# A Survey of Research on Deductive Database Systems

Raghu Ramakrishnan
*University of Wisconsin – Madison*

Jeffrey D. Ullman
*Stanford University*

November 17, 1993

**Abstract**

The area of deductive databases has matured in recent years, and it now seems appropriate to reflect upon what has been achieved and what the future holds. In this paper, we provide an overview of the area and briefly describe a number of projects that have led to implemented systems.

## 1 Introduction

Deductive database systems are database management systems whose query language and (usually) storage structure are designed around a logical model of data. As relations are naturally thought of as the "value" of a logical predicate, and relational languages such as SQL are syntactic sugarings of a limited form of logical expression, it is easy to see deductive database systems as an advanced form of relational systems.

Deductive systems are not the only class of systems with a claim to being an extension of relational systems. The deductive systems do, however, share with the relational systems the important property of being *declarative*, that is, of allowing the user to query or update by saying *what* he or she wants, rather than *how* to perform the operation. Declarativeness is now being recognized as an important driver of the success of relational systems. As a result, we see deductive database technology, and the declarativeness it engenders, infiltrating other branches of database systems, especially the object-oriented world, where it is becoming increasingly important to interface object-oriented and logical paradigms in so-called *DOOD* (Declarative and Object-Oriented Database) systems.

In this survey we look at the key technological advances that led to the successful implementation of deductive database systems. As with the relational systems earlier, many of the problems concern code optimization, the ability of the system to infer from the declarative statement of what is wanted an efficient plan for executing the query or other operations on the data. Another important thrust has been the problem of coping with negation or nonmonotonic reasoning, where classical logic does not offer, through the conventional means of logical deduction, an adequate definition of what some very natural logical statements "mean" to the programmer.

This survey is not intended to be comprehensive; for example, we have not touched upon several important topics that have been explored actively in the literature, such as coupling existing Prolog and database systems, integrity constraint checking, parallel evaluation, theoretical results on complexity and decidability, many extensions of the Horn-clause paradigm (e.g., disjunctive databases, object-oriented data models), updates, and many specialized approaches to evaluation of certain classes of programs (e.g., bounded recursion, "chain-like" queries, transitive-closure-related queries, semantic query optimization). Several interesting results have been obtained in these areas, but we have chosen to limit the focus of this paper.

## 1.1 Prolog and Databases

The current crop of deductive systems drew inspiration from programming language research, especially Prolog. In a sense, deductive systems are an attempt to adapt Prolog, which has a "small-data" view of the world, to a "large-data" world. Prolog implementations have focused, as is typical for programming languages, on main-memory execution. There are two points to consider:

- Prolog's depth-first evaluation strategy leads to infinite loops, even for positive programs and even in the absence of function symbols or arithmetic. In the presence of large volumes of data, operational reasoning is not desirable, and a higher premium is placed upon *completeness* and *termination* of the evaluation method.
- In a typical database application, the amount of data is sufficiently large that much of it is on secondary storage. Efficient access to this data is crucial to good performance.

The first problem is adequately addressed by memoing extensions to Prolog evaluation. For example, one can efficiently extend the widely used Warren abstract machine Prolog architecture [War89].

The second problem turns out to be harder. The key to accessing disk data efficiently is to utilize the set-oriented nature of typical database operations and to tailor both the clustering of data on disk and the management of buffers in order to minimize the number of pages fetched from disk. Prolog's tuple-at-a-time evaluation strategy severely curtails the implementor's ability to minimize disk accesses by re-ordering operations. The situation can thus be summarized as follows: Prolog systems evaluate logic programs efficiently in main-memory, but are tuple-at-a-time, and therefore inefficient with respect to disk accesses. In contrast, database systems implement only a nonrecursive subset of logic programs (essentially described by relational algebra), but do so efficiently with respect to disk accesses.

The goal of deductive databases is to deal with a superset of relational algebra that includes support for recursion in a way that permits efficient handling of disk data. Evaluation strategies should retain Prolog's goal-directed flavor, but be more *set-at-a-time*. There are two aspects to set-orientation:

- The run-time computation should utilize traditional relational operations such as selects, projects, joins and unions; thus, conventional database processing techniques can be utilized.
- The overall computation should be organized so as to make as many operations as possible (logically) concurrent, thereby creating more flexibility in terms of re-ordering operations. In particular, it is desirable to generate and process *sets of goals*, rather than proceed one (sub)goal at a time.

Handling of disk-resident data can be addressed by building Prolog systems that support persistent data (while retaining the usual evaluation strategy) or by coupling existing Prolog and database systems. These approaches have the drawback that the interface to the disk data, or database system, becomes a potential tuple-at-a-time bottleneck. Alternatively, we can develop new technology and systems to deal with the requirements of deductive databases; this is the focus of the present paper.

## 2 Notation, Definitions, and Some Basic Concepts

Deductive database systems divide their information into two categories:

1. *Data*, or facts, that are normally represented by a predicate with constant arguments (by a *ground atom*). For example, the fact $parent(joe, sue)$, means that Sue is a parent of Joe. Here, *parent* is the name of a predicate, and this predicate is represented *extensionally*, that is, by storing in the database a relation of all the true tuples for this predicate. Thus, $(joe, sue)$ would be one of the tuples in the stored relation.
2. *Rules*, or program, which are normally written in Prolog-style notation as:

   $$p :- q_1, \ldots, q_n.$$

   This rule is read declaratively as "$q_1$ and $q_2$ and ...and $q_n$ implies $p$." Each of $p$ (the *head*) and the $q_i$'s (the *subgoals* of the *body*) are *atomic formulas* (also referred to as *literals*), consisting of a predicate applied to *terms*, which are either constants, variables, or function symbols applied to terms. Programs in which terms are either constants or variables are often referred to as *Datalog* programs. The data is often referred to as the *EDB* and the rules as the *IDB*. [1] Following Prolog convention, we use names beginning with lower-case letters for predicates, function symbols, and constants, while variables are names beginning with an upper-case letter. In later sections we also consider programs that contain features like negation and aggregation (e.g., sum) operations applied to subgoals.

**Example 2.1** Consider the following program.

$$\begin{aligned} sg(X,Y) &:- \quad flat(X,Y). \\ sg(X,Y) &:- \quad up(X,U), sg(U,V), down(V,Y). \end{aligned}$$

[1] *Extensional* and *intensional* databases.

Here, $sg$ is a predicate ("same-generation"), and the head of each of the two rules is the atomic formula $p(X,Y)$. $X$ and $Y$ are variables. The other predicates found in the rules are $flat$, $up$, and $down$. These are presumably stored extensionally, while the relation for $sg$ is *intensional*, that is, defined only by the rules. Intensional predicates play a role similar to views in conventional database systems, although we expect that in deductive applications there will be large numbers of intensional predicates and rules defining them, far more than the number of views defined in typical database applications.

The first rule can be interpreted as saying that individuals $X$ and $Y$ are at the same generation if they are related by the predicate $flat$, that is, if there is a tuple $(X,Y)$ in the relation for $flat$. The second rule says that $X$ and $Y$ are also at the same generation if there are individuals $U$ and $V$ such that:

1. $X$ and $U$ are related by the $up$ predicate.
2. $U$ and $V$ are at the same generation.
3. $V$ and $Y$ are related by the $down$ predicate.

These rules thus define the notion of being at the same generation recursively. Since common implementations of SQL do not support general recursions such as this example without going to a host-language program, we see one of the important extensions of deductive systems: the ability to support declarative, recursive queries. $\square$

The optimization of recursive queries has been an active research area, and has often focused on some important classes of recursion. We say that a predicate $p$ *depends upon* a predicate $q$ — not necessarily distinct from $p$ — if some rule with $p$ in the head has a subgoal whose predicate either is $q$ or (recursively) depends on $q$. If $p$ depends upon $q$ and $q$ depends upon $p$, $p$ and $q$ are said to be *mutually recursive*. A program is said to be *linear recursive* if each rule contains at most one subgoal whose predicate is mutually recursive with the head predicate. [2]

# 3 Optimization Techniques

Perhaps the hardest problem in the implementation of deductive database systems is designing the query optimizer. While for nonrecursive rules, the optimization problem is similar to that of conventional relational optimization, the presence of recursive rules opens up a variety of new options and problems. There is an extensive literature on the subject, and we shall attempt here to give only the most basic ideas and motivation.

[2] Sometimes, a more restrictive definition is used, requiring that no two distinct predicates can be mutually recursive, or even that there be at most one recursive rule in the program. We shall not worry about such distinctions.

## 3.1 Magic Sets

The problem addressed by the magic-sets rule rewriting technique is that frequently a query asks not for the entire relation corresponding to an intensional predicate, but for a small subset. An example would be a query like $sg(john, Z)$, that is, "who is at the same generation as John," asked of the predicate defined in Example 2.1. It is important that we answer this query by examining only the part of the database that involves individuals somehow connected to John.

