

## **The Faults with Defaults: Reply to Date.**

Tom Johnston

Published 1997, in Database Programming and Design.

### ***Default Values vs. Special Values.***

I was looking forward to Date's next iteration of his position on default values because I was anticipating something that many of us could have used, if we chose, instead of the nulls and MVL (multi-valued logic) provided by SQL. We would then have had two ways available to us for handling missing information. On the one hand, with a well-designed scheme expressing a two-valued logic approach to default values, the more conservative among us could have used this less expressive but also less complex and error-prone way of handling missing information.<sup>1</sup> On the other hand, for those of us who felt comfortable dealing with the additional complexity which increased expressive power inevitably brings with it, we could have used nulls and MVL.

But unfortunately, this is not what Date has provided. He has provided, instead, a "special values" scheme (as he now calls it) which no product today supports, and which would require SQL to abjure its commitment to MVL and build in support for this alternative approach instead. Date acknowledges this, while trying to put the shoe on the other foot, when he says that "today's SQL products might have difficulty with such a notion," *i.e.* his special values scheme, "but that's their problem." (Pt 2, p.18)

### ***UNK and Equality.***

Let me move directly to the heart of the matter concerning Date's new special values scheme.

---

<sup>1</sup> Date, of course, would disagree with the "less expressive" here. I support this claim below.

After explaining that his new scheme will require us to replace every domain--say domain XXXX--with a new domain XXXX\_OR\_UNK, Date goes on to consider the use of various operators with the UNK special value.<sup>2</sup> He begins with the equality operator, and after defining its use with the UNK special value, says: "Note that this definition implies that the comparison UNK=UNK.....returns *true*.....No three-valued logic here!" (Pt 3, p.16)<sup>3</sup>

Right here, on this clear and explicit point, we could have no better example of the *need* for three-valued logic! And since this goes right to the heart of whether a special values approach can be as expressive as an MVL approach, we must consider this claim of Date's about UNK and equality very carefully.

So consider a two-column table, of one-hundred rows, each of which is a pair of UNK values. Suppose the domain for each column is (a) integers from 1 to 100, and (b) UNK. For each row, therefore, with a real value in each column, *i.e.* a number from one to one-hundred, the odds are only 1 in 10,000 that the row contains a pair of matching values ( $100 * 100$ ). For all 100 rows, the odds are even worse--one in a million--that they *all* contain matching values ( $10,000 * 100$ ).

So let's issue the following query to a database interface utilizing Date's special values scheme: "How many rows in this table contain matching values?" The answer we will get, on Date's explicit definition, is: "All 100 of them." But since we don't know the real values for the pairs of numbers in *any* row, the odds of this being the *right* answer are, literally, one in a million! Is *this* is a scheme that any of us would want to use when querying a database for information?

---

<sup>2</sup> I find Date's name for the new domains--"XXXX\_OR\_UNK"--to be misleading, since the domains in question contain both the real values *and* UNK, not either one *or* the other. I therefore suggest that the reader will find Date's discussion a little clearer if they think of these domains as having the names "XXXX\_AND\_UNK", *i.e.* as domains containing all the XXXX values, *and also* the value UNK.

<sup>3</sup> References to Date will be to his five-part series on special values, starting in the October 1996 issue of this magazine. References will be placed in the text, in the form indicated here.

The consequences of the inadequacies of Date's approach, in the "real world", are as serious as you like. So, to make them serious, suppose that the columns of numbers are the firing coordinates, for each of one hundred targets, computed by independent targeting mechanisms for a cruise missile launching system. Next, let the fail-safe rule be that a missile will launch if and only if both targeting mechanisms agree on the coordinates. And, finally, let us suppose that conditions for targeting are so bad that, in all one hundred cases, both targeting mechanisms are unable to compute coordinates. Therefore, since the domain each is working with is TARGETING\_COORDINATE\_OR\_UNK, they put the special value UNK into their column, for each of the one-hundred rows.

We now use an interface based on Date's new scheme to determine which of the one-hundred targets we will fire at. We say to it: "For each of the one-hundred targets, fire if and only if the comparison 'Does coordinate 1 = coordinate 2?' returns *true*". And lo, all one-hundred cruise missiles lift into the air--in circumstances in which *no* coordinates are provided, by *either* targeting mechanism, for *any* of the targets!

