A More General Model For Handling Missing Information In Relational DataBases Using A 3-Valued Logic

Kwok-bun Yue
Department of Computer Science
University of Houston - Clear Lake
2700 Bay Area Boulevard
Houston, TX 77058-1098

Abstract

Codd proposed the use of two interpretations of nulls to handle missing information in relational databases that may lead to a 4-valued logic [Codd86, Codd87]. In a more general model, three interpretations of nulls are necessary [Roth, Zani]. Without simplification, this may lead to a 7-valued logic, which is too complicated to be adopted in relational databases. For such a model, there is no satisfactory simplification to a 4-valued logic. However, by making a straightforward simplification and using some proposed logical functions, a 3-valued logic can handle all three interpretations.

1. Introduction

Currently, there is no Relational Database Management System that adequately handles missing data [Gess]. To handle missing information in a relational database, Date considered using default values for replacing missing data [Date86]. Lipski proposed to use a subset of an attribute domain to represent the partial knowledge that the actual value of an attribute is one of the values in the subset [Lips]. By far, the most popular technique is by using null values. For example, the popular query language SQL supports the concept of a single null value [Date87].

In [Codd79, Codd86, Codd87], Codd proposed the use of two values to represent two possible interpretations of nulls:

A-mark: the data value is missing and applicable.
I-mark: the data value is missing and inapplicable.

Gessert argued that the distinction between applicable and inapplicable missing data is of great practical value [Gess]. Based on these two null values, either a 3-valued logic (3VL) or a 4-valued logic (4VL) can be developed. In a 3VL, an additional truth value M is added to represent that the outcome *may be* either true or false. [Codd86] contained a detailed description of implementing such a 3VL in relational databases.

4VL was proposed and discussed in [Codd86, Gess, Vass]. An additional truth value, I, is added to represent the outcome of evaluating logical expressions involving inapplicable values or inconsistency. Codd indicated that 4VL is more precise but the extra complexity of 4VL is not justified currently [Codd87]. However, he also suggested that 4VL should be incorporated in the future and external specifications of a DBMS product should permit expansion at a later time from 3VL to 4VL support without impacting users' investment in application programming [Codd87]. Gessert claimed that the 4VL he proposed is intuitively manageable and is thus of practical use [Gess].

Zaniolo proposed a new interpretation of nulls to represent a total lack of information, without knowing whether the data value is applicable or not [Zani]. Using only one null value for this interpretation, he then elaborated the set-theoretical properties based on information theory and developed efficient strategies for evaluating queries in the presence of nulls. Lerat and Lipski argued that non-applicable nulls (i.e. I-mark) are important and should not be left out [Lera]. Roth et. al. developed a theory for a more general model using all three interpretations of nulls and applied it to nested relations [Roth]. However, they did not discuss the select operation in detail and, in particular, did not discuss the effect on the choice

of a 3VL or 4VL system.

In this paper, we discuss the consequence of using three null values on selecting an n-valued logic system. Section 2 describes examples demonstrating the necessity of using three null values in a more general model. Section 3 analyzes the consequences of using three null values on the truth values of logical expressions. Such a model cannot be effectively handled by 4VL. In section 4, we show that a straightforward simplification will allow us to use a 3VL system. Together with some logical functions, the generality of using three null values can be fully incorporated. Our conclusions are in Section 5.

We will use the notations of A-mark and I-mark of Codd because of their popularity. Since the purpose of the paper is to illustrate the effect of using three null values to general readers, it is written in an informal style. For a more formal treatment of Section 2, please refer to [Roth, Zani].

2. Three Interpretations Of Nulls

Although both A-mark and I-mark represent missing *data value*, a careful analysis reveals that A-mark and I-mark are actually different from the point of view of missing *information*. An I-mark does not indicate any missing information since we are sure that the attribute is inapplicable. There is no uncertainty. On the other hand, an A-mark indicates a true lack of information since we know only that the attribute is applicable but we do not have its value. However, there is another circumstance of missing information, as illustrated by the examples below.

