Working Paper

RELATIONAL DATABASES, LOGICAL DATABASES AND THE ENTITY-RELATIONSHIPS APPROACH

Ronald M. Lee

December 1981
WP-81-164

International Institute for Applied Systems Analysis
A-2361 Laxenburg, Austria

RELATIONAL DATABASES, LOGICAL DATABASES AND THE ENTITY-RELATIONSHIPS APPROACH

Ronald M. Lee

December 1981
WP-81-164

INTERNATIONAL INSTITUTE FOR APPLIED SYSTEMS ANALYSIS
2361 Laxenburg, Austria

ABSTRACT

A comparisn of relational databases, as known in Data Management, and logical databases, as used in Artificial Intelligence is made. This comparison is then used to examine certain semantic issues raised by the Entity-Relationship Model.

CONTENTS

RELATIONAL DATABASES, LOGICAL DATABASES AND THE ENTITY-RELATIONSHIPS APPROACH

Ronald M. Lee

I. INTRODUCTION

In Artificial Intelligence (AI), the term database is often used to refer to a 'logical database' of predicate calculus assertions. In contrast to the 'structured' databases of data management (DM), e.g., relational, network, the logical form has certain advantages: queries are data structure independent (there is no data structure), and automatic inferencing is supported (indeed, this is the motivation for using the predicate calculus in AI databases). On the other hand, logical databases have the disadvantage that they require a much more rigorous modeling of the environment. In certain cases this leads to yet unsolved difficulties in logic and philosophy.

In this paper, we compare the logical database form to that of one popular model of structured databases, the relational model Codd (1970).

Aside from its popularity, the relational model, considered in combination with the relational calculus (Codd 1971) helps to minimize the e syntactic differences of the two approaches. As will be seen, the fundamental difference that remains is one of 'ontology'—i.e., the fundamental objects assumed under each framework. It is this difference that accounts for the advantages and limitations of the logical approach.

Based on this comparison, we also examine the 'entity-relationship model' of Chen (1976) which, in prescribing a semantically motivated approach to the design of relational database, serves as an intermediate case to the other two approaches.

For a broader comparison between the perspectives on databases in data management and artificial intelligence see the survey article by Wong et al. (1976). A more in-depth background on logical databases is given in Gallaire and Minker (1978) and other references suggested therein. For a good introduction to various theorem proving techniques used on logical databases, see the later chapters of Nilsson (1980).

II. PREDICATE CALCULUS

a rudimentary familiarity with the predicate calculus (PC) is presumed. (For background, a suggested text is Suppes (1957).) The notation used herein is as follows:

Propositional and predicate constant names are denoted as a capital letter, followed by zero or more other capital letters, digits or embedded hyphens. Predicates in addition have an argument list following the name. E.g., P, Q, RED( ), COMMON-STOCK.

Individual and functional constant names are denoted by the character "@" followed by one or more lower case letters or digits or embedded hyphens.

Individual variables are denoted as single lower case letters followed by an optional numeric subscript, e.g., x, y, z_1, z_2.

Logical connectives:

~	negation
&	conjunction
V	inclusive disjunction
W	exclusive disjunction
→	implication
↔	equivalence

Quantifiers:

∀	universal
∃	existential

Further syntactic rules of well formedness, inference rules and axioms could be given. However, since our use of the PC notation here is largely illustrative, we will omit these aspects. (They are in any case well studied and available in standard logic texts.)

While the PC syntax will play a role later in this paper, the more fundamental point at present is the structure of predicate calculus systems.

The syntax presented above is *uninterpreted*. That is, the range of the individual variables and the definitions of predicates, etc. is so far left unexplained. To construct a logical system using the PC, one begins first by defining a 'universe,' the set of objects which the subsequent logical statements are 'about.' This universe is defined *extra- logically*, i.e., in natural language. For instance, for a PC system applied to mathematics, the universe might be the integer numbers. For a system describing familial relations, the universe would be the set of people.

This universe may be either finite or infinite, but it is presumed that each element therein is identifiable in that a unique name is potentially assignable to each.

The next step is that various primitive predicates and functions are introduced. These too are defined extra-logically.

