At a workshop held in Toulouse, France, in 1977, Gallaire, Minker, and Nicolas stated that logic and databases was a field in its own right. This was the first time that this designation was made. The impetus for it started approximately 20 years ago in 1976 when I visited Gallaire and Nicolas in Toulouse, France. In this article, I provide an assessment about what has been achieved in the 20 years since the field started as a distinct discipline. I review developments in the field, assess contributions, consider the status of implementations of deductive databases, and discuss future work needed in deductive databases.

The use of logic and deduction in databases, as noted in Minker (1988b), started in the late 1960s. Prominent among developments was work by Levien and Maron (1965) and Kuhns (1967), and by Green and Raphael (1968a), who were the first to realize the importance of the Robinson (1965) resolution principle for databases. For early uses of logic in databases, see Minker (1988b), and for detailed descriptions of many accomplishments made in the 1960s, see Minker and Sable (1970).

A major influence on the use of logic in databases was the development of the field of logic programming: Kowalski (1974) promulgated the concept of logic as a programming language, and Colmerauer and his students developed the first Prolog interpreter (Colmerauer et al. 1973). I refer to logic programs that are function free as deductive databases (DDBs), or as datalog. I do so because databases are finite structures. Most of the results discussed can be extended to include logic programming.

The impetus for the use of logic in databases came about through meetings in 1976 in Toulouse, France, when I visited Herve Gallaire and Jean-Marie Nicolas while on sabbatical. The idea of a workshop on logic and databases was also conceived at this time. It is clear that a number of other individuals also had the idea of using logic as a mechanism to handle databases and deduction, and they were invited to participate in the workshop. The book Logic and Data Bases (1978), edited by Gallaire and Minker, was highly influential in the development of the field, as were the two volumes of Advances in Database Theory (Gallaire, Minker, and Nocholas 1984a, 1981) that were the result of two subsequent workshops held in Toulouse. Another influential development was the article by Gallaire, Minker, and Nicolas (1984b), which surveyed work in the field to that point.

The use of logic in databases was received by the database community with a great deal of skepticism: Was deductive databases (DDBs) a field? Did DDBs contribute to database theory or practice (Harel 1980)? The accomplishments I cite in this article are testaments to the fact that logic has contributed significantly both to the theory and the practice of databases. It is clear that logic has everything to do with the theory of databases, and many of those who were then critical of the field have changed their position. In the remainder of this article, I describe what I believe to be the major intellectual developments in the field, the status of commercial implementations, and future trends. As we see, the field of logic and databases has been prolific.

Intellectual Contributions of Deductive Databases

In 1970, Codd (1970) formalized databases in terms of the relational calculus and the relational algebra. He provided a logic language and the relational calculus and described how to compute answers to questions in the relational algebra and the relational calculus. Both
the relational calculus and the relational algebra provide declarative formalisms to specify queries. This was a significant advance over network and hierarchic systems (Ullman 1989, 1988), which only provided procedural languages for databases. The relational algebra and the relational calculus permitted individuals who were not computer specialists to write declarative queries and have the computer answer the queries. The development of syntactic optimization techniques (Ullman 1989, 1988) permitted relational database systems to retrieve answers to queries efficiently and compete with network and hierarchic implementations. Relational systems have been enhanced to include views. A view, as used in relational databases, is essentially a nonrecursive procedure. There are numerous commercial implementations of relational database systems for large database manipulation and for personal computers. Relational databases are a forerunner of logic in databases.

Although relational databases used the language of logic in the relational calculus, it was not formalized in terms of logic. The formalization of relational databases in terms of logic and the extensions that have been developed are the focus of this article. Indeed, formalizing databases through logic has played a significant role in our understanding of what constitutes a database, what is meant by a query, what is meant by an answer to a query, and how databases can be generalized for knowledge bases. It has also provided tools and answers to problems that would have been extremely difficult without the use of logic.

In the remainder of the article, I focus on some of the more significant aspects contributed by logic in databases: (1) a formalization of what constitutes a database, a query, and an answer to a query; (2) a realization that logic programming extends relational databases; (3) a clear understanding of the semantics of large classes of databases that include alternative forms of negation as well as disjunction; (4) an understanding of relationships between model theory, fixpoint theory, and proof procedures; (5) an understanding of the properties that alternative semantics can have and their complexity; (6) an understanding of what is meant by integrity constraints and how they can be used to perform updates, semantic query optimization (SQO), cooperative answering, and database merging; (7) a formalization and solutions to the update and view-update problems; (8) an understanding of bounded recursion and recursion and how they can be implemented in a practical manner; (9) an understanding of the relationship between logic-based systems and knowledge-based systems; (10) a formalization of how to handle incomplete information in knowledge bases; and (11) a correspondence that relates alternative formalisms of nonmonotonic reasoning to databases and knowledge bases.

I address the area of implementations of DDBs in Implementation Status of Deductive Databases, where commercial developments have not progressed as rapidly as intellectual developments. I then discuss some trends and future directions in Emerging Areas and Trends.

Formalizing Database Theory

Reiter (1984) was the first to formalize databases in terms of logic and noted that underlying relational databases were a number of assumptions that were not made explicit. One assumption deals with negation, that facts not known to be true in a relational database are assumed to be false. This assumption is the well-known closed-world assumption (CWA), expounded earlier by Reiter (1978). The unique-name assumption states that any item in a database has a unique name and that individuals with different names are different. The domain-closure assumption states that there are no other individuals than those in the database. Reiter then formalized relational databases as a set of ground assertions over a language \( \mathcal{L} \), together with a set of axioms. The language \( \mathcal{L} \) does not contain function symbols. These assertions and axioms are as follows:

**Assertions:** \( R(a_1, \ldots, a_n) \), where \( R \) is an \( n \)-ary relational symbol in \( \mathcal{L} \), and \( a_1, \ldots, a_n \) are constant symbols in \( \mathcal{L} \).

**Unique-name axiom:** If \( a_1, \ldots, a_p \) are all the constant symbols of \( \mathcal{L} \), then
\[
(a_1 \neq a_2), \ldots, (a_1 \neq a_p), (a_2 \neq a_3), \ldots, (a_{p-1} \neq a_p)
\]

**Domain-closure axiom:** If \( a_1, \ldots, a_p \) are all the constant symbols of \( \mathcal{L} \), then
\[
\forall x \ (x = a_1) \lor \cdots \lor (x = a_p)
\]

**Completion Axioms:** For each relational symbol \( R \), if \( R(a_1, \ldots, a_m) \), \( R(a_1', \ldots, a_m') \), \( R(a_1'', \ldots, a_m'') \) denote all facts under \( R \), the completion axiom for \( R \) is
\[
\forall x_1 \cdots \forall x_n \ (R(x_1, \ldots, x_n) \rightarrow (x_1 = a_1 \land \cdots \land x_n = a_m) \lor \cdots \lor (x_1 = a_1' \land \cdots \land x_n = a_m') \lor \cdots)
\]

**Equality Axioms:**
\[
\forall x \ (x = x)
\]
\[
\forall x \forall y \ ((x = y) \rightarrow (y = x))
\]
\[
\forall x \forall y \forall z \ ((x = y) \land (y = z) \rightarrow (x = z))
\]
\[
\forall x_1 \cdots \forall x_n \ (p(x_1, \ldots, x_n) \land (x_1 = y_1) \land \cdots \land (x_n = y_n) \rightarrow (p(y_1, \ldots, y_n))
\]
Example 1 illustrates the translation of a small database to logic. It is clear that handling such databases through conventional techniques will lead to a faster implementation. However, it serves to formalize previously unformalized databases.

The completion axiom was proposed by Clark (1978) as the basis for his negation-as-failure rule: It states that the only tuples that a relation can have are those that are specified in the relational table. This statement is implicit in every relational database. The completion axiom makes this explicit. Another contribution of logic programs and databases is that the formalization of relational databases in terms of logic permits the definition of a query and an answer to a query to be defined precisely. A query is a statement in the first-order logic language \( L \). \( Q(a) \) is an answer to a query, \( Q(X) \), over a database \( DB \) if \( Q(a) \) is a logical consequence of \( DB \).

**Deductive Databases**

Relational databases are a special case of DDBs. A DDB can be considered as a theory, \( DB \), in which the database consists of a set of ground assertions, referred to as the extensional database (EDB), and a set of axioms, referred to as the intensional database (IDB), of the form

\[ P \leftarrow Q_1, \ldots, Q_n, \quad (1) \]

where \( P, Q_1, \ldots, Q_n \) are atomic formulas in the language \( L \). Databases of this form are termed datalog databases (Ullman 1989, 1988). A datalog database is a particular instance of a more general Horn program that permits function symbols in clauses given by formula (1). The recognition that logic programs are significant for databases was understood by a number of individuals in 1976 (see Galler and Minker [1978] for references). The generalization permits views to be defined that are recursive.

The recognition that logic programming and databases are fundamentally related has led to more expressive and powerful databases than is possible with relational databases defined in terms of the relational algebra.

That logic programming and DDBs are fundamentally related is a consequence of the fact that databases are function-free logic programs. As shown in many papers and, in particular, Gottlob (1994), the expressive power of logic programming extends that of relational databases.

In addition to defining a database in terms of an EDB and an IDB, it is necessary to formalize what is meant by an integrity constraint. Kowalski (1978) suggests that an integrity constraint is a formula that is consistent with the DDB, but for Reiter (1984) and Lloyd and Topor (1985), an integrity constraint is a theorem of the DDB. For alternative definitions of integrity constraints, see Reiter (1990, 1988) and Demolombe and Jones (1996).

