Extending the Database Relational Model to Capture More Meaning

E. F. CODD
IBM Research Laboratory

During the last three or four years several investigators have been exploring "semantic models" for formatted databases. The intent is to capture (in a more or less formal way) more of the meaning of the data so that database design can become more systematic and the database system itself can behave more intelligently. Two major thrusts are clear:

(1) the search for meaningful units that are as small as possible—*atomic semantics*;
(2) the search for meaningful units that are larger than the usual n-ary relation—*molecular semantics*.

In this paper we propose extensions to the relational model to support certain atomic and molecular semantics. These extensions represent a synthesis of many ideas from the published work in semantic modeling plus the introduction of new rules for insertion, update, and deletion, as well as new algebraic operators.

Key Words and Phrases: relation, relational database, relational model, relational schema, database, data model, database schema, data semantics, semantic model, knowledge representation, knowledge base, conceptual model, conceptual schema, entity model
CR Categories: 3.70, 3.73, 4.22, 4.29, 4.33, 4.34, 4.39

1. INTRODUCTION

The relational model for formatted databases [5] was conceived ten years ago, primarily as a tool to free users from the frustrations of having to deal with the clutter of storage representation details. This implementation independence coupled with the power of the algebraic operators on n-ary relations and the open questions concerning dependencies (functional, multivalued, and join) within and between relations have stimulated research in database management (see [30]). The relational model has also provided an architectural focus for the design of databases and some general-purpose database management systems such as MACAIMS [13], PRTV [38], RDMS(GM) [41], MAGNUM [19], INGRES [37], QBE [46], and System R [2].

During the last few years numerous investigations have been aimed at capturing

A version of this work was presented at the 1979 International Conference on Management of Data (SIGMOD), Boston, Mass., May 30–June 1, 1979.
Author's address: IBM Research Laboratory K01/282, 5600 Cottle Road, San Jose, CA 95193.

(in a reasonably formal way) more of the meaning of the data, while preserving independence of implementation. This activity is sometimes called *semantic data modeling*. Actually, the task of capturing the meaning of data is a never-ending one. So the label "semantic" must not be interpreted in any absolute sense. Moreover, database models developed earlier (and sometimes attacked as "syntactic") were not devoid of semantic features (take domains, keys, and functional dependence, for example). The goal is nevertheless an extremely important one because even small successes can bring understanding and order into the field of database design. In addition, a meaning-oriented data model stored in a computer should enable it to respond to queries and other transactions in a more intelligent manner. Such a model could also be a more effective mediator between the multiple external views employed by application programs and end users on the one hand and the multiple internally stored representations on the other.

In recent papers on semantic data modeling there is a strong emphasis on structural aspects, sometimes to the detriment of manipulative aspects. Structure without corresponding operators or inferencing techniques is rather like anatomy without physiology. Some investigations have retained clear links with the relational model and have therefore benefited from inheriting the operators of this model—just as the relational model retained clear links with predicate logic and can therefore inherit its inferencing techniques.

With regard to meaning, two complementary quests are evident:

(1) What constitutes an atomic fact (atomic semantics)?
(2) What larger clusters of information constitute meaningful units (molecular semantics)?

After a review of the relational model, we introduce a classification scheme for entities, properties, and associations. We then discuss extensions to the relational model to reflect this classification and to support such aspects of molecular semantics as abstraction by generalization and by Cartesian aggregation. The extended model is intended primarily for database designers and sophisticated users.

2. THE RELATIONAL MODEL

We shall now give a brief definition of the relational model, in which we emphasize that the algebraic operators are just as much a part of the model as are the structures. The operators permit, among other things, precise discussion of alternative schemata (both base and view) for particular applications of the relational model. We shall also point out the close relationship that exists between the relational model and first-order predicate logic (although it is incorrect to equate the two as in [43]).

To help distinguish relational systems from nonrelational ones, we suggest the following definitions. A database system is *fully relational* if it supports:

(1) the structural aspects of the relational model;
(2) the insert-update-delete rules;
(3) a data sublanguage at least as powerful as the relational algebra, even if all facilities the language may have for iterative loops and recursion were deleted from that language.

A database system that supports (1) and (2), but not (3) is *semirelational.* Note that a fully relational system need not support the relational algebra in a literal sense, but must support its power. Besides being a yardstick of power, the algebra is intended to be a precise intellectual tool for treating such issues as model design, view definition, and restructuring.

2.1 Structures

A *domain* is a set of values of similar type: for example, all possible part serial numbers for a given inventory or all possible dates for the class of events being recorded. A domain is *simple* if all of its values are atomic (nondecomposable by the database management system).

Let $D_1, D_2, \ldots, D_n$ be n ($n > 0$) domains (not necessarily distinct). The *Cartesian product* $\times\{D_i: i = 1, 2, \ldots, n\}$ is the set of all n-tuples $\langle t_1, t_2, \ldots, t_n \rangle$ such that $t_i \in D_i$ for all i. A relation R is defined on these n domains if it is a subset of this Cartesian product. Such a relation is said to be of degree n.

In place of the index set $(1, 2, \ldots, n)$ we may use any unordered set, provided we associate with each tuple component not only its domain, but also its distinct index, which we shall henceforth call its *attribute.* Accordingly, the n distinct attributes of a relation of degree n distinguish the n different uses of the domains upon which that relation is defined (remember that the number of distinct domains may be less than n). A *tuple* then becomes a set of pairs $(A:v)$, where A is an attribute and v is a value drawn from the domain of A, instead of a sequence $\langle v_1, v_2, \ldots, v_n \rangle$.

A *relation* then consists of a set of tuples, each tuple having the same set of attributes. If the domains are all simple, such a relation has a tabular representation with the following properties.

(1) There is no duplication of rows (tuples).
(2) Row order is insignificant.
(3) Column (attribute) order is insignificant.
(4) All table entries are atomic values.

The notation $R(A:a, B:b, C:c, \ldots)$ is used to represent a time-varying relation R having an attribute A taking values from a domain a, an attribute B taking values from a domain b, etc. When, for expository reasons, the domains can be ignored, such a relation will be represented as $R(A, B, C, \ldots)$ or even as R. However, for correct interpretation of an expression (and especially an assignment statement), the order in which attributes are cited may be crucial (see THETA-JOIN below).

A *relational database* is a time-varying collection of data, all of which can be accessed and updated as if they were organized as a collection of time-varying tabular (nonhierarchic) relations of assorted degrees defined on a given set of simple domains. *Base relations* are those which are defined independently of other relations in the database in the sense that no base relation is completely derivable (independently of time) from any other base relation(s). *Derived relations* are those which can be completely derived from the base relations. It is this kind of relation which is normally employed to provide users or application programs with their own *views* of the database. The declared relations may include derived relations as well as all of the base relations. Later, when we have

introduced certain additional concepts, we shall define *semiderived relations*, a class which subsumes the derived relations.

If U is a collection of attributes of a relation, the *U-component* of a tuple t of that relation is the set of $(A:v)$ pairs obtained by deleting from t those pairs having an attribute not in U.

Between tabular relations there are no structural links such as pointers. Associations between relations are represented solely by values. These associations are exploited by high-level operators.

With each relation is associated a set of candidate keys. K is a *candidate key* of relation R if it is a collection of attributes of R with the following time-independent properties.

(1) No two rows of R have the same K-component.
(2) If any attribute is dropped from K, the uniqueness property (1) is lost.

For each base relation one candidate key is selected as the *primary key*. For a given database, those domains upon which the simple (i.e., single-attribute) primary keys are defined are called the *primary domains* of that database. Note that not all component attributes of a compound (i.e., multiattribute) primary key need be defined on primary domains. Primary domains are important for the support of transactions such as "remove supplier 3 from the database," in which we wish to remove 3 wherever it occurs as a supplier serial number, but not in any of its other uses.

All insertions into, updates of, and deletions from base relations are constrained by the following two rules.

Rule 1 (entity integrity): No primary key value of a base relation is allowed to be null or to have a null component.

Rule 2 (referential integrity): Suppose an attribute A of a compound (i.e., multiattribute) primary key of a relation R is defined on a primary domain D. Then, at all times, for each value v of A in R there must exist a base relation (say S) with a simple primary key (say B) such that v occurs as a value of B in S.

The *relational model* consists of

(1) a collection of time-varying tabular relations (with the properties cited above—note especially the keys and domains);
(2) the insert-update-delete rules (Rules 1 and 2 cited above);
(3) the relational algebra described in Sections 2.2 and 2.3 below.

Closely associated with the relational model are various decomposition concepts which are semantic in nature (being time-invariant properties of time-varying relations). Examples of such concepts are nonloss (natural) joins and functional dependencies [6], multivalued dependencies [10, 44], and normal forms. For details see [3] which provides a tutorial on the subject; see also [39].

2.2 Relational Algebra (Excluding Null Values)

Since relations are sets, the usual set operators such as UNION, INTERSECTION, and SET DIFFERENCE are applicable. However, they are constrained to apply only to pairs of *union-compatible* relations, i.e., relations whose attributes

are in a one-to-one correspondence such that corresponding attributes are defined on the same domain. This constraint guarantees that the result is a relation. CARTESIAN PRODUCT is applicable without constraint.

We now define operators specifically for the manipulation of n-ary relations. In what follows R, S denote relations; A, B_1, B_2, C denote collections of attributes; c is a tuple of appropriate degree, and with appropriate domains.

THETA-SELECT (sometimes called RESTRICT)

Let θ be one of the binary relations $<$, $\leq$, $=$, $\geq$, $>$, $\neq$ that is applicable to attribute(s) A and tuple c. Then $R[A\ \theta\ c]$ is the set of tuples of R, each of whose A-components bears relation θ to tuple c. Instead of tuple c, other attribute(s) B of R may be cited, provided that A, B are defined on common domains. Then $R[A\ \theta\ B]$ is the set of tuples of R, each of which satisfies the condition that its A-component bear relation θ to its B-component. When θ is equality (a very common case), the THETA-SELECT operator is simply called SELECT.

Examples of THETA-SELECT

R

A	B	C
p	1	2
p	2	1
q	1	2
r	2	5
r	2	3

$R[A \neq r]$

A	B	C
p	1	2
p	2	1
q	1	2

$R[A = r]$

A	B	C
r	2	5
r	2	3

$R[B > C]$

A	B	C
p	2	1

PROJECTION

$R[A_1, A_2, \ldots, A_n]$ is the relation obtained by dropping all columns of R except those specified by $A_1, A_2, \ldots, A_n$ and then dropping redundant duplicate rows.

Examples of PROJECTION

R

A	B	C
p	1	2
p	2	1
q	1	2
r	2	5
r	2	3

$R[A, B]$

A	B
p	1
p	2
q	1
r	2

$R[B]$

B
1
2

$R[B, C]$

B	C
1	2
2	1
2	5
2	3

We can now define the third class of relations. *Semiderived relations* are those which have a projection (with at least one attribute) that is a derived relation (see weak redundancy in [5]). For example, if $R(A, B)$ is a base relation and $S(A, C)$ is a relation such that

$$S[A] = (R[B = b])[A]$$

and attribute C is defined on a domain not used in any of the base relations

(hence S is not derivable), then S is semiderived. As we shall see, there are many uses for semiderived relations. Note that there is no stipulation that a relational database will be designed to have minimal redundancy, although this is an option that may be chosen. Thus, the declared relations may include semiderived and even derived relations as well as the base relations.

