Reprinted from JOURNAL OF COMPUTER AND SYSTEM SCIENCES
All Rights Reserved by Academic Press, New York and London

Vol. 28, No. 1, February 1984
Printed in Belgium

Database Relations with Null Values

CARLO ZANIOLO

Bell Laboratories, Holmdel, New Jersey 07733

Received June 21, 1982; revised June 13, 1983

A new formal approach is proposed for modeling incomplete database information by means of null values. The basis of our approach is an interpretation of nulls which obviates the need for more than one type of null. The conceptual soundness of this approach is demonstrated by generalizing the formal framework of the relational data model to include null values. In particular, the set-theoretical properties of relations with nulls are studied and the definitions of set inclusion, set union, and set difference are generalized. A simple and efficient strategy for evaluating queries in the presence of nulls is provided. The operators of relational algebra are then generalized accordingly. Finally, the deep-rooted logical and computational problems of previous approaches are reviewed to emphasize the superior practicability of the solution.

1. INTRODUCTION

Database programmers have long recognized the convenience of using special symbols to fill in for incomplete or missing information in database records; these special symbols are commonly called *null values*. Recently, a number of formal investigations have focused on the topic of incomplete information and null values [3, 5, 9, 11, 13, 14, 16, 19, 23, 24, 26]. Along with interesting results, these works have shown that complex topical issues and problem areas remain open. In this paper we briefly review some of these issues and then concentrate on the problem of generalizing the formal framework of the relational data model to include null values.

A basic problem with null values is that they have many plausible interpretations. The ANSI/SPARC interim report, for instance, cites 14 different manifestations of nulls. Most authors, however, agree that the various manifestations of nulls can be reduced to two basic interpretations. These are:

(a) the *unknown* interpretation: a value exists but it is not known; and

(b) the *nonexistent* interpretation: a value does not exist.

A formal treatment of null values under the "unknown" interpretation was proposed by Codd [5]. This approach uses a three-valued logic which, along with the usual TRUE and FALSE, also features the additional value MAYBE. In Codd's approach a relational expression such as $X > Y$ evaluates to TRUE or FALSE in the usual fashion if neither X nor Y is null. But if either X or Y is null, then this expression evaluates to MAYBE. Thus, Codd proposes an extended relational

0022-0000/84 $3.00

algebra, where the operations of select, join, and divide come in two distinct versions: the TRUE version and the MAYBE version. For instance, the result of a TRUE selection operation contains only tuples where the selection expression evaluates to TRUE. The MAYBE version instead contains those tuples where the selection expression evaluates to MAYBE. Codd also suggests that relational query systems can be extended to enable the users to retrieve, not only those tuples which satisfy the query in the "TRUE" sense, but also those which satisfy it in the "MAYBE" sense.

There exist a number of problems in Codd's treatment of null values. A first area of practical concern is simply the high cost, for little additional information, characterizing the MAYBE versions of queries (due to their low selectivity). Therefore, most relational systems implement execution strategies which, with minor variations, correspond to Codd's TRUE version of queries [1, 29]. A second area of concern relates to logical problems. First of all, Codd's three-valued logic does not always model correctly the intended "unknown" interpretation of nulls [9]: expressions that, under the "unknown" interpretations, should always evaluate to TRUE (tautologies), evaluate instead to MAYBE.

There is also a second logical problem area not previously mentioned in the open literature. This relates to the set properties of relations and their update behavior. Say, for instance, that we have the following two instances of a parts–suppliers relation (here we follow Codd in using the symbol ω to denote the null value).

$$\begin{array}{ll} \text{PS}'(\text{P\#}, & \text{S\#}) \\ \omega & s1 \\ p1 & s2 \end{array} \tag{1.1}$$

$$\begin{array}{ll} \text{PS}''(\text{P\#}, & \text{S\#}) \\ \omega & s1 \\ p1 & s2 \\ p2 & s2. \end{array} \tag{1.2}$$

Under Codd's approach, an expression such as PS″ $\supseteq$ PS′ is evaluated using the so-called ***null substitution principle*** [5]. This replaces each occurrence of ω by a *possible distinct* nonnull value. Then, an expression which yields TRUE (FALSE) under every substitution evaluates to TRUE (FALSE) in the three-valued system. However, an expression which yields both TRUE and FALSE, depending on the values used in the substitution, evaluates to MAYBE in the three-valued system. Therefore the expression PS″ $\supseteq$ PS′ evaluates to MAYBE: For if the ω in PS′ and the ω in PS″ are both replaced by one value, say by $p1$, then the expression yields TRUE; but if the null value in PS′ is replaced by $p2$, while the null value in PS″ is replaced by $p1$, then this expression evaluates to FALSE.

Note now that PS″ can be obtained from PS′ by adding the tuple ($p2$, $s2$). Now, the everyday user, after adding in new information, expects that his new database properly ***contains*** his old information as a matter of fact (TRUE) and not of speculation (MAYBE). Indeed, whatever data he could find in the old database he can also find in the new one. Thus, the definition of set containment in Codd's three-valued logic does not model one's intuitive understanding of the dynamic behavior of an information system. Moreover, the set operations proposed in [5] do not possess even the most basic set algebraic properties, since:

$$\mathrm{PS}' \cup \mathrm{PS}'' \supseteq \mathrm{PS}'$$

and

$$\mathrm{PS}' \cap \mathrm{PS}'' \subseteq \mathrm{PS}'$$

do not evaluate to TRUE but rather to MAYBE. Even more surprisingly,

$$\mathrm{PS}' = \mathrm{PS}'$$

and

$$\mathrm{PS}' = \mathrm{PS}''$$

both evaluate to MAYBE and not, respectively, to TRUE and FALSE as expected.

Thus, the generalization of the set-theoretic properties of relations in the presence of nulls represents an open problem—and a very important one since set theory provides the bedrock on which the relational model is built and a complete relational algebra includes the operations of set union and difference [22].

Interesting notions relating to the treatment of nulls under the "unknown" interpretation have also been proposed by other authors [11, 16, 24]. We discuss them later. Our more immediate concern is to illustrate the many facets of the problem at hand. Among these we find the "nonexistent" interpretation of nulls. This problem was studied by Lien [14]. He proposes join and select operations which basically coincide with the TRUE version of Codd's operations. Then Lien proceeds by formalizing the concept of multivalued dependencies with nulls, for which he derives a complete set of inference rules.

The third facet of the null-value issue is how to deal with both the "unknown" and the "nonexistent" interpretations at once. This problem was addressed by Vassiliou [23] who notes that serious semantic problems arise if one tries to extend the three-valued logic to a four-valued one. Vassiliou then shows that the query interpretation problem in the presence of nulls of both types can be solved in the framework of Scott's denotational semantics. A drawback of this approach is the high computational cost of evaluating certain queries (Vassiliou shows that in his approach query evaluation is Co-NP complete [8]).

A final facet of the null value scenario does not address the problem of generalizing old concepts and constructs (such as query execution, relational operators, etc.) in the presence of nulls, but rather explores and pursues new conceptual tools and applications which are made possible by the use of nulls.

Generally recognized as useful, for instance, are the information-preserving joins independently introduced in [13, 25]. Also, null values have been found useful in mapping network schemas into relational schemas [15, 26, 27], in distributed databases, and in ensuring the universal relation assumption [6]. There is a need for integrating these new concepts and *ad hoc* applications in a complete and consistent framework.

This paper presents a new approach that avoids the dilemma of the "unknown" versus the "nonexistent" interpretation and provides an extension that preserves two key advantages of the relational model:

(1) its set-theoretic foundations—which are preserved through a lattice-based generalization of the relational algebra, and

(2) efficient query-evaluation algorithms based upon the well-known correspondence between the relational calculus and the relational algebra.

The paper is developed as follows: In Section 2 we introduce the notion of no-information nulls that provides the basis of our approach. In Section 3 we formally define the notion of relations with null values. In Section 4 we examine the set-theoretic properties of these relations. In Sections 5 and 6 we discuss the treatment of nulls in queries and relational operators. In Section 7 we discuss the algebraic properties of relations with nulls, and prove that they have the closure property with respect to relational algebra.