In DDBs, the semantic aspects of a database's design can be captured by integrity constraints. Information about functional dependencies—that a relation's key functionally determines the rest of the relation's attribute—can be written via integrity constraints. For example, assume the predicate flight for an airline database and that the attributes Airline and No. are a composite key for the relation.

One of the functional dependencies—that the departure time is functionally determined by airline and flight number—is represented by

\[ \text{Dtime}[1] = \text{Dtime}[2] \leftrightarrow \text{flight}(\text{Airline}, \text{No., Dtime}, \ldots), \text{flight}(\text{Airline, No., Dtime}, \ldots) \]

where \( \leftrightarrow \) is used to distinguish a rule from an integrity constraint.

Likewise, inclusion dependencies, which are

---

Consider the family database to consist of the Father relation with schema Father(father, child) and the Mother relation with schema Mother(mother, child). Let the database be

<table>
<thead>
<tr>
<th>FATHER</th>
<th>father</th>
<th>child</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>j</td>
<td>m</td>
</tr>
<tr>
<td></td>
<td>j</td>
<td>s</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MOTHER</th>
<th>mother</th>
<th>child</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>m</td>
<td></td>
</tr>
<tr>
<td>r</td>
<td>s</td>
<td></td>
</tr>
</tbody>
</table>

The database translated to logic is given as follows; we do not include the equality axioms because they are obvious.

**Assertions:** Father(j, m), Father(j, s), Mother(r, m), Mother(r, s), where Father and Mother are predicates, and \( j, m, s, r \) are constants.

**Unique-Name Axiom:**

\[ (i \neq j, (i \neq s), (j \neq r), (r \neq m), (r \neq s), (m \neq s)) \]

**Domain-Closure Axiom:**

\[ (\forall X)((X = j) \vee (X = m) \vee (X = s) \vee (X = r)) \]

**Completion Axioms:**

\[ (\forall X, \forall Y, \forall Z)(\text{Father}(X, Y) \leftarrow ((X = j) \land (X = m)) \lor ((X = j) \land (X = s))) \]

\[ (\forall X, \forall Y, \forall Z)(\text{Mother}(X, Y) \leftarrow ((X = j) \land (X = m)) \lor ((X = j) \land (X = s))) \]

---

Example 1. Translation of a Small Database to Logic.
A cooperative answering system provides information to users about why a particular query succeeded or failed.

Also common semantic information from a database's design, are easily represented. For example, the predicate airport records various information about airports known to the database. We want to ensure that any airport that serves as a departure or an arrival of any flight known to the database is also in the airport relation. The first of these—that the departure airport is known—could be represented as follows:

\[
\text{airport}(-, \ldots, -, \text{Fieldcode}) \iff \text{flight}(-, \ldots, -, \text{Fieldcode}) \]

The major use made of integrity constraints has been in updating, to assure that the database is consistent. Nicolas (1979) used techniques from DDBs to speed database update. Blaustein (1981) has also made contributions to this problem. Reiter (1978a) showed that one can query a Horn database with or without integrity constraints, and the answer to the query is the same. However, integrity constraints can be used to advantage in the query process. Although integrity constraints do not affect the result of a query, they can affect the efficiency with which answers can be computed. Integrity constraints provide semantic information about data in the database. If a query requests a join for which there will never be an answer because of system constraints, then an unnecessary join on two potentially large relational tables in a relational database system or performing a long deduction in a DDB is not needed when integrity constraints imply the answer is empty. The process of using integrity constraints to constrain a search is called semantic query optimization (SQO) (Chakravarthy, Grant, and Minker 1990). McKim and Minker (1977) were the first to use integrity constraints for SQO in DDBs. Hammer and Zdonik (1980) and King (1981) were the first to apply SQO to relational databases. Chakravarthy, Grant, and Minker (1990) formalized SQO and developed the partial subsumption algorithm and method of residues. The partial subsumption algorithm and method of residues provides a general technique applicable to any relational database or DDB that is able to perform SQO. The general approach to SQO described in Chakravarthy, Grant, and Minker (1990) has been extended to perform bottom-up evaluation (Godfrey, Gryz, and Minker 1996); to include databases with negation in the body of clauses (Gaasterland and Lobo 1993); and to handle recursive IDB rules (Levy and Sagiv 1995).

A topic related to SQO is that of cooperative answering systems. A cooperative answering system provides information to users about why a particular query succeeded or failed. When a query fails, a user, in general, cannot tell why the failure occurred. There can be several reasons: The database currently does not contain information to respond to the user, or there will never be an answer to the query. The distinction could be important to the user. Another aspect related to integrity constraints is that of user constraints. A user constraint is a formula that models a user's preferences. It can constrain providing answers to queries in which the user might have no interest (for example, stating that in developing a route of travel, the user does not want to pass through a particular city) or provide other constraints that might restrict the search. As shown by Gaasterland, Godfrey, and Minker (1992b), user constraints, which are identical in form to integrity constraints, can be used for this purpose. Although integrity constraints provide the semantics of the entire database, user constraints provide the semantics of the user. User constraints can be inconsistent with the database; hence, these two semantics are maintained separately. To maintain the consistency of the database, only integrity constraints are relevant. A query can be thought of as the conjunction of the query itself and the user constraints. A query can be optimized semantically based on both integrity constraints and user constraints.

As noted previously, integrity constraints are versatile; they do more than just represent dependencies. General semantic information can be captured as well. Assume that at the national airport in Washington, D.C. (DCA), that no flights are allowed (departures or arrivals) after 10:00 PM or before 8:00 AM because the airport is downtown, and night flights would disturb city residents. This information can be captured as an integrity constraint.

Such knowledge, captured and recorded as integrity constraints, can be used to answer queries to the database more intelligently and more informatively. If someone asks for flights out of DCA to, say, Los Angeles International Airport leaving between 10:30 PM and 12:00 AM, the database could simply return the empty answer set. (There will be no such flights if the database is consistent with its constraints.) It would be better, however, for the database system to inform the querier that there can be no such flights because of the Washington, D.C., flight regulations.

Using user constraints and integrity constraints, one can develop a system that informs users why a query succeeds or fails (Gaasterland et al. 1992). Other features can be incorporated, such as the ability to relax a query, termed query relaxation, given that it fails, so that an answer to a related request can be found. See
Horn Semantics and Datalog are described in the following subsections. Alternative extensions and their significance in the body of rules were permitted. These other DDBs that might contain negated atoms are DDBs, or datalog databases. Subsequently, a rule is a conjunction of atoms. These databases in which the head of a rule is an atom, and the body of a rules in a database, that is, rules in which the was to permit function-free recursive Horn Database Semantics

The first generalization of relational databases was to permit function-free recursive Horn rules in a database, that is, rules in which the head of a rule is an atom, and the body of a rule is a conjunction of atoms. These databases are DDBs, or datalog databases. Subsequently, other DDBs that might contain negated atoms in the body of rules were permitted. These alternative extensions and their significance are described in the following subsections.

Horn Semantics and Datalog

One of the early developments was by van Emden and Kowalski (1976), who wrote a seminal paper on the semantics of Horn theories. Van Emden and Kowalski made a significant contribution to logic and databases by recognizing that the semantics of Horn databases can be characterized in three distinct ways: (1) model theory, (2) fixpoint theory, and (3) proof theory. These three characterizations lead to the same semantics.

Model theory deals with a collection of models that capture the intended meaning of the database. Fixpoint theory deals with a fixpoint operator that constructs the collection of all atoms that can be inferred to be true from the database. Proof theory deals with a procedure that finds answers to queries with respect to the database. van Emden and Kowalski (1976) showed that the intersection of all Herbrand models of a Horn DDB is a unique minimal model. The unique minimal model is the same as all the atoms in the fixpoint and are the only atoms provable from the theory.

To find if the negation of a ground atom is true, one can subtract, from the Herbrand base (the set of all atoms that can be constructed from the constants and the predicates in the database), the minimal Herbrand model. If the atom is contained in this set, then it is assumed to be false, and its negation is true. Alternatively, answering queries that consist of negated atoms that are ground can be achieved using negation-as-finite failure as described by Reiter (1978b) and Clark (1978).

The first approaches to answering queries in DDBs did not handle recursion and were primarily top-down (or backward reasoning) (Gallaire and Minker 1978). Answering queries in relational database systems was a bottom-up (or forward-reasoning) approach because all answers are usually required, and it is more efficient to do so in a bottom-up approach. The major approaches to handling recursion are based on the renaming of the Alexander (Rohmer, Lescoeur, and Kerisit 1986) and magic set (Bancilhon et al. 1986) methods, which make use of constants that appear in a query and perform search by bottom-up reasoning. Bry (1990) reconciles the bottom-up and top-down methods to compute recursive queries.

The major approaches to handling recursion are based on the renaming of the Alexander (Rohmer, Lescoeur, and Kerisit 1986) and magic set (Bancilhon et al. 1986) methods, which make use of constants that appear in a query and perform search by bottom-up reasoning. Bry (1990) reconciles the bottom-up and top-down methods to compute recursive queries. He shows that the Alexander and magic set methods based on rewriting and the methods based on resolution implement the same top-down evaluation of the original database rules by means of auxiliary rules processed bottom-up. For pioneering work on recursion and alternative methods, see Minker (1996). Minker and Nicolas (1982) were the first to show that there are forms of rules that lead to bounded recursion, in which the deduction process must terminate in a finite number of steps. This work has been extended by Naughton and Sagiv (1987). Example 2 illustrates a rule that terminates finitely regardless of the state of the database.

The efficient handling of recursion and the recognition that some recursive cases might inherently be bounded contributes to the practical implementation of DDBs. An understanding of the relationship between resolution-based (top-down) and fixpoint-based (bottom-up) techniques and how the search space of the latter can be made identical to top-down resolution with program transformation is another contribution of DDBs.

