Codd, E.F.
"Extending the database relational...

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

$R[B, C]$

B	C
1	2
2	1
2	5
2	3

$R[B]$

B
1
2

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.

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	
ω	ω
u	ω
u	1
ω	1

S	
ω	ω
u	ω
u	1

$R \cup S$	
ω	ω
u	ω
u	1
ω	1

$R \cap S$	
ω	ω
u	ω
u	1

$R - S$	
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		
A	B	C
u	ω	ω
v	1	ω
w	ω	1
x	1	ω
y	ω	1

$R[B, C]$	
B	C
ω	ω
1	ω
ω	1

$R[C]$
C
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 = ωC]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 < ωC]S

A	B	C
same as R[B = ωC]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 ÷ C]S

A
u

R[B ÷ C]T

A
empty

R[B ÷ ωC]S

A
w

R[B ÷ ωC]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 $SJ = S[SCITY \circleddash JCITY]J$

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

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

Example of OUTER NATURAL JOIN. Define $T(S\#, CITY, J\#) = S[SCITY \odot JCITY]J$ 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 $a, b, \ldots, z$ 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 "ε."

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 "ε" 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 "ε."

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

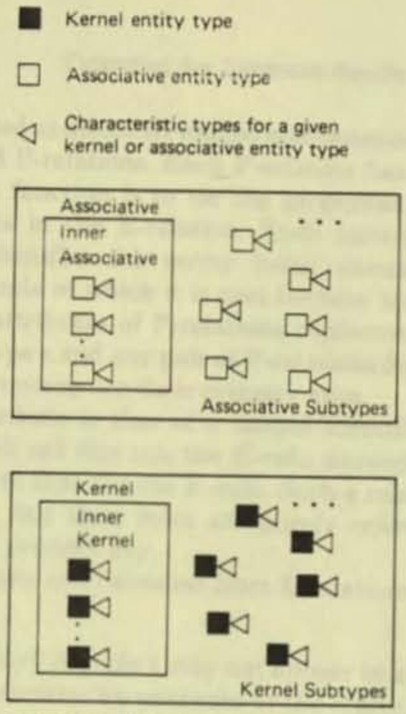


Fig. 1. Classification of entity types

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.

Employee

Emp ¢
α

(E-Relation)

Employee_Number

Emp ¢	ID
α	7777

(P-Relation)

Employee_Name

Emp ¢	First	Middle	Last
α	John	James	Buck

(P-Relation)

Employee_Address

Emp ¢	No	Street	City	State
α	765	Joy	Fun	Tas

(P-Relation)

Employee_Birthdate

Emp ¢	Yr	Mo	Da
α	1945	1	30

(P-Relation)

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.

Job

Job ¢
δ

Job_Employee

Job ¢	Emp ¢
δ	α

Job_Date_Jobname

Job ¢	Date	Jobname
δ	78-1-30	Clerk

Salary

Salary ¢
ε

Salary_Job

Salary ¢	Job ¢
ε	δ

Salary_Date_Amount

Salary ¢	Date	Amount
ε	78-1-30	400

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 *Se*

Assign

Assign ¢
λ

Assign_Emp_Project

Assign ¢	Emp ¢	Project ¢
λ	α	μ

Assign_Date

Assign ¢	Start_Date
λ	78-1-1

Fig. 4. Associative entity

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

$$\begin{array}{ll} IS & (\ S\epsilon{:}e \quad P\epsilon{:}e \quad C\epsilon{:}e\ ) \\ CAN & (\ S\epsilon{:}e \quad P\epsilon{:}e \quad C\epsilon{:}e\ ) \end{array}$$

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 $TO(S\epsilon{:}e, P\epsilon{:}e, J\epsilon{:}e)$, where the attribute $J\epsilon$ 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

$$TO[S\epsilon, P\epsilon] \subseteq IS[S\epsilon, P\epsilon]$$

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:

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

where the attribute $IS\epsilon$ 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 ɛ	Project ɛ	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.

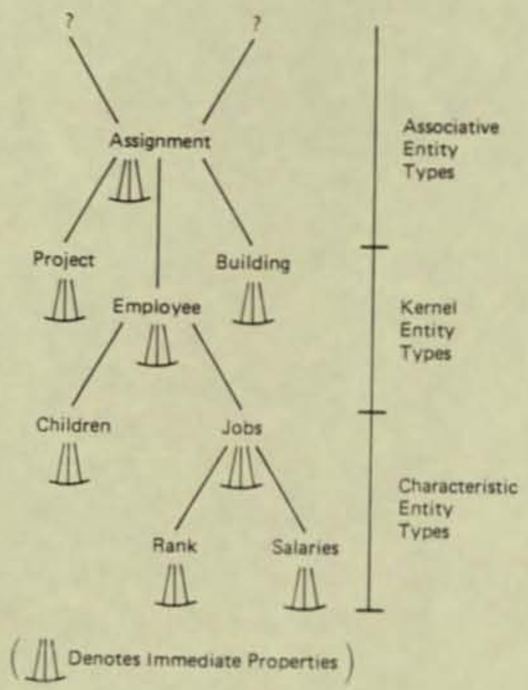


Fig. 6. Cartesian aggregation

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,

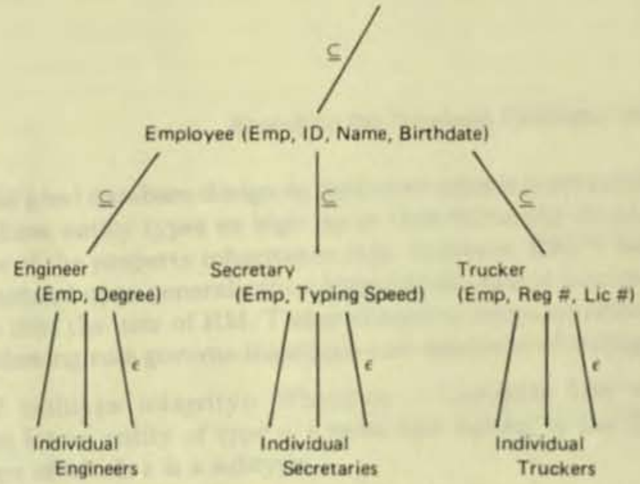


Fig. 7. Unconditional generalization

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

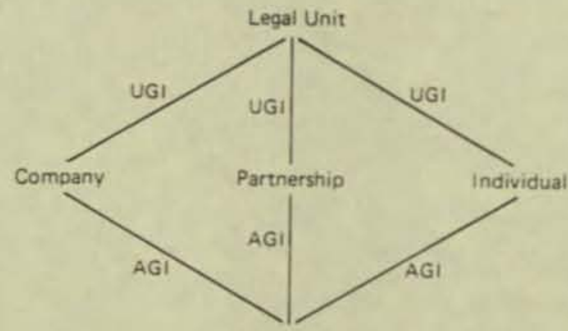


Fig. 8. Alternative generalization

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.

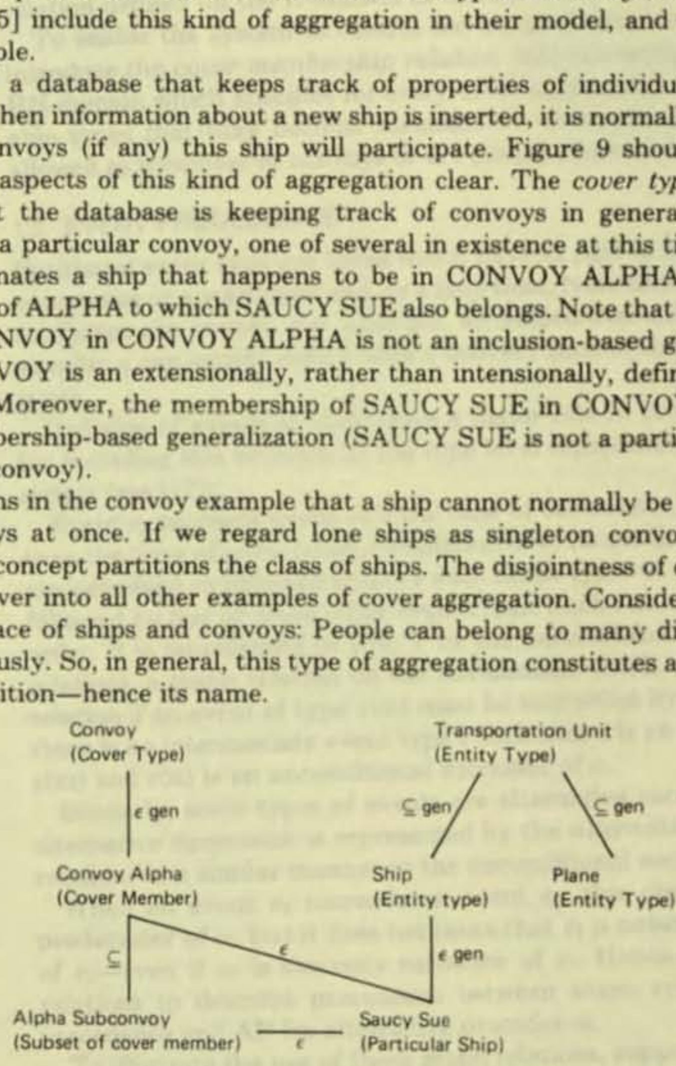


Fig. 9. Cover aggregation and generalization

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 (Rε RELNAME RELTYPE)
CATRA (RAε Rε Aε)
CATA (Aε ATTNAME USERKEY)
CATAD (ADε Aε Dε)
CATD (Dε DOMNAME VTYPE ORDERING)
CATC (Cε PERNAME)
CATRC (RCε Rε Cε)

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 Rε, Aε, Dε, Cε are defined on the E-domain and contain surrogates for entities of type relation, attribute, domain, and category label, respectively; attributes RAε, ADε, RCε 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 × 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 $\odot$ on the attribute EMPϵ (common to all relations in the set):

$$R_3 \leftarrow (\odot\ \text{EMP}\epsilon)/R_2,$$

where $\odot$ 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:

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

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.

$$R_4 \leftarrow \text{APPLY}(R_3[\text{EMP}\mathfrak{e}]).$$

R_4 is a set of relations obtained by projecting each relation in R_3 on the attribute EMP$\mathfrak{e}$.

$$R_5 \leftarrow (\text{PROPERTY(emp)})[\text{EMP}\mathfrak{e}, \text{NAME}, \text{JOBTYPE}].$$

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

$$R_6 \leftarrow \text{UNION/APPLY}(R_4[\text{EMP}\mathfrak{e} = \text{EMP}\mathfrak{e}]R_5).$$

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.

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

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.

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

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

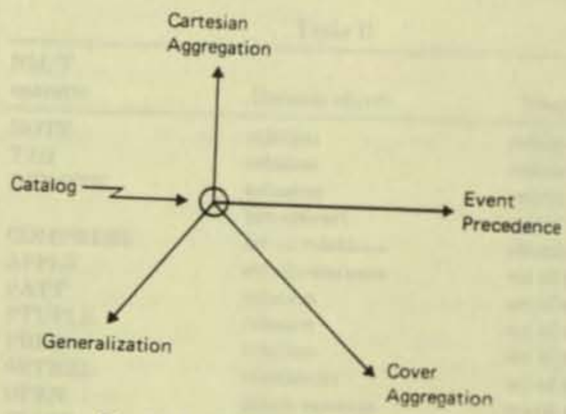


Fig. 10. Four dimensions of RM/T

elation in R_3, using relname
d by repeated application of

RM/T graph relations into
, and RN, where (SUB:m,

d (SUB:m, SUP:n, PER:p)

on, p is null, and (SUB:m,

e no assumption about the

= G] [RN]),

se not all graph relations in

e of nonentity associations)
ints on atomic semantics,
neaningful unit is always a
four dimensions of molec-
ggregation, generalization,
0).
ors we have introduced in
ts, while Table II lists the
iations for "attribute" and

dditional type of object for
applicable to these higher

Event
Precedence

ation

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	ℰ* list of all relnames and their reltypes
CATRA-rel	ℰ* relations and their attributes
CATA-rel	ℰ* list of all attributes
CATAD-rel	ℰ* attributes and their domains
CATD-rel	ℰ* list of all domains
CATC-rel	ℰ* list of all categories
CATRC-rel	ℰ* 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 ℰ 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 d
LaCroix and Pirott
McLeod. The stim
utterances contain
25]. The author is
Date, Ronald Fag
comments on a dra

REFERENCES

(Note. References [1, 7

1. AHO, A. H., BEERI
IEEE Symp. on Fo
2. ASTRAHAN, M. M.,
Database Syst. 1, 2
3. BEERI, C., BERNST
ization theory. Pro
4. CADIOU, J. M. On
Foundations of Co
Springer-Verlag, pp
5. CODD, E. F. A rela
1970), 377–387.
6. CODD, E. F. Furth
Courant Computer
1971, pp. 65–98.
7. CODD, E. F. Recen
North-Holland Pub
8. CODD, E. F. Under
4 (Dec. 1975), 23–2
9. CODD, E. F. Exten
Comptr. Sci. Conf.,
10. FAGIN, R. Multiva
Trans. Database S
11. FAGIN, R. Normal
Mass., May 30–Jun
12. FALKENBERG, E. C
Systems, G. M. Nij
13. GOLDSTEIN, R. C.,
ACM SICFIDET V
14. HALL, P., OWLETT
Management Syste
15. HAMMER, M. M.,
database applicatio
16. HEATH, I. J. Priva
17. HEMPHILL, L. G., A
query systems. IBM
18. HENDRIX, G. G. En
Menlo Park, Calif.,
19. JORDAN, D. E. Imp
Conf., San Francisc
20. LACROIX, M., AND
15.
21. LACROIX, M., AND
Manufacture Belge
Sept. 1977.

ıIFFERENCE. Various
o them. To create these
ıe graph relations, the
nd "range object" refer
me set means a unary

el that captures more of
larger than the individ-
t apparently competing
represented therein or
iety of objects than the
rules, and some addi-
nd unfortunately more
meaningful units is a
tly more semantic than

ırmatted databases and
views

ır personality (e.g., the
oretic personality (e.g.,
sonality (e.g., predicate
y (e.g., labeled, directed
for displaying and/or
o need to be protected
g the extensional data).
vithout navigation. The
f intensional knowledge
e graphical personality
for the special class of
pporting knowledge, or

f RM/T are presented
an be associated with
pergraphs, each graph
graph (in certain cases

ideration: for example,
ıonforgetting mode of
nto a general-purpose
ered, however, that the
consisting of database
y prefer the simplicity

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

Efficier
of Rela

A.V. AHO
Bell Labora
and
Y. SAGIV
Princeton U

The design of
This paper dis
select, project.
value of a que
equivalent to
among tableau
exists to optim
Key Words an
queries, NP-co
CR Categories:

1. INTRODU

The relatior
system desi
[13]. Its chi
nonprocedu
algebra.

A penalty
it becomes
efficient im
have investi
20, 22, 24, 2
transform q

EXTRA

BY EDGAR F. CODD

INSIDE IBM'S RELATIONAL 'STRATEGY'

Ironically, IBM, which had to be dragged kicking and screaming into the relational approach, is now laughing all the way to the bank. IBM should implement a relational DBMS on every distinctly programmed IBM system.

The relational model for data base management was invented within IBM, and it was very strongly in IBM's interest and in its customers' interest for the company to develop software products based on it. Nevertheless, IBM has been one of the slowest firms to develop and market the products needed to support this model.

The decision to develop a relational DBMS product took IBM management longer to make than the decision to move into the manufacturing and marketing of electronic digital computers as products — and both decisions were really forced by competitors' moves. Remington Rand, Inc. announced the Univac early in the '50s; Relational Technology, Inc., Oracle Corp. and Dun & Bradstreet Corp. announced Ingres, Oracle and Nomad, respectively, in the second half of the '70s.

IBM's top management apparently does not realize that the future in computing and data processing belongs to software, with hardware playing a follow-on role — an implementation role. The sluggishness of its action on relational data base management can be attributed principally to the following problems in IBM, particularly in software development and marketing:

- A stick-in-the-mud attitude on the part of software developers and their line management: "I want to continue doing things the way I am accustomed to doing them."
- The assumption that IBM can take its time about entering any new market because it has such a large share of the existing market.

Associated with this attitude is the "Detroit syndrome."

- Extreme parochialism.
- The treatment of IBM corporate strategies as holy writ.
- A severe lack of knowledge of levels of abstraction.
- A severe lack of knowledge of predicate logic. Employees frequently and erroneously think there is nothing more to logic than propositional logic, often incorrectly called Boolean logic in the computer field.

The first problem is basically attributable to people being lazy about learning new ways of doing things. In many lines of work we expect this attitude, but in a field with such fast-paced development and change as the computing and data processing field, it continues to surprise me.

In the relational approach there are several sharp changes in the way data bases are managed. The four that the old-timers find most difficult to accept are:

- The use of domains (think of them as application data types) as the glue that makes a data base integrated, in place of hierarchic links or pointer-based network links. Domains, in contrast to links, do not jeopardize distribution of data into distinct, geographically separated sites or its redistribution.
- The change from single-record-at-a-time to multiple — in which multiple includes the special cases zero, one, two or more records, and none of these cases is given

CONTINUED ON NEXT PAGE

■ CONTINUED FROM PAGE 37

special treatment.

■ The incorporation of a very powerful query capability into a general-purpose data base management system. Query systems used to be quite separate systems, often added on as afterthoughts, and many vendors claiming total solutions today still take this add-on approach.

■ The specification of integrity constraints linguistically in the DBMS catalog, instead of the old way: that is, structurally. This approach resulted in a serious interdependence between these constraints and the kinds of structures that the DBMS supported.

The Detroit syndrome takes its name from the car manufacturers in that city. These companies prospered for many years, and their top management became conceited and arrogant about its management skill and about the products those companies were producing. At the time of the first Arab oil crisis, they declared that their companies were unable to produce small, fuel-efficient cars at a profit. In California alone, it was not long before Japan acquired 50% of the new car market. Now, in the '80s, the Detroit manufacturers are beginning to change their old habits and manufacture fuel-efficient cars.

IBM has prospered for many years and has had good top management for most of its existence. However, from time to time it has adopted the approach of waiting to be second, or even later, to enter a market. Consider the following inventions and their usually attributed sources:

■ Stored-program electronic computers and von Neumann.

■ Virtual memory and the University of Manchester, England.

■ Transistors and Bell Laboratories.

■ The personal computer and Apple Computer, Inc.

■ The relational approach to data base management and IBM.

A noteworthy exception is the introduction of disk storage. For once, IBM was not only first in invention but also in marketing. In support of my argument, it is also worth noting that the IBM General Products Division, which manufactures disk storage products, is reputed to be the division that generates the most revenue for IBM now.

> After IMS was declared to be IBM's strategic DBMS, any internal proposal for a DBMS incompatible with IMS was treated as if it were a conspiracy. However, it is not within IBM's power to stop the process of invention.

The problem of extreme parochialism is another that has been present throughout my association with IBM, even when the company was much smaller. Individual employees somehow acquire the anomalous view that IBM is the entire computer field, and they do not need to know anything beyond whatever is required within their particular niche of IBM. Very few employees are aware of competitors' activities, the contributions of universities or the reaction of customers to various IBM products.

As one might guess, atrociously little attention is paid to technical papers in general and to non-IBM authors especially. This parochial attitude is encouraged by the fact that IBM places its manufacturing plants and development laboratories in areas so remote that even if employees wished to communicate with their peers in other companies, they are hard pressed to do so. Moreover, employee evaluation is itself parochial.

The problem of corporate strategies being treated as holy writ is a difficult one for IBM to solve. A corporate strategy is clearly needed by IBM in various key parts of the computer field in order to coordinate the activities of roughly 400,000 employees worldwide. Nevertheless, a corporate strategy developed in one year can conceivably be made obsolete a short time later as a result of an unanticipated hardware or software discovery or invention. Occasionally, communication within a company as big as IBM is so bad that an invention already made within the company makes a strategy obsolete at its time of birth.

This is what actually happened when IBM selected its Information Management System (IMS) in 1971 as its strategic DBMS product. After IMS was declared to be IBM's strategic DBMS, any internal proposal for a DBMS incompatible with IMS was treated as if it were a conspiracy to undermine IBM. However, it is not within the power of IBM or any large company or government to stop the process of invention.

To remain competitive, IBM must establish an internal communication channel directly from the lowest levels of the managerial hierarchy, where the inventions occur, to the top level. This channel can be very effective if there are explicit penalties for middle management if it attempts to block the channel.

The last two problems pertain to a severe lack of knowledge in regard to levels of abstraction and predicate logic. Employees in software vendor companies can no longer afford to be without this knowledge, and these companies, including IBM, have to solve these two problems swiftly if they wish to stay in the software products game.

DBMSs are systems that support the shared use of data — the kind of data formerly called formatted data, but now often called structured data, an equally poor term — without requiring either any scheduling in advance of execution of the actual sharing activities or any cooperative engineering of activities that are candidates to be mixed. The development of programs that depend heavily on exploiting data bases (data shared between numerous users who have conceived their actions dynamically and independently of one another, some in the development of application programs, others in terminal interaction) is rapidly overtaking the development of programs that access and manipulate private data only.

This clear trend still remains to be discovered by the following:

■ The language fraternity, including the developers and promoters of Ada and of the still-undefined class of fourth-generation languages.

■ The computer hardware and logical design fraternity, with the exception of the data base machine designers.

Early during the development of IBM's relational DBMS, DB2, the planning and design team was alerted by Chris Date and me to the fact that several important features of the relational model were missing from the specifications.

The omitted features included the declaration in the catalog of primary keys, foreign keys, domains, referential integrity and support for user-defined integrity. Further, the support for updating views was extremely ad hoc and partial — a direct result of the deplorable omission of the declaration of primary and foreign keys.

The response from the designers to our memoranda was lacking in technical foundation. They ridiculed the features in question as "religious" and "academic," which in their use of the words meant impractical and useless. An additional response was: "We could not have met our June deadline." My reply to this was, "If IBM can waste a clear six years (maybe more) before it put a serious effort into developing a relational DBMS product, what is the significance of meeting a June deadline? It is essential to do a first-rate and thorough job on this approach."

These designers had little or no experience in interacting with customers on the customers' sites. They now realize that the features they deliberately omitted are neither impractical nor useless and should have been supported in the initial release. DB2 was released after much field testing, but with "limited availability," in September 1984.

The early complaints and criticisms from users of DB2 were strongly correlated with those features of the relational model for which these designers had omitted support.

It is now 17 years since I wrote the unclassified IBM Research report RJ599 introducing many of the concepts of the relational model. During that period, no IBM employee has managed to come forward with any valid technical criticism. All of the criticisms from that source have been emotional or based on IBM's corporate strategy of the '70s. Some employees wasted a decade of their career fighting it tooth and nail, but not on technical grounds.

In sharp contrast, the System R team in the San Jose Research Laboratory, together with isolated people elsewhere, provided useful and well-conceived ideas that augmented the relational approach either abstractly or in implementation. IBM's late but substantial investment in the System R and R* prototypes has paid off already and will continue to reap substantial benefits for IBM. One principal benefit is that the product DB2, based on System R, has saved IBM from conceding the leading role in DBMS products to other software vendors. In fact, DB2 has given IBM a significant lead, in spite of its inadequacies at the time of release.

IBM has done well technically and

commercially on the hardware side — especially in the development, manufacturing and marketing of disk storage — but it has struggled painfully in data base management software, as if it were trapped and unable to accept technological change. Its blind adherence to an absurdly low-level approach (that of IMS) has trapped IBM into spending large sums and manpower on the development of tools such as the Application Development Facility and the IMS data dictionary and on incremental improvements in IMS to keep IMS alive.

Technically, these tools were appallingly weak and poorly engineered, but the main source of the problem was the design of IMS: the complexities inherent in its use, the lack of effective support for the data base administrator, the serious exposure of users to errors and the unforgiving nature of the system in contrast to a relational system.

The declaration by top management that IMS was the strategic DBMS product gave rise to a strong reluctance on

To win the struggle for the relational approach in the face of the IMS strategy-based dominance, it was necessary to create a public demand for relational DBMS products.

the part of most IBM developers to get into any approach to data base management different from IMS. After all, their careers could be damaged. A few proposals based on non-IMS approaches were attempted in the early '70s, but all were cut short, except for my activity in research — an under-the-table, one-man effort at that time — and a prototype project, the Peterlee Relational Test Vehicle, started in late 1971 in the IBM Scientific Center in England. This project was eventually squashed by IBM management in England.

In order to win the struggle for the relational approach in the face of the IMS strategy-based dominance, not technical dominance, it was necessary to win the battle within IBM division by division, to publish enough substantial technical papers and make enough public talks — primarily in North America and Western Europe, but also in other parts of the world — in order to create a public demand for relational DBMS products.

An important additional goal was to spur a few people to invest in starting small companies with the purpose of developing relational DBMS products. Several such companies were in fact launched. Relational Technology and Oracle are just two of them.

It was 1976 before the IBM General Products Division came around to planning to develop a relational DBMS product, and even then it was unable to put a team of adequate size behind the plan and wasted another three years, chiefly because of IMS distractions. Eventually, in the '80s, the SQL/Data System (DS) and DB2 products were announced. Now came the battle to turn the marketing divisions around, especially the most conservative of them, those in the U.S. and UK.

The struggle over which approach to take to data base management was three cornered. In one corner was IBM with IMS, taking a hierarchical approach, as its prime DBMS product. In a second corner was the Codasyl Data Base Task Group (DBTG) with a network-structured approach. Members of this group included most of the major users, like AT&T, General Motors Corp. and the U.S. Department of Defense, and most of the major hardware vendors, including IBM. The DBTG leaders openly declared their intention of pulling off another Codasyl coup, like that when Cobol was made a standard.

I was in the third corner, alone from 1968 to 1970, but joined in spirit by two other IBM employees in late 1970 — Sharon Weinberg, located in New York and Chris Date, located in Winchester, England. These two people are today my partners in The Relational Institute in San Jose, Calif.

It was not until the first announcement of SQL/DS by IBM early in 1982 that I felt that IBM and I were finally on one and the same side of the DBMS fence.

Ironically, IBM, which had to be dragged kicking and screaming into the relational approach, is now laughing all the way to the bank. Some of the early customers using DB2 have started with the minimum hardware that supports DB2 and within a very short time, six months in one case, as a result of positive experience with DB2, have escalated their installation to the largest configuration IBM offers. This is an important reason why IBM should implement a relational DBMS on every distinctly programmed IBM system.

One large customer, an international firm and heavy user of IMS in several countries, recently stated to me after a year's use of DB2 that the company expects to develop more than 80% of its new applications worldwide on DB2. This particular case is interesting, because a year earlier two of the firm's technical people attended a series of lectures I presented in San Francisco and told me their firm was an early user of DB2, and that they had applied DB2 solely to information center use so far.

When I asked why they had not tried it on production data, they said they had proposed just that to their manager

■ CONTINUED ON PAGE 41

■ CONTINUED FROM PAGE 39

The manager's response was, "Before we do, I'll have to ask the IBM rep." A week later, the manager informed them that the IBM representative had advised him that DB2 was not intended to handle production data, and it should therefore be used on information center data only.

The manager accepted this advice and told these two employees not to use DB2 for production data — not even to try it on such data. My response was, "It is fine for a company that sells software to take responsibility for its quality, and I believe IBM tries to do this. However, who do you think is responsible for the profitability of your firm? Is it your firm or is it IBM?" They departed and evidently thought about it, decided to try DB2 on production data, measured its performance and found it perfectly satisfactory.

In early 1982, IBM announced its first relational DBMS as a regular product, SQL/DS. As an IBM employee at that time, I had seen in advance a copy of the announcement and found it seriously disappointing. The IBM management in the White Plains, N.Y., marketing headquarters was obviously planning on introducing SQL/DS purely as a query product for information center use — one more tactic to prop up IMS and DL/I where they were weak. I wrote an article for *Computerworld*, published to appear immediately after the IBM announcement, declaring SQL/DS to be the product it is and always was: a data base management system, not just a query product.

In the fall of 1985 [CW Oct. 14 and 21, 1985], I wrote a two-part article defining 12 rules by which to judge a vendor's claim that its DBMS product is fully relational. IBM's score for SQL/DS and DB2, seven out of 12, was good relative to the competition but quite poor considering the following:

- The relational approach was an IBM invention.
- IBM was the first computer company to develop a working prototype.
- IBM has poured a significant amount of money and manpower into the research and development of relational data base management.

Not too long ago, I made the statement to IBM that "it was irresponsible of IBM to put DB2 on the market with no support for referential integrity or user-defined integrity or integrity independence." There was no shortage of excuses in reply, but I did not find any of them convincing.

In Dallas in the fall of 1984, IBM organized and hosted a meeting of about 300 software vendors for the purpose of telling them how they could interface their products with DB2 and SQL/DS. This step was an excellent one for IBM to take, since it will help all parties concerned. An increasing number of products that interface with DB2 have been announced by these vendors.

A word of caution, however, for the user: It is possible for a software package on top of a DBMS to be unable to deliver the performance of the DBMS.

■ CONTINUED ON NEXT PAGE

■ CONTINUED FROM PAGE 41

dúe to certain kinds of counter-optimizing properties of the package. An example that comes to mind — and this is by no means the only one — is Dun & Bradstreet's Nomad package; its source language is the counter-optimizing feature.

In 1980 I felt that IBM should market its relational products with at least as much vigor as that given to IMS, and I am still of that opinion. I have never taken the position, either privately or in public, that IMS should be dumped by IBM. My position has always been that IBM should let the users and customers decide. With equal treatment by IBM marketing, the relational products

I have never taken the position, either privately or in public, that IMS should be dumped by IBM. My position has always been that IBM should let the users and customers decide.

would win hands down because the customers would, sooner or later, select those products that deliver the overwhelming economic and ease-of-use advantages.

Since announcing DB2, IBM marketing headquarters has broadcast its "dual data base strategy," attempting to convince the public that it needs two different DBMSs to handle its data bases. Competitors have not been slow to capitalize on this by declaring in their advertisements: "Who needs two data base management systems?"

Only those firms that invested heavily in IMS need the two DBMSs, and then only during the period of transition — unless the firm has made the necessary sacrifices in productivity, data independence and adaptability to become an IMS Fastpath user, for which there is now no relational DBMS with a comparable performance.

No one is claiming that the transition from IMS to DB2 can completed overnight or should be. No one is claiming that it is simple and cheap to migrate application programs from one system to the other. However, the sooner the transition is planned as a step-by-step operation, the sooner it is started and the sooner it will be completed. In this way, the firm acquiring and using the new relational DBMS will make earlier gains in the following respects:

- A marked increase in the productivity of its programmers and users.
- Significantly reduced program maintenance costs.
- Sharply increased control of the data base integrity.
- Significant economies in time and money for application development.
- More round-the-clock operation.

The last item results from the significantly more dynamic character of the relational apprpach. Changes in the catalog contents (the data description) can be made without bringing the data base traffic to a halt. In fact, with sophisticated locking by the system, such changes can be made concurrently with ordinary transactions that merely change the regular data. These changes can entail dynamically altering the types and numbers of alternative access paths for any relations.

The one thing that gets IBM marketing off its hindquarters is competition. Almost all of IBM's competitors in the software field have chosen either to support the relational approach to data base management or to claim that their DBMS product is a relational DBMS. Although a few of these claims by vendors are ludicrous and result from questionable ethics, generally speaking the software vendors have been forging ahead in sales.

As a result, IBM marketing is beginning to awake from its long sleep. It is beginning, at last, to call for help from the few IBM employees in nonmarketing divisions who have real knowledge of the relational approach to data base management.

It is inherently difficult to discern what levels of abstraction are used in human problem solving and thinking activities. In the relational approach, we openly declare these levels. For example, the relational model is at a higher level of abstraction than any of the relational languages SQL, QUEL and QBE. People frequently ignore these declarations.

My experience with software developers and software salesmen generally (not just IBM employees) has been that most of them do not understand levels of abstraction. This means that they do not understand one of the principal objectives of superimposing a layer of software on top of the hardware or on top of another layer of software: namely, to introduce higher and still higher levels of abstraction.

I sometimes ask people who are skeptical concerning levels of abstraction whether they would employ a bricklayer to construct a skyscraper or whether they would ask a person to build a large suspension bridge if that person's only bridge-building experience consisted of positioning a plank of wood across a narrow stream.

Here are some of the steps I believe IBM should take as soon as possible:

1. IBM must find a way to manage software development in a leading mode, not a reactive mode. It must avoid a recurrence of the Detroit syndrome.

2. It must improve its internal communications and its evaluation techniques applied to software engineers to prevent the parochialism and bureaucracy of the past decades.

3. It must start to take software design and engineering at least as seriously as hardware by its actions, not by talk alone.

4. It must find some software experts within the corporation and place them in higher level management. It should use outside help in evaluating the knowledge of these experts.

5. All employees concerned with software products should be instructed in levels of abstraction, application of that concept to software and increased use of all applicable mathematics.

An increasing emphasis should be placed on learning these topics and on avoiding ad hoc, seat-of-the-pants inventions and special casing — activities that result in incomprehensible software with a proliferation of special-purpose additions, where each addition represents a last-minute increment intended to solve one more small problem uncovered late in the game.

> In the long run, IBM's strategy should be to incorporate numerous DB2 services and components step by step into MVS and VM.

Points 4 and 5 should at least enable managers to decide on a more rational basis which software project proposals are worth pursuing.

Today, software developers tend to come up with many claimed new approaches that are, in fact, inadequately new. They do not provide any fundamental change.

There is a clear and urgent need to get all types of general-purpose computers to communicate with one another about their data bases, especially computers of quite different sizes.

In the long run, IBM's strategy should be to incorporate numerous DB2 services and components step by step into the large-scale operating systems MVS and VM. This approach will mean a gradual redesign for both of these systems. In principle, this work can be done without adversely affecting customer investment in applications programming and training. The main advantages for IBM are the following:

- It would make MVS easier to maintain and expand.
- It could improve DB2 performance even more.
- Parts of MVS could be incorporated in hardware or firmware.
- It could become more difficult for competitors to develop hardware clones of IBM mainframes and obtain a free ride with IBM software.