A top-down, or backward-chaining search would start from the query as a goal and use the rules from head to body to create more goals, and none of these goals would be irrelevant to the query, although some may cause us to explore paths that happen to "dead end," because data that would lead to a solution to the query happens not to be in the database. Prolog evaluation is the best known example of top-down evaluation. However, the Prolog algorithm, like all purely top-down approaches, suffers from some problems. It is prone to recursive loops, it may perform repeated computation of some subgoals, and it is often hard to tell that all solutions to the query goal have been found.

On the other hand, a bottom-up or forward-chaining search, working from the bodies of the rules to the heads, would cause us to infer *sg* facts that would never even be considered in the top-down search. Yet bottom-up evaluation is desirable because it avoids the problems of looping and repeated computation that are inherent in the top-down approach. Also, bottom-up approaches allow us to use set-at-a-time operations like relational joins, which may be made efficient for disk-resident data, while the pure top-down methods use tuple-at-a-time operations.

Magic-sets is a technique that allows us to rewite the rules for each *query form* (i.e., which arguments of the predicate are bound to constants, and which are variable), so that the advantages of top-down and bottom-up methods are combined. That is, we get the focus inherent in top-down evaluation combined with the looping-freedom, easy termination testing, and efficient evaluation of bottom-up evaluation. Magic-sets is a rule-rewriting technique. We shall not give the method, of which many variations are known and used in practice. [Ull88a] contains an explanation of the basic techniques, and the following example should suggest the idea.

**Example 3.1** Given the rules of Example 2.1, together with the query $sg(john, Z)$, a typical magic-sets transformation of the rules would be:

$$
\begin{array}{rcl}
sg(X,Y) & :- & magic\_sg(X), flat(X,Y). \\
sg(X,Y) & :- & magic\_sg(X), up(X,U), sg(U,V), down(V,Y). \\
magic\_sg(U) & :- & magic\_sg(X), up(X,U). \\
\multicolumn{3}{l}{magic\_sg(john).}
\end{array}
$$

Intuitively, the *magic_sg* facts correspond to queries or subgoals. The definition of the magic_sg predicate mimics how goals are generated in a top-down evaluation. The set of *magic_sg* facts is used as a 'filter' in the rules defining $sg$, to avoid generating facts that are not answers to some subgoal. Thus, a purely bottom-up, forward chaining evaluation of the rewritten program achieves a restriction of search similar to that achieved by top-down evaluation of the original program. □

The original paper on magic sets was [BMSU86], and its extension to general programs was in [BR87b]. Independently, the article [RLK86] described the "Alexander method," which is essentially the "generalized supplementary magic sets method" of [BR87b], for the case of left-to-right evaluation within rules. There are a number of other approaches optimizing rules that had similar effects without rewriting rules. These include Earley deduction [PW83], Query-subquery [Vie87, Vie86], Sygraf [KL86], and related tabulation techniques [Die87] (see also the survey [War92]). Article [Bry90] discusses how all these ideas are related. As shown in [BR87b, Bry90, Ram88, Sek89], the magic sets and Alexander methods perform the same set of inferences as corresponding top-down methods such as query-subquery.

While the magic-sets technique was originally developed to deal with recursive queries, it is clearly applicable to nonrecursive queries as well. Indeed, it has been adapted to deal with SQL queries (which contain features such as grouping, aggregation, arithmetic conditions and multiset relations that are not present in pure logic queries), and found to be superior to techniques used in commercial database systems for *nonrecursive* "nested" SQL queries [MFPR90a].

Other variations of magic-sets include minimagic [SZ87], variants for propagating arithmetic constraints as selections [BKMR89, MFPR90b, SR92], a variant that can mimic the tail-recursion optimization of Prolog systems [Ros91], and magic templates [Ram88], in which tuples with variables in them are used to represent related facts succinctly. (Seki generalized the Alexander method similarly [Sek89].) This technique or a technique from [Ull89] are needed to guarantee that the running time of the transformed rules is no greater than that of top-down evaluation of Datalog programs. The results of [Ull89], which introduced a detailed cost model for comparing top-down and bottom-up evaluation methods, are extended to general programs in [RS91, SR93]. In [SR93], it is shown that the running time of the transformed rules (using a somewhat refined version of the magic templates algorithm) for general logic programs is no more than $O(tloglogt)$ where top-down evaluation takes time $O(t)$. (Of course, Prolog-style evaluation is likely to be faster in practice for many programs.)

## 3.2 Other Rule-Rewriting Techniques

There are a number of other approaches to optimization that sometimes yield better performance than magic-sets. These optimizations include the *counting* algorithm [BMSU86, SZ86, BR87b], the *factoring* optimization [NRSU89, KRS90], techniques for deleting redundant rules and literals [NS89, Sag88], techniques by which "existential" queries (queries for which a single answer — any answer — suffices) can be optimized [RBK88], and "envelopes" [SS88, Sag90]. A number of researchers [IW88, ZYT88, Sar89, RSUV89] have studied how to transform a program that contains nonlinear rules into an equivalent one that contains only linear rules.

### 3.3 Iterative Fixpoint Evaluation

Most rule-rewriting techniques like magic-sets expect implementation of the rewritten rules by a bottom-up technique, where starting with the facts in the database, we repeatedly evaluate the bodies of the rules with whatever facts are known (including facts for the intensional predicates) and infer what facts we can from the heads of the rules. This approach is called *naive evaluation*.

We can improve the efficiency of this algorithm by a simple "trick." If in some round of the repeated evaluation of the bodies we discover a new fact $f$, then we must have used, for at least one of the subgoals in the utilized rule, a fact that was discovered on the previous round. For if not, then $f$ itself would have been discovered in a previous round. We may thus reorganize the substitution of facts for the subgoals so that at least one of the subgoals is replaced by a fact that was discovered in the previous round. The details of this algorithm are explained in [Ull88b].

**Example 3.2** Consider the same-generation rules of Example 2.1. The first rule has a body, $flat(X, Y)$, that never changes, so after the first round, it can never yield any new *sg* facts. The second rule's body can only have new facts for the $sg(U, V)$ subgoal; the $up(X, U)$ and $down(V, Y)$ subgoals are extensional and do not change during the iteration. Thus, we can, on each round, use only the new *sg* facts from the previous round, along with the full *up* and *down* relations. Since in general, only a small fraction of the *sg* facts will be new on any one round, we significantly reduce the amount of work required. □

A number of researchers have independently proposed this evaluation technique. ([FU76, PS77, Bay85a, Ban85, BR87a]). The formulation presented in [BR87a] is probably the most widely used. It is now known widely as *seminaive evaluation*. Several refinements and variations of the basic technique have been studied, e.g., [GKB87, RSS90, Sch91, SKGB87].

The fixpoint evaluation of a logic program can also be refined by taking certain algebraic properties of the program into consideration. Such refinements, and techniques for detecting when they are applicable, have been investigated by several researchers [Hel88, IW88, Mah85, Nau88, RSUV89].

## 4 Extensions of Horn-Clause Programs

### 4.1 Negation

A deductive database query language can be enhanced by permitting negated subgoals in the bodies of rules. However, we lose an important property of our rules. When rules have the form introduced in Section 2, there is a unique *minimal model* of the rules and data. A *model* of a program is a set of facts such that for any rule, replacing body literals by facts in the model results in a head fact that is also in the model. Thus, in the context of a model, a rule can be understood as saying, essentially, "if the body is true, the head is true". A *minimal model* is a model such that no subset is a model. The existence of a unique minimal model, or least model, is clearly a fundamental and desirable property. Indeed, this least model is the one computed by naive or seminaive evaluation, as discussed in Section 3.3. Intuitively, we expect the programmer had in mind the least model when he or she wrote the logic program. However, in the presence of negated literals, a program may not have a least model.

**Example 4.1** The program:

$$p(a) :- \neg p(b).$$

has two minimal models: $\{p(a)\}$ and $\{p(b)\}$. $\square$

The meaning of a program with negation is usually given by some "intended" model ([CH85, ABW88, Prz88, PP88, GL88, Ros90, Prz90, VRS91], among others). [3] The challenge is to develop algorithms for choosing an intended model that:

1. Makes sense to the user of the rules, and
2. Allows us to answer queries about the model efficiently. In particular, it is desirable that it works well with the magic-sets transformation, in the sense that we can modify the rules by some suitable generalization of magic-sets, and the resulting rules will allow (only) the relevant portion of the selected model to be computed efficiently. (Alternatively, other efficient evaluation techniques must be developed.)

