DBMS, March 1996

DBMS online

SQL for Smarties

Joe Celko

A Look at SQL Trees

Joe travels from New York to Boise, stopping only to consider trees (in SQL, that is).

In December, I attended both DB/Expo in New York and Database and Client/Server World in Chicago. This is no small feat, because these two major conferences are paired against each other in the same week and have been for the past couple of years. This really needs to stop. I did not see much of either conference and what I did see is blurred together.

DB/Expo

I arrived at La Guardia Airport on the Saturday before the conference in order to spend some time with a friend who lives in the West Village. By the way, I won the *DBMS* Readers' Choice award again this year (thank you, gentle readers, thank you). However, I managed to miss the awards breakfast at DB/Expo because my friend had to make an unplanned service station stop for air and gasoline on the way to the Javits Convention Center that morning.

Database World

I flew out of New York the night of December 5, 1995 and arrived at O'Hare Airport in time to teach a class at 8:30 the next morning. I have been doing two SQL overview classes at Database World for the past few years and you would think that anyone who wants to hear it has already done so by now. But every year, I play to a packed house.

The Client/Server Connection Ltd. booths at both DB/Expo and Database World were being called the "Dilbert booths," but not because the company used the popular comic strip in its advertising. Actually, its booths at both shows featured lots of models in

heels and black evening dresses doing demonstrations. Anyone remember the girls in Robert Palmer's 1980s rock video, "Addicted to Love"? These ladies were clearly not technical --I watched them get briefed on the demo the first morning (purely research for my column, you understand). The techy women working other booths were really put off and used terms such as "booth bunnies" to describe them.

By the way, the reason the nickname "Dilbert booth" was coined had to do with the Dilbert comic strip of December 4, the first day of the shows. It featured Dilbert (I'm assuming you all know who he is), another engineer, and a saleswoman sitting at a table, with the saleswoman saying, "On one hand, my company does use inferior technology in our products . . . but on the other hand, I'm the most attractive female who has paid attention to you this year." Dilbert's response is, "What kind of engineers do you think we are??!" But his partner's response is, "Do you have pictures of your support people?"

Boise

The following week, I was supposed to attend an ANSI X3H2 Database Standards Committee meeting in Little Rock, Ark. But instead I was in Boise, Idaho to teach a four-day SQL class at the headquarters of Albertson's Food Stores. (The SQL Explorer does not always visit the exotic locales of North America, despite what you might think.)

However, Boise was more fun than I thought it would be, even though I did not get out to see much of the town. Teaching a class of programmers with mixed experience levels, positions, and backgrounds was interesting, and it seemed to work fairly well after we settled into a schedule. I have taught mixed classes before and it has usually failed, so I was impressed.

One reason for this problem is that companies will send the bosses along with their immediate subordinates. The bosses will not ask questions because they do not want to look like fools in front of their people, and their people will not ask questions because they do not want to look like fools in front their bosses.

The Business in Little Rock

Two things happened at the X3H2 meeting in Little Rock that are of interest. First, Jean Shildneck at the ANSI SC21 Secretariat sent us a message that the XA Editing Meeting scheduled for December 18-19 in London was canceled. DIS 14834 (Draft Information Standard), the XA specification for distributed transaction processing, is a proposal that X/Open brought to ISO, and is favored by most companies in this arena. The response to the DIS ballot was 100 percent in favor, so Shildneck delegated authority for the revision of the document to the Project Editor in conjunction with the SC 21 Rapporteurs for TP (Andrew Bainbridge) and DBL (Len Gallagher of NIST and ANSI X3H2). What this means in English is that the X/Open transaction model is getting closer to becoming an international standard.

The second interesting thing concerns the existing Remote Database Access standard (RDA/SQL Specialization, ISO/IEC 9579-2:1993), which currently provides RDA facilities to support the entry-level SQL-92 standard. There is now a proposed draft amendment to RDA/SQL that would require it to support the intermediate and full levels of SQL-92. Together with the Generic RDA standard (ISO/IEC 9579-1:1993), the amended RDA/SQL specialization will provide communications protocols that enable a conforming RDA client to communicate with a conforming RDA server. This would provide complete access to all features in a database managed by a processor that conforms to any level of the SQL-92 standard. This is a protocol standard only -- programming language bindings and call-level interfaces are provided by other SQL standards (such as ISO/IEC 9075-3:1995 and SQL/CLI).