2. A New Approach

Our approach is based upon the observation that the "unknown" and the "nonexistent" interpretations do not constitute the most basic and elementary interpretations for the null value. There exists a more primitive and unpretentious interpretation underlying these two. To illustrate this point we will consider a typical application of null values. Say that a database contains a relation (or if you prefer a record type or a file) EMP with columns (attributes), E#, NAME, SEX, and MGR# (the *E#* of the employee's manager):

$$\text{EMP}(\text{E\#}, \text{NAME}, \text{SEX}, \text{MGR\#}). \tag{2.1}$$

TABLE I

The Employee Relation

EMP	(E#,	NAME,	SEX,	MGR#)
	1120	SMITH	M	2235
	4335	BROWN	F	2235
	8799	GREEN	M	1255

Say that the current content of the relation is that represented in Table I. Say now that the database administrator (anticipating future needs of the enterprise) decides to change the schema to include a new column TEL#,

$$\text{EMP}(\text{E\#}, \text{NAME}, \text{SEX}, \text{MGR\#}, \text{TEL\#}) \tag{2.2}$$

to contain the home number of each employee. This change in the schema does not imply that each employee will be requested to supply his or her telephone number at once. This piece of information will be entered in the database when it becomes available. Therefore, the database administrator is faced with the problem of having to operate, at least for the immediate future, with an expanded schema, while no change in the information content of the database has occurred. The obvious solution is to view the database as in Table II. Thus, the TEL# entries in the rows of our relations have been filled with the symbol "—" which, from now on, we use to denote a null value. Table II demonstrates a very plausible and useful usage of null values. Clearly, neither the "unknown" nor the "nonexistent" interpretation is applicable in this situation. Here the symbol "—" neither denotes that a telephone number of a given employee does not exist, nor that the telephone number exists but is not known. Here the null value simply denotes that *no information* whatsoever exists on the TEL# of an employee. Thus our null value can be regarded as a place holder for either a nonexistent or an unknown value.[1]

On the contrary, if "—" were interpreted as either "unknown" or "nonexistent," then Table II would contain more information than Table I, and this would contradict the assumption that no additional data were gathered and stored when the schema was modified. Under the "no information" interpretation of nulls, it is correct to say instead that Table I and Table II are *information-wise equivalent*. The notion of information-wise equivalence is central in our approach and will be further discussed and formally defined later. (Obviously, the above relations are equivalent in

TABLE II

The Employee Relation after the Addition of the New Attribute TEL#

EMP	(E#,	NAME,	SEX,	MGR#,	TEL#)
	1120	SMITH	M	2235	—
	4335	BROWN	F	2235	—
	8799	GREEN	M	1255	—

[1] A nonexistent value can be formalized either in a local context or in a global one [28]. Both contexts are equivalent in the sense that they produce the same end results, but the mechanics of their treatments are somewhat different. Here, we assume that our no information null is a place holder for an unknown value and for a nonexistent value interpreted in the local context.

terms of content, but the intentional information represented in the two schemas is different. For instance, if E# is a key then the functional dependency of TEL# on E# is embodied in schema (2.2) but not in (2.1).)

In summary, no viable solution is currently available for handling both the "unknown" and "nonexistent" concepts. Moreover, even if this becomes available, there will remain the problem of those situations such as the one of Table II where neither of these interpretations can correctly be assigned to the null (since an arbitrary assignment would result in non-factual information being recorded in the database).

On the contrary, if a "no information" null value is used to model every kind of missing or incomplete information, then all the information stored in the database is factual and correct. However, partial knowledge which may be available to users may be lost in the process—a price that, as discussed next, is worth paying in many situations.

Our basic argument for the soundness of the "no information" approach relies on the observation that a database can only provide an *approximation* to the real world. Different types of nulls can be assessed for the extent by which they can improve this approximation.[2] For instance, take our database schema (2.2). If no null value is allowed in the TEL# field then no information can be stored about employees who do not have a telephone, or whose telephone number is, for some reason, not available. The use of "no information" nulls allows a dramatic improvement in the accuracy and completeness of our database, since the information available about the E#, the NAME, the SEX, and the MGR# of these employees can be stored in records with null TEL# values. Admittedly, this approximation may be improved even further, if these employees are known to have or not to have telephones, by using two kinds of nulls: one to denote the nonexistence of a number, the other that the number is unknown. However, the added complexity which results from the use of several nulls will not provide the ultimate solution but only a somewhat better approximation. Let us illustrate this point with an example. A case of incomplete information, often occurring in the real world, can be expressed by a sentence such as, "Bob Smith's manager is a woman." This sentence states that although the identity of Smith's manager is not known the sex of this manager is known. To preserve this information one will have to use marked nulls [11, 17] to link together different tuples. For our EMP relation, for instance, there must be a tuple with a unique null E# where the value of SEX is F (female) and also the same unique null must appear in the MGR# of Bob Smith's record. Thus, while this marked null will be treated as a regular "unknown" when a select operation is performed, it will be treated as a regular nonnull value when performing a join on MGR#.

In practice, moreover, much is often known about an "unknown" value. For

[2] Horgan recently provided a rigorous setting to the concept of "better approximation" using a lattice where **ni** is the bottom and the "nonexisting" and "unknown" nulls are two incomparable nodes. Thus join, project and select, and various dependencies are continuous functions in the lattice of generated relations [10].

instance, although we do not know the specific color of an object, we may know that it is either red or blue. An approach to this kind of information has been described in [16]. More generally, a probability distribution for an unknown value within a domain may be either given or computable from the current database. A systematic approach to this problem is described in [24]. Moreover, additional information may be available on a specific instance of a null (e.g., although the exact age of an employee is not known, we could in fact know that he is young). This information could be preserved using a very sophisticated kind of null, but the expected improvement in the quality of information may not justify the additional complexity. In fact, when additional knowledge is important enough it is generally easier to preserve it through schema modifications than through complex null values. For instance, if it were necessary to record the fact that an employee has no telephone, the attribute NO_OF_TELS, with a value zero for this employee, could be added.

In conclusion, a database system can approximate the real world only to a certain degree of accuracy. The degree chosen for a system is a matter for practical trade-offs. In choosing to operate under the "no information" interpretation, we accept a somewhat coarser approximation, but we obtain significant benefits in return. The first obvious advantage is the generality which follows from the fact that one type of null can be used as the place-holder for every manifestation of missing or incomplete information. The second is conceptual simplicity, which leads to a simple generalization of the relational data model, as discussed later in this paper. A final advantage, as we shall also see, is computational efficiency in evaluating queries.

3. Basic Concepts

From now on, we will refer to relations with null values simply as relations. To denote relations without nulls we explicitly say "fully defined relations" or "total relations."

A relation R, defined over a set of attributes $W = \{A_1, \ldots, A_n\}$, is denoted $R(W)$. Underlying each variable attribute $A_i \in W$, there is a domain denoted $\mathrm{DOM}(A_i)$. We *extend* each domain to include the distinguished symbol **ni** which denotes the null value under the "no information" interpretation. For $A \in W$, an A-value is an assignment from the *extended A-domain.* Generalizing this notion, an X-value, where $X \subseteq W$, is an assignment of values to the attributes in X from their respective extended domains. A relation $R(W)$ is a set of W-values. The elements of this set are called rows or tuples of R.

A relation can be represented as a table, where the rows represent the tuples of the relation and the columns correspond to the attributes of the relation. In our tables we represent **ni** by the dash "—" (see Table II). Say that r is a W-value, e.g., some tuple of $R(W)$. Also, let $A \in W$ and $X \subseteq W$. Then $r[A]$ and $r[X]$, respectively, denote the A-value and the X-value of r. We will assume, without loss of generality, that all the attributes of our relations are contained in a finite universe of attributes, U. We use the first letters of the alphabet, such as A, B, and C, to denote single attributes in U, and the last letters, such as W, X, Y, and Z, to denote subsets of U.

The notion of *more informative tuple* [26] supplies the cornerstone of our approach.

DEFINITION 3.1. An X-value r is said to be *more informative* than a Y-value t, when for each $B \in Y$, if $t[B]$ is not **ni** then $B \in X$ and $r[B] = t[B]$.

