Null Values in SQL

John Grant
Department of Mathematics, Towson University
Towson, MD 21252 and
Department of Computer Science, University of Maryland,
College Park, MD 20742
jgrant@towson.edu

ABSTRACT
In various writings over the past 20 years, such as [3], Date has pointed out that SQL produces incorrect answers to some queries where a null value is included in a table. In a recent article in the ACM SIGMOD Record, [8], Rubinson states that "Date misinterprets the meaning of his example query" and "SQL returns the correct answer to the query posed". The purpose of this article is to show that, contrary to Rubinson's claim, Date's critique of query evaluation in the presence of null values in SQL is completely justified.

1. INTRODUCTION
In the past 20 years, in various writings Date pointed out flaws in the method of query evaluation in SQL in the presence of null values. The problem is due to the way that a 3-valued logic is used in SQL to evaluate such queries. Actually, there are various types of null values; this article deals only with the type where the value the null represents exists but is unknown.

In a recent article in the ACM SIGMOD Record, [8], Rubinson claims that "Date misinterprets the meaning of his example query" and "SQL returns the correct answer to the query posed". The purpose of this article is to give some historical background to the evaluation of queries in relational databases in the presence of null values and to refute Rubinson's claim. The fact is that Date's critique is correct and completely justified. Section 2 reviews Date's example, as given in [8]. Section 3 provides a historical overview of the evaluation of queries in relational databases involving null values. Section 4 shows that various generalizations extending null values also support Date's critique. The paper ends in Section 5 with a brief discussion.

2. DATE'S EXAMPLE
This material is taken directly from [8], copied from Date's example in [3], using a slightly different notation. The example database has 2 tables: Suppliers(sno,city) and Parts(pno,city). The primary keys are sno and pno respectively. Each table contains a single row: Supplier has $< S1, London >$ and Parts has $< P1, Null >$. Part P1 has a city whose identity is unknown, hence the null value. Date's query in English states "Get sno-pno pairs where either the supplier and parts cities are different or the part city isn't Paris (or both)". In SQL this is written as

```
Select sno, pno
From Suppliers, Parts
Where Suppliers.city <> Parts.city
Or Parts.city <> 'Paris';
```

For this table and query SQL returns the empty table as the result. However, as Date explains, the correct answer is the tuple $< S1, P1 >$. Date considers 3 possibilities for P1's city: Paris, London, or some other city. Actually, there are only 2 relevant cases: either P1's city is Paris or it is not Paris. In the former case the Where condition is true because S1's city is London and `London <> Paris` is true. In the latter case the Where condition is true because `Parts.city <> 'Paris'` is true. No matter what city the unknown Null value represents, the tuple $< S1, P1 >$ satisfies the query and should be in the answer. This simple example illustrates the flaw in SQL pointed out by Date.

3. HISTORICAL BACKGROUND
In the early 1970s in a series of highly influential papers E. F. Codd introduced the relational database model including the relational calculus, relational algebra, and relational database normalization. He also had a column in the predecessor to the ACM SIGMOD Record, called the FDT Bulletin of ACM-SIGMOD, where he explained various relational database concepts in a series of installments.

In [1] he answered a question about handling queries in the presence of null values in a relational database. He used the relational calculus language for illustration. Codd proposed a 3-valued logic with truth values 'True', 'False', and 'Unknown'. When a null value appears in a table, its evaluation in a condition produces the 'Unknown' truth value. He gave truth values to complex conditions by giving truth-tables for the connectives 'and', 'or', and 'not'. For example, 'True or Unknown' has the truth value 'True' because a disjunction is true if one of its disjuncts is true. Codd evaluated 'Unknown or Unknown' to 'Unknown'. In Date's example both `Suppliers.city <> Parts.city` and `Parts.city <> 'Paris'` are evaluated as 'Unknown' for P1's row; hence Codd's method evaluates the combined condition to 'Unknown'.

I recall reading Codd's article in the summer of 1976 when I was visiting at the University of Pennsylvania. I immediately realized that I have already dealt with this issue in a different context several years earlier. In [4] using Kleene's 3-valued logic I showed that a truth-functional (i.e. the connectives are defined by truth-tables) 3-valued logic, where the third truth value stands for "unknown", will not give some formulas the correct truth value, and proposed a non-truth-functional 3-valued logic that gives all formulas correct truth values. In the case of null values for a relational database this means that the 3-valued logic truth tables used by Codd (the same as in Kleene's 3-valued logic) do not always give correct answers to queries. First I wrote to Dr. Codd explaining the problem and after his reply I wrote a short article [5] pointing out the problem. In fact I used as my example a Suppliers table and a condition Suppliers.city = 'Paris' taken from Date's pioneering textbook [2] (the first edition!). I also proposed the solution I gave in [4] translated appropriately to relational database queries: for the correct evaluation of a query in the presence of a null value, all different cases must be considered. This is exactly what I did for Date's example in the previous section where there were 2 cases: either the city is Paris or it is not Paris. When all cases evaluate to 'True' for a tuple, that tuple should be in the answer. Incidentally, I also showed in [5] how to handle the case where the null value stands for an inapplicable attribute, such as the spouse of a person who is not married.

