EVALUATING QUERIES IN DEDUCTIVE DATABASES BY GENERATING.

Eliezer L. Lozinskii
Institute of Mathematics and Computer Science
The Hebrew University of Jerusalem, Israel.

ABSTRACT.

A strong advantage of bottom up generating techniques is their ability to guarantee finiteness of all inferences in a deductive database system involving recursive axioms. But a "brute force" generating answers to a query would be very inefficient producing many facts useless for the query evaluation. An economy generating method is presented based on discovering explicit facts relevant to the query, and applying preselected axioms as generating rules. The method is proved to be complete in the sense that it generates all the existing answers to the query in a finite time.

1. INTRODUCTION AND MOTIVATION.

Recently deductive database systems have become an area of extensive research (Chang, 1976.1978,1981, Gallaire, Minker and Nicolas, 1978, Henschen and Naqvi, 1982.1984. Kellogg.Klahr and Travis. 1978. Klahr.1978. Kowalski,1978, McKay and Shapiro, 1981, Minker. 1975.1978, Minker and Nicolas.1983, Naqvi and Henschen, 1980, Naqvi, Fishman and Henschen, 1982, Nicolas and Galiaire.1978, Nicolas and Yazdanian.1978, Reiter.1978. Shapiro and McKay, 1980, Ullman.1984). Consider a database *DB* (relational database, in particular) consisting of a set of *relations {Ri}, extensions* of some relations $\{R_i\}$, and axioms $\{A_j\}$ in the form of *Horn clauses.* A relation presented explicitly by its extension is called base *relation,* while a relation defined by an axiom is a *virtual relation.* Base relations constitute the *extensional database (EDB),* axioms are contained in the *intentional database (IDB).* A query, *q*, is a formula (in a first order language), and answering *q* means finding a proof that *q* is implied by the *EDB* and *IDB.* Suppose a relation, *R,* is defined in terms of itself. If such a *recursive definition* of *R* is used in the course of a query processing this may lead to an infinite search for a proof. A number of elegant solutions to the problem of handling recursive axioms have been proposed (Chang. 1981, Henschen and Naqvi, 1982.1984, McKay and Shapiro, 1981, Minker and Nicolas, 1983, Naqvi and Henschen, 1980, Reiter,197B, Shapiro and McKay, 1980, Ullman,1984).

Most of known methods employ top-down goal-driven deduction which may be drawn by recursive axioms into an inflnite (or at least, too long) evaluation process. Another serious problem of goal-driven strategies is a fast growth of the search space because being not restricted enough by a specific information the search develops in many directions including the ones neither relevant to the query nor supported by the extensional database. On the other hand, it seems very natural to exploit the problem-specific information provided by a query, and the actual data stored in the *EDB* for directing the query answering process. As an extreme example consider a query about the academic achievements of a student, Tom Steel. If there is no Tom Steel among the students of the University then any search and deduction will be useless. Instead, it could be found out prior to a processing that the sought name does not appear in the *EDB,* so the search should be stopped in the very beginning.

2. ANSWERS CAN BE GENERATED BOTTOM-UP.

Given a Horn axiom. A_1 and a set, *F,* of facts asserting its antecedent, the axiom can be used for producing a fact corresponding to its consequent and implied by *F.* Then A_1 plays the role of a *generation rule* (Minker and Nicolas, 19B3, Nicolas and Gailaire,1978. Nicolas and Yazdanian.1978). If an instance of the consequent of an axiom, A_2, is considered as a *goal* of deduction then by backward reasoning it can be *reduced* to a set of subgoals corresponding to the predicates of the antecedent of A_2; in this case A_2 is said to be used as a *derivation rule.* Using a recursive axiom (or a *cycle* of axioms) as a derivation rule can lead to an infinite process. On the other hand, it is shown in (Minker and Nicolas, 1983) that when axioms are used as generation rules infinite inferences do not arise. Indeed, because the language describing a database is function-free the *Domain Closure Axiom* (Reiter,1978) ensures that no new constants can be produced during a processing, and so out of a finite number of constants contained in the database only finitely many different tuples of certain kind can be generated.

