Towards a Logical Reconstruction of Relational Database Theory

Raymond Reiter
University of British Columbia

ABSTRACT *Insofar as database theory can be said to owe a debt to logic, the currency on loan is* model theoretic *in the sense that a database can be viewed as a particular kind of first order interpretation, and query evaluation is a process of truth functional evaluation of first order formulae with respect to this interpretation. It is this model theoretic paradigm which leads, for example, to many valued propositional logics for databases with null values.*

In this chapter I argue that a proof theoretic *view of databases is possible, and indeed much more fruitful. Specifically, I show how relational databases can be seen as special theories of first order logic, namely theories incorporating the following assumptions:*

1. *The domain closure assumption. The individuals occurring in the database are all and only the existing individuals.*
2. *The unique name assumption. Individuals with distinct names are distinct.*
3. *The closed world assumption. The only possible instances of a relation are those implied by the database.*

It will follow that a proof theoretic paradigm for relational databases provides a correct treatment of:

1. *Query evaluation for databases that have incomplete information, including null values.*
2. *Integrity constraints and their enforcement.*
3. *Conceptual modelling and the extension of the relational model to incorporate more real world semantics.*

1. Introduction

There is in our midst a small group of researchers whose devotion to logic and databases[1] is viewed with some perplexity by the majority of database theoreticians and practitioners. Their literature is peppered with obscure logical notation and theorems. As befits logicians, they claim privileged sovereignty over the Truth about databases. Can this cabal possibly be saying anything of interest to the database community?

Of course, everyone is at least dimly conscious of some logical debt owed by database theory, if only because the relational calculus relies on a first order language. What other outstanding logical loans are generally acknowledged? Well, a relational calculus query is a first order formula that is *evaluated* with respect to a database of facts. Since logic dictates that formulae have values (truth values) only with respect to *interpretations*, a database is commonly viewed as just that—a first order interpretation in the standard Tarskian sense. The value of a relational calculus query is determined by those instances of its free variables that make the query true with respect to the interpretation specified by the underlying database. This view of a database as a first order interpretation also neatly accommodates the concept of an integrity constraint. Insofar as one can view an integrity constraint as a first order formula, a database can be said to satisfy this constraint iff the constraint is true with respect to the database as interpretation. That is, given a set of integrity constraints, one cannot admit just any interpretation as a correct representation of one's domain of application; the interpretation must be a *model* (again, in the standard Tarskian sense) of the integrity constraints.

I think it is fair to say that, as far as database theoreticians conceive the field in logical terms, it is this *model theoretic* point of view that prevails. A database is a model of some set of integrity constraints, and a query is some formula to be evaluated with respect to this model. Now I invite you to survey the literature of the database logicians. You will, for the most part, find little mention of models and interpretations. Poor Tarski gets short shift here. And the relational algebra at best is granted footnote status. Instead, you will find most theoretical constructs couched in proof theoretic terms. A database is viewed as a set of first order formulae, not as a model. Queries are formulae to be proven, given the database as premises. Satisfaction of integrity constraints is defined in terms of consistency.[2] Considerable

[1] See, for example, [GM78].

[2] I shall provide a different definition in this chapter, but one which nevertheless is proof theoretic.

energy is invested in obtaining algorithms for efficiently finding proofs. In short, the logicians adopt a *proof theoretic* view of database theory.

What, then, is the preferred formal perspective on database theory—the model theoretic or the proof theoretic? Without a careful analysis, of course, one cannot say. This chapter presumes to provide such an analysis. My conclusion will be that both paradigms are reconcilable, but that the proof theoretic view is richer and more fruitful. More precisely, I shall show how, when given a model theoretic database *DB without null values*, one can transform *DB* into a suitable set of first order axioms, such that the resulting first order theory provides a proof theoretic characterization of query evaluation and integrity constraints. By itself this would not be a very exciting result. Curious perhaps, but not exciting. The idea bears fruit only in its capacity for generalization. For now that databases can be perceived as special kinds of first order theories, one can generalize these theories in order to provide answers to a variety of outstanding questions about databases:

1. How can the relational model be extended in order to incorporate more real world knowledge?
2. A number of null values have been proposed. What is their semantics?
3. What really are databases that have incomplete information?
4. What is the correct notion of an answer to a query in the presence of semantically rich databases such as those incorporating the features mentioned in 1-3 above?
5. For such semantically rich databases, what is an appropriate notion of an integrity constraint and what does it mean to satisfy a constraint?

My purpose in this chapter is to show how answers to these questions emerge in a very natural way from a proof theoretic characterization of database theory.

2. Databases and Logic: The Model Theoretic Perspective

This section outlines in some detail what I take to be the model theoretic paradigm in relational database theory.[3] To that end we require some formal preliminaries.

[3] Many of the ideas of this section, in particular the concept of a database as a first order model of a set of integrity constraints, derive from [NG78]. In effect, Sections 2.1-2.3 below formalize this concept.

2.1 First Order Languages

A first order language F is specified by a pair $(\mathcal{A}, \mathcal{W})$ where $\mathcal{A}$ is an alphabet of symbols and $\mathcal{W}$ is a set of syntactically well formed expressions called well formed formulae constructed using the symbols of $\mathcal{A}$. The rules for constructing the formulae of $\mathcal{W}$ are the same for all first order languages F; only the alphabet $\mathcal{A}$ may vary. $\mathcal{A}$ must contain symbols of the following kind, and only such symbols:

Variables: $x, y, z, x_1, y_1, z_1, \ldots$.
There must be infinitely many of these.

Constants: a, b, c, part17, acme, ...,
There may be 0 or more of these, possibly infinitely many.

Predicates: P, Q, R, *SUPPLIES*, *EMPLOYEE*, ...,
There must be at least one of these, possibly infinitely many. With each is associated an integer $n \geqslant 0$, its *arity*, denoting the number of arguments it takes.

Punctuation Signs: (,) ,.

Logical Constants: $\supset$ (implies), $\wedge$ (and), $\vee$ (or), $\sim$ (not), $\equiv$ (iff).

Notice that function symbols are not included in this alphabet. I omit them because their introduction leads to severe difficulties for database theory [REIT78a] (pp. 173-175). Fortunately, they are not required for a formal treatment of current ideas in databases.

With such an alphabet $\mathcal{A}$ in hand, we can construct a set of syntactically well formed expressions, culminating in a definition of the set $\mathcal{W}$ of well formed formulae, as follows:

Terms

A variable or a constant of $\mathcal{A}$ is a *term*.

Atomic Formulae

If P is an n-ary predicate of $\mathcal{A}$ and $t_1, \ldots, t_n$ are terms, then $P(t_1, \ldots, t_n)$ is an *atomic formula*. $P(t_1, \ldots, t_n)$ is a *ground* atomic formula iff $t_1, \ldots, t_n$ are all constants.

Well Formed Formulae

$\mathcal{W}$ is the smallest set such that:

1. An atomic formula is a well formed formula (wff).
2. If W_1 and W_2 are wffs, so also are $(W_1 \wedge W_2)$, $(W_1 \vee W_2)$, $(W_1 \supset W_2)$, $(W_1 \equiv W_2)$, $\sim W_1$.
3. If x is a variable and W is a wff, then $(x)(W)$ and $(Ex)(W)$ are wffs. Here (x) is a *universal quantifier* and (Ex) an *existential quantifier.*

For the purposes of formally defining a relational database, we won't require arbitrary first order languages; a suitable proper subset of these will do. Accordingly, define a first order language $F = (\mathcal{A}, \mathcal{W})$ to be a *relational language* iff $\mathcal{A}$ has the following properties:

1. There are only finitely many constants in $\mathcal{A}$, but at least one.
2. There are only finitely many predicates in $\mathcal{A}$.
3. Among the predicates of $\mathcal{A}$ there is a distinguished binary predicate $=$ which will function for us as equality.
4. Among the predicates of $\mathcal{A}$ there is a distinguished subset, possibly empty, of unary predicates. Such unary predicates are called *simple types.* Not all unary predicates of $\mathcal{A}$ need be simple types. Such simple types, together with boolean combinations of simple types, will, in part, model the concept of the domain of a relation as it arises in standard database theory.

Given a relational language $R = (\mathcal{A}, \mathcal{W})$ we can define the set of *types* of R as the smallest set such that:

1. A simple type of $\mathcal{A}$ is a type.
2. If τ_1 and τ_2 are types, so also are $(\tau_1 \wedge \tau_2)$, $(\tau_1 \vee \tau_2)$, $\sim\tau_1$.

For a relational language $R = (\mathcal{A}, \mathcal{W})$ it is convenient to define appropriate syntactically sugared *abbreviations* for certain of the wffs of $\mathcal{W}$, as follows:

If τ is a type, then

$$(x/\tau)(W) \text{ abbreviates } (x)(\tau(x) \supset W)$$
$$(Ex/\tau)(W) \text{ abbreviates } (Ex)(\tau(x) \wedge W)$$

where

1. If τ is $(\tau_1 \wedge \tau_2)$ then $\tau(x)$ is $(\tau_1(x) \wedge \tau_2(x))$.
2. If τ is $(\tau_1 \vee \tau_2)$ then $\tau(x)$ is $(\tau_1(x) \vee \tau_2(x))$.
3. If τ is $\sim\tau_1$ then $\tau(x)$ is $\sim\tau_1(x)$.

Here $(x/\tau)(W)$ should be read as: "For all x which are τ, W is the case," and $(Ex/\tau)(W)$ as: "there is an x, which is a τ, such that W is the case." Thus these *type restricted quantifiers* are meant to restrict the possible x's to just those which belong to the class τ. Notice that quantifiers may be restricted only by types, not by arbitrary predicates.

Example 2.1

If *MALE, EMPLOYEE, MANAGER, SUPPLIER,* and *PART* are simple types, the following are type restricted quantified wffs:

$$(x/SUPPLIER)\ (Ey/PART)SUPPLIES(x,y)$$

which abbreviates the ordinary wff

$$(x)[SUPPLIER(x) \supset (Ey)(PART(y) \wedge SUPPLIES(x,y))]$$

i.e., "Every supplier supplies at least one part."

$$(x/MALE \wedge EMPLOYEE \wedge \sim MANAGER)[DEPT(x, 13) \supset PENSION\text{-}PLAN(x)]$$

which abbreviates the ordinary wff

$$(x)[MALE(x) \wedge EMPLOYEE(x) \wedge \sim MANAGER(x) \supset [DEPT(x, 13) \supset PENSION\text{-}PLAN(x)]]$$

i.e., "All male employees of department 13 who are not managers belong to the pension plan."

In this example I have omitted a lot of parentheses on the assumption (correct, I hope) that you all know what these formulae *mean.* I shall continue this practice whenever no ambiguity will result.

2.2 The Semantics of First Order Languages

The objective here is to assign a precise meaning to each of the symbols of the alphabet $\mathcal{A}$ of a first order language $F = (\mathcal{A}, \mathcal{W})$ and, using this assignment as a basis, to define the truth values of arbitrary wffs in $\mathcal{W}$ constructed from these symbols. The required definitions are by now standard (see, for example, [MEND64]).

An *interpretation I for the first order language* $F = (\mathcal{A}, \mathcal{W})$ is a triple (D,K,E) where

1. D is a non empty set, called the *domain* of I, over which the variables of $\mathcal{A}$ are meant to range.
2. K is mapping from the constants of $\mathcal{A}$ into D (*i.e.*, for each constant c, $K(c) \in D$).
3. E is a mapping from the predicates of $\mathcal{A}$ into sets of tuples of elements of D (*i.e.*, for each n-ary predicate symbol P, $E(P) \subseteq D^n$). $E(P)$ is called the *extension* of P in the interpretation I.

Example 2.2

Consider a relational language $R = (\mathscr{A}, \mathscr{W})$ where the only predicates and constants of $\mathscr{A}$ are the following:

Predicates: $TEACHER(\cdot)$, $COURSE(\cdot)$, $STUDENT(\cdot)$, $TEACH(\cdot,\cdot)$, $ENROLLED(\cdot,\cdot)$, $=(\cdot,\cdot)$.

Simple Types: $TEACHER(\cdot)$, $COURSE(\cdot)$, $STUDENT(\cdot)$.

Constants: A, B, C, a, b, c, d, $CS100$, $CS200$, $P100$, $P200$.

Then the following defines an interpretation for R, with domain $\{A, B, C, a, b, c, d, CS100, CS200, P100, P200\}$:

TEACHER	COURSE	STUDENT	TEACH	ENROLLED	=	
A	CS100	a	A CS100	a CS100	A	A
B	CS200	b	A CS200	a P100	B	B
C	P100	c	B P100	b CS100	C	C
	P200	d	C P200	c P100	a	a
				d CS200	b	b
				d P200	c	c
					d	d
					CS100	CS100
					CS200	CS200
					P100	P100
					P200	P200

Here the tables define the extensions of the predicate symbols *TEACHER*, *COURSE*, *etc.* Notice that, strictly speaking, the domain elements A, B, C, *etc.*, are *not* the same as the constant symbols A, B, C, *etc.*, which are part of our alphabet of symbols. In effect, I have chosen to name the domain elements by the constant symbols. So think of the domain elements in the tables as coloured red, and the constant symbols as coloured black. In subsequent examples I shall freely name domain elements by constant symbols.