In the late 1970s null values were generalized by several researchers (including me) to the concept of incomplete or partial information (a concept I investigated in the early 1970s in a logic context). By the early 1980s in his pioneering textbook on database theory, [7], Maier devoted a whole chapter to this topic. The first standard for SQL was published several years later, in 1986, by the American National Standards Institute. In spite of my work 10 years earlier, Codd's proposal, suitable modified from the relational calculus to SQL, was adopted for standard SQL. After the standard was established Date began to criticize it for various reasons, including its handling of null values.

4. EXTENSIONS OF RELATIONAL DATABASES

Database researchers have done a tremendous amount of work over the past 30 years adding various capabilities to relational databases. Some of these efforts generalize the concept of null values in various ways. This section considers two such generalizations: disjunctive databases and probabilistic databases, considering how Date's example would be treated.

A disjunctive database, [6], allows disjunctive facts such as, for Date's example,
P1.city = 'Paris' or P1.city = 'London' or P1.city = 'New York'. If these are all the allowed cities then the meaning of the statement is the same as P1.city = Null, but if there are more cities then the former statement is stronger because it restricts P1's city to be one of the above 3 values. The facts can be written as follows:
$Suppliers(s1, london) \leftarrow$
$Parts(p1, paris) \vee Parts(p1, london) \vee Parts(p1, newyork) \leftarrow$
The query is written as 2 definitions for the query predicate Q because of the disjunction:
$Q(Sno, Pno) \leftarrow Suppliers(Sno, City1), Parts(Pno, City2), City1 \neq City2$
$Q(Sno, Pno) \leftarrow Suppliers(Sno, City1), Parts(Pno, City2), City2 \neq paris$
A disjunctive database will then give $< s1, p1 >$ as the answer to the query $\leftarrow Q(Sno, Pno)$.

A probabilistic database can be defined as a probability distribution on the set of instances [9]. In this case the information about the identity of the null value is probabilistic. Suppose, for instance that in Date's example there are 3 possible worlds: all 3 have Supplier(S1,London), but for Parts, let the probabilities be assigned as follows, Pr(Parts(P1,London)) = .5, Pr(Parts(P1,Paris)) = .3, Pr(Parts(P1, New York)) = .2. Consider now Date's query. Both the possible answers semantics and the possible tuples semantics give the value Pr($< S1, P1 >$) = 1. That is, again, $< S1, P1 >$ is in the answer with probability 1. The same answer is obtained no matter how many cities there are or how the probabilities are distributed among the cities for part P1.

5. DISCUSSION

Over many years Date criticized the evaluation of queries in SQL involving null values. This article explained that the SQL evaluation of such queries follows a proposal made by Codd that I showed incorrect (in some cases) over 30 years ago. The semantics of various extensions to relational databases proposed by researchers over the past 30 years agree with the meaning of the example query as given by Date. Rubinson's claim that "Date is mistaken" is incorrect.

It is appropriate to end this article refuting Rubinson's article by one more quotation that clearly illustrates his misunderstanding of the issue: "Date's query cannot properly be translated into SQL because it assumes conventional, two-valued logic while SQL operates with three-valued logic." Of course, Date's query can be translated into SQL, just as Date did it (see Section 2). Rubinson appears to assume that the evaluation method used in SQL is intrinsic to the language, but that is not the case. As I explained in Section 3, the query evaluation method used in SQL is not intrinsic to relational databases in general, or SQL in particular; it is just a choice made by the committee that standardized the language. So the problem is not that SQL uses three logic values rather than two; the problem is in the way that SQL uses the three-valued logic in query evaluation.

6. REFERENCES

[1] E. F. Codd. Understanding relations (installment #7). *FDT Bulletin of ACM-SIGMOD*, 7(3-4):23–28, 1975.

[2] C. J. Date. *An Introduction to Database Systems.* Addison-Wesley Publishing Co., Reading, MA, 1975.

[3] C. J. Date. *An Introduction to Database Systems, 7th Edition.* Addison-Wesley Publishing Co., Reading, MA, 2000.

[4] J. Grant. A non-truth-functional 3-valued logic. *Mathematics Magazine*, 47(4):221–223, September-October 1974.

[5] J. Grant. Null values in a relational data base. *Information Processing Letters*, 6(5):156–157, October 1977.

[6] J. Lobo, J. Minker, and A. Rajasekar. *Foundations of Disjunctive Logic Programming.* The MIT Press, Cambridge, MA, 1992.

[7] D. Maier. *The Theory of Relational Databases.* Computer Science Press, Rockville, MD, 1983.

[8] C. Rubinson. Nulls, three-valued logic, and ambiguity in sql: Critiquing Date's critique. *ACM SIGMOD Record*, 36(4):13–17, December 2007.

[9] D. Suciu and N. Dalvi. Foundations of probabilistic answers to queries. In *Tutorial at SIGMOD'05*, 2005. `http://www.cs.washington.edu/homes/suciu/tutorial-sigmod2005.pdf`.