We note that relying upon such an intended model in general results in a treatment of negation that differs from classical logic. In Example 4.1 we just saw, choosing one of the two minimal models over the other cannot be justified in terms of classical logic, since the rule is logically equivalent to $p(a) \vee p(b)$. One important class of negation that has been extensively studied is stratified negation [CH85, ABW88, Van89, Naq86] A program is *stratified* if there is no recursion through negation. Programs in this class have a very intuitive semantics and can also be efficiently evaluated [BNR$^+$87, KP88a, BPRM91]. The following example describes a stratified program.

**Example 4.2** Consider the following program $P2$:

$$\begin{aligned} r1 : anc(X,Y) \quad &:- par(X,Y). \\ r2 : anc(X,Y) \quad &:- par(X,Z), anc(Z,Y). \\ r3 : nocyc(X,Y) &:- anc(X,Y), \neg anc(Y,X). \end{aligned}$$

Intuitively, this program is stratified because the definition of the predicate *nocyc* depends (negatively on) the definition of *anc*, but the definition of *anc* does not depend on the definition of *nocyc* at all.

[3] Clark's *completed program* and Reiter's *closed world assumption* approaches do not fall into this category.

A bottom-up evaluation of $P2$ would first compute a fixpoint of rules $r1$ and $r2$ (the rules defining $anc$). Rule $r3$ is applied only when the all $anc$ facts are known. □

A natural extension of stratified programs is the class of locally stratified programs [Prz88]. Intuitively, a program $P$ is locally stratified for a given database if, when we substitute constants for variables in all possible ways, the resulting instantiated rules do not have any recursion through negation. Local stratification has been extended to modular stratification in [Ros90] (see also [Bry89]). A program $P$ is said to be modularly stratified if each strongly connected component (SCC) of $P$ is locally stratified after removing instantiated rules containing literals that are false in lower SCCs.

**Example 4.3** Consider the following program:

$r1 : even(0).$
$r2 : even(s(X)) :- \neg even(X).$

This program can be seen to be locally stratified, even though the predicate $even$ depends on itself negatively. The reason is that when we substitute any value, say $x_0$, for $X$, rule $r2$ becomes

$$even(s(x_0)) :- \neg even(x_0).$$

Evidently, the use of $even$ in the body has fewer uses of the function symbol $s$ than the use in the head, so no proposition $even(s(x_0))$ can depend negatively on itself.

Consider the following variant of the above program:

$r1 : even(0).$
$r2 : even(X) :- succ(X, Y), \neg even(Y).$
$succ(1, 0). \quad succ(2, 1). \quad succ(3, 2).$

Since rule $r2$ can be instantiated with the same value for $X$ and $Y$, this program is not locally stratified. However, it is modularly stratified. The evaluation of the magic-sets transformation of this class of programs has also been considered in the literature ([Bry89, Ros90, KSS91, RSS92a]). □

The well-founded model [VRS91] is a general approach to assigning semantics to a logic program that generalizes the approaches based on stratification. The well-founded model of a program can be 3-valued, assigning the truth value "unknown" to some atoms. However, it coincides with the intended (2-valued) model for modularly stratified programs. Evaluation of well-founded programs is considered in [CW93, Mor93]. The former is a memoing variation of a top-down evaluation, and the latter adapts the magic-sets method; both rely upon the alternating fixpoint formulation [Van93]. Another approach to negation is the *inflationary fixpoint* semantics proposed in [KP88b], which we do not discuss here.

### 4.2 Set-Grouping and Aggregation

The following example illustrates the use of a *grouping* or *aggregation* construct $< >$:

$$set\_of\_grades(Class, < Grade >) :- student(Name, Class, Grade).$$

We first (conceptually) create a set of tuples for $set\_of\_grades$ using the rule

$$set\_of\_grades(Class, Grade) :- student(Name, Class, Grade).$$

Now for each value of $Class$ (in general, each value of those arguments of the head that are not enclosed in the $< >$), we create a set containing all the corresponding values for $Grade$. For each value of $Class$ let this set be called $S_{Class}$; we then create a fact $set\_of\_grades(Class, S_{Class})$.

Aggregate operations such as *count*, *sum*, *min*, and *max*, can be combined with $< >$:

$$max\_grade\_given(Class, max < Grade >) :- student(Name, Class, Grade).$$

As before, for each value of $Class$ we create a set. But now we apply the aggregate operation $max$ to the set, and create a head fact using this value rather than the set itself. [4] A number of important practical problems, such as bill-of-materials (generating various summaries of the contents of a complex part in a part-subpart hierarchy) and shortest-paths, involve a combination of aggregation and recursion.

We observe that before any head fact can be derived, all body facts that can contribute to the multiset created in the head fact must be available. This introduces a situation that is very similar to negation, and several approaches used for negation carry over to grouping. The first approach was to assume stratification of the program [BNR+87] (as was discussed for negation). Later approaches allowed weaker forms of stratification such as group stratification and magical stratification [MPR90], or modular stratification [Ros90] or extended the well-founded and stable models to deal with aggregates [KS91, BRSS92].

In general, if a rule contains grouping in the head, the multiset created by grouping must be fully determined before generating a fact using this rule. For example, if a rule contains $p(X, < Y >)$ in the head, for a given X value, the complete multiset of associated Y values must be known in order to generate a $p$ fact with this X value. In certain contexts, it is possible to generate and use $p$ facts in which the multiset of Y values is incomplete without affecting the final answer to the user's query. Monotonic programs, where a derivation using an incomplete set does not affect the final set of facts computed, were discussed in [CN89, CM90, MPR90]. Ross and Sagiv [RS92] and Van Gelder [Van92] examine broader classes of such programs. A generalization of the well-founded model semantics that deals with such programs is presented in [SSRB93].

[4] The $< >$ construct is a generalization of SQL's *group-by* construct. It is defined to generate a nested set of values in LDL, where it was originally proposed. Defining it to generate a nested *multiset* of values, as in CORAL, brings it closer to the SQL group-by construct.

Ganguly et al. [GGZ91] and Sudarshan and Ramakrishnan [SR91] examine optimizations of programs that use aggregation.

# 5 An Historical Overview of Deductive Databases

The origins of deductive databases can be traced back to work in automated theorem proving and, later, logic programming. In an interesting survey of the early development of the field [Min87], Minker suggests that Green and Raphael [GR68] were the first to recognize the connection between theorem proving and deduction in databases. They developed a series of question-answering systems that used a version of Robinson's resolution principle [Rob65], demonstrating that deduction could be carried out systematically in a database context. [5]

Other early systems included MRPPS, DEDUCE-2, and DADM. MRPPS was an interpretive system developed at Maryland by Minker's group from 1970 through 1978 that explored several search procedures, indexing techniques, and semantic query optimization. One of the first papers on processing recursive queries was [MN82]; it contained the first description of *bounded recursive* queries, which are recursive queries that can be replaced by nonrecursive equivalents. DEDUCE was implemented at IBM in the mid 1970's [Cha78], and supported left-linear recursive Horn-clause rules using a compiled approach. DADM [KT81] emphasized the distinction between EDB and IDB and studied the representation of the IDB in the form of 'connection graphs' — closely related to Sickel's interconnectivity graphs [Sic76] — to aid in the development of query plans.

A landmark workshop on logic and deductive databases was organized by Gallaire, Minker and Nicolas at Toulouse in 1977, and several papers from the proceedings appeared in book form [GM78]. Reiter's influential paper on the closed world assumption (as well as a paper on compilation of rules) appeared in this book, as did Clark's paper on negation-as-failure and a paper by Nicolas and Yazdanian on checking integrity constraints. The workshop and the book brought together researchers in the area of logic and databases, and gave an identity to the field. (The workshop was also organized in subsequent years, with proceedings, and continued to influence the field.)

In 1976, van Emden and Kowalski [vEK76] showed that the least fixpoint of a Horn-clause logic program coincided with its least Herbrand model. This provided a firm foundation for the semantics of logic programs, and especially, deductive databases, since fixpoint computation is the operational semantics associated with deductive databases (at least, of those implemented using bottom-up evaluation).

The early work focused largely on identifying suitable goals for the field, and on developing a semantic foundation. The next phase of development saw an increasing emphasis on the development of efficient query evaluation techniques. Henschen and Naqvi proposed one of the earliest efficient techniques for evaluating

[5] Cordell Green received a Grace Murray Hopper award from the ACM for his work.

recursive queries in a database context [HN84]; earlier systems had used either resolution-based strategies not well-suited to applications with large data sets, or relatively simple techniques (essentially equivalent to naive fixpoint evaluation [Cha81, SM80]). Ullman's paper on the implementation framework based on "capture rules" [Ull85] focused attention upon the challenges in efficient evaluation of recursive queries, and noted that issues such as nontermination had to be taken into account as well.