The flniteness of generating suggests an endeavour to answer queries by using axioms as generation rules only (or mostly). Trivially: could a set of facts have been produced by applying all axioms of the *IDB* as generation rules to all facts of the *EDB,* this set would contain all possible answers of the database to any query. Alas, the set must be too huge to be practical (because *m* constants can yield $0(m^n)$ different n-tuples). In this respect several questions arise: Can an answer to a query be generated in such a way that (almost) only those facts are produced which are necessary for the query evaluation? How axioms and explicit facts that are relevant to the query can be determined? The rest of the paper presents an approach to answering these questions affirmatively.

Let axioms be presented in the form

$$P_1 \wedge P_2 \wedge \cdots \wedge P_r \wedge CE \rightarrow P_{r+1} \qquad (1)$$

such that $P_1, P_2, \ldots, P_{r+1}$ are predicate signs, *CE* denotes a conjunction of equalities between variables or

constants, all terms of $P_1P2....P_{r+1}$ *are* distinct variables, all variables are typed and universally quantified, so the quantifiers are dropped, and types are shown next to the corresponding terms. For instance,

$$ANCESTOR(x_1/\,name, x_2/\,name)$$
$$\wedge\, FATHER(y_1/\,name, y_2/\,name)$$
$$\wedge\, (x_2 = y_1) \wedge (x_1 = z_1) \wedge (y_2 = z_2)$$
$$\rightarrow ANCESTOR(z_1/\,name, z_2/\,name).$$

Relations and axioms of a database can be represented by a *system graph, SG,* in the following way. Each axiom is represented by a single *ax-node,* and each relation by a single *main rel-node.* Let $<R>, <A>$ denote the nodes of *SG* representing, respectively, a relation, *R*, and an axiom, $A: P_1 \wedge P_2 \wedge \cdots \wedge P_r \wedge CE \rightarrow P_{r+1}$. Then the system graph contains arcs $(<A>, <P_{r+1}>)$ and $(<P_i>, <A>)$ for $i = 1,2, \ldots, r$. *SG* is an *AND/OR graph* (Nilsson,197l), where rei-nodes play the role of arid-nodes, and ax-nodes that of or-nodes. If a relation, *P,* is defined recursively, that is, it occurs in the consequent of an axiom as well as in its antecedent, then the occurrence of *P* in the antecedent is represented by a main rel-mode, while its occurrence in the consequent — by a *secondary* rel-node. The latter is connected to the main rel-node of *P* through an *auxiliary* ax-node. Figure 1 shows a *SG* corresponding to (2). Because all terms appearing in rel-nodes are distinct variables (while all specific relationships among the terms of an axiom are displayed by the corresponding ax-node) all occurrences of a relation in the database are represented by a single main rel-node (except its occurrence in the consequent of a recursive axiom). An *atomic query* has the form (existential quantifiers are dropped):

$$q : Q(t_1/\,T_1, t_2/\,T_2, \ldots, t_k/\,T_k). \qquad (3)$$

The rel-node representing *Q* is called the *target of q* (denoted *targ(q)*).

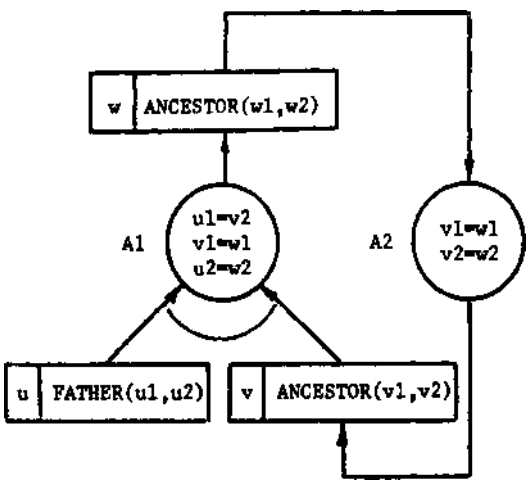

Figure 1.

Let V, W, E denote sets of rel-nodes, ax-nodes, arcs, respectively, of a system graph, *SG*. Define the following sets of nods:

immediate predecessors of a node, v,

$$IMPRED(v) = \{u \mid (u,v) \in E\};$$

predecessors of v, $PRED(v)$

$$= IMPRED(v) \cup \{u \mid \exists w((u,w) \in E \wedge w \in PRED(v))\};$$

base rel-nodes, BR, is the set of rel-nodes representing base relations;

base predecessors of v,

$$BPRED(v) = PRED(v) \cap BR;$$

immediate successors of v,

$$IMSUC(v) = \{u \mid (v,u) \in E\};$$

successors of v, $SUC(v)$

$$= IMSUC(v) \cup \{u \mid \exists w((w,u) \in E \wedge w \in SUC(v))\}.$$

Definition. An axiom. *A,* is said to be *executed* on a set of facts, *F,* if the facts of *F* which correspond to the antecedent of *A* are used for producing of all generable facts corresponding to the consequent of *A*.

