FOUR VALUED LOGIC FOR RELATIONAL DATABASE SYSTEMS

G. H. Gessert
Director, Corporate MIS
Primerica Corporation
300 St. Paul Place
Baltimore, MD 21202
Telephone 301-332-3993

Abstract

This paper proposes a specific four-valued logic (4VL) as a means of handling missing data in Relational Data Base Management systems. The proposed 4VL is a minor variant of the standard 4VL in which the "least true" condition is interpreted as "inapplicable" rather than "false". The use of this 4VL is defended on the grounds that by defining several additional unary operators, the 4VL can be rendered intuitively manageable. The proposed unary operators contribute to the conceptual utility of the 4VL by providing an explicit way for users to relate familiar results of two-valued logic to analogous results in 4VL.

1. Introduction

No Relational Database Management System (RDBMS) adequately handles missing data. In the hope of promoting discussion by providing something concrete to discuss, this paper will propose an approach to handling missing data based on four-valued logic (4VL). The proposal makes use of 4VL because, as will be argued below, 4VL is necessary to capture the critical distinction between unknown missing data and inapplicable missing data.

There are probably two reasons why current RDBMSs are deficient in handling missing data. First, missing data is essentially a practical problem; it is much easier for an academic computer scientist to ignore than for a data processing professional to ignore. In particular, in a RDBMS, conceptually at least, inapplicable fields can often be avoided by distinguishing between different entity types. For example, suppose that one is designing a loan database. If some loans have balloon payments and others do not, one can avoid having missing data in the balloon payment field by defining an additional base table for balloon payment loans. This gambit, however, complicates queries, which then require processing two tables instead of one. Processing multiple tables will likely not only waste programmer and machine resources, but will make the application harder for end-users to understand and use. Multiple tables may, for example, put the query beyond the scope of simple report writers and visual query interfaces.

Second, handling missing data in a way that distinguishes between applicable and inapplicable missing data requires four valued logic. This is so because any statement about a field (and hence any selection of records based on compound statements about fields) can have four possible outcomes: inapplicable, false, applicable but unknown, or true. Four valued logic is complex because there are different kinds of negative statements and many different choices for how truth values can be assigned to conjunctions and disjunctions, given the truth values of the components. A popular school of thought, represented by Date [CJD],

and to a lesser extent by Codd [EFC1], holds that because 4VL is complex and DP practitioners are not subtle, 4VL should not be incorporated into RDBMSs.

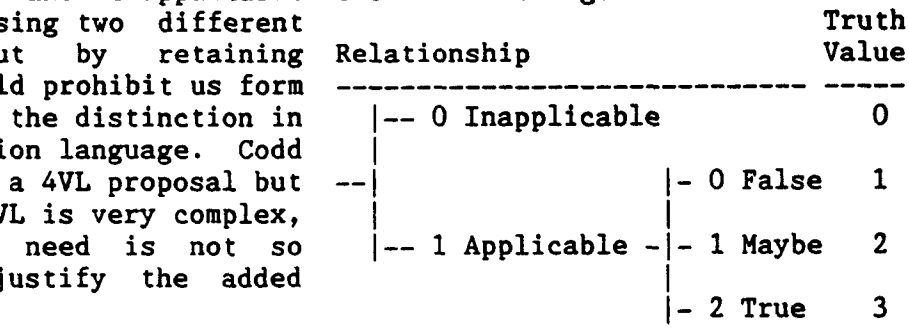
Nonetheless, the distinction between applicable and inapplicable missing data is of great practical value. For example, if a field represents a special charge or fee (as in the case of a balloon payment), it is of great practical importance to know if the fee amount is missing because it is not yet known, or because it does not apply. Many DP practitioners, though not subtle, place considerable value on knowing when money is owed. In addition, it is not obvious that the practical task of handling missing data is simplified by oversimplifying the data manipulation language to the extent that important distinctions can not be made. Date would prohibit even the use of null values to represent missing data. Codd [EFC] would allow us to represent the distinction between applicable and inapplicable missing data by using two different "NULL" marks, but by retaining 3-valued logic, would prohibit us form making full use of the distinction in the data manipulation language. Codd [EFC 1, 2] outlines a 4VL proposal but then argues that 4VL is very complex, and that the the need is not so pressing as to justify the added complexity.

