The UNKNOWN Factor

Pedro Bizarro

Sometimes conditions might produce UNKNOWN instead of an expected Boolean value of TRUE or FALSE. If you have several Boolean values ANDed, ORed, and NOTed together, and if one value is UNKNOWN, it might not be obvious what the final combined value is. In this article, Pedro Bizarro presents an example of an UNKNOWN value that propagates its insidious effects, producing a surprising result set.

As an example for this article, I'm using the scott/tiger schema deployed with Oracle. The EMP table has, among others, two columns: EMPNO (the primary key) and MGR (a foreign key to EMPNO). With this relation, a hierarchy is established: Some employees manage others who manage others and so on. But life is harsh, and, of course, there are some employees who manage no one.

If I want to select just the employees who manage someone, I do the following:

```
/* query 01 */
SELECT empno, ename
  FROM emp
 WHERE empno IN (SELECT DISTINCT mgr FROM emp);

EMPNO      ENAME
---------- ----------
      7566 JONES
      7698 BLAKE
      7782 CLARK
      7788 SCOTT
      7839 KING
      7902 FORD
6 rows selected.
```

Because I know that EMP has 14 records, I now know that if six employees are managers, then eight aren't. Let's find their names:

```
/* query 02 */
SELECT empno, ename
  FROM emp
 WHERE NOT empno IN (SELECT DISTINCT mgr FROM emp);

EMPNO      ENAME
---------- ----------
0 rows selected.
```

What's wrong with this statement? Why don't I get the eight employees that don't manage? Before going any further, please try to mentally execute the query and discover why it doesn't give us the employees that don't manage.

The problem is threefold:

1. The MGR column allows NULL values.
2. "IN" is the same as "= ANY".
3. There's one UNKNOWN here.

Let's see, step by step, why query 02 doesn't return any rows.

King has a NULL

There's one employee, KING, who has no manager. His MGR value is NULL. Therefore, the complete list of MGR available includes a NULL. Let's execute just the subquery:

```
/* query 03 */
SELECT DISTINCT mgr FROM emp;

MGR
----------
      7566
      7698
      7782
      7788
      7839
      7902

7 rows selected.
```

Remember that the first query yielded six rows, and this one retrieves one extra row. The extra value, the last one, is a NULL. This can be shown more explicitly with the following:

```
/* query 04 */
SELECT DISTINCT nvl(to_char(mgr), 'NULL') MGR
  FROM emp;

MGR
----------------------------------------
7566
7698
7782
7788
7839
7902
NULL
7 rows selected.
```

IN is the same as "= ANY"

I can safely replace query 02:

```
/* query 02 */
SELECT empno, ename
  FROM emp
```

```
 WHERE NOT empno IN (SELECT DISTINCT mgr FROM emp);
```

with this:

```
/* query 05 */
SELECT empno, ename
  FROM emp
 WHERE NOT empno IN (7566, 7698, …, 7902, NULL);
```

I can expect the results to be the same, which they are; no rows are returned. I can further change the query knowing that "IN" is equivalent to "= ANY":

```
/* query 06 */
SELECT empno, ename
  FROM emp
 WHERE NOT empno = ANY (7566, 7698, …, 7902, NULL);
```

Because "= ANY" is the same as a sucession of ORs, the query can be rewritten as follows:

```
/* query 07 */
SELECT empno, ename
  FROM emp
 WHERE NOT (   empno = 7566
            OR empno = 7698
            OR empno = 7902
               ...
            OR empno = NULL);
```

Probably one could say that step 06 isn't necessary. Anyone can change from query 05 to query 07. But I wanted to make it very evident that the comparison implied and used by the operator IN is the equality ("=") no matter what we compare. The operator IN could be smarter and use IS NULL whenever a NULL was involved. But it's not, and that will lead to UNKNOWN.

The UNKNOWN factor

The condition "empno = NULL" returns UNKNOWN. Most of the time, UNKNOWN behaves as FALSE. For instance:

```
/* query 08 */
SELECT empno, ename
  FROM emp
 WHERE empno = NULL;
```

Query 08 returns no rows. UNKNOWN is like FALSE when the DBMS is deciding whether or not to retrieve rows. However, UNKNOWN doesn't behave like FALSE when Boolean operations like AND, OR, and NOT are involved. Figure 1 shows the truth tables involving UNKNOWNs. Figure 2 shows which conditions return UNKNOWNs and which don't.

Having the tables of Figure 1 and Figure 2, the explanation of why query 02 doesn't return rows is straightforward. All of the numeric conditions ORed in the subquery in query 07 will yield either TRUE or FALSE.