The following propositions hold (proofs are omitted in this abstract for brevity):

Proposition 1. Let *q* : *Q* be a query, *DP* - a top-down derivation proof of *q* , *ANSW* — the set of answers (facts) supplied for *q* by *DP, DER* — the set of axioms used in *DP* as derivation rules, *F* — the set of facts (belonging to base relations) used in *DP*. Then all facts of *ANSW* can be produced out of *F* by using the axioms of *DER* as generation rules only.

Proposition 2. Ail answers to a query, *q* :Q, can be generated by executing a set of axioms, *GEN,* called *generators* of *q,* on a set of facts, Φ, such that Φ is represented by *BPRED(Q),* and *GEN* consists of axioms displayed by predecessors of *Q*, $GEN = W \cap PRED(Q)$.

3. AN ECONOMY WAY.

A "brute force" generating answers to a query according to Proposition 2 would be very inefficient producing many facts unnecessary for the query evaluation because in most practical cases *DER* and *F* are small subsets of *GEN* and Φ, respectively. An economy generating process should use those facts and axioms which are relevant to the query. Thus, the question is how to discover them.

Definition. In the course of generating facts a term, t_{1t} can be assigned a value of another term. t_2, (when expressions are unified (Chang and Lee. 1973) or axioms are executed); we say that t_2 can *migrate* to t_x . Assume that all terms of relations are distinct variables, and define a *migration set* of x, $MIG(x)$, containing all terms to which *x* can migrate:

(a) if there is an axiom,

$$A: \cdots \wedge P(\cdots x \cdots) \wedge \cdots \wedge (x = y) \wedge \cdots \rightarrow R(\cdots y \cdots)$$

then $y \in MIG(x)$;

(b) if there are any two predicates such that their unifier contains a substitution x/y then $y \in MIG(x)$;
(c) if $y \in MIG(x)$ and $z \in MIG(y)$ then $z \in MIG(x)$;
(d) no other rules define $MIG(x)$.

$MIG(x)$ can be easily computed by traversing directed paths in the system graph starting from the rel-node that represents the relation containing x, and checking the equalities displayed by the ax-nodes belonging to these paths.

Definition. Consider a query, $q: Q(\cdots, x_i = \alpha_i, \cdots)$, with a variable, x_i, bound to a constant, α_i, and a fact, $P(y_1=\beta_1, \ldots, y_j=\beta_j, \ldots, y_k=\beta_k)$, with all variables bound. We say that P is *relevant* to q if P contains y_j such that $x_i \in MIG(y_j)$, and $\beta_j = \alpha_i$. If q specifies no bound variables then P is relevant to q if there are x_i and y_j such that $x_i \in MIG(y_j)$.

During a query processing no new constants are produced, but the constants appearing in the *EDB* just migrate from tuple to tuple. This implies

Proposition 3. If a set of facts, F, has been used for producing an answer to a query, q, then F must contain some facts relevant to q.

This observation suggests an *economy way* of generating answers to $q : Q$. Select a subset, φ, of Φ containing facts relevant to q. Start generating from φ by trying to execute generators of q on the facts of φ. Consider an axiom, A: $P_1 \wedge \cdots \wedge P_i \wedge \cdots \wedge P_k \rightarrow P_{k+1}$. If the antecedent of A is satisfied by φ then execute A producing a new fact corresponding to P_{k+1}, and insert it into φ. Let P' be a fact of φ unifiable with P_i by a unifier, ϑ. In order to execute A as a generation rule using P' we need facts satisfying $(P_1 \wedge \cdots \wedge P_{i-1} \wedge P_{i+1} \wedge \cdots \wedge P_k)\vartheta$. To get them solve all *auxiliary* queries $q_j : P_j\vartheta$ for $j = 1,2,\ldots,i-1,i+1,\ldots,k$ using the facts of Φ. Upon generating answers to all q_j execute A augmenting φ by newly produced facts, and keep executing axioms of *GEN* on φ. Terminate if no new facts can be produced. Select from φ answers to q.

