Logic and Databases: A Deductive Approach

HERVÉ GALLAIRE

Compagnie Générale d'Électricité Laboratoire de Marcoussis, Marcoussis, France

JACK MINKER

University of Maryland, Computer Science Department, College Park, Maryland

JEAN-MARIE NICOLAS

ONERA-CERT, Département d'Informatique, Toulouse, France

The purpose of this paper is to show that logic provides a convenient formalism for studying classical database problems. There are two main parts to the paper, devoted respectively to conventional databases and deductive databases. In the first part, we focus on query languages, integrity modeling and maintenance, query optimization, and data dependencies. The second part deals mainly with the representation and manipulation of deduced facts and incomplete information.

Categories and Subject Descriptors: H.2.1 [**Database Management**]: Logical Design—*data models*; H.2.3 [**Database Management**]: Languages—*query languages*; H.2.4 [**Database Management**]: Systems—*query processing*

General Terms: Deductive Databases, Indefinite Data, Logic and Databases, Null Values, Relational Databases

INTRODUCTION

As emphasized by Codd [1982], theoretical database studies form a fundamental basis for the development of homogeneous and sound database management systems (DBMS), which offer sophisticated capabilities for data handling. A comprehensive study of the many problems that exist in databases requires a precise formalization so that detailed analyses can be carried out and satisfactory solutions can be obtained. Most of the formal database studies that are under way at present are concerned with the relational data model introduced by Codd [1970], and use either a specially developed database theory [Maier 1983; Ullman 1982] or other formal theories such as mathematical logic as their framework. The purpose of this paper is to provide an overview and a survey of a subfield of logic as it is applied to databases. We are mostly concerned with the application of logic to databases, where logic may be used both as an inference system and as a representation language; we primarily consider relational type databases. Some important efforts in the application of other aspects of logic theory to databases (e.g., see Maier [1983], Ullman [1982], and the references provided there) or those that deal with nonrelational (i.e., hierarchical and network) databases (e.g., see Jacobs [1982] and Jacobs et al. [1982]) are not covered here.

The use of logic for knowledge representation and manipulation is primarily due to the work of Green [1969]. His work was the basis of various studies that led to so-called

Current address of Hervé Gallaire and Jean-Marie Nicolas: European Computer-Industry Research Centre (ECRC), Arabellastrasse 17, D-8000 München 81, FRG.

CONTENTS

question-answering systems, which are concerned mainly with a highly deductive manipulation of a small set of facts, and thus require an inferential mechanism provided by logic. Similar techniques have been adapted to databases to handle large sets of facts, negative information, open queries, and other specific database topics. These techniques have given rise to what is called deductive databases. However, the use of logic to study databases is not restricted to providing deductive capabilities in a DBMS; the pioneering work of Kuhns [1967, 1970] also uses logic for conventional databases to characterize answers to queries.

Aside from the introduction and conclusion to this paper, there are two main sections, which are devoted respectively to conventional databases and deductive databases. In this introduction we provide background material to familiarize the reader with the terminology used throughout the paper, introducing the reader to concepts in relational databases, to the area of mathematical logic, and to the basic relationships between logic and databases. Section 1 is an extended and revised version of material that appeared in Gallaire [1981]. This material shows how logic provides a formalism for databases, and how this formalism has been applied to conventional databases. Its use in query languages, integrity modeling and maintenance, query evaluation, and database schema analysis is described. In Section 2 we show how logic extends relational databases to permit deduction and describe how logic provides a sound basis for a proper treatment and understanding of null values and incomplete information.

In the remainder of this introduction, we first describe the main concepts of the relational data model. Following this, we specify what is meant by mathematical logic, focusing on logic relevant to databases rather than logic in general. Finally, we briefly introduce two ways in which databases can be considered from the viewpoint of logic.

I.1 Relational Model

To define a relational model we need some concepts. A *domain* is a usually finite set of values. The *Cartesian product* of domains $D_1, \ldots, D_n$ is denoted by $D_1 \times \cdots \times D_n$ and is the set of all tuples $(x_1, \ldots, x_n)$ such that for any i, $i = 1, \ldots, n$ $(x_1 \in D_1)$. A *relation* is any subset of the Cartesian product of one or more domains. A database (instance) is a finite set of finite relations. By a finite relation we mean that the extension of the relation (i.e., the totality of all tuples that can appear in a relation) is finite. The arity of a relation $R \subseteq D_1 \times \cdots \times D_n$ is n. One may envision a relation to be a table of values. Names are generally associated with the columns of these tables; these names are called *attributes*. Values of an attribute associated with column i of a relation are taken from domain D_i. A relation R with attributes $A_1, \ldots, A_n$ defines a relation scheme denoted as $R(A_1, \ldots, A_n)$, whereas the specific relation R (i.e., the relation with specified tuples) is said to be an *instance* or *extension* of the relation scheme.

Not all instances of a relation scheme have meaningful interpretations; that is, they do not correspond to valid sets of data according to the intended semantics of the database. One therefore introduces a set of constraints, referred to as *integrity constraints*, associated with a relation scheme to ensure that the database meets the intended semantics. Integrity constraints may involve interrelationships between relations.

To summarize, a database scheme consists of a collection of relation schemes together with a set of integrity constraints. A database instance, also called a database state, is a collection of relation instances, one for each relation in the database scheme. A database state is said to be valid if all relation instances that it contains obey the integrity constraints. In this paper, values in database/relation instances are referred to as elements, constants, or individuals, depending on the context.

To manipulate data in a relational database, a language is introduced. One may introduce an algebraic language based on algebraic operators, or a calculus language, which we discuss in the following section. In an algebraic language we need only two operators for our purposes: the project and join operators. Given a relation R, and X a set of attributes of R, then the *projection* of R on X is $\{s[X] \mid s \in R\}$, where $s[X]$ is a tuple constructed from s by keeping all and only those components that belong to attributes in X. Given two relations R and S, the *natural join* $R * S$ is formed by computing the Cartesian product, $R \times S$, selecting out all tuples whose values on each attribute common to R and S coincide, and projecting one occurrence of each of the common attributes. For a more thorough presentation of the relational model, the reader is referred to Date [1977, 1981] and Ullman [1982]. See also Delobel [1980] and Maier [1983] for an overview and a survey of relational database theory.

I.2 Mathematical Logic

As is true for any formal system, mathematical logic relies upon an object language, a semantics or interpretation of formulas in that language, and a proof theory.

As the *object language* we shall use a first-order language such as that of the first-order predicate calculus. Primitive symbols of such a language are (1) parentheses, (2) variables, constants, functions, and predicate symbols, (3) the usual logical connectors, $\neg$ (not), & (and), $\vee$ (or), $\rightarrow$ (implication), $\leftrightarrow$ (equivalence), and (4) quantifiers, $\forall$ (for all), $\exists$ (there exists). Throughout the paper we use lowercase letters from the start of the alphabet to represent constants ($a, b, c, \ldots$), those from the end of the alphabet to represent variables (u, v, w, x, y, z), and letters such as ($f, g, h, \ldots$) to denote functions.

A *term* is defined recursively to be a constant or a variable, or if f is an n-ary function and $t_1, \ldots, t_n$ are terms, then $f(t_1, \ldots, t_n)$ is a term. There are no other terms. We usually assume that a term in the context of databases is *function free*; that is, it is either a constant or a variable.

If P is an n-ary predicate symbol and $t_1, \ldots, t_n$ are terms, then $P(t_1, \ldots, t_n)$ is an *atomic formula*. An atomic formula or its negation is a *literal*. *Well-formed formulas* (wffs) are defined recursively as follows. An atomic formula is a wff. If w_1 and w_2 are wffs, then $\neg(w_1)$, $(w_1) \vee (w_2)$, (w_1) & (w_2), $(w_1) \rightarrow (w_2)$, and $(w_1) \leftrightarrow (w_2)$ are wffs. A *closed* wff is one that does not contain any *free* variable (i.e., it contains only quantified variables and constants).

In dealing with wffs it is sometimes convenient to place them in a normal form. A wff is in *prenex normal form* if all quantifiers appear in front of the formula. The wff corresponding to the statement "Every teacher has a diploma" is

$$(1) \quad (\forall x\ \forall y)(\mathrm{TEACH}(x, y) \rightarrow (\exists z)\mathrm{DIPLOMA}(x, z)).$$

It is indeed possible to place all quantifiers in front of the formula to achieve the prenex normal form (see Chang and Lee [1973] for details). When this is done, formula (1) becomes formula (2):

$$(2) \quad (\forall x)(\forall y)(\exists z)(\neg\mathrm{TEACH}(x, y) \vee \mathrm{DIPLOMA}(x, z)).$$

Similarly, the prenex normal form of the wff

$$(3) \quad (\forall x)(\forall y)(((\exists z)(P(x, z)\ \&\ P(y, z))) \rightarrow (\exists u)Q(x, y, u))$$

is

(4) $(\forall x)(\forall y)(\forall z)(\exists u)$
$$(\neg P(x, z) \vee \neg P(y, z) \vee Q(x, y, u)).$$

A prenex formula is in *Skolem normal form* when all existential quantifiers are eliminated by replacing variables they quantify with arbitrary functions of all universally quantified variables that precede them in the formula. These functions are called Skolem functions; a Skolem function of 0 arguments is called a Skolem constant. A *clause* is a disjunction of literals, all of whose variables are implicitly universally quantified. The Skolem normal form of (2) is

(5) $$\forall x\ \forall y(\neg \mathrm{TEACH}(x, y) \vee \mathrm{DIPLOMA}(x, f(x, y)),$$

where the existentially quantified variables are eliminated and replaced by Skolem functions. Similarly, the Skolem normal form of (4) is

(6) $$(\forall x)(\forall y)(\forall z)(\neg P(x, z) \vee \neg P(y, z) \vee Q(x, y, g(x, y, z))),$$

where the existentially quantified variable u has been eliminated and replaced by the Skolem function $g(x, y, z)$. When a wff is in Skolem normal form, all the quantifiers remaining in the front of the formula may be eliminated since all variables that remain are, by convention, universally quantified. Formula (5), above, may be replaced by

(7) $$\neg \mathrm{TEACH}(x, y) \vee \mathrm{DIPLOMA}(x, f(x, y)).$$

Thus

$$\neg A_1 \vee \cdots \vee \neg A_m \vee B_1 \vee \cdots \vee B_n,$$

where the A_1 and the B_j are positive literals, is a clause. We shall write a clause in an equivalent form as

$$A_1 \ \& \cdots \& \ A_m \rightarrow B_1 \vee \cdots \vee B_n.$$

In a clause, whenever n is equal to 0 or 1, the clause is said to be a *Horn clause.* If both m and n are equal to 0, there are no atoms on the left- or right-hand side of the implication sign, and the clause is called the *empty clause.* A clause (a literal) in which no variables appear is called a *ground clause* (*ground literal*). Every closed, well-formed formula may be placed in clause form. We note that the transformation of a wff into prenex normal form preserves equivalence, but this is not the case for transformations into Skolem or clause form. The latter transformations only preserve satisfiability, which is sufficient for provability purposes.

Two complementary aspects of wffs are of interest. One deals with *semantics* (or *model theory*), the specification of truth values to wffs, whereas the other deals with *proof theory*, the derivation of a wff from a given set of wffs.

1.2.1 Semantics: Model and Interpretation

In semantics we are concerned with interpretations, where an *interpretation* of a set of wffs consists of the specification of a nonempty set (or domain) E, from which constants and variables are given values. Each n-ary function symbol is assigned a function from E^n to E. Each n-ary predicate is assigned a relation on E^n.

In an interpretation with domain E, a closed wff is either true or false, whereas a (open) wff with n ($n \geq 1$) free variables determines a set of n-tuples (i.e., a relation) on E^n. Each of these n-tuples is such that when its components are substituted for the corresponding free variables in the open wff, then in this interpretation, the closed wff that is obtained is true. If the set of n-tuples is empty, then the open wff is said to be false, and if the set of n-tuples coincides with E^n, then the open wff is said to be true. Broadly, the truth value of a closed wff is obtained as follows. If R is the relation assigned to a n-place predicate symbol P, then $P(e_1, \ldots, e_n)$ evaluates to true if $\langle e_1, \ldots, e_n \rangle \in R$; otherwise it evaluates to false. Now, if w_1 and w_2 are closed wffs, $\neg w_1$ evaluates to true if w_1 is false; otherwise it evaluates to false. $w_1 \ \& \ w_2$ evaluates to true if both w_1 and w_2 are true; otherwise it evaluates to false. $w_1 \rightarrow w_2$ evaluates to true if either w_1 is false or w_2 is true; otherwise it evaluates to false. Well-formed formulas constructed using the other logical symbols may be evaluated similarly. Finally, if x is a variable in w, $\forall x w(x)$ (respectively, $\exists x w(x)$) evaluates to true if for all elements e_1 in E (respectively, there is an element $e_1 \in E$ such that) $w(e_1)$ is true; otherwise it evaluates to false.

A *model* of a set of wffs is an interpretation in which all wffs in the set are true. A wff w is said to be a *logical consequence* of a set of wffs W iff w is true in all models of W. This is denoted by $W \models w$.

I.2.2 Syntax: First-Order Theory

The first-order predicate calculus is a formal system that has as object language a first-order language, a set of axiom schemas (the logical axioms), and two inference rules: *modus ponens* and *generalization.*

When other wffs are added as axioms, the formal system that is obtained is called a *first*-order theory. The new axioms are called *nonlogical* or *(proper)* axioms. A first-order theory is essentially characterized by its nonlogical axioms. A set of nonlogical axioms could be, for example,

$$\text{Man(Turing)},$$

$$(\forall x)(\text{Man}(x) \rightarrow \text{Mortal}(x)).$$

A *model* of a theory is an interpretation in which all axioms are true; logical axioms are, in fact, chosen to be true in all interpretations. For the above theory, setting

$$\text{Man(Turing)} = \text{True},$$

$$\text{Mortal(Turing)} = \text{True}$$

yields a model since it makes all statements in the above theory true. A wff w is *derivable* from a set of wffs W in a theory $T(W \vdash w)$ iff w is deducible from W and from the axioms of T by a finite application of the inference rules.

Using the inference rule of modus ponens, which states that from p and $p \rightarrow q$ one can conclude q, we obtain from the above theory the derived result: Mortal(Turing). If W is empty, then w is a *theorem* of T ($\vdash_T w$, or equivalently $T \vdash w$). Whenever T is clear, we shall write $W \vdash w$ for $W \vdash_T w$.

Inference rules other than modus ponens and generalization can be used to derive theorems; in fact most theorem-proving techniques are based on the inference rule termed the Robinson Resolution Principle [Robinson 1965], which applies to wffs in clausal form.

The Robinson Resolution Principle is a rule of inference that permits a new clause to be derived from two given clauses; further, the derived clause is satisfiable (i.e., has a model) if the two given clauses are satisfiable. In the context of databases assumed to be function free, the principle can be described in terms of the following example. From

C1: $\neg P(a, b, c) \vee Q(d, e)$, and

C2: $P(x, y, z) \vee R(x, y)$,

one obtains the derived clause

C3: $Q(d, e) \vee R(a, b)$.

The clause C3 is found by considering the literals in the two clauses that have the same predicate name, but one is negated and the other is not. The only predicate of this type is P. One then determines if the two atoms $\{P(a, b, c), P(x, y, z)\}$ can be made identical by some substitution to the variables, where a, b, and c are assumed to be constants and x, y, and z are assumed to be variables. The substitution $\{a/x, b/y, c/z\}$ is such a substitution, and is to be read: Substitute a for x, b for y, and c for z. One then eliminates the two literals made identical by the substitution (said to be unified) from each clause, forms the disjunction of the remaining literals in the two clauses, and applies the unifying substitution to the remaining literals to obtain the derived clause. Thus in this example the clause C3 is derived.