Thus, r must match t in each nonnull value of t. We write $r \geqslant t$ to denote that r is more informative than t. Conversely, if $r \geqslant t$ we say that t is less informative than r and write $t \leqslant r$. If $r \geqslant t$ and $t \geqslant r$, then we say that r and t are (information-wise) equivalent and write $r \cong t$.[3] For instance, say that

$$r_1 = (5555, \text{JONES}, -, 2231), \qquad r_2 = (5555, \text{JONES}, \text{F}, 2231)$$

denote values of {E#, NAME, SEX, MGR#}, and

$$r_3 = (5555, \text{JONES}, \text{F}, 2231, -), \qquad r_4 = (5555, \text{JONES}, \text{F}, 2231, 2639452)$$

denote values of {E#, NAME, SEX, MGR#, TEL#}. Then, $r_1 \leqslant r_2, r_2 \cong r_3$ and $r_3 \leqslant r_4$. (Also, note that each tuple in Table I is equivalent to the corresponding enlarged tuple in Table II.) Let $X \subseteq Y \subseteq U$. Given an X-value r, an equivalent Y-value $t \cong r$ can be constructed from r by filling the $(Y - X)$ values with nulls. Therefore, we will prescribe by convention that, if r is an X-value and the attribute A is not in X, then $r[A] \cong \mathbf{ni}$. Therefore, any two tuples consisting only of null values are equivalent and any such tuple is equivalent to the tuple consisting of **ni**. These tuples will be called *null tuples*. A tuple without nulls will be called *total* and a tuple with a total X-value will be called *X-total*.

Say that U^* denotes the set of all possible tuples (i.e., the set containing every X-value for each $X \subseteq U$). Then, the notion of more informative, being transitive and reflexive, establishes a *quasi-ordering* of U^*[2]. A tuple t will be called a *meet* of two tuples r_1 and r_2, denoted $t \cong r_1 \wedge r_2$, when for each A in U,[4]

$$\begin{aligned} t[A] &= r_1[A] && \text{if} \quad r_1[A] = r_2[A], \\ &= \mathbf{ni} && \text{if} \quad r_1[A] \neq r_2[A]. \end{aligned}$$

Clearly, if $r_1' \cong r_1$, then $r_1 \wedge r_2 \cong r_1' \wedge r_2$. Therefore, if one does not distinguish between equivalent tuples, then there always exists the meet of any two tuples in U^*, and it is unique. The meet of r_1 and r_2 is more informative than any tuple which is less informative than both r_1 and r_2.

[3] The relationship $\geqslant$ is reflexive and transitive. Thus, "$\cong$" is an equivalence relation since it is also symmetric.

[4] Notice that, in this definition, it is immaterial whether we assume that $\mathbf{ni} = \mathbf{ni}$, or $\mathbf{ni} \neq \mathbf{ni}$.

While there exists a meet for every two tuples in U^*, a join may not exist. Two tuples r_1, r_2 will be called *joinable* when the following is true for each $A \in U$:

$$\text{If } r_1[A] \neq r_2[A], \text{ then either } r_1[A] = \mathbf{ni} \text{ or } r_2[A] = \mathbf{ni}.$$

A tuple t will be called a *join* of two tuples r_1 and r_2, denoted $t \cong r_1 \wedge r_2$, when r_1 and r_2 are joinable and for each $A \in U$,

$$\begin{aligned} t[A] &= r_1[A] \quad &\text{if} \quad r_1[A] \geqslant r_2[A], \\ &= r_2[A] \quad &\text{if} \quad r_2[A] \geqslant r_1[A]. \end{aligned}$$

Clearly, if $r_1' \cong r_1$ then r_1' and r_2 are joinable if and only if r_1 and r_2 are, and so $r_1' \vee r_2 \cong r_1 \vee r_2$ and $r_1' \wedge r_2 \cong r_1 \wedge r_2$. Often we will disregard distinctions between equivalent tuples and speak of *the join or the meet* of two tuples.[5] In this context we may say that the join (the meet) of r_1 and r_2 is the least (the most) informative among the tuples which are more (less) informative than r_1 and r_2.

4. Extended Relations

The notion of being more informative can be extended to relations, which will be said to be ***more informative than*** or to ***subsume*** other relations.

Definition 4.1. A relation R_1 subsumes a relation R_2, written $R_1 \supseteq R_2$, when for each nonnull tuple $r_2 \in R_2$ there is a tuple $r_1 \in R_1$ with $r_1 \geqslant r_2$.

This $\supseteq$ relationship is transitive and reflexive. We can now define the notion of information-wise equivalence as

Definition 4.2. The relations R_1 and R_2 are information-wise equivalent, written $R_1 \cong R_2$, when $R_1 \supseteq R_2$ and $R_2 \supseteq R_1$.

The equivalence relation $\cong$ (reflexive, symmetric, and transitive) partitions the universe of relations into disjoint subclasses. We can thus use the basic generalization mechanism used to extend natural numbers to real numbers [20] to introduce the notion of ***extended relations*** (written x-relations for short).

Definition 4.3. An x-relation is an equivalence class under $\cong$. The class of relations equivalent to R is denoted $\hat{R}$. R is called a *representation* of $\hat{R}$.

Thus $\hat{R}_1 = \hat{R}_2$ iff $R_1 \cong R_2$. If $R_1' \cong R_1$ and $R_2' \cong R_2$, then R_1' subsumes R_2' iff R_1 subsumes R_2. Therefore, we can now define the notion of set inclusion or set containment for x-relations.

Definition 4.4. $\hat{R}_1$ contains $\hat{R}_2$, written $\hat{R}_1 \supseteq \hat{R}_2$, when R_1 subsumes R_2.

[5] If one does not distinguish between equivalent tuples then the relation $\geqslant$ defines a partial ordering—actually a semilattice.

Clearly if $\hat{R}_1 \cong \hat{R}_1$, $R_2' \cong R_2$ and $\hat{R}_1 \supseteq \hat{R}_2$ then $\hat{R}_1' \supseteq \hat{R}_2'$, as expected. It also follows directly from the definitions that:

PROPOSITION 4.1. *$\hat{R}_1 = \hat{R}_2$ iff $\hat{R}_1 \supseteq \hat{R}_2$ and $\hat{R}_2 \supseteq \hat{R}_1$.*

We will also say that $\hat{R}_1$ properly contains $\hat{R}_2$, and write $\hat{R}_1 \supset \hat{R}_2$, when $\hat{R}_1 \supseteq \hat{R}_2$ but $\hat{R}_1 \neq \hat{R}_2$. The converse of $\supseteq$ and $\supset$ will be denoted by $\subseteq$ and $\subset$, as usual.

It is also convenient to generalize the notion of a tuple being an element, or a member, of an x-relation as

DEFINITION 4.5. A tuple t is said to x-belong to, or to be an x-element of $\hat{R}$ written $t \tilde{\in} \hat{R}$, when, for some R' in $\hat{R}$, $t \in R'$.

The following proposition supplies a simpler characterization of $\tilde{\in}$. (Its proof follows directly from the definition.)

PROPOSITION 4.2. *$t \tilde{\in} \hat{R}$ iff there exists a tuple $r \in R$ s.t. $r \geqslant t$.*

Thus a tuple t belongs to an x-relation iff its representation contains a tuple which is more informative than t. Also we will say that a tuple t x-belongs to a relation R, and write $t \tilde{\in} R$, to denote that for some $r \in R$, $r \geqslant t$. Although $\tilde{\in}$ is now used in two different contexts no confusion arises, since $t \tilde{\in} \hat{R}$ iff $t \tilde{\in} R$. We also write $t \tilde{\notin} R$ or $t \tilde{\notin} \hat{R}$ to denote that $\neg(t \tilde{\in} R)$ or $\neg(t \tilde{\in} \hat{R})$ holds.

Given a set of tuples $\{t_1, t_2, ..., t_n\}$, one can eliminate all tuples that are less informative than some other tuples, and enlarge the others to their equivalent U-values. The x-relation represented by the set of U-values so obtained will be denoted

$$\hat{\{t_1, t_2, ..., t_n\}}.$$

We can now define union, x-intersection, and difference using this handy notation. (The reason for the term x-intersection will become clear later.) We have

$$\textit{union:} \qquad \hat{R}_1 \cup \hat{R}_2 = \hat{\{r \mid r \tilde{\in} \hat{R}_1 \text{ or } r \tilde{\in} \hat{R}_2\}}, \tag{4.1}$$

$$x\textit{-intersection:} \qquad \hat{R}_1 \tilde{\cap} \hat{R}_2 = \hat{\{r \mid r \tilde{\in} \hat{R}_1 \text{ and } r \tilde{\in} \hat{R}_2\}}, \tag{4.2}$$

$$\textit{difference:} \qquad \hat{R}_1 - \hat{R}_2 = \hat{\{r \mid r \tilde{\in} \hat{R}_1 \text{ and } r \tilde{\notin} \hat{R}_2\}} \tag{4.3}$$

Also, it follows from the definitions that these operations have the substitution property, with respect to equality.