This approach is implemented by an algorithm called APEX because it employes extensively procedures *APPLY* and *EXECUTE.* Given an axiom. *A,* and a set of facts, *F, APPLY* returns a set of all facts corresponding to the consequent of *A* such that each of them is deducibie from the *EDB* and *IDB* by a generating process that uses necessarily *F* and *A* while *EXECUTE* returns a set of all facts generable out of *F* only, by using *A* as a generation rule.

All facts are clustered into extensions of the corresponding relations. This allows to employ well developed tools of Relational Databases, and efficient algorithms for performing procedures on sets of facts instead of tuple-at-a-time processing- For every rel-node, R, the sets of its base predecessors, *BPRED(R),* and generators, *GEN{R),* are preprocessed at the time of designing of the system graph to reduce run-time processing.

Each fact has an associated variable the value of which is the distance in the system graph from the node representing the fact to the target node of *q.* Axioms of *GEN* are executed in the increasing order of their distances from *targ(q),* that is, first are used the facts nearest to *targ(q),* and therefore likely to produce an answer to *q* in a short time. This *ordered execution* of axioms improves performance especially for such queries that do not require all the possible answers (e.g. "Give a name of *any employee,* who ..."), or for which the number of all answers is known (e.g. "Who were the *(four)* grandparents of John?"), including such "boolean" queries as *"Is there* ...?"

Any axiom is executed or applied as soon as the necessary data are available. So *APEX* is well suited for a parallel multiprocess implementation, in particular, in such a manner that rel-nodes and ax-nodes are associated with a set of communicating processes (Hoare,1978) exchanging messages along the arcs of the *SG.*

It is proved that *APEX* is *complete* in the following sense:

Proposition 4. Given a query, *APEX* generates all answers implied by the database.

4. AN ILLUSTRATION.

In this section *APEX* is illustrated by an example taken from (Henschen and Naqvi, 1984) involving a recursive definition. Consider a database containing relations *M,N,P,R,S* which may be thought of as special kinds of relations between ancestors and descendants. $M(m_1,m_2)$. $P(p_1,p_2)$, $R(r_1,r_2)$ are base relations with the extensions:

M

$m1$	$m2$
b	c
c	e
f	l

P

$p1$	$p2$
d	a
g	h

R

$r1$	$r2$
e	d
l	g

$N(n1,n2)$ and $S(s1,s2)$ are virtual relations defined by the following axioms (all variables are of the same type, e.g., person names, so type signs are dropped):

$$A1: S(s1,s2) \wedge P(p1,p2) \wedge (s2=p1) \wedge (s1=n1)$$

$$\wedge (p2=n2) \rightarrow N(n1,n2);$$

$$A2: M(m1,m2) \wedge N(n1,n2) \wedge (m2=n1) \wedge (m1=s1)$$

$$\wedge (n2=s2) \rightarrow S(s1,s2);$$

$$A3: R(r1,r2) \wedge (r1=n1) \wedge (r2=n2) \rightarrow N(n1,n2).$$

The system graph of this database is depicted in Figure 2.

Let a query be $q: S(x,a)$ which means "Find all S-ancestors of a". Generating an answer for q (displayed in Figure 3) may be informally described as follows. First, select from the *EDB* facts relevant to q. $P(d,a)$ is such a fact (and the only one) because $s2 \in MIG(p2)$. In order to execute $A1$ on $P(d,a)$ we need a fact of the form $S(x_1,d)$. To get such a fact solve $q_1: S(x_1,d)$. $R(e,d)$ is relevant to q_1 because $s2 \in MIG(r2)$. $A3$ executed on $R(e,d)$ generates $N(e,d)$. To execute $A2$ on

