A Critique of Claude Rubinson's Paper Nulls, Three - Valued Logic, and Ambiguity in SQL: Critiquing Date's Critique

C. J. Date

I'd like to thank Claude Rubinson for his thoughtful critique [3] of my remarks in reference [1] on nulls and three-valued logic (3VL). Clearly we're in agreement on the major issues; as Rubinson says, "I agree with Date that three-valued logic is incompatible with database management systems." We also agree that null isn't a value; as Rubinson says, "SQL defines null not as a value but a flag." However, I'd like to comment on three specific issues arising from Rubinson's article. Note: All otherwise unattributed quotes are from that article. Note too that I follow Rubinson (for the most part) in using the SQL terminology of tables, columns, and rows.

THE ORIGINAL EXAMPLE

The database I used as a basis for my examples in reference [1] looked like this (S = suppliers, P = parts):

S

SNO	CITY
S1	London

P

PNO	CITY
P1	

In this database, "the CITY is null" for part P1. What's more (as I said in reference [1]):

Note carefully that the empty space in [the] figure, in the place where the CITY value for part P1 ought to be, stands for nothing at all; conceptually, there's nothing at all?not even a string of blanks or an empty string?in that position (which means the "tuple" for part P1 isn't really a tuple, a point I'll come back to [later]).

I then posed the query "Get SNO-PNO pairs where either the supplier and part cities are different or the part city isn't Paris (or both)," and offered the following as "the obvious SQL formulation of this query":

```
SELECT S.SNO , P.PNO
FROM   S , P
WHERE  S.CITY <> P.CITY
OR P.CITY <> 'Paris'
```

I then showed that, given the sample database, the result produced by this SQL expression differed from the result that the user would expect from the original formulation (i.e., the natural language version) of the query. But Rubinson says:

The problem [with Date's example] is not that SQL's results disagree with reality but, rather, that Date poorly formulated his original query ... The formulated SQL statement does not, in fact, correspond to [the natural language] query; in fact, Date's query cannot properly be translated into SQL.

But that was exactly my point! I agree that "the formulated SQL statement" doesn't properly correspond to the natural language query; of course it doesn't, because it produces different results. In particular, pace Rubinson, I most certainly didn't claim that this state of affairs "indicates a flaw in SQL's logic." SQL's logic as such isn't flawed (at least, let's assume not for the sake of this discussion). Rather, what I did claim was that "SQL's logic" is different from the logic we normally use "in the real world." That's all.

In any case (and regardless of whether Rubinson agrees with me here or whether we simply agree to disagree), I really don't think it's worth wasting a lot of time on this particular example, nor on others like it. The real question is: How are we supposed to interpret the tables in the database? Which brings me to my next point.

THE ISSUE OF INTERPRETATION

Now, in reference [1], I deliberately did not spell out in detail how tables S and P were meant to be interpreted. That's because I knew that if I did so carefully enough, the fact that nulls are nonsense would have been completely obvious (implying among other things that it wouldn't have made much sense to discuss the sample query at all). The trouble is, the argument based on interpretation is a little esoteric and might, for some readers, be a little hard to follow; rightly or wrongly, therefore, I gave an argument that I thought would be intuitively easier to understand ("more accessible," as

Rubinson puts it). However, let me give that argument based on interpretation now.

First of all, in case readers aren't familiar with the terminology I'll be using, let me explain that:

1. Each table t is supposed to correspond to some predicate pred.

2. If table t has n columns, then predicate pred has n parameters.

3. Each row r in table t contains n column values. Further, each such row is supposed to correspond to some proposition prop: namely, a proposition obtained from predicate pred by using the n column values from r as arguments to replace the n parameters in pred (each such proposition is thus an instantiation of the predicate pred).

4. Each proposition prop so obtained?i.e., each such instantiation of predicate pred?is one that we believe, or know, to be true [2].