Now given an interpretation $I = (D,K,E)$ for a first order language $F = (\mathscr{A}, \mathscr{W})$, let ρ be a mapping from the variables of $\mathscr{A}$ into D (*i.e.*, for each variable $x \in \mathscr{A}$, $\rho(x) \in D$). ρ is called an *environment* for the variables of $\mathscr{A}$. For a given environment ρ, define a mapping:

$\|\cdot\|^{\rho}_{I}$: terms $\rightarrow D$ as follows:

$\|c\|^{\rho}_{I} = K(c)$ for each constant symbol $c \in \mathscr{A}$.

$\|x\|^{\rho}_{I} = \rho(x)$ for each variable $x \in \mathscr{A}$.

Next define a relation $\models_{I,\rho}$ by:

1. $\models_{I,\rho} P(t_1, \ldots, t_n)$ iff $(\|t_1\|^{\rho}_{I}, \ldots, \|t_n\|^{\rho}_{I}) \in E(P)$ for each atomic formula $P(t_1, \ldots, t_n) \in \mathscr{W}$.
2. $\models_{I,\rho} W_1 \wedge W_2$ iff $\models_{I,\rho} W_1$ and $\models_{I,\rho} W_2$.
3. $\models_{I,\rho} W_1 \vee W_2$ iff $\models_{I,\rho} W_1$ or $\models_{I,\rho} W_2$.
4. $\models_{I,\rho} \sim W$ iff not $\models_{I,\rho} W$.
5. $\models_{I,\rho} W_1 \supset W_2$ iff $\models_{I,\rho} \sim W_1 \vee W_2$.
6. $\models_{I,\rho} W_1 \equiv W_2$ iff $\models_{I,\rho} (W_1 \supset W_2) \wedge (W_2 \supset W_1)$.
7. $\models_{I,\rho} (x)W$ iff for all $d \in D$, $\models_{I,\rho[x\rightarrow d]} W$ where $\rho[x\rightarrow d]$ denotes an environment identical to ρ except that this new environment maps the variable x to the domain element d.
8. $\models_{I,\rho} (Ex)W$ iff $\models_{I,\rho} \sim(x)\sim W$.

Finally, define $\models_{I} W$ iff $\models_{I,\rho} W$ for all environments ρ, in which case W is said to be *true in the interpretation* I. W is *false in the interpretation* I iff for no environment ρ is it the case that $\models_{I,\rho} W$. An interpretation I is a *model* of the wff W iff W is true in I. I is a *model* of a set S of wffs iff W is true in I for each $W \in S$.

Example 2.2 (continued)

The previous interpretation is a model for each of the following formulae:

$$(x)(y)[TEACH(x,y) \supset TEACHER(x) \wedge COURSE(y)] \quad (2.1)$$

$$(x)(y)[ENROLLED(x,y) \supset STUDENT(x) \wedge COURSE(y)] \quad (2.2)$$

$$(x/COURSE)(Ey/TEACHER)TEACH(y,x) \quad (2.3)$$

$$(x/TEACHER)(Ey/COURSE)TEACH(x,y) \quad (2.4)$$

Notice that, on the view that types formalize the concept of "domain of a relation," then (2.1) and (2.2) specify the domains of the relations *TEACH* and *ENROLLED*.[4] Formulae (2.1)-(2.4) can be viewed as integrity constraints that happen to be true in the given interpretation.

Notice also that the logician's fancy definition of truth in an interpretation, involving as it does the notion of an environment ρ for variables, is motivated by the requirement of maintaining the distinction between the objects of the interpretation and the purely syntactic symbols of the first order language. Of course, no one really thinks of

[4] Notice that I have not yet defined the concept of a relation. It should be clear from Example 2.2, however, that *TEACH* and *ENROLLED* will be examples of relations by whatever definition I eventually come up with.

interpretations in this way, at least not in the database setting. Rather, one thinks of the tables of an interpretation as defining a set of propositions. In Example 2.2, the true propositions are $TEACHER(A)$, $TEACHER(B)$ . . . . . $ENROLLED(d,P200)$, $=(a,a)$ . . . . . $=(P200, P200)$. Those propositions not included in this set are treated as false. For example, $TEACHER(d)$, $TEACH(B,CS100)$ and $=(A,B)$ are false. Then a wff $(x)\,W(x)$ is true in an interpretation iff for every d in the domain of the interpretation, $W(d)$ is true. $(Ex)\,W(x)$ is true iff for some d, $W(d)$ is true. Of course the logical constants $\wedge$, $\vee$, $\sim$, $\supset$ and $\equiv$ are given their usual truth table definitions. Thus, in the case of finite interpretations, determining the truth of an arbitrary wff reduces to purely propositional truth table evaluations.

2.3 Relational Databases Defined

Recall that a relational language $R = (\mathcal{A}, \mathcal{W})$ is a first order language for which $\mathcal{A}$ contains finitely many constants and finitely many predicates, among which is a distinguished equality predicate and possibly some distinguished unary predicates called simple types. Among all of the possible interpretations for a relational language R, we can single out the class of relational interpretations as follows:

Let $R = (\mathcal{A}, \mathcal{W})$ be a relational language. An interpretation $I = (D,K,E)$ for R is a *relational interpretation* for R iff

1. K: constants of $\mathcal{A} \xrightarrow[onto]{1-1} D$ (so that D must be finite).
2. $E(=) = \{(d,d) \mid d \in D\}$.

The interpretation of Example 2.2 is a relational interpretation.

A *relational database* is a triple (R,I,IC) where:

1. R is a relational language.
2. I is a relational interpretation for R.
3. IC is a set of wffs of R, called *integrity constraints*. In particular, it is required that for each n-ary predicate P distinct from = and the simple types, IC must contain a wff of the form

$$(x_1) \ldots (x_n)[P(x_1, \ldots, x_n) \supset \tau_1(x_1) \wedge \ldots \wedge \tau_n(x_n)]$$

where the τ_i are types. $\tau_1, \ldots, \tau_n$ are called the *domains* of P.

For each predicate P distinct from the simple types, the extension $E(P)$ is called a *relation*. When the context is clear, I shall often refer to a relation by the name of the corresponding predicate P. Thus I will speak of "the relation P" in referring to P's extension.

The integrity constraints IC of a relational database (R,I,IC) are said to be *satisfied* iff I is a model for IC. Wffs (2.1)-(2.4) of Example 2.2 (continued) might well be taken to define a set of integrity constraints in which case Example 2.2, together with its continuation, defines a relational database. The wffs (2.1) and (2.2) then define the domains of the predicates $TEACH$ and $ENROLLED$. For this example, the relational database satisfies its integrity constraints.

A few remarks are in order.

1. Since the extension of the equality predicate is the set of all pairs (d,d) of domain elements, $=(d,d')$ is false for all distinct domain elements d,d'. This is in keeping with the universally adopted assumption in database theory that distinctly named individuals are in fact distinct. From our model theoretic perspective, this means that different domain elements denote different individuals, so that in Example 2.2, the proposition $=(P100,P200)$ is false, whereas $=(P100,P100)$ is true.
2. Relational database theory generally incorporates a set of arithmetic comparison operators like $<$, $=$, $>$ *etc.*, as needed. I have chosen only to represent the equality "operator," primarily because it will play a prominent role in the subsequent theory. It would be a simple matter to modify my definition of a relational database to include the binary predicates $<$, $>$, and indeed any set of desired binary "operators." The basic difference between my approach and the conventional one is that I treat these operators as predicates that are extensionally defined within the theory, whereas conventionally these operators are viewed as procedures whose formal properties are understood by everyone and therefore are not defined within the theory. They are, so to speak, "external operators."
3. The concept of an integrity constraint as defined above corresponds to the so-called *static* integrity constraints or *state laws* of [NY78]. Such constraints are meant to be satisfied by any state of the database. In contrast there is also the concept of a *dynamic* integrity constraint or *transition law* [NY78]. Satisfaction of a dynamic constraint is a function of not just the current state of the database but also of its successor state. I do not, in this chapter, address this latter class of integrity constraints, except to point out their intimate connection with the well known "frame problem" in Artificial Intelligence [RAPH71].

2.4 A First Order Query Language

The query language I will appeal to is one first defined in [REIT77] and used subsequently in [REIT78a] and [REIT80a]. It is obviously a close relative of that used in the domain calculus of [ULLM80] (pp. 116-117).

Queries are defined relative to a given relational language $R = (\mathcal{A}, \mathcal{W})$. Specifically, a *query for* R is any expression of the form $<\overline{x}/\overline{\tau} \mid W(\overline{x})>$ where:

1. $\overline{x}/\overline{\tau}$ denotes the sequence $x_1/\tau_1, \ldots, x_n/\tau_n$, and the x_i are variables of $\mathcal{A}$.
2. Each τ_i is a type composed of simple types of $\mathcal{A}$.
3. $W(\overline{x}) \in \mathcal{W}$ and the only free variables of $W(\overline{x})$ are among $\overline{x} = x_1, \ldots, x_n$. Moreover, all of the quantifiers occurring in $W(\overline{x})$ are type restricted quantifiers.

If $DB = (R,I,IC)$ is a relational database then a query for R is said to be *applicable* to DB. The intention here is that information may be retrieved from a relational database only by posing queries that are applicable to that database.

Intuitively, the query $<\overline{x}/\overline{\tau} \mid W(\overline{x})>$ is meant to denote the set of all tuples of constants $\overline{c} = c_1, \ldots, c_n$ such that each c_i satisfies the type τ_i, and such that the database satisfies $W(\overline{c})$. A formal definition will follow the next example.

Example 2.3

The following are sample queries applicable to the education database of Example 2.2:

Who teaches $P100$?

$$<x/TEACHER \mid TEACH(x,P100)>$$

Who are all of A's students?

$$<x/STUDENT \mid (Ey/COURSE)\,TEACH(A,y) \wedge ENROLLED(x,y)>$$

What courses does a take, and who teaches them?

$$<x/COURSE,\ y/TEACHER \mid ENROLLED(a,x) \wedge TEACH(y,x)>$$

Who teaches all of the students?

$$<x/TEACHER \mid (y/STUDENT)\,(Ez/COURSE)\,TEACH(x,z) \wedge ENROLLED(y,z)>$$

The following queries are not applicable to this database because they involve constants or predicates that are not part of the alphabet of the relational language for the database:

$$<x/TEACHER \mid TEACH(x,MATH100)>$$

$$<x/SUPPLIER \mid (y/PART)\,SUPPLIES\,(x,y)>$$

Formally, let $DB = (R,I,IC)$ and let $Q = <\overline{x}/\overline{\tau} \mid W(\overline{x})>$ be a query applicable to DB. A tuple $\overline{c}$ of constants of R's alphabet is an *answer* to Q *with respect to* DB iff

1. $\tau_i(c_i)$ is true in I, $i = 1, \ldots, n$.
2. $W(\overline{c})$ is true in I.

Notice that the concept of an answer is defined only for queries applicable to DB. A query not applicable to DB must involve predicates not contained in R's alphabet and which therefore have no extensions in I, or constants not contained in R's alphabet and which thus have no corresponding domain elements in I. In other words, DB *does not know* about these predicates or constants, in which case the query must be viewed as meaningless.

Finally, notice that there is no correlate in my definition of a query to the notion of a safe [ULLM80] or definite [KUHN67] or range separable [CODD72] query. Essentially these latter restrictions on queries are deemed necessary in order to avoid ever computing the unrestricted complement of a relation; this because such complements are seen as either infinite or undefined. But notice that when a relational database is a triple (R,I,IC), the complement of a type or a relation is finite and perfectly well defined since there are but finitely many individuals in the domain of I. These are the only individuals the database *knows* about; as far as it is concerned *these are all and only the existing individuals.* There is no need for the concept of a safe query.

The source of the safe query constraint in conventional database theory can be traced to the concept of a domain for a relation as the totality of all individuals of a certain kind. Thus, the totality of all parts might be a domain for an inventory database, or the totality of all suppliers. Domains might be infinite, as is the set of all integers. Whether these domains are conceived as being finite or infinite, it is the *completed totality* of such individuals that is somehow seen to be part of the database, despite the fact that in any state of the database only a finite subset of this totality will be explicitly represented. Unrestricted complements of relations are understood to be defined with respect to the completed totality of database individuals, not with respect to the finitely many explicitly present representatives of this totality. Hence the requirement of safe queries.

Now I must confess to a certain discomfort over this notion of complementation with respect to completed totalities. For this totality is never explicitly represented in the database; rather, it is a conceptualization that we, as humans, entertain. There is no way that the database can be said to know about, say, the set of all integers, at least not without some representation of Peano arithmetic. It knows only about some finitely many integers, and precious little about them. It seems to me that queries are about things the database *knows about* (suppliers, integers, *etc.*, that it has explicit representations for). A query $<\bar{x}/\bar{\tau} \mid W(\bar{x})>$ asks for all tuples $\bar{x}$ *known to the database*, satisfying $\bar{\tau}$ and W. In this view, complementation is perfectly respectable. $<x/INTEGER \mid \sim P(x)>$ denotes the set of all integers known to the database for which $\sim P(x)$ is known to be true.