Next axioms are introduced which indicate the inter-dependence of these primitive predicates, constraining their manipulation.

Also, logical definitions (within the PC) may introduce further *derived* predicates and functions.

III. THE RELATIONAL MODEL AND CALCULUS AS A LOGIC SYSTEM

Codd's Relational Calculus (1971), one of a variety of retrieval languages for relational databases, is based on a predicate calculus syntax.

However, if one consider the relational model and calculus together, we can regard them as a logical system. This is instructive for compareative purposes.

However, because the RC was developed for data retrieval queries, it differs somewhat from the usual PC syntax, which is designated to express true/false assertions.

Basically an RC query has two parts—'qualification expression' which selects some subset of tuples in the database, and a 'target list', which extracts certain attributes of these selected tuples to be returned to the user.

It is the qualification expression part of the query which was a PC syntax. The tuples to be extracted correspond to unbound variables in this expression.

This leads us to look first at the universe of the system. As seen in Codd (1971), the quantified variables range over *tuples* in the database. These ranges are often restricted to certain relations—i.e., the relations serve the role of one-place predicates.

Thus, while Codd uses the notation ($x \in R$) to indicate that the range of variable x is limited to the tuples in relation R, an equivalent notation is R(x).

Beyond this, further restrictions of tuples are made by testing the vales of attributes, i.e., the 'columns' or data items within a tuple. While Codd's examples use only numeric data, we will assume date items to be classified as character or numeric. For character data, tests are limited to equality, whereas numeric tests may also include inequalities (greater than, less than, etc.).

The tuples we are discussing are merely tuples of such data items. Thus these are tests on one or another position in the tuple. Since these positions are named, Codd uses a dot notation to express the test—e.g., x.COLOR = 'RED', y.AGE = 20. We are therefore comparing the content of these positions to some character or numeric data value.

However, under the interpretation as a logical system, this is rather strange since the individuals of the system are assumed to be elementary. Here, the individuals of the system are tuples, in turn composed of lower level elements the positions of the tuple. If we wish to recognize these in the calculus, then our variables should range over these positions as elementary objects. But this would be to ignore the principal value of the relational model, i.e., as a convenient *structuring* of data.

However, we can maintain the calculus with tuples as the basic individuals if we regard the attribute values not as parts of the tuples, but as *functions* mapping from the tuple to a data value character string or number. Under this interpretation, an (equivalent) functional notation seems more appropriate—e.g., COLOR(x) = 'RED', AGE(y) = 20.

Note that while we are still maintaining the notion of a tuple as a locus of properties, we are now ignoring its existence as a structured data object.

Thus, under this logical interpretation, the relational model and calculus is recast as a logical system whose principal universe is the set of database tuples, regarded as elementary (non-decomposable) objects. Two auxilliary universes are added, numbers and character strings, with functions mapping from the universe of tuples to each of these. This is sketched in Diagram 1.

With this extension to multiple universes, we must correspondingly extend the predicate calculus to a *multi- sorted logic*. Essentially, this involves only additional syntactic constraints on the calculus including typing of individual and functional constants, individual variables, quantifiers, and the places of functions and predicates according to

t = tuple

c = character string

n = number.

As observed in Enderton (1972), these additional syntactic constraints of a multi-sorted logic can just as well be controlled within the calculus itself, e.g., using predicates

TUPLE(x)

CHAR-STRING(x)

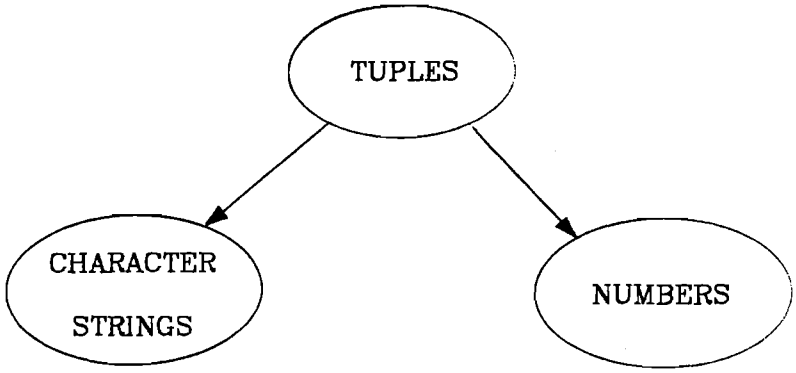

Diagram 1.