The area of deductive databases, and in particular, recursive query processing, became very active in 1984 with the initiation of three major projects, two in the U.S.A. and one in Europe. The Nail! project at Stanford, the LDL project at MCC in Austin, and the deductive database project at ECRC all led to significant research contributions and the construction of prototype systems. The ECRC and LDL projects also represented the first major deductive database projects outside of universities. Although we do not address this issue, we note that the use of this emerging technology in real-world applications is also progressing (see e.g., [Tsu91], and recent workshops at ICLP, ILPS and other conferences).

## 5.1 The Work at ECRC

The research work at ECRC was led by J.-M. Nicolas. The initial phase of research (1984–1987) led to the study of algorithms and the development of early prototypes (QSQ/SLD-AL/QoSaQ/DedGin by L. Vieille [Vie86, Vie87] ), integrity checking (Soundcheck by H. Decker) and a prototype system that explored consistency checking (Satchmo by R. Manthey and F. Bry) [BDM88], a combination of deductive and object-oriented ideas (KB2 by M. Wallace), persistent Prolog (Educe by J. Bocca), and the BANG file system by M. Freeston [Fre87]. A second phase (1988-1990) led to more functional prototypes: Megalog ( 1988-1990 by J. Bocca), DedGin* (1988-1989 by Vieille), EKS-V1 (1989-1990, also by Vieille). The EKS system supported integrity constraints [VBK91] and also some forms of aggregation through recursion [Lef92]. Bry's work on reconciling bottom-up and top-down algorithms [Bry90] and extending Magic Sets to programs with negation [Bry89], was also carried out as part of the ECRC project. More recently, ECRC has been involved in an ongoing ESPRIT project called IDEA, and is developing a temporal deductive database system in its ChronoBase project [Sri93]. It is anticipated that ChronoBase will be used for the development of a large real-world application from the airline industry, as part of a newly commenced ESPRIT project.

An interesting spinoff of the research at ECRC is a project that is currently underway at Groupe Bull to develop a commercial deductive, object-oriented database system. The deductive technology derives from the EKS prototype, but it incorporates object-oriented features both at the architectural and language levels. In addition to a data model with objects and values not unlike that of O2, it supports a data manipulation language with a Datalog-based declarative component (to write deduction rules and integrity constraints) and a more classical imperative component (to write methods and functions). It is built on a storage manager having many features in common with object-managers, rather than with more traditional relational data stores.

## 5.2 The LDL Project

The LDL project at MCC, also initiated in 1984, led to a number of important advances. By 1986, it was recognized that combining Prolog with a relational database was an unsatisfactory solution, and a decision was made to develop a deductive database system based on bottom-up evaluation techniques [TZ86]. During this period, there were a number of significant research developments including the development of evaluation algorithms (work on seminaive evaluation, Magic Sets and Counting [BMSU86, Ban85, BR91, SZ86, SZ87]), semantics for stratified negation and set-grouping [BNST91], research into the issue of safety, or finiteness of answer sets, compilation of set-terms, generation of explanations of logic program evaluation, the treatment of updates in logical rules, and join order optimization.

An initial prototype called EVE was developed in Prolog, producing target code in an extended relational language. The LDL prototype (also called SALAD), which was developed by 1988, and released with refinements from 1989 through 1991, was the first deductive database system that was widely available. It supported stratified negation and set-terms, and was a compiled system that produced C code [CGK+90]. It has been distributed to universities and to shareholder companies in the MCC consortium. A good presentation of the LDL language is in [NT89]. Subsequent research has focused on aggregate operations [GGZ91] and nondeterministic choice constructs (e.g., [SZ90]). The LDL++ project at MCC is a direct successor to LDL, and was initiated in 1990. In addition to adopting a more interpretive style of evaluation, nonstratified negation and aggregation features have been explored [ZAO93], along with support for abstract data types.

## 5.3 The NAIL! Project

The NAIL! (Not *Another* Implementation of Logic!) project was started at Stanford in 1985. Following the plan laid out in [Ull85], the initial goal was to study the optimization of logic using the database-oriented "all-solutions" model. In collaboration with the MCC group, the first paper on Magic Sets [BMSU86] came out of this project, as did the first work on regular recursions [Nau87]. The work on regular recursions was developed further in [NRSU89]. Many of the important contributions to coping with negation and aggregation in logical rules were also made by the project. Stratified negation ([Van86]), well-founded negation [VRS91], and modularly stratified negation [Ros90] were also developed in connection with this project.

An initial prototype system [MNS+87] was built but later abandoned because the purely declarative paradigm was found to be unworkable for many applications. The revised system uses a core language, called Glue, which is essentially single logical rules, with the power of SQL statements, wrapped in conventional language constructs such as loops, procedures, and modules. The original NAIL language becomes in effect a view mechanism for Glue; it allows fully declarative specifications in situations where declarativeness is appropriate [PDR91, DMP93].

## 5.4 Other Deductive Database Projects

The Aditi project was initiated in 1988 at the University of Melbourne. The research contributions of this project include a formulation of seminaive evaluation that is now widely used [BR87a], adaptation of Magic Sets for stratified programs [BPRM91], optimization of right- and left-linear programs [KRS90], parallelization, indexing techniques, and optimization of programs with constraints [BKMR89]. The work of the Aditi group was also driven by the development of their prototype system, which is notable for its emphasis on disk-resident relations. All relational operations are performed with relations assumed to be disk resident, and join techniques such as sort-merge and hash-join are used. An overview is provided in [VRK+91]. [6]

The ConceptBase system, developed since 1987 at the Universities of Passau and Aachen in Jarke's group, seeks to combine deductive rules with a semantic data model based on Telos. The language aspects are presented in [JS93]. The system also provides support for integrity constraints; this is described in [JJ91]. ConceptBase has been used in a number of applications at various universities in Europe, and is now being commercially developed.

The CORAL project at U. Wisconsin, which was started in 1988, can also be traced to LDL. [7] The original impetus for the project was the development of the Magic Templates algorithm [Ram88], which offered the potential to support nonground tuples. The research contributions included work on optimizing special classes of programs (notably, right- and left- linear programs) [NRSU89] (jointly with the Glue-Nail group), development of a multiset semantics for logic programs and optimizations dealing with duplicate checks [MR89], the first results on space-efficient bottom-up evaluation techiques [NR90, SSRN91], refinements of seminaive evaluation for programs with large numbers of rules [RSS90], evaluation of programs with aggregate operations [SR91], arithmetic constraints [SR92], modular-stratified negation [RSS92a], and nonground tuples [SR93]. The result that bottom-up evaluation asymptotically dominates top-down evaluation for all Horn-clause programs was obtained as part of this project [SR93]. A first prototype of the CORAL system was functional in 1990. The first widely available release of the system was in 1993. This version supported nonstratified aggregation and negation using an algorithm proposed in [RSS92a], provided high-level features for controlling evaluation, and provided support for nonground tuples. [8] A language overview is provided in [RSS92b], and the implementation is described in [RSSS93]. The extension to support object-oriented features, called Coral++, is described in [SRSS93].

The DECLARE project at MAD Intelligent Systems, which ran from 1986 to 1990 and was led by W. Kiessling, was perhaps the first attempt to commercialize deductive databases. Given the focus on commercialization, it is

[6] LDL and LDL++ are memory resident systems, and CORAL supports disk-resident relations by building upon the EXODUS storage manager, without providing additional join algorithms tailored to disk-resident relations.

[7] Ramakrishnan, who initiated the CORAL project, was involved in the LDL project until he moved to Wisconsin.

[8] It has been distributed to over 200 academic and research sites, and has been used for teaching as well as research. It is freely available by ftp from ftp.cs.wisc.edu, in C++ source code.

understandable that the research was not widely publicized, but the implemented system was quite sophisticated. It supported stratified negation and sets, and was implemented using Magic Sets and seminaive evaluation. Many variations of seminaive evaluation were implemented and evaluated as part of this work [Kie92]. It also provided conventional database facilities such as crash recovery and concurrent transactions.

The LOGRES project at Politecnico di Milano ran from 1989 to 1992. The prototype is built on top of an extended relational system (ALGRES), and is notable for its integration of an object-oriented data model with deductive rules. It is one of the first of the new family of 'DOOD' (deductive, object-oriented database) systems. While the LOGRES project did not produce a practical prototype, the research has influenced the ongoing IDEA project, which aims to develop efficient prototype systems.

The LOLA project at TU Muenchen, led by R. Bayer and U. Guentzer, ran from 1988 to 1992, and led to the development of a prototype system that is used in a number of related projects. This is one of the more complete prototypes, and supports stratified negation and aggregation, disk-resident base relations, interfaces to commercial databases, integrity constraint maintenance and an explanation facility. It is available freely, and is being further developed. A notable research contribution of the LOLA group is the development of interesting refinements to seminaive evaluation. (Bayer, one of the leaders of the LOLA project, was also one of the initial proponents of seminaive evaluation [Bay85b].)