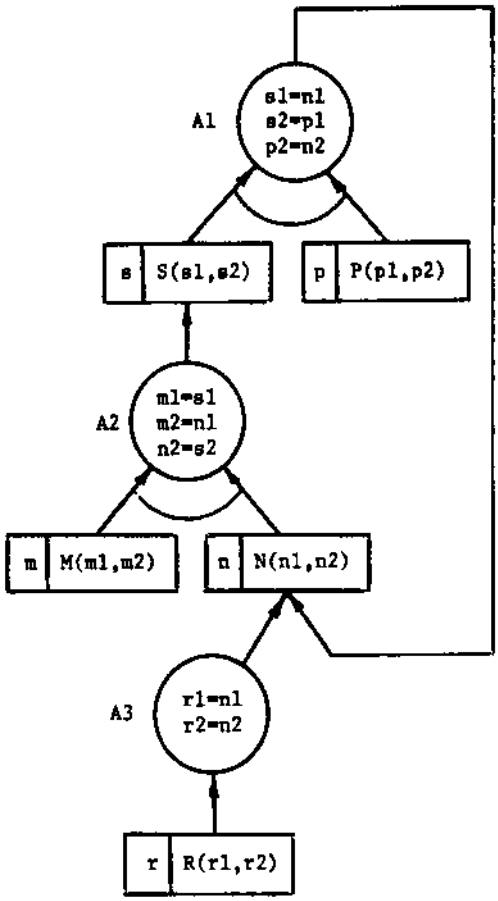

Figure 2.

$N(e,d)$ a fact $M(x_2,e)$ is need, so solve q_2:$M(x_2,e)$. This is simple because M is a base relation, and a proper selection produces $M(c,e)$. Now $A2$ generates $S(c,d)$, and $A1$ $N(c,a)$. A fact $M(x_3,c)$ is needed and produced as an answer to q_3:$M(x_3,c)$ by selecting $M(b,c)$ from the corresponding extension. Finally, $A2$ generates the answer $S(b,a)$, and the process terminates because no new answers to q can be produced.

5 CONCLUSION.

APEX employs a certain kind of *bidirectionality:* generating of facts is performed bottom-up, while the axioms used for evaluation of a query, *q: Q,* are determined in a top-down manner as predecessors of *Q* in the system graph. A bottom-up generation is directed by actual information stored in the *EDB* in contrast to a top-down derivation process that can be developed in directions not supported by the database. But although *APEX* starts only with relevant facts and restricts the set of used axioms, yet a number of facts irrelevant to the query are likely to be generated in the course of the query evaluation. And so, in some cases a generating process can be inferior to a top-down derivation. Ullman (19B4) describes *very* convincing examples in which

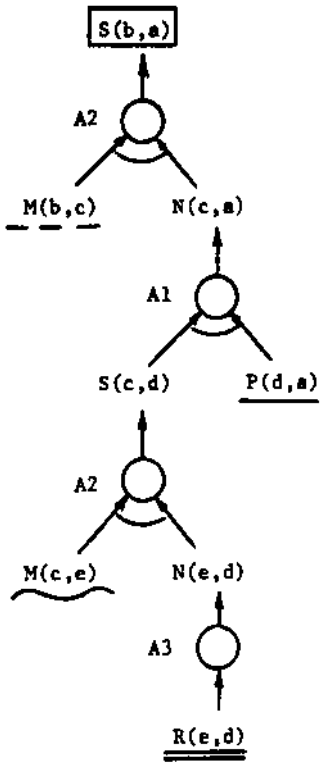

Figure 3.

derivation is more efficient than generation, like in the case (section V) where axioms define recursion on structure with a certain kind of decreasing structural complexity. On the other hand, in systems with a very developed *IDB* and relatively not large *EDB* generation promises to be thriftier than derivation because of a significant reduction of the search space. Hence, to achieve a high performance of each specific database system a certain combination of top-down and bottom-up techniques should be employed depending on the actual characteristics of the system.

For all that, a strong advantage of bottom-up generating is that it by itself guarantees finiteness of all inferences, and so it deserves certain attention like the one it is paid in this paper. It is shown also that generating methods may influence database structuring providing a trade-off between storage space and query evaluation time.

ACKNOWLEDGMENTS.

The author is grateful to Michael Rabin and Jeffrey Ullman for most inspiring discussions of the subject of this paper.

REFERENCES.

[1] Chang, C.-L. DEDUCE - a deductive query language for relational databases. In *Artificial Intelligence and Pattern Recognition,* Chen, C. G. (Ed.), Academic Press. New York, 1976, 108-134.

[2] Chang,C.-L. DEDUCE 2: further investigations of deduction in relational databases. In *Logic and Data Bases,* Gallaire, H., and Minker, J. (Eds.), Plenum Press, New York, 1978, 201-236.

[3] Chang. C.-L On evaluation of queries containing derived relations in a relational data base. In *Advances in Data Base Theory,* Gallaire, H., Minker, J., and Nicolas, J.-M. (Eds.), v.I, Plenum Press, New York. 1981, 235-280.

[4] Chang, C.-L., and Lee, R.C.-T. *Symbolic Logic and Mechanical Theorem Proving,* Academic Press, New York, 1973.

