
Tips and Tricks for Developer Forms Hierarchical Tree Tools

Jennifer Croy
Rumken, Inc.

Rumpi Gravenstein
Rumken, Inc

Michael Rife
*The Goodyear Tire
and Rubber Company*

Gerhardt Martin
*The Goodyear Tire
and Rubber Company*

Abstract

The Oracle-provided hierarchical tree display tools can be quite powerful. However, the NavWiz demonstration form and the succeeding Forms 6.0 hierarchical tree object have inherent deficiencies in functionality. Perhaps most importantly, the query method proves too slow for large hierarchies. These hierarchical tree tools can be tuned to dramatically improve their speed, utility, and features. This presentation will focus on describing the methods and techniques used to successfully implement the NavWiz tool in an actual Oracle Web Forms deployment. Techniques that will be discussed include optimization approaches for handling large hierarchies, branch tree selections, and much more.

Topics to Discuss

- [Introduction](#)
- [Primer on the Oracle Provided Hierarchical Tree Tools](#)
- [Problems In Paradise](#)
- [Hierarchical Query Optimizations](#)
- [Branch Tree Selections](#)
- [Supporting Multiple Tree Objects](#)
- [Conclusion](#)
- [Appendix](#)

Introduction

Definition of Hierarchical Query

A hierarchical query is a SQL query that shows parent child relationships between records that are often in the same table. Almost all data has a natural hierarchical structure which can be used to present data in a more meaningful way. An example is the employee hierarchy within an organization where you can determine, just by

looking at the hierarchy, a number of important attributes like the number of departments and the size of a department. Another reason that data is easier to understand when displayed in a hierarchical structure is that it becomes much easier to search. For instance, if you wanted to find the secretary or treasurer of a company it is easier to look through an organizational chart than having to scan each name and the associated job title. For these reasons I have been using hierarchical displays in an increasing number of applications.

Consider the EMP demonstration table provided by Oracle.

EMP

EMPNO Employee Number
 ENAME Employee Name
 JOB Employee job title
 MGR Employee number of this employee's manager

In this table the MGR column refers to the EMPNO column in another row in the same table. All but one employee, the President, has a manager whose employee record can also be found in the EMP. To make this type of search easy, Oracle has provided support for what is referred to as a hierarchical query. The SQL constructs that support this type of query are START WITH and CONNECT BY PRIOR. Start with is used to identify where in the hierarchical tree the query should start while connect by describes how one record relates to the next. In the case of the emp table the following query can be used to display the entire hierarchy. In this query it is assumed that the President of the company will not have a manager and therefore if the query is to display the entire organizational chart it needs to start where the mgr field is either NULL or the JOB is PRESIDENT. I have added some spacing and the pseudo column, Level, to illustrate this tree in the following SQL.

```
select lpad(' ',3*(level-1))||ename name, job, mgr, empno, level
  from emp
 start with mgr is null
 connect by prior empno = mgr
```

| NAME | JOB | MGR | EMPNO | LEVEL |
|-------|-----------|------|-------|-------|
| KING | PRESIDENT | | 7839 | 1 |
| JONES | MANAGER | 7839 | 7566 | 2 |
| SCOTT | ANALYST | 7566 | 7788 | 3 |
| ADAMS | CLERK | 7788 | 7876 | 4 |
| FORD | ANALYST | 7566 | 7902 | 3 |
| SMITH | CLERK | 7902 | 7369 | 4 |
| BLAKE | MANAGER | 7839 | 7698 | 2 |
| ALLEN | SALESMAN | 7698 | 7499 | 3 |

| | | | | |
|--------|----------|------|------|---|
| WARD | SALESMAN | 7698 | 7521 | 3 |
| MARTIN | SALESMAN | 7698 | 7654 | 3 |
| TURNER | SALESMAN | 7698 | 7844 | 3 |
| JAMES | CLERK | 7698 | 7900 | 3 |
| CLARK | MANAGER | 7839 | 7782 | 2 |
| MILLER | CLERK | 7782 | 7934 | 3 |

14 rows selected.

The Level pseudo column shows the depth at which the row is in the hierarchy. Thus the president is at Level 1 while those that report to him are at level 2 and so on. If you are interested in just the part of the tree that includes Blake and all of the people that report to him, the query would be

```
select lpad(' ',3*(level-1))||ename name, job, mgr, empno
from emp
start with ename = 'BLAKE'
connect by prior empno = mgr
```

| NAME | JOB | MGR | EMPNO |
|--------|----------|-------|-------|
| ----- | ----- | ----- | ----- |
| BLAKE | MANAGER | 7839 | 7698 |
| ALLEN | SALESMAN | 7698 | 7499 |
| WARD | SALESMAN | 7698 | 7521 |
| MARTIN | SALESMAN | 7698 | 7654 |
| TURNER | SALESMAN | 7698 | 7844 |
| JAMES | CLERK | 7698 | 7900 |

6 rows selected.

