December 2004

The Buffer Pool

Using Nulls in DB2

by Craig S. Mullins

A null represents missing or unknown information 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 situations, too.

A null is not the same as 0 or blank. Null means no entry has been made for the column and it implies that the value is either unknown or not applicable.

Because DB2 supports null you can distinguish between a deliberate entry of 0 (for numerical columns) or a blank (for character columns) and an unknown or inapplicable entry (NULL for both numerical and character columns). Null indicates that the user did not explicitly make an entry or has explicitly entered NULL for the column. For example, a null value in the Price column of the ITEM table in some database does not mean that the item is being given away for free; instead it means that the price is not known or has not yet been set.

Nulls sometimes are inappropriately referred to as "null values." Using the term value to describe a null is inaccurate because a null implies the lack of a value. Therefore, simply use the term null or nulls (without appending the term "value" or "values" to it).

A Few Examples

When are nulls useful? Well, defining a column as NULL provides a place holder for data you might not yet know. For example, when a new employee is hired and is inserted into the 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.

Let's consider another example. Suppose that we also capture employee's hair color when they are hired. Consider three potential entity occurrences: a man with black hair, a woman with unknown hair color, and a bald man. The woman with the unknown hair color and the bald man both could be assigned as null, but for different reasons. The woman's hair color would be null meaning presently unknown; the bald man's hair color could be null too, in this case meaning not applicable.

How could you handle this without using nulls? You would need to create special values for the HairColor column that mean "bald" and "unknown." This is possible for a CHAR column like HairColor. But what about a DB2 DATE column? All occurrences of a column assigned as a DATE data type are valid dates. It might not be possible to use a special date value to mean "unknown." This is where using nulls is most practical.

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.

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 data structures you can usually eliminate the need to use nulls to indicate that a column is inapplicable for a specific row. For example, consider the following table:

```
CREATE TABLE EMP
  (EMPNO         INTEGER     NOT NULL,
   LAST_NAME     CHAR(20)    NOT NULL,
   FIRST_NAME    CHAR(15)    NOT NULL,
   STREET_ADDR   CHAR(30)    NOT NULL WITH DEFAULT,
   CITY          CHAR(12)    NOT NULL WITH DEFAULT,
   STATE         CHAR(2)     NOT NULL WITH DEFAULT,
   POSTAL_CODE   CHAR(10)    NOT NULL WITH DEFAULT,
   EMP_TYPE      CHAR(1)     NOT NULL
     CHECK(EMP_TYPE IN 'F', 'C', 'P'),
   HIRE_DATE     DATE,
   SALARY        DECIMAL(9,2),
   BILLING_RATE DECIMAL(5,2));
```

In this case, we have a code in the EMP_TYPE column that can contain F (full-time), C (contractor), or P (part-time). We also have a SALARY column that is populated for full-time and part-time employees, but is set to null

for contractors; and a BILLING_RATE column that is populated for contractors but set to null for full-time and part-time employees. Additionally, the HIRE_DATE column is set to null for contractors.

Well, here we have three columns that are set to null (or not) based on other values in the table. We can design our way out of this problem by creating a separate table for employees and contractors. If additional columns were needed for full-time employees that did not apply part-time employees we might even split the employee table into two: one for full-time and another for part-time. After doing so, there is no more need to use null for inapplicable data.

Indicator Variables

DB2 represents null in a special "hidden" column known as an indicator variable. An indicator variable is defined to DB2 for each column that can accept nulls. The indicator variable is transparent to an end user, but must be provided for when programming in a host language (such as COBOL or PL/I).

The null indicator is used by DB2 to track whether its associated column is null or not. A positive value or a value of 0 means the column is not null and any actual value stored in the column is valid. If a CHAR column is truncated on retrieval because the host variable is not large enough, the indicator value will contain the original length of the truncated column. A negative value indicates that the column is set to null. If the value is -2 then the column was set to null as the result of a data conversion error.

Let's take a moment to clear up a common misunderstanding right here: nulls NEVER save storage space in DB2 for OS/390 and z/OS. Every nullable column requires one additional byte of storage for the null indicator. So, a CHAR(10) column that is nullable will require 11 bytes of storage per row – 10 for the data and 1 for the null indicator. This is the case regardless of whether the column is set to null or not.

DB2 for Linux, Unix, and Windows has a compression option that allows columns set to null to save space. Using this option causes DB2 to eliminate the unused space from a row where columns are set to null. This option is not available on the mainframe, though.

Syntax

Every column defined to a DB2 table must be designated as either allowing or disallowing nulls. A column is defined as nullable – meaning it can be set to NULL – in the table creation DDL. Null is the default if nothing is specified after the column name. To prohibit the column from being set to NULL you must explicitly specify NOT NULL after the column name. In the following sample table, COL1 and COL3 can be set to null, but not COL2, COL4, or COL5:

```
CREATE TABLE SAMPLE
     (COL1     INTEGER,
      COL2     CHAR(10)   NOT NULL,
      COL3     CHAR(5),
      COL4     DATE       NOT NULL WITH DEFAULT,
      COL5     TIME       NOT NULL
```

In SELECT statements, testing for null is accomplished differently than testing for other "values." You cannot specify WHERE COL = NULL, because this does not make any sense. Remember, null is a lack of a value, so the column does not equal anything. Instead, you would have to code WHERE COL IS [NOT] NULL.