[5] Gallaire, H., Minker, J., and Nicolas, J.-M. An overview and introduction to logic and data bases. In *Logic and Data Bases,* Gallaire, H., and Minker, J. (Eds.) Plenum Press. New York, 1978. 3-32.

[6] Henschen, L. J., and Naqvi, S.A. On compiling queries in recursive first-order databases. Proc. Workshop on Logical Bases for Data Bases, Nicolas J.-M. (Ed.), Toulouse, 1982.

[7] Henschen, L. J., and Naqvi. S. A. On compiling queries in recursive first-order databases. *J. ACM 31,* 1 (Jan. 1984). 47-85.

[8] Hoare, C, A. R. Communicating sequential processes. *Comm ACM 21,* 8 (Aug. 1978), 666-677.

[9] Horn, A. On sentences which are true on direct unions of algebras. *J. Symb. Logic 16*(I95I), 14-21.

[10] Kellogg, C, Kiahr. P., and Travis, L. Deductive planning and pathfInding for relational data bases. In *Logic and Data Bases,* Gallaire, H., and Minker, J. (Eds.), Plenum Press, New York, 1978, 179-200.

[11] Klahr, P. Planning techniques for rule selection in deductive question-answering. In *Pattern-Directed Inference Systems,* Waterman. D. A., and Hayes-Roth. F. (Eds.), Academic Press, New York, 1978, 223-240.

[12] Kowalski. R. Logic for data description. In *Logic and Data Bases,* Gallaire, H., and Minker, J. (Eds.), Plenum Press, New York. 1978. 77-106.

[13] McKay, D. P.. and Shapiro, S.C. Using active connection graphs for reasoning with recursive rules. Proc. 7th Int. Joint Conf. on Artificial Intelligence, Vancouver, 1981, 368-374.

[14] Minker, J. Performing inference over relational databases. ACM SJGM0D Int. Conf. on Management of Data, San Jose, 1975, 79-91.

[15] Minker, J. An experimental relational database system based on logic. In *Logic and Data Bases,* Gallaire, H., and Minker, J. (Eds.), Plenum Press, New York, 1978, 107-147.

[16] Minker, J. Search strategy and selection function for an inferential relational system. *ACM TODS 3,* 1 (March 1978). 1-31.

[17] Minker, J., and Nicolas, J. -M. On recursive axioms in deductive databases. *Information Systems 8,* 1 (1983), 1-13.

[18] Naqvi, S.A., and Henschen. L. J. Performing inferences over recursive data bases. Proc. First Annual Nat. Conf. on Artificial Intelligence, Stanford, 1980, 263-285.

[19] Naqvi, S. A., Fishman, D. H., and Henschen, L. J. An improved compiling technique for first-order databases. Proc. Workshop on Logical Bases for Data Bases, Nicolas, J. -M. (Ed.), Toulouse, 1982.

[20] Nicolas. J. -M., and Gallaire. H. Data base: theory vs. interpretation. In *Logic and Data Bases,* Gallaire. H.. and Minker. J. (Eds.). Plenum Press. New York. 1978. 33-54.

[21] Nicolas, J. -M., and Yazdanian, K. Integrity checking in deductive data bases.In *Logic and Data Bases,* Gallaire, H., and Minker, J. (Eds.), Plenum Press, New York, 1978, 325-346.

[22] Nilsson, N. J. *Problem-Solving Methods in Artificial Intelligence,* McGraw-Hill, 1971.

[23] Reiter, R. On structuring a first order data base. Proc. 2nd Nat. Conf. of the Canadian Society on Computational Studies of Intelligence, Perrault, R. (Ed.), Toronto, 1978, 90-99.

[24] Reiter, R. On closed world data bases. In *Logic and Data Bases,* Gallaire. H.. and Minker. J. (Eds.), Plenum Press. New York. 1978, 55-76.

[25] Reiter, R. Deductive question-answering on relational data bases. In *Logic and Data Bases,* Gallaire, H.. and Minker, J. (Eds.). Plenum Press, New York, 1978, 149-177.

[26] Shapiro. S. C, and McKay, D. P. Inference with recursive rules. Proc. First Annual Nat. Conf. On Artificial Intelligence. Stanford, 1980, 151-153.

[27] Ullman, J. D. Implementation of logical query languages for databases. Unpublished manuscript. Jerusalem, 1984.