Missing Information (Applicable and Inapplicable) in Relational Databases

by

E. F. Codd

President
The Relational Institute

ABSTRACT

There has been some technical and justified criticism of the treatment of missing information in the data sublanguage SQL and in IBM's Database 2 system (a relational database management system). Some of this criticism has been directed (by mistake) at the relational model. The purpose of this paper is to clarify and extend the treatment of missing information by the relational model.

The clarification places heavy emphasis on the semantic aspects of missing information. The extension, which is relatively minor, provides a systematic approach (independent of data type) to dealing with the inapplicability of certain properties to some objects. This extension does not invalidate any part of the present version of the relational model.

The Relational Institute
Suite 106
6489 Camden Avenue
San Jose, CA 95120

Telephone: 408-268-8821

TRI Technical Report EFC-4

CONTENTS PAGE

This material is planned for inclusion in a book
by E. F. Codd to be published soon by Addison Wesley

PART 1

1. Introduction

In Section 2 of my paper on the extended model RM/T [EFC1], I gave an account of how the basic relational model represents and handles missing information, but there was very little emphasis on why that approach was adopted. Included in that part of section 2 dealing with the manipulation of missing information was an account of the three-valued logic proposed for determining the possibilities if some of the missing information were conceptually and temporarily replaced by known values. I have recently become aware of Date's paper [CJD1] in which he fires some criticisms at the approach in [EFC1] and proposes (in effect) a return to the "good old days" when, for each attribute that is permitted to have missing information, the database administrator or some suitably authorized user is forced to select a specific value from the domain on which the attribute is defined to denote missing information in that attribute.

In Part 1 of this paper we clarify the approach taken in [EFC1] to the representation and handling of missing information, and provide a stronger semantic underpinning for this approach. We also show in Part 1 how easily the approach in [EFC1] can be extended to handle the case of a property that is generally applicable to a class of objects, but inapplicable to certain members of that class. In Part 2 we take a close look at the various technical criticisms of the [EFC1] approach that have recently come to my attention. In addition, we provide some strong technical arguments against Date's proposed alternative scheme of "default values".

1.1 Definitions

In logic and in algebra, when the value or possible values of an item are unknown, we assign a named variable to the item and often call it "an unknown". Distinct items with unknown values are assigned variables with distinct names. Thus, a formula in logic or in algebra may involve several variables, and a common task in solving a problem is to find the values of these variables using a collection of equations.

In database management we could follow the same approach. Thus, if a database contained information about employees and projects, each occurrence of an unknown birthdate of an employee and of an unknown start-date for a project would be recorded as a distinctly named variable. Under certain circumstances the database management system (DBMS for brevity) might be able to deduce equality or inequality between two distinctly named variables -- or it might be able to deduce certain other constraints on the variables. However, it would rarely be possible for the DBMS to deduce the actual values of these variables. Instead, most of the missing or unknown items are eventually supplied by users in the form of late-arriving input.

Note that in the few cases where it is conceivably possible for the DBMS to deduce actual values for missing information, the cost of such deduction is likely to be too expensive in relation to the actual benefit. In the present version of the relational model we continue to assume that we should avoid the potential complexities of using the variables of algebra or logic for missing information. However, we also continue to pursue an approach which places more burden on the system than pre-relational approaches, and less on the application programmers and terminal users.

I make NO claim (now or in the past) that the relational approach to missing information places NO burden at all on these users. However, I do believe that any attempt to put missing information on a systematic basis (one that is uniformly applied to data whatever its type) will necessarily entail a learning burden. What is important is that this learning burden should pay off in terms of a safer and more reliable treatment of databases (a treatment that will strengthen the retention of database integrity).

To prepare for definitions to come, we adopt the term "elementary database value" (or db-value for brevity) to mean any value which a single-column attribute may have in any relation. Except for certain special functions, a db-value is ATOMIC in the context of the relational model. As a term for this concept, "datum" would have been preferred, except that its plural "data" has very broad use.

With regard to missing information, two questions seem dominant:

1) what kind of information is missing?
2) what is the main reason for its being missing?

In the relational approach we can interpret question 1) regarding the kind of information to be a question concerning structural context: is the missing information a whole tuple (row) or is it merely a component atomic value of a tuple (a db-value)? There appears to be no need to consider the consequences of an entire relation being missing, because a database necessarily models just a micro-world.

Moreover, we can interpret the main reasons to mean: is the information missing simply because its present value is unknown to the users, but that value is applicable and can be entered whenever it happens to be forthcoming; or is it missing because it represents a property that is inapplicable to the particular object represented by the tuple involved? Figure 1 summarizes the classification by kind (the structural context) and by reason.

Note that there are many other ways to classify missing information, but only these types appear to justify general support by the DBMS at this time. In this context, "general" means independent of the particular attribute and of its domain

or application data type. Since DBMS users (and even designers) are not yet accustomed to using techniques of this generality for handling missing information, we believe gradual introduction is appropriate. That is the main reason I introduced only one type of missing information in [EFC1], and am now introducing only one additional type. The two types which are stressed in this paper are:

A missing and applicable
I missing and inapplicable.

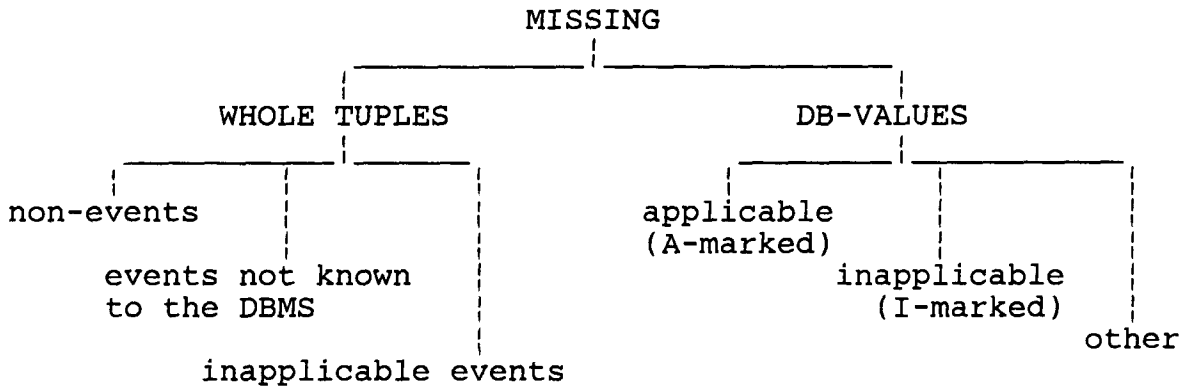

Fig. 1: Classification of Missing Information in a relational DBMS

In section 1.13 dealing with operators which generate marks, we briefly discuss the relationship between:

1) A and I applied to whole tuples;
2) A and I applied to components of a tuple.