PROPOSITION 4.3. *If $\hat{R}_1' = \hat{R}_1$ and $\hat{R}_2' = \hat{R}_2$ then*

(1) $\hat{R}_1' \cup \hat{R}_2' = \hat{R}_1 \cup \hat{R}_2$,

(2) $\hat{R}_1' \tilde{\cap} \hat{R}_2' = \hat{R}_1 \tilde{\cap} \hat{R}_2$,

(3) $\hat{R}_1' - \hat{R}_2' = \hat{R}_1 - \hat{R}_2$.

The union and the x-intersection, respectively, define the least upper bound and the greatest lower bound with respect to the partial ordering $\supseteq$. In fact:

PROPOSITION 4.4. *If $\hat{R} \supseteq \hat{R}_1$ and $\hat{R} \supseteq \hat{R}_2$ then $\hat{R} \supseteq \hat{R}_1 \cup \hat{R}_2$.*

Proof. If $r \tilde{\in} \hat{R}_1 \cup \hat{R}_2$ then $r \tilde{\in} \hat{R}_1$ or $r \tilde{\in} \hat{R}_2$. Say that $r \tilde{\in} \hat{R}_1$. Then $r \tilde{\in} R \supseteq \hat{R}_1$. Thus, $\hat{R} \supseteq \hat{R}_1 \cup \hat{R}_2$.

PROPOSITION 4.5. *If $\hat{R} \subseteq \hat{R}_1$ and $\hat{R} \subseteq \hat{R}_2$ then $\hat{R} \subseteq \hat{R}_1 \cap \hat{R}_2$.*

Proof. Easy.

Thus we have a lattice of x-relations, with the well known properties associated with it [2]. This lattice is also *distributive* since

$$\hat{R}_1 \tilde{\cap} (\hat{R}_2 \cup \hat{R}_3) = (\hat{R}_1 \tilde{\cap} \hat{R}_2) \cup (\hat{R}_1 \tilde{\cap} \hat{R}_3) \tag{4.4}$$

and

$$\hat{R}_1 \cup (\hat{R}_2 \tilde{\cap} \hat{R}_3) = (\hat{R}_1 \cup \hat{R}_2) \tilde{\cap} (\hat{R}_1 \cup \hat{R}_3). \tag{4.5}$$

The proof of these properties follows from the definitions and is left to our reader. (Only one of the above needs to be proven since the validity of either one implies the validity of the other in a lattice.)

Our lattice has a bottom element, denoted $\hat{\varnothing}$ which is characterized by the property that for every x-relation $\hat{R}$, $\hat{R} \tilde{\cap} \hat{\varnothing} = \hat{\varnothing}$. $\hat{\varnothing}$ can be represented by an empty relation. The top of our lattice, denoted $\hat{\text{TOP}}_U$, is characterized by the property that $\hat{R} \cup \hat{\text{TOP}}_U = \hat{\text{TOP}}_U$, for all $\hat{R}$. If $U = \{A_1, ..., A_p\}$ then $\hat{\text{TOP}}_U$ can be represented by

$$\text{TOP}_U = \text{DOM}(A_1) \times \cdots \times \text{DOM}(A_p).$$

(Note that TOP_U is a proper subset of U^*.)

In general, an x-relation $\hat{R}$ does not have a complement (i.e., there is no relation $\hat{R}'$ for which $\hat{R} \cap \hat{R}' = \hat{\varnothing}$ and $\hat{R} \cup \hat{R}' = \hat{\text{TOP}}_U$). This can be seen from the following example:

$$U = \{A, B\},\ \text{DOM}(A) = \{a_1\},\ \text{DOM}(B) = \{b_1, b_2\}.$$

Thus the following two tuples are x-elements of $\hat{\text{TOP}}_U$:

$$r_1 : (a_1, b_1), \qquad r_2 : (a_1, b_2).$$

Now take a relation R x-containing r_1 but not r_2. Then an x-relation $\hat{R}'$, to yield $\hat{R} \cup \hat{R}' = \hat{\text{TOP}}_U$, must have r_2 as an x-element: $r_2 \tilde{\in} \hat{R}'$. But then the tuple $(a_1, —)$ x-belongs to both $\hat{R}$ and $\hat{R}'$. Therefore it also belongs to $\hat{R} \tilde{\cap} \hat{R}' \neq \hat{\varnothing}$.

Likewise, the intersection $(\hat{R}_1 - \hat{R}_2) \tilde{\cap} \hat{R}_2$ may not be empty. However, the following two properties hold:

PROPOSITION 4.6. *For any two x-relations $\hat{R}_1$ and $\hat{R}_2$, where $\hat{R}_1 \supseteq \hat{R}_2$,*

$$(\hat{R}_1 - \hat{R}_2) \cup \hat{R}_2 = \hat{R}_1.$$

PROPOSITION 4.7. *If $\hat{R} \cup \hat{R}_2 = \hat{R}_1$ then $\hat{R} \supseteq (\hat{R}_1 - \hat{R}_2)$.*

Thus $(\hat{R}_1 - \hat{R}_2)$ is the smallest x-relation (in terms of $\subseteq$, of course) whose union with $\hat{R}_2$ will give $\hat{R}_1$. The proofs of these propositions follow immediately from the definitions.

The notion of *minimal representation* is convenient for representing and handling x-relations.

DEFINITION 4.6. A relation R constitutes a minimal representation for $\hat{R}$, when no proper subset of R is also a representation of $\hat{R}$.

A minimal representation can be constructed by starting with an arbitrary one and removing the null tuple, if present, along with every tuple which is less informative than some other tuple. This process can be regarded as an extension of the one of removing duplicate tuples in tables representing conventional relations.

The minimal representation of an x-relation over a given attribute set is unique. As shown by examples (2.1) and (2.2), however, an x-relation can have two distinct minimal representations over two different sets of attributes. To introduce the notion of a minimal attribute-set, we will define the notion of *scope.*

DEFINITION 4.7. The set of attributes W is said to be the scope of $\hat{R}$, when $\hat{R}$ can be represented by a relation on W but cannot be represented by any relation with an attribute set smaller than W.

Definitions (4.1)–(4.3), in their present form, are not conducive to efficient implementation. In fact, the definition of $\tilde{\in}$ suggests a combinatorial explosion in which a plethora of less informative tuples are tested and possibly included in the result relation. This problem can be solved by deriving equivalent formulations which do not use $\tilde{\in}$. For instance, the following three can easily be derived from (4.1)–(4.3)

$$\hat{R}_1 \cup \hat{R}_2 = \widehat{\{r \mid r \in R_1 \text{ or } r \in R_2\}}, \tag{4.6}$$

$$\hat{R}_1 \tilde{\cap} \hat{R}_2 = \widehat{\{r_1 \wedge r_2 \mid r_1 \in R_1 \text{ and } r_2 \in R_2\}}, \tag{4.7}$$

$$\hat{R}_1 - \hat{R}_2 = \widehat{\{r \mid r \in R_1 \text{ and } \forall t \in R_2 : \neg(t \geqslant r)\}}. \tag{4.8}$$

Therefore, the scope of a union is the union of the scopes of its operands; the scope of an x-intersection is not larger than the intersection of the scopes of its operands; the scope of a difference is not larger than the scope of the minuend.

A simple-minded implementation of (4.6) yields a running time of order $|R_1| + |R_2|$ while (4.7) and (4.8) suggest an upper bound of order $|R_1| \times |R_2|$. However, more sophisticated techniques, such as combinatorial hashing [12], can provide more efficient solutions. These techniques are also useful for reducing relations to minimal form. For instance, if R_1 and R_2 are minimal representations for $\hat{R}_1$ and $\hat{R}_2$, then the result of (4.8) supplies a minimal representation for $\hat{R}_1 - \hat{R}_2$ (indeed a subset of a minimal representation is always minimal). However, even if R_1 and R_2 are minimal, the application of (4.6) or (4.7) may introduce less informative tuples, which will have to be eliminated to reduce the relations to minimal form.