NUMBER(x)

so that these extensions do not introduce any substantially different factors from a logical standpoint.

Thus far, we have merely re-interpreted the relational model plus calculus as a predicate calculus system. This is essentially only a syntactic transformation: *any* relational database can be so transformed and the information contained is identical. The important difference is that this form recognizes the tuple as an elementary individual rather than as a composite object with internal structure.

a brief example may help illustrate the correspondence. Consider the following sample database:

R(A,B,C)	S(D,E)	T(F,G,H)
3 X 7	3 2	5 P M
6 Y 3	3 5	2 Q L
	6 5	

Under the logical interpretation, we must first assign names to each of the tuples. Arbitrarily, we name these from top to bottom, left to right as t1, t2, t3, etc.

In logical form the database is re-interpreted as follows:

R(t1) & A(t1) = 2 & B(t1) = 'X' & C(t1) = 7

R(t2) & A(t2) = 6 & B(t2) = 'Y' & C(t2) = 3

S(t3) & D(t3) = 3 & E(t3) = 2

S(t4) & D(t4) = 3 & E(t4) = 5

S(t5) & D(t5) = 6 & E(t5) = 5

T(t6) & F(t6) = 5 & G(t6) = 'P' & H(t6) = 'M'

T(t7) & F(t7) = 2 & G(t') = 'Q' & H(t7) = 'L'

This translation could similarly be reversed. The single place predicates are relation names and the functional maps become attribute names and values. Note that it is the recognition of tuples as logical individuals that allow recovery of the former data structure.

IV. LOGICAL DATABASES

What we have called a 'logical database' is not presented in AI by a single specification analogous to Codd's papers. These have evolved as part of efforts in automatic theorem proving and to a certain extent vary in their notation depending on their implementation framework (e.g., LISP, PLANNER, PROLOG). The predicate calculus, however, serves as the most frequently used abstract specification of their features.

It should be noted that competing with these logical database forms are representations based on semantic networks. (This competition is a little reminiscent of the relational vs. network databases.)

Among the semantic network representations, however, there is much less agreement about representational form, and so we have preferred the predicate calculus. In any case, our comments about the differences between the structured database orientation of DM vs that in logical databases extend for the most part to semantic net type database as well.

A logical database, as we have said, consists of a collection of predicate calculus assertions. The characteristics of objects and their relationships are represented as predicates. (Functions, which map one or more individuals to another, are reparded as a special case of a multiplace predicate which defines a logical relation among individuals.)

As we have seen, a relational database can also be transformed (without loss of information) into a predicate calculus syntax. There is however a fundamental difference in a logical database: the universe to which predicates are applied consists not of data objects (i.e., tuples), but of some set of objects or individuals in the external environment. For instance, consider a logical database that records familial relationships (e.g., of royal families). The universe in the case is the set of persons, living or dead. The basic predicates might be

MALE(x)

SIBLING(x,y)

PARENT(x,y)

with various derived predicates such as the following:

$\forall x$ FEMALE(x) $\Longleftrightarrow$ ~MALE(x)

$\forall x \forall y$ CHILD(y,x) $\Longleftrightarrow$ PARENT(x,y)

$\forall x \forall y$ BROTHER(x,y) $\Longleftrightarrow$ SIBLING(x,Y) & MALE(x)

etc.

These definitions are universal statements, i.e., which hold throughout the universe. Particular facts involve application of these predicates to individuals, e.g.,

MALE(charles)

PARENT(elizabeth, charles),

etc.

The important observation is that the range of variables and the application of predicates is to individuals in the external environment, not to data objects.

V. THE ENTITY-RELATIONSHIP MODEL

Chen (1976) presented a modeling approach for relational databases called the Entity Relationship Model (ERM). The essence of this approach is to identify the types of objects (entities) in the environment whose characteristics are to be described in the database. Properties of single entities are described in 'entity relations' for which each tuple in the database corresponds to an individual object the the environment. Properties applying to more than one object are represented in 'relationship' relations.