New features in this specification include support for additional SQL-92 data types (date, time, timestamp, and bit string), as well as new protocol elements for the exchange of SQL diagnostic and SQL descriptor information, and enhancements for connection and transaction management. These new facilities provide a SQL application on the client side with more efficient support for dynamic SQL queries and runtime management of SQL warnings and exceptions. What this means in English is that you are going to need the full SQL-92 language to function in the real world.

We also lost another X3H2 member to an acquisition, but this time it was not Computer Associates or Sybase doing the swallowing. On December 20, Informix Software Inc. announced that it will acquire Illustra Information Technologies Inc.

(Oakland, Calif.).

Illustra was founded by Michael Stonebreaker (Ingres inventor and founder). Illustra produces and markets what it calls "dynamic content management database software," but what most of us would call an object-oriented database. The acquisition of Illustr a will be accounted for as a tax-free pooling of interest. Informix wants to integrate Illustra's dynamic content management system into its core parallel database technology, Dynamic Scalable Architecture (DSA), to enable it to leapfrog ahead of the mar ket and be ready for the next wave of database applications. Informix sees the world, and the Web in particular, as shifting from static and fixed data types to dynamic and rich content data, which requires users to interact with 3D graphics, video, audi o, HTML, spatial data, and other complex data.

I think that the Informix move makes more sense than other acquisitions; too many recent acquisitions in the software industry were attempts to patch up past shortcomings with some other vendor's technology. This looks like an aggressive move forward -- it attempts to predict the market instead of react to it.

Informix and Computer Systems Advisers (CSA) Inc. also stuck a deal (announced at DB/Expo) to integrate CSA's Silverrun CASE toolset with Informix's NewEra client/server development environment. Silverrun is made up of four integrated modules: Entity-Rel ationship Expert (ERX), Relational Data Modeler (RDM), Business Process Modeler (BPM), and Workgroup Repository Manager (WRM).

I will make a prediction that there is about to be a revival in CASE, because when you get into a data warehouse project, the first thing you need is some sort of repository for your metadata. Silverrun is worth a look because it runs on all the major op erating systems, is nicely integrated, and has a feature CSA calls SuperViews, which are views with rules that are generated from the underlying data model. While not really a fully object-oriented system (which would not be supported in SQL anyway), Sup erViews lets you work at a higher level of abstraction. For example, you can build something called an "invoice" rather than constantly referencing a set of tables for the invoice header, invoice details, shipping calculation, tax calculation, and the ho st of other database objects that are joined together to define an invoice and its behavior.

Trees in SQL

The Committee also did something else at Little Rock to improve the way that SQL handles graph structures. It approved the DB2-style WITH <subquery> recursion clause for SQL3, and dropped the previous RECURSIVE UNION operator. Trust me, getting rid of RECURSIVE UNION was a good idea, but I want to look at the details of the new statement. You can expect a column on this topic later in 1996. In the meantime, working SQL programmers keep asking me how to handle trees (the most common recursive data structure) in SQL, and I have a solution that I feel is better than what the vendors provide.

A tree is a special kind of directed graph. Graphs are data structures that are made up of nodes or vertices (usually shown as boxes) connected by arcs or edges (usually shown as lines with arrowheads). Each edge represents a one-way relationship between the two nodes it connects. In an organizational chart, the nodes are personnel, and each edge is the "reports to" relationship. In a parts explosion (also called a bill of materials), the nodes are assembly units (eventually resolving down to individual parts), and each edge is the "is made of" relationship.

The top of the tree is called the root. In an organizational chart, it is the highest authority; in a parts explosion, it is the final assembly. The number of edges coming out of the node are its outdegree, and the number of edges entering it are its ind egree. A binary tree is one in which a parent can have no more than two children; more generally, an n-ary tree is one in which a node can have no more than outdegree n, or any number of child nodes.