The resolution principle is used mostly to carry out refutation proofs: In order to prove $W \vdash w$, one tries to show that W and $\neg w$ are not simultaneously satisfiable. As resolution preserves satisfiability, if one can, by resolution from the clausal forms of W and $\neg w$, derive the empty clause, then W and $\neg w$ cannot simultaneously be satisfiable. For example, $\neg P(a, b, c)$ and

$P(x, y, z)$, where x, y, and z are variables, would resolve to yield the empty clause.

A resolution proof consists of applying resolution to all clauses in the original set, adding those newly derived clauses to the set, and iterating the process.

The most important relationships between the semantic and the syntactic approaches are soundness and completeness. An inference system is *sound* iff for all W and w, whenever $W \vdash w$, it implies that $W \models w$; it is *complete* iff for all W and w, whenever $W \models w$, it implies that $W \vdash w$. The inference rules of modus ponens and generalization are complete and sound for the propositional calculus. Similarly, resolution refutation is complete and sound for first-order theories: The empty clause is derived if and only if the initial clause (which is negated to apply resolution) is a theorem in the theory. However, there is an element of undecidability; if the clause proposed to be proved is not a theorem, the inference process may never terminate. Resolution refutation is also complete and sound. The meaning of completeness and soundness is that the same results are obtained by using semantics, which deals with truth assignments, and provability, which deals with inference rules.

The reader should refer to Enderton [1972] and Mendelson [1978] for general background on logic and to Chang and Lee [1973] and Loveland [1978] for more material on the resolution principle.

I.3 Databases Viewed through Logic

Before considering the formalization of databases in terms of logic, we shall mention some assumptions that govern query (and integrity constraint) evaluation of databases. On the one hand, these assumptions express a certain implicit representation of negative facts (e.g., "Paul is not the father of Peter:" ¬Father(Paul, Peter)) and, on the other hand, they make precise the universe of reference to which queries refer. There are three such assumptions:

(1) The *closed world assumption* (CWA), also called convention for negative information, which states that facts not known to be true are assumed to be false (i.e., $\neg R(e_1, \ldots, e_n)$ is assumed to be true iff the tuple $\langle e_1, \ldots, e_n \rangle$ fails to be found in relation R).

(2) The *unique name assumption*, which states that individuals with different names are different.

(3) The *domain closure assumption*, which states that there are no other individuals than those in the database.

Answers to "For all" queries or queries involving negation are obtained by using the above hypotheses. For example, the query, "Who is not a Full Professor?", addressed to a database whose current state consists of

Full-Prof.(Jean),
Full-Prof.(Paul),
Associate-Prof.(Andre),
Assistant-Prof.(Pierre),

will get as an answer {Pierre, Andre}. Indeed, the domain closure assumption restricts the individuals to be considered to the set {Jean, Paul, Pierre, Andre}. Furthermore, according to the unique name assumption, one gets the following: Pierre $\neq$ Jean, Pierre $\neq$ Paul. Consequently, Pierre $\notin$ Full-Prof., which, according to the closed world assumption, leads to ¬Full-Prof.(Pierre). The second element of the answer, ¬Full-Prof.(Andre), is obtained in a similar way.

We note that one way to avoid calling for the domain closure assumption is to consider as acceptable queries (and integrity constraints) only expressions that restrict their own reference domain. This is the case for any expression of the relational algebra and for the so-called class of definite (or safe, or range-restricted) logical formulas (see Section 1.1.1).

Although the query evaluation process in any DBMS (implicitly) works under the above hypotheses, these assumptions were made explicit and clearly understood only through a logical formalization of databases.

As first characterized by Nicolas and Gallaire [1978], a database can be considered from the viewpoint of logic in two different ways: either as an *interpretation* (of a first-order theory) or as a (first-order) *theory*. When considered from the viewpoint of interpretations, queries (and integrity constraints) are formulas that are to be evaluated on the interpretation using the semantic definition of truth. From the viewpoint of a theory, queries and integrity constraints are considered to be theorems that are to be proved. The interpretation viewpoint and the theory viewpoint respectively formalize the concepts of conventional and deductive databases.

Reiter [1984] and Kowalski [1981a] have investigated these two approaches more thoroughly. Reiter refers to the two approaches as the *model-theoretic view* and the *proof-theoretic view*, respectively, whereas Kowalski refers to them as the *relational structure view* and the *logic database view*.

The three terms "interpretation," "model," and "relational structure" are closely related. A model is an interpretation that makes all axioms true. The "relational structure" view means that queries are evaluated by assuming the database entries to be true. All these terms relate to the semantic definition of truth. We shall use the term "model-theoretic view" for these three terms throughout this paper. The three terms "theory," "proof theoretic," and "logic database" connote that, in order to determine answers to queries, one derives data from axioms. We use the term "proof theoretic" for these terms throughout the paper.

Both Kowalski and Reiter have shown that, although conventional databases are generally considered from a model-theoretic view, they can also be considered from the proof-theoretic view and can thus be considered as a particular logic database.

This section provides an intuitive characterization of these two views of a database through the perspective of logic. Further details are provided in Section 2 of this paper for the proof-theoretic view.

Let DB be an instance of a relational database. Then DB consists of a set of relations (i.e., a relation R for each relation schema $R(A_1, \ldots, A_n)$) and a set of integrity constraints, IC. Let D be the union of the underlying domains of all attributes that occur in the relation schema. Now define a first-order language L to consist of an n-place predicate symbol $\mathbf{R}$ for each n-ary relation in DB and a set of constants, one for each element in D; the language is assumed to have no function symbols. DB can be seen as an interpretation of formulas of the language as defined in Section I.2, and the formulas of L can be evaluated in this interpretation as follows: Variables range over the domain D, and $\mathbf{R}(e_1, \ldots, e_n)$ is true iff $\langle e_1, \ldots, e_n \rangle \in R$. The language can be extended to include arithmetic comparison operators ($<, =, >, \leq, \geq$) as particular predicate symbols, which are assigned their usual interpretation.

If the integrity constraints in IC are expressed as formulas of L, then the database DB will be a valid database state iff every constraint in IC evaluates to true in DB, that is, iff DB is a model of IC. We note that, according to the very definition of an interpretation, the evaluation of logical formulas (on an interpretation) is done in accord with the closed world, unique name, and domain closure assumptions stated at the beginning of this section.

The above constitutes a description of the model-theoretic view of a database. The proof-theoretic view of DB is obtained by constructing a theory T that admits DB as a unique model. Then for any wff w in L, $T \vdash w$ iff w is true in DB.

The process of defining T consists of making its (proper) axioms precise. The axioms (see Reiter [1984]) are of three kinds:

(1) *Assertions.* For any relation R in DB and any tuple $\langle e_1, \ldots, e_n \rangle \in R$, an axiom $\mathbf{R}(e_1, \ldots, e_n) \in T$.

(2) *Particularization Axioms.* Particularization axioms explicitly state the evaluation hypotheses that, in the model-theoretic view, are conveyed by the notion of interpretation:

(i) *The completion axioms.* There is one such axiom for any relation R in DB. If $\langle e_1^1, \ldots, e_1^n \rangle$, …, $\langle e_p^1, \ldots, e_p^n \rangle$ are all the tuples in R, it is written as

$$\forall x_1 \cdots \forall x_n (R(x_1, \ldots, x_n) \rightarrow (x_1 = e_1^1 \;\&\; \cdots \;\&\; x_n = e_1^n) \vee \cdots \vee (x_1 = e_p^1 \;\&\; \cdots \;\&\; x = e_p^n)).$$

The completion axiom effectively states that the only values tuples that the relation R can have are

$$\langle e_1^1, \ldots, e_1^n \rangle, \ldots, \langle e_p^1, \ldots, e_p^n \rangle.$$

(ii) *The unique name axioms.* If $e_1, \ldots, e_q$ are all the individuals in DB, the unique name axioms are

$$(e_1 \neq e_2), \ldots, (e_1 \neq e_q),$$
$$(e_2 \neq e_3), \ldots, (e_{q-1} \neq e_q).$$

(iii) *The domain closure axiom.* This is

$$\forall x((x = e_1) \vee (x = e_2) \vee \cdots \vee (x = e_q)).$$

(3) *Equality Axioms.* Equality axioms are needed since the axioms in (2) involve the equality predicate. These axioms specify the usual properties of equality:

- *reflexivity:*

$$\forall x(x = x).$$

- *symmetry:*

$$\forall x \, \forall y((x = y) \rightarrow (y = x)).$$

- *transitivity:*

$$\forall x \, \forall y \, \forall z((x = y) \,\&\, (y = z) \rightarrow (x = z)).$$

- *principle of substitution of equal terms:*

$$\forall x_1 \cdots \forall y_n(P(x_1, \ldots, x_n)$$
$$\& \, (x_1 = y_1) \,\&\, \cdots \,\&\, (x_n = y_n)$$
$$\rightarrow P(y_1, \ldots, y_n)).$$

Let us briefly give the underlying reason why T admits DB as a unique model (up to an isomorphism). The only interpretation of T in which the domain closure axioms and the unique name axioms are all satisfied is such that their individuals are in one-to-one correspondence with elements in D. Thus all the possible models of T have the same domain as DB (up to an isomorphism). For any such model M, since its domain is fixed, in order to be different from DB, it must assign to at least one **R** a relation R' different from R. But this is not possible. Indeed, if a tuple $\langle e_1, \ldots, e_n \rangle$ belongs to R but not to R', then M does not satisfy one of the axioms in (1). Conversely, if $\langle e_1, \ldots, e_n \rangle$ belongs to R' but not to R, then M does not obey the completion axiom associated with **R**.

As defined above T provides the proof-theoretic view of DB. According to this view, DB satisfies a constraint w in IC iff $T \vdash w$. Furthermore, the answer to a query formulated as $W(x_1, \ldots, x_p)$—where $x_1, \ldots, x_p$ are the free variables in the formula W—consists of those p-tuples $\langle e_1, \ldots, e_p \rangle$ such that $T \vdash W(e_1, \ldots, e_p)$.

It is worth noting that, although according to this view query (and integrity constraint) evaluation calls for proof techniques, DB is and remains a conventional (i.e., nondeductive) database. No other (positive) facts than those explicitly stated in (1) can be derived from T.

At this stage one may notice that the above proof-theoretic view is not intended to be used directly as a basis for a DBMS implementation. The combinatorial complexity of the particularization axioms would lead to inefficient systems, but, as emphasized by Reiter [1984], the value of this view is found in the generalizations that it suggests for databases: (1) Add some disjunctive facts or existentially quantified literals among the assertions and one obtains a database with null values and incomplete information; (2) suppress from the set of ICs some of its formulas and add those formulas as axioms to the theory and one obtains a new theory that is a deductive database. However, the formulation of the completion axioms then has to be reconsidered, as is seen in Section 2. Except for work in deductive databases, the applications of logic to databases has mainly referred, either explicitly or implicitly, to the model-theoretic view. This work is reported upon in Section 1.

1. CONVENTIONAL DATABASES

The goal of Section 1 is to show how logic can provide formal support to study classical database problems, and in some cases, how logic can go further, helping to comprehend and then to solve them. We describe contributions published in the literature that relate to logic and databases with respect to query languages, integrity modeling and maintenance, query evaluation, and database schema analysis.

1.1 Query Languages

1.1.1 Toward Relational Calculus

One of the first impacts of logic on databases was the use of its language as a basis for defining assertional query languages. This can be done in any one of four ways [Pirotte 1976, 1978], depending on whether one-sorted or many-sorted languages [Enderton 1972] are used and whether tuples of relations or elements of domains are considered as primitive objects [Ullman 1980]. In fact, only two out of these four possibilities have been truly exploited: the so-called *domain relational calculus* (DRC, one sorted/elements of domains) and the *tuple relational calculus* (TRC, one sorted/tuples of relations). Both of these languages have the same expressive power [Ullman 1980]. The respective pioneering and fundamental efforts of Kuhns [1967] and Codd [1972] are shown in each of these two cases. We shall focus on the DRC.

As described in Section I.3, the reason for considering the language of logic as a basis for defining query languages is that a relational database (instance) can be viewed as an interpretation of a first-order language. Thus the answer to a formula $W(x_1, \ldots, x_p)$, where $x_1, \ldots, x_p$ are the free variables in W, considered as a query, is the set of p-tuples $\langle e_1, \ldots, e_p \rangle \in D^p$ such that $W(e_1, \ldots, e_p)$ is true. However, when considered as queries, some formulas may be "unreasonable" [Kuhns 1967], since their answer may be different in two database states, where the relations they refer to are the same. A typical example is the formula $\neg\mathbf{R}(x_1, \ldots, x_n)$, which characterizes all tuples in D^n except those that are in R as opposed to $P(x_1, \ldots, x_n)$ & $\neg\mathbf{R}(x_1, \ldots, x_n)$.

A semantic characterization of formulas that can be considered to be reasonable queries led to the notions of *definite* formulas [Kuhns 1967] and *safe* formulas [Ullman 1980]. Roughly, such formulas are domain independent since they self-restrict the range of the variables that they contain.[1] However, once these classes were defined, a new problem appeared, that of finding machine-recognizable criteria for determining whether a given formula is definite (or safe). Unfortunately, as proved by Di Paola [1969] and Vardi [1981], the decision problem for definite formulas is recursively unsolvable. Thus what remained to be done was to look for the largest of its subclasses that was recursive. This was one of the motivations for various authors who introduced purely syntactically defined subclasses of definite formulas such as "proper" [Kuhns 1970], range separable [Codd 1972], acceptable [Artaud and Nocolas 1974], range restricted [Nicolas 1979a, 1979b], and evaluable [Demolombe 1982] formulas.

Using many-sorted logic as a basis for defining query languages while considering elements of domains as primitive objects was exploited by Pirotte [1978] (see also Minker [1978b] and Reiter [1978b]). In such a case each sort is assigned to a database domain, and the well formedness of formulas is checked with regard to sort requirements. Languages obtained in this manner have the same expressive power as the preceding ones.

Many-sorted languages offer a more "precise" definition of the model, but they freeze to some extent any evolution of the application. In that respect one should note how important the issue of *knowledge representation* is to many applications, not only in the database field, but also in artificial intelligence and programming languages. An important workshop sponsored by SIGART, SIGMOD, and SIGPLAN brought to light many points common to these fields [Brodie and Zilles 1980] (see, e.g., papers by Codd [1980], Lacroix and Pirotte [1980], and Mylopoulos [1980]).

Two additional points indicating the importance of a relational calculus language are that Codd [1972] proposed it as a reference for measuring the "completeness"

[1] It turns out, and it is not fortuitous, that "reasonable" queries precisely correspond to formulas that avoid calling for the domain closure assumption (see Section I.3).

of query languages (discussed below), and query languages known to be "user friendly" (e.g., Query-by-example, Zloof [1977] and Quel, Stonebraker et al. [1976]) are based on it. However, improvements to user interfaces are still required, regardless of the query language.

1.1.2 Extensions to Query Languages

In this section we argue that logic supports powerful extensions to basic query languages in two directions: natural languages and programming languages. Logic languages are close to natural languages. In addition to Codd [1972], who used this argument to support the relational calculus as a yardstick to measure other languages, many linguists have also made this observation; thus querying databases using natural language has been and is a subject of active research [Colmerauer 1973; Colmerauer and Pique 1981; Dahl 1982; Moore 1981; Warren 1981; Woods 1967].

