Nulls, Three-Valued Logic, and Ambiguity in SQL: Critiquing Date's Critique

Claude Rubinson*
Department of Sociology
University of Arizona
rubinson@u.arizona.edu

Abstract

Date's popular critique of SQL's three-valued logic [4, 3] purports to demonstrate that SQL queries can produce erroneous results when nulls are present in the database. I argue that this critique is flawed in that Date misinterprets the meaning of his example query. In fact, SQL returns the correct answer to the query posed; Date, however, believes that he is asking a different question. Although his critique is flawed, I agree with Date's general conclusion: SQL's use of nulls and three-valued logic introduces a startling amount of complexity into seemingly straightforward queries.

1 Introduction

A common critique of SQL is that the inclusion of nulls breaks the relational model. Date enumerates a number of reasons for this position. Most fundamentally, Date argues that—since SQL defines null not as a value but a flag indicating that the value of a particular attribute is missing—domains cannot properly include nulls since domains are, by definition, sets of values. Therefore, relations that include nulls are not, in fact, relations, undermining the very foundation of the relational model [3]. Date also make a more accessible argument in which he contends that the three-valued logic incurred by the use of nulls can generate nonsensical results. In this essay, I critique this second argument and demonstrate that Date misapplies SQL's three-valued logic. Consequently, the critique is logically flawed and does not, in fact, indict SQL as Date supposes. Note, however, that my critique of Date is not a defense of nulls or SQL's three-valued logic; rather, it underscores just how confusing three-valued logic is. The introduction of nulls alters the meaning of seemingly straightforward queries and is likely responsible for numerous errors, errors which may frequently go unrecognized.

2 Date's Critique

Date's most prominent critique of nulls employs the simple SQL database illustrated in Figure 1. There are two tables. The Suppliers table (S) has two columns: the supplier number (the primary key) and the supplier's city. The Parts table (P) also has two columns: the part number (the primary key) and the part's city. In Figure 1, each table has only one record. Supplier S1 is located in London. We do not know in which city Part P1 is located.[1]

*I would like to thank Garrett Hoxie and Rick Snodgrass for their advice and support of this paper. I also wish to thank the *SIGMOD Record* reviewers for their helpful comments.

[1]Nulls often introduce confusion when it is unclear why information is missing from the database. Among the more common reasons for incomplete data entry are that the value of the attribute is (temporarily) unknown or that the attribute, itself, is not applicable to the represented entity. With regard to the present example, Date's discussion of the database described in Figure 1 makes it clear that the NULL marker in Table P indicates that the city associated with Part P1 is (temporarily) unknown. I proceed with this premise. In the conclusion, I return to this topic and discuss the additional complications

S

SNO*	CITY
S1	London

P

PNO*	CITY
P1	NULL

Figure 1: SQL Database

Date [4, page 54] seeks to demonstrate that SQL's three-valued logic produces erroneous results:

> The fundamental point I want to make is that certain boolean expressions—and therefore certain queries—produces results that are correct according to three-valued logic but *not* correct in the real world.

To do so, he poses the following query: "Get SNO-PNO pairs where either the supplier and part cities are different or the part city isn't Paris (or both)" [4, page 54] and writes the corresponding SQL implementation of the query:

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

Substituting in the data from the mock database, the expression `(S.CITY <> P.CITY) OR (P.CITY <> 'Paris')` becomes `('London' <> NULL) OR (NULL <> 'Paris')` which, in accordance with the rules of SQL's three-valued logic, evaluates to `(NULL OR NULL)` which, in turn, reduces to `NULL`. The query, therefore, returns no records.

Date [4, page 55] contends that this result reveals a flaw in SQL's three-valued logic, arguing that:

> But of course part P1 does have *some* corresponding city in the real world; in other words, "the null CITY" for part P1 does stand for some real value, say *xyz*. Obviously, either *xyz* is Paris or it isn't.

Date then demonstrates that the `WHERE` clause will always evaluate to `TRUE`, regardless of where part P1 is located. In essence, there are three possibilities: city *xyz* is Paris, London, or some other city. If city *xyz* is Paris, the above expression becomes `('London' <> 'Paris') OR ('Paris' <> 'Paris')`. This expression evaluates to `(TRUE OR FALSE)` which, in turn, evaluates to `TRUE`. If city *xyz* is London, the expression becomes `('London' <> 'London') OR ('London' <> 'Paris')` which evaluates to `FALSE OR TRUE` which evaluates to `TRUE`. If city *xyz* is some other city, for example, New York, the expression becomes `('London' <> 'New York') OR ('New York' <> 'Paris')` which evaluates to `(TRUE OR TRUE)` which, again, reduces to `TRUE`.