The nodes of the tree that have no subtrees beneath them are called the leaf nodes. In a parts explosion, they are the individual parts, which cannot be broken down any further. The descendants, or children, of a parent node are every node at all lower l evels in the subtree that has the parent node as its root.

Trees are often drawn as charts. (See Figure 1.) Americans like to put the root at the top and grow the tree downward; Europeans

will often put the root at the bottom and grow the tree upward, or grow the tree from left to right ac ross the page. Another way of representing trees is to show them as nested sets (see Figure 2); this is the basis for the nested set representation in SQL that I use.

In SQL, any relationship must be shown explicitly as data. The typical way to represent trees is to put an adjacency matrix in a table. That is, one column is the parent node, and another column in the same row is the child node (the pair represents an e dge in the graph). For example, consider the organizational chart of this six-person company:

```
CREATE TABLE Personnel
(emp CHAR(20) PRIMARY KEY,
boss CHAR(20) REFERENCES Personnel(emp),
salary DECIMAL(6,2) NOT NULL);
```

Personnel

emp	boss	salary
'Jerry'	NULL	1000.00
'Bert'	'Jerry'	900.00
'Chuck'	'Jerry'	900.00
'Donna'	'Chuck'	800.00
'Eddie'	'Chuck'	700.00
'Fred'	'Chuck'	600.00

This model has advantages and disadvantages. The PRIMARY KEY is clearly emp, but the boss column is functionally dependent upon it, so you have normalization problems. The REFERENCES constraint will keep you from giving someone a boss who is not also an employee. However, what happens when 'Jerry' changes his name to 'Geraldo' to get a television talk show? You have to cascade the change to the 'Bert' and 'Chuck' rows as well.

Another disadvantage of the adjacency model is that path enumeration is difficult. To find the name of the boss for each employee, the query is a self-join, such as:

```
SELECT B1.emp, 'bosses', E1.emp
        FROM Personnel AS B1, Personnel AS E1
WHERE B1.emp = E1.boss;
```

But something is missing here. This query gives you only the immediate bosses of the personnel. Your boss's boss also has authority over you, and so on up the tree until you find someone who has no subordinates. To go two levels deep in the tree, you nee d to write a more complex self-join, such as:

```
SELECT B1.emp, 'bosses', E2.emp
        FROM Personnel AS B1, Personnel AS E1,
        Personnel AS E2
WHERE B1.emp = E1.boss
        AND E1.emp = E2.boss;
```

To go more than two levels deep in the tree, simply extend the pattern:

```
SELECT B1.emp, 'bosses', E3.emp
        FROM Personnel AS B1, Personnel AS E1,
```

```
        Personnel AS E2, Personnel AS E3
WHERE B1.emp = E1.boss
        AND E1.emp = E2.boss
        AND E2.emp = E3.boss;
```

Unfortunately, you have no idea just how deep the tree is, so you must keep extending this query until you get an empty set back as a result.

A leaf node has no children under it. In an adjacency model, this set of nodes is fairly easy to find: It is the personnel who are not bosses to anyone else in the company:

```
SELECT *
    FROM Personnel AS E1
WHERE NOT EXISTS (SELECT *
            FROM Personnel AS E2
        WHERE E1.emp = E2.boss);
The root of the tree has a boss that is null:

SELECT *
    FROM Personnel
WHERE boss IS NULL;
```

The real problems are in trying to compute values up and down the tree. As an exercise, write a query to sum the salaries of each employee and his/her subordinates; the result is:

Total Salaries

emp	boss	salary
'Jerry'	NULL	4900.00
'Bert'	'Jerry'	900.00
'Chuck'	'Jerry'	3000.00
'Donna'	'Chuck'	800.00
'Eddie'	'Chuck'	700.00
'Fred'	'Chuck'	600.00

Nested-Set Model of Trees

Another way of representing trees is to show them as nested sets. Because SQL is a set-oriented language, this is a better model. The root of the tree is a set that contains all the other sets, and the child relationship is shown as set containment.