Several authors have questioned Codd's proposal to "define" completeness (i.e., expressiveness) of query languages with reference to a language (relational calculus) whose expressiveness had not been assessed from a semantic viewpoint. This proposal has also been questioned because some well-known operations on data such as the transitive closure of a relation [Aho and Ullman 1979] and aggregation operations used to compute sums, averages, and other operations are not expressible either in the relational calculus or in the relational algebra. This completeness notion has thus attracted much discussion.

Bancilhon [1978] and Paredaens [1978] have shown that, when restricted to finite relations, the relational calculus and the relational algebra are both complete in the sense that for a given database they can express all (and only those) relations whose extension is definable over the set of all domains of that database. This completeness definition is still restrictive as one cannot, for instance, express the transitive closure of a relation with a single expression independently of the extension of that relation. To attain such a capability, various authors proposed that the relational calculus or relational algebra be embedded in a host language [Aho and Ullman 1979; Chandra and Harel 1979, 1980]. Such an embedding will allow a simple expression (i.e., a program in the host language) to define an operator, for example, transitive closure, from the primitive constructs of the host language (such as iteration, recursion, or least fixed points). In some cases, all computable functions can be expressed; this is the ultimate notion of completeness.

Of course, the host language can be a logic programming language. Its expressive power attains completeness, according to Cooper [1980], or can be limited to subsets of computable functions. But logic offers an alternative way to provide extensions such as those that motivate the embedding of the query language (logic) in a host programming logic language. Indeed, the same effect is obtained by extending the representation and manipulation capabilities of the database system itself (rather than of the query language); this is precisely the idea of the deductive database system, where the database system is a theory, usually first order, with nonunit axioms (e.g., see Gallaire and Minker [1978]).

Accepting the view that a database system consisting of a theory that contains nonunit axioms is similar to extending the host language of a database system may not be entirely apparent. Harel [1980], in a review of the book edited by Gallaire and Minker [1978], attempts to refute the above view, but provides no convincing arguments. The database theory view has many theoretical advantages, if not practical ones, which are developed in Section 2 of the paper. Harel [1980] provided another critique of the work presented by Gallaire and Minker [1978], namely that some queries are not characterizable in first-order logic and hence the language must be extended to a higher order language. Although it is true that such extensions may yield answers to queries, that is, relation extensions that may not be first-order definable (e.g., the transitive closure of a relation), the query language remains precisely first order in the database-theoretic view, and may possibly remain first order in a model-theoretic view, depending on the choice of the host logic programming language. Thus the second critique need not apply. The possible failure to note the distinction between regarding a database an an interpretation or as a theory may have been the cause for the comments by Harel as conjectured by Kowalski [1981b].

1.1.3 Null Values

Null values are a special case of the incomplete information problem that is addressed further in Section 2. Null values have been investigated in the ANSI/X3/SPARC [1975] report. Although many different meanings can be ascribed to missing or null values in the instance of a relation, most researchers have dealt primarily with "attributes applicable but values are unknown," whereas only a few deal with "value does not exist" [Zaniolo 1981]. An unknown value can be represented readily in a database, but problems arise with respect to its manipulation and interpretation in a query language.

A model-theoretic approach is to define a three-valued logic [Codd 1979], based on the truth values {true, false, undefined}. The logical connectors truth definition is extended appropriately: Should any component in a tuple be unknown, the corresponding literal has the truth value true, false, or undefined, depending on whether one obtains a literal that is systematically true, systematically false, or either true or false when substituting any value for unknown. This approach has been criticized by several authors [Grant 1977; Vassiliou 1979] because the theory does not provide for several unknown values. Some of the unknown values may be known to be equal even though their precise values are unknown. Furthermore, an expression should be evaluated globally, and not recursively in terms of its subexpressions, in order to infer some of the external knowledge from the incomplete internal knowledge. As a typical example, if the age of John is unknown, the expression

$$\exists x(\text{Age}(\text{John}, x) \mathbin{\&} x < 60)$$
$$\vee\ (\text{Age}(\text{John}, x) \mathbin{\&} x \geq 60)$$

should receive the value true although both operands of the disjunction have value undefined.

Lipski [1979] defines information which *surely*, alternatively *possibly*, can be extracted from a database in the presence of unknown values. He then defines a query language that encompasses such modal operators. Although such an approach has been criticized on grounds of efficiency [Vassiliou 1980], where a denotational semantics approach is specified, Imielinski and Lipski [1981] improve upon Lipski's earlier approach and define systems capable of handling null values when subsets of the relational operators are used.

Where does logic stand in tackling this problem? As already noted, Codd's approach can be considered to be a model-theoretic approach. Other approaches that use logic as a basis and are more general encompass more forms of incomplete information such as indefinite data. Data are said to be indefinite if they are of the form $P(a) \vee Q(b)$, and it is unknown whether $P(a)$ is true, $Q(b)$ is true, or both are true. Reiter [1983, 1984] gives precise solutions to some of these issues on the basis of the proof-theoretic view of databases with null value and indefinite data. The theory that models such a database is obtained from the theory given for a standard database in Section I.3 by the addition of a new class of axioms that stand for facts with null values (Skolem constants in logical terms) and for indefinite data and by a reformulation of the particularization axioms that account for the presence of these new axioms. For details on these axioms see Reiter [1984]. Grant and Minker [1983] provide a precise algorithm to answer queries on such databases for a subclass of such data when null values are contained within the given domain of elements and only positive ground clauses are permitted.

Three additional approaches to the problem of null values use a tool from metalanguage techniques. In Levesque [1981] a language is defined that extends predicate calculus in that one can refer to the state of the database and thus to what is currently known. Both the semantics and proof theory are covered, cases where one can fall back on predicate calculus are studied, and connections with nonmonotonic logic (described in Section 2) are stressed and shown to yield a simpler semantics of the concept of nonmonotonicity. It becomes possible to specify that not all tuples of a relation are known and to query a database as to what is known and what is not known. A slightly different framework is provided by Konolige [1981], who uses a metalanguage based on first-order logic to describe information known about the domain of discourse (i.e., the actual world) and the database language. Queries are specified in the metalanguage. Some queries can be translated into a database language and hence can be evaluated. Others cannot be answered from the database; that is, they have no equivalent answer because of the incompleteness of the data. Both of these approaches can be dealt with within a general framework investigated by Bowen and Kowalski [1982]. They consider predicate logic as an object language and as a metalanguage where the provability relation of the object language can be formulated in the metalanguage. One can reason at the metalanguage level and at the same time provide answers at the object language level.

At this time the results discussed above tend to be more theoretical than practical. In the case of null values, a more practical solution combining logic and relational algebra has been studied by Biskup [1981], who also further investigates Codd's proposals, providing them with a sound foundation and arguing for their practical applicability [Biskup 1982]. In the case of more general incomplete information, we mention here the work by Bossu and Siegel [1981], where a promising approach based on model theory is taken, and the work by Minker [1982]. Finally, an interesting complement to logic for handling null values can be found in Siklossy and Lauriere [1982]. As mentioned above, work in artificial intelligence that deals with nonmonotonic logics is relevant to this topic. The interested reader is referred to papers in the 1980 special issue of the *Artificial Intelligence Journal* on nonmonotonic logic [AIJ 1980].

1.2 Integrity Constraints

1.2.1 Formulation and Enforcement

Database consistency is enforced by integrity constraints, which are assertions that database instances (states) are compelled to obey. Integrity constraints have been classified according to various criteria. The first criterion distinguishes between state constraints, which characterize valid database states, and transition constraints, which impose restrictions on the possible state transitions of a database. Among state constraints different subclasses can be isolated: for example, *type constraints*, which require the arguments of relations to belong to specified domains, or *dependency constraints*, which are discussed in Section 1.4.

As stated by Ullman [1980], a fundamental idea concerning integrity constraints is that query languages can be used to express them, although transition constraints require special attention [Casanova and Bernstein 1979; Florentin 1974; Nicolas and Yazdanian 1978]. It is therefore not surprising that various authors have used a first-order language to study integrity constraints and have appealed to both the model-theoretic and the proof-theoretic logical views.

The model-theoretic view is exploited by Nicolas and Yazdanian [1978] for characterizing those integrity constraints in a database that might be falsified by a given update, and must consequently be evaluated to determine whether the resulting database state is valid. Once such a constraint, say C, has been selected, one can take advantage of the fact that C is known to be satisfied in the state before the update, in order to derive (according to the update) a simplified form of C, say $S(C)$, such that $S(C)$ is satisfied in the new database state iff C is satisfied, and the evaluation cost of $S(C)$ is less than or equal to the evaluation cost of C. Then the evaluation of $S(C)$ can be substituted for the evaluation of C, thus reducing the cost of integrity constraint checking. Such a simplification method, which relies upon truth, preserving instantiations of formulas, is defined by Nicolas [1979a, 1979b] for constraints expressed in the domain relational calculus.

An implementation of this simplification method is reported upon by Homeier [1981]. A similar method was also introduced later by Blaustein [1981] for constraints expressed in the tuple relational calculus. Finally, in Casanova and Bernstein [1979] the same database logical view is used to define a data manipulation language with a logic system that permits one to prove whether or not a transaction preserves consistency (see Section 1.2.2).

In addition, Henschen et al. [1984] describe a technique for extracting integrity tests from database constraints expressed as first-order formulas. The tests can be generated at database design time and are applied when updates to the database appear. Of particular interest in this approach is that tests are applied before the update is made.

An alternative formulation of integrity constraints as first-order formulas has been exploited by McSkimin [1976], McSkimin and Minker [1977], Minker [1978b], and Reiter [1981]. Although their work was done in the context of deductive databases, it can be applied equally well to integrity checking in conventional databases. Both approaches consider a (principal) database augmented with a type database. Types are distinguished by unary relations (or Boolean combinations of them); a type database is a set of formulas (represented as a semantic graph in Minker [1978b]), expressing relations among types (e.g., inclusion, and disjointness) and also the inclusion of certain data values to a particular type. The connection between both databases is made via type integrity constraints that force arguments of relations to be of the same type.

Minker uses these integrity constraints to reject queries that are not well formed, such as a query that requires two relations to be joined on attributes that belong to disjoint types. A refutation-like procedure checks the well formedness of a query by using type constraints and the type database.

Reiter [1981] addresses the problem of detecting the violation of these constraints when the database is updated. Both insertions of data and of general laws are considered (integrity constraints on standard databases, axioms in a deductive database). The method relies upon a transformation of these general laws, which are universally quantified formulas, into a form for which simple criteria for the detection of type constraint violations are proposed.

1.2.2 Proving Consistency of Transactions

It is clearly easier to prove the consistency of transactions when data is defined in a formal framework, and integrity constraints and database transactions (which retrieve, insert, delete, and update information) are stated formally and in the same language. Starting from a database state that complies with given constraints, consistency of transactions is proved when the state arrived at after the transaction has been executed also complies with these constraints. In order to prove consistency, a formal system can be provided whose objects are the transactions: A syntax, semantics, and proof theory for reasoning about objects are needed. A transaction is expressible in a programming language, including expressions used to define sets of data upon which the transaction acts; thus transaction languages include data definition languages. If the data definition language itself is endowed with a proof theory, the transaction formal system can use it. This is the case with the logic interpretation of databases.

Casanova and Bernstein [1979, 1980] offer an elegant, albeit theoretical, answer to these problems. The data definition language, viewed through its logic perspective, includes the integrity constraints as axioms of a first-order theory. The transaction language then is embedded in regular programs [Pratt 1976], supported by a formal system, first-order dynamic logic (DL). This embedding is accomplished by expressing the operations of retrieve, insert, delete, and update in terms of assignment, tests, random tuple selection, union, composition, and iteration operators, the semantics of which encompass that of all computable queries (Section 1.1.2). Regular first-order

DL [Harel 1979] then is extended to a system called modal dynamic logic (MDL) to reason about such programs. MDL is shown to have the necessary power to prove most essential database questions: consistency, transaction, and serializability of transactions. One should note that this work was also extended to deal with aggregate operators (Section 1.1.2). This approach obviously needs a more practical counterpart. Gardarin and Melkanoff [1979] offer a partial answer, using Hoare's logic rather than dynamic logic.

1.3 Query Optimization

Optimization of query evaluation, or improvement of query evaluation as it might better be termed, has been attacked in many different ways. A traditional approach is to use low-level information such as statistical information about various costs to access individual relations. Systems that have been implemented, for example, System R [Chamberlin et al. 1981; Selinger et al. 1979] and more experimental systems [Demolombe 1980; Grant and Minker 1981], have demonstrated that significant gains in efficiency can be achieved by using such information. However, it is clear that additional gains can be obtained by using higher level information, whether syntactic or semantic.

Syntactic transformations, yielding a logical equivalent of the initial query have been studied by Aho et al. [1979] and Chandra and Merlin [1976]. For example, the query

$$\begin{aligned}(\exists u, v, w, y, z)&R(x, v, w)\\ &\& R(x, z, w) \& S(u, w)\\ &\& S(u, v) \& S(y, w)\\ &\& S(y, v) \& S(y, z),\end{aligned}$$

where x is a free variable, can be shown to be equivalent to

$$\exists u, v, w\ R(x, v, w) \& S(u, v).$$

But perhaps the most promising technique is found in the so-called semantic transformations. A first step in that direction was taken by McSkimin [1976] and McSkimin and Minker [1977], who used a form of integrity constraint that describes domains of relations and relates them to each other. Additional information, which takes into account the cardinality of intersections or unions of domains, is used to simplify queries and also to interrupt the process of extracting an answer whenever the additional information justifies it.

A more general approach, related to global problem-solving strategies [Kowalski 1979], is described by King [1981] and Hammer and Zdonik [1980], where the idea of query modification based on general rules is addressed. A set of general rules (integrity constraints and/or deductive laws—see Section 2) such as "a ship carries no more cargo than its rated capacity" or "the only ships whose deadweight exceeds 150 thousand tons are supertankers" [King 1981] can also be used to transform a query submitted by a user into a query less costly to evaluate, eliminating unnecessary relations or propagating constraints. This process then interacts with, and uses information from, a more classical optimizer, which can take into account such factors as indexing of attributes. Obviously a major problem is to control the derivation of queries from the original query. This is a classical problem in artificial intelligence systems, and has been studied by King, who derived a set of heuristics and specified and implemented a plan-generate-test process that gives interesting and practical results. Logic is seen at its best in such applications. Much remains to be done in this important area.

A different use of logic is reported by Warren [1981] in relation to the application of natural language database querying. The underlying database is relational, and logic is used to (1) write a translator from natural language input to an internal representation, (2) represent the internal form of queries as logic formulas, (3) write an optimizer of the querying process, which analyzes the query and uses the traditional type of statistical information already referred to above, to modify the query, and (4) evaluate the query (which could be interfaced to the access level of a standard database management system). The performance of the overall process is very acceptable; the interested reader is referred to Warren [1981] for further details. Warren's approach to optimization is similar to the approach taken by Selinger et al. [1979]. See Chakravarthy et al. [1982] for a discussion of how a logic database may be interfaced with a relational database and related work by Kunifuji and Yokota [1982] and Yokota et al. [1983].

Before leaving this extremely promising field, we note another use of constraints or general laws describing the domain of discourse in the query interpretation process. As discussed by Janas [1979, 1981], whenever a query has an empty answer, a reasonable set of subqueries can be constructed whose failure explains the empty answer of the original query; this reasonable set is obtained by taking particular integrity constraints into account. A reason for not having an answer is that no tuples that currently exist in the database satisfy the query, in which case the subquery that fails can be identified or the constraints of the database are such that no answers that satisfy the query will ever be possible from the database. Such information could be of considerable value to a user. See also Demolombe [1981] for related work.