According to Date, if SQL correctly took account of the real world—specifically, that part P1 is associated with some city, despite that this fact is missing from the database—it should return the pair S1-P1. That SQL returns an empty set indicates a flaw in its logic: "In other words, the result that's correct according to the logic (that is, 3VL) and the result that's correct in the real world are different!" [4, page 55].

3 Critiquing the Critique

But Date is mistaken. The problem is not that SQL's results disagree with reality but, rather, that Date poorly formulated his original inquiry. Recall Date's original query: "Get SNO-PNO pairs where either the supplier and part cities are different or the part city isn't Paris (or both)." The formulated SQL statement does not, in fact, correspond to this query; in fact, Date's query cannot properly be translated into SQL because it assumes conventional, two-valued logic while SQL operates with three-valued logic.

In conventional logic, propositions are true or false. That is, part P1 is in Paris or it is not. In the three-valued logic employed by SQL, propositions are true, false, or unknown. By introducing the possibility of unknown propositions, it is no longer the case that part P1 is or is not in Paris. Rather: we know that part P1 is in Paris, we know that part P1 is not in Paris, or we don't know where part P1 is.

The logic system within which one works demands that queries be formulated appropriately. Within a conventional, two-valued logic system, statements must be able to be classified as "true" or "false."

that arise when the meaning of a null is ambiguous.

Within a three-valued logic system, statements must also permit a classification of "unknown." Date's original query assumes two-valued logic. Consider the first clause of the query: "Get SNO-PNO pairs where ... the supplier and part cities are different." This query assumes that supplier and part cities are different or that they are the same. But within SQL's three-valued logic system, supplier and part cities may be the same, they may be different, or we might not know if they are the same or different. The second clause of the query is similar: "Get SNO-PNO pairs where ... the part city isn't Paris." Again, this query assumes that the part city is or is not Paris. Within SQL, however, the part city may be Paris, it may not be Paris, or we might not know what city it is. And, in fact, the null value in the database indicates that we do not know which city is associated with part P1.

Date argues that "in the real world" the city for part P1 either is or is not Paris. This is certainly true. But it is also true that "in the real world" we may not know what city is associated with part P1. These are two different propositions. The cities to which parts correspond is a set of facts that is distinct from whether we know which cities correspond to which parts. In SQL, queries always imply knowledge of the relationship in question and not simply the existence of said relationship. We can therefore reformulate Date's original query as "Get SNO-PNO pairs where either we know that the supplier and part cities are different or we know that the part city isn't Paris (or both)." The results of the SQL statement now make sense. An empty set is returned because—even though part P1 "does have *some* corresponding city in the real world"—we do not know to which city the part corresponds.

This understanding of the incongruity between two-valued and three-valued logic is made more clear by examining Date's second example. Date [4, page 55] presents the following SQL statement

```
SELECT P.PNO
FROM   P
WHERE  P.CITY = P.CITY
```

and contends that "The real-world answer here is obviously the set of part numbers currently appearing in P." What is obvious is that Date thinks that the above SQL syntax is functionally equivalent to the statement "Get the PNO numbers for the parts that are associated with cities." Because "in the real-world" all parts must be associated with a city, Date concludes that the query should return a set of part numbers. But Date is again misreading the query. Because SQL uses three-valued logic the statement expresses a distinctly different query: "Get the PNO numbers for the parts for which we know the associated city." Again, SQL correctly returns an empty set because, according to table P, we do not know which city is associated with part P1.

Date [4, page 55] contends that his examples demonstrate that SQL is fundamentally broken:

> To sum up: if you have any nulls in your database, you're getting wrong answers to some of your queries. What's more, you have no way of knowing, in general, just which queries you're getting wrong answers to; *all* results become suspect. *You can never trust the answers you get from a database with nulls.* In my opinion, this state of affairs is a complete showstopper. (Emphasis in original.)

I have shown that Date has not demonstrated what he thinks he has. SQL returns the correct answer for the query posed but Date believes that he is asking a different question. This confusion is understandable. SQL's three-valued logic is not intuitive. We are used to two-valued logic in which propositions are either true or false. But three-valued logic also permits unknown propositions. When working with SQL databases, it is imperative that we formulate our queries correctly; otherwise, we risk making mistakes similar to Date.

4 Discussion

SQL's use of three-valued logic and its inclusion of the null marker requires that we formulate our database queries to reflect the possibility that the relationships between entities may be unknown. When we fail to

do so, we risk posing a different question than intended. We must keep in mind that SQL's logic is non-intuitive. Rarely will the questions we put to a SQL database approach what we would ask in normal conversation. We cannot simply ask for the "SNO-PNO pairs where the supplier and part cities are different;" rather, we must ask for "SNO-PNO pairs where the supplier and part cities are known to be different." More crucially, we must understand the difference between these two formulations.