The Starburst project at IBM Almaden was primarily involved with extensibility, but made important contributions to the development of Magic Sets for programs with SQL features like grouping, aggregation, and arithmetic selections [MPR90]. A performance study demonstrated that the techniques developed for dealing with recursive queries actually beat the techniques used in current relational database systems [MFPR90a]. They have also extended SQL with recursive view definitions, providing greater expressive power to SQL users, and utilizing the I/O optimization facilities of SQL implementations in a direct manner for recursive applications. They have also made contributions to the evaluation of left- and right-linear programs.

Another substantial implementation of a deductive database, using an evaluation strategy similar to QSQ, was carried out at Zurich during 1986 to 1990. This system, called IISProlog, supported linear recursive programs, but was unfortunately never widely known. While the research embodied in it is difficult to evaluate, due to the limited literature, it appears to have been a significant early effort [Nus92].

Hy+ is an ongoing project at the University of Toronto [CM93] that was initiated in 1987 by A. Mendelzon, and is a successor to G+/GraphLog [CM93]. The focus of the project is the class of path queries, and the goal is to develop high level visual interfaces and query languages for this domain. While the project has not been focused on deductive databases in general, systems such as LDL and CORAL have been used as the underlying inference engine, and interesting results in the area of path queries have been developed.

An interesting ongoing effort is the XSB project at StonyBrook, led by D. S. Warren. They are developing a system that supports modularly stratified negation and aggregation (plus a meta-interpreter for well-founded programs), nonground tuples, and disk-resident relations. The implementation is based on OLDT resolution [TS86], a top-down evaluation with memoing, and is particularly well suited for integration with Prolog systems. (This is essentially the Extension Tables technique described in [Die87].) In fact, the WAM, a widely used abstract machine for implementing Prolog systems, has been adapted to support the memoing top-down evaluation used in XSB [War89]. (This is a tuple-at-a-time approach.) Future extensions supporting constraints in tuples are planned.

We note that there are other projects, such as the RDL effort at INRIA, that have not focused upon deductive databases, but are closely related [KdMS90].

# 6 Deductive Database System Implementations

There have been a number of implementations of deductive databases. The results of a survey conducted by the authors over the Internet, presented below, indicate that the extensive published research on deductive databases was accompanied by quite a large number of prototyping efforts. [9]

## 6.1 Summary of Deductive Database Prototypes

Figures 1 and 2 summarize some of the important features of current deductive systems. In Fig. 1 we compare the way these systems handle the following issues:

1. *Recursion.* Most systems allow the rules to use general recursion. However, a few limit recursion to linear recursion or to restricted forms related to graph searching, such as transitive closure.
2. *Negation.* Most systems allow negated subgoals in rules. When rules involve negation, there are normally many minimal fixpoints that could be interpreted as the meaning of the rules, and the system has to select from among these possibilities one model that is regarded as *the* intended model, against which queries will be answered. Section 4.1 discusses the principal approaches.
3. *Aggregation.* A problem similar to negation comes up when aggregation (sum, average, etc.) is allowed in rules. More than one minimal model normally exists, and the system must select the appropriate model. See Section 4.2.

The following issues are summarized in Fig. 2.

[9] We thank the respondents of the survey for their detailed comments. The actual responses provide additional information about the projects, and are available by ftp from ricotta.cs.wisc.edu. Since many of these systems are not currently distributed, we have not verified the information presented below — our summary is based upon information provided by the implementors.

| Name | Developed | Refs. | Recursion | Negation | Aggregation |
|---|---|---|---|---|---|
| Aditi | U. Melbourne | [VRK+91] | General | Stratified | Stratified |
| COL | INRIA | [AG91] | ? | Stratified | Stratified |
| Concept-Base | U. Aachen | [JS93] | General | Locally Stratified | No |
| CORAL | U. Wisconsin | [RSS92b] | General | Modularly Stratified | Modularly Stratified |
| EKS-VI | ECRC | [BLV93] | General | Stratified | Superset of Stratified |
| LogicBase | Simon Fraser U. | | Linear, some nonlinear | Stratified | No |
| DECLARE | MAD Intelligent Systems | [KS93] | General | Locally Stratified | Superset of Stratified |
| Hy+ | U. Toronto | [CM93] | Path Queries | Stratified | Stratified |
| X4 | U. Karlsruhe | [ML91] | General, but only binary preds. | No | No |
| LDL<br>LDL++ | MCC | [CGK+90] | General | Stratified<br>Restricted Local | Stratified<br>Restricted Local |
| LOGRES | Polytechnic of Milan | [CCCR+90] | Linear | Inflationary Semantics | Stratified |
| LOLA | Technical U. Munich | [FSS91] | General | Stratified | Computed Predicates |
| Glue-Nail | Stanford U. | [DMP93] | General | Well-Founded | Glue only |
| Starburst | IBM Almaden | [MFPR90a] | General | Stratified | Stratified |
| XSB | SUNY Stony Brook | | General | Well-Founded | Modularly Stratified |

Figure 1: Summary of Prototypes, Part I

| Name | Updates | Constraints | Optimizations | Storage | Interfaces |
|---|---|---|---|---|---|
| Aditi | No | No | Magic sets, SN<br>Join-order selection | EDB, IDB | Prolog |
| COL<br>ConceptBase | No<br>Yes | No<br>Yes | None<br>Magic sets, SN | Main memory<br>EDB only | ML<br>C,Prolog |
| CORAL | Yes | No | Magic sets, SN<br>Context Factoring<br>Projection pushing | EDB, IDB | C, C++,<br>Extensible |
| EKS-VI | Yes | Yes | Query-subquery,<br>left/right linear | EDB, IDB | Persistent<br>Prolog |
| LogicBase | No | No | "Chain-based<br>evaluation" | EDB, IDB | C, C++,<br>SQL |
| DECLARE | No | No | Magic sets, SN<br>Projection pushing | EDB only | C,<br>Lisp |
| Hy+ | No | No | | Main<br>memory | Prolog, LDL,<br>CORAL, Smalltalk |
| LDL, LDL++ | Yes | No | Magic sets, SN<br>Left/right linear,<br>Projection pushing,<br>"Bushy depth-first" | EDB only | C, C++,<br>SQL |
| LOGRES | Yes | Yes | Algebraic, SN | EDB, IDB | INFORMIX |
| LOLA | No | Yes | Magic Sets, SN<br>Projection pushing<br>Join-order selection | EDB | TransBase (SQL) |
| X4 | No | Yes | None,<br>Top-down eval. | EDB only | Lisp |
| Glue-Nail | Glue only | No | Magic sets, SN<br>Right-linear<br>Join-order selection | EDB only | None |
| Starburst | No | No | Magic sets, SN variant | EDB, IDB | Extensible |
| XSB | No | No | Memoing, top-down | EDB, IDB | C, Prolog |

Figure 2: Summary of Prototypes, Part II

1. *Updates.* Logical rules do not, in principle, involve updating of the database. However, most systems have some approach to specifying updates, either through special dictions in the rules or update facilities outside the rule system. We have identified systems that support updates in logical rules by a "Yes" in the table. (Some limitations as to the order of evaluation are usually enforced with respect to rules containing updates.)

2. *Integrity Constraints.* Some deductive systems allow logical rules that serve as integrity constraints. That is, rather than defining IDB predicates, constraint rules express conditions that cannot be violated by the data.

3. *Optimizations.* Deductive systems need to provide some optimization of queries. Common techniques include Magic-Sets or similar techniques for combining the benefits of both top-down and bottom-up processing, and seminaive Evaluation for avoiding some redundant processing. A variety of other techniques are used by various systems, and we attempt to summarize the principal techniques here. Quotations around a method indicates that the method has not been defined in this survey, and the reader should look at the source paper.

4. *Storage.* Most systems allow EDB relations to be stored on disk, but some also store IDB relations in secondary storage. Supporting disk-resident data efficiently is a significant task.

5. *Interfaces.* Most systems connect to one or more other languages or systems. Some of these connections are embeddings of calls to the deductive system in another language, while other connections allow other languages or systems to be invoked from the deductive system. We have not, however, distinguished the direction of the call in this brief summary. Some systems use external language interfaces to provide ways in which the system can be customized for different applications (e.g., by adding new data types, relation implementations etc.). We refer to this capability as extensibility; it is very useful for large applications.

# 7 Conclusion

We have reviewed several results in the field of deductive databases, with an emphasis on efficient evaluation techniques, and presented a summary of several projects that led to implemented systems. The main points to note are the following.

1. There exist efficient evaluation methods that are sound and complete with respect to an intuitive declarative semantics for large classes of programs with powerful features like negation and aggregation.