1.4 Database Design

There is an area where logic plays an increasing role in the specification of data models. In general, there are several methods of formal specification and several formalisms for each method; databases are but one kind of object to formalize, and techniques developed for programming languages in general can be applied. We have seen in Section 1.2.2 how such a specification can be used for a precise purpose: proving the consistency of a transaction. Veloso et al. [1981] provide a comprehensive review of these techniques, for a large part on the basis of logic (see also Borgida and Wong [1981], where logic is used to define the formal semantics of the Taxis data model). Logic is a very important tool in this area, if only because it blends nicely with all the other tools described in this paper for different purposes.

We now turn to data dependencies, a concept central to database design. Data dependencies are special cases of integrity constraints that express structural properties of relations and permit relations to be decomposed, and retain certain properties described below. A number of dependencies of various kinds have been characterized and studied in the literature (see Delobel [1980] and Maier [1983] for comprehensive surveys on dependencies). In this section we see how logic has been used to study the properties of some of these dependencies and, in some cases, define them. We also note that special formal systems have been developed for that purpose (e.g., see Armstrong [1974]). This section is divided into two parts; the first is concerned with studies involving propositional logic, and the second with studies involving first-order logic.

1.4.1 Propositional Logic and Dependencies

Delobel, Fagin, Parker, and Sagiv [Delobel and Parker 1978; Sagiv and Fagin 1979; Sagiv et al. 1981] have shown that an equivalence exists between some dependencies and a fragment of the propositional logic. They have shown that functional dependencies (FDs) [Codd 1970] and multivalued dependencies (MVDs) [Fagin 1977b; Zaniolo 1976] can be associated with propositional logic statements. The equivalence developed between these dependencies and propositional logic extends earlier results of Fagin [1977a] and also Delobel and Casey [1973] that relate FDs to the theory of Boolean switching functions.

The above-mentioned equivalence provides new techniques for proving properties of FDs and MVDs, and for solving the membership problem for those dependencies. Additionally, shorter and simpler proofs have been obtained for important theorems about FDs and MVDs, and strategies developed for special-purpose theorem provers provide efficient algorithms for the membership problem. Furthermore, on the basis of a proof of this equivalence, a characterization of the dependency basis in terms of truth assignments has been given by Fagin [1977b]. This has led to the development of an efficient membership algorithm for FDs and MVDs by Sagiv [1980], which has been supplanted by a faster algorithm by Galil [1982]. As another application of this equivalence, Parker and Delobel [1979] have developed an algorithm to determine whether a set of attributes is a *key* for a relation, that is, whether a set of attributes uniquely determines a tuple in the relation and is itself not contained in any other set of attributes that uniquely determines a tuple.

The equivalence, first established for FDs, was extended later to include MVDs [Sagiv et al. 1981]. It has also been extended to other kinds of dependencies, for example, Boolean dependencies, which are expressions of attributes built using the Boolean operators &, $\vee$, $\neg$. For example, $A \vee (B \,\&\, \neg C)$ is a Boolean dependency whose meaning is "for every pair of tuples, either the two tuples agree on attribute A, or the two tuples agree on attribute B and disagree on attribute C." However, although this shows that the equivalence can be extended to some generalization of FDs, it cannot be extended to embedded MVDs (MVDs that hold for a projection of a relation) or to mutual dependencies for which the inferential structure of propositional logic seems to be too weak [Delobel 1978]. Nicolas [1978] was the first to suggest that first-order logic be used. However, before considering first-order logic, we note a similar equivalence result between FDs obtained by Vassiliou [1980], who redefined a FDs interpretation in order to account for null values, and implicational statements of a model propositional logic system. Vassiliou exploited this equivalence, notably, for proving the completeness of a set of inference rules for those "newly interpreted" FDs.

1.4.2 First-Order Logic and Dependencies

By considering dependencies as first-order formulas, one provides advantages similar to those sketched for propositional logic; results from proof theory and model theory can be used to study their properties.

Dependency statements can be expressed as first-order formulas. For example, given a relation scheme $R(ABCD)$, the FD $C \rightarrow D$ and the MVD $A \rightarrow\rightarrow B$ are, respectively, equivalent to the following two first-order formulas:

$$\forall x \cdots \forall v'(R(x, y, z, v) \,\&\, R(x', y', z, v') \rightarrow (v = v')),$$

$$\forall x \cdots \forall v'(R(x, y, z', v') \,\&\, R(x, y', z, v) \rightarrow R(x, y, z, v)).$$

New kinds of dependencies have been characterized and defined directly as particular first-order formulas. Typical of this are generalized dependency statements (GDs) and their embedded version (EGDs) [Grant and Jacobs 1982], implicational dependencies (IDs) and their embedded version (EIDS), and extended embedded IDs (XEIDs) [Fagin 1980, 1982], and tuple- and equality-generating dependencies [Beeri and Vardi 1980, 1981]. We briefly specify the main results related to these dependencies below.

Essentially, generalized dependencies are Horn clauses that contain no function symbols; they capture FDs, MVDs, JDs (join dependencies), IDs, and some other constraints. Horn clauses are defined in Section I.2 and in Section 2. After studying the implication problem for GDs, Grant and Jacobs [1982] proposed a decision procedure for determining whether a GD is a logical consequence of a set of GDs. This procedure is related to both techniques from automatic theorem proving and the "chase method," a decision procedure for dependencies described by Maier et al. [1979] on the basis of the tableau formalism of Aho and Ullman [1979].

Horn clauses were also used to define EIDs, which were studied to "help bring order to the chaos by presenting certain mathematical properties shared by all (the previously defined) dependencies" [Fagin 1980, 1982]. Among these properties are *domain independence*, which means that whether or not a dependency holds for a relation can be determined independently of the underlying domains of the attributes in the relation, *satisfiability on empty relations* (i.e., relations with no tuples), and *faithfulness* with regard to a version of the Cartesian product called direct product.

As principle results for EIDs, Chandra et al. [1981] have shown that any set D of EIDs admits an Armstrong relation (i.e., a relation that obeys all dependencies in D—and their consequences—but no others) and that the decision problem for this class of dependencies is undecidable. However, a complete set of inference rules has been given for the equivalent class of algebraic dependencies [Yannakakis and Papadimitriou 1982].

On the basis of the formulation of dependencies as first-order formulas, Beeri and Vardi [1980, 1981] have studied the implication problem for a general class of dependencies, the tuple- and equality-generating dependencies (tgds and eqds), and for some of its subclasses. These dependencies, which, in fact, correspond to EIDs, intuitively require that, "if some tuples fulfilling certain conditions exist in the database, then either some other tuples (possibly with unknown values), fulfilling certain conditions, must also exist in the database (tgds), or some values in the given tuples must be equal (eqds)" [Beeri and Vardi 1980, 1981].

In the same work Beeri and Vardi proposed an extension to the chase method (see also Sadri and Ullman [1980, 1982]), which provides a proof procedure for these dependencies and a decision procedure for total dependencies (i.e., nonembedded). A decision procedure is guaranteed to terminate in any case, whereas a proof procedure is not guaranteed to terminate when D does not imply d. It is worth noting that, when described in the formalism of logic, the chase method corresponds to a well-known theorem-proving procedure by refutation (using resolution and paramodulation) [Beeri and Vardi 1981].

We have described results by using logic that relates to conventional relational databases; in the following section, we describe how logic extends conventional databases to permit deduction and sheds new insight into problems concerning negative and incomplete information.

2. DEDUCTIVE DATABASES

A deductive database is a database in which new facts may be derived from facts that were explicitly introduced. We consider such databases here from a proof-theoretic viewpoint as a special first-order theory. In this framework, we focus upon several subjects: the manner in which *negative data* are to be treated in a database, the *null value* problem in which the value of a data item is missing, and *indefinite data* in which one knows, say $P(a) \vee P(b)$ is true, but one does not know if $P(a)$ is true, $P(b)$ is true, or both are true.

We first provide the background for defining deductive databases, and then treat two different kinds of deductive databases: definite and indefinite. It will be seen that assumptions generally made with respect to definite databases do not apply directly to indefinite databases. Finally, we briefly discuss other extensions to deductive databases and logic databases.

For additional material on the subjects of deductive databases or logic and databases not covered in this survey article, see Gallaire et al. [1984], the *Proceedings of the First Conference on Logic Programming* [1982], *the Logic Programming Workshop Proceedings* [1983], *the International Joint Conference on Artificial Intelligence* [1983], *the International Symposium on Logic Programming* [1984], and other conferences devoted to artificial intelligence and logic programming.

2.1 Definition of Deductive or Logic Databases

In general, we shall consider a database to consist of a finite set of constants, say $\{c_1 \cdots c_n\}$, and a set of first-order clauses without function symbols (see Section I.2). Functions are excluded in order to have finite and explicit answers to queries. Initially our theory precludes null values that arise in a database when one has statements such as $(\exists x)P(a, x)$, that is, linked to "a" in the predicate P there is a value, but its precise value is unknown. When one skolemizes the formula $(\exists x)P(a, x)$ and places it in clause form, the clause $P(a, \omega)$ results, where ω is a Skolem constant (i.e., a constant whose value is otherwise unconstrained).

The general form of clauses that will represent facts and deductive laws is

$$P_1 \;\&\; P_2 \;\&\; \cdots \;\&\; P_k \rightarrow R_1 \vee \cdots \vee R_q.$$

It is equivalent to the clause

$$\neg P_1 \vee \cdots \vee \neg P_k \vee R_1 \vee \cdots \vee R_q.$$

The conjunction of the P_i is referred to as the left-hand side of the clause and the disjunction of the R_j as the right-hand side. Since the clauses that we will consider are function free, terms that are arguments of the P_i and R_j are either constants or variables. Whenever any variable that occurs in the right-hand side of a clause also occurs in the left-hand side, the clause is said to be *range restricted.* We shall briefly discuss various types of clauses depending on the respective values of k and q, as in Minker [1983]:

Type 1: $k = 0$, $q = 1$. Clauses have the form

$$\rightarrow P(t_1, \ldots, t_m).$$

(a) If the t_i are constants, $c_{i1}, \ldots, c_{im}$, then one has

$$\rightarrow P(c_{i1}, \ldots, c_{im}),$$

which represents an assertion or a fact in the database. The set of all such assertions for the predicate letter P corresponds to a "table" in a relational database. The arrow preceding an assertion will generally be omitted.

(b) When some, or all, of the t_i are variables, the clause corresponds to a general statement in the database. For example,

$$\rightarrow \text{Ancestor}(\text{Adam}, x),$$

states that Adam is an ancestor of all individuals in the database (the database consists only of human beings). Clearly, such data, which are not range-restricted clauses and therefore assume that all the individuals in the database are of the same "type," appear very seldom.

Type 2: $k = 1$, $q = 0$. Clauses have the form

$$P(t_1, \ldots, t_m) \rightarrow.$$

(a) When all of the t_i are constants, then we have

$$P(c_{i1}, \ldots, c_{im}) \rightarrow,$$

which stands for a negative fact. Negative statements may seem peculiar since relational databases do not contain negative data. We shall return to this topic in a later section.

(b) Some of the t_i are variables. This may either be thought of as an integrity constraint (as a particular Type 3 clause; see below), or as the "value does not exist" meaning for null values (see Section 1.1.3).

Type 3: $k > 1$, $q = 0$. Clauses have the form

$$P_1 \;\&\; \cdots \;\&\; P_k \rightarrow.$$

Such axioms may be thought of as integrity constraints. That is, data to be added to a database must satisfy the laws specified by the integrity condition to be allowed in the database. For example, one may specify an integrity law that states that "no individual can be both a father and a mother of another individual." This may be specified as

$$\text{FATHER}(x, y) \;\&\; \text{MOTHER}(x, y) \rightarrow.$$

If FATHER(JACK, SALLY) is already in the database, an attempt to enter MOTHER(JACK, SALLY) into the database should lead to an integrity violation. This does not rule out other kinds of integrity constraints.

Type 4: $k \geq 1$, $q = 1$. Clauses have the form

$$P_1 \;\&\; P_2 \;\&\; \cdots \;\&\; P_k \rightarrow R_1.$$

The clause may be considered to be either an integrity constraint or a definition of the predicate R_1 in terms of the predicates $P_1, \ldots, P_k$ (such a definition is a deductive law).

Type 5: $k = 0$, $q > 1$. Clauses have the form

$$\rightarrow R_1 \vee R_2 \vee \cdots \vee R_1.$$

If the x_i, $i = 1, \ldots, n$ are constants, then we have an *indefinite assertion.* That is, any combination of one or more R_i is true, but we do not know which ones are true.

Type 6: $k \geq 1$, $q > 1$. Clauses have the form

$$P_1 \,\&\, P_2 \,\&\, \cdots \,\&\, P_k \rightarrow R_1 \vee R_2 \vee \cdots \vee R_q.$$

The clause may be interpreted as either an integrity constraint or the definition of indefinite data. An integrity constraint that states that each individual has at most two parents may be written as

$$P(x_1, y_1) \,\&\, P(x_1, y_2) \,\&\, P(x_1, y_3) \rightarrow (y_1 = y_2) \vee (y_1 = y_3) \vee (y_2 = y_3).$$

As a general rule of deduction we might have

$$\text{Parent}(x, y) \rightarrow \text{Mother}(x, y) \vee \text{Father}(x, y).$$

This general law could also be interpreted as an integrity constraint.

Finally, a clause where $k = 0$, $q = 0$ (the empty clause) denotes falsity and should not be part of a database. Furthermore, we shall call a clause *definite* if its right-hand side consists of exactly one atom (i.e., Type 1b or Type 4).

All the types of clauses defined above, except ground facts (Type 1a), are treated as integrity constraints in conventional databases. In a deductive database some of them may be treated as deductive laws. We shall distinguish two classes of databases: definite databases in which no clauses of either Type 5 or Type 6 appear and indefinite databases in which such clauses do appear.

2.2 Definite Deductive Databases (DDDBs)

2.2.1 A Formal Definition of DDDBs

A definite deductive database is defined as a particular first-order theory (together with a set of integrity constraints). This theory is obtained from the theory given for conventional databases in Section I.3 by the addition of a new class of axioms, which stand for the deductive laws, and by a reformulation of the completion axioms, which account for the presence of these new axioms. More precisely, a definite deductive database consists of the following:

(1) *A theory T* whose proper axioms are

- Axioms 1 (the particularization axioms): the domain closure axiom, the unique name axioms, the equality axioms (as given in Section I.3), and the completion axioms (one for each predicate in T) whose formulation is given below.
- Axioms 2 (the elementary facts): a set of ground atomic formulas defined by clauses of Type 1a in Section 2.1.
- Axioms 3 (the deductive laws): a set of function-free definite clauses of Type 4 (or Type 1b) in Section 2.1.

(2) *A set of integrity constraints IC*, which consists of any closed formulas.

The completion axiom for a predicate P is now not only built from the facts related to P (which occur in Axioms 2), but also from the "only if" missing part of the definite clauses defining P (which occur in Axioms 3). For example, let P have the following assertions in T:

$$P(c_i, c_j),$$
$$P(c_p, c_q),$$

and let

$$Q(x, y) \,\&\, R(y, z) \rightarrow P(x, z),$$

and

$$S(x, y) \rightarrow P(x, y),$$

be all the clauses in Axioms 3 that imply P; then, the completion axiom for P is

$$P(x, y) \rightarrow ((x = c_1) \,\&\, (y = c_j)) \vee ((x = c_p) \,\&\, (y = c_q)) \vee (Q(x, y) \,\&\, R(y, z)) \vee (S(x, y)).$$

Such a completion axiom permits one to derive a negative fact $\neg P(d, e)$ whenever $P(d, e)$ is neither in Axioms 2 nor derivable through Axioms 3. Thus from the database just specified we can derive $\neg P(c_i, c_p)$.