THETA-JOIN

Given relations $R(A, B_1)$ and $S(B_2, C)$ with B_1, B_2 defined on a common domain, let θ be one of the binary relations $=$, $<$, $\leq$, $\geq$, $>$, $\neq$ that is applicable to the domain of attributes B_1, B_2. The theta-join of R on B_1 with S on B_2 is denoted by $R[B_1\ \theta\ B_2]S$. It is the concatenation of rows of R with rows of S whenever the B_1-component of the R-row bears relation θ to the B_2-component of the S-row. When θ is equality, the operator is called EQUI-JOIN. Of all the THETA-JOINS, only EQUI-JOIN yields a result that *necessarily* contains two identical columns (one derived from B_1, the other from B_2). More generally, θ may be permitted to be any binary relation that is applicable to the domain of B_1 and B_2.

Examples of THETA-JOIN

R (A	B	C)
p	1	2
p	2	1
q	1	2
r	2	5
r	3	3

S (D	E)
2	u
3	v
4	u

$R[C = D]S$ (A	B	C	D	E)
p	1	2	2	u
q	1	2	2	u
r	3	3	3	v

$R[C > D]S$ (A	B	C	D	E)
r	3	3	2	u
r	2	5	2	u
r	2	5	3	v
r	2	5	4	u

If the relations being theta-joined have some attribute names in common, the names for the attributes of the resulting relation must be specified. For example, if each of the relations R, S has attributes A, B, and all four attributes are defined on a common domain, we may define several possible theta-joins of R with S. One such definition is:

$$T(D, E, F, G) = R(A, B)[B > B]S(A, B)$$

and, using an order-of-citation convention, this means that the source of values for attribute D in T is attribute A in R. Similarly, for attributes E, F, G in T, the respective sources are attributes B in R, A in S, and B in S.

NATURAL JOIN

This join is the same as EQUI-JOIN except that redundant columns generated by the join are removed. Natural join is the one used in normalizing a collection of relations.

Example of NATURAL JOIN. Relations R, S are those tabulated above.

$R[C*D]S$

A	B	C	E
p	1	2	u
q	1	2	u
r	3	3	v

DIVIDE

Given relations $R(A, B_1)$ and $S(B_2)$ with B_1 and B_2 defined on the same domain(s), then, $R[B_1 \div B_2]S$ is the maximal subset of $R[A]$ such that its Cartesian product with $S[B_2]$ is included in R. This operator is the algebraic counterpart of the universal quantifier.

Example of DIVIDE

R

A	B
p	1
p	2
p	3
q	1
r	1
r	3

S

C
1
3

$R[B \div C]S$

A
p
r

2.3 Extensions of the Algebra for Null Values

The two most important types of null value have the meanings "value at present unknown" and "property inapplicable." An approach that handles both types of nulls is described in [40]. A rather general attack on the problem of dealing with partial information is described in [22]. Here, we shall concern ourselves with only the "value at present unknown" type of null and denote it by ω (see [5] for more details). The following treatment should be regarded as preliminary and in need of further research.

In the basic relational model nulls are excluded from every component of a primary key of a base relation. Apart from this constraint, any occurrence of the value-unknown type of null can be replaced in an updating operation by a nonnull value, and vice versa, unless there is an explicit integrity constraint disallowing this.

The first question which arises is: what is the truth value of $x = y$ if x or y or both are null? An appropriate result in each of these cases is the unknown truth value, rather than true or false. Accordingly, we adopt a three-valued logic for use in extracting data from databases that may contain null values. We use the same symbol "ω" to denote the unknown truth value, because truth values can be stored in databases and we want the treatment of all unknown or null values to be uniform. The three-valued logic is based upon the following truth tables:

AND	F	ω	T
F	F	F	F
ω	F	ω	ω
T	F	ω	T

OR	F	ω	T
F	F	ω	T
ω	ω	ω	T
T	T	T	T

NOT(F) = T; NOT(ω) = ω; NOT(T) = F

The existential and universal quantifiers behave like iterated OR and AND, respectively.

With regard to set membership $\in$ and set inclusion $\subseteq$, we assign the truth value ω to the expressions: $\omega \in S$ and $\{\omega\} \subseteq S$, whenever S is a nonempty unary relation (even if S does contain a null value). This may seem a bit counterintuitive at first, but one way to make it seem more acceptable is to think of each occurrence of ω as a placeholder for a possibly distinct value. To be more precise, a truth-valued expression has the value ω if and only if (after replacing any defined variables by their defining expressions in terms of individual variables) both of the following conditions hold.

(1) Each occurrence of ω in the expression can be replaced by a nonnull value (possibly a distinct one for every occurrence) so as to yield the value T for the expression.
(2) Each occurrence of ω in the expression can be replaced by a nonnull value (possibly a distinct one for every occurrence) so as to yield the value F for the expression.

We shall call this the *null substitution principle*. The three-valued logic described above is consistent with this principle. The following examples illustrate the application of this principle to set membership and set inclusion. Let $\varnothing$ denote the empty set and R, S, T, U, V denote the following relations:

R	S	T	U	V
ω	ω	ω 1	x ω	x ω
1	1	y ω	ω 3	y 3
	2			z 1

The following expressions have the truth value F:

$$\omega \in \varnothing \qquad T \subseteq S \qquad V \subseteq U \qquad U \subseteq R.$$

The following expressions have the truth value ω:

$$R \subseteq S \qquad S \subseteq R \qquad T \subseteq U \qquad U \subseteq T.$$
$$T \subseteq V \qquad U \subseteq V$$

In passing, we note that this scheme for nulls has certain properties which may appear paradoxical at first. For example, take the relation EMP with attributes NAME and AGE. The expression

$$(\text{EMP}[\text{AGE} \leq 50] \cup \text{EMP}[\text{AGE} > 50])[\text{NAME}]$$

does not necessarily yield the set of all employee names. If, however, we interpret the term EMP[AGE $\leq$ 50] as the set of tuples in EMP whose AGE-component is known in the database to be less than or equal to 50, and EMP[AGE > 50] as the set whose AGE-component is known to be greater than 50, the paradoxical aspect disappears. This kind of interpretation does not require that all of the tautologies of two-valued logic be preserved by the three-valued logic (contrast with [40]).

By applying the null substitution principle to inequality testing, we can avoid the arbitrary step of giving ω any place in a numerical or lexicographic ordering. In accordance with this principle, we assign the truth value ω to the expressions $x\ \theta\ y$, where θ is any one of $<, \leq, \geq, >$ whenever x or y is null.

For every positive integer n, the n-tuple consisting of n null values (each of course accompanied by its attribute) is a legal tuple, but a nonbase n-ary relation may contain at most one such tuple, and a base relation cannot contain such a tuple at all. As usual, no relation may contain duplicate tuples. In applying this nonduplication rule, a null value in one tuple is regarded as the same as a null value in another. This identification of one null value with another may appear to be in contradiction with our assignment of truth value to the test $\omega = \omega$. However, tuple identification for duplicate removal is an operation at a lower level of detail than equality testing in the evaluation of retrieval conditions. Hence, it is possible to adopt a different rule. The consequences for UNION, INTERSECTION, and DIFFERENCE are illustrated below.

R		S		$R \cup S$		$R \cap S$		$R - S$	
ω	ω	ω	ω	ω	ω	ω	ω		
u	ω	u	ω	u	ω	u	ω		
u	1	u	1	u	1	u	1		
ω	1			ω	1			ω	1

Now, let us look at the effect of this type of null upon the remaining operators of the relational algebra. CARTESIAN PRODUCT remains unaffected. PROJECTION behaves as expected, provided that one remembers how the nonduplication rule is applied to tuples with null-valued components. The following examples illustrate projection.

	R		$R[B, C]$		$R[C]$
A	B	C	B	C	C
u	ω	ω	ω	ω	ω
v	1	ω	1	ω	
w	ω	1	ω	1	1
x	1	ω			
y	ω	1			

The THETA-JOIN operator entails concatenation of pairs of tuples subject to some specified condition θ holding between certain components of these tuples. The evaluation of the condition for any candidate pair of tuples yields the truth value F or ω or T. We retain the join operator that concatenates only those pairs of tuples for which the condition evaluates to T and call it a TRUE THETA JOIN. In addition, we introduce a MAYBE THETA JOIN that concatenates only those pairs of tuples for which the specified condition evaluates to ω.

The MAYBE version of an operator is denoted by placing the symbol ω after the theta symbol (e.g., $=\omega$) or operator symbol (e.g., $\div\omega$). The following examples illustrate the TRUE and MAYBE EQUI-JOINs and the TRUE and MAYBE LESS-THAN JOINs.

R	
A	B
u	ω
ω	2
w	1

S
C
ω
2

$R[B=C]S$		
A	B	C
ω	2	2

$R[B=\omega C]S$		
A	B	C
u	ω	ω
u	ω	2
ω	2	ω
w	1	ω

$R[B<C]S$		
A	B	C
w	1	2

$R[B<\omega C]S$
A B C
same as $R[B=\omega C]S$

If we wish to select only those rows of R that have ω as their B-component, we may form the MAYBE EQUI-JOIN of R with a relation T whose only element is a single nonnull value (any such value will do, provided it is drawn from the same underlying domain that attribute B is defined on) and then PROJECT the result on A, B. In the case above, the reader can verify that the final result is a relation whose only element is the pair $(A{:}u, B{:}\omega)$. Treatment of null values by the THETA-SELECT operator (TRUE and MAYBE versions) follows the same pattern as the THETA-JOIN operators.

DIVISION is treated in a similar manner. The original operator based upon true inclusion (inclusion testing that yields T) is retained and called TRUE DIVISION. A new division operator $\div\omega$ is introduced which entails only maybe inclusion (inclusion testing that yields ω), and this is called MAYBE DIVISION. The following examples illustrate the two kinds of division.

R	
A	B
u	1
u	2
u	3
w	2
w	ω
z	3

S
C
2
3

T
C
2
ω

$R[B \div C]S$
A
u

$R[B \div C]T$
A
empty

$R[B \div \omega C]S$
A
w

$R[B \div \omega C]T$
A
u
w

The following operator permits two relations to be subjected to union, even if they are not union-compatible. Nevertheless, the result is always a relation.

OUTER UNION

Let R, S be relations which have attribute(s) B in common and no others. Let the remaining attribute(s) of R be A, and those of S be C. Let

$$R_1(A, B, C) = R \times (C{:}\omega)$$

$$S_1(A, B, C) = (A{:}\omega) \times S$$

where × denotes Cartesian product. The outer union of R and S is given by

$$R \textcircled{\cup} S = R_1 \cup S_1.$$

Note that in the special case that R and S are union-compatible,

$$R \textcircled{\cup} S = R \cup S.$$

Example of OUTER UNION

R	A	B	C
	p	1	2
	p	2	1
	q	1	2

S	B	D
	2	u
	3	v

$R \textcircled{\cup} S$	A	B	C	D
	p	1	2	ω
	p	2	1	ω
	q	1	2	ω
	ω	2	ω	u
	ω	3	ω	v

In a similar manner, we could define OUTER versions of INTERSECTION and DIFFERENCE also.

Both the NATURAL and EQUI-JOINs lose information when the relations being joined do not have equal projections on the join attributes. To preserve information regardless of the equality of these projections, we need joins that can generate null values whenever necessary. Such joins were proposed independently in [16, 20, 23, 44].

OUTER THETA-JOIN

Given relations $R = R(A, B_1)$ and $S = S(B_2, C)$ with B_1, B_2 defined on a common domain, let

$$T = R[B_1 \, \theta \, B_2]S$$

$$R_1 = R - T[A, B_1]$$

$$S_1 = S - T[B_2, C].$$

Then the outer theta-join is defined by

$$R[B_1 \textcircled{\theta} B_2]S = T \cup (R_1 \times (B_2{:}\omega, C{:}\omega)) \cup ((A{:}\omega, B_1{:}\omega) \times S_1)$$

where ∪ denotes union and × denotes Cartesian product.