The problem is only aggravated by the fact that information can be missing from a database for a variety of reasons. Date [1] identifies seven common causes of incomplete data entry: *value not applicable*, *value unknown*, *value does not exist*, *value undefined*, *value not valid*, *value not supplied*, and *value is the empty set*. If a value might be missing due to, for example, an inapplicable attribute, queries must be formulated and interpreted in consideration of this potential condition. When null markers are loaded with multiple meanings, the construction of associated queries rapidly becomes unmanageable: "Get SNO-PNO pairs where the part city attribute is applicable and either we know that the supplier and part cities are different or the part city isn't Paris (or all three conditions apply)." To address this latter situation, some practitioners advocate the use of descriptive truth values [5, 7]. Constituting actual values rather than null markers, such solutions permit designers to construct databases that do not permit nulls and, consequently, may be queried using conventional, two-valued logic.

It may also be useful to note that any query that assumes three-valued logic may be decomposed into two correlated queries assuming two-valued logic.[2] Take, for example, the query "Get SNO-PNO pairs where we know that the part city isn't Paris." As discussed above, this query assumes three-valued logic because, for any given SNO-PNO pair, the part city may be in Paris, it may not be in Paris, or the part city may be unknown. This query may be decomposed into the compound query "Get SNO-PNO pairs where we know the part city and, from the resulting set, get SNO-PNO pairs where the part city is not Paris." It is often helpful to perform this decomposition, particularly when constructing complex queries.

Ultimately, I agree with Date that three-valued logic is incompatible with database management systems. While I am not convinced that three-valued logic violates the relational model *per se*, I agree with McGoveran [6, page 355] that

> many-valued logic means that database designers, developers, and users must all learn a whole new way of thinking. The practical costs of this approach are hard to assess; certainly they do violence to the goals we set out to satisfy with an RDBMS.

That Date, himself, misinterprets the meaning of his SQL syntax underscores the severity of the problem.

5 Conclusion

We develop databases in order to organize and make sense of information. The problem is that the world is complex. One manifestation of this complexity is that we sometimes lack complete information. I echo those who suggest that SQL practitioners avoid nulls as much as possible. By default, database designers should constrain columns as non-nullable. Operations that generate nulls such as outer joins should be avoided when possible, particularly as the basis of views and subqueries. Since, by definition, nulls indicate exceptional circumstances, nullable columns often indicate where the database design might be improved. The use of nulls in SQL is not the most fundamental concern raised by the database presented in Figure 1. Rather, it is: Where the heck is part P1? If part P1 is in transit to Paris, that information needs to be recorded in the database. So too if part P1 is lost. Notably, inclusion of such information elsewhere in the database increases both the value of the database as well as its integrity by permitting the problematic record to be dropped.

Proper design techniques, then, naturally minimize the number of nulls in the database. A database design is a model of a particular domain and it is only by thoroughly interrogating that domain—by

[2] I thank Charles Ragin for clarifying this principle for me.

circumscribing its boundaries, delineating its constituent components, and identifying the relationships therein—that one can produce an accurate representation. Of course, the goal of a database design is not to represent a domain perfectly but only those aspects that are salient to the problem at hand. If part P1 is on a truck bound for Paris, its projected arrival time is probably relevant; that the truck driver just had a fight with his spouse, probably not. Nulls permit us to simplify our models by generalizing across anomalies that produce missing data and unknown relationships. But the cost of this simplified representation is three-valued logic and the associated increase in the complexity of our queries.

It is rare that one can guarantee the complete absence of nulls from a database. Even if database vendors were persuaded to deprecate nulls and three-valued logic, we would remain saddled with them for the foreseeable future. And since the presence of a single null value taints the entire database [6], one must generally assume three-valued logic. Consequently, the burden is on us to carefully review our queries to ensure that they mean what we intend.

References

[1] C. J. Date. Not is not 'not'! (notes on three-valued logic and related matters). In *Relational Database Writings, 1985–1989*. Addison Wesley, 1990.

[2] C. J. Date. *Relational Database Writings, 1994–1997*. Addison Wesley Longman, 1998.

[3] C. J. Date. *An Introduction to Database Systems*. Addison Wesley Longman, Reading, MA, seventh edition, 2000.

[4] C.J. Date. *Database in Depth: Relational Theory for Practitioners*. O'Reilly, Sebastopol, CA, 2005.

[5] G. H. Gessert. Handling missing data by using stored truth values. *SIGMOD Record*, 20(3):30–42, Summer 1991.

[6] David McGoveran. Nothing from nothing (part 2 of 4) classical logic: Nothing compares 2 u. In *Relational Database Writings, 1994–1997* [2], chapter 6, pages 347–365.

[7] David McGoveran. Nothing from nothing (part 4 of 4): It's in the way that you use it. In *Relational Database Writings, 1994–1997* [2], chapter 8, pages 377–394.