In such a deductive database the definition of answers to queries and of the satisfiability of integrity constraints is equivalent to their definition in a conventional database viewed from the proof-theoretic perspective. An answer to a query $W(x_1, \ldots, x_p)$, where $x_1, \ldots, x_p$ are free variables in W, is the set of tuples $\langle c_{i1}, \ldots, c_{ip} \rangle$ such that $T \vdash W(c_{i1}, \ldots, c_{ip})$. Now, a deductive database obeys the integrity constraints in IC iff for any formula ϕ in IC $T \vdash \phi$.

An alternative definition can be given for the satisfiability of integrity constraints: A deductive database obeys the integrity constraints in IC iff $T \cup$ IC (the axioms in T together with the formulas in IC) is consistent [Kowalski 1979]. Since the theory T, as defined above for a definite deductive database, is complete (i.e., for any closed formula W, either $T \vdash W$ or $T \vdash \neg W$), both definitions are equivalent in that case. However, for more general deductive databases whose corresponding theory is not complete, the second definition is less stringent than the first and its impact is worth investigating.

Deductive laws (in Axioms 3) that imply a relation R provide an extended definition for R. The tuples $\langle c_{i1}, \ldots, c_{im} \rangle$ that "satisfy" R are not only those tuples such that $R(c_{i1}, \ldots, c_{im})$ is a fact in Axioms 2, but also those tuples such that $R(c_{i1}, \ldots, c_{im})$ is derivable through the deductive laws. Relations that are jointly defined by deductive laws and elementary facts in a deductive database, called derived relations, constitute a generalization of relations defined as "views" in a conventional database. A "view" is a relation, not stored in the database, that is defined in terms of database relations or other views by a relational algebra (or calculus) expression. A derived relation reduces to a view when

(i) there are no elementary facts in Axioms 2 related to this relation, and
(ii) no recursive deductive law or cycle appears among the deductive laws that imply this relation in Axioms 3.

In such a case, if $E_1 \to R, \ldots, E_q \to R$ are all the deductive laws that imply R, this derived relation corresponds to the view $V = E_1 \vee \cdots \vee E_q$. We note that Point (i) is not significant since two different names may be given to the "explicit part" and the "derived part" of a relation. However, Point (ii) has more impact. Some relations may be defined as derived relations, whereas strictly, they cannot be defined as views. A typical example is the Ancestor relation, which is the transitive closure of the Parent relation. Incidentally, we note that the relational calculus (or algebra) may be extended (or embedded into another language) in order to be able to define such a relation as a view. However, since from a model-theoretic viewpoint the transitive closure of a relation is not first-order definable, one has to call for languages that are more powerful than first-order language. But, in a (definite) deductive database defined from a proof-theoretic point of view as above, the Ancestor relation can easily be defined as a derived relation in terms of two deductive laws:

$$\begin{aligned}&(\text{Parent}(x, y) \to \text{Ancestor}(x, y),\\ &\text{Ancestor}(x, y) \;\&\; \text{Parent}(y, z)\\ &\qquad\qquad \to \text{Ancestor}(x, z)),\end{aligned}$$

and strictly remain in the context of first-order logic (see Section 1.1.2).

Clearly, on considering the combinatorial complexity of the particularization axioms (Axioms 1), it would be quite inefficient to implement a (definite) deductive DBMS, while clinging to the formal definition of a DDDB given here, namely, to implement such a DBMS as a standard theorem prover that treats axioms in Axioms 1 in the same way as axioms in Axioms 2 or Axioms 3. The solution is similar to what is done for conventional DBMSs. It consists of substituting adequate metarules (or metaconventions) for the particularization axioms, thus obtaining a so-called operational definition of a DDDB. The following section is devoted to this issue.

2.2.2 An Operational Definition of DDDBs

Particularization axioms may be eliminated for DDDBs in a way that is similar to conventional databases, thus providing a convenient way to implement corresponding systems. First, calling for the domain closure axiom may be avoided by dealing with range-restricted formulas for query, integrity constraint, and deductive law formulation (see Section 1.1.1). Then, as discussed by Kowalski [1978, 1979] and proved by Clark [1978], the unique name and completion axioms may be removed, provided that negation is interpreted as (finite) failure (see also Jaffar et al. [1983] for a stronger result). The metarule of *negation as failure* states that for any positive literal P, $\vdash \neg P$ iff $\nvdash P$; that is, failure to prove P permits one to infer $\neg P$. Finite failure further requires that all proof attempts terminate. Finally, since the equality axioms were needed only for the presence of the equality predicate in the above axioms, they are no longer required.

To summarize, keeping in mind that any formula now has to be range restricted, from an operational point of view a DDDBs consists of the following:

(1) A set of axioms: Axioms 2 (elementary facts) $\cup$ Axioms 3 (deductive laws).
(2) A set of integrity constraints: IC.
(3) A metarule: negation as finite failure (nff).

Query answering and integrity constraint satisfiability remain defined as in Section 1.3.1 except that, now "$\vdash$" has to be interpreted as "provable under nff."

Negation as finite failure generalizes in the deductive databases case to the usual assumption underlying conventional databases for negative facts (see Section I.3). The use of this concept was discussed as the "convention for negative information representation" by Nicolas and Syre [1974] and Nicolas and Gallaire [1978], and also described as the "closed world assumption" (CWA)[2] by Reiter [1978a, 1980]. This concept is also used in the artificial intelligence languages PLANNER [Hewitt 1972] and PROLOG [Roussel 1975].

The syntactic definition (i.e., according to proof theory) of negation as failure given above has a semantic (i.e., according to model theory) counterpart. Let G be the set of all possible ground (positive) atomic formulas constructible from the symbols in a given set of definite clauses. A Herbrand model for this set of clauses is a subset of G that makes all the clauses true. It has been shown [Van Emden and Kowalski 1976] that the intersection of all such models is itself a model, and indeed is the minimal model (i.e., it contains the minimal number of atomic formulas). If one substracts the set of atomic formulas in the minimal model from G, the remaining set of atomic formulas is the set of all formulas whose negations may be assumed to be true. These are the same atomic formulas obtained by the CWA. This shows that the CWA and negation as failure assume complete knowledge, and there are no unknown facts. A discussion of the semantic definition of negation as failure can be found in Minker [1982] and Van Emden [1978].

Although the formal and operational definitions of a DDDB are equivalent in the sense that they will give the same answers to queries, they are in fact not strictly equivalent. The formal definition is stated in standard first-order logic, which is monotonic, whereas the use of negation as failure in the operational definition leads to a nonmonotonic logic. A logic is said to be *monotonic* if one is given a theory T (i.e., a set of axioms) in which a formula w can be proved (i.e., $T \vdash w$), then the addition to T of an axiom A still permits one to prove w; that is, $T \cup \{A\} \vdash w$. According to negation as failure, $\neg P(b)$ can be inferred from $\{P(a), Q(b)\}$ but not from $\{P(a), Q(b)\} \cup \{P(b)\}$; thus we have a nonmonotonic logic. For an analysis of the relation between predicate completion and work in artificial intelligence on nonmonotonic logic, the reader is referred to Reiter [1982].

Finally, we note that, as shown by Nicolas and Gallaire [1978] and formally proved by Reiter [1978a], a definite deductive database is always consistent under the CWA. The intuitive reason is that definite clauses preclude the derivation of positive facts from negative facts. We see in Section 2.3 that this is not the case when indefinite clauses are accepted as deductive laws.

2.2.3 Deductive Laws and Integrity Constraints

Both deductive laws in Axioms 3 and integrity constraints in IC correspond to general knowledge of the world modeled by the

[2] There is the open world assumption (OWA) corresponding to the CWA that provides a standard interpretation of negation as given with a full first-order theory.

database. Given such general knowledge, we might inquire as to the basis on which one can decide to consider a general rule as a deductive law, and thus incorporate it in Axioms 3, or as an integrity constraint, and thus incorporate it as part of the IC. There is no final answer to this (database design) question, but some suggestions are provided by Nicolas and Gallaire [1978] and Reiter [1978a, 1984]. They are briefly stated below.

(i) If one wants to obtain finite and explicit answers to queries (i.e., sets of tuples of elements in the database), deductive laws have to be function free. Thus general knowledge that corresponds to formulas that do not fulfill this constraint should be treated as integrity constraints. For example, the general knowledge "every teacher has a diploma,"

$$(\forall x\ \forall y(\mathrm{TEACH}(x, y) \rightarrow (\exists z)\mathrm{DIPLOMA}(x, z))),$$

should be treated as an integrity constraint since the clause form of the axiom contains a Skolem function.

(ii) In order to avoid inconsistency with the CWA, one retains as deductive laws only general knowledge that corresponds to definite clauses (and thus use the other clauses as integrity constraints). However, as we shall see in Section 2.3, there is another possibility, which consists of modifying the CWA.

(iii) Since purely negative clauses (i.e., clauses of Type 3 in Section 2.1) will never produce new facts (under the CWA), they need only be used as integrity constraints [Reiter 1978a].

(iv) General knowledge that implies instances of a relation that is completely defined independently of it will not, if used as a deductive law, produce any new valid facts and should be used as an integrity constraint.

For example, the general knowledge, "the age of any person is less than 150,"

$$(\forall x\ \forall y)(\mathrm{Age}(x, y) \rightarrow (y < 150)),$$

used as a deductive law, would always produce facts which are either inconsistent (e.g., $180 < 150$) or redundant (e.g., $35 < 150$). A functional dependency statement such as

$$(\forall x\ \forall x'\ \forall y)(\mathrm{Father}(x, y)\ \&\ \mathrm{Father}(x', y) \rightarrow (x = x')),$$

is another example of this kind. Hence these rules are best used as integrity constraints.

In the two examples above, the implied relations are of a particular kind, and it is generally agreed that they are defined independently of any reference to a specific database. For more standard relations, it is a matter of choice (dependent upon database design) to decide what general knowledge and which assertions will participate in the complete definition of a relation. As an extreme case, in a conventional database it is (implicitly) assumed that every relation (other than views) is completely defined in terms of elementary facts (the tuples in the corresponding table), and thus all general knowledge is used as integrity constraints.

As noted by Kowalski [1979] and as we shall emphasize here, a deductive database can be viewed as a logic program that produces facts (the facts in the minimal model characterized in Section 2.2.2) and whose integrity constraints are the program properties. Modifications to the program (viz. database updates involving either deduction laws or elementary facts) must preserve those properties.

When general knowledge has been partitioned between deductive laws and integrity constraints, they have to be exploited conjointly. Integrity checking in a deductive context is discussed by Nicolas and Yazdanian [1978] and Reiter [1981], and a methodology for updating databases with integrity constraints and deductive laws was suggested by Fagin et al. [1983]. Further, query evaluation and update handling are uniformly treated by dos Santos et al.

[1981], where deduction provides either an analysis of a state of the database, that is, an answer to a query, or a plan, that is, a sequence of database modifications to reach a desired state. We now focus on deductive laws.

As mentioned in the preceding section, the theory that constitutes a definite deductive database admits a (unique) minimal model. This (minimal) model consists of a set of facts that connotes that these are the facts that are true. These facts constitute the conventional database (CDB) underlying the deductive database. Now, one can choose to exploit the deductive database in either of two ways. The underlying CDB can be kept implicit, in which case the deductive laws, or their compiled form (see Section 2.2.4.2), have to be run at query evaluation time to find the (implicit) deducible facts. The alternative consists of making the underlying CDB explicit. To do so, deductive laws have to be run when new facts are inserted into the database to make explicit the deducible facts. Then query evaluation can be done as in a conventional database. In the first case deductive laws are said to be used as derivation rules, whereas in the second case, they are said to be used as generation rules [Nicolas and Gallaire 1978]. The respective advantages of derivation and generation rules are discussed by Nicolas and Gallaire [1978] and Nicolas and Yazdanian [1982]. We only mention here that, as opposed to derivation paths, "generation paths" stop naturally even when recursive rules or cycles among the rules appear. The interested reader will find a description of a prototype deductive DBMS by using generation rules in Nicolas and Yazdanian [1982].

Finally, we note that exploiting deductive laws as generation rules can be viewed as an automatic revalidation of integrity constraints in a conventional database. This can also be viewed as providing a generalization of "concrete views" [Blaustein 1981], that is, views whose corresponding set of tuples is explicitly stored.

In the following section we focus on proof-theoretic techniques used for exploiting derivation rules.

2.2.4 Inference Methods and Database Access

We shall describe two ways to perform the inference process: the interpretive and the compiled methods.

The *interpretive method* works with a problem solver, using the deductive laws and interleaves search of the extensional DB (which contains the elementary facts, i.e., Axioms 2). In the *compiled approach*, the problem solver uses all of the deductive laws until a point is reached at which either the problem is solved or all that remains is to search for facts in the extensional DB. Both methods work well when deductive laws are assumed to be free of cycles, that is, when there are no recursive axioms. Otherwise, both have difficulties handling the termination problem, that is, detecting at which point no new solutions will be found. However, it should be possible to find such termination conditions, as only a finite number of tuples can be generated as answers to any query to a definite deductive database (function free, with a finite set of constants and consisting of only definite clauses). This problem has been studied by a number of authors [Chang 1981; Kunifuji and Yokota 1982; Minker and Nicolas 1982; Naqvi and Henschen 1984; Reiter 1978c; Shapiro and McKay 1980; Yokota et al. 1983].

2.2.4.1 Interpretive Method of Deduction. We shall describe the interpretive method for the following simple database. Let the facts in the database (the extensional DB) be

$F(e, b_1)$	$M(c, e)$	$H(a, c)$
$F(e, b_2)$	$M(c, f)$	
$F(e, b_3)$	$M(c, g)$	
	$M(g, d)$	

where F, M, and H stand, respectively, for Father, Mother, and Husband. Let the deductive laws be

$$(A_1)\quad M(x, y)\ \&\ M(y, z) \rightarrow GM(x, z),$$

$$(A_2)\quad M(x, y)\ \&\ F(y, z) \rightarrow GM(x, z),$$

$$(A_3)\quad GM(z, y)\ \&\ H(x, z) \rightarrow GF(x, y).$$

Relation GM (Grandmother) is said to be intensionally defined in terms of the extensional relations F and M. Relation GF (Grandfather) is intensionally defined in terms of the extensional relation H and intensional relation GM, and hence in terms of the extensional relations F and M as well. There is no loss of generality in assuming that no relation is hybrid; that is, all relations are either (purely) intensional or (purely) extensional (e.g., see Minker [1982]). We illustrate the interpretive approach, starting from a query $GF(a, y)$:

(1) $$GF(a, y),$$

and applying A3, using the Robinson Resolution Principle (Section I.2) yields

(2) $$GM(z, y) \;\&\; H(a, z)$$

as the subproblems to be solved. At this point a selection function could be used to provide advice to solve $H(a, z)$ first. The reason for selecting $H(a, z)$ first is that it contains a constant and is presumably easier to satisfy than the case where one arbitrarily looks for a tuple that satisfies $GM(z, y)$ and hopes that the value of z, when substituted into $H(a, z)$, will be in the database. By accessing the database, we find that $H(a, c)$ is in the database and z is bound to c. Now, the subproblem

(3) $$GM(c, y)$$

remains to be solved. Since there are two ways to solve this last goal on the DB (Axioms A_1 and A_2), a choice is made to select Rule A2, yielding

(4) $$M(c, y_1) \;\&\; F(y_1, y),$$

which now must be solved. These subproblems may be solved in two steps, to obtain one answer, $\{b_1\}$. However, the process is not finished, and backtracking at previous choice points will yield $\{b_2, b_3, d\}$ as further answers. So, for the sake of efficiency, at each step one has to involve a selection function and a choice function. Some of the variants of this basic method have explored the idea of obtaining the whole set of answers at each access to the DB, rather than one at a time [Chakravarthy et al. 1982; Minker 1975a, 1975b, 1978a, 1978b]. Although no termination conditions are known in the general case, one should note that this problem also exists in logic programming, where it is left up to the skill of the programmer to specify a termination condition.