5. Query Evaluation

As noted in [16], if a query Q is formulated on a database with incomplete information, then there are two important bounds of interest:

(1) A *lower bound* $\|Q\|_*$: the set of objects which, on the *basis of the available information*, can be concluded to satisfy Q, *for sure*, and

(2) An *upper bound* $\|Q\|^*$: the set of objects which *may possibly* satisfy Q (i.e., on the basis of the available information, they cannot be ruled out).

In this paper we are interested in the problem of evaluating the lower bound $\|Q\|_*$ for a language based upon relational calculus or relational algebra. This is the bound of more direct interest in real-life situations. The evaluation of the upper bound $\|Q\|^*$ is of less practical interest and also the source of some difficult problems which will be treated in future reports.[6] The solution here proposed is similar to Codd's solution, since it employs a three-valued logic. However, it uses a different interpretation of this logic and a new treatment of sets.

Predicate calculus based languages contain simple relational expressions such as

$$t.A \ \theta \ m.B$$

$$t.A \ \theta \ k,$$

where t and m are tuple variables, A and B are attributes, k is a (nonnull) constant, and θ is one of the comparison operators, $>, <, = \geqslant, \leqslant, \neq$. If the A-value of t is null then these two relational expressions evaluate to **ni**. Also if the $m.B$ value is null then $t.A \ \theta \ m.B$ evaluates to **ni**. Otherwise these expressions evaluate to TRUE or FALSE as usual. Boolean expressions combining relational expressions like the above are evaluated according to Table III.

The lower bound $\|Q\|_*$under the **ni** interpretation is computed by selecting only those tuples which evaluate to TRUE. Tuples which evaluate to FALSE or **ni** are discarded.

The three-valued logic and method of query evaluation described above are equivalent to Codd's TRUE-evaluation strategy. It has been shown that this strategy does not produce the correct lower bound for the "unknown" interpretation, for

[6] These problems result from the difficulty of preserving the *closed world assumption* [18] when dealing with incomplete databases [3].

TABLE III

Three-Valued Logic Tables

OR	T	F	**ni**
T	T	T	T
F	T	F	**ni**
ni	T	**ni**	**ni**

AND	T	F	**ni**
T	T	F	**ni**
F	F	F	F
ni	**ni**	F	**ni**

	NOT
T	F
F	T
ni	**ni**

queries which correspond to tautologies [9]. Fortunately the **ni** interpretation avoids this problem. To illustrate this point let us consider the issue of tautologies in more detail. Take for instance the QUEL [21] query of Fig. 1,

Q_A: **range of** *e* is EMP
retrieve (*e*.NAME, *e*.E#)
where ($e.\text{SEX} = \text{“F”} \wedge e.\text{TEL\#} > 2634000$)
$\vee$ ($e.\text{TEL\#} \leqslant 2634000$)

FIG. 1. In EMP find the NAME and *E#* of all female employees with TEL# > 2634000 and all employees with TEL# ⩽ 2634000.

Since a null value is a place-holder for another value, the correct strategy, for deciding whether a tuple satisfies a **where** expression, consists in substituting for each null in the tuples under consideration all values *which do not violate the integrity constraints* of the schema. If, under every possible substitution, the **where** clause evaluates to TRUE then it must be included when constructing $\|Q_A\|_*$. Otherwise, it must be discarded. Now, consider query Q_A of Fig. 1, and the second tuple in Table II,

(4335, BROWN, F, 2235, —).

If the null value "—" is interpreted as the place holder of an existing although unknown TEL#, then it is clear that whatever number we substitute for "—" the **where** clause of Q_A evaluates to TRUE. Thus under the "unknown" interpretation, (4335, BROWN) should be included in $\|Q_A\|_*$. Under the **ni** interpretation, however, the null value fills in for both unknown and nonexistent values. Now, in conformity with [15, 23], we assume that a nonexistent value does not satisfy any relational expression (i.e., one that involves a comparison operator, such as the three of Fig. 1).[7] Therefore a TEL# which does not exist is neither greater than 2634000, nor smaller than, nor equal to it. Thus, EMP tuples having a null TEL# cannot be

[7] The rational behind this policy is that a nonexistent value is outside the domain where valued-based comparison operators are defined. Also, it leads to a consistent and complete formal framework for the treatment of nonexistent values [28].

included in the lower bound $\|Q_A\|_*$. Therefore, the **ni** interpretation of nulls avoids the need for detecting tautologies in queries—a problem which besets the unknown interpretation.[8] From the practical viewpoint, this constitutes an important advantage of the **ni** interpretation: as we show in the Appendix detecting tautologies in queries represents an inordinately difficult and complex problem for any database system.

We can now define the operation of selection in conformity with the query interpretation discipline just discussed. The *selection operation* comes in the two flavors,

$$\hat{R}[A\theta B] \qquad \text{and} \qquad \hat{R}[A\theta k],$$

where A and B are two attributes in U from the same underlying domain, k is a constant from DOM(A)—not the **ni** symbol—and θ denotes a relational operator such as $=$, $>$, etc. The definitions of these two operations for x-relations are

$$\hat{R}[A\theta B] = \hat{\{}r \mid r \in R \text{ is } A\text{-total and } B\text{-total and } r[A]\theta r[B]\hat{\}}, \tag{5.1}$$

$$\hat{R}[A\theta k] = \hat{\{}r \mid r \in R \text{ is } A\text{-total and } r[A]\theta k\hat{\}}. \tag{5.2}$$

The *cartesian product* of two relations $\hat{R}_1$ and $\hat{R}_2$ is defined as

$$\hat{R}_1 \times R_2 = \hat{\{}r_1 \vee r_2 \mid r_1 \in R_1 \text{ and } r_2 \in R_2 \text{ are not null}\hat{\}}. \tag{5.3}$$

As in the case of total relations, the various θ-joins can thus be defined as selections on the cartesian product,

$$\hat{R}_1[A\theta B]\hat{R}_2 = (\hat{R}_1 \times \hat{R}_2)[A\theta B]. \tag{5.4}$$

In the case of equijoins one need not repeat the join columns. This lead to the definition of the *join on* X of $\hat{R}_1$ and R_2, denoted $R_1 \langle \cdot X \rangle R_2$, as

$$\hat{R}_1 \langle \cdot X \rangle \hat{R}_2 = \hat{\{}r_1 \vee r_2 \mid r_1 \in R_1, r_2 \in R_2 \text{ are } X\text{-total}\hat{\}}.$$

When both operands of a selection or a join operation are in minimal form then the results calculated according to (5.1)–(5.4) are in minimal form as well. This convenient property does not generalize to the projection and union–join operations discussed next.

The projection of a relation $\hat{R}$ on a set of attributes X, denoted $R[X]$, is defined as

$$\hat{R}[X] = \hat{\{}r[X] \mid r \in R\hat{\}}. \tag{5.5}$$

It was first noted in [25], and independently in [13], that the use of null values allows the definition of new information preserving joins. These have been called or-joins [25], extended joins [13], and also outer joins [5]. As we will see next, the

[8] In [28] we show that, in addition to these propositional-calculus tautologies, those tautologies that can occur in the more general framework of Relational Calculus (with quantifiers) are avoided as well.

name ***union–join*** best fits their nature. Indeed, the union–join on X of $\hat{R}_1$ and $\hat{R}_2$, denoted $\hat{R}_1\langle *X\rangle\hat{R}_2$, is defined as

$$\hat{R}_1\langle *X\rangle\hat{R}_2=\hat{R}_1\langle\cdot X\rangle\hat{R}_2\cup\hat{R}_1\cup\hat{R}_2.$$

Thus the union–join contains those tuples of the joined relations that do not participate in the join.

In passing, we note that the concept of natural join does not find an obvious extension in this framework. The fact that x-relations are not explicitly associated with a set of attributes represents a first source of difficulties. A second one is that both equijoins and union–joins are candidate as the basis for such an extension—each having some, but not all, of the desirable properties.