Example 1 Consider a simplified relation: r(Name, Marital_Status, Spouse_Name). It is assumed that Marital_Status can have only two values: S (single, including divorced or widowed) or M (married). Marital_Status is always applicable but Spouse_Name is inapplicable if Marital_Status is S. There are three possible interpretations of nulls as demonstrated in the relation of Figure 1.

Since Kim is single and does not have a spouse, then ?1 should mean that the attribute Spouse_Name is inapplicable. Jane is married and has a spouse so ?2 should mean that Spouse_Name is applicable but its value is currently missing. Similarly, ?3 should also mean applicable but missing. Since we do not know whether Mary is married or not, we do not know whether Spouse_Name is applicable to her or not. Hence, ?4 should mean that the value is missing and it is not known whether the attribute is applicable or not. It is a *complete* lack of information. []

	Name	Marital_Status	Spouse_Name
t_1	Mary	M	Joe
t_2	Kim	S	?1
t_3	Jane	M	?2
t_4	May	?3	?4

Figure 1 An instance of a relation r

Example 2 Suppose an employee relation has an attribute Dept, indicating the department the employee belongs to. Whereas ordinary employees must work under a department, it is assumed that some may not belong to any department and Dept can be inapplicable. Hence, there are four possible situations for any employee:

(1) The department the employee is working for is known.

(2) The employee is not an ordinary employee and does not work under any department (eg. the president, VP's, etc).

(3) The employee is an ordinary employee but is not yet assigned a department (eg. beginning trainees).

(4) It is not known which department the employee is working under or whether the employee is an ordinary employee or not (eg. special consultants, cousins of the president).

The last three situations correspond to three different null values for the attribute Dept. []

Therefore, in a more general model of relational databases, there should be three null values:

I-mark: the data value is inapplicable.
A-mark: the data value is applicable but missing.
U-mark: the data value is missing and its *applicability* is unknown.

In [Roth], these values were called **unk** (unknown), **dne** (do not exist) and **ni** (no information) respectively. Note that U-mark represents a complete lack of information since it is not known whether the data is applicable or not. With A-mark, at least we know that the data value is applicable. Furthermore, only U-mark and A-mark represent missing information. I-mark does not represent any lack of information -- we know that the value is inapplicable. No more information is necessary. Refer to [Roth] for a more formal treatment.

Present hardware does not support the distinction between null values and ordinary values [Codd86]. One possible way to support three null values is to use two additional bits. The Inapplicable_bit is set to 1 if and only if it is known that the attribute value is inapplicable. The Applicable_bit is set to 1 if and only if it is known that the attribute value is applicable. Hence, if an attribute has an ordinary value, its Applicable_bit is set to 1. The three null values can then be implemented as below:

A-mark: the attribute does not have a normal value and the Applicable_bit is set.
I-mark: the Inapplicable_bit is set.
U-mark: the attribute does not have value and both the Applicable_bit and Inapplicable_bit are not set.

The advantage of this implementation is its relative ease to evaluate logical functions described in Section 4. Another advantage is that it is possible to define constraints to automatically set the bits. For example, in relation r of Figure 1, we can define a constraint such that a value 'M' in Marital_Status will set the Applicable_bit of Spouse_Name whereas a value 'S' in Marital_status will set the Inapplicable_bit of Spouse_Name.

3. Multi-Valued Logic

Since two null values lead to a 4VL, it may be tempting to think that, without simplification, three null values will lead to a 5VL. For example, the evaluation of the expression X = C, where X is an attribute and C is a constant, can be:

(1) T: if the value of X is applicable and equal to C,
(2) F: if the value of X is applicable and not equal to C,
(3) I: if the value of X is I-mark,
(4) M_1: if the value of X is A-mark; this represents a meaning of 'may be T or F', and
(5) M_2: if the value of X is U-mark; this represents a meaning of 'may be T, F or I'.

However, the situation is more complicated as illustrated by the following example.