Of course, if we had only had the foresight to have said, instead, "For each one-hundred targets, fire if and only if the comparison 'Does coordinate 1 = coordinate 2? AND Does coordinate 1 NOT = "UNK"?' returns *true*", then our missiles would *not* have fired.

Now Date may call this revised query a "solution" if he wants to. But I think we can all recognize it for what it is--evidence that something is wrong. For what is going on is that the user has been forced to *compensate* for an error in the semantics Date's scheme assigns to the equality operator. What has gone wrong, as all of us but Date and company know, is that the answer to the question "Are two unknown values equal?" is *not* "Yes they are"!

And, if Date's reply to this point is that he defined equality for his UNK special value, *not* for "unknown values", then he has merely conceded that his scheme using UNK values does not represent the semantics of the "real world's" unknown values.

### ***UNK and Inequality: the Basics.***

After this unfortunate attempt at defining equality, Date goes on to try to define greater-than and less-than. Running into immediate problems, he suggests that it is "debatable" that we would ever need to use these operators with an UNK operand. After struggling with a possible interpretation that would be as flawed as his interpretation of equality is, he settles on the position that to use these two inequality operators when one or both operands is UNK "makes no sense".

We can now see what I meant by saying that "what's wrong with the default value approach, from a semantic perspective, is that it has the semantics wrong, and so the ability of a database using default values to inform the inquirer fully is correspondingly hampered". (quoted in Pt 1, p.22, and again, while saying that he still "completely disagrees", in Pt. 5, p.18). For this position of Date's is precisely that his special values scheme cannot return any information to the user when either of these two operators is used in the presence of UNK. So, together with his incorrect handling of equality, I suggest that Date has now demonstrated the truth of this claim of mine.

### ***UNK, Inequality, and the "Real World".***

Up to this point, however, I have simply *assumed* that it makes sense to use the inequality operators (greater-than, less-than) with the UNK special value. But I think that Date would now attempt to argue that the basic question is what we would want to do in the "real world", when confronted with unknown values. For if refusing to give any answer to a question of the form "Is X greater-than (less-than)Y?", where X, Y or both may be unknown, is indeed what we would do "in the real world", then we would have sacrificed nothing at all with Date's approach to these two operators. On the contrary, we would have accurately captured the semantics of unknown values with UNK and two-valued logic (for greater-than and less-than, at least).

I shall now argue that the right response to a query using greater-than or less-than, when a value for one or both operands is unknown, is that the question certainly does make sense, but that the answer is *unknown* because

one or both of the operands is unknown. (And if this seems so obvious to you that you are wondering why I am trying to build an *argument* to demonstrate it, let me remind you that this is precisely what Date's scheme denies!)

Let's look, now, at exactly how much information is inexpressible with Date's scheme, even though the information which that scheme cannot give us *is* contained in the database!

Suppose now that our table of pairs of numbers represents one-hundred cases in which two players cut cards--the numbers 2 through 14 representing card values 2 through ace, and the first player being Jones and the second player Smith. Let's suppose, next, that (a) in 10 cases, both values are unknown, (b) in 23 cases one but not both values are unknown, and (c) in the remaining 67 cases, both values are known. Of those 67 cases, finally, let's assume (d) that 4 were ties, (e) that Jones beat Smith in 42 of those cases and, therefore, (f) that Smith beat Jones in the remaining 21.<sup>4</sup>

Now consider the following questions:

- How many times did Jones definitely beat Smith?
- How many times may Jones have beaten Smith?
- How many times did Jones and Smith definitely tie?
- How many times may Jones and Smith have tied?

Clearly, the answers to the questions are these:

- 42
- 75 (*i.e.*  $42 + 10 + 23$ )
- 4
- 37 (*i.e.*  $4 + 10 + 23$ )

The answers that Date's special values scheme, would give, however, are these:

---

<sup>4</sup> To avoid some unnecessary complexity relating to the highest and lowest values in a deck of cards, assume that in case (b) above, the known value is never 2 or 14, since if it were the former, the player could tie but not win the card cut, and if it were the latter, the player could tie but not lose the card cut.

- 42
- at least 42, and no more than 65--but with 23 instances of "invalid question"
- 14 (*i.e.* 4 + 10)
- at least 14 (*i.e.* 4 + 10), but with 23 instances of "invalid question"

Now let's compare results.

