Craig S. Mullins

Return to Home Page

Vol. 11, No. 1 (May 2004)

From IDUG Solutions Journal...

The Buffer Pool

Recursion in DB2 V8, V8, V8…

By Craig S. Mullins

One of the most intriguing new features of DB2 for z/OS V8 is the ability to code recursive SQL. A recursive query is one that refers to itself. I think the best way to quickly grasp the concept of recursion is to think about a mirror that is reflected into another mirror and when you look into it you get never-ending reflections of yourself. This is recursion in action.

Recursion is implemented in DB2 using common table expressions (CTEs), which also are new to DB2 V8. A CTE

can be thought of as a named temporary table within a SQL statement that is retained for the duration of that statement. There can be many CTEs in a single SQL statement; however, each must have a unique name. A CTE is defined at the beginning of a query using the WITH clause. After it is defined, the CTE can be referenced by name within the rest of the SQL statement.

Now before we dive into recursion, let's first look at some data that would benefit from being read recursively. Figure 1 shows a hierarchic organization chart.

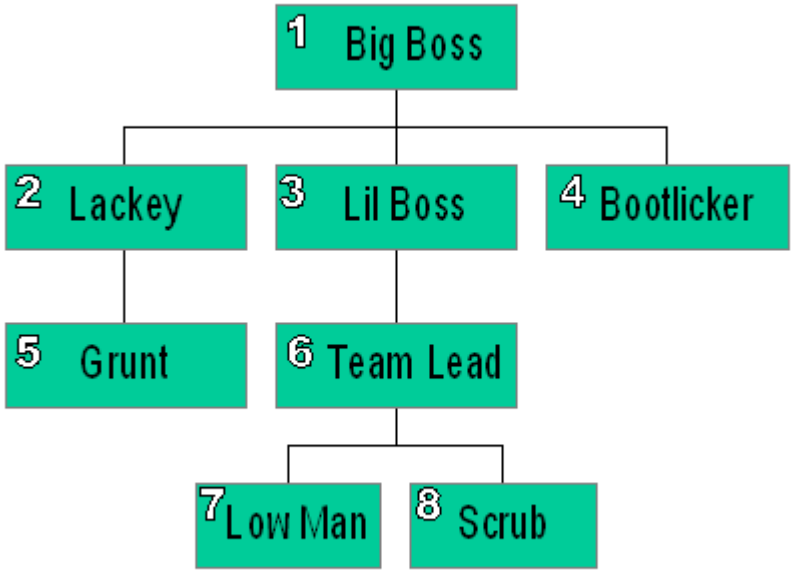

Figure 1. *A sample hierarchy*.

A DB2 table holding this data could be set up as follows:

```
CREATE TABLE ORG_CHART

 (MGR_ID       SMALLINT,
  EMP_ID       SMALLINT,
  EMP_NAME     CHAR(20))
;
```

Of course, this is a simple implementation and many more columns would likely be needed for a production hierarchy. But the simplicity of this table will suit our purposes for learning recursion. To make the data in this table match our diagram, we would load the table as follows:

MGR_ID	EMP_ID	EMP_NAME
-1	1	BIG BOSS
1	2	LACKEY
1	3	LIL BOSS
1	4	BOOTLICKER
2	5	GRUNT
3	6	TEAM LEAD
6	7	LOW MAN

The MGR_ID for the top-most node is set to some value indicating that there is no parent for this row; in this case –1 is used. Now that we have loaded the data we can code a query to walk the hierarchy using recursive SQL. Suppose we need to report on the entire organizational structure under LIL BOSS. The following recursive SQL using a CTE will do the trick (note that we have named our CTE “EXPL”):

```
WITH EXPL (MGR_ID, EMP_ID, EMP_NAME) AS
(
 SELECT ROOT.MGR_ID, ROOT.EMP_ID,
ROOT.EMP_NAME
 FROM   ORG_CHART   ROOT
 WHERE  ROOT.EMP_ID = 3

 UNION ALL

 SELECT CHILD.MGR_ID, CHILD.EMP_ID,
CHILD.EMP_NAME
 FROM   EXPL PARENT, ORG_CHART CHILD
 WHERE  PARENT.EMP_ID = CHILD.MGR_ID
)
```

```
SELECT   DISTINCT MGR_ID, EMP_ID, EMP_NAME
FROM     EXPL
ORDER BY MGR_ID, EMP_ID;
```

The results of running this query would be:

MGR_ID	EMP_ID	EMP_NAME
1	3	LIL BOSS
3	6	TEAM LEAD
6	7	LOW MAN
6	8	SCRUB

Let's break this somewhat complex query down into its constituent pieces to help understand what is going on. First of all, a recursive query is implemented using the WITH clause (using a CTE). The CTE is named EXPL. The first SELECT primes the pump to initialize the "root" of the search. In our case, to start with EMP_ID 3, that is LIL BOSS.

Now "here comes the tricky part." The next SELECT is an inner join combining the CTE with the table upon which

the CTE is based. This is where the recursion comes in. A portion of the CTE definition refers to itself. Finally, we SELECT from the CTE. Similar queries can be written to completely explode the hierarchy to retrieve all the descendants of any given node, if such data is so desired.

Recursive SQL can be very elegant and efficient. However, because of the difficulty developers can have understanding recursion, it is sometimes thought of as "too inefficient to use frequently." But, if you have a business need to walk or explode hierarchies in DB2, recursive SQL will likely be your most efficient option. What else are you going to do? You can create pre-exploded tables, but this requires denormalization and a lot of pre-processing which will not be very efficient either. Or you might write your own code to walk a hierarchy. This, too, is fraught with potential problems. You would probably retrieve more data than you need, causing inefficient I/O. And how would you assure that your code is more efficient than DB2?

If every row processed by the query is required in the answer set ("find all employees who work for LIL BOSS"), then recursion will most likely be quite efficient. If only a

few of the rows processed by the query are actually needed (“find all flights from Houston to Pittsburgh, but show only the three fastest”) then a recursive query can be quite costly. The bottom line is that you should consider recursive SQL when business requirements call for it. But be sure that suitable indexes are available and always examine your access paths.

From *IDUG Solutions Journal*, April 2004.

Home.