IBM has publicly declared that it intends to extend its relational software to provide full support for the relational model. I believe it should execute this step swiftly to protect its customers from additional significant changes in their application programs and to discourage the American National Standards Institute from continuing to standardize on too limited a version of the language SQL, which is what the present standard represents.

I do not want readers to go away with the idea that I can find nothing right with IBM. Looking back at that part of my career as an IBM employee, I feel positive about the following aspects at least:

- In most countries in which IBM operates, it provides its employees with fair treatment and reasonable salaries and benefits, thus avoiding the problems associated with trade unions.
- As a company, IBM operates with excellent ethical standards. Occasional departures from these standards seem attributable to particular employees disobeying the rules.
- IBM has sound and fair internal rules for handling employees.
- IBM has made, and can continue to make, some solid contributions to the computing field (a good example is disk storage, cited earlier) when its inventors and innovators are not obstructed by a strategy-based fixation.
- From time to time, IBM has shown remarkable tolerance (not support) for counterstrategic activities, even though its tolerance was not outstanding in the particular case described in this article.

An example of adherence to ethical standards is that IBM refrains from misrepresenting its products to potential customers. Thus, there is no claim from IBM that it offers a "DL/I transparency" product, a claim that some other vendors are making and a claim that I believe is as technically infeasible as a product that translates correctly from assembly language to Fortran or Cobol. An additional example is IBM's claims regarding performance of DB2, which, in my opinion, are conservative.

For several reasons, IBM has a more difficult task than most companies. It develops, manufactures and markets a very wide range of products with a multitude of interdependencies among them, and most of these interdependencies are both intricate and unavoidable. The scope of application of these products is tremendous. In many cases, it takes a good degree of intelligence to understand just the application.

From the standpoint of the U.S., I believe that IBM represents a tower of strength in the world economy. It is a shining example of a company spawned by a country that emphasizes free enterprise, free trade, free speech and democracy. So, I hope those readers who object to my criticisms will use them to make improvements in the policies and methods of all hardware and software vendors, including IBM.

Codd is president of The Relational Institute in San Jose, Calif., and the developer of the relational model.

The relational DBMS debate rages on . . .

Codd blames vendors of data bases for confusion over relational theory

John Cullinane's letter to the editor [CW, Oct. 28] has publicly exposed the continuing hostility of Cullinet Software, Inc. management to the relational DBMS approach — and thereby has given many in the field of data base management systems cause to wonder how relational the product IDMS/R can be. Is the "R" merely creativity in advertising in place of creativity applied to the quality of the product? It is time for Cullinet to take a clear-cut stand that is consistent with the realities of its products and to stop trying to mislead the public.

There are several things wrong with Cullinane's response to my two-part article, "Is your DBMS really relational?" [CW, Oct. 14 and 21]. Over the past 10 years or so, Cullinet has placed all of its eggs in the Codasyl basket — in spite of the resounding defeats the Codasyl Data Base Task Group suffered in other public debates since then. After several years of significant growth and profits resulting from this Codasyl-oriented policy, Cullinet finds itself in the unenviable position of being unable to support the rather sudden and substantial change in the DBMS market, namely a decided preference on the part of buyers for DBMS that are authentically relational.

Cullinane's assertion that my article is really intended to defend the IBM dual data base strategy is not only totally incorrect, but it is also laughable because on several occasions I have publicly stated that this duality is neither in IBM's interest nor in its customers' interest. This strategy is rather like declaring assembler language and Cobol or PL/I to be equally strategic.

A second reason I have openly opposed this strategy is that, even if there were any sound reason to support it, competitiors of IBM in the DBMS field such as Cullinet take advantage of it by asserting that, in contrast to IBM, their customers need only one data base management system.

Now that the marketplace has suddenly turned in favor of the relational approach, Cullinet finds itself in a highly ambiguous and inconsistent position. It wants to convince potential customers that its product IDMS/R fully supports the approach.

However, there is a lot more to relational technology than merely supporting tables and adding the letter "R" to the name of the product. Therefore, it is not easy for a vendor to portray the impression of mastering relational technology if it has not.

Cullinane's remarks about performance are equally ludicrous and depend entirely on a conveniently incorrrect quotation from my article. Tests have shown that at least one relational DBMS product — and very likely two from distinct vendors — can outperform both IDMS and IDMS/R. Now, at this early stage in the development of relational DBMS products, it is all too easy for a vendor to produce a relational DBMS that performs poorly.

This does not mean that poor performance is a necessary consequence of choosing the relational approach — a myth that Cullinane appears anxious to perpetuate. It does mean, however, that it is necessary for a DBMS vendor's technical people to study the hundreds of technical papers that have been published on this subject and to do at least some research and prototyping themselves.

I have encountered all of the executive-style criticism before. When executives experience a favorable run of profitable years, and this has been the case with Cullinet, they convince themselves that the good fortune is entirely because of their executive skills. Thus, Cullinet executives have chosen to ignore the relational research and development of the past 15 years or more.

A letter to the editor from Lee Gruenfeld [CW, Nov. 4] displays an attitude toward theory that is unfortunately all too common in this field of software developments. Instead of examining a theory to determine if it has practical application, anything with a theoretical flavor is assumed to have no practical value.

I suspect that the land surveyors in ancient Greece were hotly opposed to the plane geometry that Euclid introduced because it threatened to convert their black art into a more systematic and rigorous enterprise. It is the assumption by practitioners that anything theoretical cannot be practical that keeps the field of software development from becoming a branch of engineering based on solid principles. It is fortunate for us today that Euclid's work survived and even now finds great use.

Let me assure you that each feature of the relational model was included in the model only if it had clear practical value. Further, as every IBM Guide users group member can confirm, the collection of requests made to IBM by early users of DB2 was strongly correlated with the features of the relational model not implemented in the first release of DB2.

One final point: I have never taken the position

that, if a DBMS is not fully relational, it cannot be considered to be relational at all. It sounds as though this purpose would be simply to avoid doing work that is necessary in today's market. The whole purpose of my article was to show the extremity of the differences in support for the relational model provided by DBMS products from different vendors — products claimed to be relational data base management systems — and to help the public sort out this confusion by providing a relatively easy way of evaluating how relational these products are.

It may help your readers to understand why I went to the trouble of developing the article if I contrast the relational approach to data base management with the fourth-generation language flag, which many vendors are now flying.

There is no fourth-generation language definition worth its salt, let alone any theoretical foundation. James Martin's purported definition fails to mention what capabilites a fourth-generation language should have — it is quite inadequate to say that it must express tasks 10 times more concisely than Cobol. A mere change in Cobol syntax could probably accomplish that goal, and incidentally, the language APL would qualify as a fourth-generation language.

Thus, any vendor can claim to provide a product that supports a fourth-generation language, and there is a no basis for checking or challenging such a claim. With the relational approach, however, the relational model can be used, is being used and will continue to be used to check vendor claims. This action will protect all the work that has gone into the relational approach from being undermined by vendors' inadequate implementations and extravagant claims.

E. F. Codd
President
The Relational Institute

When making systems investments, users choose reliability over speed

John Cullinane's letter to the editor [CW, Oct. 28] on a highly interesting two-part article by E. F. Codd, "Is your DBMS really relational?" [CW, Oct. 14 and 21] is a bit worrying to me because it seems that Cullinane is familiar neither with customer requirements nor with data base technology.

The data technology part he explained by admitting that Codd's article is "about the silliest" he has read. He only forgets that Codd's article is about relational data technology, and I hope Cullinane will realize that there is a slight difference between the relational concept and networking.

Cullinane is trying to corrupt Codd's article — and rules — by saying it is "analogous to building an airplane according to an aeronautical engineer's design specifications, and once completed, he finds it will not fly."

The analogy is excellent; there is only one part that is wrong: the conclusion. Data base technology is really "flying." There is no question about that. Cullinane is talking, or trying to talk, about speed.

I leave that point for others to decide, but I think the airplane analogy may be worth elaboration for just a moment. I do think most passengers appreciate airplanes that are safe and reliable and reach their targets. And most think it's better late than never.

So speed is not the only vital point here. What's important is to be part of a technology that is present and will be present in the future, thereby protecting the large investments users are putting in data base systems today. In that context, we should read Codd's article. I give credit to *Computerworld* for printing it.

Rolf Lind
Oslo, Norway

Theory vs. reality: the same goal from two different perspectives

It seems the recent argument in *Computerworld* over relational data base systems is once again a demonstration of the differences between a theoretical and a practical viewpoint.

First, E. F. Codd presented 12 rules and other features of a fully relational DBMS in his two-part article "Is your DBMS really relational?" [CW, Oct.14 and 21]. It is clear that there is no better authority on this subject than Codd; hence we can easily accept his article as a means of comparing a vendor's DBMS with the fully relational model.

From the other viewpoint, John Cullinane is a proven leader in the DBMS field. Judging by the growth of Cullinet Software, Inc. during the last 15 years, we can safely assume his products have been accepted by industry as valuable tools that can help solve corporate software requirements in acceptable time frames. The Cullinet DBMS is expanding its capabilities to include the relational approach but at the same time keeping the performance formula that has made it so successful.

Both viewpoints are working toward the same goal, to satisfy corporate needs, but from opposite directions. One has an excellent model and must implement a DBMS with appropriate performance considerations, the other must adapt its performance DBMS to fit the model if it wishes to be considered fully relational.

Finally, when consultants such as Joan Boroff make statements in her letter to the editor that "90% to 97% of all applications can be supported by the transaction processing capabilities of today's authentic relational DBMS" [CW, Nov. 18], we should all be aware of two major concerns. First, from what source are such statistics gathered? And second, the main goal for corporate users is for all of their applications to be supported by the same DBMS at the same time with aceptable performance measurements. No relational DBMS has proven to industry that it can do this.

David Richardson
Greenville, S.C.

Computerworld 12-16-85 v.19 n.50

"The real strenght of relational systems"

E.F. Codd

The real strengths of relational systems:

Computer World 2-6-84 p 44-45

READER'S PLATFORM

The essence of William H. Inmon's In Depth article "What price relational?" [CW, Nov. 28], Inmon's principal message, in other words, is that "poor performance . . . is inherent to the relational environment." The argument presented in support of this conclusion is that relational systems will be running a mixture of planned transactions and ad hoc queries (short-running activities and long-running activities, to use Inmon's terms), and that those two kinds of activity are mutually disruptive.

Now it is true for any system (not just a relational one) that these two kinds of activity will tend to interfere with each other somewhat, and there is no harm in drawing attention to that fact. But to suggest that relational systems will, therefore, have significantly worse performance than less flexible (hierarchical or network) systems is completely unwarranted for at least the following two reasons:

■ First, it is an apples-and-oranges comparison. It is extremely difficult to perform any kind of ad hoc activity at all on hierarchical and network data bases, with the result that those systems are almost invariably (de facto) devoted to planned activities. This fact does not mean that users would not like to be able to perform ad hoc access to those data bases if they could.

■ Second, there is no requirement to mix the two kinds of activity in a relational system. It is ridiculous to suggest that "no . . . controls are imposed" in the relational environment. Of course such controls can be imposed, if the installation requires them. Even then, if such controls do prove necessary in certain installations, the user will still enjoy all the other advantages of relational technology (ease of use, speed of application development, resilience to change and so on) and in addition will be able to perform ad hoc access at controlled times. Furthermore, there will be many installations where such controls will not be necessary, because the overall performance requirements will be less stringent. Control vs. performance is a trade-off like any other.

Inmon makes a large number of specific but unsubstantiated claims regarding the performance (actual or potential) of relational systems. He says that "a typical relational environment consists of many separate tables, none of which are physically connected," and that therefore the system has to "search for data in diverse places and [has to] construct relationships dynamically."

Such statements demonstrate a thorough confusion between the physical and logical levels of the system. Inmon seems to be unaware of — at least he makes no reference to — the many physical-level (implementation) facilities that have been provided in relational systems specifically for performance reasons. Those facilities include (but are not limited to) the following:

■ Physical data clustering (both intra- and intertable). Tables *can* be physically connected or stored physically close to one another (even interleaved, as in a hierarchic or network system), if required — but they don't have to be. Note that physically separate tables are actually a better structure (from a performance standpoint) for many applications.

■ The optimization compiler technology pioneered in System R (and subsequently incorporated into the IBM products SQL/DS and DB2) means that a significant portion of the process of "searching for data and constructing relationships" is done statically, instead of dynamically, which thereby reduces the amount of runtime I/O.

■ The provision of indexes and the ability to create and destroy indexes dynamically represent an important aspect of performance that is totally ignored in the article.

There are numerous additional errors of fact and judgment in the article. Some of the more egregious ones are as follows:

■ Inmon states that "resistance to the notion that relational systems do not perform poorly goes back to the relational movement itself . . ." He seems to be saying here that "the relational movement" (whatever that is) has always claimed that relational systems *do* perform poorly. Surely that cannot be what he meant. More to the point, he goes on to say that "the relational movement is founded in a batch mentality." In fact, exactly the opposite is the case. The original intent was to make the data base directly accessible to *end users* (implying interactive access).

■ "Given the syntax of a language like SQL, it is very difficult, if not impossible, to separate requests" — that is, into long-running vs. short-running — "until runtime." Actually this is untrue, though it is true that systems today do not attempt to make such a separation.

VIEWPOINT

Two experts review the performance issue

■ The "second operating system solution" (involving the temporary removal and subsequent reinstatement of long-running activities) is absurd and would not be worth discussing here at all, were it not for Inmon's final remark to the effect that the notion of "the standard work unit" is "violated at a most basic level by relational systems." This is arrant nonsense, as the most superficial examination of a system such as DB2 (for example) would immediately demonstrate.

■ "In Codd's original specification of the relational environment, it was specified that content-addressable memory be used, rather than conventionally addressable memory." This is completely untrue. There was never any suggestion on the part of either of the undersigned that content-addressable memory was a prerequisite to good relational performance, and the success of modern relational systems (implemented on conventional memories) has shown that indeed it is not.

■ "Much has been said about user satisfaction, nearly all of it at the syntax level." It is absurd to suggest that the many benefits of relational systems — increased productivity, ability to prototype, direct end-user access and so forth — derive exclusively from the syntax of the user language. To take one simple example, the fact that the relational user can dynamically join any number of tables *without having to be aware of the physical representation of those tables in the data base*, which is a significant ease-of-use and productivity factor, is certainly not a question of syntax.

The general tone of Inmon's article is reminiscent of the unfounded criticisms of stored-program computers that appeared when those computers were first under consideration as products. A typical claim at that time was that sorting data on stored-program computers, for example, must inevitably be slower than sorting data using punched-card equipment.

Today, nobody would make any such claim. When a new technology is introduced, there is never any shortage of people ready to decry it; the absence of adequate knowledge of the technology in question does not seem to deter people from making such attacks. If the picture presented in Inmon's article were even close to the truth, is it likely that so many software vendors would be working so hard to provide relational products, and so many users would be so loudly demanding them? ‡

Codd, the original architect of the relational model, is an IBM Fellow in research. Date is an author, lecturer and consultant specializing in relational data base systems.

PUNDITS from page 43

than the Josephson Junction-based stuff for very high-speed integrated circuits.

Phoning the Ultimate Source, who holds him on the line while making a call, our indefatigable analyst learns the answers to the last two of his questions. With the requested figures cascading from his mouth like falling dominoes, he relays the newly acquired information to the Penultimate Source, then to the second person he called, who thereupon supplies him with an answer for the first person he called, who thereupon provides him with an answer to the original question.

To the original caller he triumphantly proclaims — "four billion dollars, give or take a hundred million" — gilding the lily of his day's success by asking if there is any other way he can be of assistance. "As a matter of fact," his friend replies, "I need to know which is preferred for use in very high-speed integrated circuits: gallium arsenide or Josephson Junction-based technologies?"

Our pundit is startled: the Ultimate Source had just given him the answer. He responds smugly: "Gallium arsenide, naturally. Everyone knows that." The caller thanks him, adding that he wanted to confirm the answer he'd just given to another friend — the friend was, of course, the Ultimate Source. And so it goes. Could anyone disagree that it's a hard-knocks life being a pundit prognosticator? His plight may be as pathetic, in fact, as that of the typical user of his prognostications. It's no wonder that all of the computer industry's punditry lives for the same fantasy. They dream of boarding a jet, reaching into the seat-pocket for the in-flight magazine and finding a portfolio in its stead. On it, big as life itself, are the words: "TOP SECRET:

"Relational database: a practical foundation for productivity" E.F.Codd

The 1981 ACM Turing Award Lecture

Delivered at ACM '81, Los Angeles, California, November 9, 1981

The 1981 ACM Turing Award was presented to Edgar F. Codd, an IBM Fellow of the San Jose Research Laboratory, by President Peter Denning on November 9, 1981 at the ACM Annual Conference in Los Angeles, California. It is the Association's foremost award for technical contributions to the computing community.

Codd was selected by the ACM General Technical Achievement Award Committee for his "fundamental and continuing contributions to the theory and practice of database management systems." The originator of the relational model for databases, Codd has made further important contributions in the development of relational algebra, relational calculus, and normalization of relations.

Edgar F. Codd joined IBM in 1949 to prepare programs for the Selective Sequence Electronic Calculator. Since then, his work in computing has encompassed logical design of computers (IBM 701 and Stretch), managing a computer center in Canada, heading the development of one of the first operating systems with a general multiprogramming capability, contributing to the logic of self-reproducing automata, developing high level techniques for software specification, creating and extending the relational approach to database management, and developing an English analyzing and synthesizing subsystem for casual users of relational databases. He is also the author of *Cellular Automata*, an early volume in the ACM Monograph Series.

Codd received his B.A. and M.A. in Mathematics from Oxford University in England, and his M.Sc. and Ph.D. in Computer and Communication Sciences from the University of Michigan. He is a Member of the National Academy of Engineering (USA) and a Fellow of the British Computer Society.

The ACM Turing Award is presented each year in commemoration of A. M. Turing, the English mathematician who made major contributions to the computing sciences.

Relational Database: A Practical Foundation for Productivity

E. F. Codd
IBM San Jose Research Laboratory

It is well known that the growth in demands from end users for new applications is outstripping the capability of data processing departments to implement the corresponding application programs. There are two complementary approaches to attacking this problem (and both approaches are needed): one is to put end users into direct touch with the information stored in computers; the other is to increase the productivity of data processing professionals in the development of application programs. It is less well known that a single technology, relational database management, provides a practical foundation for both approaches. It is explained why this is so.

While developing this productivity theme, it is noted that the time has come to draw a very sharp line between relational and non-relational database systems, so that the label "relational" will not be used in misleading ways. The key to drawing this line is something called a "relational processing capability."

CR Categories and Subject Descriptors: H.2.0 **[Database Management]**: General; H.2.1 **[Database Management]**: Logical Design–*data models*; H.2.4 **[Database Management]**: Systems

General Terms: Human Factors, Languages

Additional Key Words and Phrases: database, relational database, relational model, data structure, data manipulation, data integrity, productivity

Author's Present Address: E. F. Codd, IBM Research Laboratory, 5600 Cottle Road, San Jose, CA 95193.

1. Introduction

It is generally admitted that there is a productivity crisis in the development of "running code" for commercial and industrial applications. The growth in end user demands for new applications is outstripping the capability of data processing departments to implement the corresponding application programs. In the late sixties and early seventies many people in the computing field hoped that the introduction of database management systems (commonly abbreviated DBMS) would markedly increase the productivity of application programmers by removing many of their problems in handling input and output files. DBMS (along with data dictionaries) appear to have been highly successful as instruments of data control, and they did remove many of the file handling details from the concern of application programmers. Why then have they failed as productivity boosters?

There are three principal reasons:

(1) These systems burdened application programmers with numerous concepts that were irrelevant to their data retrieval and manipulation tasks, forcing them to think and code at a needlessly low level of structural detail (the "owner-member set" of CODASYL DBTG is an outstanding example[1]);

(2) No commands were provided for processing multiple records at a time—in other words, DBMS did not support *set processing* and, as a result, programmers were forced to think and code in terms of iterative loops that were often unnecessary (here we use the word "set" in its traditional mathematical sense, not the linked structure sense of CODASYL DBTG);

(3) The needs of end users for direct interaction with databases, particularly interaction of an unanticipated nature, were inadequately recognized—a query capability was assumed to be something one could add on to a DBMS at some later time.

Looking back at the database management systems of the late sixties, we may readily observe that there was no sharp distinction between the programmer's (logical) view of the data and the (physical) representation of data in storage. Even though what was called the logical level usually provided protection from placement expressed in terms of storage addresses and byte offsets, many storage-oriented concepts were an integral part of this level. The adverse impact on development productivity of requiring programmers to navigate along access paths to reach the target data (in some cases having to deal directly with the layout of data in storage and in others having to follow pointer chains) was enormous. In addition, it was not possible to make slight changes in the layout in storage without simultaneously having to revise all programs that relied on the previous structure. The introduction of an index might have a similar effect. As a result, far too much manpower was being invested in continual (and avoidable) maintenance of application programs.

Another consequence was that installation of these systems was often agonizingly slow, due to the large amount of time spent in learning about the systems and in planning the organization of the data at both logical and physical levels, prior to database activation. The aim of this preplanning was to "get it right once and for all" so as to avoid the need for subsequent changes in the data description that, in turn, would force coding changes in application programs. Such an objective was, of course, a mirage, even if sound principles for database design had been known at the time (and, of course, they were not).

To show how relational database management systems avoid the three pitfalls cited above, we shall first review the motivation of the relational model and discuss some of its features. We shall then classify systems that are based upon that model. As we proceed, we shall stress application programmer productivity, even though the benefits for end users are just as great, because much has already been said and demonstrated regarding the value of relational database to end users (see [23] and the papers cited therein).

2. Motivation

The most important motivation for the research work that resulted in the relational model was the objective of providing a sharp and clear boundary between the logical and physical aspects of database management (including database design, data retrieval, and data manipulation). We call this the *data independence objective.*

A second objective was to make the model structurally simple, so that all kinds of users and programmers could have a common understanding of the data, and could therefore communicate with one another about the database. We call this the *communicability objective.*

A third objective was to introduce high level language concepts (but not specific syntax) to enable users to express operations upon large chunks of information at a time. This entailed providing a foundation for set-oriented processing (i.e., the ability to express in a single statement the processing of multiple sets of records at a time). We call this the *set-processing objective.*

There were other objectives, such as providing a sound theoretical foundation for database organization and management, but these objectives are less relevant to our present productivity theme.

[1] The crux of the problem with the the CODASYL DBTG owner–member set is that it combines into one construct three orthogonal concepts: one-to-many relationship, existence dependency, and a user-visible linked structure to be traversed by application programs. It is the last of these three concepts that places a heavy and unnecessary navigation burden on application programmers. It also presents an insurmountable obstacle for end users.

3. The Relational Model

To satisfy these three objectives, it was necessary to discard all those data structuring concepts (e.g., repeating groups, linked structures) that were not familiar to end users and to take a fresh look at the addressing of data.

Positional concepts have always played a significant role in computer addressing, beginning with plugboard addressing, then absolute numeric addressing, relative numeric addressing, and symbolic addressing with arithmetic properties (e.g., the symbolic address $A + 3$ in assembler language; the address $X(I + 1, J - 2)$ of an element in a Fortran, Algol, or PL/I array named X). In the relational model we replace positional addressing by totally associative addressing. Every datum in a relational database can be uniquely addressed by means of the relation name, primary key value, and attribute name. Associative addressing of this form enables users (yes, and even programmers also!) to leave it to the system to (1) determine the details of placement of a new piece of information that is being inserted into a database and (2) select appropriate access paths when retrieving data.

All information in a relational database is represented by values in tables (even table names appear as character strings in at least one table). Addressing data by value, rather than by position, boosts the productivity of programmers as well as end users (positions of items in sequences are usually subject to change and are not easy for a person to keep track of, especially if the sequences contain many items). Moreover, the fact that programmers and end users all address data in the same way goes a long way to meeting the communicability objective.

The *n*-ary relation was chosen as the single aggregate structure for the relational model, because with appropriate operators and an appropriate conceptual representation (the table) it satisfies all three of the cited objectives. Note that an *n*-ary relation is a mathematical set, in which the ordering of rows is immaterial.

Sometimes the following questions arise: Why call it the relational model? Why not call it the tabular model? There are two reasons: (1) At the time the relational model was introduced, many people in data processing felt that a relation (or relationship) among two or more objects must be represented by a linked data structure (so the name was selected to counter this misconception); (2) Tables are at a lower level of abstraction than relations, since they give the impression that positional (array-type) addressing is applicable (which is not true of *n*-ary relations), and they fail to show that the information content of a table is independent of row order. Nevertheless, even with these minor flaws, tables are the most important conceptual representation of relations, because they are universally understood.

Incidentally, if a data model is to be considered as a serious alternative for the relational model, it too should have a clearly defined conceptual representation for database instances. Such a representation facilitates thinking about the effects of whatever operations are under consideration. It is a requirement for programmer and end-user productivity. Such a representation is rarely, if ever, discussed in data models that use concepts such as entities and relationships, or in functional data models. Such models frequently do not have any operators either! Nevertheless, they may be useful for certain kinds of data type analysis encountered in the process of establishing a new database, especially in the very early stages of determining a preliminary informal organization. This leads to the question: What is a data model?

A data model is, of course, not just a data structure, as many people seem to think. It is natural that the principal data models are named after their principal structures, but that is not the whole story.

A data model [9] is a combination of at least three components:

(1) A collection of data structure types (the database building blocks);

(2) A collection of operators or rules of inference, which can be applied to any valid instances of the data types listed in (1), to retrieve, derive, or modify data from any parts of those structures in any combinations desired;

(3) A collection of general integrity rules, which implicitly or explicitly define the set of consistent database states or changes of state or both—these rules are general in the sense that they apply to any database using this model (incidentally, they may sometimes be expressed as insert–update–delete rules).

The relational model is a data model in this sense, and was the first such to be defined. We do not propose to give a detailed definition of the relational model here—the original definition appeared in [7], and an improved one in Secs. 2 and 3 of [8]. Its *structural part* consists of domains, relations of assorted degrees (with tables as their principal conceptual representation), attributes, tuples, candidate keys, and primary keys. Under the principal representation, attributes become columns of tables and tuples become rows, but there is no notion of one column succeeding another or of one row succeeding another as far as the database tables are concerned. In other words, the left to right order of columns and the top to bottom order of rows in those tables are arbitrary and irrelevant.

The *manipulative part* of the relational model consists of the algebraic operators (select, project, join, etc.) which transform relations into relations (and hence tables into tables).

The *integrity part* consists of two integrity rules: entity integrity and referential integrity (see [8, 11] for recent developments in this latter area). In any particular application of a data model it may be necessary to impose further (database-specific) integrity constraints, and thereby define a smaller set of consistent database states or changes of state.

In the development of the relational model, there has always been a strong coupling between the structural, manipulative, and integrity aspects. If the structures are defined alone and separately, their behavioral properties are not pinned down, infinitely many possibilities present themselves, and endless speculation results. It is therefore no surprise that attempts such as those of CODASYL and ANSI to develop data structure definition language (DDL) and data manipulation language (DML) in separate committees have yielded many misunderstandings and incompatibilities.

4. The Relational Processing Capability

The relational model calls not only for relational structures (which can be thought of as tables), but also for a particular kind of set processing called *relational processing*. Relational processing entails treating whole relations as operands. Its primary purpose is loop-avoidance, an absolute requirement for end users to be productive at all, and a clear productivity booster for application programmers.

The SELECT operator (also called RESTRICT) of the relational algebra takes *one* relation (table) as operand and produces a new relation (table) consisting of selected tuples (rows) of the first. The PROJECT operator also transforms *one* relation (table) into a new one, this time however consisting of selected attributes (columns) of the first. The EQUI-JOIN operator takes *two* relations (tables) as operands and produces a third consisting of rows of the first concatenated with rows of the second, but only where specified columns in the first and specified columns in the second have matching values. If redundancy in columns is removed, the operator is called NATURAL JOIN. In what follows, we use the term "join" to refer to either the equi-join or the natural join.

The relational algebra, which includes these and other operators, is intended as a yardstick of power. It is *not* intended to be a standard language, to which all relational systems should adhere. The set-processing objective of the relational model is intended to be met by means of a data sublanguage[2] having at least the power of the relational algebra *without making use of iteration or recursion statements.*

Much of the derivability power of the relational algebra is obtained from the SELECT, PROJECT, and JOIN operators alone, provided the JOIN is not subject to any implementation restrictions having to do with predefinition of supporting physical access paths. A system has an *unrestricted join capability* if it allows joins to be taken wherein *any* pair of attributes may be matched, providing only that they are defined on the same domain or data type (for our present purpose, it does not matter whether the domain is syntactic or semantic and it does not matter whether the data type is weak or strong, but see [10] for circumstances in which it does matter).

Occasionally, one finds systems in which join is supported only if the attributes to be matched have the same name or are supported by a certain type of predeclared access path. Such restrictions significantly impair the power of the system to derive relations from the base relations. These restrictions consequently reduce the system's capability to handle unanticipated queries by end users and reduce the chances for application programmers to avoid coding iterative loops.

Thus, we say that a data sublanguage L has a *relational processing capability* if the transformations specified by the SELECT, PROJECT, and unrestricted JOIN operators of the relational algebra can be specified in L without resorting to commands for iteration or recursion. For a database management system to be called *relational* it must support:

(1) Tables without user-visible navigation links between them;
(2) A data sublanguage with at least this (minimal) relational processing capability.

One consequence of this is that a DBMS that does *not* support relational processing should be considered *non-relational.* Such a system might be more appropriately called *tabular*, providing that it supports tables without user-visible navigation links between tables. This term should replace the term "semi-relational" used in [8], because there is a large difference in implementation complexity between tabular systems, in which the programmer does his own navigation, and relational systems, in which the system does the navigation for him, i.e., the system provides *automatic navigation.*

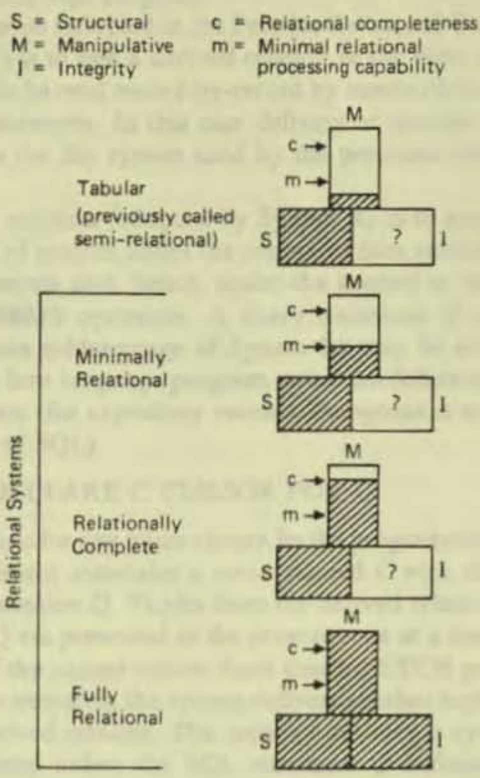
The definition of relational DBMS given above intentionally permits a lot of latitude in the services provided. For example, it is not required that the full relational algebra be supported, and there is no requirement in regard to support of the two integrity rules of the relational model (entity integrity and referential integrity). Full support by a relational system of these latter two parts of the model justifies calling that system *fully relational* [8]. Although we know of no systems that qualify as fully relational today, some are quite close to qualifying, and no doubt will soon do so.

In Fig. 1 we illustrate the distinction between the various kinds of relational and tabular systems. For each class the extent of shading in the **S** box is intended to show the degree of fidelity of members of that class to the structural requirements of the relational model. A similar remark applies to the **M** box with respect to the manipulative requirements, and to the **I** box with respect to the integrity requirements.

m denotes the minimal relational processing capability. **c** denotes relational completeness (a capability corresponding to a two-valued first order predicate logic without nulls). When the manipulation box **M** is fully shaded, this denotes a capability corresponding to the

[2] A data sublanguage is a specialized language for database management, supporting at least data definition, data retrieval, insertion, update, and deletion. It need not be computationally complete, and usually is not. In the context of application programming, it is intended to be used in conjunction with one or more programming languages.

Fig. 1. Classification of DBMS.



full relational algebra defined in [8] (a three-valued predicate logic with a single kind of null). The question mark in the integrity box for each class except the fully relational is an indication of the present inadequate support for integrity in relational systems. Stronger support for domains and primary keys is needed [10], as well as the kind of facility discussed in [14].

Note that a relational DBMS may package its relational processing capability in any convenient way. For example, in the INGRES system of Relational Technology, Inc., the RETRIEVE statement of QUEL [29] embodies all three operators (select, project, join) in one statement, in such a way that one can obtain the same effect as any one of the operators or any combination of them.

In the definition of the relational model there are several prohibitions. To cite two examples: user-visible navigation links between tables are ruled out, and database information must not be represented (or hidden) in the ordering of tuples within base relations. Our experience is that DBMS designers who have implemented non-relational systems do not readily understand and accept these prohibitions. By contrast, users enthusiastically understand and accept the enhanced ease of learning and ease of use resulting from these prohibitions.

Incidentally, the Relational Task Group of the American National Standards Institute has recently issued a report [4] on the feasibility of developing a standard for relational database systems. This report contains an enlightening analysis of the features of a dozen relational systems, and its authors clearly understand the relational model.

5. The Uniform Relational Property

In order to have wide applicability most relational DBMS have a data sublanguage which can be interfaced with one or more of the commonly used programming languages (e.g., Cobol, Fortran, PL/I, APL). We shall refer to these latter languages as *host languages*. A relational DBMS usually supports at least one end-user oriented data sublanguage—sometimes several, because the needs of these users may vary. Some prefer string languages such as QUEL or SQL [5], while others prefer the screen-oriented two-dimensional data sublanguage of Query-by-Example [33].

Now, some relational systems (e.g., System R [6], INGRES [29]) support a data sublanguage that is usable in two modes: (1) interactively at a terminal and (2) embedded in an application program written in a host language. There are strong arguments for such a *double-mode* data sublanguage:

(1) With such a language application programmers can separately debug at a terminal the database statements they wish to incorporate in their application programs—people who have used SQL to develop application programs claim that the double-mode feature significantly enhances their productivity;

(2) Such a language significantly enhances communication among programmers, analysts, end users, database administration staff, etc.;

(3) Frivolous distinctions between the languages used in these two modes place an unnecessary learning and memory burden on those users who have to work in both modes.