Example of OUTER EQUI-JOIN

S	$S\#$	$SCITY$
	$s1$	$c4$
	$s2$	$c2$
	$s4$	$c1$
	$s6$	$c1$
	$s7$	$c3$

J	$J\#$	$JCITY$
	$j1$	$c1$
	$j2$	$c2$
	$j3$	$c2$
	$j4$	$c5$

Define $SJ = S[SCITY \textcircled{=} JCITY]J$

SJ (	$S\#$	$SCITY$	$JCITY$	$J\#$)
	$s1$	$c4$	ω	ω
	$s2$	$c2$	$c2$	$j2$
	$s2$	$c2$	$c2$	$j3$
	$s4$	$c1$	$c1$	$j1$
	$s6$	$c1$	$c1$	$j1$
	$s7$	$c3$	ω	ω
	ω	ω	$c5$	$j4$

OUTER NATURAL JOIN

Given relations $R(A, B_1)$ and $S(B_2, C)$ as before, and relations T, R_1, S_1 defined as above with = replacing theta, then the outer natural join of R on B_1 with S on B_2 is defined by

$$R[B_1 \textcircled{*} B_2]S = T[A, B_1, C] \cup (R_1 \times (C{:}\omega)) \cup ((A{:}\omega) \times S_1).$$

Example of OUTER NATURAL JOIN. Define $T(S\#, CITY, J\#) = S[SCITY \textcircled{*} JCITY]J$ where relations S, J are as tabulated above.

T (	$S\#$	$CITY$	$J\#$)
	$s1$	$c4$	ω
	$s2$	$c2$	$j2$
	$s2$	$c2$	$j3$
	$s4$	$c1$	$j1$
	$s6$	$c1$	$j1$
	$s7$	$c3$	ω
	ω	$c5$	$j4$

In this treatment, if an operator generates one or more nulls, these nulls are always of the type "value at present unknown," which is consistent with the open world interpretation (see Section 3). If we were dealing with relations having a closed world interpretation, the "property inapplicable" type would be more appropriate.

3. RELATIONSHIP TO PREDICATE LOGIC

We now describe two distinct ways in which the relational model can be related to predicate logic. Suppose we think of a database initially as a set of formulas in first-order predicate logic. Further, each formula has no free variables and is in as atomic a form as possible (e.g, A & B would be replaced by the component formulas A, B). Now suppose that most of the formulas are simple assertions of the form $Pab \cdots z$ (where P is a predicate and a, b, . . . , z are constants), and that the number of distinct predicates in the database is few compared with the number of simple assertions. Such a database is usually called *formatted*, because the major part of it lends itself to rather regular structuring. One obvious way is to factor out the predicate common to a set of simple assertions and then treat the set as an instance of an n-ary relation and the predicate as the name of the relation. A database so structured will then consist of two parts: a regular part consisting of a collection of time-varying relations of assorted degree (this is

sometimes called the *extension*) and an irregular part consisting of predicate logic formulas that are relatively stable over time (this is sometimes called the *intension*, although it may not be what the logicians Russell and Whitehead originally intended by this word). One may also view the intension as a set of integrity constraints (i.e., conditions that define all of the allowable extensions) and thus decouple these notions from variability with time.

One may choose to interpret the absence of an admissible tuple from a base relation as a statement that the truth value of the corresponding atomic formula is (1) unknown; (2) false. If (1) is adopted, we have the *open world interpretation*. If (2) is adopted, we have the *closed world interpretation* (see [28]). Although the closed world interpretation is usually the one adopted for commercial databases, there is a case for permitting some relations (e.g., P-relations of Section 7) to have the open world interpretation, while others (e.g., E-relations for kernel entity types to be discussed in Sections 5 and 6) have the closed world interpretation.

Whether the open or closed interpretation is adopted, the relational model is closely related to predicate logic. It is this closeness which accounts for the plethora of relational data sublanguages that are based on predicate logic. For a probing and thorough comparison of such languages, see [20, 27].

Undisciplined application of predicate logic in designing a database could yield an incomprehensible and unmanageable set of assertions. Some issues which arise when attempting to introduce discipline are the following.

(1) Can we be more precise about what constitutes a simple assertion?
(2) What other regularities can be exploited in a formatted database?
(3) To what extent can these additional regularities be represented in readily analyzable data structures as opposed to procedures?

In attempting to provide an answer to these questions, we shall employ popular informal terms like "entity," "property," and "association" to motivate extensions to the relational model. Eventually, we arrive at a formal system called RM/T (T for Tasmania, where these ideas were first presented [9]). This system can be interpreted in many different ways. Certain interpretations should satisfy the so-called 2-concept school in semantic modeling, while others should satisfy the 3-concept school (see [25, p. 27]).

4. DESIGNATION OF ENTITIES

The need for unique and permanent identifiers for database entities such as employees, suppliers, parts, etc., is clear. User-defined and user-controlled primary keys in the relational model were originally intended for this purpose. There are three difficulites in employing user-controlled keys as *permanent surrogates* for entities.

(1) The actual values of user-controlled keys are determined by users and must therefore be subject to change by them (e.g., if two companies merge, the two employee databases might be combined with the result that some or all of the serial numbers might be changed).
(2) Two relations may have user-controlled keys defined on distinct domains

(e.g., one uses social security, while the other uses employee serial number) and yet the entities denoted are the same.

(3) It may be necessary to carry information about an entity either before it has been assigned a user-controlled key value or after it has ceased to have one (e.g., an applicant for a job and a retiree).

These difficulties have the important consequence that an equi-join on common key values may not yield the same result as a join on common entities. A solution—proposed in part in [4] and more fully in [14]—is to introduce entity domains which contain system-assigned surrogates. Database users may cause the system to generate or delete a surrogate, but they have no control over its value, nor is its value ever displayed to them.

Surrogates behave as if each entity (regardless of type) has its own permanent surrogate, unique within the entire database. Actually, under the covers, such surrogates may have to be changed (e.g., when two previously independent databases are combined into one), but the following property is preserved at all times: Two surrogates are equal in the relational model if and only if they denote the same entity in the perceived world of entities. Note that the system would create distinct surrogates for two entities as a result of user input that, in effect, asserts the distinctness of these entities. A special *coalescing command* enables a user to tell the system that two objects that were previously asserted to be distinct, are, in fact, one and the same.

In any RM/T database one of the underlying domains serves as the source of all surrogates; this is called the *E-domain*. Any attribute defined on the E-domain is called an *E-attribute*. For easy recognition of such attributes, we adopt the convention that they are given names ending in the special character "¢."

Introduction of the E-domain, E-attributes, and surrogates does not make user-controlled keys obsolete. Users will often need entity identifiers (such as part serial numbers) that are totally under their control, although they are no longer compelled to invent a user-controlled key if they do not wish to.

They will have to remember, however, that it is now the surrogate that is the primary key and provides truly permanent identification of each entity. The capability of making equi-joins on surrogates implies that users see the headings of such columns but not the specific values in those columns.

5. ENTITY TYPES

Entities may, of course, have several types (e.g., a supplier may also be a customer). When information regarding an entity is first entered into a database, the input must specify at least one type for that entity—it need not specify anything more unless it is of a type used to describe some other entity (in which case the entity whose description is being augmented must also be specified). In subsequent sections we shall deal with automatic inference of other applicable types when these are inferable from the given one(s).

In any RM/T database there is a unary relation (called an *E-relation*) for each entity type. As a matter of convention, the relation is given the same name as the entity type which the relation represents, while its sole attribute is named by appending the character "¢" at the end of the relation name. Such an attribute is also given additional names (aliases) if the corresponding entity type is a subtype

of other entity types. In such a case, there is one alias for each superentity type, and this alias consists of the relname of the supertype followed by the character "ℓ."

The main purpose of an E-relation is to list all the surrogates of entities that have that type and are currently recorded in the database. One reason for establishing these E-relations explicitly is that an entity may change type dynamically. A firm that was both a supplier and a customer may become just a supplier. We shall see other reasons below.

The possibility that an entity may change its type or types means that we must distinguish two purposes for removal of an entity surrogate from an E-relation:

(1) complete removal of the entity from the database, which means deleting tuples wherever its surrogate appears in a unique tuple identifier role and replacing all other occurrences by a special surrogate E-null that means "entity unknown" [26];
(2) dynamic loss of one type for an entity accompanied by the survival of some other type for that same entity, which means removal of its surrogate from the E-relation for that type and from E-relations for certain other types implied by the type being lost but not implied by the types being retained—this will become clearer later—plus corresponding tuple deletions and surrogate replacements as in (1), but excluding those that are associated with the entity in its remaining types.

Rule 3 (entity integrity in RM/T): In conformity with the ground rules for surrogates, E-relations accept insertions and deletions, but not updates. In conformity with Rule 1 for the basic relational model, E-relations do not accept null values.

6. CLASSIFICATION OF ENTITIES AND ASSOCIATIONS

Entities and their types can be classified by whether they

(1) fill a subordinate role in describing entities of some other type, in which case they are called *characteristic*;
(2) fill a superordinate role in interrelating entities of other types, in which case they are called *associative*;
(3) fill neither of the above roles, in which case they are called *kernel*.

Entities and their types may be related to one another by criteria other than description and association used above. Entity type e_1 is said to be a *subtype* of entity type e_2 if all entities of type e_1 are necessarily entities of type e_2. For example, in a database dealing with employees in general and salesmen employees in particular, the entity type salesman would be a subtype of the entity type employee. Any entity type (characteristic, kernel, or associative) may have one or more subtypes, which in turn may also have subtypes. A subtype of a characteristic entity type is also characteristic; a subtype of a kernel entity type is also kernel; and a subtype of an associative entity type is also associative.

Those kernel entity types that are not subtypes of any other entity type are called *inner kernel*. Each inner kernel entity type is defined independently of all other entity types. Barring any integrity constraints that are specialized to a particular database (as opposed to integrity constraints that are inherent in and

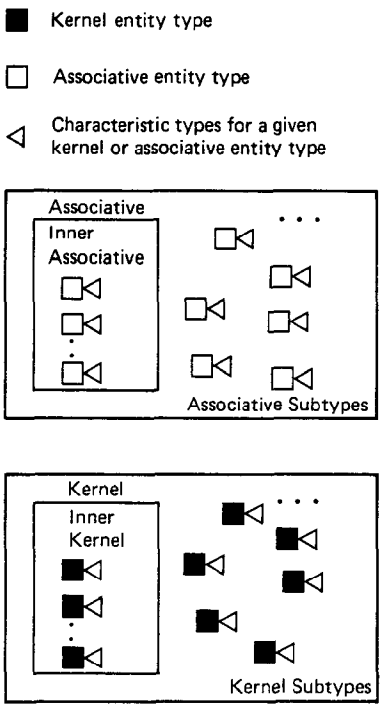

Fig. 1. Classification of entity types

a fundamental part of the data model itself), an inner kernel entity is not existence dependent on any other entity of any type.

Objects which interrelate entities but do not themselves have the status of entities will be called *nonentity associations*. The main distinction between associative entities and nonentity associations is this: Associative entities, like kernel entities, are allowed to have characteristic entities as well as immediate properties, whereas nonentity associations are allowed to have immediate properties only. These and other differences discussed below stem from the difficulty of specifying a cross reference to a particular association when it has no surrogate identifying it uniquely. The prime reason for including nonentity associations in RM/T is an expository one: to show how weak these associations are in contrast to associative entities.

Figure 1 represents the classification of entity types in a simplified way (it does not show that characteristic entity types may themselves have subtypes). Note that the term *inner associative entity type* is applied to an associative entity type that is not the subtype of any other entity type.

This classification scheme is similar in some respects, but certainly not identical, to classifications introduced in [32, 42]. Schmid and Swenson included nonentity associations in their scheme, but not associative entities—in RM/T the former are dispensable, while the latter are indispensable.