There is a sort of conceptual economics argument implicit in Codd's remarks. The argument holds that 4VL is of limited value and high intellectual cost. I have argued briefly above that 4VL has great practical benefit in handling missing data. The burden of the remainder of this paper is to argue that 4VL can be afforded at a modest intellectual cost.

The underlying concepts of the 4VL proposed here are the same as those in Codd's proposal [EFC1,2]. In particular, this proposal will rely on the concepts and terminology inherent in Codd's distinction between a "marked" and "NULL" value. The approach taken here also relies on Codd's treatment of arithmetic and comparison operators. This presentation will focus on two areas of difference: Truth Tables and Data Manipulation. The issue of joins over marked fields, though important and relevant, is outside the scope of this paper.

2. Truth Tables

There are two main issues in formulating truth tables: treating the different types of unary operators, especially, negation, and assigning values to the tables for the AND and OR functions. The goal is to approach both these issues in a way that provides a consistent semantic analysis of missing data. The semantic analysis of missing data used here is shown in the figure below.



Value	Code	Read as
0	NA	= Not Applicable
1	AF	= Applicable but False
2	AM	= Applicable and May be True
3	AT	= Applicable and True

Fig. 1: Logical Relationships and Interpretation of Truth Values

Different types of unary operators come about in 4VL because, in any multi-valued logic, the inverse of True is not simply False. In two valued logic, if we know that a statement is not true, we know that it is false. In four valued logic, if we know that a statement is not Applicable and True (AT), we do not know its truth value. Several possibilities remain: Applicable but False (AF), Applicable and Maybe true (AM) or Not Applicable (NA). It is possible, of course, to associate many other semantic interpretations with the four truth values, but this interpretation, summarized in figure 1, will be used throughout this presentation. Because the opposite of AT is not necessarily False, it is possible to have several negation operators. The term "negation" is used here in the extended sense defined by Rescher [NR]. In the extended sense, an operator (~) is a negation if for the values designated as the "most true" and "least true", the truth value of a sentence (p) and its negation (~p) are not equal.

Consider two possibilities:

p	INV(p)	p	NOT(p)
NA	AT	NA	NA
AF	AM	AF	AT
AM	AF	AM	AM
AT	NA	AT	AF

Both INV and NOT could qualify as negation operators, depending on which value (NA or AF) is designated "least true". The first "negation" operator corresponds to inverting the entire tree of possibilities given in figure 1. In this sense, Applicable and True (AT), i.e., actually true, is the inverse of Not Applicable (NA), i.e., could not ever be true or false. Similarly, with respect to the possibility of being true, Maybe true (AM) is the inverse of Applicable but False (AF), i.e., actually being false.

The second "negation" operator, represents the inverse with respect to applicable values. It corresponds to inverting the major, "Applicable" branch of the tree of possibilities, shown figure 1. By this rotation of the major branch, the Not Applicable (NA) and Applicable and Maybe true (AM) values are unchanged. True and False are inverted, as usual in discussions where the applicability of statements is assumed. This is the inverse table that Codd proposes [EFC1, p46).

The truth tables for AND and OR are the standard multi-valued logic truth tables, in the sense defined by Rescher [NR]. In standard four valued truth assignments, AND is always the the minimum of the truth values of the component statements and OR is always the maximum of the truth values of the component statements. This can be seen clearly if we replace the values {NA, AF, AM, AT} with the corresponding numeric values, {0, 1, 2, 3}. Note, however, that Not Applicable (NA), rather than false (AF), corresponds to the lowest "least true" rank. This ranking, in order of the possibility of being true, is discussed further below.

AND

	NA	AF	AM	AT
NA	NA	NA	NA	NA
AF	NA	AF	AF	AF
AM	NA	AF	AM	AM
AT	NA	AF	AM	AT

OR

	NA	AF	AM	AT
NA	NA	AF	AM	AT
AF	AF	AF	AM	AT
AM	AM	AM	AM	AT
AT	AT	AT	AT	AT

AND	0	1	2	3
0	0	0	0	0
1	0	1	1	1
2	0	1	2	2
3	0	1	2	3

OR	0	1	2	3
0	0	1	2	3
1	1	1	2	3
2	2	2	2	3
3	3	3	3	3

If /a/ represents the truth value of a sentence "a" then:
p AND q = min (/p/,/q/),
p OR q = max (/p/,/q/), and
INV(p) = (3-/p/).
Note that if the Not Applicable (0) value is dropped; AND, OR, and NOT correspond to the three valued case embodied in the SQL standard.