Example 3 Consider a relation EMP(Name, Dept, Salary, Rank) where Dept may have all three null values. The outcome of evaluating the expression (Dept = 'Account' AND Salary > 20000) OR (Dept <> 'Account' AND Rank = 7) on the tuple (Jane, U-mark, 30000, 7) may either be I or T, but not F. This is because if Dept is actually inapplicable, then the expression should be evaluated to I. On the other hand, if Dept is actually applicable, then one of the sub-expressions Dept = 'Account' or Dept <> 'Account' must be true. Since the sub-expressions Salary > 20000 and Rank = 7 are both true for the tuple, the entire expression should be evaluated to T. Since F is not possible, we need to have a new truth value, M_3, to represent a meaning of 'may be T or I'.

Following similar arguments, the evaluation of the same expression on the tuple (Kim, U-mark, 10000, 4) may either be I or F, but not T. This is because the sub-expressions Salary > 20000 and Rank = 7 are both false now. Hence, we need to have another new truth value, M_4, to represent a meaning of 'may be F or I'. Note that we have assumed that I is 'more false' than F in the sense that I AND F is equal to I [Gess]. It is possible to define I AND F to be F [Codd87]. However, even so, M_4 is still necessary, as illustrated by the evaluation of the unintelligent expression (Dept = 'Account') AND (Dept = 'Personnel') on both tuples in this example. []

That 7 truth values are necessary should not seem strange. If we consider that the 'basic' values of a logical expression are t (true), f (false) and i (inapplicable or inconsistent), then a truth value can be assigned to every non-empty subset S of the set {t,f,i} indicating that the actual basic value can be one of the elements of S. Hence, we have T for {t}, F for {f}, I for {i}, M_1 for {t,f}, M_2 for {t,f,i},

M_3 for {t,i} and M_4 for {f,i}. These exhaust the possibilities of non-empty subsets of the set {t,f,i}.

A 7-valued logic (7VL) will be too complicated for users to use or maintain. For example, the truth tables of the AND or OR logical operators will have 49 entries. Worse yet, it will be very difficult, if not impossible, to define the truth tables of the operators AND, OR or NOT according to their 'natural interpretation' (for example, what should be the value of M_1 OR M_3?). In fact, these operators are not *truth-functional* in the 7VL. The truth value of an expression cannot be determined solely by the truth values of its sub-expressions joined by these operators [Resc].

Example 4 Suppose both Dept and Rank allow all three null values. The expressions (Dept = 'Account'), (Dept = 'Personnel') and (Rank = 7) should all be evaluated to M_2 (i.e., {t,f,i}) for a tuple with U-mark as values in both Dept and Rank. The expression (Dept = 'Account') AND (Rank = 7) should still be evaluated to M_2. However, the expression (Dept = 'Account') AND (Dept = 'Personnel') should be M_4 (i.e., {f,i}) since Dept cannot be equal to 'Account' and 'Personnel' at the same time. Since we have shown two situations where (M_2 AND M_2) should be evaluated to M_2 and M_4 respectively, the AND operator is not truth-functional. []

Although the truth-functionality problem also exists in 3VL, it can be tackled much easier [Codd86, Gess]. On the other hand, it will significantly increase the complexity of a 7VL and render the 7VL impractical. Thus, it is necessary to make simplifications to produce a logic with fewer truth values.

A first attempt may be to reduce to a supposedly more precise 4VL. However, we argue against the use of a 4VL. First, a 4VL is more complicated and confusing. For example, the truth tables for the AND operator are different in [Codd86] and [Gess]. As another example, two types of negation are suggested in [Gess]. More importantly, there is no satisfactory way to reduce the number of truth values to 4 for a system with three null values.

Note that we ended up with a 7VL because we have 3 basic values: t, f and i. If we have 4 truth values, we must keep all three basic values since 2 basic values lead to at most 3 truth values. The standard 4VL [Codd86, Gess] have the truth values T, F, I and M (i.e. {i,f}). However, the standard 4VL cannot handle queries with U-mark as values for some participating attributes since the outcome should actually be {t,f,i}. Other simplifications to a 4VL are even more problematic.