7. ENTITIES AND THEIR IMMEDIATE PROPERTIES

We have seen that the E-relation for a given entity type asserts the existence of those entities having that type. The immediate (single-valued) properties of an

entity type are represented as distinctly named attributes of one or more property-defining relations, called P-relations. Each P-relation has as its primary key an E-attribute whose main function is to tie the properties of each entity to the assertion of its existence in the E-relation. Each surrogate appearing in this E-attribute uniquely identifies the entity being described. Furthermore, it uniquely identifies the tuple of which it is part because the properties are single valued. The naming of attributes of P-relations conforms to the following convention: For any entity type e and any pair of P-relations for e, the only attributes these relations have in common are their primary keys.

The role of this E-attribute is that of a unique identifier for the relation in which it appears. We shall call this role the *K-role*. Accordingly, each P-relation has exactly one E-attribute that has the K-role. Such a relation may have one or more other E-attributes, but their roles are purely *referential*, i.e., that of a foreign key rather than a primary key.

Insertions into P-relations and deletions from E-relations are governed by the following rule.

Rule 4 (property integrity): A tuple t may not appear in a P-relation unless the corresponding E-relation asserts the existence of the entity which t describes. In other words, the surrogate primary key component of t must occur in the corresponding E-relation.

There has been much debate about whether the immediate properties of an entity should be represented together in one property-defining relation (one extreme) or split into as many binary relations as there are properties to be recorded (the other extreme). The first is in accord with the PJ/NF [11] discipline, while the second conforms to the *irreducible relation* approach [12, 29]. The normal forms (other than 1NF) are not mandatory—they are merely guidelines for database design. Both the original relational model and RM/T leave this decision to the model user. RM/T (and to a lesser extent RM) provides operators to convert from one form to the other.

In database definition one advantage of binary P-relations is that each corresponding property has a relation name, an attribute name, and a domain name, all of which can be exploited to mnemonic advantage. A second *claimed* advantage for binary P-relations is that the addition of a new property type to the database can be effected by mere addition of one more P-relation. However, in RM/T this advantage is applicable no matter whether the properties are presently organized into binary relations exclusively or n-ary relations of assorted degrees.

The reader is cautioned to avoid jumping to the conclusion that binary relations are somehow superior to n-ary relations as a representational primitive. Even with immediate properties, there are questionable decompositions. Figure 2 shows one organization for the immediate properties of employees. In this and similar examples we may wish to decompose property relations no further than *minimal meaningful units*. Should, for example, the day, month, and year components of a date be represented in separate binary P-relations? Should the street number, street name, city, and state components of an address be so separated? Besides using the notion of minimal meaningful unit, we may wish to adopt the criterion of avoiding occurrences of the "property inapplicable" null value; this objective can often be reached without binary atomization.

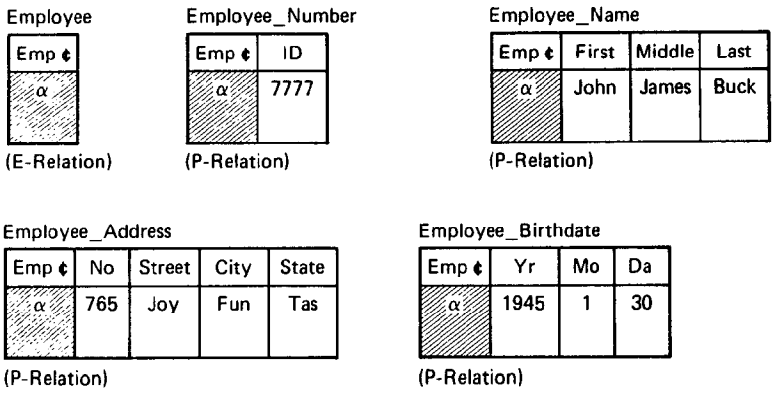

Fig. 2. Entity and property relations

Even if the principal schema were based exclusively on binary relations (and we shall return to this topic in a later section), there would still be a need to apply n-ary joins to obtain higher degree relations in order to define views, study view integration, and represent a broad class of queries. With RM/T we take the position that one man's minimal meaningful unit is not necessarily another's.

Note that the appropriate join for defining a view that encapsulates some or all of the immediate properties of an entity type in a single n-ary relation is the OUTER NATURAL JOIN of all P-relations for this type on the E-attributes with the K-role (see Example A in Section 15.4). This join is appropriate no matter how fine or coarse the property decomposition is.

To explain how the P-relations for a given entity type are tied to the E-relation for that type, we shall make use of the following RM/T objects and properties. The *relname* of a relation is the character string representation of the name of that relation. The relname of a (presumably transient) relation, to which an assignment has not been made, is null. Every base relation has a nonnull relname. Further, every derived relation which is cited on the left-hand side of an assignment statement has a nonnull relname. The *relname domain* (abbreviated RN-domain) is the domain of all relnames in the database.

Now we introduce the *property graph relation* (PG-relation) that indicates which P-relations represent property types associated with which E-relation.

Both of the attributes of PG are defined on the RN-domain. One attribute is named SUB to indicate its subordinate role, while the other is named SUP to indicate its superior role. If m, n are, respectively, the names of a P-relation and an E-relation, let the expressions $p(m)$, $e(n)$ denote the property type represented by that P-relation and the entity type denoted by that E-relation, respectively. The pair (SUB:m, SUP:n) belongs to PG iff $p(m)$ is a property type for entity type $e(n)$.

One may think of the collection of P-relations for a given E-relation as constituting a *property molecule type*, which is bound together by tuples in the PG-relation.

8. MULTIVALUED AND INDIRECT PROPERTIES OF ENTITIES

Entity types are so defined that each multivalued property of an entity p is cast in the form of a characteristic entity q together with immediate properties for q.

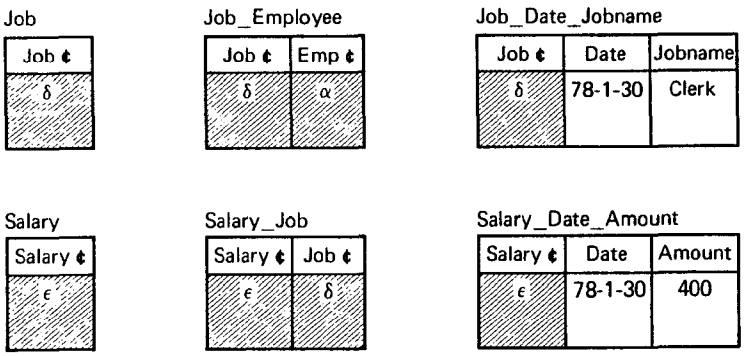

Fig. 3. Characteristic relations

A characteristic entity may itself have one or more characteristic entities subordinate to it. A familiar example is that of employees (a kernel entity type), each of whom has a job history (characteristic entity type subordinate to employees) whose immediate properties are date attained position and name of position. This information is augmented by salary history (characteristic entity type subordinate to job history) whose immediate properties are date of salary change and new salary (see Figure 3).

The need for a characteristic entity type described above arises from a strictly multivalued dependence (i.e., one that is not a functional dependence). Another way in which a characteristic entity type may arise is from a transitive functional dependence [6]. In this case an entity type e has an immediate property p, which in turn has an immediate property q (e.g., a highway segment has one of several types of surface material, which in turn has a porosity). An entity type that is characteristic with respect to highway segments can be introduced to represent the types of surface material on these segments. Porosity then becomes an immediate property of this entity type.

The characteristic entity types that provide description of a given kernel entity type form a strict hierarchy, which we call the *characteristic tree*. In this tree, entity type p is the parent of entity type q if q is an immediate characteristic of p (i.e., not a characteristic of a characteristic of p). A kernel entity type may, of course, have no characteristic entity types describing it. In this case its characteristic tree is a single node, the kernel entity type itself.

To represent the collection of characteristic trees, we introduce the *characteristic graph relation* (CG-relation), a binary relation whose two attributes are defined on the RN-domain, one with the SUB role, the other with the SUP role (as with the PG-relation). Its interpretation is as follows: The pair (SUB:m, SUP:n) belongs to CG if entity type $e(m)$ is immediately subordinate to entity type $e(n)$ in one of the characteristic hierarchies.

Insertion and deletion of characteristic entities are governed by the following rule.

Rule 5 (characteristic integrity): A characteristic entity cannot exist in the database unless the entity it describes most immediately is also in the database.

One may think of the collection of characteristic relations for a given E-relation as constituting a *characteristic molecule type*, which is bound together by tuples in the CG-relation.

9. ASSOCIATIONS

9.1 Associative Entities

The representation of associative entities in RM/T is the same as that of kernel entities. Thus, there is an E-relation for each associative entity type and zero or more P-relations. Figure 4 shows an example of an assignment association between employees and projects, where each assignment is treated as an entity and P-relations are used to record the employee and project surrogates plus the start date of the assignment.

If a given associative entity type has subordinate characteristic entity types, there will be corresponding tuples in the CG-relation to define the tree of these types and there will be characteristic relations to support each of the characteristic entity types involved.

Insertion, update, and deletion of associative entities are governed by the following rule.

Rule 6 (association integrity): Unless there is an explicit integrity constraint to the contrary, an associative entity can exist in the database (i.e., there is a corresponding surrogate in the appropriate E-relation), even though one or more entities participating in that association are unknown. In such a case the surrogate E-null is used to indicate that a participating entity is unknown.

To force automatic deletion of an association when an entity participating in that association is deleted, one may easily add the explicit constraint that the corresponding attribute of an appropriate P-relation cannot accept a null value. Such a constraint is part of the application of RM/T, rather than an integral part of RM/T itself.

An associative entity type interrelates entities of other types (kernel or associative or both). Let us refer to these other types as *immediate participants* in the given associative entity type. To support the specification of which entity types participate in which associative entity types, we introduce the *association graph relation* (AG-relation), a binary relation just like the CG-relation except for its interpretation: (SUB:m, SUP:n) belongs to AG, if the entity type $e(m)$ participates immediately in the definition of associative entity type $e(n)$. Note that the transitive closure of AG is a partial order, but not necessarily a tree or collection of trees.

It is important to observe that when one association type has another association type as a participant, proper use of surrogates in the higher level association for referencing specific lower level participants can remove a potential source of ambiguity (in the same way that proper use of user-controlled keys in the basic relational model can remove such an ambiguity). To illustrate this ambiguity, suppose we have two RM/T relations *IS* and *CAN* each having attributes *S*¢

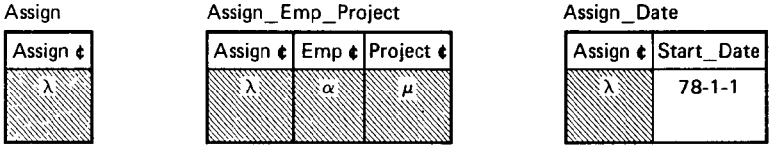

Fig. 4. Associative entity

(supplier surrogates), $P¢$ (part surrogates), and $C¢$ (city surrogates):

$$IS \quad (\; S¢{:}e \quad P¢{:}e \quad C¢{:}e \;)$$
$$CAN \; (\; S¢{:}e \quad P¢{:}e \quad C¢{:}e \;)$$

where $(s{:}e, p{:}e, c{:}e)$ belongs to IS if supplier s *is* supplying part p from city c; and $(s{:}e, p{:}e, c{:}e)$ belongs to CAN if supplier s *can* supply part p from city c.