The importance of this feature in productivity suggests that relational DBMS be classified according to whether they possess this feature or not. Accordingly, we call those relational DBMS that support a double-mode sublanguage *uniform relational*. Thus, a uniform relational DBMS supports relational processing at both an end-user interface and at an application programming interface *using a data sublanguage common to both interfaces*.

The natural term for all other relational DBMS is *non-uniform relational*. An example of a non-uniform relational DBMS is the TANDEM ENCOMPASS [19]. With this system, when retrieving data interactively at a terminal, one uses the relational data sublanguage ENFORM (a language with relational processing capability). When writing a program to retrieve or manipulate data, one uses an extended version of Cobol (a language that does not possess the relational processing capability). Common to both levels of use are the structures: tables without user-visible navigation links between them.

A question that immediately arises is this: how can a data sublanguage with relational processing capability be interfaced with a language such as Cobol or PL/I that can handle data one record at a time only (i.e., that is incapable of treating a set of records as a single operand)? To solve this problem we must separate the following two actions from one another: (1) definition of the relation to be derived; (2) presentation of the derived relation to the host language program.

One solution (adopted in the Peterlee Relational Test Vehicle [31]) is to cast a derived relation in the form of a file that can be read record-by-record by means of host language statements. In this case delivery of records is delegated to the file system used by the pertinent host language.

Another solution (adopted by System R) is to keep the delivery of records under the control of data sublanguage statements and, hence, under the control of the relational DBMS optimizer. A query statement Q of SQL (the data sublanguage of System R) may be embedded in a host language program, using the following kind of phrase (for expository reasons, the syntax is not exactly that of SQL)

DECLARE C CURSOR FOR Q

where C stands for any name chosen by the programmer. Such a statement associates a *cursor* named C with the defining expression Q. Tuples from the derived relation defined by Q are presented to the program one at a time by means of the named cursor. Each time a FETCH per this cursor is executed, the system delivers another tuple from the derived relation. The order of delivery is system-determined unless the SQL statement Q defining the derived relation contains an ORDER BY clause.

It is important to note that in advancing a cursor over a derived relation the programmer is *not* engaging in navigation to some target data. The derived relation is itself the target data! It is the DBMS that determines whether the derived relation should be materialized *en bloc* prior to the cursor-controlled scan or materialized piecemeal during the scan. In either case, it is the system (not the programmer) that selects the access paths by which the derived data is to be generated. This takes a significant burden off the programmer's shoulders, thereby increasing his productivity.

6. Skepticism About Relational Systems

There has been no shortage of skepticism concerning the practicality of the relational approach to database management. Much of this skepticism stems from a lack of understanding, some from a fear of the numerous theoretical investigations that are based on the relational model [1, 2, 15, 16, 24]. Instead of welcoming a theoretical foundation as providing soundness, the attitude seems to be: if it's theoretical, it cannot be practical. The absence of a theoretical foundation for almost all non-relational DBMS is the prime cause of their *ungepotchket* quality. (This is a Yiddish word, one of whose meanings is patched up.)

On the other hand, it seems reasonable to pose the following two questions:

(1) Can a relational system provide the range of services that we have grown to expect from other DBMS?

(2) If (1) is answered affirmatively, can such a system perform as well as non-relational DBMS?[3]

We look at each of these in turn.

6.1 Range of Services

A full-scale DBMS provides the following capabilities:

- data storage, retrieval, and update;
- a user-accessible catalog for data description;
- transaction support to ensure that all or none of a sequence of database changes are reflected in the pertinent databases (see [17] for an up-to-date summary of transaction technology);
- recovery services in case of failure (system, media, or program);
- concurrency control services to ensure that concurrent transactions behave the same way as if run in some sequential order;
- authorization services to ensure that all access to and manipulation of data be in accordance with specified constraints on users and programs [18];
- integration with support for data communication;
- integrity services to ensure that database states and changes of state conform to specified rules.

Certain relational prototypes developed in the early seventies fell far short of providing all these services (possibly for good reasons). Now, however, several relational systems are available as software products and provide all these services with the exception of the last. Present versions of these products are admittedly weak in the provision of integrity services, but this is rapidly being remedied [10].

Some relational DBMS actually provide more complete data services than the non-relational systems. Three examples follow.

As a first example, relational DBMS support the extraction of all meaningful relations from a database, whereas non-relational systems support extraction only where there exist statically predefined access paths.

As a second example of the additional services provided by some relational systems, consider views. A *view* is a virtual relation (table) defined by means of an expression or sequence of commands. Although not directly supported by actual data, a view appears to a user as if it were an additional base table kept up-to-date and in a state of integrity with the other base tables. Views are useful for permitting application programs and users at terminals to interact with constant view structures, even when the base tables themselves are undergoing structural changes at the *logical* level (providing that the pertinent views are still definable from the new base tables). They are also useful in restricting the scope of

[3] One should bear in mind that the non-relational ones always employ comparatively low level data sublanguages for application programming.

access of programs and users. Non-relational systems either do not support views at all or else support much more primitive counterparts, such as the CODASYL subschema.

As a third example, some systems (e.g., SQL/DS [28] and its prototype predecessor System R) permit a variety of changes to be made to the logical and physical organization of the data dynamically—while transactions are in progress. These changes rarely require application programs to be recoded. Thus, there is less of a program maintenance burden, leaving programmers to be more productive doing development rather than maintenance. This capability is made possible in SQL/DS by the fact that the system has complete control over access path selection.

In non-relational systems such changes would normally require all other database activities including transactions in progress to be brought to a halt. The database then remains out of action until the organizational changes are completed and any necessary recompiling done.

6.2 Performance

Naturally, people would hesitate to use relational systems if these systems were sluggish in performance. All too often, erroneous conclusions are drawn about the performance of relational systems by comparing the time it might take for one of these systems to execute a complex transaction with the time a non-relational system might take to execute an extremely simple transaction. To arrive at a fair performance comparison, one must compare these systems on the same tasks or applications. We shall present arguments to show why relational systems should be able to compete successfully with non-relational systems.

Good performance is determined by two factors: (1) the system must support performance-oriented physical data structures; (2) high-level language requests for data must be compiled into lower-level code sequences at least as good as the average application programmer can produce by hand.

The first step in the argument is that a program written in a Cobol-level language can be made to perform efficiently on large databases containing production data structured in tabular form with no user-visible navigation links between them. This step in the argument is supported by the following information [19]: as of August 1981, Tandem Computer Corp. had manufactured and installed 760 systems; of these, over 700 were making use of the Tandem ENCOMPASS relational database management system to support databases containing production data. Tandem has committed its own manufacturing database to the care of ENCOMPASS. ENCOMPASS does not support links between the database tables, either user-visible (navigation) links or user-invisible (access method) links.

In the second step of the argument, suppose we take the application programs in the above-cited installations and replace the database retrieval and manipulation statements by statements in a database sublanguage with a relational processing capability (e.g., SQL). Clearly, to obtain good performance with such a high level language, it is essential that it be compiled into object code (instead of being interpreted), and it is essential that that object code be efficient.

Compilation is used in System R and its product version SQL/DS. In 1976 Raymond Lorie developed an ingenious pre- and post-compiling scheme for coping with dynamic changes in access paths [21]. It also copes with early (and hence efficient) authorization and integrity checking (the latter, however, is not yet implemented). This scheme calls for compiling in a rather special way the SQL statements embedded in a host language program. This compilation step transforms the SQL statements into appropriate CALLs within the source program together with access modules containing object code. These modules are then stored in the database for later use at runtime. The code in these access modules is generated by the system so as to optimize the sequencing of the major operations and the selection of access paths to provide runtime efficiency. After this precompilation step, the application program is compiled by a regular compiler for the pertinent host language. If at any subsequent time one or more of the access paths is removed and an attempt is made to run the program, enough source information has been retained in the access module to enable the system to re-compile a new access module that exploits the now existing access paths *without requiring a re-compilation of the application program.*

Incidentally, the same data sublanguage compiler is used on ad hoc queries submitted interactively from a terminal and also on queries that are dynamically generated during the execution of a program (e.g., from parameters submitted interactively). Immediately after compilation, such queries are executed and, with the exception of the simplest of queries, the performance is better than that of an interpreter.

The generation of access modules (whether at the initial compiling or re-compiling stage) entails a quite sophisticated optimization scheme [27], which makes use of system-maintained statistics that would not normally be within the programmer's knowledge. Thus, only on the simplest of all transactions would it be possible for an average application programmer to compete with this optimizer in generation of efficient code. Any attempts to compete are bound to reduce the programmer's productivity. Thus, the price paid for extra compile-time overhead would seem to be well worth paying.

Assuming non-linked tabular structures in both cases, we can expect SQL/DS to generate code comparable with average hand-written code in many simple cases, and superior in many complex cases. Many commercial transactions are extremely simple. For example, one may need to look up a record for a particular railroad wagon to find out where it is or find the balance in someone's savings account. If suitably fast access paths are supported (e.g., hashing), there is no reason why a high-level language such as SQL, QUEL, or QBE should result in less efficient runtime code for these simple transactions than a lower level language, even though such transactions make little use of the optimizing capability of the high-level data sublanguage compiler.

7. Future Directions

If we are to use relational database as a foundation for productivity, we need to know what sort of developments may lie ahead for relational systems.

Let us deal with near-term developments first. In some relational systems stronger support is needed for domains and primary keys per suggestions in [10]. As already noted, all relational systems need upgrading with regard to automatic adherence to integrity constraints. Existing constraints on updating join-type views need to be relaxed (where theoretically possible), and progress is being made on this problem [20]. Support for outer joins is needed.

Marked improvements are being made in optimizing technology, so we may reasonably expect further improvements in performance. In certain products, such as the ICL CAFS [22] and the Britton-Lee IDM500 [13], special hardware support has been implemented. Special hardware may help performance in certain types of applications. However, in the majority of applications dealing with formatted databases, software-implemented relational systems can compete in performance with software-implemented non-relational systems.

At present, most relational systems do not provide any special support for engineering and scientific databases. Such support, including interfacing with Fortran, is clearly needed and can be expected.

Catalogs in relational systems already consist of additional relations that can be interrogated just like the rest of the database using the same query language. A natural development that can and should be swiftly put in place is the expansion of these catalogs into full-fledged active dictionaries to provide additional on-line data control.

Finally, in the near term, we may expect database design aids suited for use with relational systems both at the logical and physical levels.

In the longer term we may expect support for relational databases distributed over a communications network [25, 30, 32] and managed in such a way that application programs and interactive users can manipulate the data (1) as if all of it were stored at the local node—*location transparency*—and (2) as if no data were replicated anywhere—*replication transparency*. All three of the projects cited above are based on the relational model. One important reason for this is that relational databases offer great decomposition flexibility when planning how a database is to be distributed over a network of computer systems, and great recomposition power for dynamic combination of decentralized information. By contrast, CODASYL DBTG databases are very difficult to decompose and recompose due to the entanglement of the owner-member navigation links. This property makes the CODASYL approach extremely difficult to adapt to a distributed database environment and may well prove to be its downfall. A second reason for use of the relational model is that it offers concise high level data sublanguages for transmitting requests for data from node to node.

The ongoing work in extending the relational model to capture in a formal way more meaning of the data can be expected to lead to the incorporation of this meaning in the database catalog in order to factor it out of application programs and make these programs even more concise and simple. Here, we are, of course, talking about meaning that is represented in such a way that the system can understand it and act upon it.

Improved theories are being developed for handling missing data and inapplicable data (see for example [3]). This work should yield improved treatment of null values.

As it stands today, relational database is best suited to data with a rather regular or homogeneous structure. Can we retain the advantages of the relational approach while handling heterogeneous data also? Such data may include images, text, and miscellaneous facts. An affirmative answer is expected, and some research is in progress on this subject, but more is needed.

Considerable research is needed to achieve a rapprochement between database languages and programming languages. Pascal/R [26] is a good example of work in this direction. Ongoing investigations focus on the incorporation of abstract data types into database languages on the one hand [12] and relational processing into programming languages on the other.

8. Conclusions

We have presented a series of arguments to support the claim that relational database technology offers dramatic improvements in productivity both for end users and for application programmers. The arguments center on the data independence, structural simplicity, and relational processing defined in the relational model and implemented in relational database management systems. All three of these features simplify the task of developing application programs and the formulation of queries and updates to be submitted from a terminal. In addition, the first feature tends to keep programs viable in the face of organizational and descriptive changes in the database and therefore reduces the effort that is normally diverted into the maintenance of programs.

Why, then, does the title of this paper suggest that relational database provides only a foundation for improved productivity and not the total solution? The reason is simple: relational database deals only with the shared data component of application programs and end-user interactions. There are numerous complementary technologies that may help with other components or aspects, for example, programming languages that support relational processing and improved checking of data types, improved editors that understand more of the language being used, etc. We use the term "foundation," because interaction with shared data (whether by program or via terminal) represents the core of so much data processing activity.

The practicality of the relational approach has been proven by the test and production installations that are already in operation. Accordingly, with relational systems we can now look forward to the productivity boost that we all hoped DBMS would provide in the first place.

Acknowledgments. I would like to express my indebtedness to the System R development team at IBM Research, San Jose for developing a full-scale, uniform relational prototype that entailed numerous language and system innovations; to the development team at the IBM Laboratory, Endicott, N.Y. for the professional way in which they converted System R into product form; to the various teams at universities, hardware manufacturers, software firms, and user intallations, who designed and implemented working relational systems; to the QBE team at IBM Yorktown Heights, N.Y.; to the PRTV team at the IBM Scientific Centre in England; and to the numerous contributors to database theory who have used the relational model as a cornerstone. A special acknowledgement is due to the very few colleagues who saw something worth supporting in the early stages, particularly, Chris Date and Sharon Weinberg. Finally, it was Sharon Weinberg who suggested the theme of this paper.

Received 10/81; revised and accepted 12/81

References

1. Beeri, C., Bernstein, P., Goodman, N. A sophisticate's introduction to database normalization theory. *Proc. Very Large Data Bases*, West Berlin, Germany, Sept. 1978.
2. Bernstein, P.A., Goodman, N., Lai, M-Y. Laying phantoms to rest. Report TR-03-81, Center for Research in Computing Technology, Harvard University, Cambridge, Mass., 1981.
3. Biskup, J.A. A formal approach to null values in database relations. *Proc. Workshop on Formal Bases for Data Bases*, Toulouse, France, Dec 1979; published in [16] (see below) pp 299–342.
4. Brodie, M. and Schmidt, J. (Eds), Report of the ANSI Relational Task Group., (to be published ACM SIGMOD Record).
5. Chamberlin, D.D., et al. SEQUEL2: A unified approach to data definition, manipulation, and control. *IBM J. Res. & Dev., 20*, 6, (Nov. 1976) 560–565.
6. Chamberlin, D.D., et al. A history and evaluation of system R. *Comm. ACM, 24*, 10, (Oct. 1981) 632–646.
7. Codd, E.F. A relational model of data for large shared data banks. *Comm. ACM, 13*, 6, (June 1970) 377–387.
8. Codd, E.F. Extending the database relational model to capture more meaning. *ACM TODS, 4*, 4, (Dec. 1979) 397–434.
9. Codd, E.F. Data models in database management. *ACM SIGMOD Record, 11*, 2, (Feb. 1981) 112–114.
10. Codd, E.F. The capabilities of relational database management systems. *Proc. Convencio Informatica Llatina*, Barcelona, Spain, June 9–12, 1981, pp 13–26; also available as Report 3132, IBM Research Lab., San Jose, Calif.
11. Date, C.J. Referential integrity. *Proc. Very Large Data Bases*, Cannes, France, September 9–11, 1981, pp 2–12.
12. Ehrig, H., and Weber, H. Algebraic specification schemes for data base systems. *Proc. Very Large Data Bases*, West Berlin, Germany, Sept 13–15, 1978, 427–440.
13. Epstein, R., and Hawthorne, P. Design decisions for the intelligent database machine. *Proc. NCC 1980, AFIPS, Vol. 49., May 1980*, pp 237–241.
14. Eswaran, K.P., and Chamberlin, D.D. Functional specifications of a subsystem for database integrity. *Proc. Very Large Data Bases*, Framingham, Mass., Sept. 1975, pp 48–68.
15. Fagin, R. Horn clauses and database dependencies. *Proc. 1980 ACM SIGACT Symp. on Theory of Computing*, Los Angeles, CA, pp 123–134.
16. Gallaire, H., Minker, J., and Nicolas, J.M. *Advances in Data Base Theory*. Vol 1, Plenum Press, New York, 1981.
17. Gray, J. The transaction concept: virtues and limitations. *Proc. Very Large Data Bases*, Cannes, France, September 9–11, 1981, pp 144–154.
18. Griffiths, P.G., and Wade, B.W. An authorization mechanism for a relational database system. *ACM TODS, 1*, 3, (Sept 1976) 242–255.
19. Held, G. ENCOMPASS: A relational data manager. Data Base/81, Western Institute of Computer Science, Univ. of Santa Clara, Santa Clara, Calif., August 24–28, 1981.
20. Keller, A.M. Updates to relational databases through views involving joins. Report RJ3282, IBM Research Laboratory, San Jose, Calif., October 27, 1981.
21. Lorie, R.A., and Nilsson, J.F. An access specification language for a relational data base system. *IBM J. Res. & Dev., 23*, 3, (May 1979) 286–298.
22. Maller, V.A.J. The content addressable file store—CAFS. *ICL Technical J., 1*, 3, (Nov. 1979) 265–279.
23. Reisner, P. Human factors studies of database query languages: A survey and assessment. *ACM Computing Surveys, 13*, 1, (March 1981) 13–31.
24. Rissanen, J. Theory of relations for databases—A tutorial survey. *Proc. Symp. on Mathematical Foundations of Computer Science*, Zakopane, Poland, September 1978, Lecture Notes in Computer Science, No. 64, Springer Verlag, New York, 1978.
25. Rothnie, J.B., Jr. et al. Introduction to a system for distributed databases (SDD-1). *ACM TODS, 5*, 1, (March 1980) 1–17.
26. Schmidt, J.W. Some high level language constructs for data of type relation. *ACM TODS, 2*, 3, (Sept 1977) 247–261.
27. Selinger, P.G., et al. Access path selection in a relational database system. *Proc. 1979 ACM SIGMOD International Conference on Management of Data*, Boston, MA, May 1979, pp 23–34.
28. ———, SQL/Data system for VSE: A relational data system for application development. IBM Corp. Data Processing Division, White Plains, N.Y., G320-6590, Feb 1981.
29. Stonebraker, M.R., et al. The design and implementation of INGRES, *ACM TODS, 1*, 3, (Sept. 1976) 189–222.
30. Stonebraker, M.R., and Neuhold, E.J. A distributed data base version of INGRES. *Proc. Second Berkeley Workshop on Distributed Data Management and Computer Networks*, Lawrence-Berkeley Lab., Berkeley, Calif., May 1977, pp 19–36.
31. Todd., S.J.P. The Peterlee relational test vehicle—A system overview. *IBM Systems J., 15*, 4, 1976, 285–308.
32. Williams, R. et al. R*: An overview of the architecture. Report RJ3325, IBM Research Laboratory, San Jose, Calif., October 27, 1981.
33. Zloof, M.M. Query by example. *Proc. NCC, AFIPS Vol 44*, May 1975, pp 431–438.

"A relational model of data for large shared data banks" E.F. Codd

A Relational Model of Data for Large Shared Data Banks

E.F. Codd

June, 1970
Volume 13, Number 6
pp. 377–387

In 1970, Codd proposed a new model for database systems called the relational model. Through its simplicity and mathematical basis, the relational model has provided an intuitively more appealing foundation for database systems than its two major competitors: the hierarchical and network models. The model has had an enormous impact on both the theory and development of database systems. A growing number of commercial database systems are relational. In 1981, the ACM Turing Award was presented to Codd.

–D.E.D.

A Relational Model of Data for Large Shared Data Banks

E. F. Codd
IBM Research Laboratory, San Jose, California

Future users of large data banks must be protected from having to know how the data is organized in the machine (the internal representation). A prompting service which supplies such information is not a satisfactory solution. Activities of users at terminals and most application programs should remain unaffected when the internal representation of data is changed and even when some aspects of the external representation are changed. Changes in data representation will often be needed as a result of changes in query, update, and report traffic and natural growth in the types of stored information.

Existing noninferential, formatted data systems provide users with tree-structured files or slightly more general network models of the data. In Section 1, inadequacies of these models are discussed. A model based on *n*-ary relations, a normal form for data base relations, and the concept of a universal data sublanguage are introduced. In Section 2, certain operations on relations (other than logical inference) are discussed and applied to the problems of redundancy and consistency in the user's model.

KEY WORDS AND PHRASES: data bank, data base, data structure, data organization, hierarchies of data, networks of data, relations, derivability, redundancy, consistency, composition, join, retrieval language, predicate calculus, security, data integrity
CR CATEGORIES: 3.70, 3.73, 3.75, 4.20, 4.22, 4.29

1. Relational Model and Normal Form

1.1. Introduction

This paper is concerned with the application of elementary relation theory to systems which provide shared access to large banks of formatted data. Except for a paper by Childs [1], the principal application of relations to data systems has been to deductive question-answering systems. Levein and Maron [2] provide numerous references to work in this area.

In contrast, the problems treated here are those of *data independence*—the independence of application programs and terminal activities from growth in data types and changes in data representation—and certain kinds of *data inconsistency* which are expected to become troublesome even in nondeductive systems.

The relational view (or model) of data described in Section 1 appears to be superior in several respects to the graph or network model [3, 4] presently in vogue for noninferential systems. It provides a means of describing data with its natural structure only—that is, without superimposing any additional structure for machine representation purposes. Accordingly, it provides a basis for a high level data language which will yield maximal independence between programs on the one hand and machine representation and organization of data on the other.

A further advantage of the relational view is that it forms a sound basis for treating derivability, redundancy, and consistency of relations—these are discussed in Section 2. The network model, on the other hand, has spawned a number of confusions, not the least of which is mistaking the derivation of connections for the derivation of relations (see remarks in Section 2 on the "connection trap").

Finally, the relational view permits a clearer evaluation of the scope and logical limitations of present formatted data systems, and also the relative merits (from a logical standpoint) of competing representations of data within a single system. Examples of this clearer perspective are cited in various parts of this paper. Implementations of systems to support the relational model are not discussed.

1.2. Data Dependencies in Present Systems

The provision of data description tables in recently developed information systems represents a major advance toward the goal of data independence [5, 6, 7]. Such tables facilitate changing certain characteristics of the data representation stored in a data bank. However, the variety of data representation characteristics which can be changed *without logically impairing some application programs* is still quite limited. Further, the model of data with which users interact is still cluttered with representational properties, particularly in regard to the representation of collections of data (as opposed to individual items). Three of the principal kinds of data dependencies which still need to be removed are: ordering dependence, indexing dependence, and access path dependence. In some systems these dependencies are not clearly separable from one another.

1.2.1. *Ordering Dependence.* Elements of data in a data bank may be stored in a variety of ways, some involving no concern for ordering, some permitting each element to participate in one ordering only, others permitting each element to participate in several orderings. Let us consider those existing systems which either require or permit data elements to be stored in at least one total ordering which is closely associated with the hardware-determined ordering of addresses. For example, the records of a file concerning parts might be stored in ascending order by part serial number. Such systems normally permit application programs to assume that the order of presentation of records from such a file is identical to (or is a subordering of) the stored ordering. Those application programs which take advantage of the stored ordering of a file are likely to fail to operate correctly if for some reason it becomes necessary to replace that ordering by a different one. Similar remarks hold for a stored ordering implemented by means of pointers.

It is unnecessary to single out any system as an example, because all the well-known information systems that are marketed today fail to make a clear distinction between order of presentation on the one hand and stored ordering on the other. Significant implementation problems must be solved to provide this kind of independence.

1.2.2. *Indexing Dependence.* In the context of formatted data, an index is usually thought of as a purely performance-oriented component of the data representation. It tends to improve response to queries and updates and, at the same time, slow down response to insertions and deletions. From an informational standpoint, an index is a redundant component of the data representation. If a system uses indices at all and if it is to perform well in an environment with changing patterns of activity on the data bank, an ability to create and destroy indices from time to time will probably be necessary. The question then arises: Can application programs and terminal activities remain invariant as indices come and go?

Present formatted data systems take widely different approaches to indexing. TDMS [7] unconditionally provides indexing on all attributes. The presently released version of IMS [5] provides the user with a choice for each file: a choice between no indexing at all (the hierarchic sequential organization) or indexing on the primary key only (the hierarchic indexed sequential organization). In neither case is the user's application logic dependent on the existence of the unconditionally provided indices. IDS [8], however, permits the file designers to select attributes to be indexed and to incorporate indices into the file structure by means of additional chains. Application programs taking advantage of the performance benefit of these indexing chains must refer to those chains by name. Such programs do not operate correctly if these chains are later removed.

1.2.3. *Access Path Dependence.* Many of the existing formatted data systems provide users with tree-structured files or slightly more general network models of the data. Application programs developed to work with these systems tend to be logically impaired if the trees or networks are changed in structure. A simple example follows.

Suppose the data bank contains information about parts and projects. For each part, the part number, part name, part description, quantity-on-hand, and quantity-on-order are recorded. For each project, the project number, project name, project description are recorded. Whenever a project makes use of a certain part, the quantity of that part committed to the given project is also recorded. Suppose that the system requires the user or file designer to declare or define the data in terms of tree structures. Then, any one of the hierarchical structures may be adopted for the information mentioned above (see Structures 1–5).

Structure 1. Projects Subordinate to Parts

File	*Segment*	*Fields*
F	PART	part #
		part name
		part description
		quantity-on-hand
		quantity-on-order
	PROJECT	project #
		project name
		project description
		quantity committed

Structure 2. Parts Subordinate to Projects

File	*Segment*	*Fields*
F	PROJECT	project #
		project name
		project description
	PART	part #
		part name
		part description
		quantity-on-hand
		quantity-on-order
		quantity committed

Structure 3. Parts and Projects as Peers
Commitment Relationship Subordinate to Projects

File	*Segment*	*Fields*
F	PART	part #
		part name
		part description
		quantity-on-hand
		quantity-on-order
G	PROJECT	project #
		project name
		project description
	PART	part #
		quantity committed

Structure 4. Parts and Projects as Peers
Commitment Relationship Subordinate to Parts

File	*Segment*	*Fields*
F	PART	part #
		part description
		quantity-on-hand
		quantity-on-order
	PROJECT	project #
		quantity committed
G	PROJECT	project #
		project name
		project description

Structure 5. Parts, Projects, and
Commitment Relationship as Peers

File	*Segment*	*Fields*
F	PART	part #
		part name
		part description
		quantity-on-hand
		quantity-on-order
G	PROJECT	project #
		project name
		project description
H	COMMIT	part #
		project #
		quantity committed

Now, consider the problem of printing out the part number, part name, and quantity committed for every part used in the project whose project name is "alpha." The following observations may be made regardless of which available tree-oriented information system is selected to tackle this problem. If a program P is developed for this problem assuming one of the five structures above—that is, P makes no test to determine which structure is in effect—then P will fail on at least three of the remaining structures. More specifically, if P succeeds with structure 5, it will fail with all the others; if P succeeds with structure 3 or 4, it will fail with at least 1, 2, and 5; if P succeeds with 1 or 2, it will fail with at least 3, 4, and 5. The reason is simple in each case. In the absence of a test to determine which structure is in effect, P fails because an attempt is made to execute a reference to a nonexistent file (available systems treat this as an error) or no attempt is made to execute a reference to a file containing needed information. The reader who is not convinced should develop sample programs for this simple problem.

Since, in general, it is not practical to develop application programs which test for all tree structurings permitted by the system, these programs fail when a change in structure becomes necessary.

Systems which provide users with a network model of the data run into similar difficulties. In both the tree and network cases, the user (or his program) is required to exploit a collection of user access paths to the data. It does not matter whether these paths are in close correspondence with pointer-defined paths in the stored representation—in IDS the correspondence is extremely simple, in TDMS it is just the opposite. The consequence, regardless of the stored representation, is that terminal activities and programs become dependent on the continued existence of the user access paths.

One solution to this is to adopt the policy that once a user access path is defined it will not be made obsolete until all application programs using that path have become obsolete. Such a policy is not practical, because the number of access paths in the total model for the community of users of a data bank would eventually become excessively large.

1.3. A Relational View of Data

The term *relation* is used here in its accepted mathematical sense. Given sets $S_1, S_2, \cdots, S_n$ (not necessarily distinct), R is a relation on these n sets if it is a set of n-tuples each of which has its first element from S_1, its second element from S_2, and so on.[1] We shall refer to S_j as the jth *domain* of R. As defined above, R is said to have *degree* n. Relations of degree 1 are often called *unary*, degree 2 *binary*, degree 3 *ternary*, and degree n *n-ary*.

For expository reasons, we shall frequently make use of an array representation of relations, but it must be remembered that this particular representation is not an essential part of the relational view being expounded. An array which represents an n-ary relation R has the following properties:

(1) Each row represents an n-tuple of R.
(2) The ordering of rows is immaterial.
(3) All rows are distinct.
(4) The ordering of columns is significant—it corresponds to the ordering $S_1, S_2, \cdots, S_n$ of the domains on which R is defined (see, however, remarks below on domain-ordered and domain-unordered relations).
(5) The significance of each column is partially conveyed by labeling it with the name of the corresponding domain.

The example in Figure 1 illustrates a relation of degree 4, called *supply*, which reflects the shipments-in-progress of parts from specified suppliers to specified projects in specified quantities.

supply	*(supplier*	*part*	*project*	*quantity)*
	1	2	5	17
	1	3	5	23
	2	3	7	9
	2	7	5	4
	4	1	1	12

Fig. 1. A relation of degree 4

One might ask: If the columns are labeled by the name of corresponding domains, why should the ordering of columns matter? As the example in Figure 2 shows, two columns may have identical headings (indicating identical domains) but possess distinct meanings with respect to the relation. The relation depicted is called *component*. It is a ternary relation, whose first two domains are called *part* and third domain is called *quantity*. The meaning of *component* (x, y, z) is that part x is an immediate component (or subassembly) of part y, and z units of part x are needed to assemble one unit of part y. It is a relation which plays a critical role in the parts explosion problem.

component	*(part*	*part*	*quantity)*
	1	5	9
	2	5	7
	3	5	2
	2	6	12
	3	6	3
	4	7	1
	6	7	1

Fig. 2. A relation with two identical domains

It is a remarkable fact that several existing information systems (chiefly those based on tree-structured files) fail to provide data representations for relations which have two or more identical domains. The present version of IMS/360 [5] is an example of such a system.

The totality of data in a data bank may be viewed as a collection of time-varying relations. These relations are of assorted degrees. As time progresses, each n-ary relation may be subject to insertion of additional n-tuples, deletion of existing ones, and alteration of components of any of its existing n-tuples.

[1] More concisely, R is a subset of the Cartesian product $S_1 \times S_2 \times \cdots \times S_n$.

In many commercial, governmental, and scientific data banks, however, some of the relations are of quite high degree (a degree of 30 is not at all uncommon). Users should not normally be burdened with remembering the domain ordering of any relation (for example, the ordering *supplier*, then *part*, then *project*, then *quantity* in the relation *supply*). Accordingly, we propose that users deal, not with relations which are domain-ordered, but with *relationships* which are their domain-unordered counterparts.[2] To accomplish this, domains must be uniquely identifiable at least within any given relation, without using position. Thus, where there are two or more identical domains, we require in each case that the domain name be qualified by a distinctive *role name*, which serves to identify the role played by that domain in the given relation. For example, in the relation *component* of Figure 2, the first domain *part* might be qualified by the role name *sub*, and the second by *super*, so that users could deal with the relationship *component* and its domains—*sub.part super.part, quantity*—without regard to any ordering between these domains.

To sum up, it is proposed that most users should interact with a relational model of the data consisting of a collection of time-varying relationships (rather than relations). Each user need not know more about any relationship than its name together with the names of its domains (role qualified whenever necessary).[3] Even this information might be offered in menu style by the system (subject to security and privacy constraints) upon request by the user.

There are usually many alternative ways in which a relational model may be established for a data bank. In order to discuss a preferred way (or normal form), we must first introduce a few additional concepts (active domain, primary key, foreign key, nonsimple domain) and establish some links with terminology currently in use in information systems programming. In the remainder of this paper, we shall not bother to distinguish between relations and relationships except where it appears advantageous to be explicit.

Consider an example of a data bank which includes relations concerning parts, projects, and suppliers. One relation called *part* is defined on the following domains:

(1) part number
(2) part name
(3) part color
(4) part weight
(5) quantity on hand
(6) quantity on order

and possibly other domains as well. Each of these domains is, in effect, a pool of values, some or all of which may be represented in the data bank at any instant. While it is conceivable that, at some instant, all part colors are present, it is unlikely that all possible part weights, part names, and part numbers are. We shall call the set of values represented at some instant the *active domain* at that instant.

Normally, one domain (or combination of domains) of a given relation has values which uniquely identify each element (*n*-tuple) of that relation. Such a domain (or combination) is called a *primary key*. In the example above, part number would be a primary key, while part color would not be. A primary key is *nonredundant* if it is either a simple domain (not a combination) or a combination such that none of the participating simple domains is superfluous in uniquely identifying each element. A relation may possess more than one nonredundant primary key. This would be the case in the example if different parts were always given distinct names. Whenever a relation has two or more nonredundant primary keys, one of them is arbitrarily selected and called *the* primary key of that relation.

A common requirement is for elements of a relation to cross-reference other elements of the same relation or elements of a different relation. Keys provide a user-oriented means (but not the only means) of expressing such cross-references. We shall call a domain (or domain combination) of relation R a *foreign key* if it is not the primary key of R but its elements are values of the primary key of some relation S (the possibility that S and R are identical is not excluded). In the relation *supply* of Figure 1, the combination of *supplier, part, project* is the primary key, while each of these three domains taken separately is a foreign key.