Now follows a description of how a slightly extended version of the relational model handles these two types (A and I) of missing information.

A basic principle of the relational approach to missing information is that of recording the fact that a db-value is missing by means of a mark (originally called a null or null value in [EFC1]). There is nothing imprecise about a mark: either a db-value is present in an attribute of a relation in the database, or else it is absent therefrom.

The semantics of the fact that a db-value is missing are NOT the same as the semantics of the db-value itself. The former fact applies to any db-value, no matter what its type. The latter fact has semantics depending heavily on the domain (or application data type) from which the attribute draws its values.

Like a variable, a mark is a place-holder. However, it does not conform to the other accepted property of a variable: namely that semantically distinct missing values are represented by distinctly named variables.

We begin with a definition of the "missing-but-applicable-value mark" (for brevity, an A-mark). This mark is treated NEITHER as a value NOR as a variable by the DBMS, although it may be treated as a special kind of value by the host language. Consider an immediate property P of objects of type Z in a database. Normally P has a specific value in each and every tuple of that relation which provides the immediate properties of type Z objects. Suppose that, represented in the database, there is an object z of type Z and, at this time, the value of P for this object is unknown. Then, P would be assigned an A-mark in the database, providing P is considered to be applicable to the object z.

On the other hand, suppose that property P is inapplicable to the particular object z. Then P would be assigned an inapplicable-value mark (for brevity, an I-mark) in the tuple representing z. Thus, in the P attribute each tuple contains a value for P or an A-mark or an I-mark. Two examples follow:

1) an employee with a missing-but-applicable present salary -- his record would have an A-mark in the salary attribute;

2) an employee with an inapplicable sales commission (such an employee does not sell any products at this time) -- his record would have an I-mark in the commission attribute.

Why are we now calling these items "marks", rather than "values" or "null values" or "nulls"? We cite four reasons:

1) marks seldom behave like values;
2) we now have two kinds of marks, where there was previously just one kind of null;
3) some host languages deal with objects called "nulls" which are quite different in meaning from database marks;
4) "marked" and "unmarked" are better adjectives than "nulled".

To pursue 1), a mark in a numeric attribute (an attribute which normally has numeric values) cannot be arithmetically incremented or decremented by the DBMS, whereas the numeric values which are present can be subjected to such operators. To be more specific, if x denotes a db-value, A denotes an A-mark, and I denotes an I-mark:

x + x = 2x	x + A = A	A + x = A
A + A = A	A + I = I	I + A = I
I + I = I	x + I = I	I + x = I

A similar table holds for the three arithmetic operators minus, times, and divide (except that when both arguments are db-values the result is what you would expect from ordinary arithmetic). Similarly, a mark that appears in a character-string attribute (one that normally has character-string values) cannot have a second character string concatenated with that mark by the DBMS, in contrast to the character string values which are present. A table similar to the one above holds for concatenation also.

To summarize the remarks above: if we place I-marks in the top class, A-marks in the second class, and all db-values in the third class, the combination (arithmetic or otherwise) of any two items is an item of whichever class is the higher of the two operands.

How then can these marks appear in an attribute which normally contains values? Present hardware is of little help: it fails to support any special treatment of marks as distinct from values. For the same reason, present host languages (such as Cobol and PL/1) are of little help.

One way in the relational approach to support marks by software is to assign a single extra byte to any attribute which is allowed to have applicable or inapplicable marks. This approach has been adopted for A-marks in the IBM DB2 and SQL/DS products, and we believe it is a fundamentally sound approach, although some of the manipulative actions on marks need cleaning up. I agree with many of the criticisms Date directed at these implementations in [CJD1] and [CJD2], but unlike him I do not interpret these as criticisms of the way missing information is handled in the relational model, and certainly not as a justification for abandoning database nulls.

1.2 Primary keys and foreign keys of base relations

An important rule for relational databases is that, for integrity reasons, information about an unidentified (or inadequately identified) object is NEVER recorded in these databases (a sharp contrast with non-relational databases). Thus, the primary key attribute of each base relation is not permitted to include marks of either type (see section 1.4). However (and this is an aside), the mere fact that such marks are prohibited from appearing in an attribute does not of itself make that attribute the primary key attribute of a base relation! In the catalog there has to be an explicit declaration of the primary key of each base relation.

A foreign key consists of one or more attributes drawing its values from the domain or combination of domains, upon which at least one primary key is defined. Normally, I would advise DBAs or users to prohibit marks of either type explicitly from foreign key value attributes. Occasionally, there exists a strong reason to depart from this discipline (but I believe such departure needs justification).

1.3 Tuples containing A-marks and/or I-marks

Any tuple (or row) containing nothing but A-marks and/or I-marks can and should be discarded from the database. It would be an illegal tuple in a base relation (due to the entity integrity rule cited in section 1.4 below). It is not information-bearing in any derived relation (whether it be view, query, snapshot, or even an updated relation).

1.4 Integrity Rules

Two integrity rules apply to every relational database:

1 Entity integrity:

> No mark of either type is permitted in any attribute which is a component of the primary key of a base relation

2 Referential integrity:

> Let D be a domain from which one or more single-attribute primary keys draw their values. Let K be a foreign key which draws its values from domain D. Every unmarked value which occurs in K must also exist in the database as a value in the primary key of some base relation.

A single instance of referential integrity is an example of an inclusion dependency. The paper [CFP] reports interesting relationships between inclusion dependencies and functional dependencies.

