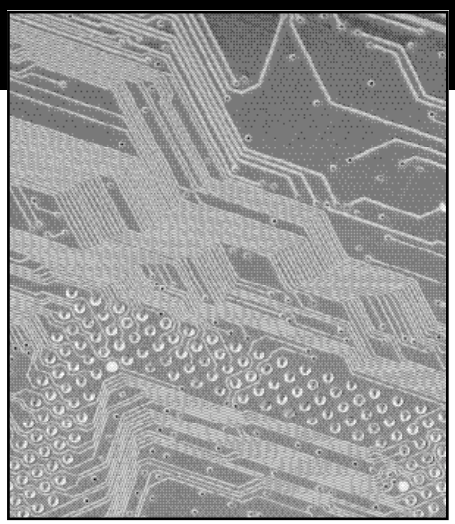

A Few SQL Tips and Techniques

BY CRAIG S. MULLINS

It is always a good idea to keep your bag of SQL tricks filled with techniques to help you deal with troubling application development problems. Hopefully you will find something useful in the ensuing sections as you build your DB2 applications.

SORTING DAYS OF THE WEEK

Here is a neat trick that you can use when you are dealing with days of the week. Assume that you have a table containing transactions, or some other type of interesting facts. The table has a CHAR(3) column containing the name of the day on which the transaction happened; let's call this column DAY_NAME. So, in this column the only valid values are as follows: SUN, MON, TUE, WED, THU, FRI, SAT.

Now, let's further assume that we need to query this table and have the results returned in order by the day of the week. Just like the calendar, though, we want Sunday first, followed by Monday, Tuesday, Wednesday, and so on. How can this be done? Well, if you write the first query that comes to mind, the results will obviously be sorted improperly:

```
SELECT   DAY_NAME, COL1, COL2 . . .
FROM     TXN_TABLE
ORDER BY DAY_NAME;
```

The results from this query would be ordered alphabetically; in other words

FRI
MON
SAT
SUN
THU
TUE
WED

One solution would be to design the table with an additional numeric or alphabetic column that would sort properly. By this I mean that we could add a DAY_NUM column that would be 1 for Sunday, 2 for Monday, and so on. But this requires a database design change. Furthermore, it requires additional coding and it is quite possible that the DAY_NUM and DAY_NAME data will get out of sync unless a lot of additional constraints are coded.

A better solution uses just SQL and requires no change to the database structures. All you need is an understanding of SQL and SQL functions – in this case, the LOCATE function. Here is the SQL:

```
SELECT   DAY_NAME, COL1, COL2 . .
FROM     TXN_TABLE
ORDER BY LOCATE(DAY_NAME, 'SUNMONTUEWEDTHUFRISAT');
```

To understand how this works we need to know how the LOCATE function works: it returns the starting position of the first occurrence of one string within another string. So, in our example, LOCATE finds the position of the DAY_NAME value within the string 'SUNMONTUEWEDTHUFRISAT', and returns the integer value of that position. If DAY_NAME is FRI, the function returns 16. Sunday would return 1, Monday 4, Tuesday 7, Wednesday 10, Thursday 13, Friday 16, and Saturday 19. This means that our results would be in the order we require.

(**Note:** Some other database systems have a function named INSTR, which is similar to LOCATE.)

Of course, you can go one step further if you'd like. Some queries may need to actually return the number for the day of week. That is, 1 for Sunday, 2 for Monday, etc. You can use the same technique with a twist to return the day of week value given only the day's name. To turn this into the appropriate day of the week number (that is, a value of 1 through 7), we divide by three, use the INT function on the result to return only the integer portion of the result, and then add one:

```
INT(LOCATE(DAY_NAME, 'SUNMONTUEWEDTHUFRISAT',)/3) + 1;
```

Let's use our previous example of Wednesday again. The LOCATE function returns the value 10. So, INT(10/3) = 3 and add 1 to get 4. And sure enough, Wednesday is the fourth day of the week.

Keep this technique in mind for when you need to wrestle with unruly days in your applications.

REMOVING SUPERFLUOUS SPACES

We all can relate to dealing with systems that have data integrity problems. But some data integrity problems can be cleaned up using a dash of SQL. Consider the common data entry problem of extraneous spaces inserted into a name field. Not only is it annoying, sometimes it can cause the system to ignore relationships between data elements because the names do not match. For example, "Craig Mullins" is not equivalent to "Craig Mullins"; the first one has three spaces between the first and last name whereas the second one only has one.

