Building the Hierarchy Bridge Table

By Ralph Kimball

In our books and classes, we describe a powerful technique for representing complex ragged hierarchies of indeterminate depth. The centerpiece of the design is a bridge table whose grain is every *path* from any given node in the tree to all the descendants below that node.

Please see pages 215 to 224 in *The Data Warehouse Toolkit, Third Edition*, for an in-depth discussion of this approach to modeling ragged hierarchies.

At first glance it is not obvious how to build the bridge table. Typically, the original representation of the tree structure is a set of recursive records, where each record represents a node in the tree and contains a recursive pointer to the unique parent of that node. For example, if we had a fictitious company called “Microsoft” whose organization structure is represented by 18 nodes, the original representation of the tree could be stored in a “Company” table as follows:

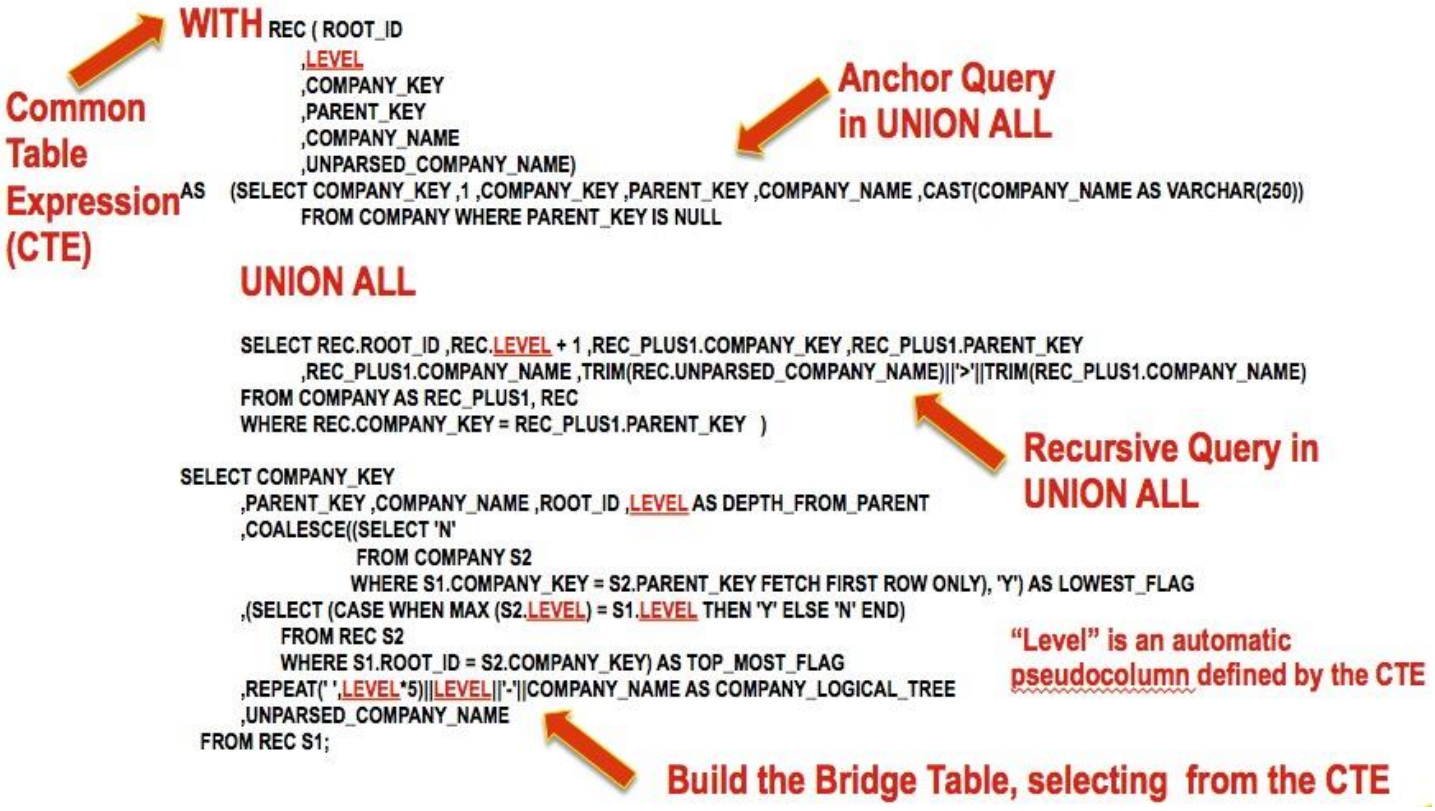
```
Create table COMPANY (
COMPANY_KEY INTEGER NOT NULL,
COMPANY_NAME VARCHAR(50),
PARENT_KEY INTEGER);

INSERT INTO COMPANY VALUES (100,'MICROSOFT',NULL);
INSERT INTO COMPANY VALUES (101,'SOFTWARE',100);
INSERT INTO COMPANY VALUES (102,'CONSULTING',101);
INSERT INTO COMPANY VALUES (103,'PRODUCTS',101);
INSERT INTO COMPANY VALUES (104,'OFFICE',103);
INSERT INTO COMPANY VALUES (105,'VISIO',104);
INSERT INTO COMPANY VALUES (106,'VISIO EUROPE',105);
INSERT INTO COMPANY VALUES (107,'BACK OFFICE',103);
INSERT INTO COMPANY VALUES (108,'SQL SERVER',107);
INSERT INTO COMPANY VALUES (109,'OLAP SERVICES',108);
INSERT INTO COMPANY VALUES (110,'DTS',108);
INSERT INTO COMPANY VALUES (111,'REPOSITORY',108);
INSERT INTO COMPANY VALUES (112,'DEVELOPER TOOLS',103);
INSERT INTO COMPANY VALUES (113,'WINDOWS',103);
INSERT INTO COMPANY VALUES (114,'ENTERTAINMENT',103);
INSERT INTO COMPANY VALUES (115,'GAMES',114);
INSERT INTO COMPANY VALUES (116,'MULTIMEDIA',114);
INSERT INTO COMPANY VALUES (117,'EDUCATION',101);
COMMIT;
```