2.5 Some Problems with the Model Theoretic Perspective

The model theoretic paradigm has an elegance and simplicity that accounts, in large measure, for the overwhelming success of Codd's original proposal for a relational model of data [CODD70]. Yet it is not without its difficulties, some of which (*e.g.*, null values) Codd had foreseen, others of which have subsequently emerged. I shall here focus on two such problems with the model theoretic world view.

2.5.1 Databases with Incomplete Information

A variety of phenomena fall under this rubric. I shall consider two of these: disjunctive information and the need for null values.

2.5.1.1 Disjunctive Information

One encounters this problem whenever there is the need to represent a fact of the kind "P is the case, or Q is the case, or . . . " but it is not known which of $P,Q, \ldots$, actually is the case. [LIPS79] proposes a treatment of this situation under certain simplifying assumptions. For the education database of Example 2.2, we face this problem in an attempt to represent the fact "a is enrolled in $P200$ or in $CS200$, but I don't know which." The obvious (and indeed only) approach within the model theoretic framework is to split the given interpretation into three interpretations I_1, I_2 and I_3, each identical to the given one, except that, in I_1, the relation *ENROLLED* contains the additional tuple $(a,P200)$. In I_2 it contains $(a,CS200)$, while in I_3 it contains both $(a,P200)$ and $(a,CS200)$. Then $\bar{c}$ will be defined to be an answer to the query $<\bar{x}/\bar{\tau} \mid W(\bar{x})>$ iff $\tau_i(c_i)$ and $W(\bar{c})$ are all true in all three interpretations I_1, I_2 and I_3. This idea generalizes in the obvious way to the concept of a database involving many interpretations, and the concept of query evaluation requiring truth in all these interpretations. Anyone familiar with the completeness theorem for first order logic will immediately detect proof theory in this observation.

Notice that we cannot avoid the problem of multiple interpretations by treating the formula $ENROLLED(a,P200) \vee ENROLLED(a,CS200)$ as an integrity constraint. For to do so would require that at least one of $(a,P200)$ and $(a,CS200)$ be included in the relation *ENROLLED* in order for this constraint to be satisfied, and we don't know which of these tuples is the case.

2.5.1.2 Null Values

This terminology embraces a multitude of necessary evils in database theory. I shall focus here on just one such null, namely "value at present unknown." In fact this null value has two distinct manifestations: "value at present unknown, but one of some finite set of known possible values," and "value at present unknown, yet *not necessarily* one of some finite set of known possible values." As an example of the former, suppose that in our education database we wish to represent the fact that e is a student who is enrolled in some course, but we don't know which course that is. Suppose further that we know that the only existing courses are the ones mentioned in the database, so that a priori, whichever course that e is taking, it is one of $CS100$, $CS200$, . . . , $P200$. Then our task is to represent the disjunctive fact:

$$ENROLLED(e,CS100) \vee ENROLLED(e,CS200)$$
$$\vee \ldots \vee ENROLLED(e,P200)$$

which is just the problem of disjunctive information discussed above.

As an example of the latter "value unknown" null, consider the ubiquitous "supplier and parts" database which contains a relation *SUPPLIES* (.,.) whose domains are specified by:

$$(x)(y)\,[SUPPLIES(x,y) \supset SUPPLIER(x) \wedge PART(y)]$$

Now suppose that p is a part with no known supplier, but we do know that someone, perhaps one of the known suppliers, perhaps not, does supply it. How shall we represent this fact? The standard approach (see, for example, [CODD79]) is to postulate a new unknown but existing entity ω, a *null value*, then add the tuple (ω,p) to the *SUPPLIES* table, add ω to the *SUPPLIER* table and p to the *PART* table. But ω is an individual with quite a different character from the other known individuals of the database, so it is deemed necessary to augment the conventional truth values {true, false} with a third truth value "unknown" in order to correctly evaluate queries over databases containing such null values. The effect of this third truth value is then an extension of the relational algebra so that, for example, equality and the join operator suitably reflect the intended meaning of this null value.

Notice that *the multiple truth valued approach to null values is a direct and natural consequence of the model theoretic paradigm of relational database theory.* Models are concerned with truth. Since two truth values suffice for the evaluation of wffs in an interpretation without nulls, it is only natural to try inventing new truth values in order to evaluate queries in an interpretation with nulls. Notice also that a correct treatment of nulls is predicated on a prior notion of what these null values *mean.* Without a correct *semantics,* no correct extended relational algebra is possible. On this view, multi-valued logics provide one possible framework within which a semantics for values may be defined. Alas, within this framework it is by no means clear how to extend the relational model to correctly represent null values. Although several approaches exist in the literature (*e.g.,* [BISK81] [CODD79] [WALK80] [VASS79] [ZANI77]), there is no general agreement about which of these, if any, provides a correct semantics for nulls. This difficulty is compounded in the presence of additional kinds of null values (*e.g.,* "no value permitted").

2.5.2 Extending the Relational Model to Incorporate More World Knowledge

It is becoming increasingly evident that the relational model provides limited expressive power, and that extensions to the formalism are required in order to incorporate more real world meaning [CODD79] [BZ81]. The following are typical examples of the kinds of real world knowledge that an extended relational model might accommodate:

1. General facts about the world such as "The subpart relation is transitive" and "All men are mortal."
2. Events: Their sequencing and times of occurrence.
3. Generalization hierarchies (IS-A hierarchies) with property inheritance.

It is true that certain kinds of knowledge can be represented within the model theoretic paradigm by treating this knowledge as an integrity constraint. For example, the fact that the subpart relation is transitive

$$(x/PART)\,(y/PART)\,(z/PART)[SUBPART(x,y) \wedge SUBPART(y,z) \supset SUBPART(x,z)]$$

could be an integrity constraint, thereby forcing the extension of *SUBPART* to be closed under transitivity. But other kinds of information, for example disjunctive information, cannot be treated as integrity constraints, as observed in Section 2.5.1.1. Because of this, and because there are settings in which various forms of inference seem necessary (for example, property inheritance in hierarchies), other approaches to the strict model theoretic have been proposed. I shall return to these issues in Section 4.2 in the context of a proof theoretic view of databases.

3. Databases and Logic: the Proof Theoretic Perspective

My objective in this section is to show how the model theoretic perspective on databases can be reinterpreted in purely proof theoretic terms. Specifically, I shall define a class of first order theories, called relational theories, and prove an equivalence result relating relational theories to relational interpretations. From this it will follow that a definition of a relational database, equivalent to the one presented in Section 2.3, is as a triple (R,T,IC) where T is a relational theory. Then, all prior definitions, involving as they do truth in a relational interpretation, can be reformulated in terms of provability in the theory T. The point of this result, namely its capacity for generalization, will be taken up in Section 4.

3.1 Relational Theories

Imagine given a database (R,I,IC). I shall assume, as I have assumed all along, that the domain elements of I are named using constant symbols of R's alphabet.[5] In addition, instead of viewing the relational interpretation I as a set of tables, think of it as a set of ground atomic formulae. Thus, in Example 2.2, think of the interpretation as being specified by the ground atomic formulae

$$\{TEACHER(A),\ TEACHER(B),\ TEACHER(C),\ \ldots,$$
$$ENROLLED(d,P100), ENROLLED(d,P200),$$
$$=(A,A),\ =(B,B),\ \ldots,\ =(P200,P200)\}.$$

I now propose viewing this set as a first order theory (*i.e.,* as a set of wffs of the underlying relational language).[6] Currently, these wffs are simply ground atomic formulae, but I shall shortly have occasion to modify this set using other kinds of formulae.

---

[5] See the comments of Example 2.2.

[6] In general if $(\mathcal{A}, \mathcal{U})$ is a first order language, then any subset of $\mathcal{U}$ is called a *first order theory* of the language.

Given such a relational interpretation, reinterpreted as a first order theory T, there are various formulae that can be proven, given T as premises. Thus, with reference to Example 2.2 we have the following:

$$T \vdash ENROLLED(c,P100)^7$$

$$T \vdash ENROLLED(a,P100) \wedge TEACH(B,P100)$$

$$T \vdash (Ey/COURSE)\,TEACH(A,y) \wedge ENROLLED(a,y)$$

Notice that all of these provable formulae also happen to be true in the original interpretation. However, there are formulae that are true in the interpretation but that are not provable from the corresponding first order theory. For example:

$$(x)[TEACHER(x) \vee COURSE(x) \vee STUDENT(x)]$$

is such a formula. This is not provable because the first order theory T does not know that $A,B,\ldots,P100,P200$ are all and only the existing individuals. As far as T is concerned, there might be other existing individuals in the world. So augment T with the following domain closure axiom:

$$(x)[=(x,A) \vee =(x,B) \vee \ldots \vee =(x,P100) \vee =(x,P200)]$$

In general, if I is a relational interpretation with domain $c_1,\ldots,c_n$, then the *domain closure axiom* for I [REIT80a] is

$$(x)[=(x,c_1) \vee =(x,c_2) \vee \ldots \vee =(x,c_n)]$$

We can also simplify the representation of the equality relation by replacing all of its instances $=(A,A), =(B,B), \ldots, =(P200,P200)$ by the single formula $(x)=(x,x)$. Our transformed first order theory T now consists of the following wffs:

$$(x)=(x,x)$$

$$(x)[=(x,A) \vee =(x,B) \vee \ldots \vee =(x,P100) \vee =(x,P200)]$$

$$TEACHER(A),\ TEACHER(B),\ \ldots,\ ENROLLED(d,CS200),$$

$$ENROLLED(d,P200).$$

Unfortunately, there still remain wffs true in the original interpretation I but unprovable from T, for example all of the inequalities $\sim=(A,B), \sim=(A,C)$, *etc.* So for each pair of distinct constants c,c' of the domain, augment T with the *unique name axioms* $\sim=(c,c')$ [REIT80a]. Since I am proposing to treat equality proof theoretically, we shall also require the standard axioms specifying the intuitive properties that equality should have, namely commutativity, transitivity, and substitution of one term for another term that is equal to it. These axioms will be given below.

Our theory T now contains unique name axioms, together with axioms for equality. The only remaining problem with T is that it fails to treat negation properly. For example, the wff $\sim TEACHER(a)$, while true in I, is not provable from T. The reason is clear enough; T has models in which $TEACHER(a)$ is true. To avoid this, we need a first order wff which says that the only individuals $TEACHER$ can be predicated of are A, B, and C. This can be done using the *completion* of the predicate $TEACHER$:

$$(x)[TEACHER(x) \supset =(x,A) \vee =(x,B) \vee =(x,C)]$$

Similarly, the completion of the predicate $TEACH$ for our education database is

$$(x)\,(y)[TEACH(x,y) \supset =(x,A) \wedge =(y,CS100)$$
$$\vee =(x,A) \wedge =(y,CS200) \vee =(x,B) \wedge =(y,P100)$$
$$\vee =(x,C) \wedge =(y,P200)]$$

We can now augment the theory T for the education example with the completions of each of the predicates of that database. The first order theory that we finally end up with consists of the following formulae:

1. Domain closure axiom:

$$(x)[=(x,A) \vee =(x,B) \vee \ldots \vee =(x,P100) \vee =(x,P200)]$$

2. Unique name axioms:

$$\sim=(A,B),\ \sim=(B,C),\ \sim=(A,a),\ \ldots.$$

3. Equality axioms specifying the reflexivity, commutativity and transitivity of equality, and the principle of substitution of equal terms.

4. The ground atomic facts:

$$TEACHER(A),\ TEACHER(B),\ \ldots.\ ENROLLED(d,P200).$$

5. Completion axioms for each predicate:

$$(x)[TEACHER(x) \supset =(x,A) \vee =(x,B) \vee =(x,C)]$$
$$(x)\,(y)[ENROLLED(x,y) \supset =(x,a) \wedge =(y,CS100) \vee \ldots \vee$$
$$=(x,d) \wedge =(y,P200)]$$

etc.

[7] If W is a set of first order formulae and if w is a first order formula, then $W \vdash w$ means that there is a first order proof of w from premises W.

Notice that *the only model of this theory is the original interpretation* of Example 2.2. Thus, whenever we had occasion to speak of *truth in this interpretation*, we can instead speak of *provability in the theory*. All of which motivates the following definition:

Let $R = (\mathcal{A}, \mathcal{W})$ be a relational language. A first order theory $T \subseteq \mathcal{W}$ is a *relational theory* of R iff it satisfies the following properties:

1. If $c_1, \ldots, c_n$ are all of the constants of $\mathcal{A}$, T contains the domain closure axiom

$$(x)[=(x,c_1) \vee \ldots \vee =(x,c_n)]$$

T contains the unique name axioms

$$\sim=(c_i,c_j) \quad i, j = 1, \ldots, n \quad i < j$$

2. T contains each of the following equality axioms:

   (i) Reflexivity
   $(x)=(x,x)$

   (ii) Commutativity
   $(x)[=(x,y) \supset =(y,x)]$

   (iii) Transitivity
   $(x)\ (y)\ (z)[=(x,y) \wedge =(y,z) \supset =(x,z)]$

   (iv) Leibnitz' principle of substitution of equal terms:

   For each m-ary predicate symbol P of $\mathcal{A}$,

$$(x_1) \ldots (x_m)\ (y_1) \ldots (y_m)[P(x_1, \ldots, x_m) \wedge$$
$$=(x_1,y_1) \wedge \ldots \wedge =(x_m,y_m) \supset P(y_1, \ldots, y_m)]$$

3. For some set $\Delta \subseteq \mathcal{W}$ of ground atomic formulae, none of whose predicates is the equality predicate, $\Delta \subseteq T$.
   For each m-ary predicate P of $\mathcal{A}$ distinct from the equality predicate define a set C_P of m-tuples of constants by

$$C_P = \{\bar{c} \mid P(\bar{c}) \in \Delta\}.$$

The set $\{P(\bar{c}) \mid \bar{c} \in C_P\}$ is called the *extension of P in T*.[8]

[8] Not to be confused with the concept of the extension of a predicate P in an interpretation.