This is interesting in that it provides something of a bridge between the relational and logical forms presented here. The step of identifying the basic entities is roughly equivalent to defining the logical universe. in a logical terms, entity relation names correspond to single place predicates, relationship relation names correspond to multi-place predicates.

For instance, suppose we are designing a vehicle registration database. The principal entities are vehicles and persons. The universe is thus the union of these two sets. The basic predicates are

PERSON(x)

VEHICLE(x)

OWN(x,y).

These logical variables range over the union of the sets of vehicles and persons or, *isomorphically*, the union of the tuples in the entity relations VEHICLE and PERSON (but not over relationship relations—the tuples here do not have an object counterpart).

Thus, in the ERM perspective, the former universe of tuples is replaced by one of 'entities,' as shown in Diagram 2.

VI. LOGICAL INTERPRETATION OF ATTRIBUTES

In the ERM, relational attributes are treated as functional maps to 'value sets' of character strings or numbers, much as in or earlier reformulation of the relational model. This, in our view, misses the observation that these attributes values are also qualities about the objects in the universe, potentially formalizable in the logical system. Rather than the usual character/number distinction, we classify these relational attributes according to the information they convey about the objects in the universe: as labels, numeric measures and predicate groups.

A. Label Functions

In both the relational model and Chen's interpretation of it, the notion of an identifying key plays an important role. In the relational model these are attributes which allow the unique identification of each tuple. In the ERM, these are attributes which allow the unique identification of each entity. In logical form, this corresponds to a logical constant name. However, in both the RM and ERM, these identifiers are provided by outside sources—e.g., social security number, part number, etc. In the logical form, the role of such identifiers is carried by the logical constant names.

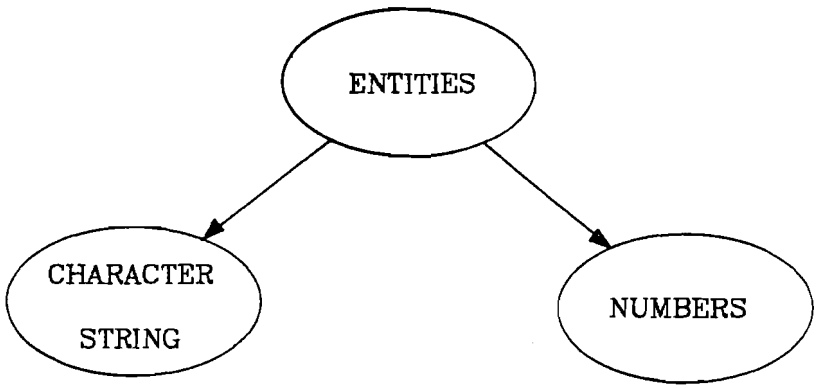

Diagram 2

However, a given object may have more than one such identifier—e.g., a car may have a manufacturer's identification code plus the license number assigned to it by the state. Further, such identifiers may change, e.g., a car gets a new license number, and we then need another mechanism to reflect the continued identity of the object. For this reason, we find it preferable to separate the roles of identification of objects within the system and the identifiers used by external parties. That is, logical constant names are taken to be 'internally generated' identifiers, i.e., invented for use within the logical database.

Thus, in logical form, the unique characteristic of such 'key' attributes is not important. These are however useful to maintain in the database since they provide identifying information to external users. We therefore distinguish these types of data values as what we will call 'labels' and relate them to logical individuals by means of 'label functions.' E.g., the social security number for an individual, a, might be

SSN(a) = '474-52-4829'.

A label, as we said, serves only as identifying information to external parties. While identifying keys are thus labels, there are also labels which are not uniquesly identifying, the most common example being the names of persons. For instance, for a person, @b, we might have

FIRST-NAME(@b) = "JULIUS"

LAST-NAME(@b) = "CAESAR".

Labels are denoted here within double quotes.

These may be either characters, as with numeric identification codes. However, in the latter case, their role continues to be purely for identification—no arithmetic operations are allowed.

B. Measurement Functions

Label functions mapped from one or more entities to a character string. Correspondingly, a measurement function is a mapping from one or more entities to a number.