It is important to observe that these rules specify a state of integrity and do not specify what action is to be taken by the system if an attempt is made to violate either rule. In the case of referential integrity, the DBMS should support at least three options: 1) refuse the command, 2) cascade the updates or deletes on the primary key values to all foreign keys defined on the same domain, or 3) replace each corresponding foreign key value by an A-mark. Users might occasionally need to have each corresponding foreign key value replaced by an I-mark. This required choice of actions is the reason that the referential integrity constraint should be supported in a general manner similar to that for user-defined integrity constraints, where a general choice of actions is also needed.

Finally, it should be possible for the database administrator (DBA for brevity) or any suitably authorized user to define additional, special-purpose integrity constraints and the action to be taken if there is an attempted violation. These constraints would be specific to the particular database involved.

Note that, if (as is usual) several attributes take their values from a common domain, marks may occur in some of these attributes and not in others. For example, a primary key attribute is not permitted to contain any occurrences of either kind of mark, whereas (at the DBA's discretion) corresponding foreign key attributes (on the same domain) may be permitted to contain occurrences of the A-mark (however, in many cases but not all, the DBA may wish to prohibit I-marks). Thus, declarations concerning whether a mark is permitted in or prohibited from an attribute should normally be associated with that attribute and not with the corresponding domain from which it draws its values.

1.5 Updating A-marks and I-marks

An A-mark in attribute C may be replaced by any db-value which complies with the domain constraints and attribute constraints declared for attribute C, and this may be done by any user authorized to make updates in attribute C. Similarly, any db-value in an attribute for which marks are permitted can be replaced by an A-mark.

An I-mark in attribute C may be replaced first by an A-mark and then by any actual value. This is made a two-step process so that attempts by a user who lacks special authorization to replace an I by an A can be prohibited by the DBMS. An A-mark in attribute C may be replaced by an I-mark and vice versa.

(In Figure 2 * means that extra authorization is needed)

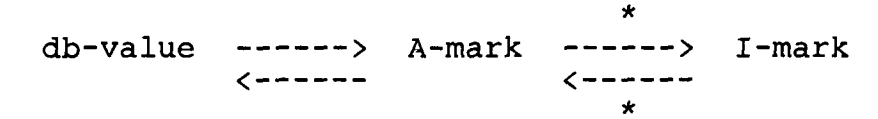

Figure 2: State diagram specifying permitted updates

The authorization mechanism should be extended to require the user who replaces any A-mark by the I-mark or vice versa to have special authorization for this action.

1.6 Application of Equality

What does it mean to assert that one "missing-but-applicable value" equals another? Is it appropriate to speak of the equality of two "inapplicable values"? In other words, under what circumstances does equality make sense? Our position is that there are two kinds of equality of marks to be considered: semantic equality and symbolic (or formal) equality. In regard to semantic equality, we must take into account how applicable and inapplicable values are expected to be used. Their uses are quite different in nature. In fact, the truth value of:

A-mark = I-mark

is FALSE with respect to both types of equality. Now let us explore equality between two occurrences of the same type of mark. Since the symbol is the same in both cases, the two occurrences are symbolically equal. The question of semantic equality needs more detailed investigation, and it is discussed below in sections 1.6.1 and 1.6.2 .

1.6.1 Missing-but-Applicable Information

Missing-but-applicable information opens up the opportunity for asking what might be true if one or more missing values were to be temporarily replaced by actual values. Frequently, "what if" databases have to be developed and manipulated separately from the so-called operational databases, because 1) the former represent what might be the case if certain events were to take place in the future (in the business or in its environment); while 2) the latter represent reality. Accordingly, updates in

the "what if" databases have to be regarded as representing conceptual actions (analytical or planning or projecting in nature). An important advantage of the A-marks is that some of the analysis can be carried out directly on the operational data without making any conceptual updates.

Suppose a database includes information about employees, including each employee's birthdate. Suppose also that birthdate is one of the immediate, single-valued properties of an employee that is allowed to be temporarily missing for one or more employees. It is quite possible that when an employee's birthdate is unknown, the actual value of this date may (eventually) prove to be ANY date which lies within the range of employee ages permitted by law and by company policy. Such a range of dates would be specified as a formula based on a variable representing the date of the current day, and this formula would be included in the catalog declarations 1) for the domain from which this attribute draws its values and possibly 2) for the specific attribute also.

The set of possible values is (in this case) quite large. In general, however, whether the set of possible values for a property is large or small, there must be at least two possibilities (otherwise the property value would be known). It would therefore be a mistake to expect the value TRUE when evaluating a logical condition which involves semantically comparing either one missing-but-applicable value to another or one such missing value to a known or specified value. For example, what is the truth value of the inequality

BIRTHDATE > 1-1-66

for a missing birthdate? It is clearly neither TRUE nor FALSE. Instead it can be said to be MAYBE (meaning maybe true and maybe false -- the DBMS does not know which holds). The logical truth value MAYBE can be thought of as a value-oriented counterpart for the A-mark when focussing on the domain of truth values.

When represented by A-marks, two missing values possess marks which match one another symbolically, but not necessarily semantically -- i.e., eventually these marks may be replaced by different values.

1.6.2 Inapplicable Information

A natural subsequent question is: must the systematic treatment of inapplicable values cause an additional extension of the underlying three-valued logic to a four-valued logic? Such an extension has been proposed in [VASS]. We believe, however, that the need for a four-valued logic is not urgent, and so I do not propose to introduce it until it is a very clear requirement.

At first sight, it appears to make sense to handle equality between two inapplicable-value marks just like equality between two actual values. Note however that any I-mark can be updated

by a specially authorized user to become an A-mark and later to become a db-value. Thus, within a relational language statement, whenever an I-mark is equated in a condition to an actual value or to an A-mark or to an I-mark, the truth value of such a condition is always taken to be MAYBE.