There are several ways to think about transforming the organizational chart into nested sets. One way is to imagine that you are pulling the subordinate ovals inside their parents using the edge lines as ropes. The root is the largest oval and contains e very other node. The leaf nodes are the innermost ovals, with nothing else inside them, and the nesting shows the hierarchical relationship. This is a natural way to model a parts explosion, because a final assembly is made of physically nested assemblie s that finally break down into separate parts.

Another approach is to visualize a little worm crawling along the "boxes and arrows" of the tree. The worm starts at the top, the root, and makes a complete trip around the tree. Computer science majors will recognize this as a modified preorder tree-tra versal algorithm.

But now let's get a smarter worm with a counter that starts at one. When the worm comes to a node box, it puts a number in the cell on the side that it is visiting and increments its counter. Each node will get two numbers, one for the right side and one for the left side.

This has some predictable results that you can use for building queries. The Personnel table will look like the following, with the left and right numbers in place:

```
CREATE TABLE Personnel
(emp CHAR(10) PRIMARY KEY,
salary DECIMAL(6,2) NOT NULL,
left INTEGER NOT NULL,
right INTEGER NOT NULL);
```

Personnel

emp	salary	left	right
'Jerry'	1000.00	1	12
'Bert'	900.00	2	3
'Chuck'	900.00	4	11
'Donna'	800.00	5	6
'Eddie'	700.00	7	8
'Fred'	600.00	9	10

The root will always have a 1 in its left-visit column and twice the number of nodes (2*n) in its right-visit column. This is easy to understand: The worm has to visit each node twice, once for the left side and once for the right side, so the final coun t has to be twice the number of nodes in the entire tree.

In the nested-set model, the difference between the left and right values of leaf nodes is always 1. Think of the little worm turning the corner as it crawls along the tree. Therefore, you can find all leaf nodes with the following simple query:

```
SELECT *
        FROM Personnel
WHERE (right - left) = 1;
```

You can use another trick to speed up queries. Build a unique index on the left column, then rewrite the query to take advantage of the index:

```
SELECT *
        FROM Personnel
        WHERE left = (right - 1);
```

The reason this improves performance is that the SQL engine can use the index on the left column when it does not appear in an expression. Don't use (right - left) = 1, because it will not take advantage of the index.

In the nested-set model, paths are shown as nested sets, which are represented by the nested sets' numbers and BETWEEN predicates. For example, to find out all of the bosses to whom a particular person reports in the company hierarchy, you would write:

```
SELECT :myworker, B1.emp, (right - left) AS height
FROM Personnel AS B1, Personnel AS E1
WHERE E1.left BETWEEN B1.left AND B1.right
AND E1.right BETWEEN B1.left AND B1.right
AND E1.emp = :myworker;
```

The greater the height, the farther up the corporate hierarchy that boss is from the employee. The nested-set model uses the fact that each containing set is larger in size (where size = (right - left)) than the sets it contains. Obviously, the root will always have the largest size.

The level function is the number of edges between two given nodes, and it is fairly easy to calculate. For example, to find the levels of bureaucracy between a particular worker and manager, you would use:

```
SELECT E1.emp, B1.emp, COUNT(*) - 1 AS levels
        FROM Personnel AS B1, Personnel AS E1
WHERE E1.left BETWEEN B1.left AND B1.right
        AND E1.right BETWEEN B1.left AND B1.right
        AND E1.node = :myworker
        AND B1.node = :mymanager;
```

The reason for using the expression (COUNT(*) - 1) is to remove the duplicate count of the node itself as being on another level, because a node is zero levels removed from itself.

You can build other queries from this basic template using views to hold the path sets. For example, to find the common bosses of two employees, union their path views and find the nodes that have (COUNT(*)>1). To find the nearest common ancestors of two nodes, UNION the path views, find the nodes that have (COUNT(*)>1), and pick the one with the smallest depth number.

I will get into more programming tricks with this model next month, but for now, try to write the sum of all subordinate salaries with this table and compare it to what you did for the edge model version of this hierarchy.