As our reader may have observed in the previous definitions, we have used the operator $\in$ rather than $\tilde{\in}$. The inconsistency here is only apparent, since the replacement of "$\in$" by "$\tilde{\in}$" in all the formulas above yields relations which are information-wise equivalent to the originals. It is also easy to see that x-relations have the equality substitution property with respect to the operators above, as expected.

6. Universal Quantifiers and Negation

The operation of division need not be considered to obtain a complete relational algebra since it is derivable from cartesian product, difference and projection [22]. Yet it deserves explicit consideration because it supplies the gateway to a correct treatment of universal quantifiers in a world of incomplete information.

Let $\hat{R}$ and $\hat{S}$ be x-relations and let R_Y denote the set of Y-total tuples of R. The Y-*quotient* of $\hat{R}$ divided by $\hat{S}$ is defined as

$$\hat{R}\langle\div Y\rangle\hat{S}=\hat{R}[Y]-(\hat{R}[Y]\times\hat{S}-\hat{R}_Y)[Y]. \tag{6.1}$$

For total relations this reduces to the usual definition of division. From this definition we have that tuples which are not Y-total do not contribute to the quotient. Thus we can also write

$$\hat{R}\langle\div Y\rangle\hat{S}=\hat{R}_Y[Y]-(\hat{R}_Y[Y]\times\hat{S}-\hat{R}_Y)[Y]. \tag{6.2}$$

The only case of practical interest is when the scopes of $\hat{R}[Y]$ and $\hat{S}$ are disjoint. In this case, the following equivalent definition of division can be obtained from (6.2):

$$\hat{R}\langle\div Y\rangle\hat{S}=\widehat{\{y\mid y \text{ is } Y\text{-total and } \forall z\,\tilde{\in}\,\hat{S},\, y\vee z\,\tilde{\in}\,\hat{R}\}}. \tag{6.3}$$

A third equivalent characterization of division can be derived from (6.3) by letting $\hat{Z}_R(y)$ be the Z-image of the Y-value y, under $\hat{R}$,

$$\hat{Z}_R(y)=\widehat{\{z\mid\exists r\,\tilde{\in}\,\hat{R}: r[Y]=y \text{ and } r[Z]=z\}}. \tag{6.4}$$

Then we have that

$$\hat{R}\langle \div Y\rangle \hat{S} = \widehat{\{y \mid y \text{ is } Y\text{-total and } \hat{S} \subseteq \hat{Z}_R(y)\}}. \tag{6.5}$$

Thus, the operation just defined constitutes a natural extension of the division operation for total relations. To better understand the properties of this operation let us consider the PARTS–SUPPLIERS relation of display (6.1). To enable an easier comparison with Codd's approach we have not eliminated less informative tuples. Display (6.6) shows a sample PARTS–SUPPLIER relation,

PS (6.6)

S#	P#
$s1$	$p1$
$s1$	$p2$
$s1$	—
$s2$	$p1$
$s2$	—
$s3$	—
$s4$	$p4$.

Consider the

Q. Find each supplier who supplies every part supplied by $s2$.

The answer to this query can be computed as

$$\hat{A} = \widehat{\text{PS}}\langle \div \text{S\#}\rangle \hat{\text{P}}_{s2}. \tag{6.7}$$

Where P_{s2} denotes the P#-image of $s2$ constructed by a selection followed by a projection as

$$\hat{\text{P}}_{s2} = \widehat{\text{PS}}[\text{S\#} = s2][\text{P\#}]. \tag{6.8}$$

We can now compare the results under the previous definition of division against those under Codd's TRUE and MAYBE version of the this operator. The application of (6.8) to (6.6) yields the following result under the TRUE version of selection:

$$\text{P}_{s2} = \{p1, -\}. \tag{6.9}$$

The MAYBE version returns the empty set.

Our definition produces the corresponding result: $\hat{\text{P}}_{s2}$. However, Codd's TRUE evaluation of (6.7) now returns,

$$A_1 = \varnothing \qquad \text{(i.e., no supplier).}$$

Codd's MAYBE evaluation produces

$$A_2 = \{s1, s2, s3\}.$$

Instead, using our definition of division we obtain

$$\hat{A}_3 = \hat{\{}s1, s2\hat{\}}.$$

Thus, Codd's TRUE-evaluation implements the following reformulation of Q:

Q_1. Find every supplier who, *for sure*, supplies every part which *may* be supplied by $s2$.

Codd's MAYBE interpretation corresponds to the following reformulation of Q:

Q_2: Find every supplier who *may be* supplying every part supplied *for sure* by $s2$.

Finally, our proposed evaluation corresponds to the following version of Q:

Q_3. Find every supplier who, *for sure*, supplies every part supplied *for sure* by $s2$.

These examples bring into the open an important issue regarding the meaning of the universal quantifier, and the set formation process specified therewith, in the presence of null values. The unanimous consensus of previous researchers on this topic is that queries such as:

"Find *all* the employees who earn more than \$20*k*,"

"Find *every* supplier who supplies red parts,"

become ambiguous when dealing with incomplete information. One must accompany the words *all* and *every* by quantifiers such as "for sure" and "maybe" [5] or by a specification such as "with more than 50% probability" [24]. In queries such as Q, and in general those involving divisions and universal quantifiers, the set formation process specified by the word "all" or "every" occurs more than once. We have elected to be consistent and to interpret all the occurrences of the words "all" and "every" in the "for sure" sense. This consistent policy is simple for the user to understand and for the system to support (since it eliminates the difficult problem of computing upper bounds). It also avoids the following paradox which besets Codd's treatment of division and universal quantifiers: Since $A_1(S\#) = \varnothing$ one must conclude that

"For sure, $s2$ does not supply all the parts $s2$ supplies."

(Note that this contradiction then arises under any interpretation of nulls.)

The difference operator also implies a universal quantification as described by (4.8). Thus, for instance, a query such as:

Q_4. Find all parts supplied by $s1$ but not by $s2$.

can be computed as

$$\hat{R}_4 = \widehat{\mathrm{PS}}[\mathrm{S\#} = s1][\mathrm{P\#}] - \widehat{\mathrm{PS}}[\mathrm{S\#} = s2][\mathrm{P\#}]. \tag{6.6}$$

Clearly the result is $\hat{R}_4 = \widehat{\{p2\}}$. This corresponds to the set of parts that are supplied, *for sure*, by $s1$, and that are not among those supplied *for sure* by $s2$.

7. A Generalization of the Relational Model

In the early seventies Codd laid the foundations of relational database theory. His main contribution [4] was the introduction of the data type *relation* with a *complete* set of relational operators (the relational algebra) to model databases and query and update operations on databases mathematically. Codd's notion of completeness was based upon the equivalence, that he proved to exist, between the expressive power of relational algebra and relational calculus. A complete relational algebra consists of the following operations [4, 22]: union, difference, selection, cartesian product, and projection. The relational calculus is a generalized version of predicate calculus from which most relational data manipulation languages evolved.

Database updates also find a precise definition in terms of the relational algebra. The result of adding a set of tuples to a relation is defined as the union of the set with the relation; likewise deletion is defined by set difference; a modification can be viewed as a deletion followed by an addition.

In the previous sections we have extended the traditional relations (let us call them Codd relations) to model incomplete information through the use of null values. The objective of this section is to prove that our extension (1) is correct and (2) completes the relational model with respect to the operators of relational algebra. For this purpose our reader should refer to the analogous problem of extending natural numbers to real numbers, which is discussed in most textbooks on algebra (e.g., [20]). A step in this generalization is the definition of rational numbers from integers. Rational numbers are defined as *equivalence classes* of integer pairs (the pair a_1/b_1 being equivalent to the pair a_2/b_2 iff $a_1b_2 = a_2b_1$). Say that $\mathbf{Z}$ denotes the set of all integers and $\mathbf{K}$ the set of all rational numbers. To prove correctness one only needs to show that for a subset $\mathbf{K}_z \subseteq \mathbf{K}$, there exists a one-to-one correspondence between $\mathbf{Z}$ and $\mathbf{K}_z$ which preserves the operations on $\mathbf{Z}$, i.e., preserves addition, subtraction, multiplication, and division, and also preserves order. This makes it totally immaterial whether one operates on $\mathbf{Z}$ or on the corresponding elements of $\mathbf{K}_z$, thus ensuring the correctness of the extension. The importance of the generalization to rational numbers follows from the fact that they complete the number system with respect to the four arithmetic operators. In fact while rational numbers have the closure property with respect to all four operators, integers do not have the closure property with respect to division.