If the result is TRUE, then we have "TRUE OR UNKNOWN" (the UNKNOWN coming from "emp = NULL"), which gives TRUE. Furthermore, because everything is NOTed, the final result is FALSE. Thus, no rows are returned.

On the other hand, the numeric conditions ORed might yield a value of FALSE. In that case, "FALSE OR UNKNOWN" gives UNKNOWN, and NOT (UNKNOWN) still gives UNKNOWN, which explains why, in any case, query 07 never returns any row.

Continues on page 19

NOT

	TRUE	FALSE	UNKNOWN
NOT	FALSE	TRUE	UNKNOWN

AND

AND	TRUE	FALSE	UNKNOWN
TRUE	TRUE	FALSE	UNKNOWN
FALSE	FALSE	FALSE	FALSE
UNKNOWN	UNKNOWN	FALSE	UNKNOWN

OR

OR	TRUE	FALSE	UNKNOWN
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	UNKNOWN
UNKNOWN	TRUE	UNKNOWN	UNKNOWN

Figure 1. Truth tables involving UNKNOWNs.

Condition	Result
10 IS NULL	FALSE
10 IS NOT NULL	TRUE
NULL IS NULL	TRUE
NULL IS NOT NULL	FALSE
10 = NULL	UNKNOWN
10 != NULL	UNKNOWN
NULL = NULL	UNKNOWN
NULL != NULL	UNKNOWN

Figure 2. Conditions returning UNKNOWNs.

Enhanced Debugging...

Continued from page 10

procedure: We really shouldn't force developers to name their dump state program "dump_state." Add get and set programs to state_pkg to allow for some variation.

API-drive utilities are neat!

The elegance (as I see it) underlying the state_pkg implementation is that I "publish" an API ("name your dump procedure dump_state") or set of rules that you should follow in setting up your code. Then, once you follow those rules, I can use dynamic PL/SQL to work all sorts of magic.

Sure, it would be wonderful to have a full set of "reflection" features in PL/SQL, as you'll find in the Java reflection classes. But what's the point in whining about what we don't have? It's so much more fun and productive to work within the constraints of our reality and produce useful utilities regardless of those constraints. ▲

DOWNLOAD FEUER03.ZIP at www.oracleprofesionalnewsletter.com

Steven Feuerstein, an Oracle Professional Contributing Editor, is considered one of the world's leading experts and trainers on the Oracle PL/SQL language, having written the O'Reilly & Associates PL/SQL series, including *Oracle PL/SQL Programming*, *Oracle Built-in Packages*, and the *Oracle PL/SQL Developer's Workbook*. When he isn't writing books, he builds developer tools and knowledge bases for RevealNet, Inc. (http://www.revealnet.com). Steven sysops the very popular PL/SQL Pipeline, an online community for PL/SQL developers (http://www.revealnet.com/plsql-pipeline) and is president of PL/Solutions, a firm that provides PL/SQL consulting and training (http://www.plsolutions.com). Finally, he's vice president of the board of directors of the Crossroads Fund, which makes grants to Chicagoland organizations working for social, environmental, racial, and economic justice (http://www.CrossroadsFund.org). www.StevenFeuerstein.com, steven@stevenfeuerstein.com.

Explanation of the Source Code File's Contents

- state_pkg.pkg—The automated state-dumping package
- err.sql—The generic error-handling package
- overdue.pkg—The overdue fines rules package
- show_overdue.sql—The show_overdue procedure
- te_book.pkg—A portion of a table encapsulation package for the book table
- test.sql—A simple test script

The UNKNOWN Factor...

Continued from page 13

Conclusions

- NULLs require IS NOT or IS NOT NULL in comparisons.
- variable = NULL always returns UNKNOWN, even if the value of variable is NULL.
- NOT (UNKNOWN) is still UNKNOWN.
- UNKNOWN isn't the same as FALSE when Boolean expressions are involved.

As a final remark, there are several solutions to the original problem of finding the employees that don't manage. Here are two examples:

```
/* query 09 */
SELECT empno, ename
  FROM emp
 WHERE NOT empno IN (SELECT DISTINCT mgr
                       FROM emp
                      WHERE mgr IS NOT NULL);

Or

/* query 10 */
SELECT empno, ename
  FROM emp
 WHERE empno IN (SELECT empno FROM emp
                 MINUS
                 SELECT encar FROM emp);
```

Acknowledgements

I want to thank to Dr. Pedro Furtado (pnf@dei.uc.pt) for helping me solve this problem. ▲

Pedro Bizarro is a graduate student at University Nova de Lisboa and a teaching assistant in all database courses at University of Coimbra, Portugal. He will soon finish work for his master's degree and is already a Fulbright grantee for his Ph.D. program, beginning September 2001. pedro.b@acm.org.