In previous work there has been a strong tendency to treat the data in a data bank as consisting of two parts, one part consisting of entity descriptions (for example, descriptions of suppliers) and the other part consisting of relations between the various entities or types of entities (for example, the *supply* relation). This distinction is difficult to maintain when one may have foreign keys in any relation whatsoever. In the user's relational model there appears to be no advantage to making such a distinction (there may be some advantage, however, when one applies relational concepts to machine representations of the user's set of relationships).

So far, we have discussed examples of relations which are defined on simple domains—domains whose elements are atomic (nondecomposable) values. Nonatomic values can be discussed within the relational framework. Thus, some domains may have relations as elements. These relations may, in turn, be defined on nonsimple domains, and so on. For example, one of the domains on which the relation *employee* is defined might be *salary history*. An element of the salary history domain is a binary relation defined on the domain *date* and the domain *salary*. The *salary history* domain is the set of all such binary relations. At any instant of time there are as many instances of the *salary history* relation in the data bank as there are employees. In contrast, there is only one instance of the *employee* relation.

The terms attribute and repeating group in present data base terminology are roughly analogous to simple domain and nonsimple domain, respectively. Much of the confusion in present terminology is due to failure to distinguish between type and instance (as in "record") and between components of a user model of the data on the one hand and their machine representation counterparts on the other hand (again, we cite "record" as an example).

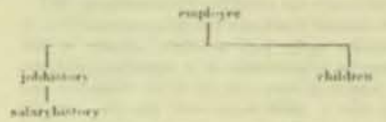
[2] In mathematical terms, a relationship is an equivalence class of those relations that are equivalent under permutation of domains (see Section 2.1.1).

[3] Naturally, as with any data put into and retrieved from a computer system, the user will normally make far more effective use of the data if he is aware of its meaning.

1.4. Normal Form

A relation whose domains are all simple can be represented in storage by a two-dimensional column-homogeneous array of the kind discussed above. Some more complicated data structure is necessary for a relation with one or more nonsimple domains. For this reason (and others to be cited below) the possibility of eliminating nonsimple domains appears worth investigating.[4] There is, in fact, a very simple elimination procedure, which we shall call normalization.

Consider, for example, the collection of relations exhibited in Figure 3(a). *Job history* and *children* are nonsimple domains of the relation *employee*. *Salary history* is a nonsimple domain of the relation *job history*. The tree in Figure 3(a) shows just these interrelationships of the nonsimple domains.



employee (*man#*, name, birthdate, jobhistory, children)
jobhistory (*jobdate*, title, salaryhistory)
salaryhistory (*salarydate*, salary)
children (*childname*, birthyear)

Fig. 3(a). Unnormalized set

employee' (*man#*, name, birthdate)
jobhistory' (*man#*, *jobdate*, title)
salaryhistory' (*man#*, *jobdate*, *salarydate*, salary)
children' (*man#*, *childname*, birthyear)

Fig. 3(b). Normalized set

Normalization proceeds as follows. Starting with the relation at the top of the tree, take its primary key and expand each of the immediately subordinate relations by inserting this primary key domain or domain combination. The primary key of each expanded relation consists of the primary key before expansion augmented by the primary key copied down from the parent relation. Now, strike out from the parent relation all nonsimple domains, remove the top node of the tree, and repeat the same sequence of operations on each remaining subtree.

The result of normalizing the collection of relations in Figure 3(a) is the collection in Figure 3(b). The primary key of each relation is italicized to show how such keys are expanded by the normalization.

[4] M. E. Sanko of IBM, San Jose, independently recognized the desirability of eliminating nonsimple domains.

If normalization as described above is to be applicable, the unnormalized collection of relations must satisfy the following conditions:

(1) The graph of interrelationships of the nonsimple domains is a collection of trees.
(2) No primary key has a component domain which is nonsimple.

The writer knows of no application which would require any relaxation of these conditions. Further operations of a normalizing kind are possible. These are not discussed in this paper.

The simplicity of the array representation which becomes feasible when all relations are cast in normal form is not only an advantage for storage purposes but also for communication of bulk data between systems which use widely different representations of the data. The communication form would be a suitably compressed version of the array representation and would have the following advantages:

(1) It would be devoid of pointers (address-valued or displacement-valued).
(2) It would avoid all dependence on hash addressing schemes.
(3) It would contain no indices or ordering lists.

If the user's relational model is set up in normal form, names of items of data in the data bank can take a simpler form than would otherwise be the case. A general name would take a form such as

$$R(g).r.d$$

where R is a relational name; g is a generation identifier (optional); r is a role name (optional); d is a domain name. Since g is needed only when several generations of a given relation exist, or are anticipated to exist, and r is needed only when the relation R has two or more domains named d, the simple form $R.d$ will often be adequate.

1.5. Some Linguistic Aspects

The adoption of a relational model of data, as described above, permits the development of a universal data sublanguage based on an applied predicate calculus. A first-order predicate calculus suffices if the collection of relations is in normal form. Such a language would provide a yardstick of linguistic power for all other proposed data languages, and would itself be a strong candidate for embedding (with appropriate syntactic modification) in a variety of host languages (programming, command- or problem-oriented). While it is not the purpose of this paper to describe such a language in detail, its salient features would be as follows.

Let us denote the data sublanguage by R and the host language by H. R permits the declaration of relations and their domains. Each declaration of a relation identifies the primary key for that relation. Declared relations are added to the system catalog for use by any members of the user community who have appropriate authorization. H permits supporting declarations which indicate, perhaps less permanently, how these relations are represented in storage R permits the specification for retrieval of any subset of data from the data bank. Action on such a retrieval request is subject to security constraints.

The universality of the data sublanguage lies in its descriptive ability (not its computing ability). In a large data bank each subset of the data has a very large number of possible (and sensible) descriptions, even when we assume (as we do) that there is only a finite set of function subroutines to which the system has access for use in qualifying data for retrieval. Thus, the class of qualification expressions which can be used in a set specification must have the descriptive power of the class of well-formed formulas of an applied predicate calculus. It is well known that to preserve this descriptive power it is unnecessary to express (in whatever syntax is chosen) every formula of the selected predicate calculus. For example, just those in prenex normal form are adequate [9].

Arithmetic functions may be needed in the qualification or other parts of retrieval statements. Such functions can be defined in H and invoked in R.

A set so specified may be fetched for query purposes only, or it may be held for possible changes. Insertions take the form of adding new elements to declared relations without regard to any ordering that may be present in their machine representation. Deletions which are effective for the community (as opposed to the individual user or subcommunities) take the form of removing elements from declared relations. Some deletions and updates may be triggered by others, if deletion and update dependencies between specified relations are declared in R.

One important effect that the view adopted toward data has on the language used to retrieve it is in the naming of data elements and sets. Some aspects of this have been discussed in the previous section. With the usual network view, users will often be burdened with coining and using more relation names than are absolutely necessary, since names are associated with paths (or path types) rather than with relations.

Once a user is aware that a certain relation is stored, he will expect to be able to exploit[5] it using any combination of its arguments as "knowns" and the remaining arguments as "unknowns," because the information (like Everest) is there. This is a system feature (missing from many current information systems) which we shall call (logically) *symmetric exploitation* of relations. Naturally, symmetry in performance is not to be expected.

To support symmetric exploitation of a single binary relation, two directed paths are needed. For a relation of degree n, the number of paths to be named and controlled is n factorial.

Again, if a relational view is adopted in which every n-ary relation $(n > 2)$ has to be expressed by the user as a nested expression involving only binary relations (see Feldman's LEAP System [10], for example) then $2n - 1$ names have to be coined instead of only $n + 1$ with direct n-ary notation as described in Section 1.2. For example, the 4-ary relation *supply* of Figure 1, which entails 5 names in n-ary notation, would be represented in the form

$$P\ (supplier,\ Q\ (part,\ R\ (project,\ quantity)))$$

in nested binary notation and, thus, employ 7 names.

A further disadvantage of this kind of expression is its asymmetry. Although this asymmetry does not prohibit symmetric exploitation, it certainly makes some bases of interrogation very awkward for the user to express (consider, for example, a query for those parts and quantities related to certain given projects via Q and R).

[5] Exploiting a relation includes query, update, and delete.

1.6. Expressible, Named, and Stored Relations

Associated with a data bank are two collections of relations: the *named set* and the *expressible set*. The named set is the collection of all those relations that the community of users can identify by means of a simple name (or identifier). A relation R acquires membership in the named set when a suitably authorized user declares R; it loses membership when a suitably authorized user cancels the declaration of R.

The expressible set is the total collection of relations that can be designated by expressions in the data language. Such expressions are constructed from simple names of relations in the named set; names of generations, roles and domains; logical connectives; the quantifiers of the predicate calculus;[6] and certain constant relation symbols such as $=$, $>$. The named set is a subset of the expressible set—usually a very small subset.

Since some relations in the named set may be time-independent combinations of others in that set, it is useful to consider associating with the named set a collection of statements that define these time-independent constraints. We shall postpone further discussion of this until we have introduced several operations on relations (see Section 2).

One of the major problems confronting the designer of a data system which is to support a relational model for its users is that of determining the class of stored representations to be supported. Ideally, the variety of permitted data representations should be just adequate to cover the spectrum of performance requirements of the total collection of installations. Too great a variety leads to unnecessary overhead in storage and continual reinterpretation of descriptions for the structures currently in effect.

For any selected class of stored representations the data system must provide a means of translating user requests expressed in the data language of the relational model into corresponding—and efficient—actions on the current stored representation. For a high level data language this presents a challenging design problem. Nevertheless, it is a problem which must be solved—as more users obtain concurrent access to a large data bank, responsibility for providing efficient response and throughput shifts from the individual user to the data system.

[6] Because each relation in a practical data bank is a finite set at every instant of time, the existential and universal quantifiers can be expressed in terms of a function that counts the number of elements in any finite set.

2. Redundancy and Consistency

2.1. Operations on Relations

Since relations are sets, all of the usual set operations are applicable to them. Nevertheless, the result may not be a relation; for example, the union of a binary relation and a ternary relation is not a relation.

The operations discussed below are specifically for relations. These operations are introduced because of their key role in deriving relations from other relations. Their principal application is in noninferential information systems—systems which do not provide logical inference services—although their applicability is not necessarily destroyed when such services are added.

Most users would not be directly concerned with these operations. Information systems designers and people concerned with data bank control should, however, be thoroughly familiar with them.

2.1.1. *Permutation.* A binary relation has an array representation with two columns. Interchanging these columns yields the converse relation. More generally, if a permutation is applied to the columns of an n-ary relation, the resulting relation is said to be a *permutation* of the given relation. There are, for example, $4! = 24$ permutations of the relation *supply* in Figure 1, if we include the identity permutation which leaves the ordering of columns unchanged.

Since the user's relational model consists of a collection of relationships (domain-unordered relations), permutation is not relevant to such a model considered in isolation. It is, however, relevant to the consideration of stored representations of the model. In a system which provides symmetric exploitation of relations, the set of queries answerable by a stored relation is identical to the set answerable by any permutation of that relation. Although it is logically unnecessary to store both a relation and some permutation of it, performance considerations could make it advisable.

2.1.2. *Projection.* Suppose now we select certain columns of a relation (striking out the others) and then remove from the resulting array any duplication in the rows. The final array represents a relation which is said to be a *projection* of the given relation.

A selection operator π is used to obtain any desired permutation, projection, or combination of the two operations. Thus, if L is a list of k indices[7] $L = i_1, i_2, \cdots, i_k$ and R is an n-ary relation $(n \geq k)$, then $\pi_L(R)$ is the k-ary relation whose jth column is column i_j of R $(j = 1, 2, \cdots, k)$ except that duplication in resulting rows is removed. Consider the relation *supply* of Figure 1. A permuted projection of this relation is exhibited in Figure 4. Note that, in this particular case, the projection has fewer n-tuples than the relation from which it is derived.

2.1.3. *Join.* Suppose we are given two binary relations, which have some domain in common. Under what circumstances can we combine these relations to form a ternary relation which preserves all of the information in the given relations?

[7] When dealing with relationships, we use domain names (role-qualified whenever necessary) instead of domain positions.

The example in Figure 5 shows two relations R, S, which are joinable without loss of information, while Figure 6 shows a join of R with S. A binary relation R is *joinable* with a binary relation S if there exists a ternary relation U such that $\pi_{12}(U) = R$ and $\pi_{23}(U) = S$. Any such ternary relation is called a *join* of R with S. If R, S are binary relations such that $\pi_2(R) = \pi_1(S)$, then R is joinable with S. One join that always exists in such a case is the *natural join* of R with S defined by

$$R*S = \{(a, b, c) : R(a, b) \wedge S(b, c)\}$$

where $R(a, b)$ has the value *true* if (a, b) is a member of R and similarly for $S(b, c)$. It is immediate that

$$\pi_{12}(R*S) = R$$

and

$$\pi_{23}(R*S) = S.$$

Note that the join shown in Figure 6 is the natural join of R with S from Figure 5. Another join is shown in Figure 7.

Π_{31}(*supply*)	(*project*	*supplier*)
	5	1
	5	2
	1	4
	7	2

FIG. 4. A permuted projection of the relation in Figure 1

R	(*supplier*	*part*)	S	(*part*	*project*)
	1	1		1	1
	2	1		1	2
	2	2		2	1

FIG. 5. Two joinable relations

R*S	(*supplier*	*part*	*project*)
	1	1	1
	1	1	2
	2	1	1
	2	1	2
	2	2	1

FIG. 6. The natural join of R with S (from Figure 5)

U	(*supplier*	*part*	*project*)
	1	1	2
	2	1	1
	2	2	1

FIG. 7. Another join of R with S (from Figure 5)

Inspection of these relations reveals an element (element 1) of the domain *part* (the domain on which the join is to be made) with the property that it possesses more than one relative under R and also under S. It is this element which gives rise to the plurality of joins. Such an element in the joining domain is called a *point of ambiguity* with respect to the joining of R with S.

If either $\pi_{21}(R)$ or S is a function,* no point of ambiguity can occur in joining R with S. In such a case, the natural join of R with S is the only join of R with S. Note that the reiterated qualification "of R with S" is necessary, because S might be joinable with R (as well as R with S), and this join would be an entirely separate consideration. In Figure 5, none of the relations R, $\pi_{21}(R)$, S, $\pi_{21}(S)$ is a function.

Ambiguity in the joining of R with S can sometimes be resolved by means of other relations. Suppose we are given, or can derive from sources independent of R and S, a relation T on the domains *project* and *supplier* with the following properties:

(1) $\pi_1(T) = \pi_2(S)$,

(2) $\pi_2(T) = \pi_1(R)$,

(3) $T(j, s) \rightarrow \exists p(R(s, p) \wedge S(p, j))$,

(4) $R(s, p) \rightarrow \exists j(S(p, j) \wedge T(j, s))$,

(5) $S(p, j) \rightarrow \exists s(T(j, s) \wedge R(s, p))$,

then we may form a three-way join of R, S, T; that is, a ternary relation such that

$$\pi_{12}(U) = R, \quad \pi_{23}(U) = S, \quad \pi_{31}(U) = T.$$

Such a join will be called a *cyclic 3-join* to distinguish it from a *linear 3-join* which would be a quaternary relation V such that

$$\pi_{12}(V) = R, \quad \pi_{23}(V) = S, \quad \pi_{34}(V) = T.$$

While it is possible for more than one cyclic 3-join to exist (see Figures 8, 9, for an example), the circumstances under which this can occur entail much more severe constraints

R	(s	p)
	1	a
	2	a
	2	b

S	(p	j)
	a	d
	a	e
	b	d
	b	e

T	(j	s)
	d	1
	d	2
	e	2
	e	2

FIG. 8. Binary relations with a plurality of cyclic 3-joins

U	(s	p	j)
	1	a	d
	2	a	e
	2	b	d
	2	b	e

U'	(s	p	j)
	1	a	d
	2	a	d
	2	a	e
	2	b	d
	2	b	e

FIG. 9. Two cyclic 3-joins of the relations in Figure 8

than those for a plurality of 2-joins. To be specific, the relations R, S, T must possess points of ambiguity with respect to joining R with S (say point x), S with T (say y), and T with R (say z), and, furthermore, y must be a relative of x under S, z a relative of y under T, and x a relative of z under R. Note that in Figure 8 the points $x = a$; $y = d$; $z = 2$ have this property.

The natural linear 3-join of three binary relations R, S, T is given by

$$R*S*T = \{(a, b, c, d) : R(a, b) \wedge S(b, c) \wedge T(c, d)\}$$

where parentheses are not needed on the left-hand side because the natural 2-join (*) is associative. To obtain the cyclic counterpart, we introduce the operator γ which produces a relation of degree $n - 1$ from a relation of degree n by tying its ends together. Thus, if R is an n-ary relation ($n \geq 2$), the *tie* of R is defined by the equation

$$\gamma(R) = \{(a_1, a_2, \cdots, a_{n-1}) : R(a_1, a_2, \cdots, a_{n-1}, a_n) \wedge a_1 = a_n\}.$$

We may now represent the natural cyclic 3-join of R, S, T by the expression

$$\gamma(R*S*T).$$

Extension of the notions of linear and cyclic 3-join and their natural counterparts to the joining of n binary relations (where $n \geq 3$) is obvious. A few words may be appropriate, however, regarding the joining of relations which are not necessarily binary. Consider the case of two relations R (degree r), S (degree s) which are to be joined on p of their domains ($p < r$, $p < s$). For simplicity, suppose these p domains are the last p of the r domains of R, and the first p of the s domains of S. If this were not so, we could always apply appropriate permutations to make it so. Now, take the Cartesian product of the first r-p domains of R, and call this new domain A. Take the Cartesian product of the last p domains of R, and call this B. Take the Cartesian product of the last s-p domains of S and call this C.

We can treat R as if it were a binary relation on the domains A, B. Similarly, we can treat S as if it were a binary relation on the domains B, C. The notions of linear and cyclic 3-join are now directly applicable. A similar approach can be taken with the linear and cyclic n-joins of n relations of assorted degrees.

2.1.4. *Composition.* The reader is probably familiar with the notion of composition applied to functions. We shall discuss a generalization of that concept and apply it first to binary relations. Our definitions of composition and composability are based very directly on the definitions of join and joinability given above.

Suppose we are given two relations R, S. T is a *composition of R with S* if there exists a join U of R with S such that $T = \pi_{13}(U)$. Thus, two relations are composable if and only if they are joinable. However, the existence of more than one join of R with S does not imply the existence of more than one composition of R with S.

Corresponding to the natural join of R with S is the *natural composition** of R with S defined by

$$R \cdot S = \pi_{13}(R*S).$$

Taking the relations R, S from Figure 5, their natural composition is exhibited in Figure 10 and another composition is exhibited in Figure 11 (derived from the join exhibited in Figure 7).

R·S	(project	supplier)
	1	1
	1	2
	2	1
	2	2

FIG. 10. The natural composition of R with S (from Figure 5)

T	(project	supplier)
	1	2
	2	1

FIG. 11. Another composition of R with S (from Figure 5)

When two or more joins exist, the number of distinct compositions may be as few as one or as many as the number of distinct joins. Figure 12 shows an example of two relations which have several joins but only one composition. Note that the ambiguity of point c is lost in composing R with S, because of unambiguous associations made via the points a, b, d, e.

R	(supplier	part)
	1	a
	1	b
	1	c
	2	c
	2	d
	2	e

S	(part	project)
	a	g
	b	f
	c	f
	c	g
	d	g
	e	f

FIG. 12. Many joins, only one composition

Extension of composition to pairs of relations which are not necessarily binary (and which may be of different degrees) follows the same pattern as extension of pairwise joining to such relations.

A lack of understanding of relational composition has led several systems designers into what may be called the *connection trap*. This trap may be described in terms of the following example. Suppose each supplier description is linked by pointers to the descriptions of each part supplied by that supplier, and each part description is similarly linked to the descriptions of each project which uses that part. A conclusion is now drawn which is, in general, erroneous: namely that, if all possible paths are followed from a given supplier via the parts he supplies to the projects using those parts, one will obtain a valid set of all projects supplied by that supplier. Such a conclusion is correct only in the very special case that the target relation between projects and suppliers is, in fact, the natural composition of the other two relations—and we must normally add the phrase "for all time," because this is usually implied in claims concerning path-following techniques.

2.1.5. *Restriction.* A subset of a relation is a relation. One way in which a relation S may act on a relation R to generate a subset of R is through the operation *restriction of R by S*. This operation is a generalization of the restriction of a function to a subset of its domain, and is defined as follows.

Let L, M be equal-length lists of indices such that $L = i_1, i_2, \cdots, i_k$, $M = j_1, j_2, \cdots, j_k$ where $k \leq$ degree of R and $k \leq$ degree of S. Then the L, M restriction of R by S denoted $R_L|_M S$ is the maximal subset R' of R such that

$$\pi_L(R') = \pi_M(S).$$

The operation is defined only if equality is applicable between elements of $\pi_{i_h}(R)$ on the one hand and $\pi_{j_h}(S)$ on the other for all $h = 1, 2, \cdots, k$.

The three relations R, S, R' of Figure 13 satisfy the equation $R' = R_{(2,3)}|_{(1,2)}S$.

R	(s	p	j)
	1	a	A
	2	a	A
	2	a	B
	2	b	A
	2	b	B

S	(p	j)
	a	A
	c	B
	b	B

R'	(s	p	j)
	1	a	A
	2	a	A
	2	b	B

FIG. 13. Example of restriction

We are now in a position to consider various applications of these operations on relations.

2.2. REDUNDANCY

Redundancy in the named set of relations must be distinguished from redundancy in the stored set of representations. We are primarily concerned here with the former. To begin with, we need a precise notion of derivability for relations.

Suppose θ is a collection of operations on relations and each operation has the property that from its operands it yields a unique relation (thus natural join is eligible, but join is not). A relation R is *θ-derivable* from a set S of relations if there exists a sequence of operations from the collection θ which, for all time, yields R from members of S. The phrase "for all time" is present, because we are dealing with time-varying relations, and our interest is in derivability which holds over a significant period of time. For the named set of relationships in noninferential systems, it appears that an adequate collection θ_1 contains the following operations: projection, natural join, tie, and restriction. Permutation is irrelevant and natural composition need not be included, because it is obtainable by taking a natural join and then a projection. For the stored set of representations, an adequate collection θ_2 of operations would include permutation and additional operations concerned with subsetting and merging relations, and ordering and connecting their elements.

2.2.1. *Strong Redundancy.* A set of relations is *strongly redundant* if it contains at least one relation that possesses a projection which is derivable from other projections of relations in the set. The following two examples are intended to explain why strong redundancy is defined this way, and to demonstrate its practical use. In the first ex-

* A function is a binary relation, which is one-one or many-one, but not one-many.

* Other writers tend to ignore compositions other than the natural one, and accordingly refer to this particular composition as *the* composition—see, for example, Kelley's "General Topology."

ample the collection of relations consists of just the following relation:

employee (*serial #*, *name*, *manager#*, *managername*)

with *serial#* as the primary key and *manager#* as a foreign key. Let us denote the active domain by Δ_t, and suppose that

$$\Delta_t(manager\#) \subset \Delta_t(serial\#)$$

and

$$\Delta_t(managername) \subset \Delta_t(name)$$

for all time t. In this case the redundancy is obvious: the domain *managername* is unnecessary. To see that it is a strong redundancy as defined above, we observe that

$$\pi_{34}(employee) = \pi_{12}(employee)\,{}_1|_1\,\pi_3(employee).$$

In the second example the collection of relations includes a relation S describing suppliers with primary key $s\#$, a relation D describing departments with primary key $d\#$, a relation J describing projects with primary key $j\#$, and the following relations:

$$P(s\#, d\#, \cdots), \qquad Q(s\#, j\#, \cdots), \qquad R(d\#, j\#, \cdots),$$

where in each case $\cdots$ denotes domains other than $s\#$, $d\#$, $j\#$. Let us suppose the following condition C is known to hold independent of time: supplier s supplies department d (relation P) if and only if supplier s supplies some project j (relation Q) to which d is assigned (relation R). Then, we can write the equation

$$\pi_{12}(P) = \pi_{12}(Q) \cdot \pi_{12}(R)$$

and thereby exhibit a strong redundancy.

An important reason for the existence of strong redundancies in the named set of relationships is user convenience. A particular case of this is the retention of semiobsolete relationships in the named set so that old programs that refer to them by name can continue to run correctly. Knowledge of the existence of strong redundancies in the named set enables a system or data base administrator greater freedom in the selection of stored representations to cope more efficiently with current traffic. If the strong redundancies in the named set are directly reflected in strong redundancies in the stored set (or if other strong redundancies are introduced into the stored set), then, generally speaking, extra storage space and update time are consumed with a potential drop in query time for some queries and in load on the central processing units.

2.2.2. *Weak Redundancy*. A second type of redundancy may exist. In contrast to strong redundancy it is not characterized by an equation. A collection of relations is *weakly redundant* if it contains a relation that has a projection which is not derivable from other members but is at all times a projection of *some* join of other projections of relations in the collection.

We can exhibit a weak redundancy by taking the second example (cited above) for a strong redundancy, and assuming now that condition C does not hold at all times. The relations $\pi_{12}(P)$, $\pi_{12}(Q)$, $\pi_{12}(R)$ are complex[*] relations with the possibility of points of ambiguity occurring from time to time in the potential joining of any two. Under these circumstances, none of them is derivable from the other two. However, constraints do exist between them, since each is a projection of some cyclic join of the three of them. One of the weak redundancies can be characterized by the statement: for all time, $\pi_{12}(P)$ is *some* composition of $\pi_{12}(Q)$ with $\pi_{12}(R)$. The composition in question might be the natural one at some instant and a nonnatural one at another instant.

Generally speaking, weak redundancies are inherent in the logical needs of the community of users. They are not removable by the system or data base administrator. If they appear at all, they appear in both the named set and the stored set of representations.

2.3. Consistency

Whenever the named set of relations is redundant in either sense, we shall associate with that set a collection of statements which define all of the redundancies which hold independent of time between the member relations. If the information system lacks—and it most probably will—detailed semantic information about each named relation, it cannot deduce the redundancies applicable to the named set. It might, over a period of time, make attempts to induce the redundancies, but such attempts would be fallible.

Given a collection C of time-varying relations, an associated set Z of constraint statements and an instantaneous value V for C, we shall call the state (C, Z, V) *consistent* or *inconsistent* according as V does or does not satisfy Z. For example, given stored relations R, S, T together with the constraint statement "$\pi_{12}(T)$ is a composition of $\pi_{12}(R)$ with $\pi_{12}(S)$", we may check from time to time that the values stored for R, S, T satisfy this constraint. An algorithm for making this check would examine the first two columns of each of R, S, T (in whatever way they are represented in the system) and determine whether

(1) $\pi_1(T) = \pi_1(R)$,

(2) $\pi_2(T) = \pi_2(S)$,

(3) for every element pair (a, c) in the relation $\pi_{12}(T)$ there is an element b such that (a, b) is in $\pi_{12}(R)$ and (b, c) is in $\pi_{12}(S)$.

There are practical problems (which we shall not discuss here) in taking an instantaneous snapshot of a collection of relations, some of which may be very large and highly variable.

It is important to note that consistency as defined above is a property of the instantaneous state of a data bank, and is independent of how that state came about. Thus, in particular, there is no distinction made on the basis of whether a user generated an inconsistency due to an act of omission or an act of commission. Examination of a simple example will show the reasonableness of this (possibly unconventional) approach to consistency.

Suppose the named set C includes the relations S, J, D, P, Q, R of the example in Section 2.2 and that P, Q, R possess either the strong or weak redundancies described therein (in the particular case now under consideration, it does not matter which kind of redundancy occurs). Further, suppose that at some time t the data bank state is consistent and contains no project j such that supplier 2 supplies project j and j is assigned to department 5. Accordingly, there is no element $(2, 5)$ in $\pi_{12}(P)$. Now, a user introduces the element $(2, 5)$ into $\pi_{12}(P)$ by inserting some appropriate element into P. The data bank state is now inconsistent. The inconsistency could have arisen from an act of omission, if the input $(2, 5)$ is correct, and there does exist a project j such that supplier 2 supplies j and j is assigned to department 5. In this case, it is very likely that the user intends in the near future to insert elements into Q and R which will have the effect of introducing $(2, j)$ into $\pi_{12}(Q)$ and $(5, j)$ in $\pi_{12}(R)$. On the other hand, the input $(2, 5)$ might have been faulty. It could be the case that the user intended to insert some other element into P—an element whose insertion would transform a consistent state into a consistent state. The point is that the system will normally have no way of resolving this question without interrogating its environment (perhaps the user who created the inconsistency).

There are, of course, several possible ways in which a system can detect inconsistencies and respond to them. In one approach the system checks for possible inconsistency whenever an insertion, deletion, or key update occurs. Naturally, such checking will slow these operations down. If an inconsistency has been generated, details are logged internally, and if it is not remedied within some reasonable time interval, either the user or someone responsible for the security and integrity of the data is notified. Another approach is to conduct consistency checking as a batch operation once a day or less frequently. Inputs causing the inconsistencies which remain in the data bank state at checking time can be tracked down if the system maintains a journal of all state-changing transactions. This latter approach would certainly be superior if few nontransitory inconsistencies occurred.

2.4. Summary

In Section 1 a relational model of data is proposed as a basis for protecting users of formatted data systems from the potentially disruptive changes in data representation caused by growth in the data bank and changes in traffic. A normal form for the time-varying collection of relationships is introduced.

In Section 2 operations on relations and two types of redundancy are defined and applied to the problem of maintaining the data in a consistent state. This is bound to become a serious practical problem as more and more different types of data are integrated together into common data banks.

Many questions are raised and left unanswered. For example, only a few of the more important properties of the data sublanguage in Section 1.4 are mentioned. Neither the purely linguistic details of such a language nor the implementation problems are discussed. Nevertheless, the material presented should be adequate for experienced systems programmers to visualize several approaches. It is also hoped that this paper can contribute to greater precision in work on formatted data systems.

Acknowledgment. It was C. T. Davies of IBM Poughkeepsie who convinced the author of the need for data independence in future information systems. The author wishes to thank him and also F. P. Palermo, C. P. Wang, E. B. Altman, and M. E. Senko of the IBM San Jose Research Laboratory for helpful discussions.

Received September, 1969; revised February, 1970

References

1. Childs, D. L. Feasibility of a set-theoretical data structure—a general structure based on a reconstituted definition of relation. Proc. IFIP Cong., 1968, North Holland Pub. Co., Amsterdam, p. 162-172.
2. Levein, R. E., and Maron, M. E. A computer system for inference execution and data retrieval. *Comm. ACM 10*, 11 (Nov. 1967), 715-721.
3. Bachman, C. W. Software for random access processing. *Datamation* (Apr. 1965), 36-41.
4. McGee, W. C. Generalized file processing. In *Annual Review in Automatic Programming 5*, 13, Pergamon Press, New York, 1969, pp. 77-149.
5. *Information Management System/360, Application Description Manual* H20-0524-1. IBM Corp., White Plains, N. Y., July 1968.
6. GIS (Generalized Information System), Application Description Manual H20-0574. IBM Corp., White Plains, N. Y., 1965.
7. Bleier, R. E. Treating hierarchical data structures in the SDC time-shared data management system (TDMS). Proc. ACM 22nd Nat. Conf., 1967, MDI Publications, Wayne, Pa., pp. 41-49.
8. IDS Reference Manual GE 625/635, GE Inform. Sys. Div., Pheonix, Ariz., CPB 1093B, Feb. 1968.
9. Church, A. *An Introduction to Mathematical Logic I*. Princeton U. Press, Princeton, N.J., 1956.
10. Feldman, J. A., and Rovner, P. D. An Algol-based associative language. Stanford Artificial Intelligence Rep. AI-66, Aug. 1, 1968.

[*] A binary relation is complex if neither it nor its converse is a function.

MASTER COPY

The Costs Of Network Ownership

A Research Report By:

DR. MICHAEL E. TREACY

and

INDEX GROUP, INC.

Contents

I. Executive Summary 2

II. Introduction 4

III. Study Methodology 6

IV. The Cost Model 8

V. Corporate Network Case Studies 14

VI. Impact Of Topology On Cost 21

VII. Impact Of Vendor On Cost 23

VIII. Multiple Field Offices Networks 26

IX. Manufacturing Site Networks 31

X. Putting the Model To Use 35

XI. Conclusion 39

XII. Notes 41

Appendix A - Case Analysis Of Consolidated Manufacturing, Inc.

Appendix B - Case Analysis Of American Equipment Corporation

I. Executive Summary

1. Background

In step with the increasing prominence of networking in corporate I.S. plans and budgets, interest in the costs of owning and operating computer networks has been growing steadily. To date, few organizations have in place an efficient and comprehensive approach to assessing their total current and potential network costs.

From May 1987 to August 1988, Dr. Michael E. Treacy and Index Group, Inc., the Cambridge, Massachusetts-based information technology management consulting firm, studied the costs of network ownership. This report describes the approach taken and presents the key findings and conclusions of the study, which was commissioned by Digital Equipment Corporation.

2. Approach

The study team developed a model for categorizing and evaluating network cost-related information. The model places network costs into five categories: equipment, software, personnel, communications carriers, and facilities. It further analyzes the costs over three phases: acquisition (one-time costs), routine operation and maintenance, and incremental change.

The study team tested the model by applying it in case studies of 17 active U.S. and European networks. The networks studied were of three types: "corporate" networks, "multiple field offices" networks and "manufacturing site" networks (Chapter III elaborates on these terms).

To enhance the efficiency of gathering accurate data on the subject networks, the team emphasized the actual resources used (e.g., the specific equipment, job categories, staffing levels, etc.) as distinct from the financial disbursements that occurred. Through the development and application of standard-price tables, the researchers could then calculate cost. Standard pricing also improved the team's ability to group and compare networks on an equal basis.