Extended Deductive Databases and Knowledge Bases

The ability to develop a semantics for theories in which there are rules with a literal (that is, an atomic formula or the
If a rule satisfies the condition that it is singular, then it is bound to terminate in a finite number of steps independent of the state of the database. A recursive rule is singular if it is of the form

\[ R \leftarrow F \land R_1 \land \ldots \land R_n, \]

where \( F \) is a conjunction of possibly empty base (that is, EDB) relations and \( R, R_1, R_2, \ldots, R_n \) are atoms that have the same relation name iff (1) each variable that occurs in an atom \( R_i \) and does not occur in \( R \) only occurs in \( R_i \), and (2) each variable in \( R \) occurs in the same argument position in any atom \( R_i \) where it appears, except perhaps in at most one atom \( R_j \) that contains all the variables of \( R \).

Thus, the rule

\[ R(X, Y, Z) \leftarrow R(X, Y', Z), R(X, Y, Z') \]

is singular because (1) \( Y' \) and \( Z' \) appear, respectively, in the first and second atoms in the head of the rule (condition 1) and (2) the variables \( X, Y, \) and \( Z \) always appear in the same argument position (condition 2).

Example 2. Bounded Recursion.

The rules

\[
\begin{align*}
&s_1: p \leftarrow q, \text{not } r \\
&s_2: q \leftarrow p \\
&s_3: q \leftarrow s \\
&s_4: s \\
&s_5: r \leftarrow t
\end{align*}
\]

make up a stratified theory. Rule \( s_5 \) is in the lowest stratum, but the other rules are in a higher stratum. The predicate \( p \) is in a higher stratum than the stratum for \( r \) because it depends negatively on \( r \). \( q \) is in the same stratum as \( p \) because it depends on \( p \). \( s \) is also in the same stratum as \( q \). The meaning of the stratified program is that \( s, q, \) and \( p \) are true, but \( t \) and \( r \) are false. \( t \) is false because there is no defining rule for \( t \). Because \( t \) is false, \( r \) is false. \( s \) is given as true; hence, \( q \) is true. Because \( q \) is true, and \( r \) is false, from rule \( s_1 \), \( p \) is true.

Example 3. Stratified Program.

negation of an atomic formula) in the head and literals with possibly negated-by-default literals in the body of a clause has significantly expanded the ability to write and understand the semantics of complex applications. Such clauses, referred to as extended clauses, are given by

\[ L \leftarrow M_1, \ldots, M_n, \text{not } M_{n+1}, \ldots, \text{not } M_{n+k}, \]

where \( L \) and the \( M_j, j = 1, \ldots, (n+k) \) are literals. Such databases combine both classical negation and default negation (represented by not immediately preceding a literal) and are referred to as extended DDBs. The combining of classical and default negation provides users with greater expressive power.

Logic programs where default negation can appear in the body of a clause first appeared in the Workshop on Foundations of Deductive Databases and Logic Programming in August 1986. Selected papers from the workshop were published in Minker (1988a). The concept of stratification was introduced to logic programs by Apt, Blair, and Walker (1988) and Van Gelder (1988), who considered stratified theories in which \( L \) and the \( M_j \) in formula 2 are atomic formulas, and there is no recursion through negation. Apt, Blair, and Walker, and Van Gelder, show that there is a unique preferred minimal model, computed from strata to strata. Przymusinski (1988) termed this minimal model the perfect model. When one has a theory that is stratified, one can place clauses in different strata, where predicates in the head of a rule are in a higher stratum than predicates that are negated in the body of the clause, as explained in example 3. Thus, one can compute the positive predicates in a lower stratum, and the negated predicate's complement is true in the body of the clause if the positive atom has not been computed in the lower stratum.

The theory of stratified databases was followed by permitting recursion through negation in formula 2, where the \( L \) and \( M_j \) are atomic formulas. Example 4 illustrates a database that cannot be stratified. In the context of DDBs, they are called normal DDBs. Many papers have been devoted to defining the semantics of these databases. A summary of these semantics is given in Minker and Ruiz (1994). The most prominent of this work for the Horn case are the well-founded semantics (WFS) of Van Gelder, Ross, and Schlipf (1988) and the stable semantics of Gelfond and Lifschitz (1988). The WFS leads to a uniquethree-valued model. Stable semantics can lead to a collection of minimal models. For some DDBs, this collection can be empty. Fitting (1985) also defined a three-valued model to capture the semantics of normal logic programs. For additional work, see Minker (1996).
There have been several implementations of the WFS. Chen and Warren (1993) developed a top-down approach to answer queries in this semantics, while Leone and Rullo (1992) developed a bottom-up method for datalog databases. Several methods have been developed for computing answers to queries in stable model semantics. Fernández et al. (1993) developed a bottom-up approach based on the concept of model trees. Every branch of a model tree is a model of the database, where a node in a tree is an atom that is shared by each branch below the node. (See example 6 for an illustration of a model tree.) Bell et al. (1993) developed a method based on linear programming. See Minker (1996) for additional methods to compute the well-founded, the stable model, and other related semantics.

A further extension of normal DDBs, proposed by Gelfond and Lifschitz (1990) and Pearce and Wagner (1989), permits clauses in formula 2, where L and M are literals, and, therefore, combines classical and default negation in one database. The semantics for normal DDBs was described by Minker and Ruiz (1994).

These notions of default negation have been used as separate ways to interpret and deduce default information. That is, each application has chosen one notion of negation and applied it to every piece of data in the domain of the application. Minker and Ruiz (1996) defined a new class of more expressive DDBs that allow several forms of default negation in the same database. In this way, different pieces of information in the domain can be treated appropriately. They introduced a new semantics called the well-founded stable semantics that characterizes the meaning of DDBs that combine the well-founded and the stable semantics. Schlipf (1995) has written a comprehensive survey article on complexity results for DDBs.

The development of the semantics of extended DDBs that permit a combination of classical negation and multiple default negations in the same DDB are important contributions. The study and development of results in the computational complexity of these databases are important contributions to database theory. They permit wider classes of application to be developed.

Knowledge bases are important for AI and expert system developments. A general way to represent knowledge bases is through logic. Work developed for extended DDBs concerning semantics and complexity applies directly to knowledge bases. Baral and Gelfond (1994) describe how extended DDBs can be used to represent knowledge bases. For an example of a knowledge base, see example 5. Extended DDBs, together with integrity constraints, permit a wide range of knowledge bases to be implemented. Many papers devoted to knowledge bases consider them to consist of facts and rules, which is one aspect of a knowledge base, as is the ability to extract proofs. However, integrity constraints supply another aspect of knowledge and differentiate knowledge bases, which can have the same rules but different integrity constraints. One should define a knowledge base as consisting of an extended DDB plus integrity constraints.

Since alternative extended deductive semantics have been implemented, the knowledge base expert can now focus on the problem to be implemented, that is, on writing rules and integrity constraints that characterize the knowledge bases, selecting the particular semantics that meets the needs of the problem, and employing a DDB system that uses the required semantics. The field of DDBs has contributed to providing an understanding of knowledge bases and their implementation.

Extended Disjunctive Deductive Database Semantics

In the databases discussed previously, information is definite. However, there are many situations where our knowledge of the world is incomplete. For example, when a null value appears as an argument of an attribute of a relation, the value of the attribute is unknown. Uncertainty in databases can be represented by probabilistic information (Ng and Subrahmanian 1993). Another area of incompleteness arises when it is unknown which among several facts are true, but it is known that one or more facts are true, but it is known that one or more
Consider the following database, where the predicate $p(X,Y)$ denotes that $X$ is a professor in department $Y$, $a(X,Y)$ denotes that individual $X$ has an account on machine $Y$, and $ab(W,Z)$ denotes that it is abnormal in rule $W$ to be individual $Z$.

We want to represent the following information, where mike and john are professors in the Computer Science Department:

First, as a rule, professors in the Computer Science Department have Vax accounts. This rule is not applicable to mike. He might or might not have an account on this machine.

Second, every computer science professor has a Vax or an IBM account but not both. These rules can be captured in the following DDB:

$$p(mike, cs) \leftarrow$$
$$p(john, cs) \leftarrow$$
$$\neg p(X, Y) \leftarrow \text{not } p(X, Y)$$
$$a(X, vax) \leftarrow p(X, cs), \text{not } ab(r4, X), \text{not } \neg a(X, vax)$$
$$ab(r4, mike) \leftarrow$$
$$a(X, vax) \lor a(X, ibm) \leftarrow p(X, cs), ab(r4, X)$$
$$\neg a(X, ibm) \leftarrow p(X, cs), a(X, vax)$$
$$\neg a(X, vax) \leftarrow p(X, cs), a(X, ibm)$$
$$a(X, ibm) \leftarrow a(X, vax), p(X, cs)$$.

The third rule states that if by default negation, predicate $p(X, Y)$ fails, then $p(X, Y)$ is logically false. The other rules encode the statements listed previously.

From this formalization, one can deduce that john has a Vax account, but mike has either a Vax or an IBM account but not both.

Example 5. Knowledge Base (Baral and Gefond 1994).

is true. Therefore, it is necessary to be able to represent and understand the semantics of theories that include incomplete data. A natural way to extend databases to include incomplete data is to permit disjunctive statements as part of the language, where clauses can have disjunctions in their heads. These clauses are represented as

$$L_1 \lor L_2 \lor ... \lor L_m \leftarrow M_1, ..., M_n, \text{not } M_{n+k}, ... \text{not } M_{n+k}$$ \hspace{1cm} (3)

and are referred to as extended disjunctive clauses. Such databases are referred to as extended disjunctive deductive databases (EDDDBs). Foundations of Disjunctive Logic Programming by Lobo, Minker, and Rajasekar (1992) describes the theory of disjunctive logic programs and includes several chapters on disjunctive deductive databases (DDDBs). Example 5 illustrates the use of such a theory of databases.