For case #1, both approaches give the same correct result.

For case #2, Date's scheme should have been able to determine that the correct answer is 75, but it couldn't because, for one thing, it couldn't count the 23 cases in which one of the operands was UNK. It couldn't count them because Date's scheme will return "undefined" (or "raise a run-time error") whenever it encounters an UNK value while evaluating a greater-than or less-than expression. However, clearly, in all 23 cases, Jones may indeed have beaten Smith. This demonstrates my point that there is information in the database which, given Date's scheme, he cannot provide to the user.

Note also that the correct answer--75--lies outside the range given by Date's scheme, so that the answer we get using his scheme is not only vague, but actually incorrect. This is because Date's scheme must count the 10 cases in which both numbers are UNK as ties, *i.e.* as cases where the two values are equal! Being equal, they are *not* possible cases in which Jones beat Smith, in Date's scheme. But in the "real world", unfortunately for Date, because they are instances of unknown values, they *are* possible cases in which Jones beat Smith.

For case #3, Date's scheme also gives an incorrect answer. Jones and Smith definitely tied in four cases, *clearly* not in 14! And why did Date's scheme give so obviously incorrect an answer? Again, because of Date's rule "...that the comparison UNK=UNK.....returns *true*.... No three-valued logic here!"

Of course, understanding the limitations of Date's scheme, we could have phrased our query looking for definite ties as follows: "...where column 1 = column 2, AND neither column = 'UNK'". *This* query would have returned the correct result--4. But this doesn't solve any problem in Date's scheme. It

just puts the burden of compensating for the fundamental semantic errors in his scheme directly on the user.

For case #4, Date's scheme gives a correct answer. However, this answer is less informative than it could be, because in all 23 cases, it could be the case that the two players tied.

Note that the answer that a database interface using nulls and MVL would give, are *exactly* those which I indicated are the clearly correct answers. This is because, with a three-valued logic, " $X = Y$ " returns unknown when either or both operands are unknown, and similarly for " $X > Y$ " and " $X < Y$ ".

So for question 4, for example, a database interface using nulls and MVL can reason thusly: "I know that Jones and Smith tied in four cases. In 10 cases, I don't know either value, so they may have tied in all ten of those cases, as well. And in 23 cases, I don't know one value, and so they may have tied in all of those cases, too. The total of all these cases is 37."

Date's attempt to convince us that, in the "real world", we don't use three-valued logic, flies in the face of this kind of everyday reasoning. Yet again, Date has provided further illustration of the truth of my statement that "what's wrong with the default value approach, from a semantic perspective, is that it has the semantics wrong, and so the ability of a database using default values to inform the inquirer fully is correspondingly hampered".

### ***Complexity.***

In his final installment, Date anticipates an obvious objection, *viz.* that his special values scheme certainly makes queries a lot more complicated than they were before. I have three comments about this feature of his new scheme.

First, a cursory glance at Date's extensive criticisms of multi-valued logic will show that underlying them is the repeated refrain that MVL is too complex, and hence too prone to generate errors in the "real world". I

therefore find it interesting that Date seems to be now hoist on his own petard of complexity. Perhaps he will explain to us why the complexity of MVL is a *bad* sort of thing, leading to errors in the "real world", while the complexity which his special values scheme requires the user to master is a *good* sort of thing--supposedly *not* leading to such errors.

Second, all this additional complexity in queries required by Date's scheme exists for one reason, and one reason only. It exists because the semantics of Date's scheme are either incorrect (for "equals") or incapable of returning information that does exist in the database (for "greater-than" and "less-than").

Finally, I do not doubt that Date will have many words to offer in reply. But I would suggest to the reader that he who publishes the last word is not therefore he who has the better position. Let us remember that the Copernican revolution did not succeed because it *proved* to the Ptolomaic theory's most ardent advocates that their theory was wrong. It succeeded because the complexities of epicycles within epicycles which new observations forced the Ptolomaic theory into, convinced everyone else *but* those ardent advocates that the theory was fatally flawed. I am content, therefore, to stand back and watch Date propose additional epicycles, to these already quite elaborate "query-related epicycles" he would require of us, for his new special values scheme.

### ***A Practical Special Values Scheme.***

In my introductory paragraph, I expressed a desire to see a two-valued logic default values scheme whose clarity and ease of use was judged a good trade-off for reduced expressive power. Since Date has not given us one, let me briefly outline such a scheme myself.