Puzzle

I am not going to give you a problem with trees this month -- I will wait until I get further into the topic and the problems are harder. Instead, suppose you have a table with the addresses of consumers to whom you wish to send junk mail. The table has a fam (family) column that links Consumers with the same stree t address. You need this because one of your rules is that you can mail only one offer to a household. The field contains the primary-key value of the Consumers record of the first person who has this address, thus:

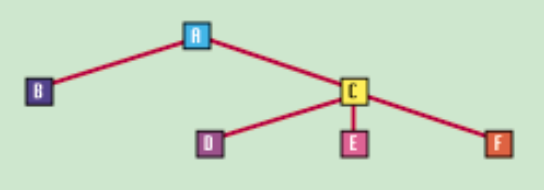
Consumers

name	address	id	fam
Bob	A	1	NULL
Joe	B	3	NULL
Mark	C	5	NULL
Mary	A	2	1
Vickie	B	4	3
Wayne	D	6	NULL

You need to delete those rows in which fam is null, but there are other family members on the mailing list. In the above example,

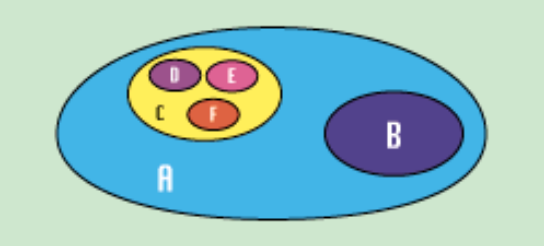
you need to delete Bob and Joe, but not Mark or Wayne.

FIGURE 1



--The top of the tree is called the root. The nodes of the tree that have no subtrees beneath them are called the leaf nodes. The descendants, or children, of a parent node are every node at all lower levels in the subtree that has the parent node as its root.

FIGURE 2



--Another way of representing trees is to show them as nested sets. Because SQL is a set-oriented language, this is a better model. The root of the tree is a set that contains all the other sets, and the child relationship is shown as set containment.

Puzzle Answer

During your first attempt, you might try to do too much work. For instance, translating the English specification directly into SQL gives you the following:

```
DELETE FROM Consumers
 WHERE fam IS NULL   - this guy has a NULL family value
   AND EXISTS   -  . .and there is someone who is
       (SELECT *
          FROM Consumers AS C1
         WHERE C1.id  id       - a different person
           AND C1.address = address  - at the same address
           AND C1.fam IS NOT NULL); - who has a family value
```

But if you think about it, you will see that the count(*) for the household must be greater than one.

```
DELETE FROM Consumers
 WHERE fam IS NULL   - this guy has a NULL family value
  AND (SELECT COUNT(*)
         FROM Consumers AS C1
        WHERE C1.address = address) > 1;
```

The trick is that the count(*) aggregate will include NULLs in its tally.--*Joe Celko*

Joe Celko is an Atlanta-based Guru with Northern Lights Software Ltd. and a member of the ANSI X3H2 Database Standards Committee. He is also the author of two books on SQL: SQL For Smarties (Morgan-Kaufmann) and Instant SQL (Wrox Press). You can contact Joe via CompuServe at 71062,1056 or via email at 71062.1056@compuserve.com.

---

* The Client/Server Connection Ltd., 103 South Bedford Rd., Ste. 202; Mt. Kisko, NY 10549; 914-241-9100 or fax 914-241-7878; http://www.cscl.com/.
* Computer Systems Advisers Inc., 300 Tice Blvd., Woodcliff Lake, NJ 07675; 201-391-6500 or fax 201-391-2210; http://www.silverrun.com.
* Informix Software Inc., 4100 Bohannon Dr., Menlo Park, CA 94025; 415-926-6300; http://www.informix.com.

March 1996 Table of Contents | Other Contents | Article Index | Search | Site Index | Home

---

DBMS and *Internet Systems* (http://www.dbmsmag.com)

Please send questions or comments to dbms@mfi.com
Updated Wednesday, November 6, 1996