PASCAL

Speaking of NULLs: Hard to Do It Right

In the summary of his column on DB2 NULLs in *IDUG Solutions Journal* Craig Mullins writes:

"Nulls are clearly one of the most misunderstood features of DB2 – of most of SQL database systems for that matter. Although nulls can be confusing, you cannot bury your head in the sand and ignore nulls if you choose to use DB2 as your DBMS. Understanding what nulls are, and how best to use them, can help you create usable DB2 databases and design useful and correct queries in your DB2 applications."

We agree that you ignore the presence of SQL NULLs at your own peril, but, as we have demonstrated over the years in our writings, there is no "best way" to use them, and they actually create less usable databases, and misleading queries and answers, *if not outright wrong*. In fact, even though Mullins warns about the dangers of NULLs, his attempt at clarifying them often gets it wrong, because they are inherently unintuitive, and difficult to even talk about correctly. We will not reiterate here all the problems with nulls and NULLs here (we refer the reader to references listed at the end of this article), but rather point out how hard it is to even talk correctly about them.

It is a good idea not to confuse (a) a 'null' in many-valued logics (MVL)-the various theoretical attempts to expand two-valued logic (true/false) to more than two truth values (e.g. true/false/unknown, true/false/unknown/inapplicable) – with (b) a 'NULL', the SQL feature based on a poorly defined, problematic version of three-valued logic, in order to distinguish between problems inherent in the former, and problems in the SQL standard specification, and in commercial implementations. Mullins uses the former term even though he refers only to the latter, the DB2 implementation. We shall use the caps.

He begins as follows: "A NULL represents missing or unknown in formation at the column level. If a column 'value' can be NULL, it can mean one of two things: the attribute is not applicable for certain occurrences of the entity, or the attribute applies to all entity occurrences, but the information may not always be known. Of course, it could be a combination of these two."
Two *fundamental* problems are right here. First, 'missing or unknown' and 'inapplicable' are logically distinct and, therefore, must be treated differently in data manipulation, to obtain "correct" answers. Thus, just 'unknown', or just 'inapplicable' require different three-valued logics (3VL), while both require an yet different four-valued-logic (4VL); by "correct" we mean 'in the pertinent many-valued logic system', not necessarily in the real-world, where two-valued logic reigns. And the problem with SQL and its commercial implementations is that it supports a poorly defined 3VL for what is incorrectly perceived as a 4VL situation. As Mullins admits: "DB2 does not differentiate between NULLs that signify unknown data and those that signify inapplicable data. This distinction must be made by the program logic of each application."

To those who believe that this can be addressed reliably and cost-effectively in applications we say good luck. For all practical purposes, using NULLs for both unknown and "inapplicable values" in SQL will often produce wrong answers. We will get to the erroneous perception and the quotes around inapplicable shortly.

The second fundamental problem is that NULLs violate Codd's most basic Information Principle for relational databases (emphasis added):

All information in a truly relational database must be represented explicitly, in only one way: as *values* in relations.

That's because the relational model is based on predicate logic, which is 2VL. Thus, with a truly relational database and DBMS, queries yield answers that are guaranteed to be logically correct *in the real world*. It follows that tables that contain anything other than values are not relations and, therefore, databases containing them are not relational.
All bets are off: even if such databases yield answers that are correct within some many-valued logic system – and SQL does not guarantee even that – they are not guaranteed to be correct in the real world.

Mullins does warn: "NULLs sometimes are in appropriately referred to as 'NULL values'. Using the term value to describe a NULL is inaccurate because a NULL implies the lack of a value." But he fails to appreciate the implications, and advises

"simply use the term NULL or NULLs" (without attaching the term 'value' or 'values to it', as if that was enough to solve the problem.)
Note: In fact, he also fails to heed his own advice. If, as he says in the first quote above: "if a column value can be NULL", then a NULL and a value are interchangeable which, logically, they are not.

Mullins does warn: "Keep in mind, though, that using NULL to indicate 'not applicable' can be an indication of improper database design. By properly modeling and normalizing your database structures you can usually eliminate the need to use NULLs to indicate that a column is inapplicable for a specific row."

But "inapplicable value" is a contradiction in terms, an artifact of poor design by definition, hence the our quotes. Consequently, it *can and must* always be avoided. Furthermore, while we recommend fully normalized – that is, correctly designed – databases, normalization has nothing to do with "inapplicable values", and it does not eliminate them (see *The Costly Illusion: Normalization, Integrity, and Performance*, www.dbdebunk.com/page/page/1103793.htm).

Mullins asks and tries to justify NULLs as follows:"When are NULLs useful? Well, defining a column as NULL provides a placeholder for data you might not yet know. For example, when a new employee is hired and is inserted into an EMP table, what should the employee termination date column be set to? I don't know about you, but I wouldn't want any valid date to be set in that column for my employee record. Instead, NULL can be used to specify that the termination date is currently unknown."

But tuples in a relational database represent propositions assumed by convention to be true in the real world. That, and proper logical inferencing by the DBMS, guarantee results that are true in the real world. And whether you like it or not, what you don't know, you cannot assert to be true. NULL is essentially an attempt to circumvent this fact, by "asserting your ignorance", so to speak.
Recording in the database both propositions known to be true, and propositions whose truth is undecidable, defeats the ability of any DBMS to guarantee that answers to queries are logically correct in the real world, with quite insidious consequences, because SQL DBMSs will produce such answers, but you may not be aware of it, and even if you are, it is not easy to figure out what is wrong.

And because many-valued logics are unintuitive, and horribly complex, implementations are certain to be highly error-prone, piling problems of their own on top, which is exactly what happened to SQL.

Despite numerous attempts over the years to devise solutions to the thorny missing information problem, none was found within the 2VL/relational framework. But a correct solution must reside within that framework, which means that it permits only assertions known to be true. We have recently outlined such a solution in *The Final NULL in the Coffin* (http://www.dbdebunk.com/page/page/1396241.htm). So there is no longer an excuse for arguing that there is nothing but NULL to deal with unknown data.

For more detailed discussions of the problems with many-valued logics, SQL NULLs, and our relational solution to the missing information problem we refer the reader to articles on the subject at DATABASE DEBUNKINGS (www.dbdebunk.com), or the references available via the site's Books page (www.dbdebunk.com/books.htm), which have copious reference lists, particularly:

- F. Pascal, PRACTICAL ISSUES IN DATABASE MANAGEMENT (Addison Wesley, 2003);
- C.J. Date, AN INTRODUCTION TO DATABASE SYSTEMS, 8th Ed. (Addison Wesley, 2004);
- C.J. Date with H. Darwen, A GUIDE TO THE SQL STANDARD, 4th Ed. (Addison Wesley, 1996);
- McGoveran, D., Nothing from Nothing, Parts 1-4 in C. J. Date, RELATIONAL DATABASE WRITINGS 1994-1997 (Addison Wesley, 1998);
- F. Pascal, NULLs Nullified (www.dbazine.com/pascal27.shtml).

Fabian Pascal
Fabian Pascal is onafhankelijk IT-analist, consultant en auteur gespecialiseerd in data management.
Zie ook zijn website www.dbdebunk.com