In two valued logic DeMorgan's law holds, i.e., "p or q = not(not p and not q)". In other words, "AND and OR are duals; to say that "either p or q is true" is to say that "it is not the case that both p and q are false". It seems desirable to preserve some analog of this result. At least, it should turn out that AND and OR are systematically related so that: given a statement cast in terms of AND, and an intuitively equivalent statement, cast in terms of OR; the results should be equivalent. "Intuitive" in this case means, intuition based on the two valued logic, expressed in ordinary language. It turns out that the analog of "p or q = not(not p and not q)" in this 4VL is: "p OR q = INV(INV(p) AND INV(q))".

Note that p OR q is not equivalent to NOT(NOT(p) AND NOT(q)). Although Codd has a different assignment of values in the truth tables, it also turns out that in his system, "p OR q" and NOT(NOT(p) AND NOT(q)) are not equivalent. Since Codd has only one form of negation, his system would be counterintuitive in that the relationship between AND and OR would not be what we would expect based on our experience with DeMorgan's law.

The point of assigning the lowest rank (0) to the Not Applicable condition can be seen in relation to the operation of computing OR using max(/p/,/q/) and AND using min(/p/,/q/). The effect of regarding Not Applicable (0) as "lower than" Applicable and False is to render the Not Applicable (0) condition ineffective in computing OR (the maximum) but to render the Not Applicable (0) condition "contaminating" when computing AND (the minimum). To use Codd's example, consider the following two predicates which might be used to select employee records:

(birthdate > 1-1-50) OR (commission > 1000)
(birthdate > 1-1-50) AND (commission > 1000)

Suppose that an employee's birthdate is unknown so that "birthdate > 1-1-50" evaluates to AM and suppose that the employee is not a salesman so that "commission > 1000" evaluates to NA. The OR condition would evaluate to max(2,0) = 2 = AM. That is, the compound, OR statement, is Applicable and Maybe True. The AND condition, would evaluate to min(2,0) = 0 = NA. That is, the NA (0) value contaminates the whole compound, AND, statement, and renders it inapplicable.

Codd's truth tables function the same way, with respect to the combination of Applicable and Maybe true AND/OR Not Applicable. Codd, however, does not carry through the idea of the Not Applicable value contaminating AND statements, to include the combination of Applicable and False AND Not Applicable. Consider the case in which the employee is born in 1947 and is not a salesman. In this case, using the above truth tables, the OR condition would remain uncontaminated and would evaluate to max(1,0) = 1 = AF. But the AND condition would evaluate to min(1,0) = 0 = NA. Using Codd's Truth tables both the OR and

the AND condition would evaluate to Applicable but False. The assignment listed here has the virtue that a conjunction never becomes false by default, because part to it does not apply and part is inapplicable. A conjunction is inapplicable if and only if any one of its coujuncts is inapplicable. A conjunction is false if none of its conjuncts is inapplicable and only if any one of its coujuncts is inapplicable.

3. Data Manipulation

Four Valued logic implies that the data manipulation language must be extended to a) recognize the two additional truth values, b) provide at least two forms of "negation".

When one formulates a predicate in SQL, the NOT operator has two roles:

NOT1: Immediately following the WHERE clause, it functions to select records where the predicate statement following it evaluates to false.

NOT2: Within a compound predicate statement, NOT reverses the truth value of component statement that follows it.

For NOT1, we would have to extend SQL to include operators that recognize the other possible truth values. Like NOT, these operators would evaluate to true (AT), whenever the statement following it evaluates to one particular truth value: AT, AF, AM, or AT. That is /AT(p)/ = AT if and only if /p/ = AT; otherwise /AT(p)/ = AF. Consider, for example the following SQL statement.

```
SELECT S#
FROM S
WHERE NOT (STATUS = 30)
```

The NOT operator (NOT1) might be replaced by an AF operator, as follows.

```
SELECT S#
FROM S
WHERE AF (STATUS = 30)
```

To select records where status is unknown, one would write:

```
SELECT S#
FROM S
WHERE AM (STATUS = 30)
```

Similarly, to select all records where a predicate was contaminated by inapplicable values, one would write:

```
SELECT S#
FROM S
WHERE NA (STATUS = 30)
```

In this proposal, the default operator would be AT.