Suppose $C_P = \{(c_1^{(1)}, \ldots, c_m^{(1)}), \ldots, (c_1^{(r)}, \ldots, c_m^{(r)})\}$. Then in addition to the wffs of Δ, T contains the following *completion axiom* for P:

$$(x_1) \ldots (x_m)[P(x_1, \ldots, x_m) \supset =(x_1,c_1^{(1)}) \wedge \ldots \wedge =(x_m,c_m^{(1)})$$
$$\vee \ldots \vee =(x_1,c_1^{(r)}) \wedge \ldots \wedge =(x_m,c_m^{(r)})]$$

If $C_P = \{\}$, then P's extension in T is empty, and T's completion axiom is

$$(x_1) \ldots (x_m) \sim P(x_1, \ldots, x_m).$$

4. The only wffs of T are those sanctioned by conditions 1-3 above.

Notice that in a relational theory T the extension of P in T together with P's completion axiom is logically equivalent to the wff

$$(x_1) \ldots (x_m)[P(x_1, \ldots, x_m) \equiv =(x_1,c_1^{(1)}) \wedge \ldots \wedge =(x_m,c_m^{(1)})$$
$$\vee \ldots \vee =(x_1,c_1^{(r)}) \wedge \ldots \wedge =(x_m,c_m^{(r)})]$$

This is the "if and only if form" of the predicate P as defined in [CLAR78]. The idea of using a completion axiom in the above definition derives from Clark's paper.

The following theorem establishes an equivalence between relational theories and relational interpretations.

Theorem 3.1. *Suppose $R = (\mathcal{A}, \mathcal{W})$ is a relational language. Then:*

1. *If T is a relational theory of R, then T has a unique model I which is a relational interpretation for R.*
2. *If I is a relational interpretation for R then there is a relational theory T of R such that I is the only model of T.*

Proof.

1. Let $I = (D,K,E)$ be the following relational interpretation for R:

   (i) $D = \{c_1, \ldots, c_n\}$ where $c_1, \ldots, c_n$ are all of the constants of $\mathcal{A}$.

   (ii) $K(c_i) = c_i$, $i = 1, \ldots, n$.

   (iii) $E(=) = \{(c,c) \mid c \in D\}$

   If P is a m-ary predicate of $\mathcal{A}$ whose completion axiom in T has the form

$$(x_1), \ldots (x_m) \sim P(x_1, \ldots, x_m) \tag{3.1}$$

   (so that P's extension in T is empty), $E(P) = \{\}$. Otherwise P's completion axiom in T has the form

$$(x_1) \ldots (x_m)[P(x_1, \ldots, x_m) \supset =(x_1, c_1^{(1)}) \wedge \ldots \wedge =(x_m, c_m^{(1)}) \vee \ldots \vee =(x_1, c_1^{(r)}) \wedge \ldots \wedge =(x_m, c_m^{(r)})] \quad (3.2)$$

where the $c_i^{(j)}$ are all constants of $\mathcal{A}$. In this case P's extension in T is $\{P(c_1^{(1)}, \ldots, c_m^{(1)}), \ldots, P(c_1^{(r)}, \ldots, c_m^{(r)})\}$ and $E(P) = \{(c_1^{(1)}, \ldots, c_m^{(1)}), \ldots, (c_1^{(r)}, \ldots, c_m^{(r)})\}$.

I is clearly a model of T. To see that I is T's only model notice first that T's domain closure and unique name axioms force any model M of T to have the same domain as I (up to renaming of I's domain elements). Secondly, T's reflexivity axiom forces the extension in M of the equality predicate to be the same as in I. Finally, the extension and completion axiom, in T, of a predicate P together with T's unique name axioms force P's extension in M to be the same as its extension in I.

2. The proof here involves constructing, from I, a relational theory T in the same fashion as in the educational database of Example 2.2. So, given a relational interpretation $I = (D,K,E)$ for R, define a first order theory $T \subseteq \mathcal{W}$ as follows:

(i) If $D = \{c_1, \ldots, c_n\}$ then T contains the wffs

$$(x)[=(x,c_1) \vee \ldots \vee =(x,c_n)]$$

$$\sim=(c_i,c_j) \quad i, j = 1, \ldots, n, \ i<j.$$

(ii) T contains axioms for the reflexivity, commutativity and transitivity of the equality predicate, together with axioms for the principle of substitution of equal terms for each predicate P of $\mathcal{A}$.

(iii)' For each m-ary predicate P of $\mathcal{A}$ distinct from the equality predicate:

If $E(P) = \{\ \}$ then T contains the wff (3.1).

If $E(P) = \{(c_1^{(1)}, \ldots, c_m^{(1)}), \ldots, (c_1^{(r)}, \ldots, c_m^{(r)})\}$ then T contains the wff (3.2), together with each of the wffs $P(c_1^{(i)}, \ldots, c_m^{(i)})$ $i = 1, \ldots, r$.

(iv) The only wffs in T are those sanctioned by (i)-(iii) above.

Then T is a relational theory of R and it is not hard to see, as in the proof of 1 above, that I is a unique model of T.

QED

Corollary 3.2. *Suppose T is a relational theory of a relational language R, and that I is a model of T. Then for any wff w of R, w is true in I iff $T \vdash w$.*

Proof. The proof follows from the fact that I must be a unique model of T and the completeness theorem for first order logic.

3.2 A Proof Theoretic Reconstruction of Relational Database Theory

Theorem 3.1 and Corollary 3.2 form the basis for a proof theoretic reconstruction of all the model theoretic concepts and definitions of Section 2. For if (R,I,IC) is a relational database, then we can construct, as in the proof of 2. of Theorem 3.1, a relational theory T of R for which I is T's only model. By Corollary 3.2, the concepts of truth in I and provability from T are equivalent. Conversely, by 1. of Theorem 3.1, any relational theory T defines a unique relational interpretation I, and again, by Corollary 3.2, truth in I is equivalent to provability from T.

Accordingly, we can equivalently define a *relational database* to be a triple (R, T, IC) where R and IC are as before, and T is a relational theory of R. The integrity constraints IC are said to be *satisfied* iff for each $w \in IC$, $T \vdash w$. If $Q = <\overline{x}/\overline{\tau} \mid W(\overline{x})>$ is a query applicable to this database, then an n-tuple $\overline{c}$ of constants of R's alphabet is an *answer* to Q *with respect to this database* iff

1. $T \vdash \tau_i(c_i)$ $i = 1, \ldots, n$ and

2. $T \vdash W(\overline{c})$

4. Generalizing the Proof Theoretic Perspective

In this section I shall show how the proof theoretic view of a relational database as a triple (R,T,IC) admits a variety of generalizations, through modification of the first order theory T.

4.1 Databases That Contain Incomplete Information

Recall that in Section 2.5.1 I discussed two manifestations of the problem of representing incomplete information within the model theoretic paradigm of database theory, namely disjunctive information and null values. Let us return to these problems and determine the

4.1.1 Disjunctive Information

This was the problem of representing disjunctive facts of the form: "P is the case, or Q is, or ... , but I don't know which," and of using such incomplete information in deriving answers to database queries.

Example 4.1

Consider the following relational theory for a supplier and parts world:

PART	*SUPPLIER*	*SUPPLIES*	*SUBPART*
p_1	Acme	Acme p_1	p_1 p_2
p_2	Foo	Foo p_2	
p_3			

where for brevity I use tables to specify the predicate extensions in the theory instead of the ground atomic formulae $PART(p_1)$, $PART(p_2)$, . . . , $SUBPART(p_1,p_2)$.

Domain Closure Axiom:

$$(x)[=(x,p_1) \vee =(x,p_2) \vee =(x,p_3) \vee =(x,Acme) \vee =(x,Foo)]$$

Unique Name Axioms:

$$\sim=(p_1,p_2),\ \sim=(p_2,p_3),\ etc.$$

Equality Axioms: as usual

Completion Axioms:

1. $(x)[PART(x) \supset =(x,p_1) \vee =(x,p_2) \vee =(x,p_3)]$
2. $(x)[SUPPLIER(x) \supset =(x,Acme) \vee =(x,Foo)]$
3. $(x)(y)[SUPPLIES(x,y) \supset =(x,Acme) \wedge =(y,p_1) \vee =(x,Foo) \wedge =(y,p_2)]$
4. $(x)(y)[SUBPART(x,y) \supset =(x,p_1) \wedge =(y,p_2)]$

Now suppose that we also wish to represent the disjunctive fact: "Foo supplies p_1 or Foo supplies p_3 but I don't know which." This item of information can be represented by the wff:

$$SUPPLIES(Foo,p_1) \vee SUPPLIES(Foo,p_3) \qquad (4.1)$$

Now one must resist the natural temptation to simply add this wff to the above theory, thinking that one has thereby provided a correct representation of this world. To see why, consider the contrapositive of 3, the completion axiom for *SUPPLIES*:

$$\vee \sim=(y,p_2)]$$

From this and from the unique name axioms we can prove, taking $x = Foo$ and $y = p_1$, the wff $\sim SUPPLIES(Foo,p_1)$. Similarly we can prove $\sim SUPPLIES(Foo,p_3)$. But these two facts are inconsistent with the disjunctive wff (4.1).

The reason for this "anomaly" is clear enough; the completion axiom 3 was designed to say that, of the original theory, the *only possible instances* of *SUPPLIES* are $(Acme,p_1)$ and (Foo,p_2). But the disjunctive wff (4.1) says that *there are other possible instances* of *SUPPLIES*, namely (Foo,p_1) and (Foo,p_3). To accommodate these new possible instances replace the completion axiom 3 by:

$$3'.\ (x)(y)[SUPPLIES(x,y) \supset =(x,Acme) \wedge =(y,p_1) \vee =(x,Foo) \wedge =(y,p_2) \vee =(x,Foo) \wedge =(y,p_1) \vee =(x,Foo) \wedge =(y,p_3)]$$

Notice that we can, with 3', still prove $\sim SUPPLIES(Acme,p_2)$ as before, but we can no longer, as we could before, prove $\sim SUPPLIES(Foo,p_1)$ or $\sim SUPPLIES(Foo,p_3)$. This is precisely what one's intuition about disjunctive facts such as (4.1) would demand.

Now consider representing, in addition to (4.1), the fact

> "If Acme does not supply p_2 then p_2 must be a subpart of p_3."

This can also be represented as a disjunctive wff

$$SUPPLIES(Acme,\ p_2) \vee SUBPART(p_2,p_3) \qquad (4.2)$$

Again we want to include this wff in the theory, but the completion axioms 3' and 4 must both be modified to accommodate the new possible instances $(Acme,\ p_2)$ and (p_2,p_3) of *SUPPLIES* and *SUBPART*. So replace 3' and 4 by

$$3''.\ (x)(y)[SUPPLIES(x,y) \supset =(x,Acme) \wedge =(y,p_1) \vee =(x,Foo) \wedge =(y,p_2) \vee =(x,Foo) \wedge =(y,p_1) \vee =(x,Foo) \wedge =(y,p_2) \vee =(x,Acme) \wedge =(y,p_2)]$$

$$4'.\ (x)(y)[SUBPART(x,y) \supset =(x,p_1) \wedge =(y,p_2) \vee =(x,p_2) \wedge =(y,p_3)]$$

All of which leads to a new theory consisting of:

1. The extensions defined by the tables.
2. The domain closure, unique name, and equality axioms.

3. The completion axioms 1, 2, 3" and 4'.

4. The disjunctive wffs (4.1) and (4.2).

This theory provides an intuitively correct representation for this incompletely specified world.

These considerations lead to a natural generalization of the concept of a relational database to incorporate disjunctive information, as follows.

Let $R = (\mathcal{A}, \mathcal{W})$ be a relational theory. A wff of $\mathcal{W}$ is called a *positive ground clause* of R iff it has the form $A_1 \vee \ldots \vee A_r$ where each A_i is a ground atomic formula whose predicate is distinct from the equality predicate. The case $r = 1$ is permitted, in which case the clause is simply a ground nonequality atomic formula. A first order theory $T \subseteq \mathcal{W}$ is a *generalized relational theory* of T iff it satisfies the following properties:

1. If $c_1, \ldots, c_n$ are all of the constants of $\mathcal{A}$, T contains the domain closure axiom

$$(x)[=(x,c_1) \vee \ldots \vee =(x,c_n)].$$

T contains the unique name axioms

$$\sim=(c_i,c_j) \quad i,j = 1, \ldots, n \quad i < j$$

2. T contains axioms for the reflexivity, commutativity and transitivity of equality, together with an axiom for the substitution of equal terms for each predicate P of $\mathcal{A}$.