2.2.4.2 Compilation Method of Deduction. Access to the database is delayed in the deductive process until all the work remaining to be done for query evaluation is to access the database. This should allow the possibility for global optimizations of DB accesses. The method was basically described by Chang [1978], Furukawa [1977], Kellogg et al. [1978], Kellogg and Travis [1981], and Reiter [1978b]. Two possible techniques can be used. In the first technique, which could be called a pseudocompilation technique, only one path at a time is pursued. That is, a single expression involving only extensional relations is produced at a given time; backtracking then produces further expressions. This is clearly a source of redundant work. With the above deductive laws, first one would get the expression

$$M(z, y_1) \;\&\; M(y_1, y) \;\&\; H(a, z),$$

which would be passed over to the DB evaluator, and along a second path one might generate the expression

$$M(z, y_1) \;\&\; F(y_1, y) \;\&\; H(a, z),$$

which has obvious redundancies with the previous expression [Chakravarthy et al. 1982; Kunifuji and Yokota 1982]. The second technique aims at producing an iterative program that synthesizes the set of all retrieval expressions. Let us illustrate it by extending the deductive laws of our previous example to the following intensional recursive relation (Ancestor):

$$F(x, y) \;\&\; A(y, z) \rightarrow A(x, z),$$
$$M(x, y) \rightarrow A(x, y).$$

In this case, a method described by Naqvi and Henschen [1980, 1984] would produce the following program from an initial query $A(?, a)$.

```
$z_2 := a$
eval(F($y_2$, $y_1$) & M($y_1$, $z_2$))
                                  /likely to access $M$ first/
print($y_1$)     /there may be several values of $y_1$/
enque($S$, $y_1$)
  /put in set $S$ those values not already put into
                                                  it/
while($S \neq$ empty) do
  $y_2$ := deque($S$)        /take the first element/
  eval(F($y_3$, $y_2$))
  print($y_3$)
  enque($S$, $y_3$)
  end
```

This method is general but suffers from some drawbacks in that some redundancy may still remain. The enque process may be time consuming and the program generated depends on the initial query.

2.3 Indefinite Deductive Databases (IDDDBs)

An indefinite deductive database differs from a definite deductive database (referring to its operational definition) in Axioms 3 and the metarule to handle negation. The difference in Axioms 3, although seemingly minor, will be seen to be substantial. We define Axioms 3′ as follows:

- Axioms 3′: a set of function-free definite or indefinite clauses. An indefinite clause is to the form given by Type 5 or Type 6 in Section 2.1, for example,

$$P(a, b) \vee Q(c, d, e),$$

or

$$R(x, y) \,\&\, T(y, z) \rightarrow P(x, z) \vee Q(x, y, z).$$

More precisely, an *indefinite deductive database* consists of

(1) A set of axioms T, where T = Axioms 2 ∪ Axioms 3′.
(2) A set of integrity constraints IC.
(3) A metarule: generalized negation as failure, described below.

The addition of indefinite clauses changes matters radically. Methods and results that apply to definite deductive databases do not apply to indefinite deductive databases. For example, an indefinite database may be inconsistent under the CWA. Consider a database that consists of a single fact,

$$\text{cat(felix)},$$

and of a single deductive law,

$$\text{cat}(x) \rightarrow \text{black}(x) \vee \text{white}(x).$$

Since black(felix) cannot be proved, application of the CWA leads to

$$\vdash \neg\text{black(felix)}.$$

Similarly, one can also conclude

$$\vdash \neg\text{white(felix)}.$$

But the set

$$\{\text{cat}(x) \rightarrow \text{black}(x) \vee \text{white}(x), \text{cat(felix)}, \neg\text{black(felix)}, \neg\text{white(felix)}\}$$

is obviously inconsistent. Hence the closed world assumption (negation as failure) as defined for definite databases is not applicable to indefinite databases.

The concept of a CWA can be extended to achieve a *generalized closed world assumption* (GCWA) as follows. Let E be the set of all purely positive (possibly empty) clauses not provable. Then assert $\neg P(x)$ iff $P(x) \vee C$ is not provable for any C in E. We also refer to this as *generalized negation as failure*. As in the case of definite databases, one can obtain a semantic interpretation of generalized negation as failure. But, whereas in a definite database there is a unique minimal model, in an indefinite database a set of minimal models arises. The GCWA as defined above was introduced by Minker [1982]. In the case where no general axioms appear other than definite ground clauses, Grant and Minker [1983] have developed an algorithm on the basis of the GCWA to compute answers to queries. Although the GCWA provides a sound formal basis, this notion has not yet been shown to be sufficiently efficient when general axioms are permitted to be directly usable in practice. A variant that may be more promising from a practical point of view was introduced independently, and widely studied by Bossu and Siegel [1981].

The GCWA treats null values correctly, where it is meant here that the null value is among the constants already known in the database. Let the database consist of $\{P(\omega), Q(a), Q(b)\}$, where '$\omega$' is a null value, or Skolem constant that arises from the statement $\exists x P(x)$. If the domain D consists only of $\{a, b\}$, then the existentially quantified statement corresponds to

$$(\exists x P(x)) \leftrightarrow P(a) \vee P(b).$$

Hence $P(\omega)$ is a shorthand notation for $P(a) \vee P(b)$. If one were to replace the entry $P(\omega)$ with $P(a) \vee P(b)$, and the database were treated under the GCWA, one could not conclude $\neg P(a)$ from this database. The null value is then treated correctly. A disadvantage to this approach is that of having to list a potentially long disjunction for a database with many constants in its domain.

Another major difference is that the deduction mechanism needs to be more complex for indefinite databases than for definite deductive databases. Indeed, proof strategies such as input resolution [Chang 1970] and LUSH resolution [Hill 1974], although complete for Horn clauses, are not complete for non-Horn clauses. When dealing with indefinite clauses, a complete proof strategy such as linear resolution [Loveland 1969, 1970; Reiter 1971], linear resolution with selection function (SL) [Kowalski and Kuehner 1971], or linear resolution with unrestricted selection function based on trees (LUST) [Minker and Zanon 1982] is required.

Answers in an indefinite deductive database are no longer definite. That is, if the entire database consists of the single entry $P(a) \vee P(b)$, then the answer to the query $P(x)$? is $x = a + b$, which denotes that either $x = a$ or $x = b$ or both satisfy the query. The problem of indefinite answers is addressed by Reiter [1978b] and Grant and Minker [1983].

Minker and Perlis [1983, 1984] treat a different kind of indefiniteness. Users may know that some of their facts are correct. They may also know that they are not willing to make statements about other facts. That is, the facts may or may not be true. In either a conventional or a deductive database there is no facility available to store facts and specify to the user that these facts may or may not be true. To account for this possibility Minker and Perlis have generalized the concept of circumscription developed by McCarthy [1980] to what they call protected circumscription. They have shown that the user can then obtain answers of the form yes, no, and unknown. Efficient computational techniques are required to make the approach practical. They have also developed a completeness and soundness result for protected circumscription in the case of finitary data, and hence for cases of interest to databases.

2.4 Logic Databases

The presentation of deductive databases given in the preceding sections essentially reflects a view from the perspective of conventional databases, that is, a "DB field" view. Starting with a conventional DB, one introduces some ad hoc deductive capabilities while keeping (as far as possible) the usual DB conventions. For example, only function-free formulas are retained, but clearly, authorizing functions allows more general forms of data to be manipulated (i.e., general terms instead of only constants and variables), which eventually give a different conceptual model to the user (e.g., semantic networks; see Deliyanni and Kowalski [1979] for a discussion as to how semantic networks can be represented in logic). We do not consider an (extended) deductive database as an unconstrained first-order theory because its implementation would be extremely inefficient.

Introducing functions into DB Horn clauses takes the deductive database field closer to the field of logic programming. It is thus not surprising to see the same enhancements carried out for both fields. Horn clauses augmented with negation as failure led to the PROLOG language, which has been demonstrated to be efficient [Roussel 1975]. A PROLOG program is quite similar to a definite deductive database (up to functions). However, this does not mean that a standard interpreter for such a language constitutes a DDBMS that must not only provide us with query facilities, but also with functions for integrity and maintenance of deduced facts. A logic database system would be obtained by combining the above-mentioned facilities with an efficient access method to a large number of facts. Such an integration can be realized in various ways (see Chakravarthy et al. [1982] and Gallaire [1983]). It should be clear that a logic database language could be continuously extended, by providing extensions to negation as failure, incorporating metalanguage capabilities and other capabilities. This could prove useful for databases, as argued by Kowalski [1981a].

3. CONCLUSION

We have attempted to cover results obtained within the framework of mathematical logic applied to databases mainly through the perspective of deductive databases. We have shown how logic applies to query languages, integrity modeling and maintenance, query evaluation, database design through dependencies, representation and manipulation of deduced facts, and incomplete information. However, the field of logic and databases, as it is called, is far from closed; logic provides an appropriate framework for many database problems that still need to be investigated thoroughly. We note some of these problems below. Many of the problems listed below as needing continued research have been drawn from a report to which the first and third authors of this paper contributed [Adiba et al. 1982] (see also Reiter [1984]):

- *Designing natural language query systems*, whatever "natural" means.
- *Optimizing query evaluation based on semantic knowledge*, which is needed, in general, for interactive access to databases and especially in a natural language context.
- *Finding criteria and methods for choosing, between equivalent sets of integrity constraints, a good set*, where "good" means constraint sets that are easy to check and to maintain.
- *Finding criteria to decide which relations should be base relations and which should be derived or hybrid relations*, in other words, what general knowledge should be used as integrity constraints and what should be used as deduction rules.
- *Finding more efficient means for detecting the violation of integrity constraints.*
- *Synthesizing a program preserving integrity from transaction specifications and integrity constraints*, both expressed in logic. Since such a program could also be written in logic (programming), the field of logic and databases appears to be particularly well suited here.
- *Relaxing some of the conditions for a formula to be an integrity constraint* and investigating the interest of such less stringent definitions.
- *Embedding data manipulation languages in programming languages.* In deductive databases one usually considers the deductive component to be part of the DBMS. However, it is possible to interface a (conventional or deductive) DBMS with a logic programming language, for example, PROLOG. Such an integration is accomplished most easily when the DBMS language is predicate calculus oriented; a full integration could result in which the DBMS appears as the part of the programming system specialized to the manipulation of facts.
- *Looking for practical solutions to handle general forms of incomplete information.* Although satisfactory solutions have been found for some incomplete information problems, major developments are needed to be able to handle practical problems. It will be necessary to investigate the concepts of circumscription and protected circumscription to expand their applicability to databases. The investigation of incomplete information is intimately connected with null value problems.
- *Investigating how logic can help in defining the so-called semantic models*, which appear as competitors to the relational model both for classical database applications and for more ambitious applications, where various types of data must be handled (e.g., text, computer-assisted design data, graphics). Such research will pursue problems identical to those in knowledge representation.

As we have attempted to demonstrate in this survey article, the field of logic and databases is important both to conventional and deductive databases. In this connection we note that "logic and databases" as have been described in this paper constitute the core of the work in Japan in the field of knowledge bases in their "Fifth-Generation Project."

Logic, we believe, provides a firm theoretical basis upon which one can pursue database theory in general. There are many research areas that remain to be investigated in addition to those listed above before a full understanding of databases is achieved. We believe that the field of logic and databases will contribute significantly to such an understanding. At the same time, we believe that logic databases may be made practical and efficient, as has been described by the many developments reported on in this survey.

ACKNOWLEDGMENTS

We thank Bob Kowalski and Ray Reiter, whose work in the field of logic and databases has been very influential, and for the many discussions that we have had with them over the past several years. The comments of the referees, the ACM technical editor, and Joachim Biskup on an earlier version of this paper were greatly appreciated.

Support for the work on this paper was received by Hervé Gallaire from Compagnie Générale d'Électricité (CGE), by Jack Minker from the Air Force Office of Scientific Research (AFOSR) under Grant 82-0303, the National Aeronautics and Space Administration (NASA) under Grant NAG-1-51, and the National Science Foundation (NSF) under Grants MCS-7919418 and MCS-8305992, by Jean-Marie Nicolas from the Direction des Recherches, Études et Techniques d'Armements (DRET), the Centre National de la Recherche Scientifique (CNRS-APT Intelligence Artificielle), and the Institut National de la Recherche en Informatique et Automatique (INRIA). We gratefully acknowledge the support that made this work possible.

REFERENCES

ADIBA, M., et al. 1982. Bases de donnes: Nouvelles perspectives. Rapport du groupe BD3 ADI-INRIA, Paris.

AHO, A. V., AND ULLMAN, J. D. 1979. Universality of data retrieval languages. In *Proceedings of the 6th ACM Symposium on Principles of Programming Languages* (San Antonio, Tex., Jan. 29–31). ACM, New York, pp. 110–120.

AHO, A. V., SAGIV, Y., AND ULLMAN, J. D. 1979. Equivalences among relational expressions. *SIAM J. Comput. 8*, 2 (May), 218–246.

ANSI/X3/SPARC 1975. Study Group on DBMS Interim Report. *SIGMOD FDT Bull. 7*, 2, 1975.

ARMSTRONG, W. W. 1974. Dependency structures of database relationships. In *Proceedings of IFIP 74*. Elsevier North-Holland, New York, pp. 580–583.

ARTIFICIAL INTELLIGENCE JOURNAL 1980. *13*, 1, 2; Special issue on nonmonotonic logic.

ARTRAUD, A., AND NICOLAS, J.-M. 1974. An experimental query system: SYNTEX. In *Proceedings of the International Computing Symposium 73*. Elsevier North-Holland, New York, pp. 557–563.

BANCILHON, F. 1978. On the completeness of query languages for relational databases. In *Proceedings of the 7th Symposium on Mathematical Foundations of Computer Science*. Springer-Verlag, Berlin and New York, pp. 112–123.

BEERI, C., AND VARDI, M. Y. 1980. A proof procedure for data dependencies. Tech. Rep., Computer Science Dept., Hebrew Univ., Jerusalem (Aug.).

BEERI, C., AND VARDI, M. Y. 1981. The implication problem for data dependencies. In *Proceedings of the 8th Colloquium on Automata, Languages, and Programming ACTC* (AKKO). Springer-Verlag, Berlin and New York, pp. 73–85.

BISKUP, J. A. 1981. A formal approach to null values in database relations. In *Advances in Database Theory*, vol. 1, H. Gallaire, J. Minker, and J.-M. Nicolas, Eds. Plenum, New York, pp. 299–341.

BISKUP, J. A. 1982. A foundation of Codd's relational maybe-operations. Tech. Rep., Computer Science Dept., Univ. of Dortmund, West Germany.

BLAUSTEIN, B. T. 1981. Enforcing database assertions: Techniques and applications, Ph.D. dissertation, Computer Science Dept., Harvard Univ., Cambridge, Mass. (Aug.).

BORGIDA, A., AND WONG, H. K. T. 1981. Data models and data manipulation languages: Complementary semantics and proof theory. In *Proceedings of the 7th Conference on Very Large Data Bases* (Cannes, France, Sept. 9–11). IEEE, New York, pp. 260–271.

BOSSU, G., AND SIEGEL, P. 1981. La saturation au secours de la non-monotonicite. These de 3eme Cycle, Départment d'Informatiques, Université d'Aix-Marseille-Luminy, Marseille, France (June).

BOWEN, K. A., AND KOWALSKI, R. A. 1982. Amalgamating language and metalanguage in logic programming. In *Logic Programming*, K. L. Clark and S. A. Tarnlund, Eds. Academic Press, New York, pp. 153–172.

