Nulls in Relational Databases: Revisited

By
Raju Kocharekar
International Bank for Reconstruction and Development
1818, H st, Washington, DC 20433

Abstract

This paper discusses the semantic issues related to null values problem in relational databases. We argue that the proposed set of maybe operations with the three valued logic is still not adequate and needs further enhancements.

1. Introduction

Support for the null values in existing relational database management systems is ad-hoc in nature and has led to confusion because of the inaccurate or ambiguous results from queries on tuples with nulls. Many researchers [Codd,86], [Codd,87], [Date,82] have addressed these problems and proposed solutions. The main thrust of the proposals is to have a new set of MAYBE operations on the relational databases. In informal terms, the maybe operations yield not only the tuples for which the query predicate holds true, but also those tuples for which the predicate may be true. The maybe operations are based on the three valued predicate logic. We briefly outline the methodology below.

For all atomic predicates 'x theta y', where theta is =, NOT=, <, NOT <, >, and NOT >, if either x or y or both are null then the result of the operation is null (denoted by '?' later). For all arithmatic operations, such as x+y, if either of the arguments is null then the result is null. The truth tables for the three logic operators 'and', 'or' and 'not' are:

AND	T	?	F
T	T	?	F
?	?	?	F
F	F	F	F

OR	T	?	F
T	T	T	T
?	T	?	?
F	T	?	F

NOT	
T	F
?	?
F	T

E.F. Codd [Codd,87] has also proposed a four valued logic based on the fact that two types of null values exist in the database; the 'applicable null' values and the 'inapplicable null' values. The former nulls indicate that the corresponding attribute value is currently unknown, but may later be replaced by a valid nonnull value. The later null indicates that the corresponding attribute is not applicable for the tuple in consideration. This null value can never be changed. In [Koch,?], we have argued that inapplicable null values are redundant with a new design methodology. The details of the proposal are beyond the scope of the paper. We restrict ourselves to the three valued logic.

2. Problems with MAYBE operations as it stands today:

2.1 Proble One.

Maybe operations have solved many of the null value problems. However, the definition of maybe operations still needs to be enhanced to get useful answers. To give an example, consider the following personnel database.

Employee

Empno	WorkDeptNo	Salary
E01	D101	40,000
E02	D101	45,000
E03	D202	?
E04	?	?
E05	D303	30,000

Department

Deptno	MgrNo	DeptName
D101	E01	Engineering
D105	?	Marketing
D202	E03	Accounting
D303	?	Research
D404	?	Personnel

```
The following referential integrity constraints exist.
Range of x is Employee
Range of y is Department
(for all x) ((x.WorkDeptNo is null)
     or ((there exists y) (y.Deptno=x.WorkDeptNo)))
(for all y) ((y.MgrNo is null)
     or ((there exists x) (y.Mgrno=x.Empno)))
```

Now if we ask the query: "find the employees that MAYBE working in D404", we get the resulting employee tuple with empno E04. If we ask the same question for WorkDeptNo D505, we still get the same answer. The difference between the first and the second query is that the department D505 does not exist. This is indicated by the fact that there is no corresponding tuple in the Department relation. We therefore argue that the answer to the second query is wrong.

At this point we digress slightly from the null value problem and talk about the entity-relationships in the database design. To begin with, all entities in the database are defined on domains. Many DBMSs support primitive domains such as integers and characters. In this context, domains are similar to types in the programming language. In particular, the domain membership check is done at the query compilation time and is based on the syntactical notation. In the database design process, we define kernel entities that are unqiuely identified by a value that is a member of a particular domain. Most of the domains have infinite number of possible members. At a given time, the database holds a set of entity values that is a subset of the domain. For example, the employee and deparments are kernel entities uniquely identified by the empno and the deptno over the domain of character strings. In addition, the kernel and associative (defined later) entities have properties that are other kernel or associative entities and are designated by the identifiers of those entities. Thus, Salary and WorkDeptNo are properties of the employee entity. As you may already have noticed, we call salary as an entity, but do not have a corresponding salary relation that identifies all salaries. The problem is that we are not interested in identifying all salary values. More important, we assume that the Salary relation contains all values that are possible in the domain, and this leads to an infinite relation. We state that the DBMS is intelligent enough to postulate the existence of such a salary relation for all practical purposes. We define associative entities to be those entities for which we do not have a seperate identifier but use a combination of one or more identifiers of other entities. We mention that this is only for convenience and assume that the associative entities are the same as kernel entities for further discussion. We also assume that the DBMS is enriched with the referential integrity management and allows the database designer to define all primary and foreign keys for the above relationships.

What we have achieved from this discussion is the introduction of a dynamically defined domain for a property of the entity. In formal terms, the only permissible non-null values in the property of an entity are those that exist in the corresponding entity relation for that property. For example, the Workdeptno property of the Employee relation can only have those values that currently exist in the Deptno of Department. If the Workdeptno value is currently unknown, the valid assumption is that the Workdeptno posseses any one of the existing values from the Deptno in the Department relation. Therefore in the example queries above, the predicate 'Employee.WorkDeptNo = D404' has a value 'maybe' for the tuples with unknown Workdeptno, while the predicate 'Employee.WorkDeptNo = D505' has a value 'false'.