3. For some set $\Delta \subseteq \mathcal{W}$ of positive ground clauses of R, $\Delta \subseteq T$. For each m-ary predicate P of $\mathcal{A}$ distinct from the equality predicate, define a set C_P of m-tuples of constants by

$C_P = \{\overline{c} \mid$ for some positive ground clause $A_1 \vee \ldots \vee A_r$ of Δ and some i, $1 \leqslant i \leqslant r$, A_i *is* $P(\overline{c})\}$

Suppose $C_P = \{(c_1^{(1)}, \ldots, c_m^{(1)}), \ldots, (c_1^{(r)}, \ldots, c_m^{(r)})\}$. Then in addition to the wffs of Δ, T contains the following *completion axiom* for P:

$$(x_1) \ldots (x_m)[P(x_1, \ldots, x_m) \supset =(x_1,c_1^{(1)}) \wedge \ldots \wedge =(x_m,c_m^{(1)}) \vee \ldots \vee =(x_1,c_1^{(r)}) \wedge \ldots \wedge =(x_m,c_m^{(r)})]$$

If $C_P = \{\}$, then T's completion axiom is

$$(x_1) \ldots (x_m) \sim P(x_1, \ldots, x_m).$$

4. The only wffs of T are those sanctioned by conditions 1-3 above.

Notice that the definition of a relational theory of Section 3.1 is a special case of the above definition, in which Δ is a set of ground nonequality atomic formulae. It is natural to define a *generalized relational database* to be a triple (R,T,IC) where T is a generalized relational theory, and R and IC are as before. Similarly, the definition of IC being *satisfied* and the definition of an *answer* to a query are as in Section 3.2.

Generalized relational theories are sufficiently complicated to cause concern about their consistency. Not to worry.

Theorem 4.1. *Every generalized relational theory T is consistent.*

Proof: This is proven by constructing a model of T. Suppose $R = (\mathcal{A}, \mathcal{W})$ and $T \subseteq \mathcal{W}$ is a generalized relational theory. Define an interpretation $I = (D,K,E)$ for R with domain $D = \{c_1, \ldots, c_n\}$ where these are all of the constants of $\mathcal{A}$. Define $E(=) = \{(c,c) \mid c \in D\}$. Then I satisfies the domain closure, unique name, and equality axioms of T. Finally, for each nonequality predicate P of $\mathcal{A}$ define $E(P) = \{\overline{c} \mid \overline{c} \in C_P\}$. Then I satisfies each wff of Δ as well as the completion axioms for each nonequality predicate of $\mathcal{A}$. Hence I is a model of T.

QED

4.1.2 The Semantics of Null Values

The concept of a relational database as developed in Sections 2 and 3 or as generalized in Section 4.1.1 did not accommodate null values. Indeed, as I remarked in Section 2.5.1.2, it is by no means clear what some of these null values even *mean*. My purpose now is to show in some detail how one particular null (namely "value at present unknown, but not necessarily one of some finite set of known possible values")[9] may be defined within the proof theoretic paradigm for database theory.[10]

To focus the discussion, consider the relational theory defined at the beginning of Example 4.1. Suppose we wish to represent the fact:

[9] See the discussion of Section 2.5.1.2.

[10] The other most common null value, namely "no value permitted," also has a simple first order representation. For example, suppose *EMP(p,m,s)* denoted that person *p* whose marital status is *m* (*M*arried or *S*ingle) has spouse *s*. Then if John-Doe is single, no value for *s* is permitted. This can be represented by *(s)PERSON) ~EMP (John-Doe,S,s)*. I shall not consider such nulls in this chapter.

"Some supplier supplies part p_3 but I don't know who it is. Moreover, this supplier may or may not be one of the known suppliers Acme and Foo."

This fact may be represented by the first order wff

$$(Ex)SUPPLIER(x) \wedge SUPPLIES(x,p_3) \tag{4.3}$$

which asserts the existence of an individual x with the desired properties. Now we can choose to name this existing individual (call it ω) and instead of (4.3), ascribe these properties to ω directly:

$$SUPPLIER(\omega) \wedge SUPPLIES(\omega,p_3) \tag{4.4}$$

In database terminology, ω is a *null value*. It is called a *Skolem constant* by logicians. Skolem constants, or more generally Skolem functions, provide a technical device for the elimination of existential quantifiers in proof theory (see, for example, [CL73]).

The problem at hand is how to correctly integrate the facts (4.4) into our supplier and parts relational theory. Notice first that ω is a new constant, perhaps denoting the same individual as some known constant, perhaps not. So the unique name axioms remain untouched. The domain closure axiom, however, must be expanded to accommodate this new constant:

$$(x)[=(x,p_1) \vee =(x,p_2) \vee =(x,p_3) \vee =(x,Acme) \vee =(x,Foo) \vee =(x,\omega)] \tag{4.5}$$

Moreover, the completion axioms for *SUPPLIER* and *SUPPLIES* must likewise be expanded:

$$(x)[SUPPLIER(x) \supset =(x,Acme) \vee =(x,Foo) \vee =(x,\omega)] \tag{4.6}$$

$$(x)(y)[SUPPLIES(x,y) \supset =(x,Acme) \wedge =(y,p_1) \vee =(x,Foo) \wedge =(y,p_2) \vee =(x,\omega) \wedge =(y,p_3)] \tag{4.7}$$

If now we add the facts $SUPPLIER(\omega)$ and $SUPPLIES(\omega,p_3)$ to this modified theory we end up with an intuitively correct representation. Notice that in this resulting theory, *the only thing that distinguishes the Skolem constant ω from the "ordinary" constants Acme, Foo, etc., is the absence of unique name axioms for ω.*

Notice also that in this theory we can prove things like $\sim SUPPLIES(Acme,p_2)$, and $\sim SUPPLIES(Foo,p_1)$, but *not* $\sim SUPPLIES(Acme,p_3)$ or $\sim SUPPLIES(Foo,p_3)$. Intuitively, this is precisely what we want. For we know $SUPPLIES(\omega,p_3)$. Moreover, we *don't know* whether ω is the same as, or different than, Acme or Foo.[11]

[11] Remember that there are no unique name axioms for ω.

So if we could prove, say $\sim SUPPLIES(Acme,p_3)$, we could also prove $\sim=(\omega,Acme)$, contradicting our presumed ignorance about the identity of ω. What we really have here is a correct formalization of the closed world assumption [REIT78b] in the presence of null values. I shall return to this issue in Section 4.2.4.

One last observation is in order. If we wanted, in addition, to represent the fact:

"Some supplier (possibly the same as Acme or Foo, possibly not) supplies p_2"

$$(Ex)SUPPLIER(x) \wedge SUPPLIES(x,p_2)$$

we must choose a name for this supplier, say ω', which must be distinct from the name of the previous unknown supplier ω. This is for obvious reasons. Moreover, the domain closure axiom (4.5) and the completion axioms (4.6) and (4.7) must be expanded to take ω' into account. In general, each time a new null value is introduced into the theory, the null must be denoted by a fresh name, distinct from all other names of the theory, and the domain closure and completion axioms must be expanded.

These ideas now can be formalized as follows: Let $R = (\mathcal{A}, \mathcal{W})$ be a relational theory, where the constants of $\mathcal{A}$ are partitioned into two disjoint sets of constants $C = \{c_1, \ldots, c_n\}$ and $\Omega = \{\omega_1, \ldots, \omega_r\}$. Here Ω may be empty, but C may not be. Each ω_i is called a *null value*. As before, a wff of $\mathcal{W}$ is called a *positive ground clause* of R iff it has the form $A_1 \vee \ldots \vee A_m$ where each A_i is a nonequality ground atomic formula. The case $m = 1$ is permitted. A first order theory $T \subseteq \mathcal{W}$ is a *generalized relational theory of R with null values* iff it satisfies the following properties:

1. T contains the domain closure axiom:

$$(x)[=(x,c_1) \vee \ldots \vee =(x,c_n) \vee =(x,\omega_1) \vee \ldots \vee =(x,\omega_r)].$$

Moreover, T contains the unique name axioms:

$$\sim=(c_i,c_j) \quad i < j, \; i, j = 1, \ldots, n.$$

In addition, T *may* contain one or more inequalitites of the following forms:

$$\sim=(\omega_i,c_j) \text{ for some } 1 \leqslant i \leqslant r, \; 1 \leqslant j \leqslant n.$$

$$\sim=(\omega_i,\omega_j) \text{ for some } 1 \leqslant i, \; j \leqslant r, \; i < j.$$

2. T contains the usual equality axioms.

3. For some set $\Delta \subseteq \mathcal{W}$ of positive ground clauses of R, $\Delta \subseteq T$. For each m-ary predicate P of $\mathcal{A}$ distinct from the equality predicate define a set K_P of m-tuples of constants from $C \cup \Omega$ by

   $K_P = \{\bar{k} \mid$ for some positive ground clause $A_1 \vee \ldots \vee A_m$ of Δ and some i, $1 \leqslant i \leqslant m$, A_i is $P(\bar{k})\}$.

   Suppose $K_P = \{(k_1^{(1)}, \ldots, k_m^{(1)}), \ldots, (k_1^{(s)}, \ldots, k_m^{(s)})\}$. Then in addition to the wffs of Δ, T contains the following *completion axiom* for P:

$$(x_1) \ldots (x_m)[P(x_1, \ldots, x_m) \supset =(x_1,k_1^{(1)}) \wedge \ldots \wedge =(x_m,k_m^{(1)}) \vee \ldots \vee =(x_1,k_1^{(s)}) \wedge \ldots \wedge =(x_m,k_m^{(s)})]$$

   If $K_P = \{\}$, then T's completion axiom is

$$(x_1) \ldots (x_m) \sim P(x_1, \ldots, x_m)$$

4. The only wffs of T are those sanctioned by conditions 1-3 above.

The definition of a generalized relational theory of Section 4.1.1 is a special case of the above definition, in which $\Omega = \{\}$.

A *generalized relational database with null values* is a triple (R,T,IC) where R and T are as above, and $IC \subseteq \mathcal{W}$ is a set of integrity constraints. The definitions of an *answer* to a query, and of *satisfaction* of the integrity constraints remain the same as before.

Having formalized a class of first order theories that accommodate null values, we can now observe that *the only formal distinction between a null value $\omega \in \Omega$ and an "ordinary" constant $c \in C$ is that some of the possible unique name axioms for ω are absent from the theory.* If in fact all of the unique name axioms for ω were present (the definition does allow this), then ω would be indistinguishable from an "ordinary" constant.

Notice also that generalized relational theories with null values provide for disjunctive information as well, and permit some quite subtle distinctions to be represented. For example:

"Someone supplies p_3 but I don't know who. Whoever it is, it is neither A nor B."

$$(Ex/SUPPLIER)SUPPLIES(x,p_3) \wedge \sim=(x,A) \wedge \sim=(x,B)$$

which, after elimination of the existential quantifier becomes

$$SUPPLIER(\omega) \wedge SUPPLIES(\omega,p_3) \wedge \sim=(\omega,A) \wedge \sim=(\omega,B)$$

"Someone supplies p_2 and someone supplies p_3. I don't know who they are but I do know they are not the same suppliers."

$$(Ex/SUPPLIER)(Ey/SUPPLIER)SUPPLIES(x,p_2) \wedge SUPPLIES(y,p_3) \wedge \sim=(x,y)$$

which becomes

$$SUPPLIER(\omega_1) \wedge SUPPLIER(\omega_2) \wedge SUPPLIES(\omega_1,p_2) \wedge SUPPLIES(\omega_2,p_3) \wedge \sim=(\omega_1,\omega_2)$$

"Someone supplies p_2 or p_3 but I don't know who. I do know it is not A."

$$(Ex/SUPPLIER)[SUPPLIES(x,p_2) \vee SUPPLIES(x,p_3)] \wedge \sim=(x,A)$$

which becomes

$$SUPPLIER(\omega) \wedge [SUPPLIES(\omega,p_2) \vee SUPPLIES(\omega,p_3)] \wedge \sim=(\omega,A)$$

The following result is comforting.

Theorem 4.2. *Every generalized relational theory T with null values is consistent.*

Proof: The proof is constructed by adding enough inequalities to T to yield a generalized relational theory. By Theorem 4.1, this enlarged theory will be consistent in which case so will any subset of it, in particular T itself.

To suitably enlarge T add to it every inequality $\sim=(c_i,\omega_j)$ such that neither this inequality nor the inequality $\sim=(\omega_j,c_i)$ is already present in T. Similarly, add to T every inequality $\sim=(\omega_i,\omega_j)$ for $i \neq j$ such that neither this nor the inequality $\sim=(\omega_j,\omega_i)$ is already present in T. The resulting theory is a generalized relational theory.