I first discuss the semantics of DDDBs, where clauses are given by formula 3, where the literals are restricted to atoms, and there is no default negation in the body of a clause. Next, I discuss the semantics of EDDDBs, where there are no restrictions on clauses in formula 3.

### Disjunctive Deductive Databases

As noted in Minker (1989), work in disjunctive theories was pursued seriously after a workshop organized in 1986 (Minker 1986). The field of DDDBs started approximately in 1982 with the appearance of a paper I wrote (Minker 1982), in which I described how one can answer both positive and negated queries in such databases. For a historical perspective of disjunctive logic programming and DDDBs, see Minker (1989). There is a major difference between the semantics of DDBs and those for DDDBs. Whereas DDBs typically have a unique minimal model that describes the meaning of the database, DDDBs generally have multiple minimal models.

As shown in Minker (1982), it is sufficient to answer positive queries over DDBs by showing that the query is satisfied in every minimal model of the database. Thus, in the DDBB $a \lor b$, there are two minimal models: (1) $\{a\}$ and (2) $\{b\}$. The query, $a\lor b$, is not satisfied in the model $b$; hence, it cannot be concluded that $a$ is true. However, the query $(a \lor b)$ is satisfied in both minimal models; hence, the answer to the query $a \lor b$ is yes. To answer negated queries, it is not sufficient to use Reiter’s (1978) CWA because as he noted, from the theory $DB = a \lor b$, it is not possible to prove $a$, and it is not possible to prove $b$. Hence, by the CWA, not $a$ and not $b$ follow. However, $\{a \lor b, \text{not } a, \text{not } b\}$ is not consistent. The generalized closed-world assumption (GCWA) (Minker 1982) resolves this problem by specifying that a negated atom be considered true if the atom does not appear in any minimal model of the database. The GCWA provides a model-theoretic definition of negation. An equivalent proof-theoretic definition, also presented in Minker (1982), is that an atom $a$ can be considered false if whenever $a \lor C$ can be proven from the database, then $C$ can also be proven from the database, where $C$ is an arbitrary positive clause. For related work on negation in disjunctive theories, see Minker (1996). For surveys on negation in DDBs and DDDBs, see Shepherdson (1987), Apt and Boi (1994), and Minker (1993).

In DDBs, it is natural for the fixpoint operator to map atoms to atoms. However, for DDDBs, it is natural to map positive disjunctions to positive disjunctions. A set of positive disjunctions is referred to as a state. A model state is a state whose minimal models all satisfy
the DDDB. The concept of a state was defined by Minker and Rajasekar (1990) as the domain of a fixpoint operator \( T_P \) whose least fixpoint characterizes the semantics of a disjunctive logic program \( P \). The operator is shown to be monotonic and continuous; hence, it converges in a countably infinite number (\( \omega \)) of iterations. The fixpoint computation operates bottom-up and yields a minimal model state that is logically equivalent to the set of minimal models of the program. The Minker-Rajasekar fixpoint operator is an extension of the van Emden–Kowalski fixpoint operator. If one considers all model states of a DDDB and intersects them, then the resultant is a model state, and among all model states, it is minimal. Hence, one obtains a unique minimal model in a Horn database, but one obtains a unique model state in a DDDB. See Decker (1991) for a related fixpoint operator for DDDBs.

Answering queries in DDDBs has been studied by a number of individuals, as described in Minker (1996). I focus on the work of Fernández and Minker (1991), who developed the concept of a model tree. They show how one can incrementally compute sound and complete answers to queries in hierarchical DDDBs. A model tree is shown in example 6. A DDDB is hierarchical if it contains no recursion. Fernández et al. (1993) show how one can develop a fixpoint operator over trees to capture the meaning of a DDDB that includes recursion. The tree representation of the fixpoint is equivalent to the Minker-Rajasekar fixpoint (Minker and Rajasekar 1990). Fernández and Minker compute the model tree of the extensional DDDB once. To answer queries, intensional database rules can be invoked. However, the models of the extensional disjunctive part of the database do not have to be generated for each query. Their approach to computing answers generalizes both to stratified and normal DDDBs.

Loveland and his students (Loveland, Reed, and Wilson 1993) have developed a top-down approach when the database is near Horn. They have developed a case-based reasoner that uses Prolog to perform the reasoning. This effort is one of the few that have implemented DDDBs. Loveland, Reed, and Wilson (1993) introduced a relevancy-detection algorithm to be used with SATCHMO, developed by Manthey and Bry (1988), for automated theorem proving. Their system, termed SATCHMORE (SATCHMO with RELevancy), improves on SATCHMO by limiting uncontrolled use of forward chaining. Seipel (1995) has developed a system, DISLOG, that incorporates many different disjunctive theories and strategies. The system is available on the World Wide Web.

See Minker (1996) for references to work on the complexity of answering queries in disjunctive logic programs and Eiter and Gottlob (1995) for complexity results for propositional logic programs.

The development of model-theoretic, fixpoint, and proof procedures placed the semantics of DDDBs on a firm foundation. Methods to handle DDDBs are being developed and should eventually enhance implementations. The GCWA and alternative theories of negation have enhanced our understanding of default negation in DDDBs. Complexity results provide an understanding of the difficulties in finding answers to queries in such systems.

**Extended Disjunctive Deductive Databases**

Fernández and Minker (1995) present a new fixpoint characterization of the minimal models of disjunctive and stratified DDDBs. They prove that by applying the operator iteratively, in the limit, it constructs the perfect model semantics (Przymusinski 1988) of stratified DDDBs. Given the equivalence between the perfect model semantics of stratified programs and prioritized circumscription (Przymusinski 1988), their fixpoint characterization captures the meaning of the corresponding circumscribed theory. Based on these results, they present a bottom-up evaluation algorithm for
Articles

stratified DDBs. This algorithm uses the model-tree data structure to represent the information contained in the database and to compute answers to queries. Fernández and Minker (1992) develop the theory of DDBs using the concept of model trees. Work on updates in DDBs is described in Fernández, Grant, and Minker (1996).

Four alternative semantics were developed for nonstratifiable normal DDBs at approximately the same time: (1) Ross (1989), the strong WFS; (2) Baral, Lobo, and Minker (1990), the generalized disjunctive WFS (GDWFS); and (3, 4) two semantics by Przymusinski, an extension of the stable model semantics (Przymusinski 1990) for normal disjunctive databases and the stationary semantics (Przymusinski 1990). A number of other important semantics have been developed. Przymusinski (1995) describes a new semantic framework for disjunctive logic programs and introduces the static expansions of disjunctive programs. The class of static expansions extends both the classes of stable, well-founded, and stationary models of normal programs and the class of minimal models of disjunctive programs. Any static expansion of a program $P$ provides the corresponding semantics for $P$ consisting of the set of all sentences logically implied by the expansion. The stable model semantics has also been extended to disjunctive programs (Gelfond and Lifschitz 1991; Przymusinski 1991). The disjunctive WFS (DWFS) of Brass and Dix (1995) is also of considerable interest because it permits a general approach to bottom-up computation in disjunctive programs.

As noted previously, a large number of different semantics exist for both EDDBs and EDDDBs. A user who wants to use such a system is faced with the problem of selecting the appropriate semantics for his/her needs. No guidelines have been developed. However, many complexity results have been obtained for these semantics. Schlipf (1995) and Eiter and Gottlob (1995) have written comprehensive survey articles that summarize the complexity results that are known for alternative semantics.

In addition to the results for extended disjunctive theories, there is work in investigating tractable cases of disjunctive theories. Ben-Eliyahu and Dechter (1994) introduced the concept of a head-cycle-free (HCF) clause. Let a clause consist of a disjunction of literals. A dependency graph $G_p$ is associated with each program $P$ as follows:

First, each clause of the form, formula 2, and each predicate in $P$ is a node.

Second, there is a positive (negative) arc from a predicate node $p$ to a rule node $\xi$ iff $p$ appears positive (negative) in the body of $\xi$ and an arc from $\zeta$ to $p$ (resp., and also an arc from $p$ to $\zeta$ if $p$ appears in the head of $\zeta$.

The positive dependency graph of $P$ is a subgraph of $G_p$ containing only positive arcs. A directed cycle in $G_p$ is called negative if it contains at least one negative arc. A DDB $P$ is HCF if for every two predicate names $p$ and $q$, if $p$ and $q$ are on a positive directed cycle in the dependency graph $G_p$, then there is no rule in $P$ in which both $p$ and $q$ appear in the head. It is shown in Ben-Eliyahu, Palopoli, and Zemlyank (1996) that answers to queries expressed in this language can be computed in polynomial time. Furthermore, the language is sufficiently powerful to express all polynomial time queries. It is further shown in Ben-Eliyahu and Palopoli (1994) that there is an algorithm that performs, in polynomial time, minimal model finding and minimal model checking if the theory is HCF. An efficient algorithm for solving the (co-NP-hard decision) problem of checking if a model is stable in function-free disjunctive logic programs is developed in Leone, Rullo, and Scarcello (1996). They show that the algorithm runs in polynomial time on the class of HCF programs, and in the case of general disjunctive logic programs, it limits the inefficient part of the computation only to the components of the program that are not HCF.