BRODIE, M. L., AND ZILLES, S. N., Eds. 1980. *Proceedings of the Workshop on Data Abstraction, Databases and Conceptual Modeling* (Pingree Park, Colo., June). *ACM SIGMOD Rec. 11*, 2 (Feb.).

CASANOVA, M. A., AND BERNSTEIN, P. A. 1979. The logic of a relational data manipulation language. In *Proceedings of the 6th ACM Symposium on*

Principles of Programming Languages (San Antonio, Tex., Jan. 29–31). ACM, New York, pp. 101–109.

CASANOVA, M. A., AND BERNSTEIN, P. A. 1980. A formal system for reasoning about programs accessing a relational database. *ACM Trans. Program. Lang. Syst. 2,* 3 (July), 386–414.

CHAKRAVARTHY, U. S., MINKER, J., AND TRAN, D. 1982. Interfacing predicate logic languages and relational databases. In *Proceedings of the 1st Conference on Logic Programming* (Marseille, France, Sept.). Université d'Aix-Marseille-Luminy, Marseille, France, pp. 91–98.

CHAMBERLIN, D. D., ASTRAHAN, M. M., BLASGEN, M. W., GRAY, J. N., KING, W. F., LINDSAY, B. G., LORIE, R., MEHL, J. W., PRICE, T. G., PUTZOLU, F., SELINGER, P. G., SCHKOLNICK, M., SLUTZ, D. R., TRAIGER, I. L., WADE, B. W., AND YOST, R. A. 1981. A history and evaluation of System R. *Commun. ACM 24,* 10 (Oct.) 632–646.

CHANDRA, A. K., AND HAREL, D. 1979. Computable queries for relational data bases. In *Proceedings of the 11th ACM Symposium on Theory of Computing* (Atlanta, Ga., Apr. 30–May 2). ACM, New York, pp. 309–318.

CHANDRA, A. K., AND HAREL, D. 1980. Structure and complexity of relational queries. In *Proceedings of the 21st IEEE Symposium on Foundations of Computer Science* (Syracuse, N.Y., Oct.). IEEE, New York, pp. 333–347. Also in *J. Comput. Syst. Sci. 25,* 1 (Aug. 1982), 99–128.

CHANDRA, A. K., AND MERLIN, P. M. 1976. Optimal implementation of conjunctive queries in relational data bases. In *Proceedings of the 9th ACM Symposium on Theory of Computing* (Boulder, Colo., May 2–4). ACM, New York, pp. 77–90.

CHANDRA, A. K., LEWIS, H. R., AND MAKOWSKY, J. A. 1981. Embedded implicational dependencies and their inference problem. In *Proceedings of the 13th ACM Symposium on Theory of Computing* (Milwaukee, Wis., May 11–13). ACM, New York, pp. 342–354.

CHANG, C. L. 1970. The unit proof and the input proof in theorem proving. *J. ACM 17,* 4 (Oct.) 698–707.

CHANG, C. L. 1978. DEDUCE 2: Further investigations of deduction in relational databases. In *Logic and Databases,* H. Gallaire and J. Minker, Eds. Plenum, New York, pp. 201–236.

CHANG, C. L. 1981. On evaluation of queries containing derived relations in a relational database. In *Advances in Database Theory,* vol. 1, H. Gallaire, J. Minker, and J.-M. Nicolas, Eds. Plenum, New York, pp. 235–260.

CHANG, C. L., AND LEE, R. C. T. 1973. *Symbolic Logic and Mechanical Theorem Proving.* Academic Press, New York.

CLARK, K. L. 1978. Negation as failure. In *Logic and Databases,* H. Gallaire and J. Minker, Eds. Plenum, New York, pp. 293–322.

CODD, E. F. 1970. A relational model of data for large shared data banks. *Commun. ACM 13,* 6 (June), 377–387.

CODD, E. F. 1972. Relational completeness of database sublanguages. In *Data Base Systems,* R. Rustin, Ed. Prentice-Hall, New York, pp. 65–98.

CODD, E. F. 1979. Extending the relational database model to capture more meaning. *ACM Trans. Database Syst. 4,* 4 (Dec.), 397–434.

CODD, E. F. 1980. Data models in database management. In *Proceedings of the Workshop on Data Abstraction, Databases and Conceptual Modeling* (Pingree Park, Colo., June), pp. 112–114; *ACM SIGMOD Rec. 11,* 2 (Feb.).

CODD, E. F. 1982. Relational database: A practical foundation for productivity. *Commun. ACM 25,* 2 (Feb.), 109–117.

COLMERAUER, A. 1973. Un systeme de communication homme–machine en francais. Rapport, Groupe Intelligence Artificielle, Université d'Aix-Marseille-Luminy, Marseilles, France.

COLMERAUER, A., AND PIQUE, J. F. 1981. About natural logic. In *Advances in Database Theory* vol. 1, H. Gallaire, J. Minker, and J.-M. Nicolas, Eds. Plenum, New York, pp. 343–365.

COOPER, E. C. 1980. On the expressive power of query languages for relational databases. Tech. Rep. 14-80, Computer Science Dept., Harvard Univ., Cambridge, Mass.

DAHL, V. 1982. On database systems development through logic. *ACM Trans. Database Syst. 7,* 1 (Mar.), 102–123.

DATE, C. J. 1977. *An Introduction to Database Systems.* Addison-Wesley, Reading, Mass.

DATE, C. J. 1981. *An Introduction to Database Systems,* 3rd ed. Addison-Wesley, Reading, Mass.

DELIYANNI, A., AND KOWALSKI, R. A. 1979. Logic and semantic networks. *Commun. ACM 22,* 3 (Mar.), 184–192.

DELOBEL, C. 1978. Normalization and hierarchical dependencies in the relational data model. *ACM Trans. Database Syst. 3,* 3 (Sept.), 201–222.

DELOBEL, C. 1980. An overview of the relational data theory. In *Proceedings of IFIP 80.* Elsevier North-Holland, New York, pp. 413–426.

DELOBEL, C., AND CASEY, R. G. 1973. Decomposition of a database and the theory of Boolean switching functions. *IBM J. Res. Dev. 17,* 5 (Sept.), 484–485.

DELOBEL, C., AND PARKER, D. S. 1978. Functional and multivalued dependencies in a relational database and the theory of boolean switching functions. Tech. Rep. 142, Université de Grenoble, Grenoble, France (Nov.).

DEMOLOMBE, R. 1980. Estimation of the number of tuples satisfying a query expressed in predicate calculus language. In *Proceedings of the 6th International Conference on Very Large Data Bases* (Montreal, Oct. 1–3). IEEE, New York, pp. 55–63.

DEMOLOMBE, R. 1981. Assigning meaning to ill-defined queries expressed in predicate calculus language. In *Advances in Database Theory*, vol. 1, H. Gallaire, J. Minker, and J.-M. Nicolas, Eds. Plenum, New York, pp. 367–395.

DEMOLOMBE, R. 1982. Utilization du calcul des predicats comme langage d'interrogation des bases de données. These de doctorat d'état, ONERA-CERT, Toulouse, France (Feb.).

DI PAOLA, R. A. 1969. The recursive unsolvability of the decision problem for the class of definite formulas. *J. ACM 16*, 2 (Apr.), 324–327.

DOS SANTOS, C. S., MAIBAUM, T. S. E., AND FURTADO, A. L. 1981. Conceptual modeling of database operations. *Int. J. Comput. Inf. Sci. 10*, 5, 299–314.

ENDERTON, H. B. 1972. *A Mathematical Introduction to Logic.* Academic Press, New York.

FAGIN, R. 1977a. Multivalued dependencies and a new normal form for relational databases. *ACM Trans. Database Syst. 2*, 3 (Sept.), 262–278.

FAGIN, R. 1977b. Functional dependencies in a relational database and propositional logic. *IBM J. Res. Dev. 21*, 6 (Nov.), 534–544.

FAGIN, R. 1980. Horn clauses and data base dependencies. In *Proceedings of the 12th Annual ACM-SIGACT Symposium on Theory of Computing.* ACM, New York, pp. 123–134.

FAGIN, R. 1982. Horn clause and database dependencies. *J. ACM 29*, 4 (Oct.), 952–985.

FAGIN, R., ULLMAN, J. D., AND VARDI, M. Y. 1983. On the semantics of updates in databases. In *Proceedings of the 2nd ACM Symposium on Principles of Database Systems* (Atlanta, Ga., Mar. 21-23). ACM, New York, pp. 352–365.

FLORENTIN, J. J. 1974. Consistency auditing of data bases. *Comput. J. 17*, 1, 52–58.

FURUKAWA, K. 1977. A deductive question-answering system on relational databases. In *Proceedings of the 5th International Joint Conference on Artificial Intelligence* (Cambridge, Mass., Aug.), pp. 59–66.

GALIL, Z. 1982. An almost linear-time algorithm for computing a dependency basis in a relational database. *J. ACM 29*, 1 (Jan.), 96–102.

GALLAIRE, H. 1981. Impacts of logic on data bases. In *Proceedings of the 7th International Conference on Very Large Data Bases* (Cannes, France, Sept. 9-11). IEEE, New York, pp. 248–259.

GALLAIRE, H. 1983. Logic databases vs. deductive databases. *Logic Programming Workshop* (Albufeira, Portugal). University of Lisboa, Lisbon, Portugal, pp. 608–622.

GALLAIRE, H., AND MINKER, J., Eds. 1978. *Logic and Data Bases.* Plenum, New York.

GALLAIRE, H., MINKER, J., AND NICOLAS, J.-M., Eds., 1981a. *Advances in Data Base Theory*, vol. 1. Plenum, New York.

GALLAIRE, H., MINKER, J., AND NICOLAS, J.-M. 1981b. Background for advances in data base theory. In *Advances in Data Base Theory*, vol. 1, H. Gallaire, J. Minker, and J.-M. Nicolas, Eds. Plenum, New York, pp. 3–21.

GALLAIRE, H., MINKER, J., AND NICOLAS, J.-M., Eds. 1984. *Advances in Data Base Theory*, vol. 2. Plenum, New York.

GARDARIN, G., AND MELKANOFF, M. 1979. Proving consistency of database transactions. In *Proceedings of the 5th International Conference on Very Large Data Bases* (Rio de Janeiro, Oct. 3–5). IEEE, New York, pp. 291-298.

GRANT, J. 1977. Null values in a relational data base. *Inf. Process. Lett. 6*, 5, 156–157.

GRANT, J., AND JACOBS, B. E. 1982. On the family of generalized dependency constraints. *J. ACM 29*, 4 (Oct.), 986–997.

GRANT, J., AND MINKER, J. 1981. Optimization in deductive and conventional relational database systems. In *Advances in Data Base Theory*, vol. 1, H. Gallaire, J. Minker, and J.-M. Nicolas, Eds. Plenum, New York, pp. 195–234.

GRANT, J., AND MINKER, J. 1983. Answering queries in indefinite databases and the null value problem. Tech. Rep. 1374, Computer Science Dept., University of Maryland, College Park.

GREEN, C. 1969. Theorem proving by resolution as a basis for question-answering systems. In *Machine Intelligence 4*, B. Meltzer and D. Michie, Eds. Elsevier North-Holland, New York, pp. 183–205.

HAMMER, M. T., AND ZDONIK, S. B., JR. 1980. Knowledge-based query processing. In *Proceedings of the 6th International Conference on Very Large Data Bases* (Montreal, Oct. 1–3). IEEE, New York, pp. 137–147.

HAREL, D. 1979. First order dynamic logic. In *Lecture Notes in Computer Science*, vol. 68. Springer-Verlag, Berlin and New York.

HAREL, D. 1980. Review of *Logic and Databases*, H. Gallaire and J. Minker. *Comput. Rev. 21*, 8 (Aug.), 367–369.

HENSCHEN, L. J., MCCUNE, W. W., AND NAQVI, S. A. 1984. Compiling constraint checking programs from first order formulas. In *Advances in Database Theory*, vol. 2, H. Gallaire, J. Minker, and J.-M. Nicolas, Eds. Plenum, New York, pp. 145–169.

HEWITT, C. 1972. Description and theoretical analysis (using schemata) of PLANNER: A language for proving theorems and manipulating models in a robot. AI Memo No. 251, MIT Project MAC, Cambridge, Mass.

HILL, R. 1974. LUSH resolution and its completeness. DCS Memo, No. 78. University of Edinburgh, School of Artificial Intelligence.

HOMEIER, P. V. 1981. Simplifying integrity constraints in a relational database: An implementation. M.Sc. thesis, Computer Science Dept., University of California, Los Angeles.

IMIELINSKI, T., AND LIPSKI, W. 1981. On representing incomplete information in a relational database. In *Proceedings of the 7th International Conference on Very Large Data Bases* (Cannes, France, Sept. 9–11). IEEE, New York, pp. 389–397.

INTERNATIONAL JOINT CONFERENCE ON ARTIFICIAL INTELLIGENCE 1983. August, 8–12, 1983, Karlsruhe, West Germany.

INTERNATIONAL SYMPOSIUM ON LOGIC PROGRAMMING 1984. Feb. 6-9, 1984, Atlantic City, N.J.

JACOBS, B. E. 1982. On database logic. *J. ACM 29,* 2 (Apr.), 310-332.

JACOBS, B. E., ARONSON, A. R., AND KLUG, A. C. 1982. On interpretations of relational languages and solutions to the implied constraint problem. *ACM Trans. Database Syst. 7,* 2 (June), 291–315.

JAFFAR, J., LASSEZ, J. L., AND LLOYD, J. 1983. Completeness of the negation by failure rule. In *Proceedings of the 8th International Joint Conference on Artificial Intelligence* (Karlsruhe, W. Germany, Aug.), pp. 500–506.

JANAS, J. M. 1979. Towards more informative user interfaces. In *Proceedings of the 5th International Conference on Very Large Data Bases* (Rio de Janeiro, Oct. 3–5). IEEE, New York, pp. 17–23.

JANAS, J. M. 1981. On the feasibility of informative answers. In *Advances in Data Base Theory,* vol. 1, H. Gallaire, J. Minker, and J.-M. Nicolas, Eds. Plenum, New York, pp. 397–414.

KELLOGG, C., AND TRAVIS, L. 1981. Reasoning with data in a deductively augmented data management system. In *Advances in Data Base Theory,* vol. 1, H. Gallaire, J. Minker, and J.-M. Nicolas, Eds. Plenum, New York, pp. 261–295.

KELLOGG, C., KLHAR, P., AND TRAVIS, L. 1978. Deductive planning and path finding for relational data bases. In *Logic and Databases,* H. Gallaire and J. Minker, Eds. Plenum, New York, pp. 179–200.

KING, J. J. 1981. QUIST: A system for semantic query optimization in relational databases. In *Proceedings of the 7th International Conference on Very Large Data Bases* (Cannes, France, Sept. 9–11). IEEE, New York, pp. 510–517.

KONOLIGE, K. 1981. A metalanguage representation of databases for deductive question-answering systems. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence* (Vancouver, B.C., Aug.), pp. 469–503.

KOWALSKI, R. A. 1978. Logic for data description. In *Logic and Data Bases,* H. Gallaire and J. Minker, Eds. Plenum, New York, pp. 77–103.

KOWALSKI, R. A. 1979. *Logic for Problem Solving.* Elsevier North-Holland, New York.

KOWALSKI, R. A. 1981a. Logic as a data base language. In *Proceedings of the Advanced Seminar on Theoretical Issues in Data Bases* (Cetraro, Italy, Sept.).

KOWALSKI, R. A. 1981b. Personal communication.

KOWALSKI, R. A., AND KUEHNER, D. 1971. Linear resolution with selection function. *Artif. Intell. 2,* 3/4, 227–260.

