.: Magic Sets: Algorithms and Examples. În: MCC Internal Report, 1985, Austin, Texas.
4. BANCILHON, F.: Naive Evaluation of Recursively Defined Predicates. În: Proc. of the Islamadora Workshop on Databases & AI, Springer Verlag, Berlin, 1986, pp.73-82.
5. BANCILHON, F.: Evaluation des clauses de Horn dans les bases de données relationelles. În: INRIA, raport nr.585, Franța, 1985.
6. BANCILHON, F., RAMAKRISHAN, R.: An Amateur's Introduction to Recursive Query Processing Strategies, MCC Internal Report 1986, Austin, Texas.
7. BANCILHON, F., RAMAKRISHNAN, R.: Performance Evaluation of Data Intensiv Logic Programs. În: "Foundation of Deductive Databases and Logic Programming", Editura Morgan-Kaufman, Los Altos, 1988, pp.323-345.
8. BEERI, C., RAMAKRISHNAN, R.: On the Power of Magic. University of Wisconsin, Madison, TR # 770, iunie 1988.
9. BERCARU, R., GALOS, T., MIHAIL ,T., SĂFTOIU, E.: STAR - User's Guide. ICI, - Raport de Cercetare - 1988.
10. CHANG, C.L., WALKER, A.: PROSQL: a PROLOG Programming Interface with SQL/DS. În: "Expert Database Systems", North Holland Press, Amsterdam, 1986, pp.75-86.
11. CHIMENTI, D., ș.a.: An Overview of the LDL System. În: Bull. Data Engineering, vol.10, nr.4, 1987, pp.32-40.
12. CODD, E.F.: Relational Completness of Data Base Sublanguages. În: Courant Computer Science Symp., Prentice Hall, 1972, pp.140-175.
13. DAYAL, U., SMITH, J.M.: PROBE - a Knowledge-oriented Database Management System. În: Proc. Islamadora Workshop Large Scale Knowledge Base and Reasoning Systems, Islamadora, 1985, pp.208-214.
14. GALLAIRE, H., MINKER, J., NICOLAS, J.M.: Logic and Databases: a Deductive Approach. În: Computing Surveys, vol.16, no.2, 1984, pp.27-39.
15. HENSCHEN, L., NAQVI, S.: On Compiling Queries in Recursive First Order Databases. În: Journal of the ACM, vol.31, no.1, 1984, pp.36-61.
16. HAN, J., HENSCHEN, L.: Handling Redundancy in the Processing of Recursive Database Queries, North Western University EECS, TR 86-09-DBM-03, Canada.
17. HAN, J., HENSCHEN, L.: Processing Linear Recursive Database Queries Be Level and Cycle Merging, North Western University, EECS TR 87-5-DBM-1, Canada.
18. KIFER, M., LOZINSKII, E.: Filtering Data Flow in Deductive Databases. În: Proc. Int. Con. on Databases Theory, Toronto, 1986, pp.102-122.
19. KIFER, M., LOZINSKII, E.: A Framwork for an Efficient Implementation of Deductive Database.

În: Proc. 6th Advanced Database Symposium, Boston, 1986, pp. 49-62.
20. KOWALSKI, R.: Logic for Problem Solving, North Holland Press, Amsterdam, 1979.
21. KRISHNAMURTHY, R., ZANIOLO, C.: Optimization in a Logic Based Language for Knowledge and Data Intensive Applications. În: MCC Austin, Texas Technical Report, 1987.
22. MERRET, T.H.: Relational Information System. În: Reston Pub., Virginia, 1984.
23. MISSIKOFF, M., WIEDERHOLD, G.: Towards a Unified Approach for Expert and Database Systems. În: Expert Database Systems, North Holland Press, Amsterdam, 1986, pp.240-262.
24. MORRIS, K., NAUGHTON, J.F., ş.a.: YAWN! (Yet Another Window on NAIL!). În: Bull Data Engineering, vol.10, no.4, 1987, pp.73-90.
25. NAQVI, S., HENSCHEN, L.: Synthesizing Least Fixed Point Queries into Iterative Programs. În: Proc. IJCAI, Karlsruhe, 1983, pp.247-275.
26. PARKER, D.S., ş.a.: Logic Programming and Database. În: Expert Database Systems, North Holland Press, Amsterdam, 1986, pp.323-340.
27. RAMAKRISHNAN, R.: Magic Templatest: a Spellbinding Approach to Logic Program. University of Wisconsin, Madison, TR # 771, iunie 1988.
28. REITER, R.: On Closed World Databases. În: "Logic and Databases", Editura Plenum, NY, 1978, pp.10-19.
29. ROSENTHAL, A., HEILER, S., DAYAL, U., MANOLA, F.: Traversal Recursion: A Practical Approach Supporting Recursive, Applications. În: Proc. ACM-SIGMOD, 1986, Int. Conf. on Management of Data, Washington D.C., 1986, pp.97-110.
30. SCIORE, E., WARREN, D.S.: Towards an Integrated Database Prolog System. În: Expert Database Systems, North Holland Press, Amsterdam, 1986, 263-275.
31. SHMUELI, O., NAQVI, S.: Set Grouping and Layering in Horn Clause Programs. În: MCC Technical Report, Austin, Texas, 1987.
32. ULLMAN, J.D.: Implementation of Logical Query Languages for Databases. În: ACM Tran. on Database Systems, vol.10, no.3, 1985, pp.23-60.
33. ULLMAN, J.D.: Testing Applicability of Top-down Capture Rules. Stanford University, TR 1086, 1985.
34. van GELDER, A.: Deriving Relations among Argument Sizes in Logic Programs. Stanford University TR, 1986.
35. ZANIOLO, C.: Prolog: A Database Query Language for All Seasons. În: "Expert Database Systems", North Holland Press, Amsterdam, 1986, pp.276-290.
36. ZLOOF, M.,M.: Query-by-example: Operations on Transitive Closure, RC 5526, IBM, Yorktown Hts., New York, 1975.