3. General Findings

The researchers concentrated on three objectives in their data analysis: to determine the cost structures of the networks under study, to observe the effects of network topology on cost, and to assess the impact on cost of vendor choice. The general findings were as follows:

- The initial cost of a network is only a fraction of the overall cost of the network; operating and incremental change costs can be substantial.
- Personnel costs are higher than one might expect. Controlling them is an important management issue, particularly in dynamic network environments.
- The choice of vendor affects much more than initial acquisition costs. It can have a significant impact on such costs as the personnel costs associated with the routine operation and incremental change of a network.
- The effects of topology on network cost are profound. The average per-port costs of the centralized corporate networks studied were double those of the distributed networks studied.

Detailed findings appear in the chapters that follow.

4. Conclusions

The model represents a practical and effective tool for managers to use in identifying and analyzing network costs. It provides a basis for drawing cost comparisons of different networks within and between organizations. When used to support the analysis of network design and/or vendor alternatives, it encourages looking beyond the basic acquisition costs to consider a comprehensive set of potential cost factors.

While the model is relevant to the design and planning of new information systems, it applies as well to the identification and tracking of cost reduction opportunities within the existing infrastructure.

II. Introduction

1. Effective management of costs can be the key to successful management of the scarce I.S. resource.

Large corporations today are under continuing pressure to control current and future costs across the board, and the Information Services (I.S.) function is not exempt from the pressure. I.S. managers also realize that a serious endeavor at cost management can do more than merely satisfy near-term cost control objectives. Indeed, it can serve to free up funds for more important, mission-critical needs.

2. A prerequisite to strong I.S. cost management is an effective approach for measuring and evaluating costs.

Dr. Michael E. Treacy and Index Group, Inc.[1] developed and applied such an approach to identify and isolate the costs of owning and operating computer networks.[2]

The focus on networking is in recognition of the growing significance of networking in I.S. budgets and the scarcity of material available on assessing total computer network costs. In future work, the cost modeling approach developed here will be applied within the broader scope of overall information systems costs.

At the core of the approach is the framework (model) shown in Exhibit II-1, wherein costs are placed in the five "line item" categories of equipment, software, personnel, communications and facilities. Each of the cost items is then accumulated over three different phases in the network lifecycle. The lifecycle begins with the acquisition of the network, moves to routine operations and troubleshooting, and finally to the support of incremental changes to the network. The model is described in more detail in Chapter IV.

Exhibit II-1

Cost Of Network Ownership Model

	Acquisition	*Operation*	*Incremental Change*
Equipment			
Software			
Personnel			
Communications			
Facilities			

3. The researchers tested the cost model by applying it to a set of live subject networks to get answers to some specific questions.

The questions addressed in the analysis are:

(1) What are the total costs of network ownership?

(2) Where do the major costs lie?

(3) How do such issues as topology and vendor selection affect network costs?

The study of actual, operational networks made it possible for the researchers to obtain realistic answers to the questions and to validate the practicality and usefulness of the cost modeling approach.

Chapter III describes the approach taken by the team in selecting and analyzing the subject networks, and Chapter IV discusses the specifics of the cost-of-ownership model. Chapters V through IX discuss the results of 17 network case studies. Chapter X reviews the practical applications of the model, and Chapter XI presents the overall conclusions and recommendations of the study.

III. Study Methodology

1. To test the cost-of-ownership model and demonstrate its flexibility, the study team worked with it across three types of case study networks.

An important objective for the study team in testing the model was to demonstrate that it is flexible enough to handle a broad range of network types. To accomplish this, the team focused on three types of networks: corporate networks, multiple field offices networks and manufacturing site networks.

The study team defined *corporate networks* to be wide-area networks whose function is to connect a geographically dispersed set of workstations to one or more shared computers. All of the corporate networks studied had over 1,000 users. Each was characterized by significant remote communications, driven by such requirements as electronic mail, file transfer, shared applications and teleprocessing. With eleven case study subjects, corporate networks were the principal focus of the study.

The *multiple field offices networks* studied are also wide-area in character, but emphasize the dedicated connection of distributed, highly similar field locations to a single corporate computer center. Each field office has its own minicomputer-based computing capability, and the wide-area network is used to upload and download data from the corporate data center, to support pass-through application transactions on the corporate data center, and to participate in corporate-wide electronic mail. Four of these networks were examined in detail.

The *manufacturing site networks* studied were local area in emphasis. The purpose of these networks is to connect shop floor control and monitoring equipment and workstations to a plant host computer. These networks support production control/management applications as well as general-purpose applications such as electronic mail, word processing and project management. Two manufacturing site networks were included in the study.

2. Seventeen networks were included in the study.

Following initial discussions with approximately 100 large businesses and government agencies, the study team worked closely with interested organizations to examine 34 active computer networks, 17 of which were analyzed in detail and covered in this report. Industries represented in the sample include automotive, chemical, consumer products, defense, discrete manufacturing, electronics, engineering, insurance, publishing and utilities.

Each of the networks was chosen based on the following criteria:

- Adhered to the definition of a corporate network, multiple field offices network or manufacturing site network as described above
- Was based principally on the technology of Digital Equipment Corporation ("Digital") or International Business Machines Corporation ("IBM")
- Was accessible enough to the researchers to yield the necessary research information

IV. The Cost Model

1. To allow data to be gathered and analyzed in a structured and efficient manner, the study team developed a model of network costs.

The researchers found almost immediately that organizations have difficulty identifying the costs of owning and operating a corporate network. Companies can often produce a figure for the cost of the equipment, but they find it more difficult to identify and assess other costs specific to the computer network. For example, some organizations have difficulty separating the computer-related vs. voice-related elements in the various local and long-distance telephone bills.

Other organizations may not have a distinct sense of which human resources to attribute to networking. Even once a set of cost items is identified and agreed upon, it is unlikely that the financial systems will neatly isolate and accumulate the needed cost information. The study team needed a framework for identifying, analyzing and comparing network ownership costs.

The team developed a model for network costs that is both *comprehensive* and *categorical*. It is comprehensive in that it is designed to account for all of the costs. It is categorical in that it allows the costs to be categorized in a way that is relevant for management decisions.

2. The model maps key network cost components across the three lifecycle phases of Acquisition, Operation and Incremental Change.

The researchers identified five components of cost:

- Equipment
- Software

- Personnel

- Communications carriers

- Facilities (space and wiring)

The study team mapped each of the cost components across three categories, corresponding to lifecycle phases of a network. The resulting 3-by-5 matrix is shown in Exhibit IV-1 with examples of the types of costs that would appear in the cells.

Exhibit IV-1

Cost Of Network Ownership Model

Cells Contain Examples Of Contributors To Cost

	Acquisition	*Operation*	*Incremental Change*
Equipment	. EQUIPMENT PURCHASE	. MAINTENANCE	
Software	. SOFTWARE PURCHASE/ ONE-TIME LICENSE	. ANNUAL LICENSE . SOFTWARE MAINTENANCE	
Personnel	. PLANNING, DESIGN AND SELECTION . EQUIPMENT AND SOFTWARE INSTALLATION	. ROUTINE MONITORING AND OPERATION . NETWORK PROBLEM CORRECTION . USER LIAISON, ADMINISTRATION	. USER CHANGES - MOVES - ADDS - DELETES . SOFTWARE VERSION CHANGES
Communications	. INITIAL HOOKUP CHARGES	. MONTHLY TARIFF CHARGES	
Facilities	. FACILITIES DEVELOPMENT . WIRING COSTS	. SPACE EXPENSE	

Acquisition costs are those related to the initial planning, purchasing and installation of a network. Included are costs for equipment, the purchase of software, personnel to plan, design, and select the network, initial hookup charges of third-party telecommunications carriers, and wiring.

Operations costs represent the expense of operating, supporting and maintaining the network over a five-year period. This category includes the cost of maintaining the equipment and software, annual license fees for some software, personnel costs associated with network management (monitoring operations, correcting problems, working with users), tariff charges of third-party carriers, and the annual costs of physical space.

Incremental change costs are the costs, over a five-year period, of supporting routine, day-to-day changes to the network such as moves, adds, deletes and minor reconfigurations. Consistent with the emphasis on routine network changes, the researchers treated the costs of large-scale changes -- bringing up a new generation of technology, for example -- as acquisition-related costs rather than as incremental change costs.

Incremental change costs are closely related to the operations costs. Often analysts define operations costs to include the change costs implicitly. For the personnel cost component, however, the researchers believe that it is important to distinguish between the two categories. With most of the case study networks, the team was successful in identifying and separating the incremental change-related personnel activities from the purely operations-related activities. As portrayed in the exhibit, an attempt was not made to break out an incremental change portion for the other four cost components in the model (equipment, software, communications and facilities).

3. Standard-resource pricing enhanced the ability of the study team to group and compare networks and gather accurate data.

With the model developed, the next task for the study team was to populate the model with case study data. To accomplish this, the team developed a data-gathering approach that meets four important requirements:

- It enhances the cross-comparability of companies experiencing one or more unique factors such as special vendor pricing deals and fully-depreciated and/or obsolete network equipment.
- It eliminates local and regional variations in the cost of such resources as network personnel, and real estate.

- It yields accurate cost information by avoiding the pitfall of trying to search for expense and asset records stored in a diffuse set of financial systems potentially as far back in time as 20 years ago.
- It reflects current replacement costs and eliminates differences due to changing products and price structures.

The approach involves two steps. First, instead of asking interviewees directly about the costs of various network components and personnel over time, the researchers asked what resources were in place. For example, the team would ask:

- "How many controllers of what type do you have in place?" instead of "What was your expenditure for each controller in this network?"
- "How many, and what types of people support this network?" instead of "What was the full cost of each of the network people in your central group and the support people in the divisions?"

Second, after learning what resources were in place to support a network, the researchers then inferred cost through the use of standard resource price tables (see the examples in Appendices A and B). Vendor list prices were used for equipment and software. In cases where a piece of equipment or software was old and no longer had a meaningful list price, the price of the most nearly-comparable current product was substituted.

For personnel, the researchers used salary survey sources to develop a table of standard network-related job categories. Personnel costs were assumed to be twice base salary to cover overhead.

For communications carrier costs, representative AT&T and local operating company prices were applied to a range of leased-line bandwidths and distances.

For the facilities cost component, the researchers applied a standard acquisition cost per network port for terminal wiring: purchasing and running a wire to each terminal on the network. The wiring charge does not include interfaces, transceivers, controllers, etc., which are accounted for explicitly within the equipment cost component. Also included in the facilities component is a standard cost per square foot of space to house network-related equipment.

To further illustrate the modeling process, Appendices A and B demonstrate the cost analysis of two corporate networks.

4. To group and compare networks of inevitably varying sizes, the study team employed two cost normalization approaches.

The cost figures developed by the researchers for each network under study were of obvious interest to the corporation involved, but do not constitute a useful basis for grouping (averaging) or comparing costs across multiple networks.

The study team employed two normalization approaches in its analyses. The first approach was to portray each cell in the model as a *percentage* of the total network cost. The approach was very useful in determining the relative significance of the network cost elements across groups of networks.

The second strategy was to normalize the dollar costs based on the size of each network, yielding *cost per port* figures. In this analysis, all modeled costs for a network are divided by the number of ports on the network. Ports were chosen as the unit of measure to resolve the ambiguity of dial-up terminals: if a network has 10 dialup access modems and 100 users with dialup terminals, is it a 10-terminal network or a 100-terminal network. The study team wanted to reflect the maximum concurrent connect capability of each network, and would therefore answer "10." Correspondingly, the number of ports is defined as the sum of two items:

(1) the number of user or shop floor control devices directly connected to the network

(2) the number of dial-up access modems on the network

Cost-per-port analysis enabled the team to make direct comparisons of cost structure across groups of networks.

5. Groupings and cost comparisons of networks are meaningful only when bounds of analysis are both understood and applied consistently.

If successive network analyses are to be grouped and compared with each other on any reasonable basis, then each analysis must include, and exclude, the same cost items. For example, if electronic mail software were to be included in the costs of one network, the analyst should ensure that the costs of analogous software are included in any other networks being compared with the first.

In its analyses, the study team worked to ensure that consistent bounds of analysis were applied to all networks within each of the three major network types examined (corporate, multiple field offices and manufacturing site networks). This allowed the team to perform comparisons of sites within each network type, but not across network types.

V. Corporate Network Case Studies

1. Eleven corporate network case studies were modeled.

As their primary test of the power and utility of the model as a tool for understanding network cost structure, the researchers applied it to the set of corporate networks summarized in Exhibit V-1. Each of these networks is in existence to provide connectivity for thousands of workstations across a wide area to one or more computing resources. Nine of the networks are in the U.S., and two are in the U.K.

Four of the networks have a distributed computing topology in which users obtain most of their service from a local processor; the job of the network in these cases is to support access to remote processors when and as it is needed. The other seven networks are centralized. In these networks, the typical user obtains service from one or more remotely located computing centers via a leased-line telecommunications link.

Five networks are supported primarily by Digital hardware, and the other six are IBM-based. The networks typically support a mix of timesharing, electronic mail, file transfer and transaction processing applications, although one of the networks, identified as USC6 in the exhibit, was heavily-oriented to transaction processing.

2. The bounds of analysis include the elements in each network between but not including the shared processing resources and user work-stations.

A discussion of how the various resources in a network are modeled in cost-of-network-ownership analysis appears as part of Chapter IV. This section elaborates on the general guidelines as they apply to corporate networks specifically.

Exhibit V-1

Corporate Network Case Studies

Network Case Code	Number Of Ports	Topology	Principal Vendor	Network Changes Per Year
USC1	4,000	DISTRIBUTED	DIGITAL	720
USC2	3,800	CENTRALIZED	IBM	664
USC3	12,030	DISTRIBUTED	IBM	2,400
USC4	40,304	DISTRIBUTED	DIGITAL	16,400
USC5	6,237	DISTRIBUTED	DIGITAL	3,000
USC6	1,650	CENTRALIZED	DIGITAL	400
USC7	5,700	CENTRALIZED	IBM	2,280
USC8	45,000	CENTRALIZED	IBM	5,000
USC9	5,925	CENTRALIZED	IBM	600
UKC1	5,550	CENTRALIZED	DIGITAL	78
UKC2	1,155	CENTRALIZED	IBM	300

The following equipment is included: front-end processors and other remote communications controllers, switches, terminal controllers, multiplexers, DSU's and modems. For installations without full-function front-end processors, the study team assumed that 5 or 10 percent of the workload on each shared processor is network-related, depending on the sophistication of the terminal control equipment in use. Of course, if the exclusive function of a given computer is to route network traffic, then its entire cost is included in the analysis.

For software, the bounds of analysis include teleprocessing (TP) monitor and network control software (processor and controller-resident). For environments without specific teleprocessing monitor software -- typical VMS and VM installations, for example -- operating system software costs are included in the analysis.

The personnel categories included are systems programmers for the TP and network control software, network operations, maintenance and administration staff, and data communications management staff.

All telephone circuits used to interconnect the workstations and processors of a corporate network are included in the analysis. In situations where only a portion of a circuit is used for the corporate network under study, it is assumed to have a bandwidth equal to that portion.

Corporate network facilities costs are modeled as described in Chapter IV. For actual examples of corporate network case study modeling, the reader should refer to the case analyses in Appendices A and B.

3. For the corporate networks studied, the researchers found that on average only one-third of the 5-year ownership costs are related to the acquisition of the network, while nearly two-thirds of the costs are for operations and routine change.

The study team modeled the 5-year costs of the 11 corporate networks studied. Exhibit V-2 presents the average cost structure for the 11 networks. At the lower right-hand corner, the exhibit shows that the average cost per port over five years was $4,969, or about $1,000 per year. As explained above, this per-port figure is exclusive of the cost of shared computers and the cost of the workstation itself.

With their thousands of geographically distributed users, the physical magnitude of these networks is obvious. As one would expect, the average level of acquisition investment in the networks is significant at $1,791 per port multiplied by thousands of ports. The surprising observation is that the costs occurring *after* acquisition, the operations and incremental change costs, are almost twice as large, at $3,178 per port over five years. The implication is that were such a network to be acquired today, the "cost of purchase" figures at the bottom of a vendor's bid might represent only 36.1 percent of the average network's five-year cost.

4. Personnel costs account for 25 percent of 5-year costs of the corporate networks studied.

The figures at the right of the exhibit show, as would be expected for these networks, that the largest individual cost component is equipment. Combined with software, it accounts for 41.5 percent of the total cost over a five-year

Exhibit V-2

Corporate Network Cost Structure

Dollars Per Port Over a Five-Year Period

	Acquisition	*Operation*	*Incremental Change*	
Equipment	1,258 (25.3%)	413 (8.3%)		1,671 (33.6%)
Software	214 (4.3%)	179 (3.6%)		393 (7.9%)
Personnel	N/A	847 (17.0%)	397 (8.0%)	1,244 (25.0%)
Communications	58 (1.2%)	1,252 (25.2%)		1,310 (26.4%)
Facilities	261 (5.3%)	90 (1.8%)		351 (7.1%)
	1,791 (36.1%)	3,178 (63.9%)		4,969 (100%)

period. Communications line costs amount to 26.4 percent of costs, as would be expected for these wide-area networks.

Surprisingly, personnel costs at 25.0 percent almost match the communications costs, and that figure is exclusive of the personnel resources involved in planning, acquiring and installing the network. Thus personnel costs, arguably the most difficult to manage, represent a major network cost item.

As denoted by the "N/A" in the exhibit, the researchers could not identify the personnel costs associated with the acquisition phase for the corporate-wide networks studied. This is because the networks did not have a specific "acquisition" period in which the bulk of the network and technology was put in place; rather, the networks evolved and expanded gradually over many years.

At 7.1 percent, facilities costs were the least prominent.

5. It is useful to express the cost of routine network changes in terms of dollars per change request.

The exhibit shows a personnel cost for incremental change of $397 per port. This dollars-per-port figure is useful for examining the magnitude of the incremental change cost relative to the overall per-port network cost of $4,969. It is not very helpful, however, as a basis for expressing and comparing the unit cost of change across networks. This is because the cost of change-related activities on a network depends more on the actual number of changes made than on the number of ports in the network.

To meet the need for examination and comparison of unit change costs across networks, the researchers calculated a *dollars per change request* figure for each network in the study. The figure is arrived at by dividing the yearly personnel cost of supporting network change requests by the number of change requests processed during the year. For the eleven networks studied, the average cost per change request is $358.

6. The model is of use in contrasting the cost structures of networks on different continents.

To explore the applicability of the model to comparing the costs of networks on different continents, the researchers compared the average cost structure of the two European corporate networks studied with that of their U.S. counterparts. The small sample size does not provide a basis for generalizing the comparison, although it does provide a rich test case for demonstrating the power of the model.

To allow for a full comparison of cost structures between the U.S. and U.K. network samples, the researchers developed an independent, U.K.-specific set of price tables for modeling the U.K. networks. For analysis, the cost figures for the U.K. networks are converted into dollars using an exchange rate of $1.60 per pound sterling.

Exhibit V-3 shows a comparison of the U.S. and U.K. network cost structures. The costs shown are the five-year totals for each of the five resource components. Since both of the U.K. networks studied have centralized topologies, their averages are compared against the averages for the five centralized U.S. networks.

Exhibit V-3

U.S. vs. U.K. Centralized Corporate Network Costs

Dollars Per Port Over a Five-Year Period

	U.S.	*U.K.*
Equipment	1,736 (27.6%)	2,538 (41.4%)
Software	382 (6.1%)	672 (10.9%)
Personnel	1,558 (24.8%)	1,255 (20.5%)
Communications	2,257 (35.9%)	1,207 (19.7%)
Facilities	353 (5.6%)	459 (7.5%)
	6,286 (100.0%)	6,131 (100.0%)

The exhibit shows the total per-port costs to be quite comparable at $6,286 for the U.S. and $6,131 for the U.K. There are contrasts, however, among the individual cost components. Underlying the cost differences are two factors:

- Differences in the *prices* of resources
- Differences in the *efficiency of use* of the resources

Actual U.K. prices for equipment and software, depending on the item, were typically in the range of 20 to 70 percent higher than in the U.S. Roughly commensurate with this range, the exhibit shows an average equipment and software cost of $3,210 per port for the U.K. networks, versus $2,118 for the U.S. networks. For the networks in the sample, the price factor appears to explain the U.K.-U.S. difference in per-port equipment and software costs.

U.K. salaries were 25-50 percent lower than the corresponding U.S. salaries. The exhibit, however, shows only a 19 percent difference in per-port personnel costs, which averaged $1,255 in the U.K. networks studied versus $1,558 in the U.S. The beneficial effect of the salary differential on per-port costs in the U.K. therefore appears to be lessened somewhat by a more personnel-intensive network management strategy. This observation is not surprising in that the a telecommunications manager in the U.K.

would have a greater incentive to make an additional investment in (relatively inexpensive) personnel in order to gain more control over (high) equipment and software costs.

Communications costs average $1,207 per port in the U.K., versus $2,257 in the U.S. Despite important differences in the pricing of telecommunications services in the two countries, the principal cause for the cost differential lies in the larger differences in network geography. In the U.K., network span can be measured in tens and hundreds of miles; in the U.S, it is measured in hundreds and thousands of miles.

VI. Impact Of Topology On Cost

1. In testing the model on topology issues, the study team learned that the average per-port cost of the centralized corporate networks is more than double that of the distributed networks in the sample.

This chapter demonstrates the applicability of the cost-of-ownership model to issues of topology. The data from the eleven corporate network case studies is used to illustrate the power of the modeling technique. Attempts should not be made to generalize the results beyond the specific networks included in the sample.

Four of the corporate networks in the study have a distributed computing topology in which users obtain most of their service from a local processor. The job of the network in these cases is to support access to remote processors when and as it is needed. The other seven networks are centralized. In these networks, the typical user obtains service from one or more remotely located computing centers via a leased-line telecommunications link.

Exhibit VI-1 shows a comparison of the average distributed network costs versus the average centralized network costs. The figures shown are the five-year per-port costs for each of the five network resource components in the cost-of-ownership model. The exhibit shows that the average cost for the centralized networks is $6,242, versus $2,741 for the distributed networks.

2. As expected, communications line costs vary greatly between the distributed and centralized networks.

The figure of 26.4 percent presented in the previous chapter for corporate network communications line costs is actually an average value for what proves to be a bimodal cost distribution. As Exhibit VI-1 shows, line costs

Exhibit VI-1

Corporate Network Cost Breakdown By Topology

Dollars Per Port Over a Five-Year Period

	Distributed		*Centralized*	
Equipment	1,157	(42.2%)	1,965	(31.5%)
Software	267	(9.7%)	465	(7.5%)
Personnel	846	(30.9%)	1,471	(23.6%)
Communications	179	(6.5%)	1,957	(31.3%)
Facilities	292	(10.7%)	384	(6.1%)
	2,741	(100%)	6,242	(100%)

represent only 6.5 percent of total costs on average for the distributed networks, while they amount to 31.3 percent for the centralized networks.

The differential of $1,778 per port for line costs ($1,957 versus $179) is the largest single contributor to the $3,501 overall difference between average distributed and centralized network costs. The finding is understandable given that in the centralized networks, adequate communications circuit capacity must be in place to support all of the workstation I/O. In the distributed networks, most of the terminal I/O passes between workstations and local processors.

Because of the added investment in communications processors, modems, multiplexers, DSU's, etc., and the extra need on the part of the centralized networks for redundancy at the computer centers, average equipment and software costs differed by almost $1000 per port.

VII. Impact Of Vendor On Cost

1. This chapter provides an example of how the model can be applied to explore the impact of vendor choice on network cost.

To demonstrate the power and usefulness of the cost-of-ownership model in looking at vendor-related cost issues, the researchers grouped the corporate networks according to primary vendor -- Digital or IBM -- and applied the model to compare the groups. The sample of eleven networks provides an interesting test of the model, but should not be relied upon as the basis for drawing general conclusions on relative vendor cost.

The results of the test are shown in Exhibit VII-1, which compares the average costs of the five Digital-based corporate networks in the study against the costs of the six IBM-based corporate networks.

2. The impact of vendor choice on network cost goes well beyond the cost of the equipment and software.

Despite the fact that the equipment and software costs shown in the exhibit correspond quite closely between the two vendors, a bottom-line cost differential exists of $1,380 per port, rooted largely in the area of personnel. The disparity underscores an important point, which is that companies involved in vendor selection decisions need to apply the full model to their specific situations and look beyond the basic equipment and software cost issues.

Exhibit VII-1

Digital vs. IBM Corporate Network Costs

Dollars Per Port Over a Five-Year Period

	Digital		IBM	
Equipment	1,625	(38.6%)	1,710	(30.5%)
Software	327	(7.7%)	448	(8.0%)
Personnel	798	(18.9%)	1,616	(28.9%)
Communications	1,155	(27.4%)	1,439	(25.7%)
Facilities	311	(7.4%)	383	(6.9%)
	4,216	(100%)	5,596	(100%)

3. <u>Among the corporate networks on which the model was tested, average personnel costs were lower for the Digital networks than for the IBM networks.</u>

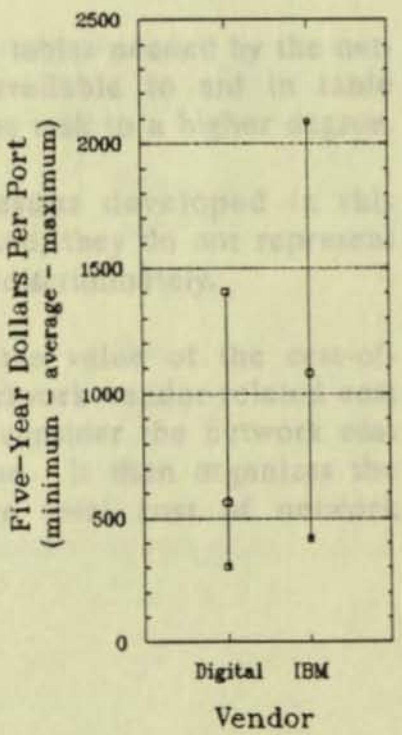
Exhibit VII-1 shows that personnel costs for the Digital-based networks studied averaged $818 less per port than for the IBM networks in the study. As explained in Chapter IV, the researchers analyzed personnel costs in two components, operations and incremental change.

Exhibit VII-2 examines the issue of operations personnel cost by plotting the average, maximum and minimum operations personnel costs encountered for each vendor in the networks studied. The costs are expressed in dollars per port over five years. The plot reveals a significant difference between the two vendors' average

Exhibit VII-2

Operations

Personnel Cost

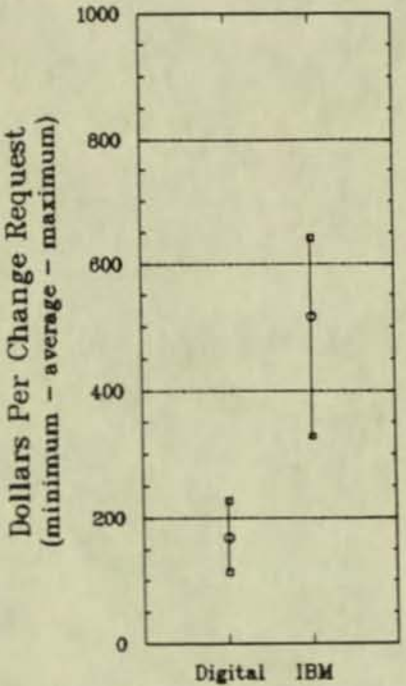


costs, although the sample ranges overlap significantly.

To examine the personnel costs for incremental change, the team calculated a "cost-per-change-request" figure for each corporate network. The figure was arrived at by dividing the human resource costs associated with incremental change in each network by the number of change requests processed. As illustrated in Exhibit VII-3, the change costs in the Digital cases are consistently lower than in the IBM cases, with no overlap at all between the cost sample ranges. The average cost per network change is $168 for the Digital networks in the study. The figure for the IBM networks is $516 -- three times as much.

Exhibit VII-3

Incremental Change Personnel Cost



The cost per change request fell below $400 in only one of the IBM networks studied. In this network, the customer had written an extensive, in-house application to aid the systems staff in translating network change requests into the extensive definition tables needed by the network. While vendor-provided software was available to aid in table generation, the in-house application automated the task to a higher degree.

The reader is reminded that although the results developed in this illustration are relevant for the 11 networks studied, they do not represent general conclusions and should not be applied indiscriminately.

The illustration presented above demonstrates the value of the cost-of-network-ownership model as a tool for use in network vendor-related cost analyses. The model encourages the analyst to consider the network cost components individually over each lifecycle phase. It then organizes the costs and consolidates them to determine the total cost of network ownership.

VIII. Multiple Field Offices Networks

1. <u>This chapter demonstrates the application of the model to multiple field offices networks.</u>

The researchers tested the cost-of-ownership model on a group of four multiple field offices networks of varying sizes. In each of these networks, a number of nearly identical field offices is tied into a corporation's central computer by leased lines. Each field office has its own computer, and most processing is done locally. The data flowing between the field offices and the central computer is related mainly to batch file transfer and terminal pass-through transactions.

Exhibit VIII-1

Multiple Field Offices Case Studies

Network Case Code	Number Of Ports	Number Of Offices	Users Per Office	Principal Vendor
F1	2,000	50	40	DIGITAL
F2	408	34	12	IBM
F3	98	7	14	DIGITAL
F4	280	7	40	IBM

As shown in Exhibit VIII-1, the total number of ports varied from 98 to 2,000. The number of field offices per network varied from 7 to 50, and both "large" (40 people) and "small" (12-14 people) field offices were represented.

2. <u>The bounds of analysis used for the multiple field offices cases were very different from what was used in the corporate network case studies.</u>

In defining the bounds of analysis for the multiple field offices networks, the researchers were sensitive to the fact that some of the networks studied were actually pieces of a much larger overall corporate network. To demonstrate the power of the model to help in segmenting networks for maximum comparability, the only equipment and software that was counted at the central data center was the set of modems and/or gateways in place to connect to the field offices. Centralized network personnel were counted only to the extent that they are involved with the field offices component of the overall network.

In addition, the researchers decided to include the entire equipment complement at the field offices -- computer, workstations, peripherals, as well as communications equipment. For personnel at the field offices, the researchers did not try to distinguish between time spent (typically a fraction of a single person) on "system support" versus "network support." They chose this approach due to the difficulty inherent in trying to apportion the resources of a small microcomputer or minicomputer installation between "network processing" and "application processing" on any reasonable basis. As in the corporate network scenario, application systems and personnel were not included within the bounds of analysis.

Because the bounds of analysis are different and the networks are different, no attempt should be made to compare these results to the corporate network results. Instead, these different bounds serve to illustrate the flexibility of the cost-of-ownership model.

3. <u>The new bounds of analysis lead to very different cost-per-port figures from what was experienced in the corporate network scenario.</u>

Exhibit VIII-2 shows that the average cost per port is over $30,000, in contrast to less than $10,000 for the corporate networks discussed previously. This difference is due chiefly to the inclusion of the field office computing resources and support staff in the analysis. The magnitude of the difference in cost underscores the importance of not trying relate the cost figures from two networks without using identical bounds of analysis in the modeling process.

Exhibit VIII-2

Multiple Field Offices Network Cost Structure

Dollars Per Port Over a Five-Year Period

	Acquisition	*Operation*	*Incremental Change*	
Equipment	7,594 (23.7%)	5,667 (17.7%)		13,261 (41.4%)
Software	1,308 (4.1%)	3,779 (11.8%)		5,087 (15.9%)
Personnel	499 (1.6%)	10,110 (31.5%)		10,609 (33.1%)
Communications	265 (0.8%)	1,493 (4.6%)		1,758 (5.4%)
Facilities	256 (0.8%)	1,098 (3.4%)		1,354 (4.2%)
	9,922 (31.0%)	22,147 (69.0%)		32,069 (100%)

Looking at the cost structure of these networks as they were modeled by the study team, the first observation is that most of the cost (two-thirds) is operational as opposed to acquisition-related, as was the case in the corporate network scenario. This further underscores the general importance of looking beyond the basic acquisition costs when considering network planning and design alternatives.

Equipment and software costs are very high, accounting for over $18,000 of the $32,000 total cost per port, reflecting the chosen bounds of analysis. At $10,609 per port, personnel costs are very significant and certainly represent an important management consideration. Communications costs are of significant magnitude ($1,758 per port), but are diminutive in proportion to the equipment/software and human resources costs.

The study team was unable to identify routine incremental change costs for the multiple field office networks, as was done with the corporate networks. The networks were all quite static, apart from initial installation and the phase-in of additional offices.

The researchers were able to quantify the "personnel acquisition" cell of the model for multiple field offices network costs. Unlike the corporate networks studied, multiple field offices networks are typically re-done in their entirety every five to ten years. In three of the four sites studied, the latest redesign project was recent enough to be modeled reliably.

4. **As a final test of the model on multiple field offices networks, the researchers applied it to examine the effect of field office size on the costs of network ownership.**

An analysis based on size of field office, shown in Exhibit VIII-3 demonstrates an interesting application of the model. Due primarily to higher per-port personnel and communications line costs, the networks with small offices are $8,500 more expensive per port than those with large-office counterparts. The study team accounts for the $7,800 difference in per-port personnel costs by noting that the cost of the person tending to a 40-person computer is spread among a greater number of network ports, and that this person does not necessarily face a proportionately larger set of duties. The $1,400 difference in per-port leased line costs may be attributable to more efficient line utilization by larger offices.

Exhibit VIII-3

Cost Structure for Networks With Large Vs. Small Field Offices

Dollars Per Port Over a Five-Year Period

	Small Offices		*Large Offices*	
Equipment	11,431	(31.5%)	15,089	(54.2%)
Software	7,341	(20.2%)	2,833	(10.2%)
Personnel	14,525	(40.0%)	6,694	(24.1%)
Communications	2,472	(6.8%)	1,043	(3.7%)
Facilities	545	(1.5%)	2,164	(7.8%)
	36,314	(100%)	27,823	(100%)

Another observation arises out of the exhibit: While total equipment and software costs amount to roughly the same figure in both the small- and large-office cases ($18,000 per port), the relative share of equipment vs. software varies dramatically. This illustrates a potential difference in the vendors' equipment/software bundling strategies for the differently-sized systems.

The sample size of four networks does not represent a large-enough foundation for drawing general conclusions and achieving certainty as to the causal factors behind each of the observations made. The example analyses are effective, however, in demonstrating the applicability of the model to key cost questions in specific multiple field offices network situations.

