

## Kimball Design Tip #17: Populating Hierarchy Helper Tables

By Lawrence Corr

This month's tip follows on from Ralph's September 1998 article ['Help for Hierarchies'](#) which addresses hierarchical structures of variable depth which are most often represented in relational databases as recursive relationships sometimes known as 'pigs ears' or 'fish hooks'.

Below is the definition of a simple company dimension which contains such a recursive relationship between the foreign key PARENT\_KEY and primary key COMPANY\_KEY

```
Create table COMPANY (
  COMPANY_KEY      INTEGER NOT NULL,
  COMPANY_NAME     VARCHAR2(50),
  PARENT_KEY       INTEGER);
```

While this is efficient for storing information on organizational structures it is not possible to navigate or rollup facts within these hierarchies using the non-procedural SQL that can be generated by commercial query tools. Ralph's original article describes a helper table similar to the one below which contains one record for each separate path from each company in the organization tree to itself and to every subsidiary below it which solves this problem.

```
Create table COMPANY_STRUCTURE (
  PARENT_KEY       INTEGER NOT NULL,
  SUBSIDIARY_KEY   INTEGER NOT NULL,
  SUBSIDIARY_LEVEL INTEGER NOT NULL,
  SEQUENCE_NUMBER  INTEGER NOT NULL,
  LOWEST_FLAG      CHAR(1),
  HIGHEST_FLAG     CHAR(1),
  PARENT_COMPANY   VARCHAR2(50),
  SUBSIDIARY_COMPANY VARCHAR2(50));
```

The last two columns in this example which denormalize the company names into this table are not strictly necessary but have been added to make it easy to see what's going on later.

The following PL/SQL stored procedure demonstrates one possible technique for populating this 'hierarchy explosion' table from the COMPANY table on Oracle:

```
CREATE or Replace procedure COMPANY_EXPLOSION_SP as
  CURSOR Get_Roots is
  select COMPANY_KEY ROOT_KEY,
         decode(PARENT_KEY, NULL,'Y','N') HIGHEST_FLAG,
         COMPANY_NAME ROOT_COMPANY
  from COMPANY;
```

```

BEGIN

For Roots in Get_Roots
LOOP
    insert into COMPANY_STRUCTURE
    (PARENT_KEY,
    SUBSIDIARY_KEY,
    SUBSIDIARY_LEVEL,
    SEQUENCE_NUMBER,
    LOWEST_FLAG,
    HIGHEST_FLAG,
    PARENT_COMPANY,
    SUBSIDIARY_COMPANY)
    select
    roots.ROOT_KEY,
    COMPANY_KEY,
    LEVEL - 1,
    ROWNUM,
    'N',
    roots.HIGHEST_FLAG,
    roots.ROOT_COMPANY,
    COMPANY_NAME
    from
    COMPANY
    Start with COMPANY_KEY = roots.ROOT_KEY
    connect by prior COMPANY_KEY = PARENT_KEY;
END LOOP;

update COMPANY_STRUCTURE
    SET LOWEST_FLAG = 'Y'
where not exists (select * from COMPANY
where PARENT_KEY = COMPANY_STRUCTURE.SUBSIDIARY_KEY);

COMMIT;
END; /* of procedure */

```

This solution takes advantage of Oracle's CONNECT BY SQL extension to walk each tree in the data. While CONNECT BY is very useful within this procedure it could not be used by an ad hoc query tool. If the tool could generate this syntax to explore the recursive relationship it can not in the same statement join to a fact table. Even if Oracle was to remove this somewhat arbitrary limitation, the performance at query time would probably be not too good.

The following fictional company data will help you to understand the COMPANY\_STRUCTURE table and COMPANY\_EXPLOSION\_SP procedure:

```

/* column order is Company_key,Company_name,Parent_key */ 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);

```

```
insert into company values (118,'Online Services',100); insert into company values
(119,'WebTV',118); insert into company values (120,'MSN',118); insert into company values
(121,'MSN.co.uk',120); insert into company values (122,'Hotmail.com',120); insert into
company values (123,'MSNBC',120); insert into company values (124,'MSNBC Online',123);
insert into company values (125,'Expedia',120); insert into company values
(126,'Expedia.co.uk',125);
/* End example data */
```

The procedure will take the 27 COMPANY records and create 110 COMPANY\_STRUCTURE records make up of one big tree (Microsoft) with 27 nodes and 26 smaller trees. For large datasets, you may find that the performance can be enhanced by adding a pair of concatenated indexes on the connect by columns. In this example one on COMPANY\_KEY,PARENT\_KEY and the other on PARENT\_KEY,COMPANY\_KEY.

If you want to visualize the tree structure textually the following query displays an indented subsidiary list for Microsoft:

```
select LPAD( ' ', 3*(SUBSIDIARY_LEVEL)) || SUBSIDIARY_COMPANY from
COMPANY_STRUCTURE order by SEQUENCE_NUMBER where parent_key = 100
```

The SEQUENCE\_NUMBER has been added since the original article, it numbers nodes top to bottom, left to right. It allows the correct level 2 nodes to be sorted below their matching level 1 nodes.

For a graphical version of the organization tree take a look at VISIO 2000 Enterprise Edition which has a database or text file driven organization chart wizard. With the help of VBA script, a view on the COMPANY\_STRUCTURE table and a fact table it might automate the generation of just the HTML pages you want.

I would be very grateful if someone could email me with DB2 UDB and Microsoft SQL Server procedures to create the same results. More efficient Oracle implementations would be interesting too. The best examples will be credited and included in a future Intelligent Enterprise column.