In INSERT statements NULL can be specified in the VALUES clause to indicate that a column is to be set to NULL; but in UPDATE statements you can use the equality predicate (=) to assign a column to NULL.

When inserting data, if the user fails to make an entry in a column that allows nulls, DB2 supplies the NULL as a default (unless another default value exists). If an attempt to insert NULL is made against a column defined as NOT NULL, the statement will fail.

Guidance

Now that you have a good understanding of the basics of nulls, let's review some guidelines for their usage.

Whenever possible, avoid nulls in columns that must participate in arithmetic logic (for example, DECIMAL money values), and especially when functions will be used. The AVG, COUNT DISTINCT, SUM, MAX, and MIN functions omit column occurrences set to null. The COUNT(*) function, however, does not omit columns set to null because it operates on rows. Thus, AVG is not equal to SUM/COUNT(*) when the average is being computed for a column that can contain nulls. To clarify with an example, if the COMM column is nullable, the result of the following query:

```
SELECT   AVG(COMM)
FROM     DSN8810.EMP;
```

is not the same as for this query:

```
SELECT   SUM(COMM)/COUNT(*)
FROM     DSN8810.EMP;
```

So to avoid confusion, avoid nulls in columns involved in math functions whenever possible.

When DATE, TIME, and TIMESTAMP columns can be unknown, consider creating them as nullable. DB2 checks to ensure that only valid dates, times, and timestamps are placed in columns defined as such. If the column can be unknown, it must be defined to be nullable because the default for these columns is the current date, current time, and current timestamp (unless explicitly defined otherwise using the DEFAULT clause). Null, therefore, is the

only viable option for the recording of missing dates, times, and timestamps (unless you pick a specific valid date that is not used by your applications to indicate unknown).

For every other column, determine whether nullability can be of benefit before allowing nulls. Consider these rules of operation:

- When a nullable column participates in an ORDER BY or GROUP BY clause, the returned nulls are grouped at the high end of the sort order.
- Nulls are considered to be equal when duplicates are eliminated by SELECT DISTINCT or COUNT (DISTINCT *column*).
- A unique index considers nulls to be equivalent and disallows duplicate entries because of the existence of nulls, unless the WHERE NOT NULL clause is specified in the index.
- For comparison in a SELECT statement, two null columns are not considered equal. When a nullable column participates in a predicate in the WHERE or HAVING clause, the nulls that are encountered cause the comparison to evaluate to UNKNOWN.
- When a nullable column participates in a calculation, the result is null.
- Columns that participate in a primary key cannot be null.
- To test for the existence of nulls, use the special predicate IS NULL in the WHERE clause of the SELECT statement. You cannot simply state WHERE *column* = NULL. You must state WHERE *column* IS NULL.
- It is invalid to test if a column is < NULL, <= NULL, > NULL, or >= NULL. These are all meaningless because null is the absence of a value.

Examine these rules closely. ORDER BY, GROUP BY, DISTINCT, and unique indexes consider nulls to be equal and handle them accordingly. The SELECT statement, however, deems that the comparison of null columns is not equivalence, but unknown. This inconsistent handling of nulls is an anomaly that you must remember when using nulls. The following are several sample SQL queries and the effect nulls have on them.

```
SELECT   JOB, SUM(SALARY)
FROM     DSN8810.EMP
GROUP BY JOB;
```

This query returns the average salary for each type of job. All instances in which JOB is null will group at the bottom of the output.

```
SELECT  EMPNO, PROJNO, ACTNO, EMPTIME
        EMSTDATE, EMENDATE
FROM    DSN8810.EMPPROJACT
WHERE   EMSTDATE = EMENDATE;
```

This query retrieves all occurrences in which the project start date is equal to the project end date. This information is clearly erroneous, as anyone who has ever worked on a software development project can attest. The query does not return any rows in which either dates or both dates are null for two reasons: (1) two null columns are never equal for purposes of comparison, and (2) when either column of a comparison operator is null, the result is unknown.

```
UPDATE  DSN8810.DEPT
   SET   MGRNO = NULL
 WHERE   MGRNO = '000010';
```

This SQL sets the MGRNO column to null wherever MGRNO is currently equal to '000010' in the DEPT table.

When creating tables, treat nullable columns the same as you would any other column. Some DBAs advise you to place nullable columns of the same data type after non-nullable columns. This is supposed to assist in administering the null columns, but it does not really help – and it might hurt. Sequencing nullable columns in this manner provides no clear benefit and should be avoided.

Guidance

Nulls are clearly one of the most misunderstood features of DB2 – indeed, of most SQL database systems. 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 to create usable DB2 databases and design useful and correct queries in your DB2 applications.

From IDUG Solutions Journal, December 2004.

DB2PORTAL.com