First of all, I agree with Date and company that we should eliminate nullable attributes in our databases, as much as possible. With some reservations (which I will not explain here), I also agree that a good way to do this is to subtype entities which contain a nullable attribute into two

subtype entities--one which contains the attribute as non-nullable, and the other which does not contain the attribute at all.

Second, I suggest that we reserve one or more values, from an attribute's usual domain, for the special values we wish to identify. Perhaps a single value, meaning "real value missing" will be all we need. Perhaps a distinction between (a) inapplicable, (b) unknown, or (c) "one or the other, not known which" is more appropriate.<sup>5</sup>

If possible, we should establish business rules which insure that the value we choose is not a homonym. For example, if an enterprise enforces a rule that invoice amounts cannot be zero, then we can use zeroes in the invoice-amount attribute to mean "real value missing".

However, sometimes this is not possible. Perhaps our enterprise *must* allow zero-amount invoices. In that case, "\$000,000.00" in the attribute means *both* (a) a zero-amount invoice, and (b) "real value missing".

If we must make a homonym out of a value, like this, we should be aware of the cost. Here, the cost is being unable to distinguish zero-amount invoices from invoices in which the actual amount is missing. This may well be too high a cost. In that case, we have other options, such as (a) adding a flag to distinguish zero-amount invoices from all others, or (b) shifting to a less critical and very infrequently used real value--say \$999,999.98"--to mean "real value missing".

In the former case, we have compensated for the homonymity, for we can now distinguish zero-amount invoices from invoices in which the invoice-amount is missing. What we now can't do is distinguish the (presumably less common) invoices for exactly \$999,999.98 from invoices in which the amount is missing. In this latter case, we have not eliminated the cost of the homonymity; but we have significantly reduced it.

---

<sup>5</sup> And in response to Date's frequently-expressed argument that there are any number of different kinds of nulls and, therefore, any number of truth values which a MVL would have to accommodate, I would reply that it seems to me somewhat naive to argue that every different way of stating a value-missing condition, in English, must correspond to a formally distinct kind of null. For the position that all the different ways in which a value-missing condition can be stated in English (or any natural language) can be reduced to the three mentioned here, see Atzeni and DeAntonellis, *Relational Database Theory* (Benjamin/Cummings, 1993), pp. 220 ff.

We have been using this kind of reduced-cost strategy for decades, actually, and in doing so, in each case, we have decided to let the semantic problem be resolved *outside* the system, *i.e.* by users who understand the convention, and the possibility of actually encountering instances of these homonyms.

Finally, note that the strategy of avoiding homonyms, *i.e.* of using a value to mean "real value missing" which does *not* mean anything else, gives us *exactly* the expressive power (and *exactly* the complexities) of Date's special values scheme. It does so because it amounts to finding a representation of Date's UNK within the standard domain for an attribute, instead of requiring vendors to extend that domain.

### ***Conclusion.***

So, with Date's new special values scheme, we have an alternative to MVL which (a) can't get equality right, (b) can't handle greater-than or less-than at all, (c) consequently returns both incorrect results and less-than-fully-informative results to queries, and (d) turns simple queries into rocket scientist queries.

Moreover, now that we understand how far from being implementable Date's scheme is, we can see that this extended debate between Date and company, and myself, has little immediate applicability to the practical work of developing databases and interfaces to them. As designers and users, we will continue to work within the constraints provided by our dialects of SQL.

The more conservative among us will avoid nulls and MVL because of understandable caution at the complexity. As I argued throughout my articles on this topic, the use of the additional inferencing power of MVL is a foray into a more complex dialogue with the database--a foray which should not be undertaken unless one is comfortable handling the complexity.

To assist the conservatives, therefore, I have sketched a two-valued logic approach to missing information that is *practical, i.e.* that can be used right now, with today's products. However, I claim no originality here. Indeed, I offer this sketch as little more than an existing best practice, dressed up in some fancy logical terminology, and accompanied by some explicit caveats.

As for the more adventurous among us, they will make use of nulls, as provided by our SQL dialects, and will probably get burned sometimes. But in the process, they will forge an understanding of nulls and MVL which they might not obtain from any textbook study. And so they will begin the process of rendering the complexity concomitant with the greater expressive power of nulls and MVL less daunting to us all.