QED

One final observation: any generalized relational theory with null values is decidable, basically because the domain closure axiom restricts the class of its models to those whose domains are no larger than the finite set of constants of the theory. Of course testing a wff for theoremhood by testing it for truth in all these models is hardly an exemplary procedure. A theorem proving approach would certainly be preferable. Better still would be a suitable generalization of the relational algebra, but whether this is even possible remains to be seen.

4.2 Conceptual Modelling: Incorporating More World Knowledge

As I remarked in Section 2.5.2, there is a perceived need within the database community to extend the relational model to accommodate more real world knowledge, and many of the required extensions cannot be accommodated by the model theoretic paradigm for relational databases. A bewildering variety of proposals have been advanced in response to this need. Representative examples include the "Tasmanian" relational model [CODD79], TAXIS, an object oriented programming language [MBW80], class oriented data models [HM78], and semantic networks [SOWA76]. Now there are two problems with this embarrassing number of proposals:

1. How can one begin to compare them? In what formal sense could one claim that two such proposals have the same representational "powers," or that one is a generalization of another? Most such proposals involve different representation languages and different (and in some cases underspecified) semantics, making mappings between them virtually impossible.
2. Insofar as the concept of an answer to a query is defined at all, it is defined operationally, for example by a generalization of the relational algebra, or by some set of retrieval routines which may or may not perform inferences. Now these data models are complicated. Therefore these operational definitions for answers to queries are also complicated. Why should one believe that these definitions are *correct* (*i.e.*, that any answer returned will be intuitively appropriate)? Why should one believe that these definitions are *complete* (*i.e.*, that anything that intuitively should be an answer will be returned)?

My purpose in this section is to indicate how a logical framework can alleviate these problems. Specifically, I shall argue that the kinds of real world knowledge that these extended data models attempt to capture have natural representations as first order formulae. If you grant me this claim for the moment, it follows that such non logical data models can be equivalently formalized by suitably restricted classes of first order theories, much as Section 4.1.2 formalized the relational model with disjunctive information and null values as the class of generalized relational theories with null values. Provided this mapping from a non logical data model to a logical one can be done, we would enjoy a number of immediate benefits [REIT80b]:

1. The semantics of the non logical data model would be precisely defined by its logical translation.
2. Two different non logical data models could be compared (say, with respect to their representational "power"), by comparing their translations.
3. The definition of an answer to a query remains the same as in Section 3.2. This is a central point: no matter how one extends one's data models to incorporate more real world meaning, *the definition of an answer to a query remains the same, as long as this extension is first order definable.* This is not to say that one's query evaluation algorithms must resemble the logician's proof procedures. The relational algebra is such an algorithm, and it looks nothing like proof theory. Nevertheless, logic is the final arbiter of the correctness of proposed query evaluation mechanism for any first order definable data model. Thus *we can prove the correctness of proposed query evaluation algorithms.*
4. Similar remarks hold for integrity constraints. The definition of satisfaction of an integrity constraint remains as it was expressed in Section 3.2 for any first order data model. Thus *we can prove the correctness of proposed integrity maintenance algorithms.*

It remains for me to argue that the kinds of real world knowledge that various semantic data models attempt to capture are representable within first order logic. Space limitations prevent an exhaustive or detailed survey of the kinds of knowledge modelled in the database literature, so I shall focus instead on some of the more prominent semantic requirements.

4.2.1 The Representation of Events

First order event based representations have been used extensively in Artificial Intelligence for modelling verbs and their associated case frames for natural language understanding systems [BRUC75]. These ideas translate very naturally into the database setting. The idea is to extend one's ontology to include a new class of individuals of type *EVENT*, and then to postulate various properties that these individuals may possess. For example, in an inventory database, one may want to represent the fact that an order has been received on June 12, 1981, to be filled by Sept. 1, 1981, and which is to be shipped to Acme. The order is for 12 pipewrenches, catalogue number 1376, and for 24 doors, catalogue number 2001, colour brown. This has as its event-based first order representation:

$$(Ex/ORDER\text{-}EVENT)[DATE\text{-}RECEIVED(x, June\ 12\ 1981)$$
$$\wedge\ DATE\text{-}TO\text{-}BE\text{-}FILLED(x, Sept\ 1\ 1981) \wedge SHIP\text{-}TO(x, Acme)$$
$$\wedge\ GOODS\text{-}ORDERED(x, pipewrench)$$
$$\wedge\ CATALOGUE\text{-}NO(x, pipewrench, 1376)$$

$\wedge$ *QUANTITY*(*x,pipewrench,* 12) $\wedge$ *GOODS−ORDERED*(*x,door*)

$\wedge$ *CATALOGUE−NO*(*x,door,* 2001) $\wedge$ *QUANTITY*(*x,door,* 24)

$\wedge$ *COLOUR*(*x,door,brown*)]

Associated with any individual of type *ORDER−EVENT* might be an integrity constraint specifying that there must be someone to whom the goods are to be shipped, that there are some goods on order, and that the date the order is received must precede the date it is to be filled.

$(x/ORDER-EVENT)\ (Ey/DATE)\ (Ez/DATE)\ (Eu/BUYER)$

$(Ew/INVENTORY-ITEM)[DATE-RECEIVED(x,y)$

$\wedge\ DATE-TO-BE-FILLED(x,z) \wedge y<z$

$\wedge\ SHIP-TO(x,u) \wedge GOODS-ORDERED(x,w)]$

4.2.2 Hierarchies and the Inheritance of Properties

The modelling task here is to provide a first order representation of generalization (IS-A) hierarchies, the properties associated with "classes" in the hierarchy, and how these properties are inherited by classes "lower down" in the hierarchy. These features are common to virtually every attempt in the literature to define data models with more "meaning" (*e.g.*, [CODD79] [HM78] [MBW80] [SS77b]).

For example, consider an educational domain with the hierarchy of simple types (classes) of Figure 4.1. The semantics of this hierarchy can be specified by the following first order wffs:

$(x)[UNDERGRADUATE(x) \supset STUDENT(x)]$

$(x)[GRADUATE(x) \supset STUDENT(x)]$

$(x)[FRESHMAN(x) \supset UNDERGRADUATE(x)]$

$(x)[JUNIOR(x) \supset UNDERGRADUATE(x)]$

etc.,

together with wffs specifying the disjointness of these types, namely:

$(x)\sim[UNDERGRADUATE(x) \wedge GRADUATE(x)]$

$(x)\sim[FRESHMAN(x) \wedge JUNIOR(x)]$

$(x)\sim[FRESHMAN(x) \wedge SOPHOMORE(x)]$

etc.

In addition to this hierarchy, there might be properties that generally hold for simple types "high up" in the hierarchy and that are inherited by any instances of simple types "lower down." For example, it will likely be the case that every student should have a student number:

$(x/STUDENT)\ (Ey/INTEGER)STUDENT-NO(x,y)$

This is an example of a *property* associated with the type *STUDENT*. In general a property of the simple type τ is a wff of the form $(x/\tau)\ (Ey/\theta)P(x,y)$ where θ is some type and P is a binary predicate.

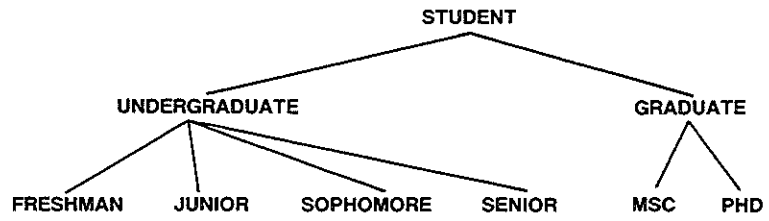

Figure 4.1 A Hierarchy of Simple Types

For the example at hand it is easy to see that the following wffs are all *deducible* from the wffs defining the hierarchy, and the student number property:

$(x/GRADUATE)\ (Ey/INTEGER)STUDENT-NO(x,y)$

$(x/FRESHMAN)\ (Ey/INTEGER)STUDENT-NO(x,y)$

etc.

This is an example of the *inheritance of properties* applying to superclasses down the hierarchy to subclasses. Properties only inherit "downwards." If every freshman must be enrolled in English 100

$(x/FRESHMAN)ENROLLED(x,E100)$

it does not follow, either intuitively or logically from our representation, that every undergraduate must be enrolled in English 100.

The transitivity of "IS-A" is a simple consequence of the transitivity of "implies." Thus the following is provable:

$(x)[MSC(x) \supset STUDENT(x)]$

Finally, the concept of a token t of a class C translates into the logical ground atomic formula $C(t)$. Thus John Doe as a token of the class *GRADUATE* is represented by *GRADUATE*(*John−Doe*).

4.2.3 Aggregations

This modelling notion was introduced in [HM78] and is also treated in [CODD79] [MBW80]. An aggregation is a set of some kind to which one wishes to ascribe various properties. I shall indicate how to represent aggregations in first order logic by modelling certain aspects of professional societies. The simple type *SET* takes sets as its argument. To improve readability, I use upper case symbols for set variables and constants.

Subset defined:

$$(X/SET)\,(Y/SET)[SUBSET(X,Y) \equiv (z)[MEMBER(z,X) \supset MEMBER(z,Y)]]$$

A professional society is a set of people representing a field. Any member of the society is interested in at least one subfield of this field.

$$(X)[PROF\text{-}SOC(X) \supset SET(X)]$$

$$(X)[PROF\text{-}SOC(X) \supset (y)[MEMBER(y,X) \supset PERSON(y)]]$$

$$(X)[PROF\text{-}SOC(X) \supset (Ex/FIELD\text{-}TYPE)[FIELD(X,x)$$

$$\wedge\ (y)[MEMBER(y,X) \supset (Ez/FIELD\text{-}TYPE)SUBFIELD(z,x)$$

$$\wedge\ INTERESTS(y,z)]]]$$

Notice that this is not a definition of a professional society. The wffs merely define various properties that anything called a professional society must possess.

ACM is a professional society of computer scientists.

$$PROF\text{-}SOC(ACM)$$

$$FIELD\text{-}TYPE(cs)$$

$$FIELD(ACM,cs)$$

The executive board of a professional society is a subset of the members of the society and always has a president, a secretary, a treasurer, and members-at-large. Neither the president, treasurer nor secretary may be members-at-large.

$$(X)\,(Y/PROF\text{-}SOC)[EX\text{-}BOARD(X,Y) \supset SET(X) \wedge SUBSET(X,Y)]$$

$$(X/SET)\,(Y/PROF\text{-}SOC)[EX\text{-}BOARD(X,Y) \supset$$

$$[(Eu/PERSON)MEMBER(u,X) \wedge PRESIDENT(u,Y)]$$

$$\wedge\ [(Ev/PERSON)MEMBER(v,X) \wedge SECRETARY(v,Y)]$$

$$\wedge\ [(Ew/PERSON)MEMBER(w,X) \wedge TREASURER(w,Y)]]$$

$$(X/SET)\,(Y/PROF\text{-}SOC)[EX\text{-}BOARD(X,Y) \supset$$

$$(EZ/SET)[SUBSET(Z,X) \wedge MEMBERS\text{-}AT\text{-}LARGE(Z,Y)$$

$$\wedge\ (x/PERSON)[MEMBER(x,Z) \supset \sim PRESIDENT(x) \wedge$$

$$\sim SECRETARY(x) \wedge \sim TREASURER(x)]]]$$

Lady Lovelace is a member of ACM's executive board.

$$(EX)\,EXECUTIVE\text{-}BOARD(X,ACM) \wedge MEMBER(Lady\text{-}Lovelace,X)$$

If one replaces the existentially quantified variable X by a Skolem constant Ω (*i.e.*, a null value) this latter wff becomes

$$EXECUTIVE\text{-}BOARD(\Omega,ACM) \wedge MEMBER(Lady\text{-}Lovelace,\Omega)$$

Using these wffs together with some of the earlier ones we can deduce, among other things

$$PERSON(Lady\text{-}Lovelace)$$

$$MEMBER(Lady\text{-}Lovelace,ACM)$$

A special interest group of a professional society is a set of individuals interested in some subfield of the society.

$$(X)\,(Y/PROF\text{-}SOC)[SIG(X,Y) \supset SET(X)]$$

$$(X/SET)\,(Y/PROF\text{-}SOC)\,[SIG(X,Y) \supset$$

$$(z)\,[MEMBER(z,X) \supset PERSON(z)]]$$

$$(X/SET)\,(Y/PROF\text{-}SOC)\,(u/FIELD\text{-}TYPE)\,(v/FIELD\text{-}TYPE)$$

$$[SIG(X,Y) \wedge FIELD(X,u) \wedge FIELD(Y,v) \supset$$

$$SUBFIELD(u,v) \wedge (z/PERSON)[MEMBER(z,X) \supset INTERESTS(z,u)]]$$

Notice that one may be a member of a special interest group without being a member of the professional society.