In addition to work on tractable databases, consideration has been given to approximate reasoning. In such reasoning, one can give up soundness or completeness of answers. Efforts have been developed both for DDBs and DDBs by Kautz and Selman (1992) and Selman and Kautz (1996), who developed lower and upper bounds for Horn (datalog) databases and compilation methods; Cadoli (1993), who developed computational and semantic approximations; and del Val (1995), who developed techniques for approximating and compiling databases. See also Cadoli (1996) for additional references concerning compilation, approximation, and tractability of knowledge bases.

A second way to determine the semantics to be used is through their properties. Dix (1992) proposed a large number of criteria that are useful in determining the appropriate semantics to be used. Properties deemed useful are (1) elimination of tautologies, where one wants the semantics to remain the same if a tautology is eliminated; (2) generalized principle of partial evaluation, where if a rule is replaced by a one-step deduction, the semantics is unchanged; (3) positive-negative reduction; (4) elimination of nonminimal rules, where a subsumed rule is
The field of DDBs has made significant intellectual contributions over the past 20 years. However, these contributions have not been matched by implementations that are available in the commercial market.

Implementation Status of Deductive Databases

The field of DDBs has made significant intellectual contributions over the past 20 years. However, these contributions have not been matched by implementations that are available in the commercial market. In the early 1970s, when Codd (1970) introduced the relational model, there were numerous debates in the database community about the efficacy of such systems relative to network and hierarchical systems (Date 1995). These debates ended when an effective relational system was implemented and shown to be comparable to these systems. Now, some of those individuals who are prominent in relational databases claim that DDBs are not effective and are not needed. Although I believe otherwise, these comments can be addressed better when a full commercial implementation of a DDB is available or when many of the techniques introduced in DDBs find their way into relational databases. I believe that both of these are beginning to happen.

In the following subsection, I discuss the stages through which implementations of DDBs have progressed and some contributions made during each stage. Following this, I discuss the reasons why I believe that no current
systems are commercially marketed and speculate on how this situation might change.

**Deductive Database Systems**

There have been three stages of implementations of DDBs: (1) pre-1970, (2) 1970 to 1980, and (3) 1980 to the present. Each stage has contributed toward understanding the problems inherent in developing DDB systems.

**First Stage: Pre-1970s**

Two efforts stand out during this period: the first by Levien, and Maron (1965) and Kuhns (1967), who developed a prototype system that demonstrated the feasibility of performing deduction in databases and the second by Green and Raphael (1968a, 1968b), who recognized that the resolution method of Robinson (1965) was a uniform procedure based on a single rule of inference that could be used for DDBs. This was the first general approach to DDBs. The work by Levien and Maron (1965) and Kuhns (1967) on Relational Data File (RDF) started in 1963. A procedural language, INFEREX, executed inference routines. Plausible and formal inferencing were both possible in RDF, as was temporal reasoning. The system was implemented on a file consisting of some 55,000 statements. The work by Green and Raphael (1968a, 1968b) resulted in a system termed the question-answering system (QA-3.5). It was an outgrowth of Raphael's thesis on semantic information retrieval (SIR) (Raphael 1968) that performed deduction. QA-3.5 included a natural language component. Another deductive system, relational store structure (RSS), started in 1966 and was developed by Marrill (Computer Corporation 1967). The system had 12 deductive rules built into the program and was able to incorporate other deductive rules. The association store processor (ASP), developed by Savitt, Love, and Troop (1967), also performed deduction over binary relational data. The inference rules, specified as relational statements, were handled by breadth-first, followed by depth-first, search. These efforts, as well as those cited in Minker and Sable (1970), were important precursors to DDBs. In table 1, adapted from Minker and Sable (1970), I list some capabilities of systems developed during this stage.

**Second Stage: 1970 to 1980**

Whereas the first stage could be characterized as using ad hoc techniques for deduction (except for the work by Green and Raphael), the second-stage systems were based on the Robinson resolution principle, as first recognized by Green and Raphael. The SYNTEX system built by Nicolas and Syre (1974) used logic as the basis for deduction. The work by Chang (1978) on the DEDUCE 2 system, Kellogg, Klahr, and Travis (1978) on the Deductively Augmented Data Management (DADM) system, and Minker (1978) on the Maryland Refutation Proof Procedure System (MRPPS 3.0) represent work during the second stage of development of DDBs. These papers appear in Gallaire and Minker (1978). Table 2 provides a brief summary of some of the features of these systems.

DADM precomputed unifications among premises so they did not have to be recomputed during deduction. Variables were typed. Inference plans and database-access strategies were created from a premise file without requiring access to database values.

MRPPS 3.0 performed top-down searches for large databases. It permitted arguments of predicates to contain function symbols and had a knowledge base index to access the data. The deductive system used a typed unification algorithm and a semantic network. The SQO method described in McSkimin and Minker (1977) was incorporated into the system. Answer extraction, natural language processing, and voice output were part of the system.

The DEDUCE 2 system performed deduction over databases. Nonrecursive Horn rules were used and were compiled in terms of base relations. Integrity constraints were used to perform SQO on queries. Problems with respect to recursive rules and termination were also discussed (Chang 1981).

**Third Stage: 1980 to Present**

A large number of prototype DDBs were developed, and most are described in Ramakrishnan and Ullman (1995). I briefly discuss several major efforts: work at the European Computer Research Consortium (ECRC) led by Nicolas, at the University of Wisconsin led by Ramakrishnan, and at Stanford led by Ullman. They attempted to develop operational and possibly commercial DDBs. They contributed significantly to both the theory and the implementation of DDBs. Detailed descriptions of contributions made by these systems and others can be found in Ramakrishnan and Ullman (1995).

In table 3, I list some of the capabilities of systems developed in this stage, adapted from Ramakrishnan and Ullman (1995).

Implementation efforts at ECRC on DDBs started in 1984 and led to the study of algorithms and prototypes: deductive query-evaluation methods (OSQ-SLD and others) (Vieille 1986); integrity checking (SOUNDCHECK) (Decker 1986); the DDB EKS(-V1) by Vieille and his team (1990); hypothetical reasoning and checking (Vieille, Bayer, and Kuechenhoff 1996); and aggregation through recursion (Lefebvre 1994). The EKS system used a topdown evaluation method and was released to
ECRC shareholder companies in 1990.

Implementation efforts at MCC Corporation on a DDB started in 1984 and emphasized bottom-up evaluation methods (Tsur and Zaniolo 1986) and query evaluation using such methods as seminaive evaluation, magic sets and counting (Beeri and Ramakrishnan 1991; Bancilhon et al. 1986; Sacca and Zaniolo 1986), semantics for stratified negation and set grouping (Beeri et al. 1991), investigation of safety, the finiteness of answer sets, and join-order optimization. The LDL system was implemented in 1988 and released in the period 1989 to 1991. It was among the first widely available DDBs and was distributed to universities and shareholder companies of MCC.

<table>
<thead>
<tr>
<th>Name</th>
<th>Organization</th>
<th>Computer</th>
<th>Programming language</th>
<th>Input language model</th>
<th>Syntactic analysis technique</th>
<th>Semantic analysis technique</th>
<th>Intermediate language</th>
<th>Data structures</th>
<th>Inference procedures</th>
<th>Output language</th>
</tr>
</thead>
<tbody>
<tr>
<td>QA-3.5 (Green and Raphael 1968a, 1968b) (question-answering system)</td>
<td>Stanford Research Institute</td>
<td>PDP 10</td>
<td>Lisp 1.5</td>
<td>Near-natural language model based on simple transformations and context-free grammar</td>
<td>Earley algorithm for context-free grammar</td>
<td>Semantics stack built during syntax analysis phase</td>
<td>First-order predicate calculus</td>
<td>Lisp-chained list structures</td>
<td>Formal theorem proving by Robinson resolution procedure</td>
<td>Near-natural language generated in a synthesis phase</td>
</tr>
<tr>
<td>ASP (Savitt, Love, and Troop 1967) (association storing processor)</td>
<td>Hughes Aircraft Corp.</td>
<td>IBM 360/65</td>
<td>Assembly language</td>
<td>Stylized input form and a procedural language</td>
<td>N/A</td>
<td>N/A</td>
<td>Binary relations</td>
<td>Relational statement elements randomized (coded) and replicated statements stored under each element</td>
<td>Inference rules specified as relational statements handled by breadth first followed by depth</td>
<td>Relational statements</td>
</tr>
<tr>
<td>RDF (Levien and Maron 1965) (relational data file)</td>
<td>RAND Corp.</td>
<td>IBM 7044 360/65</td>
<td>Assembly language</td>
<td>Stylized input forms analyzed by FOREMAN language</td>
<td>N/A</td>
<td>N/A</td>
<td>Binary relations</td>
<td>Files quadruplicated and ordered by statement number and three elements</td>
<td>Plausible inference rules specified in a procedural language called INFEREX</td>
<td>Relational statements</td>
</tr>
<tr>
<td>RSS (Computer Corporation 1967) (relational structures system)</td>
<td>Computer Corporation of America</td>
<td>IBM 360/75</td>
<td>Assembly language</td>
<td>Near-natural language model based on matching sentence templates</td>
<td>Match of sentence against stored templates</td>
<td>Pattern ⇒ action operation invoked as a result of template match</td>
<td>n-ary relations (n ≤ 7 as implemented)</td>
<td>Statement elements are hash coded and open statements linked to corresponding closed statements</td>
<td>Twelve general rules of deductive logic used</td>
<td>Near-natural language generated from n-ary relational statements</td>
</tr>
</tbody>
</table>

Table 1. First-Stage Deductive Database Implementations (adapted from Minker and Sable [1970]).
Implementation efforts at the University of Wisconsin on the CORAL DDBs started in the 1980s. Bottom-up and magic set methods were implemented. The system, written in C and C++, is extensible and provides aggregation for modularly stratified databases. CORAL supports a declarative language and an interface to C++ that allows for a combination of declarative and imperative programming. The declarative query language supports general Horn clauses augmented with complex terms, set grouping, aggregation, negation, and relations with tuples that contain universally quantified variables. CORAL supports many evaluation strategies and automatically chooses an efficient evaluation strategy. Users can guide query optimization by selecting from among alternative control choices. CORAL provides imperative constructs such as update, insert, and delete rules. Disk-resident data are supported using the EXODUS storage manager, which also provides transaction management in a client-server environment.