4. A 3-Valued Logic

An obvious simplification to 7VL is to allow only 2 basic values: t and f. This will give only 3 truth values: T for {t}, F for {f} and M for {t,f}. A simple way to do so is *to equate the value i to the value f* in the new 3VL. Hence, T now stands for true (and thus applicable), whereas F stands for false *or* inapplicable.

Example 5 Referring to the relation in Figure 1, the query "Who has a spouse named Joe?" involves the logical expression: Spouse_Name = 'Joe'. In a 7VL, the evaluations of the expression on the four tuples in the relation r of Figure 1 are:

t_1 ==> T: {t},
t_2 ==> I: {i},
t_3 ==> M_1: {t,f}, and
t_4 ==> M_2: {t,f,i}.

In the 3VL, the basic value i is equated to f and hence the evaluation of the expression on the tuples are:

t_1 ==> {t}: T,
t_2 ==> {i} --> {f}: F,
t_3 ==> {t,f}: M, and
t_4 ==> {t,f,i} --> {t,f,f} = {t,f}: M.

For example, the expression is evaluated to F, instead of I in the 7VL, for t_2. This is reasonable since Kim is single and does not have a spouse. In particular, Kim does not have a spouse named Joe.

As another example, the expression Dept = 'Account' can be used for the query "who is working in the Account department?". If the value of Dept is I-mark for an employee, then he is not working in any department, including the Account department. Hence, it is natural to evaluate the expression to F. []

We can then define the truth value of the

equality operator as in Figure 2, where C is a non-null constant. Note that this definition for the case of I-mark is not the same as that given by Codd. Codd suggested that I-mark = C should be evaluated to M no matter whether C is an I-mark, an A-mark or of a non-null value [Codd86]. The reason he gave is that an I-mark can be updated to an A-mark and later to a non-null value [Codd86]. In other word, Codd considered I-mark to represent a lack of information and hence uncertainty. However, as stated earlier, I-mark actually does not represent any lack of information. There is no uncertainty about the value I-mark and thus I-mark = C should not be M (except when C is U-mark). We should evaluate an expression based on the current value of a tuple, not its possible value in the future update.

expression	truth value
U-mark = U-mark	M
U-mark = I-mark	M
U-mark = A-mark	M
U-mark = C	M
I-mark = I-mark	T
I-mark = A-mark	F
I-mark = C	F
A-mark = A-mark	M
A-mark = C	M

Figure 2 Truth values of evaluating the equality operator

expression	truth value
U-mark > U-mark	M
U-mark > I-mark	F
U-mark > A-mark	M
U-mark > C	M
I-mark > I-mark	F
I-mark > A-mark	F
I-mark > C	F
A-mark > A-mark	M
A-mark > C	M

Figure 3 Truth values of evaluating the operator >

The truth value of simple expressions using other relational operators can be defined in a manner similar to that of the equality operator. The difference is that only the equality operator can be defined for I-mark. For example, I-mark > I-mark has no meaning since I-mark cannot be ordered. Hence, the operator > is *inapplicable* in this case, and in accordance with the idea of equating the basic value of i to f, the expression should be evaluated to F. Figure 3 shows the evaluation of the operator >. Other relational operators can be defined likewise.

A compound expression consists of expressions joined by the logical operators AND, OR and NOT. The evaluation of a compound expression can be obtained by the truth tables of the operators given in Figure 4. However, the operators AND and OR are still not truth-functional. Although the intended truth value of both (M OR M) and (M AND M) is M, other truth values are actually possible. To be more specific, (M AND M) may be evaluated to F and (M OR M) to T.