SIGART is a special interest group of ACM for Artificial Intelligence.

$$SIG(SIGART,ACM)$$

$$FIELD(SIGART,ai)$$

Using the wffs on hand we can deduce:

$$SUBFIELD(ai,cs)$$

Suppose *rr* is a member of SIGART.

$$MEMBER(rr,SIGART)$$

We can deduce:

$PERSON(rr)$

$INTERESTS(rr,ai)$

4.2.4 Discussion

I have indicated how a variety of data modelling concepts can be naturally represented as first order formulae. Now my earlier conclusions (Section 4.1) were that various species of relational databases are all formalizable by suitable triples (R,T,IC). It is natural, then, to persevere with this notion and to further generalize relational databases to accommodate these new data modelling concepts. More precisely, insofar as a semantic data model admits first order formulae of a certain kind (*e.g.*, formulae for aggregations), then some of these formulae normally will be viewed as integrity constraints. Put them in IC. The remaining formulae then serve as general world knowledge for the inferential retrieval of answers. Put them in T.

Now this leaves us with the mildly uncomfortable view that, in order to do arbitrary conceptual modelling, we must accept databases (R,T,IC) where T and IC are arbitrary first order theories of the relational language R. While this is essentially true, there are certain constraints that one is likely to impose upon T:

1. It should contain a domain closure axiom.
2. It should contain unique name axioms for the known constants of R (but not necessarily for its null values).
3. T should represent the closed world assumption.

This latter point requires amplification. In [REIT78b] I studied the problem of representing negative information in first order databases without null values. My point of departure was the observation that in conventional relational databases, a negative fact like $\sim SUPPLIES(s,p)$, is held to be true provided its positive part (*i.e.*, $SUPPLIES(s,p)$), is not in the database. In other words, a tuple satisfies the negation of a relation iff the tuple is absent from the relation's table. In keeping with my proof theoretic bias, I generalized this notion to first order theories T as follows:

$$\text{Infer} \sim R(\overline{c}) \text{ iff } T \nvdash R(\overline{c}).$$

This characterization of negation in database theory I termed the *closed world assumption*. For a number of reasons, this particular version of the closed world assumption is unsuitable:

1. It treats null values incorrectly.
2. In the presence of disjunctive information it leads to inconsistencies.
3. Since it is a rule of inference, and not a wff or set of wffs, it is not, strictly speaking, first order representable. It is a meta-notion.

Now there is a different way of viewing the closed world assumption, one which provides a strong clue for its first order representability. For it assumes that the given information about the world being modelled is complete in the sense that *all and only* the relationships that can possibly hold among the known individuals are those implied by the given information. It is this point of view that led to the completion axioms for generalized relational theories with null values (Section 4.1.2). These axioms permit the derivation of negative facts from the theory, but only such facts as do not conflict with the unknown individual property of null values, and which do not lead to inconsistencies with the disjunctive information. In this limited setting, the completion axioms provide a correct first order representation of the closed world assumption.

I do not know whether suitable completion axioms can be formulated for more general first order settings, for example settings representing hierarchies and/or aggregations. Whether this is possible or not, some representation of the closed world assumption is necessary. Moreover, this is not a problem peculiar to a logical view of database theory. Any formalism for extended conceptual modelling must provide for the representation of negative information and its use in query evaluation, although this problem is rarely addressed in the literature.

It is of some interest to observe that variants of the closed world assumption arise in contexts other than database theory, for example in providing a semantics for negation in PROLOG and PLANNER-like programming languages [CLAR78] [AV80], and for Artificial Intelligence applications [MCCA80] [REIT80c]. In particular, Clark and McCarthy provide different but extremely interesting first order approaches to the closed world assumption, approaches well worth investigating for their potential impact on database theory. In this connection [REIT82] shows how, for certain classes of databases viewed as first order theories, McCarthy's formalization of the closed world assumption is a generalization of Clark's.

5. Conclusions

I have, in some detail, carried out a logical reconstruction of various aspects of conventional relational database theory. The value of this logical embedding is, in my view, primarily *semantic:* a number of central concepts in database theory have been given precise definitions. Among these are: databases that have incomplete information, including null values; integrity constraints and what it means for them to be satisfied; queries and their answers; and conceptual modelling and what it might mean to represent more real world knowledge.

As I see it, the major *conceptual* advantage of this logical reconstruction is its uniformity:

1. Representational uniformity. Queries, integrity constraints and facts in the database are all represented in the same first order language.
2. Operational uniformity. First order proof theory is the sole mechanism for query evaluation and the satisfaction of integrity constraints.

This uniformity provides a number of practical advantages:

1. Nonlogical data models can be given precise semantics by translating them into logical terms.
2. Different data models may be compared.
3. Non proof theoretic query evaluation algorithms may be proven correct with respect to the logical semantics of queries.
4. Integrity maintenance algorithms may be proven correct with respect to the proof theoretic definition of constraint satisfaction.

A wide variety of questions have not been explored in the chapter, and they require further research.

1. Can the relational algebra be generalized to deal correctly with null values? With disjunctive information?
2. What is an appropriate formalization of the closed world assumption for arbitrary first order theories?
3. Which first order theories admit efficient query evaluation procedures? In this connection, notice that so-called Horn theories accommodate efficient theorem proving techniques [KOWA79] that can be directly applied to query evaluation.
4. What are some criteria for deciding whether a given wff should be treated as an integrity constraint or as knowledge to be used in deriving answers?
5. Suppose we restrict attention to relational databases as defined in Section 3.2. Determine natural classes of integrity constraints for which efficient and provably correct integrity maintenance algorithms can be found. Contrast this approach to correctness proofs with that of [BBC80].

6. Discussion: Why Logic?

In this chapter I have made some arguments favouring a logical (specifically proof theoretic) perspective for relational database theory. While this logical perspective was couched in the first order predicate calculus, other logics are certainly possible, perhaps even desirable in certain settings. (See, for example, Levesque's chapter on incomplete knowledge bases, or [JACO82].) Exactly which logic as appropriate for conceptual modelling can be a contentious issue, as the chapter by Israel and Brachman indicates, and I do not wish here to take sides in this dispute. But there is a prior issue which I do wish to address, and that is whether logic, whatever its species, is even a suitable formalism for conceptual modelling.

The standard opposing view to the logical paradigm has it that data models are definable by a choice of data representation together with suitable *operations* on the data representation (*i.e.*, by the database operations performed for retrieval, updates, deletions, *etc.*, [TL82]). It is the total constellation of these operations that defines the "meaning" of one's representation. But surely this confuses *implementation* with *specification*, for whatever else it might be, a database is a representation of various things which are *known* (or better, *believed*) *about some aspect of the real world.* Logic provides an abstract *specification* language for expressing this knowledge. The logical formulae which presume to specify this knowledge are things which are either true or false in the real world. Of course, they are intended to be true. Nevertheless, they are open for inspection by the sceptical as well as the curious. For example, in Section 4.2.3, I proposed a collection of formulae specifying what I mean by a professional society. You, in turn, are free to decide whether these formulae are true of the world, and whether there are important features (for your application) of a professional society which I failed to specify. If you agree with my formulae, well and good. If not, then at least we have a solid basis for dispute. Either way, the logical formulae are completely up front; they unambiguously specify exactly what I mean by a professional society, no more, no less. In this sense, logic provides a rigorous specification of meaning. Moreover, it does so at a very high level of abstraction, in the sense that the

specification is entirely nonprocedural. It tells us *what* knowledge is being represented. It tells us *what* is meant, for example, by an answer to a query, namely, any tuple of constants which make the query true in all models of the formulae. Similarly, a logic suitable for representing state changes would tell us *what* should be the result of a database update. In no sense does a logical specification include procedures detailing *how* to perform database operations; hence its nonprocedural character.

Of course this emphasis on logic as a specification language ignores a crucial aspect of conceptual modelling, the *implementation* problem. How do we computationally *realize* the abstract logical specification? It is at this implementation level that database operations assume their proper role. A wide variety of options are possible. One extreme is literally to encode the formulae as themselves, and define the database operation for retrieval, say, to be a theorem prover. This approach is advocated by the PROLOG community whenever the formulae are Horn clauses [VANE78]. Another possibility is to represent formulae by a semantic network of some kind, and define the database operation of retrieval by some sort of network interpreter. Usually, such network representations are strongly oriented towards hierarchies and property inheritance, and the associated network interpreter is designed to search up hierarchies for inherited properties [MBW80]. (See Section 4.2.2 for a sketch of a logical specification of such hierarchies. See also [SCHU76a].) Conventional relational database theory encodes ground atomic formulae as themselves (*i.e.*, as a set of relational instances), and the relational algebra supplies the operations for retrieval. Whatever one's choice of data model for realizing a logical specification, this choice will provide a lower level of abstraction, reflecting a concern for implementing the specification, hence a preoccupation with database operations. While necessary, this emphasis on database operations has unpleasant semantic consequences. The semantic effects of certain formulae in the logical specifications are buried in the operations. For example, the effects of the domain closure axiom in the logical specification of a relational database (Section 3.1) are realized in the relational data model by the division operator. The completion axioms are realized by the operation of set difference. *Such axioms are not encoded as part of the data representation, but as data model operations.* The more complex data models become, the more we can expect such operational encodings of specification axioms (not to mention the operational encoding of logical deduction). Provided there is a logical specification to begin with, this makes for good computer science; one can prove that one's data model is a correct realization of the specification.

But what if, as is current practice, a data model is served up without benefit of an abstract specification of the assumptions made by that data model about the world being modelled? Whatever these assumptions might be, they are likely to be buried deeply within the data model's operations. When these are complex operations, how is one to know what the assumptions are? Are they correctly and completely realized by the operations? For that matter, what can "correct" and "complete" even mean in this setting? Without a specification of the "knowledge content" of the database, one which provides a direct connection to the real world being modelled, there can be no concept of the correctness and completeness of a data model's operations. A mature theory of databases will provide for this distinction between a logical specification and its realization by a procedurally oriented data model, and it will require that the operations and data representations of this data model be proven correct and complete with respect to a given specification.

In summary, I have argued the following advantages of logically defined data models for conceptual modelling:

1. Logic is precise and unambiguous. It has a well defined semantics that provides the crucial connection between its formulae and the real world being modelled.

2. Logical data models provide a very high level of abstraction because there are no database operations. They are entirely nonprocedural. They act as specifications of those aspects of the real world being modelled, and of the assumptions one is making about that world.

3. A logical data model is transparent. All and only the knowledge being represented is open for inspection, including assumptions that might otherwise be buried in procedurally oriented data models.

4. Because they are specifications, logical data models can be realized in a variety of ways by procedurally oriented data models. Such data models can be proven correct and complete with respect to the logical specifications that they realize. Since a logical specification provides a connection with the world being modelled (See 1. above), this notion of correctness and completeness is probably the best that one can hope for.

7. Acknowledgments

The bulk of this research was supported by the National Science and Engineering Research Council of Canada under grant A7642. Additional support was provided by NSF Grant MCS-8203954.

I am grateful to David Etherington, Hervé Gallaire, Randy Goebel, Jean Marie Nicolas, Moshe Vardi and K. Yazdanian, all of whom read an earlier draft of this chapter and provided valuable comments and corrections.

8. References

[AV80] [BBC80] [BISK81] [BRUC75] [BZ81] [CL73] [CLAR78] [CODD70] [CODD72] [CODD79] [HM78] [JACO82] [KOWA79] [KUHN67] [LIPS79] [MBW80] [MCCA80] [MEND64] [RAPH71] [REIT77] [REIT78a] [REIT78b] [REIT80a] [REIT80b] [REIT80c] [REIT82] [SCHU76a] [SOWA76] [SS77b] [TL82] [ULLM80] [VANE78] [VASS79] [WALK80] [ZANI77]

Discussion

This chapter uses first order logic (FOL) to express popular DB concepts such as events, hierarchies, and integrity constraints to illustrate the utility of FOL in dealing with issues surrounding the relational data model (RDM).

Current relational database theory is based on model theoretic (MT) notions. The chapter attempts to show that the proof theoretic (PT) approach is better. MT definitions are generalized to PT definitions to provide a good basis for extending RDM theory and to talk precisely about the semantics of extensions, such as those needed for incomplete knowledge (disjunctive information and null values), integrity constraints, and conceptual modelling extensions (events, hierarchies, inheritance, aggregation, and association). The resulting theories are all decidable since the domain of a database (DB) is finite.

Domains

There are at least two different domain notions, one from MT and the other from the RDM. The MT view is that a domain is a set of constants. The DB approach presumes to enumerate all possible values of a domain (like a type in programming language (PL) without the related operations). For example, an *address* domain is the set of *all* possible addresses. Some people attempt to embed as much semantics as possible in the data type of an object (*e.g.*, address and its operations, salary and its operations). In that case, a type such as a string is only a representation of the address object.

Reiter's notion of domain does not make a distinction between type and variable, although such a distinction is generally useful for semantic integrity checking. Reiter claims that such benefits can be gained through axioms (*e.g.*, manager is a subtype of employee). However, the issue here may not be expressibility, but convenience.

Ray Reiter: "This comment implies a misunderstanding of the chapter that addresses *semantic* issues, *not* convenience, efficiency, or implementation. It should be clear that knowledge representation (viewed as an abstract formalism) and database representation (viewed as an implementation) are two different issues."