1.7 Three-valued Logic

A database retrieval may, of course, include several conditions like BIRTHDATE > 1-1-66, and the conditions may be combined in many different logical combinations (including the logical connectives AND, OR, NOT and the quantifiers UNIVERSAL and EXISTENTIAL). Suppose as an example a second immediate property recorded for each employee is the present salary of that employee. Suppose also that this attribute is allowed to have missing db-values. How does the DBMS deal with a query involving the combination of conditions:

(BIRTHDATE > 1-1-66) OR (SALARY < 20,000)

where either the birthdate condition or the salary condition or both may evaluate to MAYBE? Clearly, we need to know what is the truth value of MAYBE or TRUE, TRUE or MAYBE, MAYBE or MAYBE. From this we see that there is a clear need in any systematic treatment of missing values to extend the underlying two-valued predicate logic to a three-valued predicate logic.

In the following truth tables for the three-valued logic of the relational model [EFC1], we use P and Q to denote propositions, each of which may have any one of the truth values: t for true or m for maybe or f for false.

P	not P
t	f
m	m
f	t

P or Q	Q		
	t	m	f
t	t	t	t
m	t	m	m
f	t	m	f

P & Q	Q		
	t	m	f
t	t	m	f
m	m	m	f
f	f	f	f

Figure 3: The truth tables of three-valued logic

In the relational model the universal and existential quantifiers are applied over finite sets only. Thus, the universal quantifier behaves like the logic operator AND, and the existential quantifier behaves like OR -- both operators being extended to apply the specified condition to each and every member of the pertinent set.

When an entire condition based on three-valued, first-order predicate logic is evaluated, the result can be any one of the three possibilities TRUE, MAYBE, or FALSE. If such a condition is part of a query which does not include the MAYBE option, the result consists of all the cases in which this condition evaluates to TRUE, and no other cases.

If to this query we add the keyword MAYBE, then the result consists of all the cases in which this condition evaluates to MAYBE, and no other cases. This qualifier is used only for exploring possibilities and special authorization would be necessary if a user is to incorporate it in one of his programs or in a terminal interaction.

One problem of which DBMS designers and users should be aware is that in rare instances the condition part of a query may be a tautology. In other words, it may have the value TRUE no matter what data is in the pertinent columns and no matter what data is missing. An example would be the condition pertaining to employees (where B denotes BIRTHDATE):

(B < 1-1-66) OR (B = 1-1-66) OR (B > 1-1-66)

If the DBMS were to apply 3-valued logic to each term and it encountered a marked value in the birthdate column, each of the terms in this query condition would receive the truth value MAYBE. However, MAYBE OR MAYBE yields the truth value MAYBE. So the condition as a whole evaluates to MAYBE, which is incorrect, but not traumatically incorrect.

There are two options: 1) warn users not to use tautologies as conditions in their relational language statements (tautologies are a waste of the computer's resources); 2) develop a DBMS which examines all conditions, not in excess of some clearly specified complexity, and determines whether each condition is a tautology or not. Naturally, in this latter case, it would be necessary to place some limitation on the conplexity of each and every query, because with predicate logic the general problem is unsolvable. It is my opinion that option # 1 is good enough for now, because this is not a burning issue.

1.8 Selects, equi-joins, inequality joins, relational division

The manner in which algebraic selects (originally called restrict), equi-joins, inequality joins, and relational division treat A-marks and I-marks is determined by the remarks on the semantic treatment of equality in sections 1.6 and 1.7. An inequality join is a special kind of join using the inequality comparator: NOT EQUAL TO.

Thus, whenever an equi-join involves comparing two items for equality and one or both of them are marks, the pertinent tuples are glued together if and only if the MAYBE qualifier has been specified.

1.9 Ordering of Values and Marks

Our position on ordering is similar to that taken with regard to equality. There are two kinds of ordering to be considered: semantic ordering and symbolic ordering. The semantic version applies when using a less than or greater than condition in a statement of a relational data sublanguage. The symbolic version

applies when using the ORDER BY clause (e.g., to determine how a report is to be ordered). Let us consider symbolic ordering first.

The present ordering as implemented in DB2 in the ORDER BY clause of SQL involves "null values" (i.e., A-marks) representing missing-but-applicable values (the case of inapplicable values is not yet handled by the language SQL or by the DB2 system). DB2 places nulls at the high end of the value-ordering scale. In order to be compatible with this ordering, we also place A-marks at the high end (immediately after values), but then follow the new I-marks. This is the symbolic (or formal) ordering.

Note that inapplicable information in a particular attribute could be supported in extended SQL by requiring the DBA or a suitably authorized user to declare a value from the pertinent domain as the one to represent such inapplicability for a given attribute. However, I am definitely not advocating this approach. For one thing, this approach would implicitly and potentially define as many different orderings for the missing db-values relative to the existing db-values as there are attributes which are allowed to have missing db-values. Instead, I believe it is more systematic and more uniform across different data types to use a special mark (the I-mark), because these marks are not database values.

Now let us consider the semantic ordering. The truth value of each of the following expressions is MAYBE (not TRUE):

(db-value < mark) (mark < db-value) (mark < mark)

for any type of mark and any db-value. The same applies to these expressions if "<" is replaced by ">". If such an expression involves either one or two occurrences of marks, the truth value of the expression is MAYBE.

There has been some criticism that the symbolic ordering of marks relative to values runs counter to the semantic ordering and the application of three-valued logic. However, I fail to see any problem, because the use of truth-valued conditions involving ordering when applying a relational data sublanguage is at a higher level of abstraction than the use of the ordering of marks relative to values in the ORDER BY clause of a relational command.

1.10 Joins involving value-ordering

Joins involving the comparators:

less than or equal to	greater than or equal to
less than	greater than

treat applicable and inapplicable marks as determined by the the usual orderings of db-values and the "semantic ordering" of marks defined in the immediately preceding section. If the MAYBE qualifier does not accompany the request for a join, then (as usual) only those items are generated, for each of which the

entire pertinent condition has the truth value TRUE. On the other hand, if the MAYBE qualifier is present in the command, only those items are generated, for each of which the entire pertinent condition has the truth value MAYBE.