2. Systems based upon these methods are being developed, and offer good support for rule-based applications.

There is also ongoing work that seeks to combine the powerful query language capability of deductive databases with features from object-oriented systems, and this will likely lead to a new generation of more powerful systems that bring database languages and programming languages closer to each other.

## 8 Acknowledgements

The work of R. Ramakrishnan was supported by a David and Lucile Packard Foundation Fellowship in Science and Engineering, a Presidential Young Investigator Award, with matching grants from Digital Equipment Corporation, Tandem and Xerox, and NSF grant IRI-9011563.

The work of J. D. Ullman was supported by ARO grant DAAL03–91–G–0177, NSF grant IRI–90–16358, and a grant of Mitsubishi Electric.

## References

[ABW88] K. R. Apt, H. Blair, and A. Walker. Towards a theory of declarative knowledge. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 89–148. Morgan-Kaufmann, San Mateo, Calif., 1988.

[AG91] Serge Abiteboul and Stephane Grumbach. A rule-based language with functions and sets. *ACM Transactions on Database Systems*, 16(1):1–30, 1991.

[Ban85] Francois Bancilhon. Naive evaluation of recursively defined relations. In Brodie and Mylopoulos, editors, *On Knowledge Base Management Systems — Integrating Database and AI Systems*. Springer-Verlag, 1985.

[Bay85a] R. Bayer. Query evaluation and recursion in deductive database systems. Unpublished Memorandum, 1985.

[Bay85b] R. Bayer. Query evaluation and recursion in deductive database systems. Technical Report 18503, Technische Universitaet Muenchen, February 1985.

[BDM88] F. Bry, H. Decker, and R. Manthey. A uniform approach to constraint satisfaction and constraint satisfiability in deductive databases. In *Proceedings of the International Conference on Extending Database Technology*, February 1988.

[BKMR89] Isaac Balbin, David B. Kemp, Krishnamurthy Meenakshi, and Kotagiri Ramamohanarao. Propagating constraints in recursive deductive databases. In *Proceedings of the North American Conference on Logic Programming*, pages 16–20, October 1989.

[BLV93] P. Bayer, A. Lefebvre, and L. Vieille. Architecture and Design of the EKS Deductive Database System. Technical report, ECRC, March 1993. Submitted to the VLDB Journal.

[BMSU86] Francois Bancilhon, David Maier, Yehoshua Sagiv, and Jeffrey D. Ullman. Magic sets and other strange ways to implement logic programs. In *Proceedings of the ACM Symposium on Principles of Database Systems*, pages 1–15, Cambridge, Massachusetts, March 1986.

[BNR+87] Catriel Beeri, Shamim Naqvi, Raghu Ramakrishnan, Oded Shmueli, and Shalom Tsur. Sets and negation in a logic database language. In *Proceedings of the ACM Symposium on Principles of Database Systems*, pages 21–37, San Diego, California, March 1987.

[BNST91] Catriel Beeri, Shamim Naqvi, Oded Shmueli, and Shalom Tsur. Set constructors in a logic database language. *Journal of Logic Programming*, 10(3&4):181–232, 1991.

[BPRM91] I. Balbin, G. S. Port, K. Ramamohanarao, and K. Meenakshi. Efficient bottom-up computation of queries on stratified databases. Journal of Logic Programming, 11:295–345, 1991.

[BR87a] I. Balbin and K. Ramamohanarao. A generalization of the differential approach to recursive query evaluation. *Journal of Logic Programming*, 4(3), September 1987.

[BR87b] Catriel Beeri and Raghu Ramakrishnan. On the power of Magic. In *Proceedings of the ACM Symposium on Principles of Database Systems*, pages 269–283, San Diego, California, March 1987.

[BR91] Catriel Beeri and Raghu Ramakrishnan. On the power of Magic. *Journal of Logic Programming*, 10(3&4):255–300, 1991.

[BRSS92] C. Beeri, R. Ramakrishnan, D. Srivastava, and S. Sudarshan. The valid model semantics for logic programs. In *Proceedings of the ACM Symposium on Principles of Database Systems*, pages 91–104, June 1992.

[Bry89] Francois Bry. Logic programming as constructivism: A formalization and its application to databases. In *Proceedings of the ACM SIGACT-SIGART-SIGMOD Symposium on Principles of Database Systems*, pages 34–50, Philadelphia, Pennsylvania, March 1989.

[Bry90] Francois Bry. Query evaluation in recursive databases: Bottom-up and top-down reconciled. *IEEE Transactions on Knowledge and Data Engineering*, 5:289–312, 1990.

[CCCR+90] F. Cacace, S. Ceri, S. Crespi-Reghizzi, L. Tanca, and R. Zicari. Integrating object-oriented data modeling with a rule-based programming paradigm. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, May 1990.

[CGK+90] D. Chimenti, R. Gamboa, R. Krishnamurthy, S. Naqvi, S. Tsur, and C. Zaniolo. The LDL system prototype. *IEEE Transactions on Knowledge and Data Engineering*, 2(1):76–90, 1990.

[CH85] Ashok K. Chandra and David Harel. Horn clause queries and generalizations. *J. Logic Programming*, 2(1):1–15, April 1985.

[Cha78] C.L. Chang. Deduce 2: Further investigations of deduction in relational databases. In H. Gallaire and J. Minker, editors, *Logic and Databases*. Plenum Press, 1978.

[Cha81] C.L. Chang. On the evaluation of queries containing derived relations in a relational data base. In H. Gallaire, J. Minker, and J. Nicolas, editors, *Advances in Data Base Theory, Volume 1*. Plenum Press, 1981.

[CM90] M. P. Consens and A. O. Mendelzon. Low complexity aggregation in Graphlog and Datalog. In *Proceedings of the International Conference on Database Theory*, Paris, 1990.

[CM93] Mariano Consens and Alberto Mendelzon. Hy: A hygraph-based query and visualization system. In *Proceedings of the ACM-SIGMOD 1993 Annual Conference on Management of Data*, pages 511–516, 1993.

[CN89] I. F. Cruz and T. S. Norvell. Aggregative closure: An extension of transitive closure. In *Proc. IEEE 5th Int'l Conf. Data Engineering*, pages 384–389, 1989.

[CW93] Weidong Chen and Davis S. Warren. Query evaluation under the well-founded semantics. In *Proceedings of the ACM Symposium on Principles of Database Systems*, 1993.

[Die87] Suzanne W. Dietrich. Extension tables: Memo relations in logic programming. In *Proceedings of the Symposium on Logic Programming*, pages 264–272, 1987.

[DMP93] M. Derr, S. Morishita, and G. Phipps. Design and implementation of the glue-nail database system. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 147–167, Washington, D.C., 1993.

[Fre87] M. Freeston. The bang file: A new kind of grid file. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, 1987.

[FSS91] B. Freitag, H. Schűtz, and G. Specht. LOLA - a logic language for deductive databases and its implementation. In *Proceedings of 2nd International Symposium on Database Systems for Advanced Applications (DASFAA)*, 1991.

[FU76] A. C. Fong and J.D. Ullman. Induction variables in very high-level languages. In *Proc. Third ACM Symposium on Principles of Programming Languages*, pages 104–112, 1976.

[GGZ91] Sumit Ganguly, Sergio Greco, and Carlo Zaniolo. Minimum and maximum predicates in logic programming. In *Proceedings of the ACM Symposium on Principles of Database Systems*, 1991.

[GKB87] U. Güntzer, W. Kiessling, and R. Bayer. On the evaluation of recursion in (deductive) database systems by efficient differential fixpoint iteration. In *International Conference on Data Engineering*, pages 120–129, 1987.

[GL88] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Proc. Fifth International Conference and Symposium on Logic Programming*, 1988.

[GM78] H. Gallaire and J. Minker, editors. *Logic and Databases*. Plenum Press, 1978.

[GR68] Cordell C. Green and Bertram Raphael. The use of theorem-proving techniques in question-answering systems. In *Proceedings of the 23rd ACM National Conference*, Washington, D.C., 1968.

[Hel88] A. Richard Helm. Detecting and eliminating redundant derivations in deductive database systems. Technical Report RC 14244 (#63767), IBM Thomas Watson Research Center, December 1988.

[HN84] Lawrence J. Henschen and Shamim A. Naqvi. On compiling queries in recursive first order databases. *Journal of the ACM*, 31(1):47–85, 1984.

[IW88] Yannis E. Ioannidis and Eugene Wong. Towards an algebraic theory of recursion. Technical Report 801, Computer Sciences Department, University of Wisconsin-Madison, October 1988.

[JJ91] M. Jeusfeld and M. Jarke. From relational to object-oriented integrity simplification. In M. Kifer C. Delobel and Y. Masunaga, editors, *Proc. Deductive and Object-Oriented Databases 91*. Springer-Verlag, 1991.