In Section 4 of this paper we have defined an x-relation $\hat{R}$ to be the equivalence class under $\cong$ which contains R as an element. Say that $R(W)$ is a traditional relation without nulls—let us call it a Codd relation. Then $\hat{R}(W)$ is a total x-relation with scope W. Moreover two distinct Codd relations map into two distinct total x-relations. Thus there exists a one-to-one correspondence between Codd relations and total x-relations. This correspondence preserves all the operators of the complete relational algebra: union, difference, cartesian product, selection, projection. To verify this, one only needs to recall the conditions under which the relational operators are defined for Codd relations and to apply definitions (4.1), (4.3), (5.1), (5.2), and (5.5) and the definition of cartesian product to conclude that:

(1) if R_1 and R_2 are union-compatible Codd relations, and

$$\text{if} \quad R_1 \cup R_2 = R_3, \quad \text{then} \quad \hat{R}_1 \cup \hat{R}_2 = \hat{R}_3,$$

and

$$\text{if} \quad R_1 - R_2 = R_4, \qquad \text{then} \quad \hat{R}_1 - \hat{R}_2 = \hat{R}_4,$$

and

$$\text{if} \quad R_1 \supseteq R_2, \quad \text{then} \quad \hat{R}_1 \supseteq \hat{R}_2;$$

(2) if R_1 and R_2 are Codd relations and

$$R_1 \times R_2 = R_3, \text{ then } \hat{R}_1 \times \hat{R}_2 = \hat{R}_3;$$

(3) if A is an attribute of a Codd relation R, and if

$$R[A\theta k] = R_1, \text{ then } \hat{R}[A\theta k] = \hat{R}_1;$$

(4) if A and B are attributes of a Codd relation R, and if

$$R[A\theta B] = R_1, \text{ then } \hat{R}[A\theta B] = \hat{R}_1;$$

and

(5) if W is a subset of the attributes of a Codd relation R, and if

$$R[W] = R_1, \text{ then } \hat{R}[W] = \hat{R}_1.$$

In conclusion, one can operate on the realm of total x-relations instead of operating upon Codd relations, for all situations in which operations on the latter are defined.[9] However, operations on Codd relations are defined only if their attribute sets satisfy conditions (1)–(5). Not so for x-relations, as our reader can verify by referring back to the definitions (4.1), (4.3), and (5.1)–(5.5): x-relations have the closure property

[9] Therefore, it is correct to use the same notation to denote the corresponding operands for x-relations and Codd relations. A similar conclusion applies to the $\supseteq$ notation inasmuch as, when R and S are union-compatible Codd relations, then $R \supseteq S$ iff $\hat{R} \supseteq \hat{S}$.

with respect to union, difference, cartesian product, selection and projection. Therefore the extension of Codd relations to x-relations *completes* the *data-type relation with respect to the relational algebra.*

For the reasons given above, x-relations are of great interest for database management systems. Moreover they provide an interesting example of generalization in basic set theory. Unlike sets, they do not constitute a Boolean algebra. Rather they constitute a distributive, pseudo-complemented lattice [2], where the pseudo-complement of $\hat{R}$, denoted $\hat{R}^*$, is defined (U being the universe of attributes of discourse) as

$$\hat{R}^* = \hat{\mathrm{TOP}}_U - \hat{R}. \tag{7.1}$$

Thus, $\hat{R}^*$ is the smallest x-relation which when unioned with $\hat{R}$ gives $\hat{\mathrm{TOP}}_U$. Pseudo-complemented, distributive lattices are also known as ***Brouwerian*** lattices after Brouwer and Heyting (1930), who characterized an important generalization of Boolean algebra having very similar properties [2, pp. 45, 128, 138, 281].[10] Brouwerian lattices have many interesting properties [2, 7]. In particular it is known that the pseudo-complements of such a lattice form a Boolean lattice. In our case the set $\{\hat{R}^*\}$ is simply the family of total x-relations with scope U, the universe. It is also known that every Brouwerian lattice (our x-relations) and the Boolean lattice of its pseudo-complements share the join, i.e., the union, and the (pseudo)-complement operation. However, the proof that the two may have two different meet operations was published in its full generality only in 1962 [7]. Now, x-relations supply a most interesting example of such a difference: The meet for the complements (U-total x-relations) is the usual set intersection while the meet for x-relations is the x-intersection (4.2). Obviously these two are different, as illustrated by the simple case of the two x-relations on the universe $U = \{A, B\}$,

$$\hat{R}_1 = \hat{\{}(a, b_1)\hat{\}}, \qquad \hat{R}_2 = \hat{\{}(a, b_2)\hat{\}}.$$

Here the set intersection of R_1 and R_2 is empty while the x-intersection of $\hat{R}_1$ and $\hat{R}_2$ x-contains the tuple $(a, -)$.

8. Conclusions

Database systems are designed to store large amounts of real-world knowledge and to answer questions on the basis of this knowledge. However, unlike the knowledge-based question–answering systems of AI, database systems do not attempt to preserve the boundless thesaurus of real-world knowledge as it is structured in human minds and communicated through the rich nuances of natural languages. Databases can approximate this complex and boundless thesaurus only in a very limited and

[10] More precisely x-relations form the dual of a Brouwerian lattice, where the pseudo-complement of an element a is usually defined as the largest element a^* for which $a \wedge a^* =$ bottom.

imperfect way. All this being well understood, database users have long accepted reasonable limits on the scope and sophistication with which their databases can model the real world. In return, they demand systems which perform correctly and efficiently, and are simple to understand and to use.

The solution proposed in this paper ensures logical simplicity and correctness combined with computational efficiency. In particular, it extends the set-theoretic foundations of the relational model, and guarantees efficient query-evaluation algorithms through the well-known correspondence between the relational calculus and the relational algebra.

This has been accomplished by using the **ni** interpretation of nulls, which is capable, although imperfectly, of modeling and retaining incomplete real-world knowledge. This **ni** interpretation avoids the serious computational problems which occur when even a slightly more accurate approximation, such as the "unknown" interpretation, is used. Thus, we suggest that our approach is of superior practicability in many real-life situations.

In this paper we have proven the theoretical soundness of the **ni** approach. We have seen that this allows the definition of informationwise equivalence on relations with arbitrary attribute sets. Then we have introduced the concept of extended relations as classes of informationwise equivalent relations, and generalized the operators of the complete relational algebra to apply to extended relations. Finally, we have shown that the proposed generalization is correct and completes database relations with respect to relational algebra. Therefore, the approach is theoretically sound and practical and avoids many of the complexities and inconsistencies presented by other approaches.

However, there remain many problem areas that require further research. An investigation of practical interest is to derive a taxonomy of null values and understand their relative tradeoffs, and two approaches to this problem are presented in [10, 11]. In particular, one would like to know to which extent the **ni** interpretation is adequate in "real life" applications, and study those that require more informative interpretations.

An important topic not addressed by this paper is that of data dependencies and formal approaches to schema design. Basic constraints, such as uniqueness of keys and referential integrity, can be extended and enforced in the presence of null values, without major problems [5]. However, at the time of this writing, we do not know of any generalization of concepts such as functional or multivalued dependencies, which preserves all the properties that makes them so useful in the formal analysis and design of relational schemas. This fact, combined with the lack of a satisfactory generalization for the notion of natural joins, suggests that much more work is needed before a clear understanding—perhaps a solution—is reached on this complex topic.

APPENDIX: THE TAUTOLOGY PROBLEM FOR UNKNOWN NULLS

In this Appendix we discuss the difficult problem of dealing with tautologies in executing queries under the "unknown" interpretation of null values.

For concreteness we will use the language QUEL as the syntactic framework for our discussion. This can, however, be easily generalized to queries expressed in other relational query languages. Consider the queries of Fig. 1 and 2. As one can see, a query statement consists of a **range** clause which identifies a set of tuple variables, a **retrieve** clause which identifies the target list, and a **where** clause which gives the qualifying conditions on the **range** variables. Say that a query Q has the **range** variables $r_1,..., r_n$ with respective ranges $R_1,..., R_n$. To answer a query on total relations one only needs to consider all tuple occurrences $r_1 \in R_1,..., r_n \in R_n$ in all possible combinations (i.e., the cartesian product of the **range** relations) and to test whether they satisfy the **where** clause. If so, they contribute to the final result as per the target list; otherwise they do not.