1.11 Application of statistical functions

In applying a statistical function to the db-values in one or more attributes of a relation, it is desirable to be able to specify how A-marks and I-marks are to be treated by this function, if it should encounter either type of mark. A practical approach is to support two temporary replacements: one for the A-mark occurrences and another for the I-mark occurrences. A convenient (but not mandatory) way of expressing the replacement action is by means of two separate, single-argument functions AR (which stands for A-mark replacement) and IR (which stands for I-mark replacement). In each case the single argument is a scalar constant.

We propose that the function or qualifier specifying the replacement be called the substitution qualifier. This qualifier is applicable to every kind of statistical function. However, if the statistical function has two or more arguments and these are applied to two or more attributes, the specified replacement action must apply to all of these attributes. An example of practical use of this qualifier is the calculation of a salary budget for each department based on the present salary of each member of a department. If a few salaries are missing (and therefore A-marked), one may wish to compute the total for each department by requesting temporary replacement of each A-mark occurrence by the maximum salary of those known to the DBMS for that department.

In certain special cases the two replacements may be values equal to one another. In certain other special cases one or both of the replacements may each be a mark identical to the mark being replaced (the need for this case is determined by the default action for omitted substitution qualifiers being specified as "ignore marks of the corresponding type").

Note that the state of the database is not changed by the execution of any one of these statistical functions alone. In other words, the substitutions replacing marks by values or by marks are in effect during and only during the execution of the pertinent statistical function. One advantage of making these substitutions temporary is that certain kinds of possibilities can be investigated without setting up a separate "what-if" database.

If the substitution qualifier is omitted altogether from a statistical function request, the DBMS would assume that only the unmarked values should contribute to the result. On the other hand, if the existence of any occurrence of a mark of type t in the operand is to yield an A-mark as the result, there must be a qualifier in the command requesting that marks of type t be

detected, but not modified in the temporary substitution sense (i.e., a mark of type t is replaced by itself).

The subject of applying statistical functions to sets that might happen to be empty is postponed to section 2.5.

1.12 Removal of duplicate tuples

It is unfortunately true that in certain relational DBMS "derived relations" (usually called tables) may contain duplicate tuples (an example of such a system is IBM's DB2). However, it is possible in using these systems to specify that all but one occurrence of any duplicate tuples be removed. A common example of this is a projection which does not happen to include the primary key of the operand relation.

If two or more tuples happen to contain the same actual values and no marks (applicable or inapplicable), the removal of duplicates is obvious. The interesting case for this paper is that in which some of the values are missing.

In this case it seems reasonable to assert that, if the actual values in comparable attributes of two or more tuples are the same, and if the marks in these tuples are also symbolically equal (i.e., equal in formal terms), then these tuple occurrences should be considered duplicates of one another -- and all except one tuple occurrence should be removed.

There has been some criticism [CJD1] of the fact that this scheme for removal of duplicates does not conform to the semantic notions of equality described in section 1.6 . However, I fail to see any problem, because the semantic notions of equality are applicable at a higher level of abstraction than the symbolic equality involved in removal of duplicate tuples.

1.13 Operator-generated marks

The operators OUTER JOIN and OUTER UNION are capable of generating derived relations in which some (possibly all) of the attributes have one or more missing db-values. Which type of mark should the DBMS generate?

The introduction of a new attribute C for a selected base relation R is achieved by appending to the catalog a description of this attribute. This causes the DBMS to record in R itself an appropriate mark for attribute C for each tuple in R. In this case as in the previous case, which type of mark should the DBMS generate?

A simple (and, we believe, adequate) solution is for the DBMS:

1) to generate either A-marks only or I-marks only in executing a particular command involving an operator which is capable of generating one or more occurrences of marks, and (as part of that command) the user should specify the kind of mark to

be generated for that command;
2) to use A-marks as the default kind if no kind is explicitly specified.

To justify the choice of A-marks as the default, note that A-marks are weaker than I-marks in that a user requires no special authorization beyond the usual update authorization if he wishes to update an A-mark into a db-value (see section 1.5). Therefore, it is easier for him to delve into "what if" kinds of interactions on the relation (he does not need to get special authorization beyond that for querying).

1.14 Some necessary language changes

For convenience, we express these changes as changes to SQL. It should be a simple matter to adapt them to any other reasonably complete, relational data sublanguage. We need an ability to refer to marks similar to the way SQL presently refers to nulls, except that it should allow the user to distinguish between the two types of marks when he wishes to do so. We suggest the use of the clauses IS A-MARKED, IS NOT A-MARKED, IS-I-MARKED, and IS NOT I-MARKED to refer to the presence and absence of A-marks and I-marks specifically, and the clauses IS MISSING and IS NOT MISSING in case the user does not care which type of mark is involved. The SQL clauses IS NULL and IS NOT NULL should be abandoned.

Example: find the employees who are (a) eligible (b) ineligible to receive sales commissions. In many companies query (a) is much more likely than query (b), because it is usually only a minority of employees who are eligible for such commissions. In pseudo-SQL, appropriate statements for these queries would be:

```
(a)  SELECT serial_number FROM employees
           WHERE commission IS NOT I-MARKED
(b)  SELECT serial_number FROM employees
           WHERE commission IS I-MARKED
```

Additional needs are the substitution qualifiers AR and IR, when applying a statistical function to any column in which marks may occur (see section 1.11 for details), and the empty set qualifier ESR when applying a statistical function to a collection of sets, some of which may be empty (see section 2.5 for details).

1.15 Normalization

The concepts and rules of functional dependence, multi-valued dependence, and join dependence were all developed without considering missing db-values. A comparatively recent paper on these dependencies is [BFH]. All the normal forms based on these dependencies were also developed without considering missing db-values. Does the possible presence of marks in some attributes (each mark indicating the FACT that a db-value is missing) undermine all these concepts and theorems based on them? Fortunately, the answer is no! The main reason is that a mark is

not itself a db-value. To be more specific, a mark in attribute C is semantically different from the db-values in C. Thus, the normalization concepts do NOT apply, and should NOT be applied, globally to those combinations of attributes and tuples containing marks. Instead, they should be applied:

1) to a conceptual version of the database in which tuples containing missing-but-applicable information in the pertinent attributes have been removed; and

2) when each attempt is made to replace a mark by a db-value.