Implementation at Stanford started in 1985 on NAIL! (Not Another Implementation of Logic!). The effort led to the first paper on recursion using the magic set method (Bancilhon et al. 1986). Other contributions were aggregation in logical rules and theoretical contributions to negation: stratified negation by Van Gelder (1988); well-founded negation by Van Gelder, Ross, and Schlipf (1991); and

Table 2. Second-Stage Deductive Database Implementations.

<table>
<thead>
<tr>
<th>Name</th>
<th>MRPPS 3.0 (Minker 1978) (Maryland refutation proof procedure system)</th>
<th>DADM (Kellogg, Klahr, and Travis 1978) (Deductively augmented data management)</th>
<th>DEDUCE 2 (Chang 1978)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Organization</td>
<td>University of Maryland</td>
<td>System Development Corp.</td>
<td>IBM San Jose</td>
</tr>
<tr>
<td>Designers</td>
<td>Minker, McSkimin, Wilson, and Aronson</td>
<td>Kellogg, Klahr, and Travis</td>
<td>Chang</td>
</tr>
<tr>
<td>Computer</td>
<td>UNIVAC 1108</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Programming language</td>
<td>SIMPL</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Input language model</td>
<td>Multisorted, well-formed formulas</td>
<td>Primitive conditional statements and natural language</td>
<td>DEDUCE (Chang 1976)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(based on symbolic logic)</td>
</tr>
<tr>
<td>Intermediate language</td>
<td>Clausal form</td>
<td>Predicate array, premise array, semantic network, predicate connection graph (Sickel 1976; Kowalski 1975)</td>
<td>DEDUCE</td>
</tr>
<tr>
<td>Data structures</td>
<td>Semantic networks, knowledge base index</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Inference procedures</td>
<td>SL resolution (Kowalski and Kuehner 1971) and Lush resolution (Hill 1974)</td>
<td>Connection graph</td>
<td>Connection graph</td>
</tr>
<tr>
<td>Output language</td>
<td>Natural language voice and English (Powel 1977)</td>
<td>Primitive condition statements</td>
<td>DEDUCE</td>
</tr>
<tr>
<td>Features</td>
<td>Semantic query optimization, multisorted variables, no recursion, non-Horn clauses, clauses not necessarily function free, relations not in first normal form</td>
<td>Semantic query optimization, multisorted variables, no recursion</td>
<td>Semantic query optimization</td>
</tr>
</tbody>
</table>

SL = Linear resolution with Selection function.
LUSH = Linear resolution with Unrestricted Selection function for Horn clauses.
<table>
<thead>
<tr>
<th>Name</th>
<th>Developed</th>
<th>Recursion</th>
<th>Negation</th>
<th>Aggregation</th>
<th>Update</th>
<th>Integrity constraints</th>
<th>Optimization</th>
<th>Storage</th>
<th>Interfaces</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADITI (Vaghani et al. 1991)</td>
<td>U. Melbourne</td>
<td>General</td>
<td>Stratified</td>
<td>Stratified</td>
<td>No</td>
<td>No</td>
<td>Magic sets, semi-naive, database, intensional database</td>
<td>Prolog</td>
<td></td>
</tr>
<tr>
<td>COL (Abiteboul and Grumbac k 1991)</td>
<td>INRIA</td>
<td>?</td>
<td>Stratified</td>
<td>Stratified</td>
<td>No</td>
<td>No</td>
<td>None</td>
<td>Main memory</td>
<td>Machine learning</td>
</tr>
<tr>
<td>CONCEPT BASE (Jeusfeld and Staudt 1993)</td>
<td>U. Aachen</td>
<td>General</td>
<td>Locally Stratified</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Magic sets, semi-naive, database, intensional database</td>
<td>EDB only</td>
<td>C, Prolog</td>
</tr>
<tr>
<td>EKS-V1 (Viéville et al. 1992)</td>
<td>ECRC</td>
<td>General</td>
<td>Stratified</td>
<td>General</td>
<td>Yes</td>
<td>Yes</td>
<td>Query-subquery, left-right linear</td>
<td>Extensional database, intensional database</td>
<td>Persistent Prolog</td>
</tr>
<tr>
<td>DECLARE (Kiessling and Schmidt 1994)</td>
<td>MAD Intelligent Systems</td>
<td>General</td>
<td>Locally Stratified</td>
<td>General</td>
<td>No</td>
<td>No</td>
<td>Magic sets, semi-naive, projection pushing</td>
<td>EDB only</td>
<td>C, Common Lisp</td>
</tr>
<tr>
<td>LDL, LDL++, SALAD (Chimienti et al. 1990)</td>
<td>MCC</td>
<td>General</td>
<td>Stratified</td>
<td>Stratified</td>
<td>Yes</td>
<td>No</td>
<td>Magic sets, semi-naive, left-right linear, projection pushing</td>
<td>Extensional database, intensional database</td>
<td>None</td>
</tr>
<tr>
<td>EXTENDS database only</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>C, C++, SQL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOGRES (Cacace et al. 1990)</td>
<td>Polytech. of Milan</td>
<td>Linear</td>
<td>Inflationary semantics</td>
<td>Stratified</td>
<td>Yes</td>
<td>Yes</td>
<td>Seminaive, algebraic X forms</td>
<td>Extensional database, intensional database</td>
<td>INFORMIX</td>
</tr>
<tr>
<td>NAIL-GLUE (Morishita, Derr, and Phipps 1993)</td>
<td>Stanford U.</td>
<td>General</td>
<td>Well founded</td>
<td>Glue only</td>
<td>Glue only</td>
<td>No</td>
<td>Magic sets, semi-naive, right linear</td>
<td>EDB only</td>
<td>None</td>
</tr>
<tr>
<td>STARBURST T (Mumick et al. 1990)</td>
<td>IBM Almaden</td>
<td>General</td>
<td>Stratified</td>
<td>Stratified</td>
<td>No</td>
<td>No</td>
<td>Magic sets, semi-naive (variant)</td>
<td>Extensional database, intensional database</td>
<td>Extensible</td>
</tr>
</tbody>
</table>

Table 3. Existing Implementations of Deductive Databases (adapted from Ramakrishnan and Ullman [1995]).
modularly stratified negation by Ross (1990). A
language called GLUE (Morishita, Derr, and
Phipps 1993), developed for logical rules, has
the power of SQL statements as well as a con-
ventional language for the construction of loops, procedures, and modules.
Implementations of DDBs in the first, sec-
ond, and third stages of their development
have demonstrated the feasibility and practi-
cality of the technology. Tools and techniques
have been developed to produce efficient
DDBs.
Prospects for Commercial Implemen-
tation of Disjunctive Databases:
One might address why after 20 years of theo-
retical research in DDBs, no commercial sys-
tems exist. To place this statement in perspec-
tive, it is well to recall that approximately 12
years passed before relational systems were
available commercially. As Ullman has stated
on a number of occasions, DDB theory is more
subtle than relational database theory. Howev-
er, many prototypes have been developed start-
ing from the 1960s, as described in the previous
subsection. However, none of the systems in
Ramakrishnan and Ullman (1995) are likely to
become commercial products, with, possibly,
two exceptions: (1) ADITI (Ramamohanarao
1993) and (2) VALIDITY (Friesen et al. 1995; Ling,
Mendelzon, and Vielle 1995), developed at the
Bull Corporation. According to a personal com-
munication with Ramamohanarao, leader of
the ADITI effort, that I had in May 1996, they are
perhaps one year from having a commercial
product. In a communication that I received
from him on 20 February 1997, he stated: “We
have now completed most of the difficult parts
of the low-level implementation of ADITI. I am
very hopeful of getting the beta release of the
system by December 1997. The task was much
harder and time consuming than I have ever
anticipated.”