Suppose also there is a need to represent a higher level association that relates each IS pair (s, p) to the project(s) receiving parts with serial number p. Suppose one were to establish an RM/T relation $TO(S¢{:}e, P¢{:}e, J¢{:}e)$, where the attribute $J¢$ is defined on project surrogates. It is not clear from this declaration whether the pairs (s, p) in TO are pairs from IS or pairs from CAN or just any arbitrary pairs of supplier and part surrogates. A separate integrity constraint of the form

$$TO[S¢, P¢] \subseteq IS[S¢, P¢]$$

helps to resolve this ambiguity at the type level, but not at the instance level. This is because there may be two or more occurrences of the pair (s, p) in the IS relation—say $(s, p, c1)$ and $(s, p, c2)$—and it is then not clear whether an occurrence of (s, p) in the TO relation is referring to $(s, p, c1)$ or $(s, p, c2)$.

By use of associative entities in RM/T the ambiguity can be resolved both at the type and instance level. We would have RM/T relations as follows:

$$\begin{array}{lllll} IS & (\; IS¢{:}e & S¢{:}e & P¢{:}e & C¢{:}c¢ \cdots \;) \\ CAN & (\; CAN¢{:}e & S¢{:}e & P¢{:}e & C¢{:}c¢ \cdots \;) \\ TO & (\; TO¢{:}e & IS¢{:}e & \cdots \;) & \end{array}$$

where the attribute $IS¢$ in the relation TO refers to specific entities and hence specific tuples in the IS relation.

One may think of the collection of entity types participating (immediately or otherwise) in a given associative entity type as constituting an *associative molecule type*, which is bound together by tuples in the AG-relation.

9.2 Nonentity Associations

A nonentity association type has no E-relation. There is no surrogate associated with an association of this type. Hence, there is no dependable way (i.e., system-controlled way) to refer to it in either the PG-relation or the AG-relation. For the same reason, it cannot participate as a component in another association.

A nonentity association type is represented by a single n-ary relation whose attributes include the E-attributes identifying the entity types participating in the association together with the immediate properties (if any) of this association. Figure 5 shows how the assignment of employees to projects might be treated as a nonentity association type.

The insertion, update, and deletion behavior is governed by Rule 2 of the basic

Assignment

Emp ¢	Project ¢	Start_Date
α	μ	78-1-1

Fig. 5. Nonentity association

relational model. Thus, a nonentity association may not exist in the database unless the entities it interrelates are present therein.

9.3 Decomposition of Associations

Thoughts, including those that pertain to description of a database, do not arise neatly decomposed into minimal meaningful units.

Given an association involving n ($n > 2$) participating entity types, a database designer who has only binary relational tools to work with would very likely immediately decompose such an association into n *anchored* binary relations (each relating one participant to the entity domain for the association itself). Suppose that, had he cast the association in n-ary form and studied its possible nonloss decompositions, he could have found that the association is decomposable into two or more relatively independent associations of lower degree, each of which could then be separately decomposed (if desired) into binary relations. We would then say that his immediate decomposition into binaries was premature. We call this the *premature binary decomposition trap*. This trap is complementary to the connection trap in [5].

In attempting to arrive at minimal meaningful units, the designer would be well advised to make use of all the theory of n-ary relations that has been built up over the past decade. There are now such concepts as PJ/NF (otherwise known as 5NF) [11], irreducible relations, atomic decomposition [45], well-defined relations [33], independent relations [29], and primitive relations [26], all of which can be used as guidelines for decomposition. While all these concepts deal primarily with projections that are invertible by nonloss natural joins, the last two also take into account new interrelation integrity constraints that might be needed if decomposition is taken too far or poor choices are made when two or more decomposition options are available.

Note that, in general, a nonentity association cannot be split up (without information loss) into anchored binary projections in the same way associative entities can because there is no entity domain to rejoin the projections together. For this and other reasons, RM/T may be applied to database design completely avoiding the nonentity association concept altogether.

10. CARTESIAN AGGREGATION

An important dimension for forming larger meaningful units is that of *Cartesian aggregation*. Smith and Smith [33] call it simply aggregation, but we wish to distinguish it from other forms of aggregation such as statistical aggregation and cover aggregation (discussed below). According to Smith and Smith, Cartesian aggregation is an abstraction in which a relationship between objects is regarded as a higher level object.

Cartesian aggregation in RM/T is broken down into three types:

(1) aggregation of simple properties yields an entity type (characteristic or kernel or associative);
(2) aggregation of characteristic entities yields an entity type (characteristic or kernel or associative);
(3) aggregation of any combination of kernel and associative entity types yields either an associative entity type or a nonentity association type.

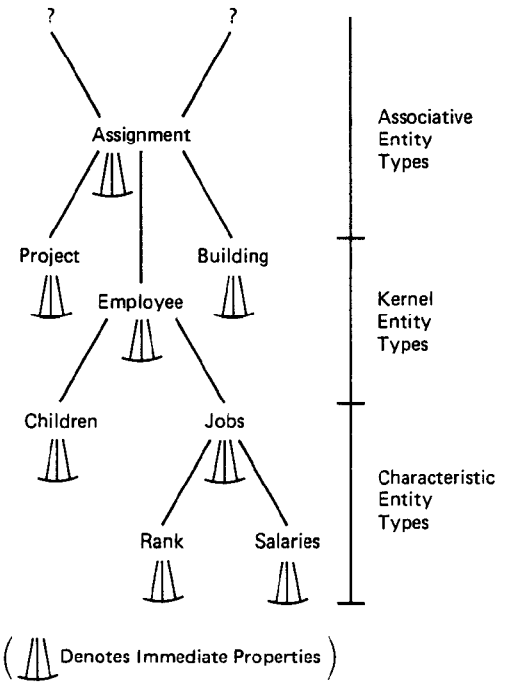

Fig. 6. Cartesian aggregation

The first kind of Cartesian aggregation is supported in RM/T by the P-relations together with the PG-relation; the second type by the characteristic relations together with the CG-relation; and the third type by the kernel relations, associative relations, and the AG-relation. Figure 6 provides an example of Cartesian aggregation.

While RM/T can be applied with the Smith and Smith constraint that abstraction by Cartesian aggregation must yield a concept namable by a simple English noun, the model itself is not constrained in this way, since this constraint is too imprecise.

11. GENERALIZATION

11.1 Unconditional Generalization

Another important dimension for forming larger meaningful units is that of generalization. It has received a good deal of attention in the context of semantic nets [18, 31, 35]. Here we are concerned with it in the context of n-ary relations. Smith and Smith [34] define *generalization* as an abstraction in which a set of similar objects is regarded as a generic object. There are two aspects to this notion: instantiation and subtype. Both are forms of *specialization*, and their inverses are forms of generalization. The extensional counterpart of instantiation is set membership, while that of subtype is set inclusion. As shown in Figure 7, to obtain particular engineers from the generic object (or type) engineer, instantiation must be applied. The types engineer, secretary, and trucker are each subtypes of the type employee. An entity type e together with its immediate subtypes,

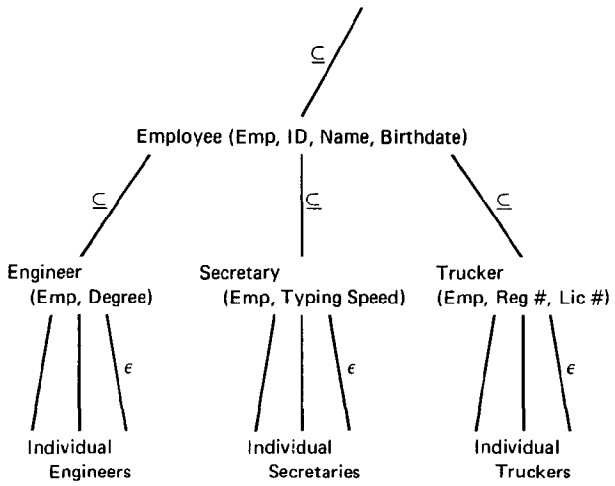

Fig. 7. Unconditional generalization

their subtypes, and so on constitute the *generalization hierarchy* of e. This hierarchy is yet another molecule type.

Why should we separate the members of a generalization hierarchy into different entity types? We do this only if different kinds of facts are to be recorded about different members of the hierarchy. If these types were not represented separately, we would have a single large relation with many occurrences of the special null value which means "value inapplicable." Associated with a generalization hierarchy is the *property inheritance rule*: Given any subtype e, all of the properties of its parent type(s) are applicable to e. For example, all of the properties of employees in general are applicable to salesmen employees in particular.

The E-relations introduced above take care of generalization by membership. To handle generalization by inclusion, we introduce the *unconditional gen inclusion relation* (UGI-relation), a ternary relation representing a labeled graph. Two attributes of UGI are defined on the RN-domain (one with the SUB role, the other with the SUP role), while the third attribute is defined on the category label domain called *PER*. The triple (SUB:m, SUP:n, PER:p) belongs to UGI if entity type $e(m)$ is an immediate subtype of entity type $e(n)$ per category p. In other words, the E-relation whose name is represented by character string m is constrained to be included (by reason of generalization per category p) in the E-relation whose name is represented by the character string n. Note that UGI contains only the immediate unconditional inclusion constraints that are associated with the semantic notion of generalization. Thus, if (SUB:m, SUP:n, PER:p) and (SUB:n, SUP:k, PER:p) belong to UGI, (SUB:m, SUP:k, PER:p) does not.

The transitive closure of the UGI-relation represents a partial order of the entity types, but not necessarily a collection of trees, since an entity type may be generalized by inclusion into two or more entity types. For example, female engineers might be generalized into engineers on the one hand and female employees on the other.

Consider the family of entity types in some generalization hierarchy. Normally,

it would be good database design to represent common properties and characteristics of these entity types as high up in that hierarchy as possible, taking full advantage of the property inheritance rule. However, RM/T itself does not place such a constaint upon generalization hierarchies—this is considered to be a design discipline that the user of RM/T may choose to adopt or reject.

The following rule governs insertions and deletions of surrogates.

Rule 7 (subtype integrity): Whenever a surrogate (say s) belongs to the E-relation for an entity of type e, s must also belong to the E-relation for each entity type of which e is a subtype.

11.2 Alternative Generalization

We may augment the usual notion of generalization hierarchy by noting that an entity type may be generalized into two or more *alternative* types. For example, in a database concerning customers (see Figure 8), suppose that a customer may be a company, partnership, or individual person and each of these is a legal unit. Suppose also that different attributes are to be recorded for each of these five entity types. Then, in addition to recording in UGI the unconditional inclusion of customers, companies, partnerships, and individuals in legal units, we should also record elsewhere the alternative or conditional inclusion of customers in companies, partnerships, and individuals. To support this, we introduce the *alternative gen inclusion relation* (AGI-relation), a ternary relation just like the UGI-relation, except for its interpretation: (SUB:m, SUP:n, PER:p) belongs to AGI if the E-relation with name m is constrained to be conditionally included in E-relation n by reason of generalization per category p.

Suppose information about a new entity is being inserted and just one of its several types is specified. Then the system can (and, according to Rule 7, must) automatically insert the surrogate generated for this entity not only in the E-relation directly representing the declared type, but also in the E-relation for every entity that, according to UGI and AGI, is superordinate to the declared entity. Both graph relations must be consulted, because A may be alternatively subordinate to B and C, which in turn are unconditionally subordinate to D; hence A is unconditionally, but not immediately, subordinate to D.