When an attempt is made to insert a new tuple into a relation and a certain component db-value is missing, it is pointless for the system to base acceptance or rejection of this tuple on whether the missing db-value does meet or might meet or fails to meet certain integrity constraints based on a dependence in which the pertinent attribute is involved. The proper time for the system to make this determination is when an attempt is made to replace the pertinent mark by an actual db-value.

One might be tempted to treat I-marks differently from A-marks, because an I-mark cannot be replaced in one step by a db-value. However, one or more users may have the authorization to replace an I-mark by an A-mark -- and then the door is wide open to further replacement of this mark by a db-value. So we advocate treating all marks alike (regardless of type) in the matter of testing any dependence constraint (functional, multi-valued, join or inclusion): for every tuple that contains a mark in the column or columns being tested, the DBMS should wait until an attempt is made to replace the marked item(s) by an actual db-value!

Some time in the future, a fully relational DBMS should have the capability of storing (in its catalog) statements defining the various kinds of dependencies as they apply to the particular database being managed (certainly functional, multi-valued, join and inclusion dependencies). A program should also be available to deduce all the dependencies that are a consequence of those supplied by the DBA or other suitably authorized users, so that no dependence which is logically implied by others is overlooked by the DBMS, when attempting to change a mark into a db-value. Further, such a DBMS should be able to check the database (whenever necessary and without explicit invocation by an application program) against one or more of these integrity constraints. In general, such checking is likely to be necessary whenever a mark is replaced by a db-value.

PART 2 Response to Technical Criticisms

2. General comments

Publication of the relational model in 1970 [EFC2] preceded the development of prototypes of relational DBMS products by several years. The model is more abstract than these systems and has an existence independent of them. This is an advantage in many ways. First, it provides a useful goal, target, and tool for the designers and developers of new DBMS products. Second, it provides a special kind of standard against which dissimilar DBMS products may be measured and compared. No DBMS product or data sublanguage marketed in the western world today fully supports each and every feature of the relational model [EFC2,3,4]. Third, it provides a foundation upon which theoretical work in database management has been and will be based.

It is therefore important to distinguish between technical criticisms of the model on the one hand and of the products based on that model on the other hand. With respect to the treatment of missing information, the technical criticisms have strayed across the boundary without proper justification. I shall cite and discuss criticisms appearing in Date's recent collection of technical articles [CJD1 & 2], partly because Date has the support of a number of database practitioners with similar views on nulls, partly because his books are widely read, and partly because his books deal technically with many other topics in a way that I fully support. Accordingly, the page numbers designated hereafter refer to pages in the book cited in [CJD2]. For brevity, I shall also refer to his "default value scheme" as the DV scheme.

2.1 The value-oriented misinterpretation

The representation in IBM relational DBMS products of missing information (in the IBM manuals the corresponding marks are sometimes called nulls and sometimes null values) by means of an extra byte seems correct. However, not all of the ways in which these products process missing information conform to the relational model (even that version of the model which preceded the extensions cited in this article). In [CJD1] Date provides numerous cases in which the processing of nulls (i.e., A-marks) in DB2 is non-systematic. I find myself in agreement with these comments as criticisms of this particular implementation of the relational model. However, I do not accept these comments as criticisms of the relational model itself.

Any approach to the treatment of missing information should consider what it means for a db-value to be missing, including how such occurrences need to be processed. A basic principle in the relational model is that treatment of all aspects of shared data in databases is not just a representation issue. There are always other considerations which the DBMS must handle, especially 1) the approach to manipulating the data;
and 2) the preservation of database integrity.

It is quite inappropriate to leave these considerations to be handled by users in a variety of ways, and buried in a variety of application programs. This principle applies just as forcefully to missing information. Thus, we insist that missing information is not just a representation issue.

2.2 The Alleged Counter-Intuitive Nature

Under the heading "Intuitive Difficulties" (p.323-326) Date discusses the examples 1) suppliers in London and 2) suppliers not in London, where the city attribute for suppliers is allowed to have missing-but-applicable indicators (A-marks). He criticizes the relational model for requiring the user to make the distinction between 1) "suppliers known to the system not to be in London" and 2) "suppliers not in London".

In [WL] Lipski appears to agree with me on this issue. Date, on the other hand, asserts that this distinction is "subtle" and "one that is likely to mystify the user". However, subtlety (like beauty) is in the eyes of the beholder. Something that is more important than subtlety is that a user -- in failing to make this very distinction -- may cause serious errors (errors which could have serious consequences for the user's business). Even if Date's counter-proposal (the DV scheme) were adopted, this would not prevent or help prevent the occurrence of this type of error.

Let us look at this example in more detail, to show how Date's DV scheme constitutes a non-solution to the problem. Consider a relation S identifying suppliers and describing their immediate, single-valued properties. Let one of these properties be the city in which the supplier is based. A sample snapshot follows:

S (S# SNAME CITY ...)

S#	SNAME	CITY	...
s1	JONES	LONDON	...
s2	SMITH	BRISTOL	...
s3	DUPONT	v	...
s4	EIFFEL	PARIS	...
s5	GRID	v	...

where v denotes a character string, declared in the catalog to be the "default value" for the attribute CITY in the relation S (which in Date's scheme means "unknown" or "missing" for this attribute only). Note that v may be any character string that does not represent the actual name of any existing city (for example, it could be "???" or "XXX").

Now consider the two queries:

Q1 Find the suppliers in London;
Q2 Find the suppliers NOT in London.

If these queries are represented in a relational language (such as ALPHA [EFC3], SQL [IBM], or QUEL [RTI]) ignoring the occurrences of v and therefore ignoring the occurrences of missing db-values, the answer to Q1 would be s1, which is correct only if interpreted as those suppliers known by the system to be in London (since at any later time an occurrence of v may be updated to "LONDON"). Q2 would similarly yield the set {s2, s3, s4, s5}, which is

1) definitely incorrect when interpreted as the suppliers known by the system not to be in London; and

2) potentially incorrect when interpreted as the suppliers actually not in London.