IX. Manufacturing Site Networks

1. This chapter illustrates the application of the model to manufacturing site networks.

In contrast to the networks discussed to this point, the manufacturing site networks examined in the study are primarily local-area networks. In support of manufacturing monitoring and control applications, they connect plant floor workstations and device controllers to local processors and connect those processors to a shared corporate computer. As shown in Exhibit IX-1, the two networks are quite parallel in terms of their size and volatility. In addition, the two networks perform comparable types of functions in the same industry, the automotive industry.

Exhibit IX-1

Manufacturing Site Network Case Studies

Network Case Code	Number Of Ports	Network Changes Per Year	Principal Vendor
M1	232	250	DIGITAL
M2	260	233	IBM

The processors and terminal servers on the M1 network are tied together via a local-area network. Shop floor device controllers are wired directly to the processors. Gateways on the local-area network are tied via leased lines to the company's corporate data center.

Terminal controllers and local processors on the M2 network are all routed via telephone lines to the company's nearby corporate computer center. As in the M1 network, device controllers are wired directly to the local processors.

2. The bounds of analysis used for the manufacturing site networks are similar in concept to those used for modeling the corporate networks.

The same types of resources were counted for the manufacturing site networks as were counted for the corporate networks discussed earlier in this report, but with a much more focused scope: the manufacturing plant and its connections to the corporate network. Therefore, resources outside of the plant -- controllers, modems, lines, software, people, lines, facilities -- were included only if they directly support the network within the plant or the connection of the plant network to the corporate data center.

As with the corporate networks, all the resources between but not including the central computer and the end devices (device controllers, workstations, printers) were counted. Local processors were counted to the extent that they participated in the transmission of information to and from the plant floor devices and terminals; the allocation tended to be quite large for many of the local processors.

3. In testing the model on the two manufacturing networks, the chief observation relates to the significance of personnel costs.

Exhibit IX-2 reveals a substantial proportion for personnel cost -- over 50 percent. These network environments are very turbulent. Manufacturing networks are undergoing constant maintenance and troubleshooting as the manufacturing process is adapted and tuned; as these dynamics increase, so do the personnel costs.

A second observation, which relates in large measure to the significance of the ongoing personnel costs, is that 80 percent of the five-year costs of the two networks are incurred after acquisition. This is the largest proportion found within any of the three networking scenarios on which the model was tested, even given that more equipment and software per port is being counted in this networking scenario than in the corporate network scenario.

The analysis in this example underscores again the magnitude of the operational costs of a network, particularly the personnel costs, relative to the acquisition costs.

Exhibit IX-2

Manufacturing Site Network Cost Structure

Dollars Per Port Over a Five-Year Period

	Acquisition	*Operation*	*Incremental Change*	
Equipment	1,232 (14.6%)	964 (11.4%)		2,196 (26.0%)
Software	222 (2.6%)	1,323 (15.6%)		1,545 (18.2%)
Personnel	N/A	3,114 (36.8%)	1,181 (13.9%)	4,295 (50.7%)
Communications	4 (0.0%)	97 (1.1%)		101 (1.1%)
Facilities	250 (3.0%)	81 (1.0%)		331 (4.0%)
	1,708 (20.2%)	6,760 (79.8%)		8,468 (100%)

4. **The incremental change costs revealed by the model for the manufacturing site networks mirror closely the costs observed for corporate networks.**

The personnel costs for incremental change in the manufacturing site networks were computed using exactly the same criteria and bounds of analysis as were used for the corporate networks. For further validation of the model, the study team calculated the cost per change request for each of the two manufacturing site networks.

Upon performing these calculations, the researchers discovered that the resulting costs fell squarely within the cost-of-change ranges shown in Exhibit VII-3 for each vendor. The results seem intuitively correct since although the manufacturing networks are local in scope, the vendor network architectures and change processing procedures used are the same as with the corporate networks studied.

The results of applying the model to the two manufacturing sites were both enlightening and intuitively appealing to the researchers. However, it is important to bear in mind that the figures should not be broadly generalized. Their chief relevance is to the specific networks studied.

X. Putting the Model To Use

1. An important application of the cost-of-network-ownership model lies in analyzing and managing in-house network costs.

Based on the illustrations in this report, it is clear that the model is an effective tool for gathering network costs together in one place and analyzing them. Once a manager applies the model to in-house network costs, the most significant candidates for management attention can be identified and targeted.

In many organizations, for example, the management of telecommunications carrier lines and costs has reached a very high level of sophistication (and investment). In many of the networks studied, such as the distributed corporate networks, the line costs being managed in such an attentive fashion are far from being the most significant costs associated with the network. Personnel costs may be much greater, along with equipment costs. By modeling network costs comprehensively and categorically, the results can serve to focus management on its most important cost-saving opportunities.

An important element of any ongoing cost management strategy is the tracking of progress over time. Correspondingly, the use of the model in analyzing costs needs to be an ongoing activity.

As detailed in Chapter IV, a resource-based data-gathering approach was used by the study team to populate the model with network costs. In applying the model to the task of ongoing cost management, a company may wish to develop price tables specific to its own experiences. This step reduces the degree of comparability for structural comparisons with networks in other organizations, but will decrease the variances between modeled costs and actual costs. Where desired, and where the financial data is routinely available, a network manager may choose to substitute actual cost data in place of one or more resource/table-based cost figures.

2. The model is valid for grouping and comparing networks.

In a number of situations, it is useful for managers to compare the costs of two or more networks in much the same way as has been demonstrated in the previous chapters. Such comparisons are useful for a variety of purposes:

- Competitive comparison of cost structure with networks in other companies
- Comparison with published industry averages
- Comparative analysis with other networks within the same company
- Comparison of alternative architectures and topologies
- Evaluation of alternative network vendors

The model and methodology developed here add value to the comparisons in three ways:

- Consideration of all the costs of a network is encouraged, across all of the lifecycle phases.
- The structural comparability of the networks under comparison is improved.
- With resource-based data gathering and standard pricing, the accuracy and efficiency of the cost modeling are enhanced. This benefit is especially valuable in circumstances where hard financial data is simply not available, as would usually be the case for a competitor's network. Therefore, this model can form the basis for intelligence gathering on the cost structure of a competitor.

3. In making network comparisons, several forms of cost normalization are of benefit.

In applying the model, the researchers populated it with four types of cost analysis and normalization:

- Absolute dollar cost figures
- Relative cost percentages
- Dollars per network port
- Dollars per network change

For monitoring and managing in-house network costs, the first two types of analyses are of the most relevance. For comparisons across networks, all four types of analysis are important and should be considered.

4. To develop a complete comparison of two or more networks, it is important to examine additional factors beyond cost.

When two networks are compared and a cost difference is revealed, an important question to ask is "are all other things about the two networks equal?" The answer to the question will have a strong bearing on how a manager would be inclined to interpret the results of the cost comparison. In fact, "all other things" may be so unequal that a manager may be forced to discard the cost analysis completely.

The study team identified four areas for characterizing potential subject networks and their general comparability:

- The functionality of the network (How capable is the network of providing services today?). Examples of this are the available bandwidth, number of available network functions, availability, connectivity, etc.
- The network's flexibility (How gracefully can it evolve toward tomorrow's needs?). Examples would include the types and number of transmission options, the stability of the network during large deployment or application changes, adherence to standards, evolving applications support, etc.

- Its manageability (How easy -- or difficult -- is it to monitor, maintain and assure the security of the network?). Examples of manageability issues are security, maintainability, management of moves, adds and changes, etc.

- The affordability (cost) of the network. The nature of this area has been explored in the preceding chapters.

Finally, the team recognized that these characteristics can be interpreted quite differently if the management perspectives of the network under consideration are not comparable. For example, "functionality" to the manager of a large wide-area network is likely to connote issues such as bandwidth and availability. To the manager of a campus-level network, it is more likely to connote such issues as connectivity and multi-vendor attach. For someone concerned with a small local-area network, it may mean simply to the level of capability on the desk.

If the networks being grouped together in a cost comparison appear to differ significantly from each other along one of these dimensions, then the outcome of the cost comparison is not directly useful. The manager will have to build on results of the cost analysis to take into account the differences encountered in functionality, flexibility, manageability and management perspective.

XI. Conclusion

1. In applying the cost-of-network-ownership model to representative corporate, multiple field offices and manufacturing site networks, the study team made four key observations.

The first observation is that the major costs appear *after* the acquisition of a network. Justification of a network decision based solely on the initial acquisition costs represents an incomplete analysis and could be misleading.

Second, personnel costs can represent a significant portion of total network costs (as much as 50 percent of costs over five years). They represent an important management issue, particularly in dynamic network environments.

Third, vendor selection affects more than initial equipment costs. It can have a significant impact on other costs, such as the personnel costs associated with the routine operation and incremental change of a network.

Fourth, the effects of topology on network cost are profound. The per-port costs of the centralized corporate networks studied averaged double those of the distributed networks studied. This is an important finding, and may suggest that a distributed network configuration is better-suited to today's dynamic applications environment than the traditional, centralized network architecture.

In many cases, corporate networks were originally designed in a hierarchical fashion to support operational and transaction processing requirements. As time goes by and needs shift more in the direction of end-user applications and inter-organizational computing, these hierarchical networks may no longer be cost-effective or adequately flexible. The major vendors recognize this and now offer a broad range of distributed solutions.

2. Based on this work, the study team suggests that network owners take pause and assess carefully the total costs of owning and operating their networks.

The researchers offer four recommendations in regard to the management of network costs:

- In comparing network vendors, look beyond the initial equipment costs; these do not necessarily reveal the true differences between vendors in the cost of network ownership.

- Beware as well of designing networks strictly to minimize communications carrier costs -- you may be addressing the wrong issue. In fact, one of the network managers in the study made a decision to invest in excess line capacity as a strategy for controlling the personnel costs associated with performance monitoring and tuning.

- Apply the model of network ownership costs to help guide your network planning and investment decisions. The model places *all* the costs associated with network ownership, including operation over an extended period, on a single, simple grid.

- Use the model as a tool for managing the costs of operating and changing your existing networks. Not only does the model identify networking costs in general, it also highlights the large and important costs that warrant special management emphasis.

XII. Notes

1. The Index Group consultants on the study team were *Adam D. Crescenzi*, Officer-In-Charge; *Alexander E. Nedzel*, Project Leader; *Tauno J. Metsisto; Robert N. Barrett; Craig A. Bickel; Algis S. Leveckis; Michael A. Petro*.

2. The study was conducted during the period May 1987 to August 1988.

Appendices

Appendix A

Case Analysis: Consolidated Manufacturing, Inc.

This appendix presents a detailed cost analysis of the Consolidated Manufacturing corporate network.

The 6,237-port Digital-based network has a distributed topology, and includes six major processing locations. All links are fully redundant.

The network's principal workloads are electronic mail and file transfer. It operates 24 hours per day, 7 days per week. The availability goal is 99%-plus. The network is highly volatile, supporting 3,000 change requests per year.

Hardware planning, system software and PC support are managed centrally. The local sites have autonomous control of their computer operations and user training.

The organization of the spreadsheet is straightforward. Summary tabulations appear first, followed by a section for each of the five network cost components in the cost-of-network-ownership model.

CONSOLIDATED MANUFACTURING, INC. - USC5 COST SUMMARY 20-Dec-88

	5-YEAR TOTAL COST ($000)				5-YEAR COST PER PORT (DOLLARS)				$ PER CHANGE
	ACQUIRE	OPERATION	CHANGE	TOTALS	ACQUIRE	OPERATION	CHANGE	TOTALS	CHANGE
EQUIPMENT	5,148	2,549		7,698	825	409	0	1,234	0
SOFTWARE	189	215		404	30	34	0	65	0
HUMAN RESOURCE		2,173	3,410	5,583	0	348	547	895	227
COMMUNICATIONS	19	1,494		1,514	3	240	0	243	0
FACILITIES	1,559	484		2,043	250	78	0	328	0
TOTALS	6,916	6,915	3,410	17,241	1,109	1,109	547	2,764	227

	5-YEAR TOTAL COST ($000)				5-YEAR COST PER PORT (DOLLARS)				$ PER CHANGE
	ACQUIRE	OPERATION	CHANGE	TOTALS	ACQUIRE	OPERATION	CHANGE	TOTALS	CHANGE
EQUIP/SFTWR *	5,338	2,764	0	8,102	856	443	0	1,299	0
HUMAN RESOURCE	0	2,173	3,410	5,583	0	348	547	895	227
COMMUNICATIONS	19	1,494	0	1,514	3	240	0	243	0
FACILITIES	1,559	484	0	2,043	250	78	0	328	0
TOTALS	6,916	6,915	3,410	17,241	1,109	1,109	547	2,764	227

* Equipment and software costs are combined to mask the effects of vendor bundling strategies.

CONSOLIDATED MANUFACTURING, INC. - USC5 20-Dec-88

SUMMARY EQUIPMENT AND SOFTWARE

	EQUIPMENT DOLLARS		SOFTWARE DOLLARS	
	Acquire	Annual Operations	Acquire	Annual Operations
LOCATION1	487,613	45,973	49,478	9,550
LOCATION2	169,086	17,120	10,785	2,968
LOCATION3	238,572	25,432	11,885	2,968
LOCATION4	897,520	93,096	40,858	9,550
LOCATION5	751,199	70,012	30,073	6,582
LOCATION6	2,604,469	258,193	46,070	11,379
	5,148,459	509,826	189,148	42,995

CONSOLIDATED MANUFACTURING, INC. - USC5 LOCATION1 EQUIPMENT 20-Dec-88

NETWORK CENTRAL

Acronym	Quantity	Description	Percent For Net.	Unit Price	Extended Price	Tot. Net One Time	Monthly Maint.	Tot. Net Ann. Cost	Comments
865CD-AP	1	VAX 8650 CLUSTER	5	554,400	554,400	27,720	1,851	1,111	As is today in the corporate office
851BC-AE	1	VAX 8530	5	342,300	342,300	17,115	1,243	746	Replaces 2 VAX 785's at corporate office
835BB-AE	1	VAX 8350	5	129,150	129,150	6,458	559	335	Replaces 785 and 750 for Development Group
DELUA-M	1	ETHNT ADPTR/UNIBUS	100	4,354	4,354	4,354	33	396	
HSC70	1	HIER. STOR. CNTLR.	0	58,765	58,765	0	220	0	For Development Group as is today
HSC50	3	HIER. STOR. CNTLR.	0	40,360	121,080	0	155	0	Two for Corporate office and one for Development Group
RA81-AA	14	DISK DRIVE	0	17,640	246,960	0	95	0	Ten for Corporate office and four for Development Group
RA60-A	2	RMVBL. DISK DRIVE	0	20,340	40,680	0	105	0	For Development Group
TU79-AB	3	CLUSTER TAPE DRIVE	0	54,705	164,115	0	322	0	Replaces TU77 and TU78's at Dev't Group and Corp office
TA79-AF	3	SLAVE TAPE DRIVE	0	29,400	88,200	0	196	0	Replaces TA78's - 2 at Corporate and 1 at Dev't Group
VT220-F2	518	TERMINAL	0	260	134,680	0	12	0	Includes terminals, PC's and printers and 28 data services
DSRVB-AA	74	DS 200 TERM. SERVER	100	3,806	281,644	281,644	37	32,856	Configured at seven out of eight ports used
DECSA-EA	2	ETHERNET ROUTER	100	15,451	30,902	30,902	152	3,648	Redundant configuration
DCSAX-LB	10	56KB CARD FOR DECSA	100	744	7,440	7,440	11	1,320	Redundant configuration
DELNI	1	DELNI LOCAL CONNECT	100	1,444	1,444	1,444	10	120	For routers as is today
H4000	78	TRANCEIVER	100	315	24,570	24,570	4	3,744	
BNE3H-40	78	TRANCEIVER CABLE 40M	100	625	48,750	48,750	0	0	
BNE2A-ME	2	500M ETHERNET CABLE	100	7,035	14,070	14,070	0	0	One for Corporate office, one for Development Group
12-19816-01	4	ETHERNET TERMINATOR	100	22	88	88	0	0	
DSU-56K	6	CODEX 56KB DSU	100	1,095	6,570	6,570	0	0	Five lines for the star network + one to Dev't Group
DF126-AA	15	DEC 2400 BAUD MODEM	100	1,050	15,750	15,750	12	2,160	Remote modem pool at Corporate office
DUP11	1	HOST 56KB LN. CARD	100	2,516	2,516	2,516	13	156	Inboard 56KB line card for Dev't Group
DHU11	1	16LN. ASYNC CNTLR.	100	4,955	4,955	4,955	55	660	For controlling asynchronous modems
		TOTAL EQUIPMENT COSTS				494,346		47,252	
LESS ALLOWANCES FOR SOFTWARE									
QK001-UZ		(1)8600,8650 VMS	5	28,875	(28,875)	(1,444)	525	(315)	Costs are included under software
Q9001-UZ		(1)8500,8530 VMS	5	38,115	(38,115)	(1,906)	525	(315)	
Q7001-UZ		(1)83XX VMS	5	26,250	(26,250)	(1,313)	466	(280)	
QKD05-UZ		(1)86XX DECNET VAX	5	15,803	(15,803)	(790)	213	(128)	
Q9D05-UZ		(1)8500/8530 DECNET VAX	5	15,141	(15,141)	(757)	213	(128)	
Q7D05-UZ		(1)83XX DECNET VAX	5	10,472	(10,472)	(524)	189	(113)	
		NET TOTAL FOR LOCATION				487,613		45,973	

CONSOLIDATED MANUFACTURING, INC. - USC5 LOCATION2 EQUIPMENT 20-Dec-88

EAST COAST

Acronym	Quantity	Description	Percent For Net.	Unit Price	Extended Price	Tot. Net One Time	Monthly Maint.	Total Ann. Maint	Comments
825BB-AE	1	VAX 8250	5	97,650	97,650	4,883	469	281	Replaces VAX 785
RA81-AA	3	DISK DRIVE	0	17,640	52,920	0	95	0	
TA79-BF	1	MASTER TAPE DRIVE	0	59,500	59,500	0	0	0	Same replacement as above
VT220-F2	200	TERMINAL	0	260	52,000	0	12	0	Estimated CRT's, PC's and printers
DSRVB-AA	29	DS 200 TERM. SERVER	100	3,806	110,374	110,374	37	12,876	Configured same as above
H4000	30	TRANCEIVER	100	315	9,450	9,450	4	1,440	
BNE3H-20	30	TRANCEIVER CABLE 20M	100	314	9,420	9,420	0	0	
BNE2A-ME	1	500M ETHERNET CABLE	100	7,035	7,035	7,035	0	0	
12-19816-01	2	ETHERNET TERMINATOR	100	22	44	44	0	0	
DSU-56K	1	CODEX 56KB DSU	100	1,095	1,095	1,095	0	0	
DF126-AA	10	DEC 2400 BAUD MODEM	100	1,050	10,500	10,500	12	1,440	Open modem pool
DF129-AA	1	DEC 9.6 KB MODEM	100	2,573	2,573	2,573	14	168	
DMR11	1	HOST 9.6 LINE CARD	100	7,758	7,758	7,758	41	492	Inboard 9.6KB line card
DUP11	1	HOST 56KB LN. CARD	100	2,516	2,516	2,516	13	156	Inboard 56KB line card
DHU11	1	16LN. ASYNC CNTLR.	100	4,955	4,955	4,955	55	660	For controlling asynchronous modems
	TOTAL EQUIPMENT COSTS					170,603		17,513	
LESS ALLOWANCES FOR SOFTWARE									
Q5001-UZ	(1)	82XX VMS	5	21,000	(21,000)	(1,050)	466	(280)	
Q5D05-UZ	(1)	82XX DECNET VAX	5	9,335	(9,335)	(467)	189	(113)	
	NET TOTAL FOR LOCATION					169,086		17,120	

CONSOLIDATED MANUFACTURING, INC. - USC5 LOCATION3 EQUIPMENT 20-Dec-88

MIDWEST MANUFACTURER

Acronym	Quantity	Description	Percent For Net.	Unit Price	Extended Price	Tot. Net One Time	Monthly Maint.	Total Ann. Maint	Comments
825BB-AE	1	VAX 8250	5	97,650	97,650	4,883	469	281	Replaces VAX785
DELUA-M	1	ETHNT ADPTR/UNIBUS	5	4,354	4,354	218	33	20	
RA81-AA	4	DISK DRIVE	0	17,640	70,560	0	95	0	
TA79-BF	1	MASTER TAPE DRIVE	0	59,500	59,500	0	0	0	Same replacement as above
VT220-F2	298	TERMINAL	0	260	77,480	0	12	0	CRT's, PC's and printers
DSRVB-AA	43	DS 200 TERM. SERVER	100	3,806	163,658	163,658	37	19,092	
DECSA-EA	1	ETHERNET ROUTER	100	15,451	15,451	15,451	152	1,824	Includes one DCSAX-LA
DCSAX-LB	1	56KB CARD FOR DECSA	100	744	744	744	11	132	
H4000	46	TRANCEIVER	100	315	14,490	14,490	4	2,208	
BNE3H-20	46	TRANCEIVER CABLE 20M	100	314	14,444	14,444	0	0	
BNE2A-ME	1	500M ETHERNET CABLE	100	7,035	7,035	7,035	0	0	
12-19816-01	2	ETHERNET TERMINATOR	100	22	44	44	0	0	
DSU-56K	1	CODEX 56KB DSU	100	1,095	1,095	1,095	0	0	
DF129-AA	1	DEC 9.6 KB MODEM	100	2,573	2,573	2,573	14	168	
DF126-AA	10	DEC 2400 BAUD MODEM	100	1,050	10,500	10,500	12	1,440	
DHU11	1	16LN. ASYNC CNTLR.	100	4,955	4,955	4,955	55	660	For controlling asynchronous modems
		TOTAL EQUIPMENT COSTS				240,089		25,825	
LESS ALLOWANCES FOR SOFTWARE									
Q5001-UZ		(1)82XX VMS	5	21,000	(21,000)	(1,050)	466	(280)	
Q5D05-UZ		(1)82XX DECNET VAX	5	9,335	(9,335)	(467)	189	(113)	
		NET TOTAL FOR LOCATION				238,572		25,432	

CONSOLIDATED MANUFACTURING, INC. - USC5 LOCATION4 EQUIPMENT 20-Dec-88

WEST COAST

Acronym	Quantity	Description	Percent For Net.	Unit Price	Extended Price	Tot. Net One Time	Monthly Maint.	Total Ann. Maint	Comments
865CD-AP	1	VAX 8650 CLUSTER	5	554,400	554,400	27,720	1,851	1,111	
865XB-AE	1	VAX 8650/NO DECNET	5	522,375	522,375	26,119	1,778	1,067	Replaces three VAX 785's
825BB-AE	1	VAX 8250	5	97,650	97,650	4,883	469	281	Replaces VAX 785 at LOCATION4-CN
DELUA-M	2	ETHNT ADPTR/UNIBUS	100	4,354	8,708	8,708	33	792	
HSC70	2	HIER. STOR. CNTLR.	0	58,765	117,530	0	220	0	
RA81-AA	14	DISK DRIVE	0	17,640	246,960	0	95	0	
TA79-BF	3	MASTER TAPE DRIVE	0	59,500	178,500	0	0	0	Same replacement as above
VT220-F2	1,204	TERMINAL	0	260	313,040	0	12	0	CRT's etc. plus 40 devices for Data Services
DSRVB-AA	172	DS 200 TERM. SERVER	100	3,806	654,632	654,632	37	76,368	
DECSA-EA	1	ETHERNET ROUTER	100	15,451	15,451	15,451	152	1,824	Located as current, one DCSAX-LA in base
DCSAX-LB	2	56KB CARD FOR DECSA	100	744	1,488	1,488	11	264	
DCSAX-LA	3	9.6KB CARD FOR DECSA	100	475	1,425	1,425	11	396	Backup as current configuration
H4000	177	TRANCEIVER	100	315	55,755	55,755	4	8,496	
BNE3H-20	177	TRANCEIVER CABLE 20M	100	314	55,578	55,578	0	0	
BNE2A-ME	2	500M ETHERNET CABLE	100	7,035	14,070	14,070	0	0	One for LOCATION4-CN/SS, another for LOCATION4-E
12-19816-01	4	ETHERNET TERMINATOR	100	22	88	88	0	0	
DSU-56K	3	CODEX 56KB DSU	100	1,095	3,285	3,285	0	0	Two for LOCATION4-CN, one for LOCATION4-E
DF129-AA	4	DEC 9.6 KB MODEM	100	2,573	10,292	10,292	14	672	
DUP11	1	HOST 56KB LN. CARD	100	2,516	2,516	2,516	13	156	Inboard 56KB line card for LOCATION4-E
DF126-AA	15	DEC 2400 BAUD MODEM	100	1,050	15,750	15,750	12	2,160	
DHU11	1	16LN. ASYNC CNTLR.	100	4,955	4,955	4,955	55	660	For controlling asynchronous modems
	TOTAL EQUIPMENT COSTS					902,714		94,247	
LESS ALLOWANCES FOR SOFTWARE									
QK001-UZ	(2)	8600,8650 VMS	5	28,875	(57,750)	(2,888)	525	(630)	
Q5001-UZ	(1)	82XX VMS	5	21,000	(21,000)	(1,050)	466	(280)	
QKD05-UZ	(1)	86XX DECNET VAX	5	15,803	(15,803)	(790)	213	(128)	
QKD05-QZ	0	86XX DECNET VAX CLUS	5	9,482	0	0	213	0	Not bundled with the 8650
Q5D05-UZ	(1)	82XX DECNET VAX	5	9,335	(9,335)	(467)	189	(113)	
	NET TOTAL FOR LOCATION					897,520		93,096	

CONSOLIDATED MANUFACTURING, INC. - USC5 LOCATION5 EQUIPMENT 20-Dec-88

HEADQUARTERS

Acronym	Quantity	Description	Percent For Net.	Unit Price	Extended Price	Tot. Net One Time	Monthly Maint.	Total Ann. Maint	Comments
865CD-AP	1	VAX 8650 CLUSTER	5	554,400	554,400	27,720	1,851	1,111	Replaces three VAX 785's
865XB-AE	1	VAX 8650/NO DECNET	5	522,375	522,375	26,119	1,778	1,067	and the 8600 currently installed
DELUA-M	2	ETHNT ADPTR/UNIBUS	100	4,354	8,708	8,708	33	792	
HSC50	1	HIER. STOR. CNTLR.	0	40,360	40,360	0	155	0	
RA81-AA	16	DISK DRIVE	0	17,640	282,240	0	95	0	
TU79-AB	1	CLUSTER TAPE DRIVE	0	54,705	54,705	0	322	0	Same replacement as above
TA79-AF	1	SLAVE TAPE DRIVE	0	29,400	29,400	0	196	0	
TA79-BF	2	MASTER TAPE DRIVE	0	59,500	119,000	0	0	0	Same replacement as above
VT220-F2	820	TERMINAL	0	260	213,200	0	12	0	CRT's etc. plus 25 devices for data services
DSRVB-AA	118	DS 200 TERM. SERVER	100	3,806	449,108	449,108	37	52,392	
DECSA-EA	1	ETHERNET ROUTER	100	15,451	15,451	15,451	152	1,824	Includes one DCSAX-LA in base
DCSAX-LB	1	56KB CARD FOR DECSA	100	744	744	744	11	132	
H4000	139	TRANCEIVER	100	315	43,785	43,785	4	6,672	
BNE3H-20	139	TRANCEIVER CABLE 20M	100	314	43,646	43,646	0	0	
BNE2A-ME	10	500M ETHERNET CABLE	100	7,035	70,350	70,350	0	0	Ten Ethernet Segments to cover all campus buildings
12-19817-01	20	ETHERNET CONNECTOR	100	22	440	440	0	0	
BN25B-LO	1	1000M FBR. OPTIC CBL	100	12,253	12,253	12,253	0	0	For long range connections across road
DSU-56K	1	CODEX 56KB DSU	100	1,095	1,095	1,095	0	0	
DF126-AA	15	DEC 2400 BAUD MODEM	100	1,050	15,750	15,750	12	2,160	
DF129-AA	1	DEC 9.6 KB MODEM	100	2,573	2,573	2,573	14	168	
DEBET-RC	2	FIBER OPTIC BRIDGE	100	9,818	19,636	19,636	70	1,680	Connects segments across road
DEREP-AA	8	LOCAL LAN REPEATER	100	1,568	12,544	12,544	22	2,112	Assume local repeaters are ok for segments
DHU11	1	16LN. ASYNC CNTLR.	100	4,955	4,955	4,955	55	660	Open modem pool
		TOTAL EQUIPMENT COSTS				754,877		70,769	
LESS ALLOWANCES FOR SOFTWARE									
QK001-UZ		(2)8600,8650 VMS	5	28,875	(57,750)	(2,888)	525	(630)	
QKD05-UZ		(1)86XX DECNET VAX	5	15,803	(15,803)	(790)	213	(128)	
QKD05-QZ		0 86XX DECNET VAX CLUS	5	9,482	0	0	213	0	Not bundled with the 8650
		NET TOTAL FOR LOCATION				751,199		70,012	

CONSOLIDATED MANUFACTURING, INC. - USC5 LOCATION6 EQUIPMENT 20-Dec-88

SOUTH CENTRAL LOCATION

Acronym	Quantity	Description	Percent For Net.	Unit Price	Extended Price	Tot. Net One Time	Monthly Maint.	Total Ann. Maint	Comments
865CD-AP	1	VAX 8650 CLUSTER	5	554,400	554,400	27,720	1,851	1,111	
865XB-AE	2	VAX 8650/NO DECNET	5	522,375	1,044,750	52,238	1,778	2,134	Replaces the 8600, 4 785's and the 780 currently installed
DELUA-M	3	ETHNT ADPTR/UNIBUS	100	4,354	13,062	13,062	33	1,188	
DH-630Q2-HA	1	MICRO VAX II	5	24,600	24,600	1,230	273	164	
DELQA-M	1	ETHNT ADPTR/QBUS	100	2,500	2,500	2,500	15	180	
HSC50	2	HIER. STOR. CNTLR.	0	40,360	80,720	0	155	0	
RA81-AA	30	DISK DRIVE	0	17,640	529,200	0	95	0	
TA79-BF	2	MASTER TAPE DRIVE	0	59,500	119,000	0	0	0	Same replacement as above
TA79-AF	3	SLAVE TAPE DRIVE	0	29,400	88,200	0	196	0	
VT220-F2	3,268	TERMINAL	0	260	849,680	0	12	0	
DSRVB-AA	467	DS 200 TERM. SERVER	100	3,806	1,777,402	1,777,402	37	207,348	
DECSA-EA	1	ETHERNET ROUTER	100	15,451	15,451	15,451	152	1,824	Includes one DCSAX-LA in base
DCSAX-LB	1	56KB CARD FOR DECSA	100	744	744	744	11	132	
H4000	516	TRANCEIVER	100	315	162,540	162,540	4	24,768	
BNE3H-20	516	TRANCEIVER CABLE 20M	100	314	162,024	162,024	0	0	
BNE2A-ME	23	500M ETHERNET CABLE	100	7,035	161,805	161,805	0	0	
12-19816-01	46	ETHERNET TERMINATOR	100	22	1,012	1,012	0	0	
DSU-56K	1	CODEX 56KB DSU	100	1,095	1,095	1,095	0	0	
DF126-AA	30	DEC 2400 BAUD MODEM	100	1,050	31,500	31,500	12	4,320	
DF129-AA	1	DEC 9.6 KB MODEM	100	2,573	2,573	2,573	14	168	
DEBET-AA	20	LOCAL LAN BRIDGE	100	9,240	184,800	184,800	60	14,400	
DEREP-AA	2	LOCAL LAN REPEATER	100	1,568	3,136	3,136	22	528	
DHU11	2	16LN. ASYNC CNTLR.	100	4,955	9,910	9,910	55	1,320	
		TOTAL EQUIPMENT COSTS				2,610,742		259,584	
LESS ALLOWANCES FOR SOFTWARE									
QK001-UZ		(3)8600,8650 VMS	5	28,875	(86,625)	(4,331)	525	(945)	
QZ004-C5		(1)mVAX II VMS	5	18,900	(18,900)	(945)	390	(234)	
QKD05-UZ		(1)86XX DECNET VAX	5	15,803	(15,803)	(790)	213	(128)	
QKD05-QZ		0 86XX DECNET VAX CLUS	5	9,482	0	0	213	0	Not bundled with the 8650's
QZD05-UZ		(1)mVAX DECNET VAX	5	4,127	(4,127)	(206)	141	(85)	
		NET TOTAL FOR LOCATION				2,604,469		258,193	

CONSOLIDATED MANUFACTURING, INC. - USC5 LOCATION1 SOFTWARE 20-Dec-88

NETWORK CENTRAL

Acronym	Quantity	Description	Percent For Net.	Unit Price	Extended Price	Tot. Net One Time	Monthly Maint.	Tot. Net Ann. Cost
QKAAA-UZ	1	86XX ALL-IN-ONE	0	51,975	51,975	0	475	0
Q9AAA-QZ	1	8500/8530 A1 CLUSTER	0	31,185	31,185	0	475	0
Q7AAA-QZ	1	83XX ALL-IN-ONE CLUS	0	22,113	22,113	0	437	0
QK001-UZ	1	8600,8650 VMS	5	28,875	28,875	1,444	525	315
Q9001-UZ	1	8500,8530 VMS	5	38,115	38,115	1,906	525	315
Q7001-UZ	1	83XX VMS	5	26,250	26,250	1,313	466	280
QKD05-UZ	1	86XX DECNET VAX	100	15,803	15,803	15,803	213	2,556
Q9D05-UZ	1	8500/8530 DECNET VAX	100	15,141	15,141	15,141	213	2,556
Q7D05-UZ	1	83XX DECNET VAX	100	10,472	10,472	10,472	189	2,268
Q*Z06-*Z	3	DS200 LIC.	100	400	1,200	1,200	35	1,260
QK725-UZ	2	ROUT. LIC. 86XX	100	1,100	2,200	2,200	0	0
TOTAL FOR LOCATION						49,478		9,550