AND	T	M	F
T	T	M	F
M	M	?1	F
F	F	F	F

OR	T	M	F
T	T	T	T
M	T	?2	M
F	T	M	F

NOT	
T	F
M	M
F	T

Figure 4 Truth tables of the operators AND, OR and NOT

Example 6 The evaluation of the expression (Dept = 'Account' AND Salary > 20000) OR (Dept <> 'Account' AND Rank = 7) on the tuple (Jane, A-mark, 30000, 7) should yield T instead of M since the value of DEPT is applicable and either DEPT = 'Account' or DEPT <> 'Account' is true. The evaluation of (Dept = 'Account') AND (Dept = 'Personnel') on the same tuple should yield F instead of M. However, if the value of Dept is U-mark instead of A-mark, then the first expression should be evaluated to M, whereas the second expression should still be F. []

Hence, for the truth tables in Figure 4, ?1 usually stands for M, but may be F in some uncommon situations. Similarly, ?2 usually stands for M, but may be T in some uncommon situations.

Fortunately, these uncommon situations are quite rare and may occur only when (1) there is an attribute in the expression that has a value of A-mark or U-mark, and (2) the attribute appears in

more than one sub-expression. It may simply represent programming errors or bad programming style and should be avoided if possible. Thus, in most situations, we may simply use the truth tables to evaluate an expression, taking ?1 and ?2 as M. In the rare cases that may have doubt, it may be necessary to check for tautology [Codd86] or fallacy. Another approach is to restrict the value of the attributes as sub-expressions are evaluated [Gran]. A general discussion on the truth functionality problem can be found in [Zani].

The 3VL system will be adequate so long as (1) we remember that inapplicable is interpreted as false for expression evaluation, and (2) we do not need to find out whether an expression is applicable. If we need to find out the applicability of an expression, then we need to provide additional tests for it.

One approach is to support constants representing null values. For example, (X = I-mark) can be used to test whether the attribute X is inapplicable. The disadvantage of this approach is that the user may confuse the symbol itself with its intended meaning. For example, the expression X = U-mark can be interpreted as (1) X *is* a U-mark, or (2) the outcome of comparing the value of X to U-mark for equality. According to the first interpretation, X = U-mark returns true if X is a U-mark. In the second interpretation, comparing any values (nulls or non-nulls) to U-mark always yields a truth value of M. Of course, the tables for the evaluation of operators in Figures 2 and 3 are based on the second meaning.

Another approach is to use logical functions. Logical functions are more structured and higher in level of abstraction. It is only necessary to think in terms of concepts such as applicability or information missing, instead of the three types of null values. In fact, it may not be necessary for the users to be aware of the existence of three null values.

The basic logical functions should include Non_null, Maybe and Inapplicable. The logical function Non_null returns T if and only if its attribute argument has a non-null value. Otherwise, it returns F. The logical function Maybe returns T if and only if its Boolean argument is evaluated to M. Otherwise, it returns F.

The logical function Inapplicable returns T if and only if some attributes in its argument are inapplicable. It returns F if and only if all attributes are applicable. Otherwise, it returns M.

(1) If X is an attribute, then
   (a) if the value of X is I-mark, then Inapplicable(X) = T,
   (b) if the value of X is U-mark, then Inapplicable(X) = M,
   (c) if the value of X is A-mark, then Inapplicable(X) = F, and
   (d) if X is of a non-null value, then Inapplicable(X) = F.

(2) If X is a simple expression Y @ Z, where Y and Z are attributes or non-null constants and @ is a relational operator, then Inapplicable(X) = Inapplicable(Y) OR Inapplicable(Z).

(3) If X is a compound expression E % F, where E and F are expressions and % is a logical operator, then Inapplicable(X) = Inapplicable(E) OR Inapplicable(F).

Other logical functions, such as Applicable, Not_True or Not_False can be constructed for ease of use. These logical functions are adequate to construct the necessary queries for a system with 3 null values since we can always construct a suitable expression to obtain the set of tuples that evaluate any expression to any one of the seven truth values.