For relations with null values, each null occurrence must be assigned all *legal* nonnull values. A value is legal when it does not *violate the integrity* constraints expressed by the schema. If, under every legal assignment, the **where** clause evaluates to TRUE, we say that the set of tuples under consideration *defines a tautology* (for the query Q). Tuples which define a tautology must be included in the computation of $\|Q\|_*$ under the *"unknown" interpretation of nulls.* All the remaining must be excluded.

Therefore, for the correct execution of queries under the "unknown" interpretation we need to decide whether a set of tuples defines a tautology by taking into account the query and the database schema as well. The brute force approach consists in replacing null values in tuples with all possible nonnull values, within the integrity constraints given in the schema. Since the cardinalities of domains underlying the null attributes are usually very large, and tuples often contain several null values, this approach is not feasible in general. The alternative approach is the symbolic evaluation of the **where** clause in the query expression. For instance, for the query Q_A of Fig. 1, the system could start by recognizing that the two conditions involving TEL# are logical complements of each other, and then proceed by detecting that the resulting Boolean expression is a tautology. However, even in the simple framework of propositional logic, the detection of tautologies is *NP*-hard [8]. Moreover, this is only a very benign situation. For instance, consider a somewhat more complex situation involving the domain variables *t.A* and *t.B*,

$$\textbf{where} \qquad t.A > 3 \wedge (t.B < 12 \vee t.B > t.A).$$

Here every tuple t which has a non-null A-value satisfying the inequality $3 < t.A < 12$ defines a tautology (i.e., the **where** expression is TRUE independent of the value assigned to a null $t.B$). It appears that it is not feasible to design efficient algorithms to solve symbolically this type of equation containing Boolean expressions involving inequalities. If expressions such as

$$\textbf{where} \qquad E\# > E\# - 1$$

```
QB :    range of   e is EMP
        range of   m is EMP
        retrieve   (e.NAME)
        where      m.SEX = "M" ∧
                   e.MGR# = m.E# ∧
                   e.MGR# ≠ e.E# ∧
                   e.E# ≠ m.MGR#
```

FIG. 2. Find the employees who have a male manager and do not manage themselves or their managers.

are allowed, then our system for detecting tautologies will also have to "understand" simple mathematics. The picture becomes even gloomier if we take into account that our system will also have to "understand" the semantic constraints of the schema. Consider, for example, the query Q_B of Fig. 2.

Clearly, any pair of tuples m and e which satisfy the first two conditions in the **where** clause, define a tautology for the remaining two, since an employee cannot be his own manager, neither can he be the manager of his manager. Clearly, a system which "understands" the integrity constraint in the schema will often be complex and expensive. Moreover, no system may ever be built to interpret *constraints declared and enforced via database procedures.*

In summary, it appears that any system which attempts to handle tautologies will not succeed in all cases and may be inordinately expensive to build and to use. Needless to say, this cost will have to be paid for all queries, even those not involving tautologies. This scenario suggests that the **ni** interpretation supplies a desirable alternative in practical situations.

ACKNOWLEDGMENTS

The author would like to thank Bob Horgan, Ed Lien, Eric Wolman, and the referees for the many improvements they suggested.

REFERENCES

1. M. M. ASTRAHAN *et al.*, System R: Relational approach to data base management, *ACM Trans. Database Systems* 1(2)(1976), 97–137.
2. G. BIRKOFF, "Lattice Theory," Amer. Math. Soc., Providence, R.I., 1967.
3. J. BISKUP, A formal approach to null values in database relations, *in* "Advances in Data Base Theory" (H. Gallaire, J. Minker, and J. M. Nicolas, Eds.), Vol. 1, pp. 299–341, Plenum Press, New York, 1981.
4. E. F. CODD, Relational completeness of database sublanguages *in* "Data Base Systems" (R. Rustin, Ed.), pp. 65–98, Prentice-Hall, Englewood Cliffs, N.J., 1972.

5. E. F. Codd, Extending the database relational model to capture more meaning," *ACM Trans. Database Systems* 4(4)(1979), 397–434.
6. R. Fagin, A. Mendelzon, and J. Ullman, A simplified universal relation assumption and its properties, *ACM Trans. Database Systems* 7(3)(1982), 343–360.
7. O. Frink, Pseudo-complements in semilattices, *Duke Math. J.* **29** (1962), 505–514.
8. M. R. Garey and D. S. Johnson, "Computers and Intractability: A Guide to the Theory of *NP*-Completeness," Freeman, San Francisco, 1979.
9. J. Grant, Null values in a relational data base, *Inform. Process. Lett.*, **6**(5)(1977), 156–157.
10. J. R. Horgan, "The Semantics of Relations and Nulls," Bell Laboratories Internal Memorandum, 1981.
11. T. Imielinski and W. Lipski, On representing incomplete information in a relational data base, *in* "Proceedings, 7th Int. Conf. on Very Large Data Bases, Cannes, France," pp. 388–397, 1981.
12. D. E. Knuth, "The Art of Computer Programming—Vol. 3: Searching and Sorting," Addison-Wesley, Reading, Mass., 1973.
13. M. LaCroix and A. Pirotte, "Generalized Joins," SIGMOD Record, Vol. 8, No. 3, Assoc. Comput. Mach., pp. 14–15, 1976.
14. Y. E. Lien, Multivalued dependencies with null values in relational databases, *in* "Proceedings 5th Int. Conf. on Very Large Data Bases, Rio de Janeiro," pp. 155–168, 1979.
15. Y. E. Lien, On the equivalence of database models, *J. Assoc. Comput. Mach.* **29**(2)(1982), 333–362.
16. W. Lipski, On semantic issues connected with incomplete information databases, *ACM Trans. Database Systems* **4** (3)(1979), 262–296.
17. D. Maier, "Discarding the Universal Instance Assumption: Preliminary Results," Tech. Report, No. 80/008, Dept. of Computer Sci., State Univ. New York at Stony Brook, March 1980.
18. R. Reiter, On closed world databases, *in* "Logic and Databases" (Gallaire and Minker, Eds.), pp. 55–76, Plenum, New York, 1978.
19. R. Reiter, Towards a logical reconstruction of relational database theory, *in* "Perspectives on Conceptual Modelling" (M. L. Brodie, J. M. Mylopoulos, and J. W. Schmidt, Eds.), Springer-Verlag, Berlin/New York, in press.
20. R. Stoll, "Set Theory and Logic," Freeman, San Francisco, 1963.
21. M. Stonebraker, E. Wong, P. Kreps, and G. Held, The design and implementation of INGRES, *ACM Trans. Database Systems* 1(3)(1976), 180–187.
22. J. D. Ullman, "Principles of Database Systems," 2nd ed., Computer Science Press, Potomac, Md., 1983.
23. Y. Vassiliou, Null values in data base management: A denotational semantics approach, *in* "Proceeding, ACM SIGMOD Int. Conf. Management of Data, Boston," pp. 260–269, May 30–June 1, 1979.
24. E. Wong, A statistical approach to incomplete information in database systems, *ACM Trans. Database Systems* 7(3), 470–488, September 1982
25. C. Zaniolo, "Analysis and Design of Relational Schemata for Database Systems," Ph.D. thesis, Univ. of California, Los Angeles, Tech. Rep. UCLA-Engineering, No. 7669, July 1976.
26. C. Zaniolo, Relational views in a database system; Support for queries, *in* "Proceeding, IEEE Computer Applications and Software Conf., Chicago," pp. 267–275, November 8–11, 1977.
27. C. Zaniolo, Design of relational views over network schemas, *in* "ACM SIGMOD Int. Conf. on Management of Data, Boston," pp. 179–190, May 30–June 1, 1979.
28. C. Zaniolo, A formal treatment of nonexistent values in database relations, 1983, submitted.
29. M. M. Zloof, Query-by-example: A database language, *IBM System J.* **16**, No. (4)(1977), 324–343.

Printed by the St. Catherine Press Ltd., Tempelhof 41, Bruges, Belgium