Information to be Aware of When Using Hierarchical Queries

There are a couple of caveats that need to be pointed out. First, in a hierarchical query, avoid use of the WHERE clause. The WHERE clause is evaluated after the hierarchy has been created. Therefore, evaluation of the WHERE clause may create “gaps” in the hierarchy. When displaying the hierarchy tree via the tree tools, the results may not be correct if gaps exist. For instance, in the following example, JONES has two people reporting directly to him: SCOTT and FORD. JONES is being excluded from the hierarchy. However, the tree tool displays this query as FORD reporting to SCOTT, rather than showing FORD and SCOTT at the same level.

```
select lpad(' ',3*(level-1))||ename name, job, mgr, empno
from emp
where ename != 'JONES'
start with ename = 'JONES'
connect by prior empno = mgr
```

| NAME | JOB | MGR | EMPNO |
|-------|---------|------|-------|
| SCOTT | ANALYST | 7566 | 7788 |
| ADAMS | CLERK | 7788 | 7876 |
| FORD | ANALYST | 7566 | 7902 |
| SMITH | CLERK | 7902 | 7369 |



Instead, one should use the **CONNECT BY** and **START WITH** statements to include any needed row restrictions. This will mean that if a node is excluded, its entire subtree will also be excluded.

Second, in a hierarchical query, if you use the **ORDER BY** clause, the rows are not ordered by the hierarchy tree but rather by the **ORDER BY** columns. The following SQL demonstrates this issue:

```

select lpad(' ',3*(level-1))||ename name, job, mgr, empno, level
from emp
start with mgr is null
connect by prior empno = mgr
order by name

```

| NAME | JOB | MGR | EMPNO | LEVEL |
|--------|----------|------|-------|-------|
| ADAMS | CLERK | 7788 | 7876 | 4 |
| SMITH | CLERK | 7902 | 7369 | 4 |
| ALLEN | SALESMAN | 7698 | 7499 | 3 |
| FORD | ANALYST | 7566 | 7902 | 3 |
| JAMES | CLERK | 7698 | 7900 | 3 |
| MARTIN | SALESMAN | 7698 | 7654 | 3 |
| MILLER | CLERK | 7782 | 7934 | 3 |
| SCOTT | ANALYST | 7566 | 7788 | 3 |
| TURNER | SALESMAN | 7698 | 7844 | 3 |
| WARD | SALESMAN | 7698 | 7521 | 3 |
| BLAKE | MANAGER | 7839 | 7698 | 2 |
| CLARK | MANAGER | 7839 | 7782 | 2 |

| | | | | |
|-------|-----------|------|------|---|
| JONES | MANAGER | 7839 | 7566 | 2 |
| KING | PRESIDENT | | 7839 | 1 |

14 rows selected.

The net affect of this behavior is that there is no easy way in which to order the employees that report to BLAKE, or even to ensure that they will always appear in the same order.

There are some tricks to affect the order in which the records are displayed. One method is discussed [later in this paper](#). Another way is to create an index that will affect the order in which the records are retrieved from the database.

Hierarchical Tree Tools

A hierarchical tree tool displays a hierarchical query as a navigator-style window, similar to NT Explorer (see Figure 1). You can click on any individual line to display or hide the corresponding data.

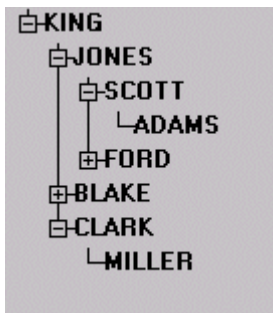


Figure 1

Developer Forms 5.0 contained a demonstration form, NavWiz.fmb, and a forms library, Navigator.pll, that displayed a hierarchical tree based on the scott/tiger schema. This demonstration could be adapted for use in your own application, but was not very sturdy or particularly easy to use.

Developer Forms 6.0 now has a built-in hierarchical tree object. The built-in object seems significantly sturdier and easier to use. It does retain the manipulation procedures that were found in the Forms 5 demo, allowing for more developer customization and improvement.

Primer on the Oracle Provided Hierarchical Tree Tools

Forms 5.0 NavWiz Demo Tool

Developer Forms 5.0 had a demonstration hierarchical tree tool. This method used just the normal items readily available within Forms: a multi-record block with a single text item, some control items, a stacked canvas, and a window. You also attached the Navigator.pll library, which provided the functions to manipulate the items.

One primary limitation I saw with this approach was that the names of the items were hard-wired into the Navigator library. This not only meant that you were required to use the exact names (“NAVIGATOR”, “NAVIGATOR_CONTROL”, etc.) it was expecting, but also meant that you could not have more than one hierarchical tree per form. I later [adapted](#) the Navigator library to allow multiple trees within a single form.