WEIGHT-IN-KG(x) = 90

WEIGHT-IN-FEET(y) = 6.

To effectively make use of numbers in the calculus, it is useful to employ the typical inequality predicates: $>$, $\geq$, $<$, $\leq$, and the numeric functions +, -, /, *, **. These enable such assertions as

$\forall x$ HEIGHT-IN-CM(x) = 2.54 * HEIGHT-IN-INCHES(x)

WEIGHT-IN-KG#(@john) > WEIGHT-IN-KG#(@mary)

However, one additional feature that is useful to employ in the description of measurement is the addition of additional individuals (in the universe of entities) known as *measurement standards*. As the name suggests, these may be considered as actual objects—e.g., as kept by the National Bureau of Standards, or their indirect proxies. Thus, rather than refer to units of measure implicitly in the choice of function and predicate names, we can treat them explicitly as individuals in the calculus. For instance,

$\forall x$ HEIGHT#(x,@cm) = 2.54 * HEIGHT#(x,@inch)

WEIGHT#(@john,@kg) > WEIGHT#(@mary,@kg).

C. Predicate Groups

So far, we have allowed for a) data used for identification of individuals (as label function) and b) data indicating quantitative features of individuals (as measure function). One other type of data needs to be accounted for, namely that indicating qualitative features of individuals. Examples are the values of such attributes as COLOR, SHAPE, SEX, STATUS, etc.

Note that the distinguishing feature here is not between character or numeric data, but rather how the data is interpreted. For instance, COLOR might be indicated by a numeric code or by such character strings as "RED," "BLUE," etc. Yet, as a number, it is not intended for numeric manipulation, while as a character string, it conveys more than a simple label—it conveys, rather, a *qualitative property* of the object. This is the job of a simple predicate on entities.

Thus, for instance, if COLOR is indicated by numeric codes, 0, 1, 2, etc. we may introduce the equivalences:

$\forall x$ COLOR#(x) = 0 $\leftrightarrow$ WHITE(x)

COLOR#(x) = 1 $\leftrightarrow$ RED(x)

COLOR#(x) = 2 $\leftrightarrow$ BLUE(x)

etc.

Correspondingly, if SHAPE were indicated as a character string, we might have

SHAPE(x) = "ROUND" $\leftrightarrow$ ROUND(x)

SHAPE(x) = "SQUARE" $\leftrightarrow$ SQUARE(x).

Since these kinds of attributes are interpreted logically as simple predicates (rather than label or measure functions), we refer to them as *predicate groups*.

However, in a relational database, collection of these descriptions under one (single valued) attribute conveys one additional piece of information; namely, that the corresponding predicates are mutually exclusive. Thus, the predicates in a predicate group have a corresponding assertion indicating their exclusive disjunction, e.g., of the form

$\forall x$ HAS-COLOR(x) $\leftrightarrow$ WHITE(x) W RED(x) W BLUE(x) W etc.

One further observation can now be made. Earlier, in the introduction, we made the passing comment that logical databases are 'data structure independent.' This claim can now be seen more clearly in that in logical form, no distinction is made between relation names and the values of predicate group attributes—both are interpreted as predicates.

For instance, one form of a personnel database might be

EMPLOYEE(RANK, SSN, AGE, etc.)

where RANK can be CLERK, DRIVER, MANAGER, etc. Alternatively, multiple relations could have been defined, e.g.,

CLERK (SSN, AGE, etc.)

DRIVER (SSN, AGE, etc.)

MANAGER (SSN, AGE, etc.)

However, in logical form, under either interpretation, CLERK, DRIVER, MANAGER, etc. would be interpreted as predicates, with the accompanying assertion.

$\forall x$ EMPLOYEE(x) $\leftrightarrow$ CLERK(x) W DRIVER(x) W MANAGER(x)

VII. CONCLUDING REMARKS: WHAT IS AN ENTITY?

A frequent criticism of the Entity-Relationship Model is that it leaves unanswered the question. What is an entity? As we have seen, the Relational Model avoids this issue entirely since it makes no semantic claims. Interpreted logically, its variables range over tuples, i.e., data objects, and the model makes no reference to other objects outside the universes of tuples, character strings and numbers.