For example, if we want to find out all tuples that evaluate the expression E to M_1 (i.e., {t,f}), then the expression E' = (Maybe(E)) AND (NOT (Inapplicable(E))) can be used. The set of tuples that evaluates E' to true is the answer. Figure 5 shows how we can do this for all 7 truth values. Some examples of using the logical functions follow.

Example 7 To find out whether the value of X is U-mark, the expression Maybe(Inapplicable(X)) is used. To find out the person that has a spouse not named 'Joe', the expression (NOT (Spouse_Name = 'Joe')) AND Applicable(Spouse_Name), or simply (Spouse_Name <> Joe), can be used. []

Truth value	Expression
T: {t}	E
F: {f}	(NOT E) AND (NOT Inapplicable(E))
I: {i}	Inapplicable(E)
M_1: {t,f}	(Maybe(E)) AND (NOT Inapplicable (E))
M_2: {t,f,i}	Maybe(E) AND Maybe (Inapplicable(E))
M_3: {t,i}	E OR Inapplicable(E)
M_4: {f,i}	NOT E

Figure 5 Expressions for tuples that evaluate an expression E to a given truth value

It is worthy to note that logical functions may bring additional possibilities for the truth-functionality problem and care should be taken to handle them. For example, Applicable(X) and Inapplicable(X) will both be evaluated to M if the value of X is U-mark. However, the expression Inapplicable(X) AND Applicable(X) should be evaluated to F, not M.

5. Conclusion

In this paper, we have presented a more general model for handling missing information by using 2three null values. Without simplification, this leads to a 7VL which is too complicated. Although there are many proposals for using a 4VL, there is no reasonable simplification from a 7VL to a 4VL that captures the generality of 3 null values. By equating the basic value i to f and using logical functions such as Inapplicable and Maybe, a 3VL will be sufficient to capture the generality of 3 null values.

Acknowledgement

We would like to thank Dr. Sharon Perkins and Ms. Chloris Yue for their invaluable suggestions.

Reference

[Codd79] E.F. Codd, RM/T: Extending the Database Relational Model to Capture More Meaning. **ACM Trans. on Database Systems** 4(4), 397-434 (Dec. 1979).

[Codd86] E.F. Codd, Missing Information (Applicable and Inapplicable) in Relational Databases. **SIGMOD RECORD** 15(4), 53-77 (Dec. 1986).

[Codd87] E.F. Codd, More Commentary on Missing Information in Relational Databases (Applicable and Inapplicable Information). **SIGMOD RECORD** 16(1), 42-47 (Mar. 1987).

[Date86] C.J. Date, Null Values in Database Management. In **Relational Databases: Selected Writings**, Chapter 15, Addison-Wesley, Mass., 1986.

[Date87] C. J. Date, **An Introduction to Database Systems, Volume 1**, Addison-Wesley, Mass., 1987.

[Gess] G.H. Gessert, Four Valued Logic For Relational Database Systems. **SIGMOD RECORD** 19(1), 29-35 (Mar. 1990).

[Gran] J. Grant, Null Values in a Relational Database. **Info. Proc. Lett.** 6(5), 156-157 (Oct. 1977).

[Koch] R. Kocharekar, Nulls in Relational Databases: Revisited. **SIGMOD RECORD** 18(1), 68-73 (Mar 1989).

[Lera] N. Lerat & W. Lipski Jr., Nonapplicable Nulls. **Theoretical Computer Science** 46, 67-82 (1986).

[Lips] W.L. Lipski Jr., On Semantic Issue Connected With Incomplete Information Databases. **ACM Trans. on Database Systems** 4(3), 262-296 (Sep. 1979).

[Resc] N. Rescher, **Many Valued Logic**, McGraw-Hill, New York, 1969.

[Roth] M.A. Roth, H.F. Korth & A. Silberschatz, Null Values in Nested Databases. **Acta Informatica** 26, 615-642 (1989).

[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, pp162-169.

[Zani] C. Zaniolo, Database Relations with Null Values. **Journal of Computer and System Sciences** 28, 142-166 (Feb. 1984).