Thus, the user of a DBMS equipped with the DV scheme MUST take into account whether an attribute is allowed to contain missing values; he must shape his query accordingly; and he must differentiate in his thinking between 1) what is known to the system and 2) what is actually a fact and 3) what could be the case. This requirement of the DV approach to missing information forces the user to make the very same distinctions for which Date criticizes the relational approach to missing information. The burden of having the user make these distinctions is not removed by having him formulate the query as: find the suppliers not based in London and not based in "???". In fact, the burden arises because the problem of dealing with missing information correctly is just not a simple problem.

Date's additional comment on p.331 that the DV approach "avoids all the difficulties associated with the null value scheme" is clearly incorrect. I would characterize the DV scheme as an approach that is likely to entice the naive user, but its claimed simplicity is quite likely to trap the unwary and give rise to serious mistakes.

Finally, consider the idea that a notion should be rejected because it is "counter-intuitive". This is a type of criticism that I cannot accept as technical in nature, precisely because it is too subjective with respect to 1) the person, 2) the culture, and 3) the era. One example should suffice, although there are many.

At least ten centuries ago, very few people on earth were concerned with making long voyages, whether over land or sea. Most people therefore had no cause to consider that the earth might not be flat, and that the shortest distance from A to B on the surface of the earth is not a straight line, but instead an arc of a great circle. Thus, when the scientific proposal was made that the earth was spherical, it was considered by most people to be extremely counter-intuitive. On the other hand, it would be quite difficult to find anybody today who considers this idea to be counter-intuitive. Thus, if an idea appears to be counter-intuitive, it is not necessarily wrong.

2.3 The Alleged Breakdown of Normalization

On p.323 of [CJD1] Date discusses the example of a base relation R(A,B,C) satisfying the functional dependence A --> B, for which it is not assumed that A is the primary key, so that A can be permitted to have missing db-values. He asserts that serious problems are bound to arise if R contains a tuple (?,b1,c1) and an attempt is made to insert another tuple (?,b2,c2) where b1 and b2 are not equal, and ? denotes what we are calling an A-mark. He asserts that either the two nulls must be considered distinct from one another or the second tuple must be rejected because "it might violate the dependency" when the null is replaced by an actual value.

He seems to have rejected a third option, which is the one adopted in the relational model: that, whenever the A component of a tuple is missing (or becomes missing), the functional dependence A --> B is not enforced by the DBMS for this tuple until an attempt is made to replace the mark (null) in attribute A by an actual db-value. In fact, if Date's proposed DV scheme were adopted, this third option would not be available, because a null or mislue is treated in the DV scheme as just another database value. Hence, in the DV scheme the functional dependence constraint must be enforced upon first entry of the tuple -- and this gives rise to the possibility that a tuple might be erroneously rejected by the DBMS.

He also asserts that, if the second tuple is not rejected upon attempted entry, "we are forced to admit that we do not have a functional dependency" of B on A. This is clearly one more instance of a value-oriented misinterpretation.

I also do not agree at all with Date's assertion at the bottom of p.323 that "the normalization procedure breaks down". It should be clear that, because nulls (or, as they are now called, marks) are NOT database values, the rules of functional dependence -- and of multi-valued dependence -- do not apply to them. Instead, they apply to all unmarked db-values.

With the DV scheme the normalization procedure does break down, precisely because it does treat missing information as database values. This is one more reason why I contend the DV scheme is not an acceptable solution to the problem of handling missing information in databases.

2.4 Implementation Anomalies

In general, the present state of relational DBMS products in regard to the representation and handling of missing information is far from satisfactory. For IBM products based on the language SQL, see [CJD1 & 2]. In non-IBM products, even the representation aspects have gone astray. In several of these products, the DBMS designer has misinterpreted nulls as db-values (see section 2.1).

In at least one well-known (and otherwise sound) DBMS product, zero has been chosen as the value to indicate missing information in all numeric attributes. I consider the number zero to be far too valuable in its normal role in all kinds of business activities -- for example, as a real number representing the actual quantity of a part in stock or the actual quantity of currency owed by one or more customers. Therefore, I do not consider zero to be an acceptable choice to be reserved by a DBMS for denoting a missing numeric db-value. In fact, I believe that, in the context of computer-supported database management, it is unacceptable to reserve any specific numeric value or any specific character string value to denote the fact that a db-value is missing.

2.5 Application of Statistical Functions

On p.330 Date uses an example involving the sum of an empty set of numbers to show that SQL encounters difficulties by unconditionally yielding the SQL null as the result. While I agree with this example when interpreted solely as an SQL blunder, I do not agree with Date's use of it as an example that justifies outright rejection of the relational approach to missing information.

Further, Date proceeds to argue at the bottom of p.330 that, if the sum were to yield zero unconditionally as the result, there would be the problem that the average of an empty set of numbers would not be the sum divided by the count (the number of elements in the set). However, this problem arises because 0/0 is normally taken to be undefined in elementary mathematics. It is NOT a difficulty stemming from the relational approach to missing information! When taking averages, it is necessary for programmers and users to provide special treatment for the case in which the divisor (i.e., the number of elements in the set) is zero, because zero exhibits a unique behavior when it is used as a divisor. Incidentally, I fail to see how the DV scheme provides any solution or simplification for this problem.

I consider this problem to be quite separable from the question of how to deal with missing information. Nevertheless, the approach taken to this problem in the relational model can be illustrated by taking the example of generating the total salaries earned by each department, where each total is computed as the sum of the salaries earned by each employee assigned to the pertinent department; and let us assume that a few departments exist which have zero employees assigned to them at this time.

In applying statistical functions there are two important alternative methods of handling occurrences of empty sets:

1) each occurrence of an empty set is ignored (i.e., passed over);
2) a value (specified by the user) is taken as the RESULT for each occurrence of an empty set.

Note the difference between these two actions if the statistical function happens to be the AVERAGE, and if the value selected in the second approach is zero. The first action omits the departments which have zero members, while the second generates zero for each such department.

The empty set qualifier, a function ESR with a single argument, say x, causes each occurrence of an empty set to yield the result x and does not affect the result obtained from each non-empty set. Omission of this qualifier altogether causes each occurrence of an empty set to be ignored.