KUHNS, J. L. 1967. Answering questions by computers—A logical study. Rand Memo RM 5428 PR, Rand Corp., Santa Monica, Calif.

KUHNS, J. L. 1970. Interrogating a relational data file: Remarks on the admissibility of input queries. Tech. Rep. TR-511-PR, Rand Corp., Santa Monica, Calif. (Nov.).

KUNIFUJI, S., AND YOKOTA, H. 1982. Prolog and relational databases for the fifth generation computer system. ICOT Rep. D02, Institute for New Generation Computer Technology, Tokyo.

LACROIX, M., AND PIROTTE, A. 1980. Associating types with domains of relational databases. In *Workshop on Data Abstraction, Databases and Conceptual Modeling* (Pingree Park, Colo., June). *ACM SIGMOD Rec. 11,* 2 (Feb.), 144–146.

LEVESQUE, H. J. 1981. The interaction with incomplete knowledge bases: a formal treatment. *Proceedings of the 7th International Joint Conference on Artificial Intelligence* (Vancouver, B.C., Aug.), pp. 240–245.

LIPSKI, W. 1979. On semantic issues connected with incomplete information systems. *ACM Trans. Database Syst. 4,* 3 (Sept.), 262–296.

LOGIC PROGRAMMING WORKSHOP PROCEEDINGS 1983. June 26–July, 1, 1983, Praia da Falesia, Algarve, Portugal. Dept. of Computer Science, University of Lisboa, Lisbon, Portugal.

LOVELAND, D. 1969. Theorem provers combining model elimination and resolution. In *Machine Intelligence,* vol. 4. B. Meltzer and D. Michie, Eds. Elsevier North-Holland, New York, pp. 73–86.

LOVELAND, D. 1970. A linear format for resolution. *Proceedings of the IRIA Symposium on Automatic Demonstration.* Springer-Verlag, Berlin and New York, pp. 147–162.

LOVELAND, D. 1978. *Automated Theorem Proving: A Logical Basis.* Elsevier North-Holland, New York.

MAIER, D. 1983. *The Theory of Relational Databases.* Computer Science Press, Rockville, Md.

MAIER, D., MENDELZON, A. O., AND SAGIV, Y. 1979. Testing implications of data dependencies. *ACM Trans. Database Syst. 4,* 4 (Dec.), 455–469.

MCCARTHY, J. 1980. Circumscription—A form of non-monotonic reasoning. *Artif. Intell. 13,* 27–39.

MCSKIMIN, J. 1976. Techniques for employing semantic information in question-answering systems. Ph.D. dissertation, Dept. of Computer Science, University of Maryland, College Park.

MCSKIMIN, J., AND MINKER, J. 1977. The use of a semantic network in a deductive question-answering system. In *Proceedings of the 5th International Joint Conference on Artificial Intelligence* (Cambridge, Mass., Aug.), pp. 50–58.

MENDELSON, E. 1978. *Introduction to Mathematical Logic,* 2nd ed. Van Nostrand-Reinhold, New York.

MINKER, J. 1975a. Performing inferences over relational data bases. In *ACM SIGMOD International Conference on Management of Data* (San Jose, Calif., May 14-16). ACM, New York, pp. 79-91.

MINKER, J. 1975b. Set operations and inferences over relational databases. In *Proceedings of the*

4th Texas Conference on Computing Systems (Nov.). Univ. of Texas, Austin, pp. 5A1.1–5A1.10.

MINKER, J. 1978a. Search strategy and selection function for an inferential relational system. *ACM Trans. Database Syst. 3,* 1 (Mar.), 1–31.

MINKER, J. 1978b. An experimental relational data base system based on logic. In *Logic and Databases,* H. Gallaire and J. Minker, Eds. Plenum, New York, pp. 107–147.

MINKER, J. 1982. On indefinite databases and the closed world assumption. In *Proceedings of the 6th Conference on Automated Deduction* (New York). Springer-Verlag Lecture Notes in Computer Science, No. 138. Springer-Verlag, Berlin and New York, pp. 292–308.

MINKER, J. 1983. On deductive relational databases. In *Proceedings of the 5th International Conference on Collective Phenomena* (July), J. L. Lebowitz, Ed., *Annals of the New York Academy of Sciences,* vol. 10. New York Academy of Science, New York, pp. 181–200.

MINKER, J., AND NICOLAS, J. M. 1982. On recursive axioms in deductive databases. *Inf. Syst. 8,* 1 (Jan.), 1–13.

MINKER, J., AND PERLIS, D. 1983. On the semantics of circumscription. Tech. Rep., Computer Science Dept., Univ. of Maryland, College Park.

MINKER, J., AND PERLIS, D. 1984. Applications of protected circumscription. In *Proceedings of the Conference on Automated Deduction 7* (Napa, Calif., May). Springer-Verlag, Berlin and New York, pp. 414–425.

MINKER, J., AND ZANON, G. 1982. An extension to linear resolution with selection function. *Inf. Process. Lett. 14,* 4 (June), 191–194.

MOORE, R. C. 1981. Problems in logical form. In *Proceedings of the 19th Annual Meeting of the Association for Computational Linguistics* (June). Association for Computational Linguistics, pp. 117–124.

MYLOPOULOS, J. 1980. An overview of knowledge representation. In *Proceedings of the Workshop of Data Abstraction, Databases, and Conceptual Modeling,* M. Brodie and S. N. Zilles, Eds. (Pingree Park, Colo., June). *ACM SIGMOD Rec. 11,* 2 (Feb.), 5–12.

NAQVI, S. A., AND HENSCHEN, L. J. 1980. Performing inferences over recursive data bases. In *Proceedings of the 1st Annual National Conference on Artificial Intelligence* (Stanford, Conn., Aug.). AAAI, Palo Alto, Calif., pp. 263–265.

NAQVI, S. A., AND HENSCHEN, L. J. 1984. On compiling queries in recursive first-order databases. *J. ACM 31,* 1 (Jan.), 47–85.

NICOLAS, J.-M. 1978. First order logic formalization for functional, multivalued and mutual dependencies. In *Proceedings of ACM-SIGMOD International Conference on Management of Data* (Austin, Tex., June 1, 2). ACM, New York, pp. 40–46.

NICOLAS, J.-M. 1979a. A property of logical formulas corresponding to integrity constraints on database relations. In *Proceedings of the Workshop on Formal Bases for Data Bases* (Toulouse, France). ONERA-CERT, Toulouse, France.

NICOLAS, J.-M. 1979b. Logic for improving integrity checking in relational databases. Tech. Rep. ONERA-CERT, Toulouse, France (Feb.). Also in *Acta Inf. 18,* 3 (Dec.), 227–253.

NICOLAS, J.-M., AND GALLAIRE, H. 1978. Database: Theory vs. interpretation. In *Logic and Databases,* H. Gallaire and J. Minker, Eds. Plenum, New York, pp. 33–54.

NICOLAS, J.-M., AND SYRE, J. C. 1974. Natural question answering and automatic deduction in the system SYNTEX. In *Proceedings of IFIP 1974.* North-Holland, Amsterdam, pp. 595–599.

NICOLAS, J.-M., AND YAZDANIAN, K. 1978. Integrity checking in deductive databases. In *Logic and Databases,* H. Gallaire and J. Minker, Eds. Plenum, New York, pp. 325–346.

NICOLAS, J.-M., AND YAZDANIAN, K. 1982. An outline of BDGEN: A deductive DBMS. Tech. Rep. TR-ONERA-CERT, Toulouse, France (Oct.). Also in *Proceedings of IFIP 83 Congress.* North-Holland, Amsterdam, 1983, pp. 711–717.

PAREDAENS, J. 1978. On the expressive power of relational algebra. *Inf. Process. Lett. 7,* 2 (Feb.), 107–111.

PARKER, D. S., AND DELOBEL, C. 1979. Algorithmic applications for a new result on multivalued dependencies. In *Proceedings of the 5th Conference on Very Large Data Bases* (Rio de Janeiro, Oct. 3–5). IEEE, New York, pp. 67–74.

PIROTTE, A. 1976. Explicit description of entities and their manipulation in languages for the relational database model. Thèse de doctorat, Université libre de Bruxelles, Brussells, Belgium (Dec.).

PIROTTE, A. 1978. High level data base query languages. In *Logic and Data Bases,* H. Gallaire and J. Minker, Eds. Plenum, New York, pp. 409–436.

PRATT, V. R. 1976. Semantical considerations on Floyd–Hoare logic. In *Proceedings of the 17th IEEE Symposium Foundations of Computer Science* (Oct.). IEEE, New York, pp. 409–120.

PROCEEDINGS OF THE 1ST CONFERENCE ON LOGIC PROGRAMMING 1982. Marseille, France (Sept.), Université d'Aix-Marseille-Luminy, Marseille.

REITER, R. 1971. Two results on ordering for resolution with merging and linear format. *J. ACM 18,* 4 (Oct.), 630–646.

REITER, R. 1978a. On closed world databases. In *Logic and Databases,* H. Gallaire and J. Minker, Eds. Plenum, New York, pp. 56–76.

REITER, R. 1978b. Deductive question-answering on relational databases. In *Logic and Data Bases,* H. Gallaire and J. Minker, Eds. Plenum, New York, pp. 149–178.

REITER, R. 1978c. On structuring a first-order database. In *Proceedings of the 2nd Canadian Society for Computer Science National Conference,* Canada (July).

REITER, R. 1980. Equality and domain closure in first-order databases. *J. ACM 27,* 2 (Apr.), 235–249.

REITER, R. 1981. On the integrity of first-order databases. In *Advances in Data Base Theory*, vol. 1, H. Gallaire, J. Minker, and J.-M. Nicolas, Eds. Plenum, New York, pp. 137–158.

REITER, R. 1982. Circumscription implies predicate completion (sometimes). In *Proceedings of the American Association for Artificial Intelligence 82 Conference* (Pittsburgh, Pa., Aug.). AAAI, Menlo Park, Calif., pp. 418–420.

REITER, R. 1983. A sound and sometimes complete query evaluation algorithm for relational databases with null values. Tech. Rep. 83-11, Computer Science Dept., University of British Columbia, Canada (June).

REITER, R. 1984. Towards a logical reconstruction of relational database theory. In *On Conceptual Modeling*, M. Brodie, J. Mylopoulos, and J. W. Schmidt, Eds. Springer-Verlag, Berlin and New York.

ROBINSON, J. A. 1965. A machine oriented logic based on the resolution principle. *J. ACM 12*, 1 (Jan.), 23–41.

ROUSSEL, P. 1975. PROLOG: Manuel de reference et d'utilisation. Tech. Rep., Département d'Informatiques, Université d'Aix-Marseille-Luminy, Marseille, France (Sept.).

SADRI, F., AND ULLMAN, J. D. 1980. A complete axiomatization for a large class of dependencies in relational databases. In *Proceedings of the 17th Annual ACM-SIGACT Symposium on Theory of Computing*. ACM, New York, pp. 117–122.

SADRI, F., AND ULLMAN, J. D. 1982. Template dependencies: A large class of dependencies in relational databases and its complete axiomatization. *J. ACM 29*, 2 (Apr.), 363–372.

SAGIV, Y. 1980. An algorithm for inferring multivalued dependencies with an application to propositional logic. *J. ACM 27*, 2 (Apr.), 250–262.

SAGIV, Y., AND FAGIN, R. 1979. An equivalence between relational database dependencies and a subset of propositional logic. Res. Rep. RJ2500, IBM Research Laboratories, San Jose, Calif. (Mar.).

SAGIV, Y., DELOBEL, C., PARKER, D. S., JR., AND FAGIN, R. 1981. An equivalence between relational database dependencies and a subclass of propositional logic. *J. ACM 28*, 3 (July), 435–453.

SELINGER, P. G., ASTRAHAN, M. M., CHAMBERLIN, D. D., LORIE, R. A., AND PRICE, T. G. 1979. Access path selection in a relational database management system. In *Proceedings ACM-SIGMOD International Conference on Management of Data* (Boston, May 30–June 1). ACM, New York, pp. 23–34.

SHAPIRO, S. E., AND MCKAY, D. P. 1980. Inference with recursive rules. In *Proceedings of the 1st Annual National Conference on Artificial Intelligence*. AAAI, Palo Alto, Calif.

SIKLOSSY, L., AND LAURIERE, J.-L. 1982. Removing restrictions in the relational database model: an application of problem solving techniques. In *Proceedings of the American Association for Artificial Intelligence 82 Conference* (Pittsburgh, Pa., Aug.). AAAI, Menlo Park, Calif., pp. 310–313.

STONEBRAKER, M. R., WONG, E., AND KEEPS, P. 1976. The design and implementation of INGRES. *ACM Trans. Database Syst. 1*, 3 (Sept.), 189–222.

ULLMAN, J. D. 1980. *Principles of Database Systems*. Computer Science Press, Potomac, Md.

ULLMAN, J. D. 1982. *Principles of Database Systems*, 2nd ed. Computer Science Press, Potomac, Md.

VAN EMDEN, M. H. 1978. Computation and deductive information retrieval. In *Formal Description of Programming Concepts*, E. J. Neuhold, Ed. Elsevier North-Holland, New York, pp. 421–440.

VAN EMDEN, M. H., AND KOWALSKI, R. A. 1976. The semantics of predicate logic as a programming language. *J. ACM 23*, 4 (Oct.), 733–742.

VARDI, M. Y. 1981. The decision problem for database dependencies. *Inf. Process. Lett. 12*, 5 (Oct.), 251–254.

VASSILIOU, Y. 1979. Null values in data base management—A denotational semantics approach. In *Proceedings of the ACM-SIGMOD International Conference on the Management of Data* (Boston, May 30–June 1). ACM, New York, pp. 162–169.

VASSILIOU, Y. 1980. Functional dependencies and incomplete information. In *Proceedings of the 6th International Conference on Very Large Data Bases* (Montreal, Oct. 1–3). IEEE, New York, pp. 260–269.

VELOSO, P. A. S., DE CASTILHO, J. M. V., AND FURTADO, A. L. 1981. Systematic derivation of complementary specifications. In *Proceedings of the 7th International Conference on Very Large Data Bases* (Cannes, France, Sept. 9–11). IEEE, New York, pp. 409–421.

WARREN, D. H. D. 1981. Efficient processing of interactive relational database queries expressed in logic. In *Proceedings of the 7th International Conference on Very Large Data Bases* (Cannes, France, Sept. 9–11). IEEE, New York, pp. 272–281.

WOODS, W. A. 1967. Semantics for question-answering systems. Ph.D. dissertation, Rep. NSF-19, Aiken Computation Laboratory, Harvard University, Cambridge, Mass.

YANNAKAKIS, M., AND PAPADIMITRIOU, C. 1982. Algebraic dependencies. *J. Comput. Syst. Sci. 25*, 1 (Aug.), 2–41.

YOKOTA, H., KUNIFUJI, S., KAKUTA, T., MIYAZAKI, N., SHIBAYAMA, S., AND MURAKAMI, K. 1983. An enhanced inference mechanism for generating relational algebra queries. Tech. Rep. 026, ICOT Research Center, Institute for New Generation Computer Technology, Tokyo, Japan.

ZANIOLO, C. 1976. Analysis and design of relational schemata for data base system. Tech. Rep. UCLA Eng. 7769, Dept. of Computer Science, University of California, Los Angeles.

ZANIOLO, C. 1981. Incomplete database information and null values: An overview. In *Proceedings of the Advanced Seminar on Theoretical Issues in Data Bases* (Cetraro, Italy, Sept.).

ZLOOF, M. M. 1977. Query-by-example: A data base language. *IBM Syst. J. 16*, 4, 324–343.

Received February 1983; final revision accepted July 1984.