To illustrate the operational distinction between UGI and AGI, consider the introduction of a new customer into a database that conforms to Figure 8. By

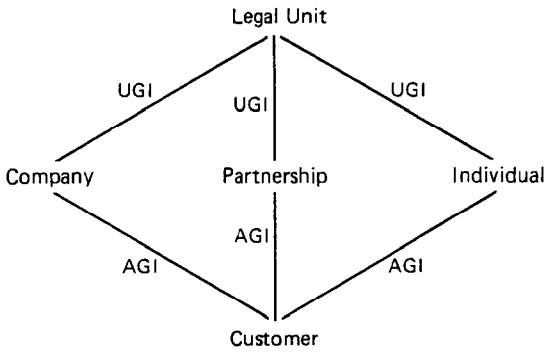

Fig. 8. Alternative generalization

consulting UGI the system ascertains that the surrogate for this customer must be entered into the E-relation for legal units as well as that for customers. By consulting AGI it ascertains that more extensional information is needed to determine whether to enter the surrogate into the E-relation for companies, partnerships, or individuals. Until this information is forthcoming, the system cannot determine whether the customer in question inherits properties from a company, partnership, or individual. Accordingly, AGI (in contrast to UGI) alerts the system to the need to obtain, if necessary, and consult extensional information for guidance.

12. COVER AGGREGATION

A convoy of ships is certainly an aggregation of some kind. However, it is not an abstraction by Cartesian aggregation, nor is it an abstraction by generalization (after all, ships are neither instantiations nor subtypes of convoys). Hammer and McLeod [15] include this kind of aggregation in their model, and we shall use their example.

Consider a database that keeps track of properties of individual ships and convoys. When information about a new ship is inserted, it is normally not known in what convoys (if any) this ship will participate. Figure 9 should make the distinctive aspects of this kind of aggregation clear. The *cover type* CONVOY means that the database is keeping track of convoys in general. CONVOY ALPHA is a particular convoy, one of several in existence at this time. SAUCY SUE designates a ship that happens to be in CONVOY ALPHA. There is a subconvoy of ALPHA to which SAUCY SUE also belongs. Note that the inclusion of SUBCONVOY in CONVOY ALPHA is not an inclusion-based generalization (SUBCONVOY is an extensionally, rather than intensionally, defined subset of ALPHA). Moreover, the membership of SAUCY SUE in CONVOY ALPHA is not a membership-based generalization (SAUCY SUE is not a particular convoy or kind of convoy).

It happens in the convoy example that a ship cannot normally be a member of two convoys at once. If we regard lone ships as singleton convoys, then the CONVOY concept partitions the class of ships. The disjointness of convoys does not carry over into all other examples of cover aggregation. Consider people and clubs in place of ships and convoys: People can belong to many different clubs simultaneously. So, in general, this type of aggregation constitutes a cover rather than a partition—hence its name.

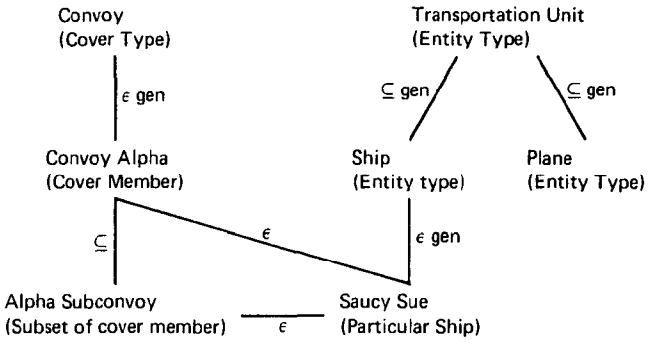

Fig. 9. Cover aggregation and generalization

A typical cover member may or may not be homogeneous in type. For example, a task force may consist of ships, planes, tanks, and personnel.

Each cover aggregation type is treated by RM/T as an entity type, having the usual E-relation plus possible P-relations and possible subordinate characteristic relations. For example, in the case of the CONVOY cover type, the E-relation would list the surrogates for existing convoys, while the P-relations and any characteristic relations would list properties of each convoy regarded as a single generic object.

Although it is possible to treat each cover member as a distinct entity type, this would normally be neither necessary nor desirable. Membership of individual entities (ships) in a cover member (particular convoy) is represented by a graph relation defined on the E-domain in the obvious way.

To enable the system to control the input of members of cover members, we introduce the *cover membership relation* (KG-relation), a graph relation on the RN-domain which specifies for every cover aggregation type what are the allowable types that may become members of cover members (e.g., are just ships allowed as members of convoys or are planes allowed too?).

13. EVENT PRECEDENCE

Entities of event type are those which have as part of their description a time of occurrence or a start time and/or a stop time. Note that not all entities with time attributes are events. For example, an associative entity which indicates that supplier x can supply item y with a delivery time of three months is not itself an event.

Ordering of events in time plays a major role in certain databases. Provision for recording this ordering at the type level represents a step toward supporting scripts (see [17]).

Event e_1 succeeds event e_2 if the time of occurrence/start of e_1 is strictly later than the time of occurrence/completion of e_2 (according to whether these events are perceived as instantaneous or not). Some types of events are unconditionally followed by one or more other event types. Such succession is normally a partial order. It is represented in RM/T by the *unconditional successor relation* (US-relation), a graph relation on the RN-domain. (SUB:m, SUP:n) belongs to this relation if an event of type $e(m)$ must be succeeded by an event of type $e(n)$, and there is no intermediate event type e such that e is an unconditional successor of $e(m)$ and $e(n)$ is an unconditional successor of e.

Similarly, some types of events are alternative successors to others, and this alternative succession is represented by the *alternative successor relation* (AS-relation) in a similar manner to the unconditional succession.

When an event e_2 succeeds an event e_1, this obviously means that e_1 is a predecessor of e_2, but it does not mean that e_1 is necessarily the *only* predecessor of e_2—even if e_2 is the only successor of e_1. Hence, we need two more graph relations to describe precedence between event types: UP for unconditional precedence and AP for alternative precedence.

To illustrate the use of these graph relations, suppose we have a database that includes records of orders placed with suppliers and records of shipments that have been accepted as input to the inventory (the corresponding event entity types will be called orders and shipments). Suppose that we prohibit acceptance

of shipments into the inventory unless there is an unfilled order covering the items in question. Then, relation UP would have a tuple (SUB:orders, SUP:shipments) that asserts that every acceptance of a shipment is unconditionally preceded by an order. In addition, relation AS would have a tuple that asserts that one possible successor event to the placing of an order is the acceptance of a shipment (shipments can, of course, be rejected). This intensional information can be used by the database system to challenge the validity of particular acceptances not covered by corresponding orders.

More generally, the relations US, AS, UP, AP provide a means of constraining insertions to and updates of the event relations supporting an event type. Otherwise, their behavior under insertion, update, and deletion is determined by whether they are kernel or associative.

14. RM/T CATALOG

RM/T contains its own extensible catalog to facilitate transformations between different organizations of common information as may be encountered in the process of view integration. The following relations constitute the catalog structure:

```
 CATR ( Rℓ    RELNAME    RELTYPE )
CATRA ( RAℓ   Rℓ    Aℓ  )
 CATA ( Aℓ    ATTNAME    USERKEY )
CATAD ( ADℓ   Aℓ    Dℓ  )
 CATD ( Dℓ    DOMNAME    VTYPE  ORDERING )
 CATC ( Cℓ    PERNAME    )
CATRC ( RCℓ   Rℓ    Cℓ  )
```

where CATR, CATA, and CATD describe the relations, attributes, and domains, respectively; CATRA interrelates relations and their attributes; CATAD interrelates attributes and their domains; CATRC interrelates relations and categories (see below for details). In addition, attributes Rℓ, Aℓ, Dℓ, Cℓ are defined on the E-domain and contain surrogates for entities of type relation, attribute, domain, and category label, respectively; attributes RAℓ, ADℓ, RCℓ are also defined on the E-domain and contain surrogates for associative entities of type relation-attribute, attribute-domain, and relation-category-label, respectively. The remaining attributes are listed below with a brief explanation:

RELNAME	relname of relation (defined on RN-domain);
ATTNAME	attname of attribute;
DOMNAME	domname of domain;
PERNAME	category label (defined on PER-domain);
RELTYPE	type of object represented by relation;
USERKEY	indicates whether attribute participates in a user-defined key for corresponding relation;
VTYPE	syntactic type of value;
ORDERING	indicates whether > is applicable between values in corresponding domain.

Given a category c, an entity type is called *top per c* if it has at least one

subordinate entity type per c, but no superordinate per c. Relation CATRC contains at least one tuple for every category. For each category in the database, it lists the relations which represent top entity types per that category. The meaning of the other relations in the catalog should be obvious.

Appropriate reltypes are specified for a relation by concatenating appropriate letters from the following list:

A associative entity type relation;
C characteristic entity type relation;
E E-relation;
G graph relation;
I inner kernel entity type relation;
K kernel entity type relation;
L edge-labeled;
N nonentity association relation;
P property relation;
T event entity type relation.

For example, a relation representing a kernel event entity type would have reltype TK; one that represents an edge-labeled digraph would have the reltype LG.

15. OPERATORS FOR RM/T

The following operators are intended to permit both the schema information and the database extension to be manipulated in a uniform way.

15.1 Name Operators

NOTE

Let R be a relation. NOTE(R) is the relname of R (i.e., the character string representation of the name of R) provided R has been assigned such a name by a user; else NOTE(R) is null. For our present purposes we do not need to extend this operator to objects other than relations. Many relations generated as intermediate results will not have relnames. Every base relation must, however, be given a relname.

TAG

Let R be a relation. Then

$$\text{TAG}(R) = R \times \{\text{NOTE}(R)\}$$

where $\times$ denotes Cartesian product.

DENOTE

Let r be the relname of a relation. Then DENOTE(r) is the relation denoted by r. When applied to relations that have relnames, the operators NOTE and DENOTE are inverses of one another.

DENOTE may also be applied to a unary relation that is a set of relnames. Let R be such a relation. Then DENOTE(R) is the set of all those relations whose relname is in R.

15.2 Set Operators

COMPRESS

Let f be an associative and commutative operator that maps a pair of relations into a relation (for example, a join). Let Z be a set of relations such that f can be validly applied to every pair of relations in Z. Then COMPRESS(f, Z) is the relation obtained by repeated pairwise application of f to the relations in Z. An alternative notation for COMPRESS(f, Z) is f/Z.

APPLY

Let f be a unary operator that maps relations into relations, and Z a set of relations (not necessarily union compatible). Then APPLY(f, Z) yields the set of all relations $f(z)$ where z is a member of Z. For convenience, we adopt the convention that if a set of relations is cited in an algebraic expression in one or more places where a relation name would be syntactically valid, then the expression is evaluated for every member of the set. However, (1) the expression must be enclosed in parentheses and preceded by the word APPLY, and (2) no more than one *set of relations* may be cited within the scope of a single APPLY (any number of individual relations may be cited).

PARTITION BY ATTRIBUTE: PATT

Let R be a relation with attribute A (possibly compound). R may have attributes other than A. Then PATT(R, A) is the set of relations obtained by partitioning R per all the distinct values of A. For all relations R having an attribute A:

$$R = \text{UNION/PATT}(R, A).$$

PARTITION BY TUPLE: PTUPLE

Let R be a relation. PTUPLE(R) is the set of relations obtained by promoting each tuple of R into a single-tuple relation. Note that R = UNION/PTUPLE(R).

PARTITION BY RELATION: PREL

Let R be a relation. PREL(R) is the set of relations whose only member is the relation R. Note that R = UNION/PREL(R).

SETREL

This operator takes as arguments any number of explicitly named relations and yields a set of relations. An appropriate expression is:

$$\text{SETREL}(R_1, R_2, \ldots, R_n).$$

15.3 Graph Operators

The following operators are included for convenient manipulation of the directed graph relations (PG, CG, AG, UGI, AGI, US, AS, UP, AP, KG). Relation R is a *digraph relation* if it is of degree at least two and has the following properties: (1) two of its attributes are defined on a common domain; (2) one of these has the SUB role, the other has the SUP role; (3) no other attributes have the SUB or SUP role. Relation R is an *edge-labeled digraph relation* if (1) it is a digraph relation of degree at least three; (2) exactly one of its attributes has the PER (labeling) role; and (3) for every m, n, p no two tuples of R have (SUB:m,

SUP:n, PER:p) in common. A digraph relation that is not edge-labeled is called *unlabeled.*

OPEN

Case 1. Let R be an *unlabeled* digraph relation (i.e., no attribute has the PER role). Then OPEN(R) yields a copy of R with all nonimmediate subordinations removed; i.e., it is the maximal subset R_1 of R having the property that if (SUB:m, SUP:n) belongs to R_1, then *either* there does not exist any k for which both (SUB:m, SUP:k) and (SUB:k, SUP:n) belong to R_1, *or else* the existence of such a k implies that $k = m$ or $k = n$.