Whether it becomes a product remains to be
seen. I believe it will depend on moving the
system from a university setting to industry.
Implementers and application specialists,
rather than university researchers, are required.
At the Bull Corporation, Nicolas and Vielle
have headed an effort to develop the VALIDITY
DDB system that integrates object-oriented fea-
tures. In Minker (1996), I reported that the
VALIDITY DDB effort had been ongoing for about
four years. It appeared to be entering the mar-
ketplace and was being moved from the Bull
Corporation to a new company that will be
responsible for its maintenance, marketing,
and improvements. In a personal communica-
tion with Nicolas on 7 March 1997, he stated:
VALIDITY is now being further developed
and marketed by Next Century Media,
Inc., a California corporation in which
Groupe Bull has some equity interests. Its
principal office is in the San Francisco
area.
The VALIDITY DOOD software platform is
currently mainly used to develop NCM’s
products in electronic media for interac-
tive media applications. Two of these
products enable marketers to target their
advertising messages to household clus-
ters, to individual households, and to spe-
cific consumers, based on the user’s
expressed and implied interests and prefer-
ences, and to convert the data coming
from the user into a database of ongoing
and useful information about these cus-
tomers. A third product enables marketers
to measure the effectiveness of their
media plan and expenditures in a timely
manner, based on a full census of the
entire audience, rather than on samples
which are fraught with inherent biases
and errors.
No commercial systems exist for several rea-
sons. First, most prototypes were developed at
universities. Without commercial backing for
the venture, universities are not positioned to
either develop or support maintenance
required for large system developments. Sys-
tems developed in research organizations con-
trolled by consortia (ECRC and MCC) have not
had full backing of consortia members. Sec-
ond, implementation efforts to develop a com-
mercial product were vastly underestimated.
A large investment must be made to develop a
DDB that both competes and extends relation-
al technology. According to industry stan-
dards, an investment on the order of $30 to
$50 million is required to develop and place a
database system in the market, no matter what
technology it relies on. Furthermore, research-
ers tend to change their interests rather than
consolidate their work and invest in technol-
ogy transfer toward industry. Third, until
recently, no convincing demonstration has
been made of a large commercial problem that
requires a DDB, which might be why the MCC
and ECRC developments were terminated.
However, now, a large number of applications
could take advantage of this technology, as evi-
denced by the book by Ramakrishnan (1995)
and the applications being performed by the
VALIDITY deductive object-oriented database
(DDOD) system. In addition, Levy et al. (1995)
studied the problem of computing answers to
queries using materialized views and note that
this work is related to applications such as
... we should not abandon all research on theories of negation and alternative semantics, but we must take stock of what we have accomplished and make it more accessible for users.
global information systems, mobile computing, view adaptation, and the maintenance of physical data independence. Levy et al. (1996) describe how DDBs can be used to provide uniform access to a heterogeneous collection of more than 100 information sources on the World Wide Web. Fourth, apparently no university researchers have tried to obtain venture capital to build a product outside the university. Efforts by some from MCC to obtain venture capital did not succeed.

Does lack of a commercial system at this date forebode the end of logic and databases? I believe that such a view is naive. First, there still is a strong prospect, as noted previously, of commercial DDBs. Second, considering that it took 12 years before relational database technology entered the marketplace, there is no need for alarm. Third, as the following developments portend, relational databases are starting to incorporate techniques stemming from research in DDBs.

Indeed, many of the techniques introduced within DDBs are finding their way into relational technology. The new SQL standards for relational databases are beginning to adopt many of the powerful features of DDBs. In the SQL-2 standards (also known as SQL-92) (Melton and Simon 1993), a general class of integrity constraints called asserts allow for arbitrary relationships between tables and views to be declared. These constraints exist as separate statements in the database and are not attached to a particular table or view. This extension is powerful enough to express the types of integrity constraint generally associated with DDBs. However, only the full SQL-2 standard includes assert specifications. The intermediate SQL-2 standard, the basis for most current commercial implementations, does not include asserts. The relational language for the next-generation SQL, SQL3, currently provides an operation called recursive union that supports recursive processing of tables (Melton 1996). As noted in Melton (1996): “The use of the recursive union operator allows both linear (single-parent, or tree) recursion and nonlinear (multiparent, or general directed graph) recursion. This solution will allow easy solutions to bill-of-material and similar applications.”

Linear recursion is currently a part of the client server of IBM’s DB2 system. IBM is using the magic set method to perform linear recursion. Also, indications are that the ORACLE database system will support some form of recursion.

A further development is that SQO is beginning to be incorporated into relational databases. In DB2, cases are recognized when only one answer is to be found, and the search is terminated. In other systems, equalities and other arithmetic constraints are being added to optimize search. I believe it will not be long before join elimination is introduced into relational technology. One can now estimate when it will be useful to eliminate a join (Godfrey, Gryz, and Minker 1996). The tools and techniques already exist, and it is merely a matter of time before users and system implementers have them as part of their database systems.

Another technology available for commercial use is cooperative databases. The tools and techniques exist, as evidenced by COBASE (Chu, Chen, and Merzbacher 1994) and CARMIN (Gaasterland et al. 1992). With the introduction of recursion and SQO techniques into relational database technology, it will be necessary to provide users with cooperative responses so they understand why certain queries fail or succeed. It will also permit queries to be relaxed when the original query fails, permitting reasonable, if not logically correct, answers to be provided to users. Because user constraints can be handled in the same way that integrity constraints are handled, we will see relational systems that incorporate the needs of individual users into a query, as represented by their constraints.

Two significant developments have taken place in the implementation of commercial DDBs. First is the incorporation of techniques developed in DDDBs into relational technology. Recursive views that use the magic set technique for implementation are being permitted, and methods developed for SQO are being applied. Second is the development of a DOOD, VALIDITY, that is in commercial use as well as the development of the ADITY DDB that is scheduled to undergo beta testing in December 1997. It remains to be seen how long one can make patches to relational technology to simulate the capabilities of DDB systems.

Emerging Areas and Trends

In the previous sections, we discussed many theories and semantics for negation in both extended DDBs and DDBs. We understand great deal about negation, except for how and when to use a given theory, which will be an area of confusion when users want to apply the work. Much more work is needed if the areas of implementation and application are to catch up with the intellectual developments achieved over the past 20 years. The field is saturated with alternative theories of semantics, and work is needed on more fertile topics. Unless we do so, funding for logic and databases will wane, as I believe it has in the United
The role of logic will be of increasing importance because of the need to handle highly complex data. However, we should not abandon all research on theories of negation and alternative semantics, but we must take stock of what we have accomplished and make it more accessible for users.

The role of logic will be of increasing importance because of the need to handle highly complex data (partly as a result of the advances in networking technology and the reduction in the cost of both processing time and primary, secondary, and tertiary memory). These data will require more complex models of data access and representation. Advances will require formal models of logic rather than ad hoc solutions. Below, I briefly mention some fertile areas for further exploration. This listing of important areas to investigate is not intended to be exhaustive.

Temporal databases, which deal with time, are important for historical, real-time databases and other aspects of databases. Work in this area has been done by Snodgrass (1987), Comicki (1995), and Sistla and Wolfson (1995). A paper on applying transition rules to such databases for integrity constraint checking appears in Martin and Sistac (1996).

Transactions and updates need further attention. Semantic models of updates exist (Fernández, Grant, and Minker 1996) that assure views and data are updated correctly. Transactions that require sequences of updates, long-duration transaction models, and work-flow management are areas that require work. In emerging applications of database systems, transactions are viewed as sequences of nested, and most probably interactive, subtransactions that can sparsely occur over long periods of time. In this scenario, new complex transaction systems must be designed. Logic-based transaction systems will be essential to assure that an appropriate and correct transaction is achieved. See Minker (1996) for references.

Active databases consist of data that protect their own integrity and describe the database semantics. They are represented by the formalism event-condition-action (ECA) (Xerox Technologies 1989) and denote that whenever an event $E$ occurs, if condition $C$ holds, then trigger action $A$. It has a behavior of its own beyond passively accepting statements from users or applications. On recognition of certain events, it invokes commands and monitors and manages itself. It can invoke external actions that interact with systems outside the database and can activate a potentially infinite set of triggers. Although declarative constraints are provided, the ECA formalism is essentially procedural in nature. Zaniolo (1995) noted the need for declarative semantics of triggers. He has developed a unified semantics for active DDBs and has shown how active database rules relate to transaction-conscious stable model semantics. Baral and Lobo (1996) proposed a first step toward characterizing active databases. A clear semantics, sound implementations, and a better understanding of complexity issues are required in active databases. Work in the situation calculus and datalog extensions apply here.

Data mining and inductive inference deal with finding generalizations extracted from a database or a logic program. Generalizations can be integrity constraints that must be true with respect to the database or generalizations that might be true of the current state but might change if there are updates. Database administrators will need to determine which generalizations are integrity constraints and which apply only to the current state. SQO can handle either case and inform the user which constraint might be applicable to a query. As demonstrated in Muggleton and De Raedt (1994), logic programming can be used to form inductive inferences, and Knowledge Discovery in Databases (Piatetsky-Shapiro and Frawley 1991) covers work on knowledge data mining. Laurent and Vrain (1996) discuss how to couple DDBs and inductive logic programming to learn query rules for optimizing databases with update rules.

Integrating production systems with DDBs is needed to develop a formal approach to integrate and develop the semantics of rule-based systems. See Minker (1996) for references.

Logical foundations of DOODs is needed. No formal definition exists that covers all aspects of object-oriented databases. Efforts have been undertaken by Kifer and his coworkers (Kifer, Lausen, and Wu 1995) to develop a formal foundation for object-oriented databases. Work is required to develop techniques, a formal theory of updating, and all tools and techniques for DDBs.

Description logics restrict knowledge representation so that deduction is tractable but sufficiently powerful to represent knowledge naturally. See Minker (1996) for references to systems that incorporate description logics. In DDBs, representational power is also limited to allow for more tractable deduction. Some of these limits are a restriction to Horn clauses, no logical terms, no existential quantification, and so forth. Research in DDBs has sought to extend the representational power yet preserve tractability to the greatest extent possible. For example, DDBs allow for general clauses, and the addition of null values allows for a type of...
existing existential quantification. DDBs and description logics have remained distinct, but their goals are similar.

Heterogeneous databases integrate multiple databases into one system that do not necessarily share the same data models. There is the need for a common logic-based language for mediation and a formal semantics of such databases. Work on HERMES by Subrahmanian et al. (1994), on TSIMMIS by Chawathe et al. (1994), and by Ramakrishnan (Miller, Ioannidis, and Ramakrishnan 1994) and his colleagues illustrate the efforts in this area. Heterogeneous databases are also needed to handle textual data. Kero and Tsur (1996) describe the UN database that uses a DDB ++ to reason about textual information. Language extensions for the semantic integration of DDBs is proposed by Asirelli, Renso, and Turini (1996). The language allows mediators to be constructed, using a set of operators for composing programs and message-passing features.