The Forms 5.0 demo tool supported features such as multi-select and range-select using the CTRL and SHIFT keys, at the developer or user’s discretion. It supported replacing the entire tree or adding and deleting nodes. The elements of the tree were simply recorded in a record group that could be modified.

Forms 6.0 Hierarchical Tree Object

With the release of Developer 6.0, a drop-in hierarchical tree tool was included. To create a tree, create a block item and set the item type to “Hierarchical tree.” You should then add a call to the FTREE.populate_tree Built-in, usually in the WHEN-NEW-FORM-INSTANCE trigger, to populate the tree.

This tool allowed multiple tree objects per form, without any modifications. It also supported the same features found in the Forms 5.0 demo tool, such as multi-select and range-select.

Problems in Paradise

Hierarchical Query Performance

Despite all the features available, neither the Version 5 nor the Version 6 Developer 2000 hierarchical tree tools are perfect “out of the box.” The biggest obstacle to widespread use of the hierarchical tree is the amount of time it can take to display a large tree. This is a problem because, in my experience, hierarchical trees are used primarily when there is a large amount of data to display. Unfortunately, this is just what the Oracle provided tool does poorly. The default hierarchical tree tool behavior is to run the entire query at once, before displaying the tree. When there are thousands of rows this can take a long time, time that is not needed if you are only interested in the first or second level or a particular branch in the tree. Clearly, this isn’t a viable solution for large interactive queries where your form users would not be able to continue until the query finished.

Branch Tree Selections

I also wished to have the option of a branch tree selection, an often requested feature. That is, when a user selects a node for use, that the application would automatically recognize that all the descendants of the “selected” node should also be selected for use. For instance, in an employee hierarchy, an example of a desired functionality might be to click on a manager and select an option to send an email to that person and all people that report underneath him. The Oracle provided tool does not include this feature.

Supporting Multiple Tree Objects

Finally I wished to be able to utilize multiple trees within the same form. The Forms 5.0 demonstration library had apparently not been built with this flexibility in mind. The ability to display multiple tree objects is very desirable in applications that have a number of complicated selections a user must make to define report parameters or enter a complex record.

Hierarchical Query Optimizations

Looking to solve the hierarchical tree performance problem, I have added my own forms library, to supplement the Oracle-provided Forms 5.0 NavWiz demo tool. This library traverses and displays the tree by querying for each hierarchy level individually starting with the main level. Then, as the user drills into a lower level, the application queries data for that level as well. This way, the only information that is accessed is that which the user has actually requested. This is similar to the behavior of NT Explorer.

Taking this approach has a second desired benefit, as well. Since each level is being retrieved individually, the results within a level can be displayed in a given order.

In my supplemental library, I created several support routines, to handle the dynamic creation and expansion of the hierarchical tree.

Support Routines in the supplemental library:

```
PROCEDURE Add_Smart_Nav_Row(tree varchar2, menu_node_id number);
PROCEDURE Create_Initial_Tree(tree varchar2);
PROCEDURE Create_Smart_Nav(tree varchar2);
FUNCTION Get_Index_Number (tree varchar2, menu_node_id number) RETURN NUMBER;
PROCEDURE Populate_Level (tree varchar2, parent_index number);
PROCEDURE Populate_Tree_Child (tree varchar2, menu_node_id number);
```

Summary of the Externally Referenced Routines:

At the time of first population (usually occurring in the WHEN-NEW-FORM-INSTANCE trigger), you would call the Create_Initial_Tree routine. This procedure populates the top level of the tree. It then populates one child under each node as a placeholder. Population of at least one child is necessary so that nodes with children are shown as collapsed nodes, rather than being shown as leaf nodes. Another option would be to enter a “bogus” node under every node. This bogus node would then be deleted when that level is populated. This method would likely save some time, since it would not be necessary to retrieve an additional child for every node. It has the disadvantage of showing all nodes as initially being collapsed, even if it is actually a leaf.

The Create_Smart_Nav routine creates a supplemental record group within Forms to keep track of the information for each node. There are three columns in the record group: menu_node_id (the value for the node), all_children_populated_flag (an indicator of whether that node has been expanded), and child_populated_menu_node_id (the value for the populated child node). If a node has no children, it is judged to be a leaf node and the all_children_populated_flag is set to “Y”, to indicate that future population is not necessary. If you use the bogus node method, the child_populated_menu_node_id column would not be needed.

So, at initial population, the supplemental record group for the employee hierarchy would appear as follows:

| Shown Name | Menu_node_id | All_children_populated_flag | Child_populated_menu_node_id | Child Name |
|------------|--------------|-----------------------------|------------------------------|------------|
| KING | 7839 | N | 7566 | JONES |
| JONES | 7566 | N | 0 | |

When the user expands the KING node, the “expand” trigger will check the value of the KING node’s all_children_populated_flag. Since the value is currently “N”, the remainder of KING’s children will be

retrieved, using the Populate_Level procedure. Populate_Level retrieves all children for the given node, retrieves one child for each of these second level nodes, and then sets the all_children_populated_flag.