Case 2. Let R be an *edge-labeled* digraph relation. OPEN(R) yields the maximal subset R_1 of R with the property that if (SUB:m, SUP:n, PER:p) belongs to R_1, then *either* there does not exist any k for which both (SUB:m, SUP:k, PER:p) and (SUB:k, SUP:n, PER:p) belong to R_1, *or else* the existence of such a k implies that $k = m$ or $k = n$.

CLOSE

Case 1. Let R be an *unlabeled* digraph relation. CLOSE(R) is the transitive closure of R; i.e., it is the minimal superset of R such that if both (SUB:m, SUP:k) and (SUB:k, SUP:n) belong to R, then (SUB:m, SUP:n) belongs to CLOSE(R). Tuples in CLOSE(R) that do not also belong to R have null values for those attributes other than the SUB and SUP attributes.

Case 2. Let R be an *edge-labeled* diagraph relation. CLOSE(R) yields the minimal superset of R such that if both (SUB:m, SUP:k, PER:p) and (SUB:k, SUP:n, PER:p) belong to R, then (SUB:m, SUP:n, PER:p) belongs to CLOSE(R). Tuples in CLOSE(R) that do not also belong to R have null values for those attributes other than the SUB, SUP, and PER attributes.

Note that for all digraph relations R:

$$\text{OPEN(OPEN }(R)) = \text{OPEN}(R),$$
$$\text{OPEN(CLOSE}(R)) = \text{OPEN}(R),$$
$$\text{CLOSE(CLOSE}(R)) = \text{CLOSE}(R),$$

while for all unlabeled digraph relations R of degree 2 and all edge-labeled digraph relations R of degree 3:

$$\text{CLOSE(OPEN}(R)) = \text{CLOSE}(R).$$

With higher degree digraph relations, OPEN may lose information (contained in attributes other than SUB, SUP, and PER) which CLOSE cannot regenerate.

STEP

Case 1. Let R be an *unlabeled* digraph relation that does not have an attribute SEP (which stands for separation). Let Z be the set of all attributes of R other than SUB and SUP. STEP(R) is the set of all tuples of the form

$$(\text{SUB}:x, \text{SUP}:y, Z:z, \text{SEP}:n)$$

where (SUB:x, SUP:y, Z:z) belongs to R and n is the least number of edges of the graph which separate node x from node y.

Case 2. Let R be an *edge-labeled* digraph relation that does not have an attribute SEP. Let Z be the set of all attributes of R other than SUB, SUP, and PER. STEP(R) is the set of all tuples of the form

$$(\text{SUB}{:}x, \text{SUP}{:}y, \text{PER}{:}p, Z{:}z, \text{SEP}{:}n)$$

where (SUB:x, SUP:y, PER:p, Z:z) belongs to R and n is the least number of edges with the label p separating node x from node y.

15.4 Examples

Example A. Combine all of the P-relations for the entity type employee into a single comprehensive P-relation, without losing information and without assuming any knowledge of the number of such relations.

First we obtain the names of all P-relations for the entity type employee.

$$R_1 \leftarrow \text{PG}[\text{SUP} = \text{emp}]\,[\text{SUB}].$$

Remember that PG is the property graph relation. Then we obtain the corresponding set of relations:

$$R_2 \leftarrow \text{DENOTE}(R_1).$$

Finally, we repeatedly apply the outer natural join $\textcircled{*}$ on the attribute EMP¢ (common to all relations in the set):

$$R_3 \leftarrow (\textcircled{*}\ \text{EMP¢})/R_2,$$

where $\textcircled{*}$ followed by an attribute or collection of attributes indicates that the outer natural join is to be performed with respect to these attributes as join attributes.

Suppose we combine the expressions for R_1, R_2, R_3 into a single expression and replace emp by r, where r is the relname of any entity type. Let us denote the result by:

$$\text{PROPERTY}(r) = (\textcircled{*}\ r, \text{'¢'})/\text{DENOTE}(\text{PG}[\text{SUP} = r\,]\,[\text{SUB}]).$$

PROPERTY accordingly maps a relname of an entity type into the corresponding comprehensive P-relation.

Example B. Obtain the employee name and jobtype for all employees with an excellent rating, assuming that:

(1) There are distinct entity types for each jobtype (e.g., secretary, trucker, engineer, etc.) and the jobtype category partitions the set of employees.
(2) The immediate generalization of these types is to the entity type employee.
(3) Employee name and jobtype are recorded in one or more of the P-relations associated with employee.
(4) Rating is recorded separately in a P-relation for each jobtype.

$$R_1 \leftarrow \text{UGI}[\text{SUP} = \text{emp}, \text{PER} = \text{jobtype}]\,[\text{SUB}].$$

Remember that UGI is the unconditional gen inclusion relation. R_1 is therefore a unary relation that lists all the names of all the E-relations that are unconditionally immediately subordinate to the employee relation.

$$R_2 \leftarrow \text{APPLY}(\text{PROPERTY}, R_1).$$

R_2 is a set of P-relations, each of which is the comprehensive P-relation for one of the relnames in R_1.

$$R_3 \leftarrow \text{APPLY}(R_2[\text{RATING} = \text{excellent}]).$$

R_3 is a set of relations just like R_2 except that each relation in R_3 is a restriction of its counterpart in R_2.

$$R_4 \leftarrow \text{APPLY}(R_3[\text{EMP}\not\epsilon]).$$

R_4 is a set of relations obtained by projecting each relation in R_3 on the attribute EMP$\not\epsilon$.

$$R_5 \leftarrow (\text{PROPERTY}(\text{emp}))[\text{EMP}\not\epsilon, \text{NAME}, \text{JOBTYPE}].$$

The comprehensive P-relation for the entity type employee is projected onto its surrogate, name, and jobtype attributes.

$$R_6 \leftarrow \text{UNION/APPLY}(R_4[\text{EMP}\not\epsilon = \text{EMP}\not\epsilon]R_5).$$

Each relation in the set R_4 is joined by entity employee to relation R_5. The result is compressed by repeated union to yield R_6, the required output.

The final expression is an example of a join by entity, in contrast to a join by property.

Example C. A database contains information about employees. The properties and characteristics pertinent to all employees are linked per PG and CG with the entity type employee. In addition, employees are categorized by

(1) jobtype—engineer, secretary, technician, etc.;
(2) employment status—permanent and temporary.

Distinct sets of properties and characteristics are recorded for all these different specializations. The generalization graph UGI shows the engineer, secretary, technician, etc., entity types being subordinate to the employee entity type per jobtype, and the permanent and temporary entity types subordinate to the employee entity type per status.

Obtain a ternary relation R such that (E-domain:x, RN-domain:y, PER-domain:z) belongs to R iff x is the surrogate of an employee, y is the entity type of x per category z. In effect, we are converting category information into a new attribute of a relation at the parent level.

$$R_1 \leftarrow \text{UGI}[\text{SUP} = \text{emp}]\ [\text{SUB}, \text{PER}].$$

Relation R_1 lists the names of all the relations that are immediate subordinates of employee in the generalization graph.

$$R_2 \leftarrow \text{DENOTE}(R_1[\text{SUB}]).$$

R_2 is the corresponding set of relations.

$$R_3 \leftarrow \text{APPLY}(\text{TAG}, R_2).$$

The set R_3 is obtained by taking each relation in R_2 and appending to it a column that contains as many occurrences of the relname for that relation as there are tuples in the relation.

$$R_4 \leftarrow \text{UNION/APPLY}(R_3[\text{RN}*\text{SUB}]R_1).$$

The natural join with relation R_1 is applied to each relation in R_3, using relname attributes. The resulting set of relations is compressed by repeated application of union to yield the desired relation.

Example D. Combine all of the information in the RM/T graph relations into one relation R having attributes SUB, SUP, PER, and RN, where (SUB:m, SUP:n, PER:p, RN:q) belongs to R iff

(1) q is the relname of a labeled graph relation and (SUB:m, SUP:n, PER:p) belongs to q; *or*
(2) q is the relname of an unlabeled graph relation, p is null, and (SUB:m, SUP:n) belongs to q.

Assume the reltype of graph relations is G. Make no assumption about the number of graph relations in RM/T or their names.

$$R_1 \leftarrow \text{DENOTE(CATR [RELTYPE} = G\text{] [RN])},$$
$$R_2 \leftarrow \text{APPLY(TAG, } R_2\text{)},$$
$$R \leftarrow \textcircled{\cup} / R_2 .$$

The outer union is needed in the last statement because not all graph relations in RM/T have the same degree.

16. SUMMARY OF RM/T

Systematic use of entity domains (including avoidance of nonentity associations) enables RM/T to support widely divergent viewpoints on atomic semantics, ranging from the extreme position that the minimal meaningful unit is always a binary relation to other more moderate positions. The four dimensions of molecular semantics supported by RM/T are Cartesian aggregation, generalization, cover aggregation, and event precedence (see Figure 10).

We now summarize the special objects and operators we have introduced in extending the relational model. Table I lists the objects, while Table II lists the algebraic operators. We use "att" and "rel" as abbreviations for "attribute" and "relation," respectively.

Sets of n-ary relations have been introduced as an additional type of object for algebraic manipulation. The conventional set operators applicable to these higher

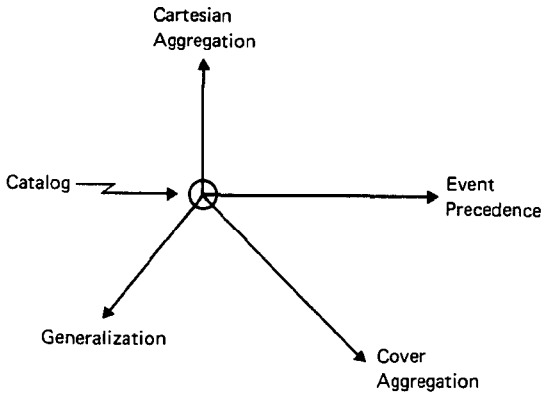

Fig. 10. Four dimensions of RM/T

Table I

RM/T object		Purpose
surrogate		system-controlled entity representative
relname		string rep of name of database relation
reltype		string rep of relation type
E-null	*	surrogate denoting "entity unknown"
E-domain	*	domain of active surrogates
PER-domain	*	domain of category labels
RN-domain	*	domain of relnames
E-att		attribute defined on E-domain
RN-att		attribute defined on RN-domain
PER-att		label in graph relation
SEP-att		separation of one node from another
SUB-att		subordinate in graph relation
SUP-att		superior in graph relation
CATR-rel	%*	list of all relnames and their reltypes
CATRA-rel	%*	relations and their attributes
CATA-rel	%*	list of all attributes
CATAD-rel	%*	attributes and their domains
CATD-rel	%*	list of all domains
CATC-rel	%*	list of all categories
CATRC-rel	%*	categories and their top entity types
E-rel		list of surrogates for a given entity type
P-rel		immediate properties of entity type
PG-rel	*	property graph
CG-rel	*	characteristic graph
AG-rel	*	association graph
UGI-rel	*	unconditional gen inclusion graph
AGI-rel	*	alternative gen inclusion graph
US-rel	*	unconditional successor graph
AS-rel	*	alternative successor graph
UP-rel	*	unconditional predecessor graph
AP-rel	*	alternative predecessor graph
KG-rel	*	membership in cover aggregate types

Note: In any RM/T database there is only one object of each type marked with an asterisk. The relations marked % have E-relation counterparts not listed explicitly here.

