

Re: Employees' Hierarchy

Source:

<http://www.tech-archive.net/Archive/SQL-Server/microsoft.public.sqlserver.programming/2004-11/4325.html>

From: Leila (Leilas_at_hotpop.com)

Date: 11/19/04

Date: Fri, 19 Nov 2004 09:06:00 +0330

Thanks for the great description Joe!

"Joe Celko" <jcelko212@earthlink.net> wrote in message
news:#KISJAczEHA.2572@tk2msftngp13.phx.gbl...

> *There are many ways to represent a tree or hierarchy in SQL. This is
> called an adjacency list model and it looks like this:*

>
> *CREATE TABLE OrgChart*
> *(emp CHAR(10) NOT NULL PRIMARY KEY,*
> *boss CHAR(10) DEFAULT NULL REFERENCES OrgChart(emp),*
> *salary DECIMAL(6,2) NOT NULL DEFAULT 100.00);*

>
> *OrgChart*
> *emp boss salary*
> =====

> *'Albert' NULL 1000.00*
> *'Bert' 'Albert' 900.00*
> *'Chuck' 'Albert' 900.00*
> *'Donna' 'Chuck' 800.00*
> *'Eddie' 'Chuck' 700.00*
> *'Fred' 'Chuck' 600.00*

>
> *Another way of representing trees is to show them as nested sets.*
>
> *Since SQL is a set oriented language, this is a better model than the
> usual adjacency list approach you see in most text books. Let us define
> a simple OrgChart table like this.*

>
> *CREATE TABLE OrgChart*
> *(emp CHAR(10) NOT NULL PRIMARY KEY,*
> *lft INTEGER NOT NULL UNIQUE CHECK (lft > 0),*
> *rgt INTEGER NOT NULL UNIQUE CHECK (rgt > 1),*
> *CONSTRAINT order_okay CHECK (lft < rgt));*

>
> *OrgChart*
> *emp lft rgt*
> =====

- > 'Albert' 1 12
- > 'Bert' 2 3
- > 'Chuck' 4 11
- > 'Donna' 5 6
- > 'Eddie' 7 8
- > 'Fred' 9 10
- >
- > *The organizational chart would look like this as a directed graph:*
- >
- > *Albert (1, 12)*
- > /\
- > /\
- > *Bert (2, 3) Chuck (4, 11)*
- > /\
- > /\
- > /\
- > /\
- > *Donna (5, 6) Eddie (7, 8) Fred (9, 10)*
- >
- > *The adjacency list table is denormalized in several ways. We are*
- > *modeling both the Personnel and the organizational chart in one table.*
- > *But for the sake of saving space, pretend that the names are job titles*
- > *and that we have another table which describes the Personnel that hold*
- > *those positions.*
- >
- > *Another problem with the adjacency list model is that the boss and*
- > *employee columns are the same kind of thing (i.e. names of personnel),*
- > *and therefore should be shown in only one column in a normalized table.*
- > *To prove that this is not normalized, assume that "Chuck" changes his*
- > *name to "Charles"; you have to change his name in both columns and*
- > *several places. The defining characteristic of a normalized table is*
- > *that you have one fact, one place, one time.*
- >
- > *The final problem is that the adjacency list model does not model*
- > *subordination. Authority flows downhill in a hierarchy, but If I fire*
- > *Chuck, I disconnect all of his subordinates from Albert. There are*
- > *situations (i.e. water pipes) where this is true, but that is not the*
- > *expected situation in this case.*
- >
- > *To show a tree as nested sets, replace the nodes with ovals, and then*
- > *nest subordinate ovals inside each other. The root will be the largest*
- > *oval and will contain every other node. The leaf nodes will be the*
- > *innermost ovals with nothing else inside them and the nesting will show*
- > *the hierarchical relationship. The (lft, rgt) columns (I cannot use the*
- > *reserved words LEFT and RIGHT in SQL) are what show the nesting. This is*
- > *like XML, HTML or parentheses.*
- >
- > *At this point, the boss column is both redundant and denormalized, so it*
- > *can be dropped. Also, note that the tree structure can be kept in one*
- > *table and all the information about a node can be put in a second table*
- > *and they can be joined on employee number for queries.*