The first value in each record is the node number, and the third value is the recursively defined parent node number.

In order to build the required path table, you can invoke an SQL common table expression (CTE) with an embedded UNION ALL statement. This specific combination of the CTE together with UNION ALL allows the first SELECT statement in the UNION ALL to act as the “anchor query” and the second SELECT statement to act as the “recursive query”. For this situation, ANSI standard SQL also exposes the pseudo-column “level” which counts the number of levels down from the topmost node. For more

information on this advanced SQL syntax, there's a Wikipedia article entitled "Hierarchical and Recursive Queries in SQL."

Using CTE with UNION ALL, you can build the desired hierarchy bridge table with the SELECT statement following the CTE in the following:




```
WITH REC ( ROOT_ID
          ,LEVEL
          ,COMPANY_KEY
          ,PARENT_KEY
          ,COMPANY_NAME
          ,UNPARSED_COMPANY_NAME)
AS   (SELECT COMPANY_KEY ,1 ,COMPANY_KEY ,PARENT_KEY ,COMPANY_NAME ,CAST(COMPANY_NAME AS VARCHAR(250))
          FROM COMPANY WHERE PARENT_KEY IS NULL

UNION ALL

     SELECT REC.ROOT_ID ,REC.LEVEL + 1 ,REC_PLUS1.COMPANY_KEY ,REC_PLUS1.PARENT_KEY
          ,REC_PLUS1.COMPANY_NAME ,TRIM(REC.UNPARSED_COMPANY_NAME)||'>'||TRIM(REC_PLUS1.COMPANY_NAME)
     FROM COMPANY AS REC_PLUS1, REC
     WHERE REC.COMPANY_KEY = REC_PLUS1.PARENT_KEY  )

SELECT COMPANY_KEY
     ,PARENT_KEY ,COMPANY_NAME ,ROOT_ID ,LEVEL AS DEPTH_FROM_PARENT
     ,COALESCE((SELECT 'N'
                FROM COMPANY S2
               WHERE S1.COMPANY_KEY = S2.PARENT_KEY FETCH FIRST ROW ONLY), 'Y') AS LOWEST_FLAG
     ,(SELECT (CASE WHEN MAX (S2.LEVEL) = S1.LEVEL THEN 'Y' ELSE 'N' END)
          FROM REC S2
          WHERE S1.ROOT_ID = S2.COMPANY_KEY) AS TOP_MOST_FLAG
     ,REPEAT(' ',LEVEL*5)||LEVEL||'-'||COMPANY_NAME AS COMPANY_LOGICAL_TREE
     ,UNPARSED_COMPANY_NAME
FROM REC S1;
```