Multimedia databases (Subrahmanian and Jajodia 1995) is an emerging area for which new data models are needed. These databases have special problems, such as the manipulation of geographic databases; picture retrieval where a concept orthogonal to time can appear in the database; space; and video databases, where space and time are combined. Temporal and spatial reasoning are needed. Logic will play a major role in the development of query languages for these new data models and will permit deductive reasoning, and a formal semantics will provide a firm theoretical basis for them.

Combining databases relates both to heterogeneous and multimedia systems. Here, one is trying to combine databases that share the same integrity constraints and schema. Such databases arise in distributed system work and the combining of knowledge bases. In addition to handling problems that arise because the combined databases might be inconsistent, one has to handle priorities that can exist among individual facts. A formal treatment and references appear in Pradhan and Minker (1995).

Integrity constraints, SQO, and constraint logic programming (CLP) are related topics. SQO uses constraints in the form of integrity constraints to prune the search space. These integrity constraints introduce equalities, inequalities, and relations into a query to help optimize search (Chakravarthy, Grant, and Minker 1990). CLP introduces domain constraints. These constraints might be equalities or inequalities and might even be relations (Jaffar and Maher 1994). Constraint databases and constraint-intensive queries are required in many advanced applications. Constraints can capture spatial and temporal behavior that is not possible in existing databases. Relationships between these areas need to be explored further and applied to DDBs. Spatial databases defined in terms of polynomial inequalities are investigated by Kuipers et al. (1996), who consider termination properties of datalog programs.

Abductive reasoning is the process of finding explanations for observations in a given theory. Given a set of sentences \( T \) (a theory) and a sentence \( G \) (an observation), the abductive task can be characterized as the problem of finding a set of sentences (abductive explanation for \( G \)) such that \( T \cup \Delta \models G \), \( T \cup \Delta \) is consistent, and \( \Delta \) is minimal with respect to set inclusion (Kakas, Kowalski, and Toni 1993).

A comprehensive survey and critical overview of the extension of logic programming to the performance of abductive reasoning (abductive logic programming) is given in Kakas, Kowalski, and Toni (1993). They outline the framework of abduction and its applications to default reasoning and introduce an augmentation-theoretic approach to the use of abduction as an interpretation for negation as failure. They show that abduction has strong links to extended disjunctive logic programming. Abduction is shown to generalize negation as failure to include not only negative but also positive hypotheses and to include general integrity constraints. They show that abductive logic programming is related to the justification-based truth maintenance system of Doyle (1979). Inoue and Sakama (1996) developed a fixpoint semantics for abductive logic programs in which the belief models are characterized as the fixpoint of a disjunctive program obtained by a suitable program transformation. For a summary of complexity results on abductive reasoning and nonmonotonic reasoning, see Cadoli and Schaerf (1993).

High-level robotics is an area of active research in which logic plays a significant role. Knowledge bases are used to solve problems in cognition required to plan actions for robots and deal with multiple agents in complicated environments. Work in deductive and disjunctive databases relates to this problem. In some instances, a robot can have several options that can be represented by disjunctions. Additional information derived from alternative information sources such as sensors can serve to disambiguate the possibilities. Universities engaged in this research are the University of Toronto (Lesperance et al. 1994), the University of Texas at El Paso (Baral, Gelfond, and
Logic and databases have helped the field of databases be a scientific endeavor rather than an ad hoc collection of techniques.

We understand what constitutes a database, a query, and an answer to a query and where knowledge has its place.

I discussed major accomplishments that have taken place in logic and databases during the 20 years since 1976. Among these accomplishments are the extension of relational databases, the development of the semantics and complexity of these alternative databases, the ability to permit knowledge base systems to be represented and developed, and the use of logic programming and DDBs to implement non-monotonic reasoning systems. I discussed many new areas that will be important in the near- and long-term future. It is clear that the field of logic and databases has a significant prehistory before 1970 and a well-defined area of research, complete with past achievements and continued future areas of work.

In the past 20 years, we have seen logic and databases progress from a fledgling field to a fully developed, mature field. The new areas that I cited that need further investigation show that we have not nearly exhausted the work in this field. I envision that many more workshops will be held on this topic. Logic and databases have helped the field of databases be a scientific endeavor rather than an ad hoc collection of techniques. We understand what constitutes a database, a query, and an answer to a query and where knowledge has its place.

I hope that I will have an opportunity, then, to look back and see a field that has accomplished much and is still vibrant. To remain vibrant, we will have to take on some of the new challenges rather than be mired in the semantics of more exotic databases. We will have to address implementation issues, and we will have to be able to provide guidance to practitioners who will need to use the significant developments in logic and databases.

Acknowledgments

This article is a condensation, and in some cases an expansion, of an invited keynote address presented at the Workshop on Logic in Databases in San Miniato, Italy, in 1996. Those interested in the longer version of the article should refer to Minker (1996). A number of my colleagues contributed their thoughts on what they considered to be the significant developments in the field, including Robert Demolombe, Hervé Gallaire, Georg Gottlob, John Grant, Larry Henschen, Bob Kowalski, Jean-Marie Nicolas, Raghu Ramakrishnan, Kotagiri Ramamohanarao, Ray Reiter, and Carlo Zaniolo. Many of my former and current students also contributed thoughts, including Sergio Alvarez, Chitta Baral, Jose Alberto Fernández, Terry Gaasterland, Parke Godfrey, Jarek Gryz, Jorge Lobo, Sean Luke, and Carolina Ruiz. Although many of the views reflected in this article might be shared by those who made suggestions, I take full responsibility for them. The full paper is dedicated in honor of Hervé Gallaire and Jean-Marie Nicolas with whom I have worked as co-author, co-editor, colleague, and friend and who have helped to make the field of deductive databases a reality. Support for this article was received from the National Science Foundation under grant IRI 9300691.
Note
1. This subsection reflects comments made at the Workshop on Logic in Databases in San Miniato, Italy, 1 to 2 July, 1996, in the panel session, Deductive Databases: Challenges, Opportunities, and Future Directions, by Arno Sebes, Shalom Tsur, Jeff Ullman, Laurent Viellie, and Carlo Zaniolo, and in a personal communication by Jean-Marie Nicolas. I am wholly responsible for the views expressed in this subsection.

References


Articles


Jack Minker is a professor of computer science in the Department of Computer Science and the Institute for Advanced Computer Studies at the University of Maryland. His research areas are deductive databases, logic programming, AI, and nonmonotonic reasoning. He was the first chairman of the Department of Computer Science at the University of Maryland from 1974 to 1979 and chairman of the Advisory Committee on Computing at the National Science Foundation from 1979 to 1982. In 1985, Minker received the Association for Computing Machinery (ACM) Outstanding Contribution Award for his work in human rights. He is a fellow of the American Association for the Advancement of Science, a founding fellow of the American Association for Artificial Intelligence, a fellow of the Institute of Electrical and Electronics Engineers, and a founding fellow of the ACM. He received the University of Maryland Presidential Medal for 1996 and is a distinguished scholar-teacher for 1997 to 1998. His e-mail address is minker@cs.umd.edu. Subject: IEA/AIE-98 in Cooperation with AAAI

---

**Call for Papers (IEA/AIE-98)**

**11th International Conference on Industrial & Engineering Applications of Artificial Intelligence & Expert Systems**

Benicassim, Castellon, Spain

June 1–4, 1998

*Sponsored by:* International Society of Applied Intelligence; Universitat Jaume-I de Castellon; Universidad Nacional de Educacion a Distancia (UNED), Madrid

*In Cooperation with:* AAAI, ACM/SIGART, CSCSI, ECCAI, IEE, INNS, JSAI, SWT

IEA/AIE-98 will focus on methodological as well as practical aspects in the development of KBS’s, knowledge modeling, hybrid techniques that integrate the symbolic and connectionistic perspectives in the industrial application of AI, and application of intelligent systems’ technology to solve real life problems. Accepted papers, either as oral presentations or as poster panels, will be published at full length in the proceedings. Selected papers will be published in the *International Journal of Applied Intelligence*.

Authors are invited to submit by November 7, 1997, five copies of papers, double spaced, written in English, of up to 10 pages, including figures, tables, and references. The format should be A4 or 8 1/2 X 12 paper, in a Roman font, 12 point in size, without page numbers, and a printing area of 15.3 X 24.2cm2 (6.0 X 9.5 sq. in.). If possible, please make use of the latex/plaintex style file available in our WWW site. In addition, one sheet must be attached including: title, authors’ names, a list of five keywords, the topic under which the paper best fits, the preferred presentation (oral or poster), and the corresponding author information (name, postal and e-mail address, phone and fax numbers). This page must also be sent by e-mail to iea98@titan.inf.uji.es before November 7, 1997.

Contributions must be sent to the Program Co-chair Prof. Angel P. del Pobil at the address below. Conference information can be obtained from the General Chair Prof. Moonis Ali, the Program Co-Chair Prof. del Pobil, and through our web site.

Moonis Ali
General Chair, IEA/AIE-98
Dept. of Computer Science
Southwest Texas State University
San Marcos, TX 78666-4616 USA
Phone: +1 (512) 245-3409
FAX: +1 (512) 245-8750
E-mail: ma04@swt.edu

Angel P. del Pobil
Program Co-chair, IEA/AIE-98
IEA/AIE-98 Sec., Informatics Dept., Jaume-I Univ., Campus de Penyeta Roja E-12071 Castellon, Spain
Phone: +34 64-345.642
FAX: +34 64-345.848
E-mail: iea98@titan.inf.uji.es
http://titan.inf.uji.es/iea98/