[JS93] M. Jeusfeld and M. Staudt. Query optimization in deductive object bases. In G. Vossen J.C. Freytag and D. Maier, editors, *Query Processing for Advanced Database Applications*. Morgan-Kaufmann, 1993.

[KdMS90] G. Kiernan, C. de Maindreville, and E. Simon. Making deductive database a practical technology: a step forward. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, 1990.

[Kie92] W. Kiessling. A complex benchmark for logic programming and deductive databases, or who can beat the n-queens? *SIGMOD Record*, 21(4):28–34, December 1992.

[KL86] Michael Kifer and Eliezer L. Lozinskii. A framework for an efficient implementation of deductive databases. In *Proceedings of the Advanced Database Symposium*, Tokyo, Japan, 1986.

[KP88a] J.M. Kerisit and J.M. Pugin. Efficient query answering on stratified databases. In *Proc. of the International Conference on Fifth Generation Computer Systems*, pages 719–725, Tokyo, Japan, November 1988.

[KP88b] P. Kolaitis and C. Papadimitriou. Why not negation by fixpoint? In *Proceedings of the ACM Symposium on Principles of Database Systems*, pages 231–239, 1988.

[KRS90] D. Kemp, K. Ramamohanarao, and Z. Somogyi. Right-, left-, and multi-linear rule transformations that maintain context information. In *Proceedings of the International Conference on Very Large Databases*, pages 380–391, Brisbane, Australia, 1990.

[KS91] David Kemp and Peter Stuckey. Semantics of logic programs with aggregates. In *Proceedings of the International Logic Programming Symposium*, pages 387–401, San Diego, CA, U.S.A., October 1991.

[KS93] Werner Kießling and H. Schmidt. DECLARE and SDS: Early efforts to commercialize deductive database technology. Submitted, 1993.

[KSS91] David Kemp, Divesh Srivastava, and Peter Stuckey. Magic sets and bottom-up evaluation of well-founded models. In *Proceedings of the International Logic Programming Symposium*, pages 337–351, San Diego, CA, U.S.A., October 1991.

[KT81] Charles Kellogg and Larry Travis. Reasoning with data in a deductively augmented data management system. In H. Gallaire, J. Minker, and J. Nicolas, editors, *Advances in Data Base Theory, Volume 1*. Plenum Press, 1981.

[Lef92] Alexandre Lefebvre. Towards an efficient evaluation of recursive aggregates in deductive databases. In *Proceedings of the International Conference on Fifth Generation Computer Systems*, June 1992.

[Mah85] Michael J. Maher. *Semantics of Logic Programs*. PhD thesis, Department of Computer Science, University of Melbourne, Melbourne, Australia, 1985.

[MFPR90a] I. S. Mumick, S. Finkelstein, H. Pirahesh, and R. Ramakrishnan. Magic is relevant. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Atlantic City, New Jersey, May 1990.

[MFPR90b] Inderpal Singh Mumick, Sheldon J. Finkelstein, Hamid Pirahesh, and Raghu Ramakrishnan. Magic conditions. In *Proceedings of the Ninth ACM Symposium on Principles of Database Systems*, pages 314–330, Nashville, Tennessee, April 1990.

[Min87] J. Minker. Perspectives in deductive databases. Technical Report CS-TR-1799, University of Maryland at College Park, March 1987.

[ML91] G. Moerkotte and P.C. Lockemann. Reactive consistency control in deductive databases. *ACM Trans. on Database Systems*, 16(4):670–702, 1991.

[MN82] J. Minker and J. M. Nicolas. On recursive axioms in deductive databases. *Information Systems*, 8(1), 1982.

[MNS+87] Katherine Morris, Jeffrey F. Naughton, Yatin Saraiya, Jeffrey D. Ullman, and Allen Van Gelder. YAWN! (Yet Another Window on NAIL!). *Database Engineering*, December 1987.

[Mor93] Shinichi Morishita. An alternating fixpoint tailored to magic programs. In *Proceedings of the ACM Symposium on Principles of Database Systems*, 1993.

[MPR90] Inderpal S. Mumick, Hamid Pirahesh, and Raghu Ramakrishnan. Duplicates and aggregates in deductive databases. In *Proceedings of the Sixteenth International Conference on Very Large Databases*, August 1990.

[MR89] Michael J. Maher and Raghu Ramakrishnan. Dèjá vu in fixpoints of logic programs. In *Proceedings of the Symposium on Logic Programming*, Cleveland, Ohio, 1989.

[Naq86] Shamim Naqvi. Negation in knowledge base management systems. pages 125–146. 1986.

[Nau87] Jeffrey F. Naughton. One sided recursions. In *Proceedings of the ACM Symposium on Principles of Database Systems*, pages 340–348, San Diego, California, March 1987.

[Nau88] Jeffrey F. Naughton. Compiling separable recursions. In *Proceedings of the SIGMOD International Symposium on Management of Data*, pages 312–319, Chicago, Illinois, May 1988.

[NR90] Jeffrey F. Naughton and Raghu Ramakrishnan. How to forget the past without repeating it. In *Proceedings of the Sixteenth International Conference on Very Large Databases*, August 1990.

[NRSU89] Jeffrey F. Naughton, Raghu Ramakrishnan, Yehoshua Sagiv, and Jeffrey D. Ullman. Argument reduction through factoring. In *Proceedings of the Fifteenth International Conference on Very Large Databases*, pages 173–182, Amsterdam, The Netherlands, August 1989.

[NS89] Jeffrey F. Naughton and Yehoshua Sagiv. Minimizing expansions of recursions. In Hasan Ait-Kaci and Maurice Nivat, editors, *Resolution of Equations in Algebraic Structures*, volume 1, pages 321–349, San Diego, California, 1989. Academic Press, Inc.

[NT89] Shamim Naqvi and Shalom Tsur. *A Logical Language for Data and Knowledge Bases.* Principles of Computer Science. Computer Science Press, New York, 1989.

[Nus92] Miguel Nussbaum. *Building a Deductive Database.* Ablex Publishing Corporation, 1992.

[PDR91] Geoffrey Phipps, Marcia A. Derr, and Kenneth A. Ross. Glue-NAIL!: A deductive database system. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 308–317, 1991.

[PP88] H. Przymusinska and T.C. Przymusinski. Weakly perfect model semantics for logic programs. In *Proceedings of the Fifth International Conference/Symposium on Logic Programming*, 1988.

[Prz88] T.C. Przymusinski. On the declarative semantics of stratified deductive databases. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 193–216, 1988.

[Prz90] T.C. Przymusinski. Extended stable semantics for normal and disjunctive programs. In *Seventh International Conference on Logic Programming*, pages 459–477, 1990.

[PS77] R. Paige and J. T. Schwatz. Reduction in strength of high level operations. In *Proc. Fourth ACM Symposium on Principles of Programming Languages*, pages 58–71, 1977.

[PW83] F.C.N. Pereira and D.H.D. Warren. Parsing as deduction. In *Proceedings of the twenty-first Annual Meeting of the Association for Computational Linguistics*, 1983.

[Ram88] Raghu Ramakrishnan. Magic templates: A spellbinding approach to logic programs. In *Proceedings of the International Conference on Logic Programming*, pages 140–159, Seattle, Washington, August 1988.

[RBK88] Raghu Ramakrishnan, Catriel Beeri, and Ravi Krishnamurthy. Optimizing existential Datalog queries. In *Proceedings of the ACM Symposium on Principles of Database Systems*, pages 89–102, Austin, Texas, March 1988.

[RLK86] J. Rohmer, R. Lescoeur, and J. M. Kerisit. The Alexander method — a technique for the processing of recursive axioms in deductive database queries. *New Generation Computing*, 4:522–528, 1986.

[Rob65] J. A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12:23–41, 1965.

[Ros90] Kenneth Ross. Modular Stratification and Magic Sets for DATALOG programs with negation. In *Proceedings of the ACM Symposium on Principles of Database Systems*, pages 161–171, 1990.

[Ros91] Kenneth Ross. Modular acyclicity and tail recursion in logic programs. In *Proceedings of the ACM Symposium on Principles of Database Systems*, 1991.

[RS91] Raghu Ramakrishnan and S. Sudarshan. Top-Down vs. Bottom-Up Revisited. In *Proceedings of the International Logic Programming Symposium*, 1991.

[RS92] Kenneth Ross and Yehoshua Sagiv. Monotonic aggregation in deductive databases. In *Proceedings of the ACM Symposium on Principles of Database Systems*, pages 114–126, 1992.

[RSS90] Raghu Ramakrishnan, Divesh Srivastava, and S. Sudarshan. Rule ordering in bottom-up fixpoint evaluation of logic programs. In *Proceedings of the Sixteenth International Conference on Very Large Databases*, August 1990.