Extending the RDM Using PT

There are many problems for which the RDM must be extended (*e.g.*, incomplete information, events, and hierarchies). Many of the approaches proposed for these extensions are ad hoc and pose new problems. The RDM can be considered, at a logical level, as a special kind of theory of FOL. The MT approach, the most direct representation for the RDM, cannot deal adequately with the desired extensions. When defining the RDM in the PT approach and one gets not only the RDM but also all the other machinery of FOL with which to resolve other RDM problems. The PT approach provides a proof theory for databases. Using PT, integrity constraints are satisfied if they can be derived from the theory given by the axioms, and queries are answered via proof theory. FOL also can be used to provide a clear semantics for the RDM and its problems and a convenient framework within which to attack the problems.

Reiter proposes that the RDM be extended by using new axioms to handle problems such as disjunctive information and nulls. An example of disjunctive information is "Foo supplies P1 or Foo supplies P3 but I don't know which." The RDM does not permit the information in this statement to be stored or returned. Although there are other such proposed extensions in the literature, their semantics often are not clear. What is meant by the answer to a query in the presence of incomplete information? To understand disjunctive information, you should build a suitable first order theory (in your mind at least) and then use proofs to determine the answer to the query on incomplete information. Having so defined a correct semantics, implement efficient mechanisms to support the semantics, and prove that the implementation corresponds to the given semantics. If one is unhappy with the semantics, then define another. *It is not wise to add a new concept without knowing its semantics.*

It is extremely important that precise specifications be given for any new data model or model extension. If the semantics are not clear, the models or languages are hard to use. In the future, we should not accept a new model or feature without a precise specification that is complete, formal, and unambiguously communicable to its users.

When using a logical formalism to express new models, some things come for free (*e.g.*, means of answering or resolving queries). Papers proposing new models usually give complex ways of retrieving information without demonstrating their correctness. If a data model is specified using FOL, the correct notion of an answer is free. The logic definition correctly characterizes an intuition of how the answer is to be derived, and every answer for which there is an intuition can be retrieved by the same mechanism.

Null Values

In the chapter, null values are special constants about which questions can be asked. There is a recognition problem involving different null values in the RDM literature. For example, if null-x supplies P1 and null-y supplies P3, are null-x and null-y the same? In the PT approach, special constant symbols (W1, W2, ...) are introduced that may or may not be equal to any existing DB constant. If two null values are the same, one must explicitly state this fact. Contrary to some DB proposals, null values are not an issue for three value logic. The issue is what is provable and what is not. If we can neither prove nor disapprove something it means that we have incompletely specified knowledge. The literature is full of conditions of the form: T *or not*(T) evaluates to unknown if T happens to be unknown, but T *or not*(T) is a tautology and so is always true.

State Versus State Transitions

The chapter considers database theory that has not addressed dynamic aspects of databases. Data structures have received considerable emphasis in DB. Now that many structural problems are solved, there is a growing emphasis on behaviour, the semantics and representation of operations over databases. Being able to express the semantics of events is very important—what actually happens when an event takes place. The chapter avoids state changes because of the frame problem, an open problem in AI. The frame problem is that of stating all the invariants for each state change. For example, if you change the state of a room by opening a window, there are unbelievably many invariants (*e.g.*, people in the room remain seated). A complementary problem is to state all changes for an event. What is to be ignored and what is to be emphasized? For a finite domain (*e.g.*, a DB) one can list all the changes and invariants. But, even if one could state all the invariants and changes there is an inference problem; a change can have many side effects. Some of these issues can be addressed using type hierarchies.

Hierarchies and Property Inheritance

In data models, there are higher-level (higher than 1st order) axioms (or model inherent constraints) stating that all hierarchies of the same type have the same properties. This saves writing the same axioms for every application. The same effect can be achieved in FOL through notational convention rather than resorting to a higher order logic.

To deal adequately with hierarchies in PT, completion axioms are needed which allow you to ask questions about elements that are not part of the hierarchy. These axioms are not well understood. Although they may turn out to be simple, they currently appear to be very difficult.

Logical Modelling and Efficiency

Some first order theories permit more efficient query evaluation than others. This fact is used in the PROLOG community (*e.g.*, Horn clauses provide efficient computation and a PROLOG program can be annotated to improve efficiency). There are two issues here. First, there are many ways to ask the same question. Second, is the efficiency of the system measured by how well it performs on the most efficient representation of a query? In this regard, input for efficient implementation should not be sought at the logical description level but in meta-knowledge (*e.g.*, query optimizers should look for hints in the dictionary, not in the database). Efficiency in the RDM is a different problem. It is hard to define predicates that will help in evaluating optimal search paths.

What is the relationship between FOL and implementation efficiency? How does one theory relate to another on the basis of efficiency, assuming that the theories have equivalent interpretations? There is no known answer to these questions. If one happens to know how PROLOG works, then annotations can be used to take advantage of it. Annotation was used long ago for FORTRAN II but annotated programs were found to be less efficient than the standard compiled programs. The RDM is a subset of FOL that is reasonably efficient to process; maybe other "theories" are even more efficient.

Efficiency concerns representation and not logic. There are several ways of improving representations. One approach is to have the compiler collect data from programs (to be kept as a meta-knowledge) and from users (in exceptional cases). Another approach would be to automatically and incrementally map from one representation, say pure PROLOG, to equivalent and more efficient representations.

Concluding Remarks

There are four main benefits of mapping nonlogical models to logical models.

1. Precise definitions of nonlogical data models.
2. Possibility to compare the representational powers of nonlogical models.
3. Defines precisely the concept of an answer to a query. Hence to prove the correctness of proposed evaluation algorithms for a query.
4. Defines satisfiability of an integrity constraint. Hence, to prove the correctness of proposed integrity maintenance algorithms.

The emphasis in using FOL should be on mental hygiene, rather than on theorem proving. FOL should be used if there are doubts about the clear semantics of new database concepts (*e.g.*, incomplete information). When the semantics of a construct is clear, go back to a level that is convenient for data modelling.

Reiter's chapter, which draws examples from DB literature, makes several questionable assumptions about databases. Examples used in the DB literature do not reflect the size or complexity of real databases. A database is a large collection of data, and database applications form large systems. Toy examples do not illustrate the real problems of designing large database applications. For example, census databases are very complex and have several hundred record types, each with several thousand attributes. Database design and definition become extremely complex. The large number of axioms needed to handle the complexity of database applications, coupled with the size of databases, raises serious questions about the practicality of using FOL for real database applications.

Ray Reiter: "The above comment misses the point of the chapter, which provides a framework for defining the semantics of data models. Implementation issues are not addressed nor is the complexity of actual applications except insofar as this complexity has to do with *representational* issues. If you are trying to capture complex *semantic* properties of the real world in a *data model* then use FOL to define that data model. That is all the chapter says."

The following characterization was proposed from the PL point of view: The essence of database is captured by sets and set oriented operations, and the whole database problem is really just an efficiency problem. The response: The characterization ignores many important aspects. Capturing structural properties by sets is a reasonable thing to do, but much of the semantics cannot be captured by using sets. The concept of transactions is ignored. DB has been in the set oriented framework for a long time. Now more difficult problems are being addressed (*i.e.*, many basic problems faced in PL embedded in data intensive applications).

In conclusion, it is fair to say that some formal system is better than no formal system. Model theory and proof theory have been discussed, but other formal systems are also candidates.

References

[AV80] Apt, K.R. and M.H. Van Emden, "Contributions to the Theory of Logic Programming," Research Report CS-80-12, Department of Computer Science, University of Waterloo, Ontario, Canada, 1980.

[BBC80] Bernstein, P.A., B.T. Blaustein and E.M. Clarke, "Fast Maintenance of Integrity Assertions Using Redundant Aggregate Data," *Proc. 6th International Conference on Very Large Databases,* Montreal, Quebec, Canada, October 1980.

[BISK81] Biskup, J., "Null Values in Data Base Relations," in [GM78].

[BRUC75] Bruce, B., "Case Systems for Natural Language," *Artificial Intelligence,* Vol. 6, pp. 327-360, 1975.

[BZ81] Brodie, M.L. and S.N. Zilles (eds.), *Proc. Workshop on Data Abstraction, Databases, and Conceptual Modelling, SIGART Newsletter,* No. 74, January 1981; *SIGMOD Record,* Vol. 11, No. 2, February 1981; *SIGPLAN Notices,* Vol. 16, No. 1, January 1981.

[CL73] Chang, C.L. and R.C.T. Lee, *Symbolic Logic and Mechanical Theorem Proving,* Academic Press, New York, 1973.

[CLAR78] Clark, K.L., "Negation as Failure," in [GM78].

[CODD70] Codd, E.F., "A Relational Model of Data for Large Shared Data Banks," *Communications of the ACM,* Vol. 13, No. 6, pp. 377-387, June 1970.

[CODD72] Codd, E.F., "Relational Completeness of Database Sublanguages," in R. Rustin (ed.), *Data Base Systems,* Prentice-Hall, Englewood Cliffs, NJ, 1972.

[CODD79] Codd, E.F., "Extending the Database Relational Model to Capture More Meaning," *ACM Transactions on Database Systems,* Vol. 4, No. 4, December 1979, pp. 397-434; IBM Research Report RJ2599, San Jose, CA, August 1979.

[GM78] Gallaire, H. and J. Minker (eds.), *Logic and Data Bases,* Plenum Press, New York, 1978.

[HM78] Hammer, M. and D. McLeod, "The Semantic Data Model: A Modelling Mechanism for Database Applications," *Proc. 1978 ACM SIGMOD International Conference on the Management of Data,* Austin, TX, May–June 1978.

[JACO82] Jacobs, B.E., "On Database Logic," *Journal of the ACM,* Vol. 29, No. 2, April 1982, pp. 310-332.

[KOWA79] Kowalski, R., *Logic for Problem Solving,* Elsevier North-Holland, New York, 1979.

[KUHN67] Kuhns, J.L., "Answering Questions by Computer—A Logical Study," Memorandum RM 2428 PR, Rand Corporation, Santa Monica, CA, December 1967.

[LIPS79] Lipski, W., Jr., "On Semantic Issues Connected with Incomplete Information Databases," *ACM Transactions on Database Systems,* Vol. 4, No. 3, September 1979, pp. 262-296.

[MBW80] Mylopoulos, J., P.A. Bernstein and H.K.T. Wong, "A Language Facility for Designing Interactive Database-Intensive Applications," *ACM Transactions on Database Systems,* Vol. 5, No. 2, June 1980, pp. 27-39.

[MCCA80] McCarthy, J., "Circumscripton—A Form of Non-Monotonic Reasoning," *Artificial Intelligence,* Vol. 13, Nos. 1 and 2, April 1980, pp. 27-39.

[MEND64] Mendelson, E., *Introduction to Mathematical Logic,* Van Nostrand, Princeton, NJ, 1964.

[RAPH71] Raphael, B., "The Frame Problem in Problem-Solving Systems," in N.V. Findler and B. Meltzer (eds.), *Artificial Intelligence and Heuristic Programming,* Edinburgh University Press, Edinburgh, Scotland, 1971.

[REIT77] Reiter, R., *An Approach to Deductive Question-Answering,* BBN Technical Report 3649, Bolt, Beranek and Newman, Inc., Cambridge, MA, September 1977.

[REIT78a] Reiter, R., "Deductive Question-Answering on Relational Databases," in [GM78], pp. 149-177.

[REIT78b] Reiter, R., "On Closed World Data Bases," in [GM78], pp. 55-76.

[REIT80a] Reiter, R., "Equality and Domain Closure in First Order Databases," *Journal of the ACM,* Vol. 27, No. 2, 1989, pp. 235-249.

[REIT80b] Reiter, R., "Databases: A Logical Perspective," in [BZ80], pp. 174-176.

[REIT80c] Reiter, R., "A Logic for Default Reasoning," *Artificial Intelligence,* Vol. 13, 1980, pp. 81-132.

[SCHU76a] Schubert, L.K., "Extending the Expressive Power of Semantic Networks," *Artificial Intelligence,* Vol 7, No. 2, Summer 1976, pp. 163-198.

[SOWA76] Sowa, J.F., "Conceptual Structures for a Database Interface," *IBM Jornal of Research and Development,* Vol. 20, No. 4, July 1976, pp. 336-357.

[SS77b] Smith, J.M. and D.C.P. Smith, "Database Abstractions: Aggregation and Generalization," *ACM Transactions on Database Systems,* Vol. 2, No. 2, June 1977, pp. 105-133.

[TL82] Tsichritzis, D. and F. Lochovsky, *Data Models,* Prentice-Hall, Englewood Cliffs, NJ, 1982.

[ULLM80] Ullman, J.D., *Principles of Database Systems,* Computer Science Press, Potomac, MD, 1980.

[VASS79] Vassiliou, Y., "Null Values in Database Management: A Denotational Semantics Approach," *Proc. 1979 ACM SIGMOD International Conference on Management of Data,* Boston, MA, May 1979, pp. 162-169.

[WALK80] Walker, A., "Time and Space in a Lattice of Universal Relations with Blank Entries," *XPI Workshop on Relational Database Theory,* Stony Brook, NY, June–July 1980.

[ZANI77] Zaniolo, C., "Relational Views in a Database System; Support for Queries," *Proc. IEEE Computer Applications and Software Conference,* Chicago, IL, November 1977, pp. 267-275.