It might also be useful to have functions, analogous to the SQL "IS NULL", that return the logical status of a field:

```
/LOGICAL (STATUS)/ =
 AT if STATUS has a value

/LOGICAL (STATUS)/ =
 NA if STATUS is marked NA

/LOGICAL (STATUS)/ =
 AM if STATUS is marked AM
```

One could then write:

WHERE AM (LOGICAL (STATUS)) or simply WHERE AM(STATUS) to form the predicate analogous to WHERE STATUS IS NULL.

The advantage of the LOGICAL function notation is that the it makes clear the connection between marks and Truth Values. Another advantage is that the extended "NOT1" operators (NA, AF, AM, AT) can be used in a consistent way both to select predicates that evaluate to a particular value and to select fields that contain marked values. Another advantage of the

extended NOT1 operators will be apparent in what follows.

For NOT2, it is necessary to include at least two forms of "negation", NOT and INV, which correspond to the two important forms of "negation", discussed above in the section on Truth Tables.

The point of including the INV operator is to:

1. Make it explicit that NOT and INV are different.

2. Provide a means of translating an expression written in terms of OR, into an equivalent expression, written in terms of AND, or vice versa.

3. Provide a means of expressing the inverse of a statement, with respect to the possibility of the statement being true.

The INV operator has the side benefit that it preserves DeMorgan's law. Having only the NOT operator would disguise the fact that obvious analog of DeMorgan's Law does not hold. Having, in addition, the INV operator, not only highlights the possibility that two valued analogs based on negation may not hold, but provides an explicit way for the practitioner to accomplish the analogous task to applying DeMorgan's law. It seems a valuable general guideline that where possible, the data manipulation language ought to provide a means of representing the analog of simple two-valued tautologies.

To return to the issue of conceptual economics, note that the NOT1 operators, especially AT(p), provide a means generating a two-valued domain of 4VL, within which the familiar two valued results hold. For example, there is another, more appealing way, to preserve our two valued intuitions concerning DeMorgan's law. DeMorgan's law holds if we restrict our attention to a binary partition of the values, AT and all others. The operator AT imposes this partition on the truth values. For example, if the expression p = (STATUS < 30) and q = (CITY = 'LONDON'); then AT(p) OR AT(q) = NOT (NOT(AT(p) AND NOT(AT(q))). The function AT maps the four valued case onto the two valued case in which all two valued tautologies hold. AT(p) corresponds, intuitively, to a strong form assertion in which p is considered true only when p is known to be true and is considered not-true if p is inapplicable, false or unknown.

For example, consider the following query.

```
SELECT COUNT(*)
FROM S
WHERE  (STATUS = 30) OR  NOT (STATUS =
30)
```

This query, based on two valued intuition ought to yield a value equal to the cardinality of S, but it will not. The following query will yield the expected result.

```
SELECT COUNT(*)
FROM S
WHERE AT(STATUS = 30) OR NOT AT(STATUS
= 30)
```

Because these additional unary operators (NA, AF, AM, AT, INV, and NOT) provide a means of bridging the gap between 4VL and our two-valued intuitions, they render 4VL manageable, and affordable at a very modest intellectual cost.

4. Acknowledgement

I wish to thank my friends and colleagues for reviewing this paper

and for their comments.

5. References

[EFC1] E. F. Codd, "More Commentary on Missing Information in Relational Databases (Applicable and Inapplicable Information)", Sigmod RecordVol 16, No. 1, March 1987

[EFC2] E. F. Codd, "Missing Information (Applicable and Inapplicable) in Relational Databases", Sigmod Record, Vol 15, No. 4, December 1986

[CJD] C. J. Date, "Null Values in Database Management", Chapter 15 in book entitled "Relational Database: Selected Writings", Addison Wesly, 1986

[NR] N. Rescher, Many Valued Logic, McGraw-Hill Inc., New York, 1969