2.2 Problem Two.

[Gran,77], [Lips,79] and [Codd,87] discuss MAYBE operations on tautologies. They show that in some cases where the arguments have null values, the MAYBE predicates yield a 'maybe' instead of a 'true'. The following example shows one such case.

(Employee.Salary >= 25,000) Or (Employee.Salary < 25,000)
- example 1

We discuss the inverse case where the result of the predicate should always be false. Consider the following example predicate:

```
(Employee.Workdeptno = 'D404') And (Employee.Workdeptno =
    'D303')                                      example 2
```

It is hard to believe that a user would explicitely pose such a predicate, but if we assume that one of the predicates is embedded in a view then it is concievable to have such a select criteria in the query obtained after merging the view definition with the user query. The evaluation of the above predicate with three-valued logic leads to a 'maybe' in cases where workdeptno is null. Note that this case is worse than the tautology because now we see an incorrect answer set. For tautologies, though the predicate value is different than what it should be, we get the correct answer set of tuples.

Codd [Codd,86] describes the problem as 'non-traumatic' and one that could temporarily be ignored. Unfortunately, as we show below, the problem with maybe set of operations exists not just in tautologies, but in other predicates where the argument containing a null value appears more than once. The following example illustrates such a case.

```
(Deptartment.Deptno=Employee.Workdeptno) and
    (Employee.WorkDeptno = 'D404')           - example 3
```

Consider a case where Department.Deptno has a value D303 and Employee.Workdeptno is null. The predicate evaluates to a 'maybe' with the three valued logic. The correct value should however be a 'false'. The problem is very clear. Employee.Workdeptno is assumed to have two different values in evaluating two different parts of the predicate.

[Gran,77] and [Lips,79] have proposed solutions to the problem. We discribe here a modified version of the Grant's proposal. In the second example discussed above, if the workdeptno contains a null for the tuple in consideration, we temporarily substitute it with D404 before evaluating the predicate Employee.Workdeptno = 'D404'. The predicate then holds 'true'. We retain the temporary value of the workdeptno for the evaluation of the rest of the predicate. Employee.Workdeptno = 'D303' therfore yields a 'false' with the temporary workdeptno value leading the final result to be a false value. In the second attempt, we try with the inverse of the first assumption, i.e. the Workdeptno is now not equal to D404. The first predicate yields a 'false'. We have two choices while evaluating the second predicate; either to assume that the Workdeptno is D303 or that it is not. Both choices are consistent with our earlier assumption that the Workdeptno is not D404. The first choice yields a 'true' while the second choice yields a 'false'. In both cases, the final result is false. Actually, we could stop the evaluation after realizing a false result for the first predicate. We can apply the same method to the first (tautology) and the third example. The basic approach then is to assign truth values to the atomic predicate containing argument with a null value. In doing so, we constrain the possible set of values that the argument could hold for the further evaluation of the predicate. The tuple in consideration is included in the answer set if the entire predicate evaluates to true, with a logically consistent substitution of nulls with atleast one non-null value. We can also work backward by first assuming the value of the entire predicate to true.

Admittedly, the mechanism described above is more complex than the maybe logic. In fact, the problem of deriving any theorem in the first order logic is unsolvable. However, by confining to a subset of the first order logic and controlling the execution, we can probably manage the problem. Many current A.I. systems incorporate the required deduction machinery to develop the mechanism.

Before closing the discussion, we want to show an example where the system knowledge of null valued arguments could be used to further optimize the queries. Consider the following query, written in an SQL format.

```
Select Department.Deptname
From Deparment
Where (not) exists ( Select Employee.Empno
              From Employee
              Where Employee.Workdeptno=Department.Deptno
                   and Employee.Salary > 10000)
```

If we find a single tuple in the Employee relation with a null WorkDeptNo and Salary greater than 10000, we no longer need to evaluate the inner query for each Department.Deptno value, since the query will have atleast one answer tuple for any value of Department.Deptno. This condition could possibly be detected at the first evaluation of the inner query.

4. Bibiliography.

[Codd,86] Codd,E.F. Missing Information (Applicable and Inapplicable) in relational databases, SIGMOD Rec. Vol. 15, No. 4, December 86.

[Codd,87] Codd,E.F. More Commentary on Missing Information (Applicable and Inapplicable) in relational databases, SIGMOD Rec. Vol. 16, No. 1, March 87.

[Date,82] Date,C.J. Null values in Database Management, Proc. 2nd British National Conference on Databases (Invited Paper), July 1982.

[Gran,77] Grant,J. Null values in a Relational Database, Information Processing Letters 5(1977), 156-157.

[Koch,?] Kocharekar,R. Unifications in Relational Databases, Submitted to the Hawaii International Conference on Systems Sciences-22, to be held in January 89.

[Lips,79] Lipski Witold, Jr. On Semantic Issues Connected with Incomplete Information, ACM-TODS, September 79, Vol 4 No 3.