Table II

RM/T operator	Domain object	Range object
NOTE	relation	relname
TAG	relation	relation
DENOTE	relname	relation
	relnameset	set of relations
COMPRESS	set of relations	relation
APPLY	set of relations	set of relations
PATT	relation	set of relations
PTUPLE	relation	set of relations
PREL	relation	set of relations
SETREL	relation(s)	set of relations
OPEN	graph relation	graph relation
CLOSE	graph relation	graph relation
STEP	graph relation	graph relation

order sets are UNION, INTERSECTION, and SET DIFFERENCE. Various other operators (e.g., OUTER UNION) may be applied to them. To create these sets of relations, manipulate them, and manipulate the graph relations, the operators have been added (the terms "domain object" and "range object" refer to the domain and range of the operator) where relname set means a unary relation that is a set of relnames (see Table II).

17. CONCLUSION

We have attempted to define an extended relational model that captures more of the meaning of the data. Meaningful units of information larger than the individual n-ary relation have been introduced in such a way that apparently competing semantic approaches recorded elsewhere may all be represented therein or translated thereto. The result is a model with a richer variety of objects than the original relational model, additional insert-update-delete rules, and some additional operators that make the algebra more powerful (and unfortunately more complicated). We reiterate that incorporation of larger meaningful units is a never-ending task, and therefore this model is only slightly more semantic than the previous one.

A data model that is to act as

(1) a conceptual framework for defining a wide class of formatted databases and
(2) a mediator between stored representations and user views

should probably have at least four personalities; a tabular personality (e.g., the extensions of relations in the relational model), a set-theoretic personality (e.g., the relational algebra), an inferential string-formula personality (e.g., predicate logic in modern notation), and a graph-theoretic personality (e.g., labeled, directed hypergraphs for relations). The tabular form is needed for displaying and/or modifying extensional data (especially for those users who need to be protected from the detailed organization of the knowledge supporting the extensional data). The set-theoretic personality is needed to support search without navigation. The predicate logic personality permits stringwise expression of intensional knowledge and the application of general inferencing techniques. The graphical personality permits psychologically attractive pictures to be drawn for the special class of users who are designing the database, maintaining the supporting knowledge, or developing specialized inferencing techniques.

Note that only the tabular and set-theoretic aspects of RM/T are presented here. Clearly, there are several kinds of graphs which can be associated with RM/T. In addition to representing n-ary relations by hypergraphs, each graph relation has an immediate representation as a directed graph (in certain cases edge-labeled).

Other extensions of the relational model are under consideration: for example, additional support for the time dimension and for a nonforgetting mode of operation. It is hoped that RM/T can be developed into a general-purpose restructuring algebra for databases. It should be remembered, however, that the extensions in RM/T are primarily intended for the minority consisting of database designers and sophisticated users; most users will probably prefer the simplicity of the basic relational model.

ACKNOWLEDGMENT

The author has drawn heavily on the published ideas of Smith and Smith; LaCroix and Pirotte; Hall, Owlett, and Todd; Schmid and Swenson; Hammer and McLeod. The stimulus to write this paper came from the many provocative utterances contained in the Proceedings of the IFIP TC-2 of 1976 and 1977 [24, 25]. The author is grateful to William Armstrong, Donald Cameron, Christopher Date, Ronald Fagin, John Sowa, Stephen Todd, and the referees for helpful comments on a draft of this paper.

REFERENCES

(Note. References [1, 7, 21, 36] are not cited in the text.)

1. AHO, A. H., BEERI, C., AND ULLMAN, J. The theory of joins in relational databases. Proc. 19th IEEE Symp. on Foundations of Comptr. Sci., 1977.
2. ASTRAHAN, M. M., ET AL. System R: Relational approach to database management. *ACM Trans. Database Syst. 1*, 2 (June 1976), 97–137.
3. BEERI, C., BERNSTEIN, P., AND GOODMAN, N. A sophisticate's introduction to database normalization theory. Proc. Int. Conf. on Very Large Data Bases, Berlin, Sept. 1978, pp. 113–124.
4. CADIOU, J. M. On semantic issues in the relational model of data. Proc. 5th Symp. on Math. Foundations of Comptr. Sci., 1976, Gdansk, Poland, *Lecture Notes in Computer Science 45*, Springer-Verlag, pp. 23–38.
5. CODD, E. F. A relational model of data for large shared data banks. *Comm. ACM 13*, 6 (June 1970), 377–387.
6. CODD, E. F. Further normalization of the database relational model. In *Database Systems*, Courant Computer Science Symposia 6, R. Rustin, Ed., Prentice-Hall, Englewood Cliffs, N.J., 1971, pp. 65–98.
7. CODD, E. F. Recent investigations in relational database systems. Information Processing 74, North-Holland Pub. Co., Amsterdam, 1974, pp. 1017–1021.
8. CODD, E. F. Understanding relations (Installment No. 7). FDT (Bulletin of ACM SIGMOD) *7*, 3–4 (Dec. 1975), 23–28.
9. CODD, E. F. Extending the database relational model. Invited talk presented at the Australian Comptr. Sci. Conf., Hobart, Tasmania, Feb. 1–2, 1979.
10. FAGIN, R. Multivalued dependencies and a new normal form for relational databases. *ACM Trans. Database Syst. 2*, 3 (Sept. 1977), 262–278.
11. FAGIN, R. Normal forms and relational database operators. Proc. ACM SIGMOD Conf., Boston, Mass., May 30–June 1, 1979.
12. FALKENBERG, E. Concepts for modelling information. In *Modelling in Data Base Management Systems*, G. M. Nijssen, Ed., North-Holland Pub. Co., Amsterdam, 1976.
13. GOLDSTEIN, R. C., AND STRNAD, A. L. The MACAIMS data management system. Proc. 1970 ACM SICFIDET Workshop on Data Description and Access, Houston, Tex., Nov. 15–16, 1970.
14. HALL, P., OWLETT, J., AND TODD, S. Relations and entities. In *Modelling in Data Base Management Systems*, G. M. Nijssen, Ed., North-Holland Pub. Co., Amsterdam, 1976.
15. HAMMER, M. M., AND MCLEOD, D. J. The semantic data model: A modelling mechanism for database applications. Proc. ACM SIGMOD Conf., Austin, Tex., May 31–June 2, 1978.
16. HEATH, I. J. Private communication, April 1971.
17. HEMPHILL, L. G., AND RHYNE, J. R. A model for knowledge representation in natural language query systems. IBM Res. Rep. RJ2304, IBM Res. Lab., San Jose, Calif., Sept. 1978.
18. HENDRIX, G. G. Encoding knowledge in partitioned networks. Tech. Note 164, SRI International, Menlo Park, Calif., June 1978.
19. JORDAN, D. E. Implementing production systems with relational data bases. Proc. ACM Pacific Conf., San Francisco, Calif., April 1975.
20. LACROIX, M., AND PIROTTE, A. Generalized joins. SIGMOD Record (ACM) *8*, 3 (Sept. 1976), 14–15.
21. LACROIX, M., AND PIROTTE, A. Example queries in relational languages. Tech. Note N107, Manufacture Belge de Lampes et de Materiel Electronique, Brussels, Belgium, Jan. 1976; revised Sept. 1977.

22. LIPSKI, JR., W. On semantic issues connected with incomplete information databases. *ACM Trans. Database Syst. 4*, 3 (Sept. 1979), 262–296.
23. MERRETT, T. H. Relations as programming language elements. *Inform. Processing Lett. 6*, 1 (Feb. 1977), 29–33.
24. NIJSSEN, G. M., Ed. *Modelling in Database Management Systems.* North-Holland Pub. Co., Amsterdam, 1976.
25. NIJSSEN, G. M., Ed. *Architecture and Models in Database Management Systems.* North-Holland Pub. Co., Amsterdam, 1977.
26. PIROTTE, A. The entity-property-association model: An information-oriented database model. Rep. R343, Manufacture Belge de Lampes et de Materiel Electronique, Brussels, Belgium, March 1977.
27. PIROTTE, A. Linguistic aspects of high-level relational languages. Rep. R367, Manufacture Belge de Lampes et de Materiel Electronique, Brussels, Belgium, Jan. 1978.
28. REITER, R. On closed world data bases. In *Logic and Data Bases*, H. Gallaire and J. Minker, Eds., Plenum Press, New York, 1978.
29. RISSANEN, J. Independent components of relations. *ACM Trans. Database Syst. 2*, 4 (Dec. 1977), 317–325.
30. RISSANEN, J. Theory of relations for databases—a tutorial survey. Proc. Symp. on Math. Foundations of Comptr. Sci., 1978, Zakopane, Poland, *Lecture Notes in Computer Science*, Springer-Verlag, pp. 536–551.
31. ROUSSOPOULOS, N., AND MYLOPOULOS, J. Using semantic networks for database management. Proc. Int. Conf. on Very Large Databases, Sept. 1975.
32. SCHMID, H. A., AND SWENSON, J. R. On the semantics of the relational data model. Proc. ACM SIGMOD Conf. on Manage. of Data, San Jose, Calif., May 1975, pp. 211–223.
33. SMITH, J. M., AND SMITH, D. C. P. Database abstractions: Aggregation. *Comm. ACM 20*, 6 (June 1977), 405–413.
34. SMITH, J. M., AND SMITH, D. C. P. Database abstractions: Aggregation and generalization. *ACM Trans. Database Syst. 2*, 2 (June 1977), 105–133.
35. SOWA, J. F. Conceptual structures for a database interface. *IBM J. Res. Develop. 20*, 4 (July 1976), 336–357.
36. SOWA, J. F. Definitional mechanisms for conceptual graphs. Proc. Int. Workshop on Graph Grammars, Bad Honnef, West Germany, Nov. 1978.
37. STONEBRAKER, M., WONG, E., KREPS, P., AND HELD, G. The design and implementation of INGRES. *ACM Trans. Database Syst. 1*, 3 (Sept. 1976), 189–222.
38. TODD, S. J. P. The Peterlee relational test vehicle. *IBM Syst. J. 15*, 4 (1976), 285–308.
39. ULLMAN, J. D. *Theory of Relational Databases.* To appear.
40. VASSILIOU, Y. Null values in data base management: A denotational semantics approach. Proc. ACM SIGMOD 1979 Int. Conf. on Manage. of Data, Boston, Mass., May 30–June 1, 1979.
41. WHITNEY, V. K. M. RDMS: A relational data management system. Proc. Fourth Int. Symp. on Comptr. and Inform. Sci., Miami Beach, Fla., Dec. 14–16, 1972, Plenum Press, New York.
42. WIEDERHOLD, G. *Database Design.* McGraw-Hill, New York, 1977.
43. WONG, H. K. T., AND MYLOPOULOS, J. Two views of data semantics: A survey of data models in artificial intelligence and database management. *Informatics 15*, 3 (Oct. 1977), 344–383.
44. ZANIOLO, C. Analysis and design of relational schemata for database systems. Tech. Rep. UCLA-ENG-7669, Ph.D. Th., U. of California at Los Angeles, Los Angeles, Calif., July 1976.
45. ZANIOLO, C., AND MELKANOFF, M. A. A formal approach to the definition and design of conceptual schemas for database systems. To appear in *ACM Trans. Database Syst.*
46. ZLOOF, M. M. Query-by-example: A data base language. *IBM Syst. J. 16*, 4 (1977), 324–343.

Received March 1979; revised August 1979