The entity-relationship model, on the other hand, prescribes that certain of these tuples (those in entity relations) designate objects in the environment. The remaining tuples, those in relationship relations, designate relationships between these entities. The question arises, however, what constitute these basic entity classes, and how are these to be

distinguished from their respective properties. Are physical objects the only legitimate entities? It seems not since we often want to designate things like departments, budgets, etc. as entities as well.

How about colors, shapes, numbers, are these entities? In a hospital database, would a disease be an entity?

As we have tried to show, part of the problem here lies in the data structure concept which the ERM has taken over from the relational model. The distinction between database structure vs. database content (i.e., data relations vs. data tuples) is a useful one as regards simple and convenient arrangements of data, but introduces an artificial distinction from a modeling standpoint. (The world does not so conveniently divide itself between structure and content.)

The problem is focused to a somewhat greater degree if we discard this distinction in favor of the logical interpretation introduced here. In this view, the problem is reduced to the definition of some universal concept of entity within which any additional properties, and relationship are ascribed by means of predicates and label and measurement functions. These latter can be viewed as essentially syntactic variants the normal predicate concept (e.g., view WEIGHT(x,@kg) = 10 as abbreviating WEIGHT-IN-KG-EQUAL-10(x)).

The advantage of reformulating the ERM in these logical terms is that it frames the question of entity-hood in a form that has already been given a great deal of study, namely in formal philosophy and logic, under the heading of formal semantics.

Actually, the problem for the ERM, indeed for databases more generally, is considerably simpler than the form in which it is usually studied, namely in ordinary language discourse, where numerous secondary effects such as social roles, conversational context, connotations, etc. also to be considered.

Nonetheless, the issue even in the constrained context of the ERM may still present fundamental difficulties, depending on the context of application. Probably the most thorough treatment of this issue is that of Strawson (1963). Relevant discussion is also to be found in Cresswell (1973) and Rescher (1975).

These issues are also considered in depth in the work on CANDID (Lee (1981)), a knowledge representation language for administrative decision support applications. (The notation presented herein is part of that used in CANDID.) Aspects given special attention in that work are the temporal dependence in the definition of entities, and the creation of so-called 'deontic' entities (comprising contractual and promissory objects).

REFERENCES

Chen, P.-S. C. 1976. The Entity-Relationship Model—Toward a Unified View of Data. ACM Transactions on Database Systems 1(1):9-36.

Codd, E.F. 1970. A Relational Model of Data for Large Shared Data Banks. *Communications of the ACM* 13(June):377-387.

Codd, E.F. 1971. A Data Base Sublanguage Founded on the Relational Calculus. Proceedings of ACM SIGFIDET Workshop on Data Description, Access, and Control, San Diego, California.

Cresswell, M.J. 1973. Logics and languages. London: Nethuen & Co. Ltd.

Enderton, H.B. 1972. A Mathematical Introduction to Logic. New York: Academic Press.

Gallaire, H. and J. Minker. eds. 1978. Logic and Data Bases. New York and London: Plenum Press.

Lee, R.M. 1981. CANDID Description of Commercial and Financial Concepts: A Formal Semantics Approach to Knowledge Representation. Forthcoming WP. Laxenburg, Austria: International Institute for Applied Systems Analysis.

Nilsson, N.J. 1980. Principles of Artificial Intelligence. Palo Alto, California: Tioga Publishing Co.

Rescher, N. 1975. A Theory of Possibility—A Constructivistic and Conceptualistic Account of Possible Individuals and Possible Worlds. Pittsburgh: University of Pittsburgh Press.

Strawson, P.F. 1963. Individuals—An Essay in Descriptive Metaphysics. Garden City, N.Y.: Doubleday and Co.

Suppes, P. 1957. Introduction to Logic. New York: D. van Nostrand Company.

Wong, H.K.T. and J. Mylopoulos. 1976. Two Views of Data Semantics: A Survey of Data Models in Artificial Intelligence and Database Management. A.I. Memo 77-2. Toronto: Department of Computer Science, University of Toronto.