You can write an UPDATE statement to clean up these type problems, if you know how to use the REPLACE function. REPLACE does what it sounds like it would do: it reviews a source string and replaces all occurrences of a one string with another. For example, to replace all occurrences of Z with A in the string BZNZNZ you would code:

```
REPLACE('BZNZNZ','Z','A')
```

And the result would be BANANA. So, let's code some SQL using the REPLACE function to get rid of any unwanted spaces in the NAME column of our EMPLOYEE table. Keep in mind that we have no idea how many extra spaces there may be in the NAME columns. One may have two extra spaces, another fifteen extra, and another only one. So the SQL has to be flexible. Consider this:

```
UPDATE EMPLOYEE

    SET NAME = REPLACE(

                  REPLACE(

                   REPLACE(NAME, SPACE(1), '<>')

                   '><', SPACE(0))

                  '<>', SPACE(1));
```

Wait-a-minute, you might be saying. What are all of those left and right carats and why do I need them? Well, let's go from the inside out. The inside REPLACE statement takes the NAME column and converts every occurrence of a single space into a left/right carat. The next REPLACE (working outward), takes the string we just created, and removes every occurrence of a right/left carat combination by replacing it with a zero length string. The final REPLACE function takes that string and replaces any left/right carats with a single space. The reversal of the carats is the key to removing all spaces except one – remember, we want to retain a single space anywhere there was a single space as well as anywhere that had multiple spaces. Try it, it works.

Of course, if you don't like the carats you can use any two characters you like. But the left and right carat characters work well visually. Be sure that you do not choose to use characters that occur naturally in the string that you are acting upon. (I cannot think of anyone with a carat in their name, can you?)

Finally, the SPACE function was used for clarity. You could have used strings encased in single quotes, but the SPACE function is easier to read. It simply returns a string of spaces the length of which is specified as the integer argument. So, SPACE(11) would return a string of eleven spaces.

Keep this technique in your bag of tricks for when you need to clean up dirty DB2 data.

AGGREGATING AGGREGATES

Let's take a look at one additional SQL technique. This one allows us to perform aggregations of aggregates. For example, you might want to compute the average of a sum. This comes up frequently in applications that are built around sales amounts. Let's assume that we have a table containing sales information. Each sales amount has additional information indicating the salesman, region, district, product, date, etc.

A common requirement is to produce a report of the average sales by region for a particular period, say the first quarter of 2005. But the data in the table is at a detail level, meaning we have a row for each specific sale.

A novice SQL coder might try to code a function inside of a function – something like this: AVG(SUM(SALE_AMT)). Of course, this is invalid SQL syntax. DB2 does not allow aggregate functions to be nested. But we can use nested table expressions along with SQL functions to build the correct query.

Let's start by creating a query to return the sum of all sales by region for the time period in question, which is the third quarter of 2005. That query should look something like this:

```
SELECT REGION, SUM(SALE_AMT)

FROM   SALES

WHERE SALE_DATE BETWEEN DATE('2005-07-01')

                    AND     DATE('2005-09-30')

GROUP BY REGION;
```

Now that we have the total sales by region for the quarter, we can embed this query into a nested table expression in another query like so:

```
SELECT NTE.REGION, AVG(NTE.TOTAL_SALES)

FROM (SELECT REGION, SUM(SALE_AMT)

       FROM   SALES

       WHERE SALE_DATE BETWEEN DATE('2005-07-01')
                       AND     DATE('2005-09-30')
       GROUP BY REGION) AS NTE

GROUP BY NTE.REGION;
```

This works for any of the aggregate functions. The second query accesses data from the first query, and each uses an aggregate function.

Summary

In this issue we examined several methods of using SQL and functions to write queries without having to use host language code. With a sound understanding of SQL, and particularly SQL functions and expressions, you can frequently find novel ways of addressing problematic requirements using nothing but SQL.

ABOUT THE AUTHOR

Craig S. Mullins is president and principal consultant with Mullins Consulting, Inc. He is an IBM Gold Consultant, the author of two books, *DB2 Developer's Guide, 5th ed.* and *Database Administration: Practices and Procedures*, and can be reached via his Web site at www.CraigSMullins.com.

Index to Advertisers