Now, Rubinson appears to be arguing in reference [3] that it's the logical difference between (a) something being true, and (b) our knowing that it's true, that lies at the heart of our difficulties with 3VL. In fact, however, we have to pay attention to that difference even without nulls and 3VL (see point 4 above), though it's certainly the case in practice that we often don't. Thus, I think Rubinson's argument here is something of a red herring. What's more, as I show in reference [2], we can still get "don't know" answers, even out of a database without nulls and without using 3VL?but that's a red herring too, perhaps. Let me get back to the issue at hand.

Consider table P. That table has two columns, PNO and CITY, and so whatever predicate it represents must have two parameters. What is that predicate? Well, the obvious candidate is: Part PNO is stored in city CITY. But we need to be more precise than that. In fact, in accordance with the remarks in the previous paragraph, a more reasonable candidate is: We know that part PNO is stored in city CITY.

But now suppose we don't know where part P1 is stored. Then a true proposition of the form We know that part P1 is stored in city CITY simply doesn't exist!?it simply isn't the case that we know, for any specific value of CITY whatsoever, that part P1 is stored in city CITY. (Note: Presumably we do know it's stored somewhere, because all parts are stored somewhere, but We know that part P1 is stored somewhere is a completely different proposition.)

Since no true proposition of the pertinent form exists, it follows that no corresponding row exists, either. And so no row for part P1 can appear in the table.

All right, then: Accepting for the moment that a row for part P1 (with a "null city") does in fact appear in the table after all, we must have the predicate wrong. Perhaps it should be:

Exactly one of the following is true: (a) we know that part PNO is stored in city CITY; (b) we don't know the city for part PNO.

(Note that there must be an exclusive, not inclusive, OR connecting the two sections (a) and (b) of this predicate. We can't allow the same part to have both a known and an unknown city.)

Observe now, however, that section (a) of this predicate has two parameters (PNO and CITY), while section (b) has just one (PNO). It follows that rows representing true instantiations of section (a) have two column values and rows representing true instantiations of section (b) have just one. It further follows that these two kinds of rows can't logically both appear in the same table. Thus, to talk of some row r in some table t as "containing a null" is, as I said before, nonsense? or at least (and this is really a better way to put it), it's a contradiction in terms.[1]

Perhaps I should add that a design that does faithfully represent the situation?and doesn't involve nulls, of course? would have two separate tables: (a) table P, with columns PNO and CITY and predicate We know that part PNO is stored in city CITY, and (b) table P', say, with a single column PNO and predicate We don't know the city for part PNO.

DO NULLS VIOLATE THE RELATIONAL MODEL?

Although he does agree with me that nulls and 3VL are undesirable, Rubinson says he is "not convinced that three-valued logic violates the relational model." But it does! The arguments of the previous section, as well as others not articulated here, clearly demonstrate that a table that "contains a null" doesn't correspond to a relation in the relational model sense, because it fails to satisfy the basic relational requirement that every row in that table contains a value for every column. Thus, the fundamental object in a system that supports nulls isn't a relational table (I don't know what it is, but it isn't a relational table). Indeed, to repeat what I said in reference [1] (and here I revert to traditional relational terminology):

- A "type" that contains a null isn't a type (because types contain values).
- A "tuple" that contains a null isn't a tuple (because tuples contain values).
- A "relation" that contains a null isn't a relation (because relations contain tuples, and tuples don't contain nulls).

Taken all in all, therefore, I believe this short paper serves to bolster the claim I made in reference [1] to the effect that, if nulls are present, then we're certainly not talking about the relational model. In other words, I stand by my claim that nulls (and 3VL) and the relational model are mutually incompatible.

[1] Incidentally, note the implications here for outer join.

REFERENCES

1. C. J. Date: Database in Depth: Relational Theory for Practitioners. Sebastopol, Calif.: O'Reilly Media, Inc. (2005).

2. C. J. Date: "The Closed World Assumption," in Logic and Databases: The Roots of Relational Theory. Victoria, BC: Trafford Publishing (2007). See www.trafford.com/07-0690.

3. Claude Rubinson: "Nulls, Three-Valued Logic, and Ambiguity in SQL: Critiquing Date's Critique," ACM SIGMOD Record 36, No. 4, December 2007.