2.6 Interface to Host Languages

The host languages, with which I am familiar, do not include support specifically aimed at the semantics of the fact that information in databases may be missing. This means there is bound to be an interface problem, whatever approach is taken in the data sublanguage. If the approach taken on the database side of the interface is a uniform, systematic one (independent of data type), the interface is likely to be simpler than an approach like the DV scheme which requires database users to keep inventing (attribute by attribute) their own techniques for dealing with this problem, not to mention the burden of communicating their inventions to other users of the database.

The relational approach therefore has a strong advantage in this area over the DV scheme. In this paper we have made one concession in the relational approach to reducing user confusion about the host language interface: namely, a change from the terms "null" and "null value" to the term "mark" for the indicator which designates the fact that a db-value is missing.

2.7 The Default-Value Approach

The main problems with the DV approach are:

1) it does not appear to provide any tools for the handling of missing information -- it merely provides a means for representing the fact that something is missing;

2) the representation proposed is by means of a db-value and that forces the testing of functional and other kinds of dependencies at time of entry of data, which is the wrong time if a missing value is involved;

3) its representation of the fact that a db-value is missing is not only dependent on the data type of each pertinent attribute, it can even vary across attributes having a common data type -- all of which presents a severe burden in thinking and in inter-personal communication for the DBA, users, and programmers;

4) the numerous and varied techniques for handling missing data will be buried in the application programs (and it is

highly doubtful that they will be documented adequately);

5) it treats each missing db-value as if it were just another db-value (i.e., it ignores the semantics and suffers from the value-oriented misinterpretation);

6) it is a step backward from the relational model to an ad hoc, unsystematic approach frequently adopted in the pre-relational era.

In the case of item 5) numerous specific consequences have been cited earlier in this paper, along with the ensuing penalties.

It is also important to realize that -- whatever the approach -- the DBA, application programmers, and end users must cope with the semantics of the fact that some db-values for some attributes may be missing. Because the DV scheme offers no tools for the handling of missing information in a uniform systematic way, users are forced to invent a variety of ad hoc, unsystematic ways, and the DBMS cannot exert any real integrity control over these. Finally, research in this area is still being pursued, and I make no claim that the relational model, as it now stands, treats missing information in a way that is unsurpassable. Any replacement, however, must be shown to be technically superior.

3. Conclusion

In the past, I have intentionally included in the relational model a systematic treatment of missing-but-applicable information only, and I am now adding similar treatment for inapplicable information. It is intended as a step toward removing from database administrators and users the burden of solving this problem in highly specialized (and often inadequate) ways.

I make no claim that this systematic treatment is either intuitive or counter-intuitive. Neither do I claim that users who have used the old style with non-relational DBMS will be able to avoid any learning process. I also do not claim that missing-but-applicable information, inapplicable information, and the derivation of deductions therefrom are thoroughly understood yet.

Nevertheless, I feel sure the reader will agree that the present scheme in the relational model is far more systematic than the old-style approach which used values specially earmarked to represent missing information (and misrepresent the semantics), where the earmarking and the invention of manipulation techniques:

1) are likely to be different for each different attribute;
2) are a significant burden on DBAs and users.

One of the great advantages of the relational approach (defined precisely in the relational model [EFC 1,2,3,4,5,6]) is the unparalleled power of its treatment of integrity. It is high time for vendors and users to place more emphasis on the

introduction and retention of database integrity, and consequently invest more effort in the systematic treatment of missing information described in this article.

4. Acknowledgment

I am grateful to Ronald Fagin of IBM Research, San Jose and to Chris Date of the Codd & Date Consulting Group for reading and commenting on a draft version of this paper.

5. References

[EFC1] E. F. Codd, "RM/T: Extending the Relational Model to capture more meaning", ACM TODS, Vol 4, No 4, Dec 1979 (see Section 2: The Basic Relational Model)

[EFC2] E. F. Codd, "A Relational Model of Data for Large Shared Data Banks", Comm. ACM, Vol 13, No 6, June 1970

[EFC3] E. F. Codd, "ALPHA: A Relational Data Sublanguage based on the Relational Calculus", Proc. ACM-SIGFIDET Workshop, San Diego, 1971

[EFC4] E. F. Codd, "Relational Database: A Practical Foundation for Productivity", Comm. ACM, Vol 25, No 2, Feb 1982

[EFC5] E. F. Codd, "How Relational is your Database Management System?", Computerworld, Nov 14 & 21, 1985

[EFC6] E. F. Codd, "The Twelve Rules for Relational DBMS", The Relational Institute, San Jose, EFC-6 / May 16, 1986

[CJD1] C. J. Date, "Null Values in Database Management", Proc. 2nd British National Conference on Databases, Bristol, England, July 1982; also Chapter 14 in book entitled "Relational Database: Selected Writings", Addison Wesley, 1986

[CJD2] C. J. Date, "A Critique of the SQL Database Language", Chapter 13 in book entitled "Relational Database: Selected Writings", Addison Wesley, 1986

[RTI] INGRES/QUEL Reference Manual, Relational Technology Inc., Alameda, California (most recent edition)

[VASS] Y. Vassiliou, "Null values in database management: A denotational semantics approach", Proc. ACM SIGMOD 1979 Int. Conf. on Management of Data, Boston, Mass., May 30 - June 1, 1979

[BFH] C. Beeri, R. Fagin, J. H. Howard, "A Complete Axiomatization for Functional and Multi-Valued Dependencies in Database Relations", Proc. ACM SIGMOD 1977 Int. Conf. on Management of Data, Los Angeles, CA

[CFP] N. Casanova, R. Fagin, C. Papadimitriou, "Inclusion Dependencies and their interaction with Functional Dependencies", JCSS Vol. 28, No. 1, February 1984

[IBM] SQL Reference Manual, IBM Corporation, White Plains, NY (most recent edition)

[WL] Witold Lipski, "On semantic issues connected with incomplete information data bases", Institute of Computer Science, Polish Academy of Sciences, Warsaw, Poland 1978