[RSS92a] Raghu Ramakrishnan, Divesh Srivastava, and S. Sudarshan. Controlling the search in bottom-up evaluation. In *Proceedings of the Joint International Conference and Symposium on Logic Programming*, 1992.

[RSS92b] Raghu Ramakrishnan, Divesh Srivastava, and S. Sudarshan. CORAL: Control, Relations and Logic. In *Proceedings of the International Conference on Very Large Databases*, 1992.

[RSSS93] Raghu Ramakrishnan, Divesh Srivastava, S. Sudarshan, and Praveen Seshadri. Implementation of the CORAL deductive database system. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, 1993.

[RSUV89] Raghu Ramakrishnan, Yehoshua Sagiv, Jeffrey D. Ullman, and Moshe Vardi. Proof-tree transformation theorems and their applications. In *Proceedings of the ACM Symposium on Principles of Database Systems*, Philadelphia, Pennsylvania, March 1989.

[Sag88] Yehoshua Sagiv. Optimizing Datalog programs. In Jack Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 659–698, Los Altos, California, 94022, 1988. Morgan Kaufmann.

[Sag90] Y. Sagiv. Is there anything better than magic? In *Proceedings of the North American Conference on Logic Programming*, pages 235–254, Austin, Texas, 1990.

[Sar89] Yatin Saraiya. Linearizing nonlinear recursions in polynomial time. In *Proceedings of the ACM SIGACT-SIGART-SIGMOD Symposium on Principles of Database Systems*, pages 182–189, Philadelphia, Pennsylvania, March 1989.

[Sch91] Helmut Schmidt. *Meta-Level Control for Deductive Database Systems*. Lecture Notes in Computer Science, Number 479. Springer-Verlag, 1991.

[Sek89] H. Seki. On the power of Alexander templates. In *Proc. of the ACM Symposium on Principles of Database Systems*, pages 150–159, 1989.

[Sic76] S. Sickel. A search technique for clause interconnectivity graphs. *IEEE Transactions on Computers*, 8(C-25):823–835, 1976.

[SKGB87] H. Schmidt, W. Kiessling, U. Güntzer, and R. Bayer. Compiling exploratory and goal-directed deduction into sloppy delta iteration. In *IEEE International Symposium on Logic Programming*, pages 234–243, 1987.

[SM80] S. E. Shapiro and D. P. McKay. Inference with recursive rules. In *Proceedings of the 1st Annual National Conference on Artificial Intelligence*, 1980.

[SR91] S. Sudarshan and Raghu Ramakrishnan. Aggregation and relevance in deductive databases. In *Proceedings of the Seventeenth International Conference on Very Large Databases*, September 1991.

[SR92] Divesh Srivastava and Raghu Ramakrishnan. Pushing constraint selections. In *Proceedings of the Eleventh ACM Symposium on Principles of Database Systems*, San Diego, CA, June 1992.

[SR93] S. Sudarshan and Raghu Ramakrishnan. Optimizations of bottom-up evaluation with non-ground terms. In *Proceedings of the International Logic Programming Symposium*, 1993.

[Sri93] S.M. Sripada. Design of the chronobase temporal deductive database system. In *Proc. International Workshop on the Infrastructure for Temporal Databases*, 1993.

[SRSS93] Divesh Srivastava, Raghu Ramakrishnan, S. Sudarshan, and Praveen Seshadri. Coral++: Adding object-orientation to a logic database language. In *Proceedings of the International Conference on Very Large Databases*, 1993.

[SS88] Seppo Sippu and Eljas Soisalon-Soinen. An optimization strategy for recursive queries in logic databases. In *Proceedings of the Fourth International Conference on Data Engineering*, Los Angeles, California, 1988.

[SSRB93] S. Sudarshan, D. Srivastava, R. Ramakrishnan, and C. Beeri. Extending the well-founded and valid model semantics for aggregation. In *Proceedings of the International Logic Programming Symposium*, 1993.

[SSRN91] S. Sudarshan, Divesh Srivastava, Raghu Ramakrishnan, and Jeff Naughton. Space optimization in the bottom-up evaluation of logic programs. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, 1991.

[SZ86] Domenico Sacca and Carlo Zaniolo. The generalized counting methods for recursive logic queries. In *Proceedings of the First International Conference on Database Theory*, 1986.

[SZ87] Domenico Sacca and Carlo Zaniolo. Magic counting methods. In *Proceedings of the ACM-SIGMOD Symposium on the Management of Data*, pages 49–59, San Fransisco, California, June 1987.

[SZ90] Domenico Sacca and Carlo Zaniolo. Stable models and non-determinism in logic programs with negation. In *Proceedings of the ACM Symposium on Principles of Database Systems*, pages 205–217, 1990.

[TS86] H. Tamaki and T. Sato. OLD resolution with tabulation. In *Proceedings of the Third International Conference on Logic Programming*, pages 84–98, 1986. (Lecture Notes in Computer Science 225, Springer-Verlag).

[Tsu91] S. Tsur. Deductive databases in action. In *Proceedings of the Tenth ACM Symposium on Principles of Database Systems*, pages 142–153, 1991.

[TZ86] Shalom Tsur and Carlo Zaniolo. LDL: A logic-based data-language. In *Proceedings of the Twelfth International Conference on Very Large Data Bases*, pages 33–41, Kyoto, Japan, August 1986.

[Ull85] Jeffrey D. Ullman. Implementation of logical query languages for databases. *ACM Transactions on Database Systems*, 10(4):289–321, September 1985.

[Ull88a] Jeffrey D. Ullman. *Principles of Database and Knowledge-Base Systems*, volume 2. Computer Science Press, 1988.

[Ull88b] Jeffrey D. Ullman. *Principles of Database and Knowledge-Base Systems*, volume 1. Computer Science Press, 1988.

[Ull89] Jeffrey D. Ullman. Bottom-up beats top-down for Datalog. In *Proceedings of the Eighth ACM Symposium on Principles of Database Systems*, pages 140–149, Philadelphia, Pennsylvania, March 1989.

[Van86] A. Van Gelder. Negation as failure using tight derivations for general logic programs. In *Proceedings of the Symposium on Logic Programming*, pages 127–139, 1986.

[Van89] A. Van Gelder. Negation as failure using tight derivations for general logic programs. *Journal of Logic Programming*, 6(1):109–133, 1989.

[Van92] A. Van Gelder. The well-founded semantics of aggregation. In *Proceedings of the ACM Symposium on Principles of Database Systems*, pages 127–138, 1992.

[Van93] A. Van Gelder. The alternating fixpoint of logic programs with negation. *Journal of Computer and System Sciences*, 41(1):185–221, 1993.

[VBK91] L. Vieille, P. Bayer, and V. Küchenhoff. Integrity checking and materialized views handling by update propagation in the EKS-V1 system. Technical report, CERMICS - Ecole Nationale Des Ponts Et Chaussees, France, June 1991. Rapport de Recherche, CERMICS 91.1.

[vEK76] M. H. van Emden and R. A. Kowalski. The semantics of predicate logic as a programming language. *Journal of the ACM*, 23(4):733–742, October 1976.

[Vie86] Laurent Vieille. Recursive axioms in deductive databases: The query-subquery approach. In *Proceedings of the First International Conference on Expert Database Systems*, pages 179–193, Charleston, South Carolina, 1986.

[Vie87] Laurent Vieille. Database complete proof procedures based on SLD-resolution. In *Proceedings of the Fourth International Conference on Logic Programming*, pages 74–103, 1987.

[VRK+91] Jayen Vaghani, Kotagiri Ramamohanarao, David B. Kemp, Zoltan Somogyi, and Peter J. Stuckey. Design overview of the Aditi deductive database system. In *Proceedings of the Seventh International Conference on Data Engineering*, pages 240–247, April 1991.

[VRS91] A. Van Gelder, K. Ross, and J. S. Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM*, 38(3):620–650, 1991.

[War89] David S. Warren. The XWAM: A machine that integrates Prolog and deductive database query evaluation. Technical Report Tec. Rep. 89/25, Department of Computer Science, SUNY at Stony Brook, October 1989.

[War92] D. S. Warren. Memoing for logic programs. *Communications of the ACM*, 35(3):93–111, March 1992.

[ZAO93] C. Zaniolo, N. Arni, and K. Ong. Negation and aggregates in recursive rules: the ldl++ approach. In *Proc. Intl. Conf. on Deductive and Object-Oriented Databases*, Phoenixx, AZ, 1993.

[ZYT88] W. Zhang, C. T. Yu, and D. Troy. A necessary and sufficient condition to linearize doubly recursive programs in logic databases. Unpublished manuscript, Department of EECS, University of Illinois at Chicago, 1988.