CONSOLIDATED MANUFACTURING, INC. - USC5 LOCATION2 SOFTWARE 20-Dec-88

EAST COAST

Acronym	Quantity	Description	Percent For Net.	Unit Price	Extended Price	Tot. Net One Time	Monthly Maint.	Total Ann. Maint
Q5AAA-UZ	1	82XX ALL-IN-ONE	0	28,382	28,382	0	437	0
Q5001-UZ	1	82XX VMS	5	21,000	21,000	1,050	466	280
Q5D05-UZ	1	82XX DECNET VAX	100	9,335	9,335	9,335	189	2,268
Q*Z06-*Z	1	DS200 LIC.	100	400	400	400	35	420
TOTAL FOR LOCATION						10,785		2,968

Comments

CONSOLIDATED MANUFACTURING, INC. - USC5 LOCATION3 SOFTWARE 20-Dec-88

MIDWEST MANUFACTURER

Acronym	Quantity	Description	Percent For Net.	Unit Price	Extended Price	Tot. Net One Time	Monthly Maint.	Total Ann. Maint.
Q5AAA-UZ	1	82XX ALL-IN-ONE	0	28,382	28,382	0	437	0
Q5001-UZ	1	82XX VMS	5	21,000	21,000	1,050	466	280
Q5D05-UZ	1	82XX DECNET VAX	100	9,335	9,335	9,335	189	2,268
Q*Z06-*Z	1	DS200 LIC.	100	400	400	400	35	420
Q5725-UZ	1	ROUT. LIC. 82XX	100	1,100	1,100	1,100	0	0
	TOTAL FOR LOCATION					11,885		2,968

CONSOLIDATED MANUFACTURING, INC. - USC5 LOCATION4 SOFTWARE 20-Dec-88

WEST COAST

Acronym	Quantity	Description	Percent For Net.	Unit Price	Extended Price	Tot. Net One Time	Monthly Maint.	Total Ann. Maint.
QKAAA-UZ	1	86XX ALL-IN-ONE	0	51,975	51,975	0	475	0
QKAAA-QZ	1	86XX ALL-IN-ONE CLUS	0	31,185	31,185	0	475	0
Q5AAA-QZ	1	82XX ALL-IN-ONE CLUS	0	17,031	17,031	0	437	0
QK001-UZ	2	8600,8650 VMS	5	28,875	57,750	2,888	525	630
Q5001-UZ	1	82XX VMS	5	21,000	21,000	1,050	466	280
QKD05-UZ	1	86XX DECNET VAX	100	15,803	15,803	15,803	213	2,556
QKD05-QZ	1	86XX DECNET VAX CLUS	100	9,482	9,482	9,482	213	2,556
Q5D05-UZ	1	82XX DECNET VAX	100	9,335	9,335	9,335	189	2,268
Q*Z06-*Z	3	DS200 LIC.	100	400	1,200	1,200	35	1,260
QK725-UZ	1	ROUT. LIC. 86XX	100	1,100	1,100	1,100	0	0
	TOTAL FOR LOCATION					40,858		9,550

CONSOLIDATED MANUFACTURING, INC. - USC5 LOCATION5 SOFTWARE 20-Dec-88

HEADQUARTERS

Acronym	Quantity	Description	Percent For Net.	Unit Price	Extended Price	Tot. Net One Time	Monthly Maint.	Total Ann. Maint.
QKAAA-UZ	1	86XX ALL-IN-ONE	0	51,975	51,975	0	475	0
QKAAA-QZ	1	86XX ALL-IN-ONE CLUS	0	31,185	31,185	0	475	0
QK001-UZ	2	8600,8650 VMS	5	28,875	57,750	2,888	525	630
QKD05-UZ	1	86XX DECNET VAX	100	15,803	15,803	15,803	213	2,556
QKD05-QZ	1	86XX DECNET VAX CLUS	100	9,482	9,482	9,482	213	2,556
Q*Z06-*Z	2	DS200 LIC.	100	400	800	800	35	840
QK725-UZ	1	ROUT. LIC. 86XX	100	1,100	1,100	1,100	0	0
TOTAL FOR LOCATION						30,073		6,582

CONSOLIDATED MANUFACTURING, INC. - USC5 LOCATION6 SOFTWARE 20-Dec-88

SOUTH CENTRAL LOCATION

Acronym	Quantity	Description	Percent For Net.	Unit Price	Extended Price	Tot. Net One Time	Monthly Maint.	Total Ann. Maint.
QKAAA-UZ	1	86XX ALL-IN-ONE	0	51,975	51,975	0	475	0
QKAAA-UZ	2	86XX ALL-IN-ONE	0	51,975	103,950	0	475	0
QZ004-C5	1	mVAX II VMS	5	18,900	18,900	945	390	234
QK001-UZ	3	8600,8650 VMS	5	28,875	86,625	4,331	525	945
QKD05-UZ	1	86XX DECNET VAX	100	15,803	15,803	15,803	213	2,556
QKD05-QZ	2	86XX DECNET VAX CLUS	100	9,482	18,964	18,964	213	5,112
QZD05-UZ	1	mVAX DECNET VAX	100	4,127	4,127	4,127	141	1,692
Q*Z06-*Z	2	DS200 LIC.	100	400	800	800	35	840
QK725-UZ	1	ROUT. LIC. 86XX	100	1,100	1,100	1,100	0	0
TOTAL FOR LOCATION						46,070		11,379

CONSOLIDATED MANUFACTURING, INC. - USC5 HUMAN RESOURCE/OPERATION 20-Dec-88

Acronym	Quantity	Description	Percent For Net.	Yearly Salary	Extended Cost	Total Ann. Cost	
ISMGMT	0.2	IS MANAGEMENT	100	80,000	160,000	32,000	CORPORATE ALL-IN-ONE MANAGER
DCOMMGMT		DATA COMM. MGMT.	100	50,000	100,000	0	
NETMON-1		NET. MONITOR LEVEL 1	100	24,000	48,000	0	
TECH-1		TECHNICIANS LEVEL 2	100	35,000	70,000	0	
FEPPGMR		FEP PROGRAMMER	100	42,500	85,000	0	
NETPLAN		NETWORK PLANNER	100	45,000	90,000	0	
NETADMIN	1.2	NET ADMINISTRATOR	100	45,000	90,000	108,000	5% OF SITE ADMINISTRATORS TIME(75% OPER)
ADMCLRK		ADMIN. CLERKS	100	25,000	50,000	0	
WIRING	3.9	WIRING TECHNICIAN	100	25,000	50,000	195,000	25% ALLOCATION OF 15.7 FTE'S
SYSPROG	1.1	SYSTEMS PROGRAMMER	100	42,500	85,000	93,500	75% ALLOCATION OF 1.5 SYSTEMS PROG FOR NETWORK
REMSYSCOORD		REMOTE SYS. COORD.	100	25,000	50,000	0	
APPLPROG	0.1	APPLICATIONS PROG.	100	30,000	60,000	6,000	
SECY		SECRETARIAL SUPPORT	100	25,000	50,000	0	
	TOTAL OPERATIONAL COST					434,500	

CONSOLIDATED MANUFACTURING, INC. - USC5 HUMAN RESOURCE/CHANGE 20-Dec-88

Acronym	Quantity	Description	Percent For Net.	Yearly Salary	Extended Cost	Total Ann. Cost	
ISMGMT	0.1	IS MANAGEMENT	100	80,000	160,000	16,000	CORPORATE ALL-IN-ONE MANAGER
DCOMMGMT		DATA COMM. MGMT.	100	50,000	100,000	0	
NETMON-1		NET. MONITOR LEVEL 1	100	24,000	48,000	0	
TECH-1		TECHNICIANS LEVEL 2	100	35,000	70,000	0	
FEPPGMR		FEP PROGRAMMER	100	42,500	85,000	0	
NETPLAN		NETWORK PLANNER	100	45,000	90,000	0	
NETADMIN	0.4	NET ADMINISTRATOR	100	45,000	90,000	36,000	5% OF SITE ADMINISTRATORS TIME(25% CHANGE)
ADMCLRK		ADMIN. CLERKS	100	25,000	50,000	0	
WIRING	11.8	WIRING TECHNICIAN	100	25,000	50,000	590,000	75% ALLOCATION OF 15.7 FTE'S
SYSPROG	0.4	SYSTEMS PROGRAMMER	100	42,500	85,000	34,000	25% ALLOCATION OF 1.5 SYSTEMS PROG FOR NETWORK
REMSYSCOORD		REMOTE SYS. COORD.	100	25,000	50,000	0	
APPLPROG	0.1	APPLICATIONS PROG.	100	30,000	60,000	6,000	
SECY		SECRETARIAL SUPPORT	100	25,000	50,000	0	
	TOTAL CHANGE COST					682,000	

CONSOLIDATED MANUFACTURING, INC. - USC5 COMMUNICATIONS 20-Dec-88

Acronym	Quantity	Description	Percent For Net.	Unit Price	Extended Price	Tot. Net One Time	Monthly Maint.	Total Ann. Maint.
9.6-1000	4	1000-MI 9.6KB ANL/D1	100	2,323	9,294	9,294	998	47,902
56-DD-1000	5	1000-MI 56KB DDS	100	1,998	9,988	9,988	4,183	250,960
	TOTAL FOR CORPORATION					19,281		298,861

CONSOLIDATED MANUFACTURING, INC. - USC5 FACILITIES 20-Dec-88

Acronym	Quantity	Description	Percent For Net.	Unit Price	Extended Price	Total Net One Time	Yearly Cost	Total Ann. Cost
COND	1,650	FT2 CONDIT. SPACE	100				50	82,500
UNCOND	950	FT2 UNCONDIT. SPACE	100				15	14,250
			TOTAL ANNUAL COST					96,750
WIRING	6,237	WIRING COST PER PORT		250		1,559,250		

Appendix B

Case Analysis: American Equipment Corporation

This appendix presents a detailed cost analysis of the American Equipment corporate network.

The 5,700-terminal IBM-based network is centralized. Its two major data centers are connected with each other by multiple 56 KB trunk circuits. All locations are supported by leased line facilities; there is minimal dial-up access. All routes have at least one alternate backup route and the capability to use dial backup facilities.

The network supports a workload of 70,000 transactions per day, primarily in support of production applications and electronic mail. It is operated around the clock with the exception of 16 hours of scheduled weekend down time. The availability goal is 99.5%, measured on a monthly basis for the whole network. The network response time goal is 2 seconds or less. The network is quite volatile, with user turnover of more than 40% per year.

The network is centrally-managed and supported. The technical support function of the central support group is split between the two data centers.

In modeling the network, the study team made three adjustments to enhance comparability with the other networks under study:

- Access from non-IBM devices were configured as IBM equivalents.
- X.25 and European portions of the network (under separate management control) were not included in the analysis. Other international locations were configured as domestic.
- Third party-managed network facilities were configured as in-house, dedicated facilities.

The organization of the spreadsheet is straightforward. Summary tabulations appear first, followed by a section for each of the five network cost components in the cost-of-network-ownership model.

AMERICAN EQUIPMENT CORPORATION - USC7 COST SUMMARY 15-Dec-88

	5-YEAR TOTAL COST ($000)				5-YEAR COST PER PORT (DOLLARS)				$ PER CHANGE
	ACQUIRE	OPERATION	CHANGE	TOTALS	ACQUIRE	OPERATION	CHANGE	TOTALS	CHANGE
EQUIPMENT	7,256	708		7,964	1,273	124	0	1,397	0
SOFTWARE	1,580	676		2,255	277	119	0	396	0
HUMAN RESOURCE		3,553	5,995	9,548	0	623	1,052	1,675	526
COMMUNICATIONS	594	13,389		13,984	104	2,349	0	2,453	0
FACILITIES	1,425	386		1,811	250	68	0	318	0
TOTALS	10,855	18,711	5,995	35,561	1,904	3,283	1,052	6,239	526

	5-YEAR TOTAL COST ($000)				5-YEAR COST PER PORT (DOLLARS)				$ PER CHANGE
	ACQUIRE	OPERATION	CHANGE	TOTALS	ACQUIRE	OPERATION	CHANGE	TOTALS	CHANGE
EQUIP/SFTWR *	8,836	1,383		10,219	1,550	243		1,793	0
HUMAN RESOURCE		3,553	5,995	9,548		623	1,052	1,675	526
COMMUNICATIONS	594	13,389		13,984	104	2,349		2,453	0
FACILITIES	1,425	386		1,811	250	68		318	0
TOTALS	10,855	18,711	5,995	35,561	1,904	3,283	1,052	6,239	526

* Equipment and software costs are combined to mask the effects of vendor bundling strategies.

AMERICAN EQUIPMENT CORPORATION - USC7 15-Dec-88

SUMMARY EQUIPMENT AND SOFTWARE

LOCATION NAME	EQUIPMENT DOLLARS		SOFTWARE DOLLARS	
	Acquire	Annual Operations	Acquire	Annual Operations
LOCATION1	1,308,485	35,208	1,069,477	51,732
LOCATION2	676,405	19,248	502,744	50,028
LOCATION3	590,000	19,368	7,500	33,360
LOCATION4	4,681,200	67,680	0	0
	7,256,090	141,504	1,579,721	135,120

AMERICAN EQUIPMENT CORPORATION - USC7 LOCATION1 EQUIPMENT 15-Dec-88

NETWORK CENTRAL

Acronym	Quantity	Description	Percent For Net.	Unit Price	Extended Price	Tot. Net One Time	Monthly Maint.	Tot. Net Ann. Cost	Comments
3725-001	2	3725 CNTLR. MOD 1	100	75,000	150,000	150,000	232	5,568	
3725-1561	4	3725 CHNL ADAPTER	100	6,750	27,000	27,000	9	408	
3725-7100	2	1 MB EXPANSION	100	6,000	12,000	12,000	4	96	
3725-4772	2	LAB TYPE B	100	26,400	52,800	52,800	30	720	
3725-4911	48	LIC TYPE 1	100	2,600	124,800	124,800	2	1,152	
3725-4931	9	LIC TYPE 3	100	3,000	27,000	27,000	2	216	
5865-2	167	IBM 9600/4800 DBU	100	4,000	668,000	668,000	12	24,048	3 domestic locations, 1 international location
5866-3	21	IBM 14.4 DBU MODEM	100	6,650	139,650	139,650	0	0	2 - INTL, 19 FOR SALES OFFICES
5865-7952	188	2 WIRE SNBU	100	450	84,600	84,600	1	2,256	BACKUP FOR ALL MODEMS
3725-8320	2	TWO PROCESSOR SWITCH	100	4,000	8,000	8,000	3	72	
DSU-56K	9	CODEX 56KB DSU	100	1,095	9,855	9,855	0	0	
3727-700	2	OPERATOR'S CONSOLE	100	2,390	4,780	4,780	28	672	
	TOTAL EQUIPMENT COSTS					1,308,485		35,208	

AMERICAN EQUIPMENT CORPORATION - USC7 LOCATION2 EQUIPMENT 15-Dec-88

DEVELOPMENT CENTER

Acronym	Quantity	Description	Percent For Net.	Unit Price	Extended Price	Tot. Net One Time	Monthly Maint.	Tot. Net Ann. Cost	Comments
3725-001	2	3725 CNTLR. MOD 1	100	75,000	150,000	150,000	232	5,568	
3725-1561	4	3725 CHNL ADAPTER	100	6,750	27,000	27,000	9	408	
3725-4772	2	LAB TYPE B	100	26,400	52,800	52,800	30	720	
3725-4911	15	LIC TYPE 1	100	2,600	39,000	39,000	2	360	
3725-4931	2	LIC TYPE 3	100	3,000	6,000	6,000	2	48	
5865-2	60	IBM 9600/4800 DBU	100	4,000	240,000	240,000	12	8,640	LOCAL SITES CONTROLLED BY LOCATION2
5865-7952	60	2 WIRE SNBU	100	450	27,000	27,000	1	720	
3725-8320	2	TWO PROCESSOR SWITCH	100	4,000	8,000	8,000	3	72	
DSU-56K	7	CODEX 56KB DSU	100	1,095	7,665	7,665	0	0	
3174-1L	9	LOCAL TERMINAL CNTLR	100	12,950	116,550	116,550	22	2,376	
3727-700	1	OPERATOR'S CONSOLE	100	2,390	2,390	2,390	28	336	
	TOTAL EQUIPMENT COSTS					676,405		19,248	

AMERICAN EQUIPMENT CORPORATION - USC7 LOCATION3 EQUIPMENT 15-Dec-88

OTHER DOMESTIC LOCATIONS

Acronym	Quantity	Description	Percent For Net.	Unit Price	Extended Price	Tot. Net One Time	Monthly Maint.	Tot. Net Ann. Cost	Comments
3174-1R	39	REMOTE TERM. CNTLR.	100	9,950	388,050	388,050	20	9,360	
3720-001	4	3720 CNTLR.	100	37,500	150,000	150,000	175	8,400	
3725-4911	10	LIC TYPE 1	100	2,600	26,000	26,000	2	240	
3727-700	4	OPERATOR'S CONSOLE	100	2,390	9,560	9,560	28	1,344	
DSU-56K	2	CODEX 56KB DSU	100	1,095	2,190	2,190	0	0	
5866-3	2	IBM 14.4 DBU MODEM	100	6,650	13,300	13,300	0	0	SUBSTITUTED FOR 19.2 AT 2 INT'L LOCATIONS
5865-7952	2	2 WIRE SNBU	100	450	900	900	1	24	SUBSTITUTED FOR 19.2 AT 2 INT'L LOCATIONS
TOTAL EQUIPMENT COSTS						590,000		19,368	

AMERICAN EQUIPMENT CORPORATION - USC7 LOCATION4 EQUIPMENT 15-Dec-88

OPERATIONS CENTERS

Acronym	Quantity	Description	Percent For Net.	Unit Price	Extended Price	Tot. Net One Time	Monthly Maint.	Tot. Net Ann. Cost	Comments
3174-1R	282	REMOTE TERM. CNTLR.	100	9,950	2,805,900	2,805,900	20	67,680	167 remote+int'l, 55 sales ofcs, 60 remotes from LOCATION2
5866-3	282	IBM 14.4 DBU MODEM	100	6,650	1,875,300	1,875,300	0	0	
TOTAL EQUIPMENT COSTS						4,681,200		67,680	

AMERICAN EQUIPMENT CORPORATION - USC7 LOCATION1 SOFTWARE 15-Dec-88

NETWORK CENTRAL

Acronym	Quantity	Description	Percent For Net.	Unit Price	Extended Price	Tot. Net One Time	Monthly Maint.	Tot. Net Ann. Cost
5665-289-40B	1	ACF VTAM V3 MVS/XA	100	108,420	108,420	108,420	302	3,624
5665-289-40D	1	ACF VTAM V3 MVS/XA	100	81,380	81,380	81,380	302	3,624
5665-285-40B	1	TSO/E MVS/XA 1.3.0	100	28,640	28,640	28,640	87	1,044
5665-285-40D	1	TSO/E MVS/XA 1.3.0	100	21,465	21,465	21,465	87	1,044
5665-362-40B	1	NETVIEW MVS XA	100	60,240	60,240	60,240	128	1,536
5665-362-40D	1	NETVIEW MVS XA	100	45,165	45,165	45,165	128	1,536
5665-403-40B	1	CICS MVS/XA	100	119,280	119,280	119,280	0	0
5665-403-40D	1	CICS MVS/XA	100	89,470	89,470	89,470	0	0
6665-272-40B	1	IMS-DC	100	240,000	240,000	240,000	575	6,900
5664-308-40B	1	VM/XA SP REL2 GRP 40	100	216,000	216,000	216,000	0	0
5668-854-40B	1	ACF NCP / 3725	100	2,085	2,085	2,085	695	8,340
5668-854-40D	1	ACF NCP / 3725	100	1,875	1,875	1,875	695	8,340
5735-XXB-B	1	EP RELEASE 4	100	1,365	1,365	1,365	321	3,852
5735-XXB-D	1	EP RELEASE 4	100	1,025	1,025	1,025	321	3,852
5665-333-40B	1	NET. PERFORM. MON.	100	36,720	36,720	36,720	0	0
5668-981B	1	NPSI NCP X.25 R 4.3	100	770	770	770	335	4,020
5668-981D	1	NPSI NCP X.25 R 4.3	100	577	577	577	335	4,020
5664-202	1	NETDA NET DESIGN AID	100	15,000	15,000	15,000	0	0
TOTAL FOR LOCATION						1,069,477		51,732

AMERICAN EQUIPMENT CORPORATION - USC7 LOCATION2 SOFTWARE 15-Dec-88

DEVELOPMENT CENTER

Acronym	Quantity	Description	Percent For Net.	Unit Price	Extended Price	Tot. Net One Time	Monthly Maint.	Tot. Net Ann. Cost
5665-289-40D	1	ACF VTAM V3 MVS/XA	100	81,380	81,380	81,380	302	3,624
5664-280-40B	1	ACF VTAM V3 VM/SP	100	44,940	44,940	44,940	247	2,964
5665-285-40D	1	TSO/E MVS/XA 1.3.0	100	21,465	21,465	21,465	87	1,044
5665-362-40D	1	NETVIEW MVS XA	100	45,165	45,165	45,165	128	1,536
5664-204-40D	1	NETVW VM FOR GRP 40	100	27,070	27,070	27,070	90	1,080
5665-403-40D	1	CICS MVS/XA	100	89,470	89,470	89,470	0	0
6665-272-40D	1	IMS-DC	100	180,000	180,000	180,000	575	6,900
5668-854-40D	1	ACF NCP / 3725	100	1,875	1,875	1,875	695	8,340
5668-854-40D	1	ACF NCP / 3725	100	1,875	1,875	1,875	695	8,340
5735-XXB-D	2	EP RELEASE 4	100	1,025	2,050	2,050	321	7,704
5668-981D	2	NPSI NCP X.25 R 4.3	100	577	1,154	1,154	335	8,040
5664-188-40B	1	RSCS VERS. 2 VM	100	6,300	6,300	6,300	38	456
		TOTAL FOR LOCATION				502,744		50,028

AMERICAN EQUIPMENT CORPORATION - USC7 LOCATION3 SOFTWARE 15-Dec-88

OTHER DOMESTIC LOCATIONS

Acronym	Quantity	Description	Percent For Net.	Unit Price	Extended Price	Tot. Net One Time	Monthly Maint.	Tot. Net Ann. Cost
5668-854-20D	4	ACF NCP / 3725	100	1,875	7,500	7,500	695	33,360
		TOTAL FOR LOCATION				7,500		33,360

AMERICAN EQUIPMENT CORPORATION - USC7 — HUMAN RESOURCE -- OPERATION — 15-Dec-88

Acronym	Quantity	Description	For Net.	Extended Quantity	Yearly Salary	Extended Cost	Total Ann. Cost	Comments
ISMGMT		IS MANAGEMENT	100	0.00	80,000	160,000	0	
DCOMMGMT		DATA COMM. MGMT.	100	0.00	50,000	100,000	0	
NETMON-1	4.0	NET. MONITOR LEVEL 1	100	4.00	24,000	48,000	192,000	NCC - 2 FIRST SHIFT, 1 SWING & MID
TECH-1		TECHNICIANS LEVEL 2	100	0.00	35,000	70,000	0	
FEPPGMR		FEP PROGRAMMER	100	0.00	42,500	85,000	0	
NETPLAN	5.0	NETWORK PLANNER	50	2.50	45,000	90,000	225,000	PLANNING NETWORK CHANGES AND PERFORMANCE MONITORING
NETADMIN		NET ADMINISTRATOR	100	0.00	45,000	90,000	0	
ADMCLRK	5.0	ADMIN. CLERKS	40	2.00	25,000	50,000	100,000	HELP FOR TERMINAL AND NETWORK RELATED PROBLEMS
WIRING		WIRING TECHNICIAN	100	0.00	25,000	50,000	0	
SYSPROG	5.5	SYSTEMS PROGRAMMER	20	1.10	42,500	85,000	93,500	CICS, NCP AND OPERATIONAL SUPT. FOR NETWORK (NO R&D @1 FTE)
REMSYSCOORD	10.0	REMOTE SYS. COORD.	20	2.00	25,000	50,000	100,000	REMOTE PROB. DETERMINATION (TERMINAL SERVICES PEOPLE)
APPLPROG		APPLICATIONS PROG.	100	0.00	30,000	60,000	0	
SECY		SECRETARIAL SUPPORT	100	0.00	25,000	50,000	0	
		TOTAL OPERATIONAL COST		11.60			710,500	

AMERICAN EQUIPMENT CORPORATION - USC7 — HUMAN RESOURCE -- CHANGE — 15-Dec-88

Acronym	Quantity	Description	Percent For Net.	Extended Quantity	Yearly Salary	Extended Cost	Total Ann. Cost	Comments
ISMGMT		IS MANAGEMENT	100	0.00	80,000	160,000	0	
DCOMMGMT		DATA COMM. MGMT.	100	0.00	50,000	100,000	0	
NETMON-1		NET. MONITOR LEVEL 1	100	0.00	24,000	48,000	0	
TECH-1		TECHNICIANS LEVEL 2	100	0.00	35,000	70,000	0	
FEPPGMR		FEP PROGRAMMER	100	0.00	42,500	85,000	0	
NETPLAN	5.0	NETWORK PLANNER	50	2.50	45,000	90,000	225,000	PROJECT MANAGEMENT FOR NETWORK CHANGES
NETADMIN		NET ADMINISTRATOR	100	0.00	45,000	90,000	0	
ADMCLRK		ADMIN. CLERKS	100	0.00	25,000	50,000	0	
WIRING	4.0	WIRING TECHNICIAN	100	4.00	25,000	50,000	200,000	HOST SITE WIRING RESPONSIBILITY
SYSPROG	5.5	SYSTEMS PROGRAMMER	80	4.40	42,500	85,000	374,000	CICS AND FEP PROGRAMMERS INVOLVED IN CHANGE
REMSYSCOORD	10.0	REMOTE SYS. COORD.	80	8.00	25,000	50,000	400,000	CONTROLLER AND NETWORK RELATED AMOUNT OF CHANGE TIME
APPLPROG		APPLICATIONS PROG.	100	0.00	30,000	60,000	0	
SECY		SECRETARIAL SUPPORT	100	0.00	25,000	50,000	0	
		TOTAL CHANGE COST		18.90			1,199,000	

AMERICAN EQUIPMENT CORPORATION - USC7 COMMUNICATIONS 15-Dec-88

Acronym	Quantity	Description	Percent For Net.	Unit Price	Extended Price	Tot. Net One Time	Monthly Maint.	Total Ann. Maint.	
9.6-500	248	500-MI 9.6KB ANLG/D1	100	2,323	576,213	576,213	788	2,345,534	2 - INTL, 19 - SALES, 167 - LOCATION1, 60 LOCATION2
56-DD-500	9	500-MI 56 KB DDS	100	1,998	17,978	17,978	3,077	332,328	2 - INTL, LOCATION1 TO LOC2, 2 - BATCH & 5 - INTERACTIVE
	TOTAL FOR CORPORATION					594,191		2,677,862	

AMERICAN EQUIPMENT CORPORATION - USC7 FACILITIES 15-Dec-88

Acronym	Quantity	Description	Percent For Net.	Unit Price	Extended Price	Total Net One Time	Yearly Cost	Total Ann. Cost
COND	512	FT2 CONDIT. SPACE	100	0	0	0	50	25,600
UNCOND	3,439	FT2 UNCONDIT. SPACE	100	0	0	0	15	51,585
			TOTAL ANNUAL COST					77,185
WIRING	5,700	WIRING COST PER PORT		250		1,425,000		

Computerworld 10-14-85
v 19 n41 p 1-2, 4+

Is your DBMS really relational?

Rule Zero: For any system that is advertised as, or claimed to be, a relational data base management system, that system must be able to manage data bases entirely through its relational capabilities.

By E. F. Codd

The originator of the relational model for data base management presents basic principles for determining how relational a DBMS product is — a question that faces many buyers today because almost every vendor claims its DBMS is relational. Some vendors may not realize how far from the mark they are.

Part 1

In recent years, the data base management system market has undergone a very rapid swing in favor of products that take the relational approach to data base management. It is hard to find a vendor that does not claim its DBMS is relational. This swing has been so extensive that some vendors of nonrelational DBMS have quickly (and recently) added a few relational features — in some, cases, very few features — in order to be able to claim their systems are relational, even though they may not meet the simple requirements for being rated "minimally relational." We shall refer to this kind of DBMS as "born again."

It is a safe bet that these Johnny-come-lately vendors have not taken the time or manpower to investigate optimization techniques needed in relational DBMS to yield good performance. This is the principal reason they continue to proclaim the "performance myth" — namely, that relational DBMS must perform poorly because they are relational!

One consequence of this rapid swing of the market to the relational approach is that products that are claimed by their vendors to be relational DBMS range from those that support the relational model with substantial fidelity to those that definitely do not deserve the label "relational," because their support is only token.

Some vendors claim that fourth-generation languages will provide all the productivity advantages. This claim conveniently overlooks the fact that most of these languages do little or nothing for shared data (the programming language fraternity

E. F. Codd is the originator of the relational model for data base management. He was the leader of the team that designed and implemented the first operating system with multiprogramming capability. Currently he is president of The Relational Institute and the Codd & Date Consulting Group, both based in San Jose, Calif.

still does not appear to realize that support for the dynamic sharing of data is an absolute requirement). In addition, there is no accepted theoretical foundation for fourth-generation languages and not even an accepted, precise definition.

This article outlines a technique that should help users determine how relational a DBMS really is. Accordingly, I shall discuss the following:

- The fidelity of DBMS to the relational model.
- The fidelity of the proposed Ansi SQL standard to the relational model.
- Conclusions regarding choosing a DBMS product.

I shall not attempt a complete description of the relational model here — a relatively brief and concise definition appears in the article "RM/T: Extending the Relational Model to Capture More Meaning," (Chapter 2, "The Basic Relational Model") in the Association for Computing Machinery's "Transactions on Data Base Systems" (December 1979). It is, however, vitally important to remember that the relational model includes three major parts: the structural part, the manipulative part and the integrity part — a fact that is frequently and conveniently forgotten.

> *The fidelity of the proposed Ansi standard to the relational model is even less than that of some relational DBMS products. However, the standard could be readily modified to be more faithful to the model, and pressure should be brought on Ansi to do so.*

In this paper, I supply a set of rules with which a DBMS should comply if it is claimed to be fully relational. **No existing DBMS product that I know of can honestly claim to be fully relational,** at this time.

The proposed Ansi standard does not fully comply with the relational model, because it is based heavily on that nucleus of SQL that is supported in common by numerous vendors. Moreover, it takes a static, schema-based approach to data base description — reminiscent of Codasyl — instead of specifying a comprehensive, dual-mode data sublanguage that provides the powerful yet easy access to relational data bases and that is unique to the relational approach. Thus, the fidelity of the proposed Ansi standard to the relational model is even less than that of some relational DBMS products.

However, the standard could be readily modified to be more faithful to the model, and pressure should be brought on Ansi to do so. In fact, vendors are advised to extend their products soon in these respects so that they support customers' DBMS needs more fully and avoid possibly large customer expenses in application program maintenance at the time of the improvement.

The 12 rules

Twelve rules are cited below as part of a test to determine whether a product that is claimed to be fully relational is actually so. Use of the term "fully relational" in this report is slightly more stringent than in my Turing paper (written in 1981). This is partly because vendors in their ads and manuals have translated the term "minimally relational" to "fully relational" and partly because in this report, we are dealing with relational DBMS and not relational systems in general, which would include mere query-reporting systems.

However, the 12 rules tend to explain why full support of the relational model is in the users' interest. No new requirements are added to the relational model. A grading scheme is later defined and used to measure the degree of fidelity to the relational model.

First, I define these rules. Although I have defined each rule in earlier papers, I believe this to be the first occurrence of all 12 of them together.

In rules eight through 11, I specify and require four different types of independence aimed at protecting customers' investments in application programs, terminal activities and training. Rules eight and nine — physical and logical data independence — have been heavily discussed for many years.

Rules 10 and 11 — integrity independence and distribution independence — are aspects of the relational approach that have received inadequate attention to date but are likely to become as important as eight and nine.

These rules are based on a single foundation rule, which I shall call Rule Zero:

For any system that is advertised as, or claimed to be, a relational data base management system, that system must be able to manage data bases entirely through its relational capabilities.

This rule must hold whether or not the system supports any nonrelational capabilities of managing data. Any DBMS that does not satisfy this Rule Zero is not worth rating as a relational DBMS.

One consequence of this rule: Any system claimed to be a relational DBMS must support data base insert, level (multiple-record-at-a-time). Another consequence is the necessity of supporting the information rule and the guaranteed access rule.

"Multiple-record-at-a-time" includes as special cases those situations in which zero or one record is retrieved, inserted, updated or deleted. In other words, a relation (table) may have either zero tuples (rows) or one tuple and still be a valid relation.

Any statement in the manuals of a system claimed to be a relational DBMS that advises users to revert to some nonrelational capabilities "to achieve acceptable performance" — or for any reason other than compatibility with programs written in the past on nonrelational data base systems — should be interpreted as an apology by the vendor. Such a statement indicates the vendor has not done the work necessary for achieving good performance with the relational approach.

What is the danger to buyers and users of a system that is claimed to be a relational DBMS and that fails on Rule Zero? Buyers and users will expect all the advantages of a truly relational DBMS, and they will fail to get these advantages.

Now I shall describe the 12 rules that, together with the nine structural, 18 manipulative and three integrity features of the relational model, determine in specific detail the extent of validity of a vendor's claim to have a "fully relational DBMS."

All 12 rules are motivated by Rule Zero defined above, but a DBMS can be more readily checked for compliance with these 12 than with Rule Zero.

> Rule 1: *All information in a relational data base is represented explicitly at the logical level and in exactly one way — by values in tables.*

The information rule.

Rule 1: *All information in a relational data base is represented explicitly at the logical level and in exactly one way — by values in tables.*

Even table names, column names and domain names are represented as character strings in some tables. Tables containing such names are normally part of the built-in system catalog. The catalog is accordingly a relational data base itself — one that is dynamic and active and represents the metadata (data describing the rest of the data in the system).