>
> *To convert the graph into a nested sets model think of a little worm
> crawling along the tree. The worm starts at the top, the root, makes a
> complete trip around the tree. When he comes to a node, he puts a number
> in the cell on the side that he is visiting and increments his counter.
> Each node will get two numbers, one of the right side and one for the
> left. Computer Science majors will recognize this as a modified preorder
> tree traversal algorithm. Finally, drop the unneeded OrgChart.boss
> column which used to represent the edges of a graph.*
>
> *This has some predictable results that we can use for building queries.
> The root is always (left = 1, right = 2 * (SELECT COUNT(*) FROM
> TreeTable)); leaf nodes always have (left + 1 = right); subtrees are
> defined by the BETWEEN predicate; etc. Here are two common queries which
> can be used to build others:*
>
> *1. An employee and all their Supervisors, no matter how deep the tree.*
>
> *SELECT O2.*
> FROM OrgChart AS O1, OrgChart AS O2
> WHERE O1.lft BETWEEN O2.lft AND O2.rgt
> AND O1.emp = :myemployee;*
>
> *2. The employee and all their subordinates. There is a nice symmetry
> here.*
>
> *SELECT O1.*
> FROM OrgChart AS O1, OrgChart AS O2
> WHERE O1.lft BETWEEN O2.lft AND O2.rgt
> AND O2.emp = :myemployee;*
>
> *3. Add a GROUP BY and aggregate functions to these basic queries and you
> have hierarchical reports. For example, the total salaries which each
> employee controls:*
>
> *SELECT O2.emp, SUM(S1.salary)
> FROM OrgChart AS O1, OrgChart AS O2,
> Salaries AS S1
> WHERE O1.lft BETWEEN O2.lft AND O2.rgt
> AND O1.emp = S1.emp
> GROUP BY O2.emp;*
>
> *4. To find the level of each emp, so you can print the tree as an
> indented listing. Technically, you should declare a cursor to go with
> the ORDER BY clause.*
>
> *SELECT COUNT(O2.emp) AS indentation, O1.emp
> FROM OrgChart AS O1, OrgChart AS O2
> WHERE O1.lft BETWEEN O2.lft AND O2.rgt
> GROUP BY O1.lft, O1.emp
> ORDER BY O1.lft;*

```
>
> 5. The nested set model has an implied ordering of siblings which the
> adjacency list model does not. To insert a new node, G1, under part G.
> We can insert one node at a time like this:
>
> BEGIN ATOMIC
> DECLARE rightmost_spread INTEGER;
>
> SET rightmost_spread
> = (SELECT rgt
> FROM Frammis
> WHERE part = 'G');
> UPDATE Frammis
> SET lft = CASE WHEN lft > rightmost_spread
> THEN lft + 2
> ELSE lft END,
> rgt = CASE WHEN rgt >= rightmost_spread
> THEN rgt + 2
> ELSE rgt END
> WHERE rgt >= rightmost_spread;
>
> INSERT INTO Frammis (part, lft, rgt)
> VALUES ('G1', rightmost_spread, (rightmost_spread + 1));
> COMMIT WORK;
> END;
>
> The idea is to spread the (lft, rgt) numbers after the youngest child of
> the parent, G in this case, over by two to make room for the new
> addition, G1. This procedure will add the new node to the rightmost
> child position, which helps to preserve the idea of an age order among
> the siblings.
>
> 6. To convert a nested sets model into an adjacency list model:
>
> SELECT B.emp AS boss, E.emp
> FROM OrgChart AS E
> LEFT OUTER JOIN
> OrgChart AS B
> ON B.lft
> = (SELECT MAX(lft)
> FROM OrgChart AS S
> WHERE E.lft > S.lft
> AND E.lft < S.rgt);
>
> 7. To convert an adjacency list to a nested set model, use a push down
> stack. Here is version with a stack in SQL/PSM.
>
> -- Tree holds the adjacency model
> CREATE TABLE Tree
> (node CHAR(10) NOT NULL,
> parent CHAR(10));
```

```
>
> -- Stack starts empty, will holds the nested set model
> CREATE TABLE Stack
> (stack_top INTEGER NOT NULL,
> node CHAR(10) NOT NULL,
> lft INTEGER,
> rgt INTEGER);
>
> CREATE PROCEDURE TreeTraversal ()
> LANGUAGE SQL
> DETERMINISTIC
> BEGIN ATOMIC
> DECLARE counter INTEGER;
> DECLARE max_counter INTEGER;
> DECLARE current_top INTEGER;
>
> SET counter = 2;
> SET max_counter = 2 * (SELECT COUNT(*) FROM Tree);
> SET current_top = 1;
>
> --clear the stack
> DELETE FROM Stack;
>
> -- push the root
> INSERT INTO Stack
> SELECT 1, node, 1, max_counter
> FROM Tree
> WHERE parent IS NULL;
>
> -- delete rows from tree as they are used
> DELETE FROM Tree WHERE parent IS NULL;
>
> WHILE counter <= max_counter - 1
> DO IF EXISTS (SELECT *
> FROM Stack AS S1, Tree AS T1
> WHERE S1.node = T1.parent
> AND S1.stack_top = current_top)
> THEN BEGIN -- push when top has subordinates and set lft value
> INSERT INTO Stack
> SELECT (current_top + 1), MIN(T1.node), counter, NULL
> FROM Stack AS S1, Tree AS T1
> WHERE S1.node = T1.parent
> AND S1.stack_top = current_top;
>
> -- delete rows from tree as they are used
> DELETE FROM Tree
> WHERE node = (SELECT node
> FROM Stack
> WHERE stack_top = current_top + 1);
> -- housekeeping of stack pointers and counter
> SET counter = counter + 1;
```

```
> SET current_top = current_top + 1;
> END;
> ELSE
> BEGIN -- pop the stack and set rgt value
> UPDATE Stack
> SET rgt = counter,
> stack_top = -stack_top -- pops the stack
> WHERE stack_top = current_top;
> SET counter = counter + 1;
> SET current_top = current_top - 1;
> END;
> END IF;
> END WHILE;
> -- SELECT node, lft, rgt FROM Stack;
> -- the top column is not needed in the final answer
> -- move stack contents to new tree table
> END;
>
> I have a book on TREES & HIERARCHIES IN SQL which you can get at
> Amazon.com right now.
>
> --CELKO--
> Please post DDL, so that people do not have to guess what the keys,
> constraints, Declarative Referential Integrity, datatypes, etc. in your
> schema are. Sample data is also a good idea, along with clear
> specifications.
>
>
> *** Sent via Developersdex http://www.developersdex.com ***
> Don't just participate in USENET...get rewarded for it!
```