Now, the supplemental record group will now appear as:

| Shown Name | Menu_node_id | All_children_populated_flag | Child_populated_menu_node_id | Child Name |
|------------|--------------|-----------------------------|------------------------------|------------|
| KING | 7839 | Y | 7566 | JONES |
| BLAKE | 7698 | N | 7521 | WARD |
| WARD | 7521 | N | 0 | |
| CLARK | 7782 | N | 7934 | MILLER |
| MILLER | 7934 | N | 0 | |
| JONES | 7566 | N | 7788 | SCOTT |
| SCOTT | 7788 | N | 0 | |

In this manner, the next level of the hierarchy is dynamically populated as each node is expanded.

Summary of the Internal Procedures:

Add_Smart_Nav_Row adds a row to the Smart Navigator record group. This is done for every node in the hierarchy. This is called from multiple routines, including Create_Initial_Tree and Populate_Tree_Child.

Create_Smart_Nav deletes and recreates the Smart Navigator record group, which is used for maintenance of the tree. This is called from Create_Initial_Tree.

Get_Index_Number finds the tree index in the Oracle Navigator tree for the given node id. This is used both internally and externally.

Populate_Tree_Child populates a single child for the given parent node, if any children exist. This is called from Populate_Level.

Future Enhancement Ideas

- Due to time pressures, I have not yet upgraded this application to Forms Developer 6.0. However, in the next phase of the project, I intend to convert the application and take advantage of the built-in hierarchical tree tool.
- My supplemental library has thus far been created specifically for my application, with hard-coded SQL cursors. The potential exists to genericize this library for use with any SQL query. However, I found the attempt to be too time-consuming to be worthwhile at this time. One possible method might be to utilize DBMS_SQL's ability to run dynamic SQL statements. It appears this approach would likely require database procedures as well as the Form libraries, due to the limitations on the use of DBMS_SQL inside of Forms.

Branch Tree Selections

Another often requested feature was branch tree selection. The desired functionality was to easily allow a node and its entire subtree (or descendants) to be selected at once. In my application, I decided not to have the screen highlight every single descendant visually, but rather to do the work behind the scenes. The user would click on a button to indicate that the currently selected node should be used as criteria. (Figure 2.) Upon that click, I would then loop through and execute my “selection” logic for the currently selected node, as well as each node in its subtree. (Figure 3.)



Figure 2



Figure 3

I feel it is best to implement branch tree selection on a case-by-case basis, rather than building this functionality into the default behavior. The desired behavior may not necessarily always be to select the descendants. For instance, in the [previously mentioned example](#) of being able to email a manager and all his employees, you might decide to also implement the functionality of allowing the user to email the manager alone, rather than emailing the entire group.

You could approach this in a few ways. Approach A: you could give two different functions, one for emailing the group and one for emailing the person alone. In Approach B, you could choose to give a generic email function and a different function that would allow the user to highlight the entire subtree of the selected node. That way, they would click to highlight the subtree, then click to email all selected nodes. Depending on the situation, either approach could be used.



The “Email this Employee and his Staff” trigger:

```

DECLARE
    CURSOR subtree IS
        SELECT empno
        FROM emp
        CONNECT BY PRIOR empno = mgr
        START WITH empno = :employee_hierarchy;
BEGIN
    FOR employee IN subtree LOOP
        Email(employee.empno);
    END LOOP;
END;
```

Supporting Multiple Tree Objects

Allowing Multiple Tree Items within the Forms 5.0 demo tool

The Forms 5.0 demo library hardcoded the expected names of the tree tool support items. This approach meant that you could not have more than one hierarchical tree object per form. I adapted the Navigator library to allow multiple trees within a single form. This was accomplished by allowing the user to pass in the name of the “tree”. For every subroutine in the Navigator.pll library that accessed a form item, I allowed a tree name to be passed in as an optional parameter. The previously hard-wired “Navigator” was made the default, so that previous references to the Navigator.pll would continue to work without needing any revisions. I based the name of all of the form items on this tree name. For instance, if the tree name was given as “TREE_A”, I would look for the control items in the “TREE_A_CONTROL” block. This limitation could be removed as well, if needed. However, since Developer 6.0 with its built-in hierarchical tree had already been released when I made this modification, I chose not to bury too much time in expanding older technology.

Conclusion

Displaying data in a hierarchical tree fashion can show much information about the structure of the data. However, the default behavior of the Oracle-provided tools is not always optimal. With the addition of some routines, the behavior and functionality can be expanded and customized to the needs of any application.