The information rule is enforced not only for user productivity but also to make it a reasonably simple job for software vendors to define additional software packages (such as application development aids, expert systems and so on) that interface with relational DBMS and, by definition, are well integrated with the DBMS.

That is, these packages retrieve information already existing in the catalog and, as needed, put new information in the catalog by the very act of using the DBMS.

An additional reason to enforce this rule is to make the data base administrator's task of maintaining the data base in a state of overall integrity both simpler and more effective. There is nothing more embarrassing to a data base administrator than being asked if his data base contains certain specific information and his replying after a week's examination of the data base that he does not know.

Guaranteed access rule.

Rule 2: *Each and every datum (atomic value) in a relational data base is guaranteed to be logically accessible by resorting to a combination of table name, primary key value and column name.*

Clearly, each datum in a relational data base can be accessed in a rich variety — possibly thousands — of logically distinct ways. However, it is important to have at least one way, independent of the specific relational data base, that is guaranteed, because most computer-oriented concepts (such as scanning successive addresses) have been deliberately omitted from the relational model.

Note that the guaranteed access rule represents an associative addressing scheme that is unique to the relational model. The rule does not depend at all on the usual computer-oriented addressing. However, the primary key concept is an essential part of it.

Systematic treatment of null values.

Rule 3: *Null values (distinct from the empty character string or a string of blank characters and distinct from zero or any other number) are supported in fully relational DBMS for representing missing information and inapplicable information in a systematic way, independent of data type.*

To support data base integrity, it must be possible to specify "nulls not allowed" for each primary key column and for any other columns where the data base administrator considers it an appropriate integrity constraint (for example, certain foreign key columns).

Past techniques entailed defining a special value (peculiar to each column or field) to represent missing information. This would be most unsystematic in a relational data base because users would have to employ different techniques for each column or domain — a difficult task because would decrease user productivity).

Dynamic on-line catalog based on the relational model.

Rule 4: *The data base description is represented at the logical level in the same way as ordinary data, so that authorized users can apply the same relational language to its interrogation as they apply to the regular data.*

One consequence of this is that each user (whether an application programmer or end user) needs to learn only one data model — an advantage that nonrelational systems usually do not offer (IBM's IMS, together with its dictionary, requires the user to learn two distinct data models).

Another consequence is that authorized users can easily extend the catalog to become a full-fledged, active relational data dictionary whenever the vendor fails to do so.

Comprehensive data sublanguage rule.

Rule 5: *A relational system may support several languages and various modes of terminal use (for example, the fill-in-the-blanks mode). However, there must be at least one language whose statements are expressible, per some well-defined syntax, as character strings and that is comprehensive in supporting all of the following items:*

- *Data definition.*
- *View definition.*
- *Data manipulation (interactive and by program).*
- *Integrity constraints.*
- *Authorization.*
- *Transaction boundaries (begin, commit and rollback).*

The relational approach is intentionally highly dynamic — that is, it should rarely be necessary to bring the data base activity to a halt (in contrast to nonrelational DBMS). Therefore, it does not make sense to separate the services listed above into distinct languages.

In the mid-'70s, the Ansi Standards Planning and Requirements Committee generated a document advocating 42 distinct interfaces and (potentially) 42 distinct languages for DBMS. Fortunately, that idea has apparently been abandoned.

View updating rule.

Rule 6: *All views that are theoretically updatable are also updatable by the system.*

Note that a view is theoretically updatable if there exists a time-independent algorithm for unambiguously determining a single series of changes to the base relations that will have as their effect precisely the requested changes in the view. In this regard, "update" is intended to include insertion and deletion as well as modification.

High-level insert, update and delete.

Rule 7: *The capability of handling a base relation or a derived relation as a single operand applies not only to the retrieval of data but also to the insertion, update and deletion of data.*

This requirement gives the system much more scope in optimizing the efficiency of its execution-time actions. It allows the system to determine which access paths to exploit to obtain the most efficient code.

It can also be extremely important in obtaining efficient handling of transactions across a distributed data base. In this case, users would prefer that communications costs are saved by avoiding the necessity of transmitting a separate request for each record obtained from remote sites.

"Rule 4: *The data base description is represented at the logical level in the same way as ordinary data, so that authorized users can apply the same relational language to its interrogation as they apply to the regular data.*"

Physical data independence.
Rule 8: *Application programs and terminal activities remain logically unimpared whenever any changes are made in either storage representations or access methods.*

To handle this, the DBMS must support a clear, sharp boundary between the logical and semantic aspects on the one hand and the physical and performance aspects of the base tables on the other; application programs must deal with the logical aspects only.

Nonrelational DBMS rarely provide complete support for this rule — in fact, I know of none that do.

Logical data independence.
Rule 9: *Application programs and terminal activites remain logically unimpared when information-preserving changes of any kind that theoretically permit unimpairment are made to the base tables.*

Take the following two examples: splitting a table into two tables, either by rows using row content or by columns using column names, if primary keys are preserved in each result; or combining two tables into one by means of a nonloss join (Stanford University and MIT authors call these joins "lossless").

To provide this service whenever possible, the DBMS must be capable of handling inserts, updates and deletes on *all views* that are theoretically updatable. This rule permits logical data base design to be changed dynamically if, for example, such a change would improve performance.

The physical and logical data independence rules permit data base designers for relational DBMS to make mistakes in their designs without the heavy penalties levied by nonrelational DBMS. This, in turn, means that it is much easier to get started with a relational DBMS because not nearly as much performance-oriented planning is needed prior to "blast-off."

Integrity independence.
Rule 10: *Integrity constraints specific to a particular relational data base must be definable in the relational data sublanguage and storable in the catalog, not in the application programs.*

In addition to the two integrity rules (entity integrity and referential integrity) that apply to every relational data base, there is a clear need to be able to specify additional integrity constraints reflecting either business policies or government regulations.

Assume the relational model is faithfully reflected. Then, the additional integrity constraints are defined in terms of the high-level data sublanguage and the definitions stored in the catalog, not in the application programs.

Information about inadequately identified objects is never recorded in a relational data base. To be more specific, the following two integrity rules apply to every relational data base:

Entity integrity. No component of a primary key is allowed to have a null value.

Referential integrity. For each distinct nonnull foreign key value in a relational data base, there must exist a matching primary key value from the same domain.

If, as sometimes happens, either business policies or government regulations change, it will probably become necessary to change the integrity constraints. Normally, this can be accomplished in a fully relational DBMS by changing one or more of the integrity statements that are stored in the catalog.

In many cases, neither the application programs nor the terminal activities are logically impaired.

Nonrelational DBMS rarely support this rule as part of the DBMS engine, where it belongs. Instead, they depend on a dictionary package, which may or may not be present and can readily be bypassed.

> Rule 11: *A relational DBMS has distribution independence.*

Distribution independence.
Rule 11: *A relational DBMS has distribution independence.*

By distribution independence, I mean that the DBMS has a data sublanguage that enables application programs and terminal activities to remain logically unimpaired:

- when data distribution is first introduced (if the originally installed DBMS manages nondistributed data only);
- when data is redistributed (if the DBMS manages distributed data).

Note that the definition is carefully worded so that both distributed and nondistributed DBMS can fully support Rule 11. IBM's SQL/DS and DB2, Oracle Corp.'s Oracle and Relational Technology, Inc.'s Ingres (all nondistributed in present releases) fully support this rule.

This has been demonstrated as follows: SQL programs have been written to operate on nondistributed data (using System R) run correctly on distributed versions of that data (using System R*, the IBM San Jose Research Laboratory prototype), and the distributed Ingres project at the University of California at Berkeley has shown the same capability for the Quel language of Ingres.

It is important to distinguish distributed processing from distributed data. In the former case, work (for example, programs) is transmitted to the data; in the latter case, data is transmitted to the work. Many nonrelational DBMS support distributed processing but not distributed data. The only systems that support the concept of making all the distributed data *appear* to be local are relational DBMS — these are prototypes right now.

In the case of a distributed relational DBMS, a single transaction may straddle several remote sites. Such straddling is managed entirely under the covers — the system may have to execute recovery at multiple sites. Each program or terminal activity treats the total data as if it were all local at the site where the application program or terminal activity is being executed.

A fully relational DBMS that does not support distributed data bases has the capability of being extended to provide that support while leaving application programs and terminal activities logically unimpaired, both at the time of initial distribution and whenever later redistribution is made.

There are four important reasons why relational DBMS enjoy this advantage:

- **Decomposition flexibility** in deciding how to deploy the data.
- **Recomposition power** of the relational operators when combining the results of subtransactions executed at different sites.
- **Economy of transmission** resulting from the fact that there need not be a request message sent for each record to be retrieved from any remote site.
- **Analyzability of intent** (owing to the very high level of relational languages) for vastly improved optimization of execution.

Nonsubversion rule.
Rule 12: *If a relational system has a low-level (single-record-at-a-time) language, that low level cannot be used to subvert or bypass the integrity rules and constraints expressed in the higher level relational language (multiple-records-at-a-time).*

In the relational approach, preservation of integrity is made independent of logical data structure to achieve integrity independence. This rule is extremely difficult for a "born-again" system to obey because such a system already supports an interface below the relational constraint interface. Vendors of "born-again" systems do not appear to have given this problem adequate attention.

(Part two: the practical consequences of the 12 rules and an evaluation of certain products against the relational model.)

Codd, E.F.
"Does your DBMS run by the rules?

Computerworld In Depth
v.19, n42 10-21-85
p. 49-52, 54+

Does your DBMS run by the rules?

To be "mid-80s" fully relational, a DBMS must support all 12 basic rules plus nine structural, 18 manipulative and all three integrity rules. There will be more requirements by the 1990s.

By E. F. Codd

Last week, the originator of the relational model described the 12 rules by which to measure any DBMS claiming to be relational. This week, Dr. E. F. Codd presents the practical consequences of his 12 rules as well as 30 additional features of a relational system. Then he asks vendors to measure up.

Part 2

No existing DBMS product that I know of can honestly claim to be fully relational at this time. The proposed ANSI standard does not fully comply with the relational model, so a DBMS' fidelity to the ANSI standard is no guarantee of relational capability. The standard could be modified, but already vendors are well advised to extend their products beyond the standard to support customers' DBMS needs fully.

In their ads and manuals, vendors have translated the term "minimally relational" to "fully relational," so more stringent criteria must be applied. Twelve rules (below) comprise a test to determine whether a product that is claimed to be fully relational is actually so. A grading scheme used to measure the degree of fidelity to the relational model follows.

A DBMS advertised as relational should comply with the following 12 rules:

1. The information rule.
2. The guaranteed access rule.
3. Systematic treatment of null values.
4. Active on-line catalog based on the relational model.
5. The comprehensive data sublanguage rule.
6. The view updating rule.
7. High-level insert, update and delete.
8. Physical data independence.
9. Logical data independence.
10. Integrity independence.
11. Distribution independence.
12. The nonsubversion rule.

E. F. Codd originated the relational model for data base management. He led the team that designed and implemented the first operating system with multiprogramming capability. This year he established two companies with Chris Date: The Relational Institute and the Codd & Date Consulting Group, both based in San Jose, Calif.

These rules are based on a single foundation rule. I call it Rule Zero:

For any system that is advertised as, or claimed to be, a relational data base management system, that system must be able to manage data bases entirely through its relational capabilities.

This rule must hold whether or not the system supports any nonrelational capabilities of managing data. Any DBMS that does not satisfy this Rule Zero is not worth rating as a relational DBMS.

But compliance with Rule Zero is not enough. Failure to support the information rule, guaranteed access rule, systematic nulls rule and catalog rule can make integrity impossible to maintain. These four rules support significantly higher standards for data base administration and control (authorization and integrity control) than earlier DBMS supported. Users should remember that a data base managed by a relational DBMS is likely to have both experienced and inexperienced users; it must be able to serve both.

Rule Zero not enough

Rules 1 and 4, the information and catalog rules, allow people with appropriate authorization (such as executives of the company) to find out easily via terminal what information is stored in a data base. I have encountered data base administrators using nonrelational systems who were unable to determine if a specific kind of information was recorded in their data base.

Rule 3, which calls for the inclusion of systematic support for unknown and inapplicable information by means of null values that are independent of data type, should help users to avoid foolish and possibly costly mistakes. The treatment of nulls, when aggregate functions such as total and average are applied, holds considerable interest for users. The Oracle DBMS in particular has an outstanding approach to null values. The user may specify whether the aggregate function is to ignore null values or yield a null result if any null value is encountered.

In general, controversy still surrounds the problem of missing and inapplicable information in data bases. It seems to me that those who complain loudly about the complexities of manipulating nulls are overlooking the fact that handling missing and inapplicable information is inherently complicated. Going back to programmer-specified default values does not solve the problem.

Rule 5, the comprehensive

> ”
>
> *The ANSI standard as now proposed is quite weak. It fails to support numerous features users need to reap the advantages of the relational approach.*

data sublanguage rule, is important for several reasons. First, it allows programmers to debug their data base statements interactively, treating them separately from whatever nondata base statements occur in their programs — a significant contributor to productivity. Second, it means that a single tool can be used for defining relations derived from the data base, whatever the purpose. The view updating rule, Rule 6, is vital for the system to support logical data independence.

Rule 7, which requires a multiple-record-at-a-time attack on insertion, update and deletion, can help save a good portion of the total cost of intersite communication in a distributed data base. If the system includes a good optimizer (an important component in relational DBMS performance), this rule can also result in substantial saving of CPU and I/O time, whether the data base is distributed or not.

Failure to support independence (Rules 8 through 11) can, and very likely will, result in skyrocketing costs in both money and time. Developing and maintaining applications programs and terminal activities will be more expensive. Managers may even be unwilling to consider changing certain business policies simply because of the anticipated program maintenance costs.

Rule 12, the nonsubversion rule, is crucial in protecting the integrity of relational data bases. All too frequently, I have seen situations in which data base administrators with nonrelational DBMS failed to control their data bases adequately; consequently, they could not maintain a state of integrity.

Domains

Many users confuse the domain concept with the concept of attribute of a relation or column of a table. Other people (often the vendors themselves) dismiss the domain concept as "academic." My reply to them is: The atom bomb was also academic!

In fact, the domain concept is very important, practical and simple. A domain consists of the whole set of legal values that can occur in a column. The column draws its values from the domain. Each column of a relational data base has precisely one domain, but any number of columns may share a domain. There are several reasons why domains should be supported.

For example, in a financial data base, there may be as many as 50 distinct columns (possibly, but not necessarily, in distinct tables) defined around the U.S. currency domain. Why repeat the definition of currency 50 times? In data bases supported by nonrelational systems, I frequently observe many inconsistent declarations of value type for fields that were intended to have the same type.

It is unreasonable to expect a DBMS to store all the legal values in a domain, unless there happen to be very few. However, it is entirely reasonable — and very worthwhile — to insist that a DBMS should store certain values:

- For each domain, a description of the type of values in that domain. This information is global since it applies to the entire database, and it should of course be recorded in the catalog.
- For each column, the name of the domain from which that column draws its values. This domain name is a reference to the global definition.

Of course, the domain description can include range restrictions. For example, it could specify that quantities of parts in an inventory must not only be integers, they must also be non-negative.

Furthermore, individual columns may include additional range restrictions where these are semantically justifiable. In this example, the quantities of very expensive parts held in the inventory may be limited to some specified maximum.

One of the benefits of supporting the domain concept is that, in cases where several columns share a common domain, the declaration of values is largely or even completely factored out. For example, when there are 50 distinct columns defined on U.S. currency, the data base is much easier to manage and manipulate if one avoids making 50 distinct declarations for U.S. currency.

Before the relational discipline arrived, users had to make separate declarations, and as a result, many of the 50 in the example would turn out to be incompatible with one another by accident. The factoring of declaration that prevents these errors is achieved in Digital Equipment Corp.'s RDB, which has a concept of "global field definition." But RDB fails to support domain constraints on certain operations, such as join.

Another benefit of supporting the domain concept is that relational operators, such as joins and divides, that involve comparison of values between different columns can be constrained by the system. A DBMS can allow data base values to be compared only when they come from the same domain and are therefore comparable from the semantic viewpoint.

Such a constraint inhibits errors caused by interactive users of terminals who choose columns to be compared in such operations as joins. The wrong answers they obtain from these errors rarely uncover the errors themselves; meanwhile, unwise business decisions may be made based on these wrong answers.

For various reasons, it is important to support as a qualifier in a command what I call "semantic override" — the ability to have the system ignore the usual comparison constraints. Users should be able to authorize this override qualifier separately from the operator involved and should authorize it rarely, reserving it chiefly for detective work.

IN DEPTH/RELATIONAL DBMS

Even when the domain concept is restricted to assigning types to data, it should not be confused with the hardware-supported data type. Consider the example of a data base listing suppliers, parts and projects. Suppose the hardware-supported data types of supplier serial numbers and part serial numbers are identical: each type consists of fixed-length strings of 12 characters. The system still needs to keep these two data types distinct and remember which columns are defined on one and which columns are defined on the other.

If it can make these distinctions, then when a request comes in to delete or archive all records containing X3 as a supplier serial number, the system can handle such a transaction correctly. The system will not delete or archive any record that contains X3 as a part serial number and that also does not contain X3 as a supplier serial number.

Today, such a data type is often called an application data type. The concept is supported in Pascal but in very few other languages that enjoy current use. The Pascal support does not, of course, include constraints on selects, unions, joins and divides.

The domain concept is basically what makes all the meaningful selects, unions, joins and divides known to the DBMS. Thus, the domain makes the data base meaningfully integrated, and it does so without prejudicing distributability.

Contrast this with CODASYL links and IMS hierarchic links. They represent the CODASYL and IMS concept that a link "integrates an otherwise unintegrated data base," but they have several unfortunate restrictions. Most importantly, they obstruct data base distribution because of the constraints and complexity their data structures introduce into decisions regarding how the data should be deployed.

A second serious drawback of links is that they are only paths. Generation of a result such as a join requires traversal of these paths by the application program. It seems superfluous to cite other difficulties with this concept.

Many relational DBMS and languages including SQL do not support the concepts of primary key and foreign key. I fail to see how these products can support the guaranteed access or the view updating rules without making the system aware of which column(s) constitute the primary key of **each base table.**

Furthermore, I fail to see how these products can support referential integrity or the view updating rule without offering clear support for both primary keys and foreign keys. For example, in SQL, the CREATE TABLE command should be extended to permit the user to declare which column or columns constitute the primary key and which constitute foreign keys. In addition, there should be a new CREATE DOMAIN command in SQL.

Rule	DB2	IDMS/R	Datacom/DB
1 Information rule	Yes	No	No
2 Guaranteed access rule	Partial	No	No
3 Systematic treatment of nulls	Partial	No	No
4 Active catalog based on resource management	Yes	No	No
5 Comprehensive data sublanguage	Yes	No	No
6 View-updating rule	No	No	No
7 High-level insert, update, delete	Yes	No	No
8 Physical data independence	Yes	Partial	Partial
9 Logical data independence	Partial	No	No
10 Integrity independence	No	No	No
11 Distribution independence	Yes	No	No
12 Nonsubversion rule	Yes	No	No
Score (1 for yes, 0 otherwise)	7	0	0

Figure 1

Fidelity

Figure 1 shows fidelity to the 12 rules by IBM's DB2, Cullinet Software, Inc.'s IDMS/R and Applied Data Research, Inc.'s Datacom/DB — examples chosen for their wide differences. These scores represent counts of compliance with each rule (score one for "yes" and zero for either "partial" or "no").

Actually, the information rule is so fundamental to the relational approach that a system's compliance with this rule should receive a much higher score than one. Weighting it as high as 10 would not be excessive. However, I shall avoid assigning different points for different features, just as I avoided a fractional score for partial support of a feature: It is too easy to be subjective in these matters.

DB2 scores quite well on the fidelity evaluation. Very few other DBMS score higher on the 12 rules, although some others score equally well. Both IDMS/R and Datacom/DB allow information to be represented in the order of records in storage and in repeating groups — directly violating the information rule. In the case of IDMS/R, information may also be represented in links between record types (CODASYL calls them "owner-member sets") and also in "areas."

Some vendors of nonrelational DBMS have quickly added a few relational features — in some, cases, very few features — in order to be able to claim their systems are relational, even though they may not meet simple requirements for being rated minimally relational.

These "born-again" systems keep their lower level languages (single record-at-a-time) open to users either to support compatibility with previously developed application programs or because the vendor takes the position that relational operators are applicable to query only.

In view of this, such systems fail to support the nonsubversion rule — a heavy penalty to pay for compatibility. IDMS/R and Datacom/DB are both born-again systems, and both fail to support the nonsubversion rule for integrity.

Features of the model

For a more detailed evaluation of DBMS, users can compare a system to the nine structural features, the 18 manipulative features and the three integrity features of the relational model. Each feature is defensible on practical as well as theoretical grounds.

The nine structural features are as follows:

S.1 Relations of assorted degrees — or equivalently tables with unnumbered rows, named columns, no positional concepts and no repeating groups.

S.2 Base tables representing the stored data.

S.3 Query tables — the result of any query is another table, which may be saved and later operated upon.

S.4 View tables — virtual tables that are represented internally by one or more relational commands, not by stored data. The defining commands are executed to the extent necessary when the view is invoked.

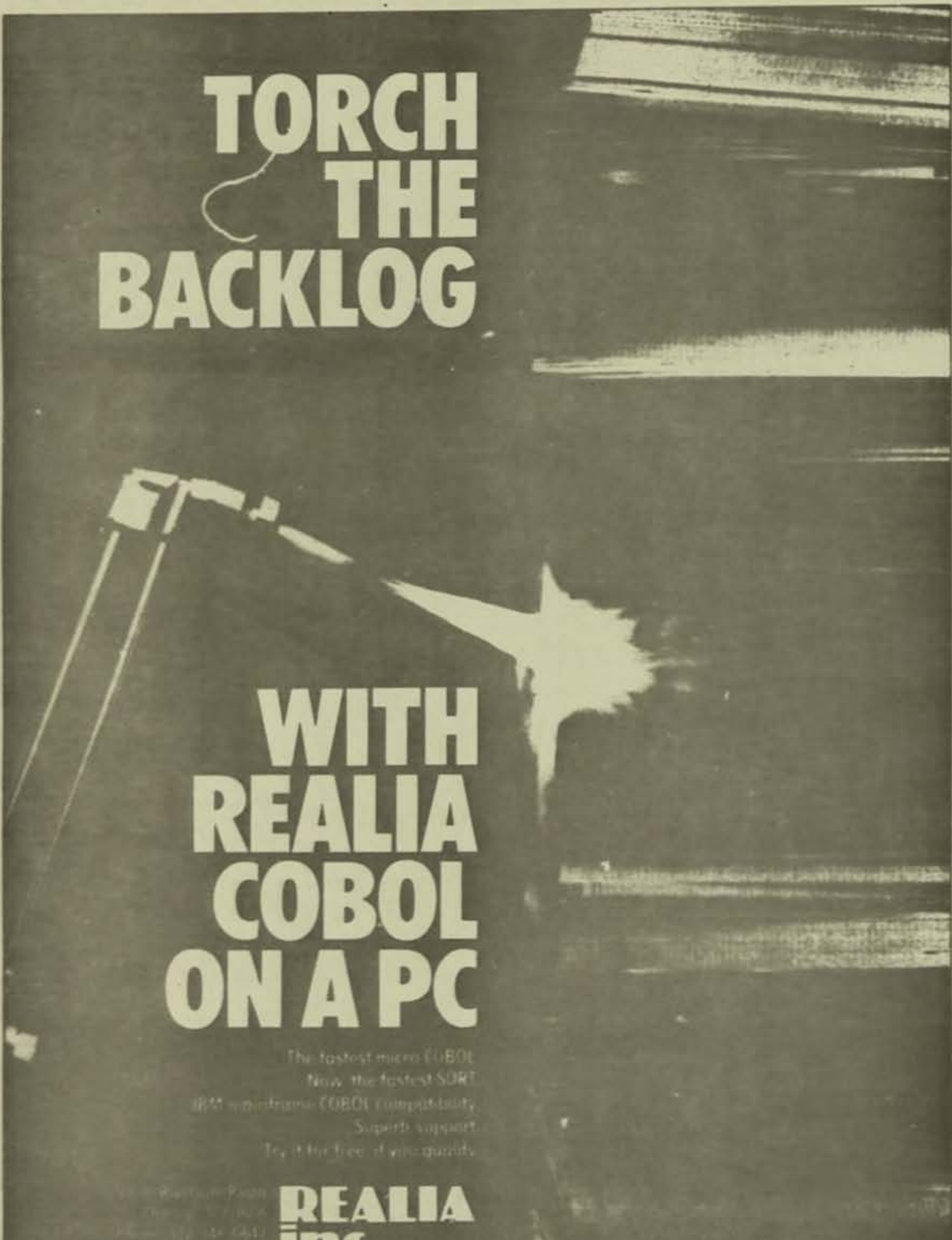
S.5 Snapshot tables — tables that are evaluated and stored in the data base, together with an entry in the catalog specifying the date and time of their creation plus a description.

S.6 Attributes — each column of each relational table is an attribute.

S.7 Domain — the set of values from which one or more columns obtain their values.

S.8 Primary key — each base table has one or more columns whose values identify each row of that table uniquely. The primary key provides the unique associative addressing property of the relational model that is implementation, software and hardware independent.

S.9 Foreign key — any column in the data base that is on the same domain as the primary key of some base relation. The foreign key serves as an important part of the support for referential integrity without introducing links into the programmer's or user's perspective.



Manipulative features

It is important to keep in mind that the relational model does not dictate the syntax of any DBMS language. Instead, it specifies the manipulative capability (that is, power) that a relational language should possess. At the same time, the model does not require the user to request the data base administrator to set up any special access paths, nor does it require the user to resort to iterative looping or recursion or Cartesian product.

The model also does not require the system to generate a Cartesian product as an intermediate result. In early papers this manipulative capability was expressed in two ways: algebraic and logic-based. The two ways were then shown to be of equal power.

This article uses the algebraic method of expressing the manipulative power, for explicative reasons.

The manipulative features are as follows:

M.1 theta select
M.2 project
M.3 theta join
M.4 outer theta join
M.5 divide
M.6 union
M.7 intersection
M.8 set difference
M.9 outer union
M.10 relational assignment

M.11 theta select maybe
M.12 theta join maybe
M.13 outer theta join maybe
M.14 divide maybe

M.15 theta select semantic override (s/o)
M.16 theta join s/o
M.17 outer theta join s/o
M.18 divide s/o

In the list above, "theta" stands for any one of the comparators: equal, not equal, greater than, less than, greater than or equal to, less than or equal to.

The integrity features of the relational model must also be followed closely:

- I.1 Entity integrity.
- I.2 Referential integrity.
- I.3 User-defined integrity.

The integrity features cited in I.3 are part of the comprehensive data sublanguage. They support the trigger and assertion approach to defining those integrity constraints that are specific to the particular data base. By contrast, I.1 and I.2 apply to all relational data bases. Examples of these extensions have been published, although not fully implemented, for both SQL and QBE.

> *Your choice of DBMS now may well determine how readily your organization adapts to changes in the future.*

A simple rating technique

To be mid-80s fully relational, a DBMS must fully support all 12 of the basic rules, as well as all nine structural, all 18 manipulative and all three integrity rules of the relational model — a total of 42 features. I use the term "mid-80s" because it is likely that there will be a few more requirements by the nineties.

To provide a simple method of rating any DBMS on its fidelity to the relational model, treat each rule or feature fully supported by that DBMS as contributing one to the overall score (otherwise the contribution is zero). Then double the total score to obtain a percentage fidelity rating for the system.

If a DBMS were to achieve a total score of 42 out of 42 (and I believe no such DBMS presently exists), add 8 points to that score before doubling it — as a reward for true fidelity. Thus its fidelity percentage would be calculated to be 100.

The resulting fidelity percentage is not highly accurate. In fact, if it falls between 10% and 90%, I would recommend rounding it to the nearest multiple of 10% in order to avoid misrepresenting the accuracy by displaying more than one significant digit.

Evaluation against the model

By today's standards, 46% is a good, but improvable, fidelity percentage. Figure 2 shows the systems DB2, IDMS/R and Datacom/DB evaluated against all 30 features of the relational model. Often the 12 rules are by themselves adequate for comparison purposes. But this more detailed evaluation of the three systems primarily serves expository purposes.

Sometimes users say of a DBMS: "Why should I worry about the degree of its fidelity to the relational model? Surely it is enough for me to know about its fidelity to the ANSI SQL standard."

Unfortunately, the ANSI standard as now proposed is quite weak. It fails to support numerous features that users really need if they are to reap all the advantages of the relational approach.

ANSI's proposed standard for relational systems functions like a convoy, which can proceed only as fast as the slowest ship. The standard is based heavily on that portion of SQL supported by several vendors.

Listing the major differences between ANSI's SQL and, as an example, the SQL implemented in IBM's DB2 shows that ANSI's SQL is even less faithful to the relational model than the vendor's SQL:

- The draft ANSI SQL does not specify catalog tables and does not allow CREATE or GRANT statements to be included in application programs. Instead, it requires a "schema" that specifies an authorization ID and a list of definitions of tables, views and privileges.
- ANSI does not support "dynamic SQL" — SQL statements that are computed at execute time.
- The set of reserved words in ANSI is significantly smaller than that in DB2.
- In ANSI, the "Unique" attribute applies to a column or combination of columns as it should, whereas in DB2 it applies to an index (which it should **not**).

The ANSI version, therefore, is inadequate as a tool for evaluating DBMS products. The remarks about DB2 apply to certain other vendors' products also.

My view of these ANSI items is as follows.

- Omitting catalog tables was a poor judgment; the catalog needs to be standardized. The ANSI version looks like a survivor of non-dynamic CODASYL.
- Failure to support dynamic SQL was another poor choice. This feature is needed and is used.
- The smaller set of reserved words places vendors with relational DBMS products that go beyond the proposed ANSI standard in a potentially difficult situation. Several vendors find themselves in this category.
- The ANSI treatment of the "Unique" attribute is good in my opinion. An index is treated by ANSI as a purely performance-oriented tool, so there are no semantic consequences of dropping one.

My main criticism of the ANSI Level 1 and Level 2 proposed standard for relational data bases is that inadequate attention is given to some very important areas. For example, the comprehensive, dual-mode data sublanguage capability that SQL (as implemented) already possesses is underemphasized. The entire range of SQL implementations from the large mainframes down to the micro is not adequately addressed.

Finally, ANSI ought to extend presently supported SQL to a version that fully supports the relational model, including distributed data base. At the very least, ANSI should generate a statement of direction adequate to permit vendors to extend the fidelity of their products without risking incompatibility with some future standard.

Extensions of SQL that provide this support now can be forecast in detail and with some reliability. Any standard adopted now should not make these extensions impossible or even difficult in the future.

> *At the very least, ANSI should generate a statement of direction to permit vendors to extend their products without risking incompatibility with some future standard.*

Three buying factors

Any buyer confronted with the decision of which DBMS to acquire should weigh three factors heavily. The first factor is the buyer's performance requirements, often expressed in terms of the number of transactions that must be executed per second. The average complexity of each transaction is also an important consideration. Only if the performance requirements are extremely severe should buyers rule out present relational DBMS products on this basis. Even then buyers should design performance tests of their own, rather than rely on vendor-designed tests or vendor-declared strategies.

The second factor is reduced costs for developing new data bases and new application programs. Relational DBMS provide significant reduction in these costs, when compared with either the CODASYL or hierarchic approaches. Fourth-generation languages are no substitute, although they may provide some additional productivity.

The third factor is protecting future investments in application programs by acquiring a DBMS with a solid theoretical foundation and reliable support for high productivity and distributability. In every case, a relational DBMS wins on factors two and three. In many cases, it can win on factor one also — in spite of all the myths about performance.

Then the question arises: Which relational DBMS? The system chosen should not only be a DBMS with a good percentage of fidelity to the relational model, but should be extensible at some future time. Ideally a good DBMS will be extended soon to provide 100% support without logically impairing the customer's investment in application programs.

Buyers should be cautious with vendors that make strong claims — claiming the system is "post relational" (especially when no definition for this term is supplied), or claiming that the DBMS choice has no importance. In fact, your choice of DBMS **now** may well determine how readily your organization adapts to changes in the future.

It is time vendors realize that all the features of the relational model are interrelated and interdependent. Missing features leave large gaps in the integrity control and usability of a DBMS implementation.

There is nothing on the horizon right now that looks strong enough and practical enough to replace the relational approach. Moreover, because the relational approach relies on such a solid theoretical foundation, its lifetime will last much longer than the CODASYL, hierarchic or tabular approaches.

I also believe that it will be much easier for relational DBMS users to convert to whatever future approach appears to be superior, for two reasons. The relational approach insists on all information being recorded explicitly. Moreover, the approach has a close tie to first-order predicate logic — a logic on which most of mathematics is based, hence a logic which can be expected to have strength, endurance and many applications. ■

[illegible] relational model

	DB2	IDMS/R	Datacom/DB
Relations	Yes	No	No
Base tables	Yes	* Yes	Yes
View tables	Yes	No	No
Query tables	Yes	No	Yes
Snapshot tables	No	No	No
Attributes	Yes	Yes	Yes
Domains	No	No	No
Primary keys	Partial	No	No
Foreign keys	No	No	No
Theta select	Yes	Yes	Yes
Project	Yes	Yes	Yes
Theta join	Yes	Partial	No
Outer theta join	No	No	No
Union	Yes	No	No
Outer union	No	No	No
Intersection	Yes	No	No
Set difference	Yes	No	No
Division	Yes	No	No
Relational assignment	Yes	No	No
Maybe theta select	Yes	No	No
Maybe theta join	Yes	No	No
Maybe outer theta join	No	No	No
Maybe divide	Yes	No	No
S/O theta select	No	No	No
S/O theta join	No	No	No
S/O outer theta join	No	No	No
S/O divide	No	No	No
Entity integrity	Partial	No	No
Referential integrity	No	Partial	No
User-defined integrity	Partial	No	No
Total score against relational model (1 for full support, 0 otherwise)	16	4	5
Total score on 12 rules	7	0	0
Grand total score	23	4	5
% fidelity	46	8	10

Figure 2