

USING HIERARCHICAL ITEMS IN DEVELOPER 6i

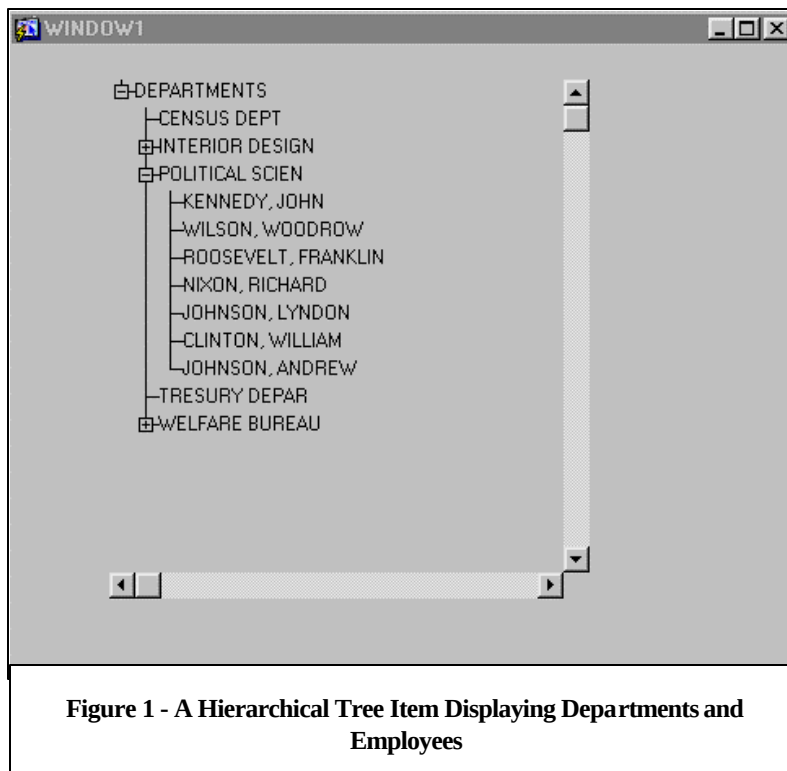
John Adolph Palinski, Omaha Public Power District, Realistic Software Training

John Adolph Palinski, "Using Hierarchical Tree Items in Oracle's Developer 6i," Oracle Internals 2 (8): 14-19 (October 2000). Copyright © 2000 CRC Press LLC. Reprinted by permission.

John Adolph Palinski, "Populating Oracle Form Hierarchies Incrementally at Run-Time," Oracle Internals 2 (9): 9-15 (November 2000). Copyright © 2000 CRC Press LLC. Reprinted by permission.

Hierarchical Item Basics

A Hierarchical Tree is a form item that displays data in the form of a tree. Each node or branch of the tree has the ability to expand or collapse. An example is shown in Figure 1. This item is similar to Developer 6i's Object Navigator or Windows Explorer. This item is very useful when the user has a great deal of related data to search through in order to find or drill down to the proper lower level data. Examples of the type of data that can be displayed in this manner are department and employees or a machine and its associated parts lists. The hierarchical tree item allows the user to initially display a list of the various machines. After a machine is identified, the tree branch can be expanded displaying the part list for the item. It is even possible to navigate from the hierarchical tree to another form containing detailed information about the selected record.



Populating the Hierarchical Tree Item

Hierarchical tree items are contained in non-database table blocks and they are the only item on the block. The tree item has a property called 'Data Query'. This property contains a five expression select statement that will be used to retrieve the data into the tree item. To repeat, the **select** clause may only contain five expressions. Each of the expressions has a specific purpose. The expression functionality follows:

- Expression 1** - This expression contains a value that expresses the state of the record. State indicates whether the record is shown (expanded), not shown (collapsed), or shown at the same level as the parent record. A value of '1' will cause the record to be shown. A value of '-1' will cause the record to be hidden with the parent record marked as having children record. A value of '0' will cause the record to be displayed at the same level as the parent. This expression is referenced as tree property **Ftree.node_state**.
- Expression 2** - This expression contains a value that ranks the record in the hierarchy. For example, if the record is a grandparent it would have a value of 1; if it were the parent it would have a value of 2; and finally the child record would have a value of 3. This expression is referenced as tree property **Ftree.node_depth**.
- Expression 3** - This expression contains the value that will be displayed in the tree when the record is displayed. The value can be literal text if the node is a description of the next level of records such as 'Departments' in Figure 1 or the name of a table column if it represents the actual table values. This expression is referenced as tree property **Ftree.node_label**.
- Expression 4** - This expression identifies the icon that can be shown next to the node. This is a comparable feature to the icons used in Object Navigator to represent various objects (i.e. canvas, data blocks, items). The referenced icons must be located in the current working directory or the directory listed in the UI_ICON string value in the registry. A value of 'null' can be used if no icon is desired. This expression is referenced as tree property **Ftree.node_icon**.
- Expression 5** - This expression contains a value that is used to identify the record used on the current node. It is generally the primary key of the record that supplied the value and can be used when navigating to another form. . This expression is referenced as tree property **Ftree.node_value**.

The hardest part of using the Hierarchical Tree item is developing the select statement that will put the proper rank on each record and sort the records. Records are displayed in the tree as they are retrieved or sorted

from the database. A special technique (shown later in the article) was used in the select statement used in Figure 1 to order the displayed the employees under the proper department.

The easiest type of hierarchical tree query is one for recursive data. This means the data comes from one table and each record is related to another record in the same table. Examples of this type of data are university courses that are a prerequisite for another course. Each record in the table has a Prerequisite column that contains a Course ID value. The field is a foreign key to the Course ID column on the same table.

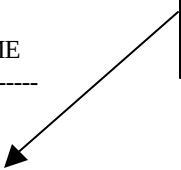
A second example is an employee record that contains the employee number of the employee's immediate supervisor. Oracle provides a sample table called EMP with employee records that have such a recursive relationship. Using the Start With/Connect By clauses, the records can be easily ordered and analyzed as Listing 1 illustrates:

Listing 1 - Using the Start With/Connect By Clauses to Order and Rank Rows

```
SQL> column employee_name format a40
SQL> select level, empno, mgr,
2   lpad(' ', level * 2)||ename employee_name
3   from emp
4   start with mgr is null
5   connect by prior empno = mgr;
```

LEVEL	EMPNO	MGR	EMPLOYEE_NAME
1	7839		KING
2	7566	7839	JONES
3	7788	7566	SCOTT
4	7876	7788	ADAMS
3	7902	7566	FORD
4	7369	7902	SMITH
2	7698	7839	BLAKE
3	7499	7698	ALLEN
3	7521	7698	WARD
3	7654	7698	MARTIN
3	7844	7698	TURNER
3	7900	7698	JAMES
2	7782	7839	CLARK
3	7934	7782	MILLER

Records are
ordered by
level



14 rows selected.

SQL>

End listing

As you can see the Start With/Connect By clauses prepare the data for display in a hierarchical item in the proper order. The clauses analyze and sort the data properly.

However, in the real world very little data is recursive. Most data comes from different tables. This causes the developer to perform special work. The data can be assembled and ranked properly using the Union All set operator. The records retrieved by each of the select statement are given their own level. The problem occurs when the ordering occurs. Ordering is not done by rank.

To illustrate, assume you want to develop a select statement that will present data in the same manner as in Figure 1. The data has the following characteristics:

The highest rank has a textual description called 'Departments'.

The next level is the various department names (i.e Census Dept, Interior Des).

The lowest level is the employees.

In order to rank the data properly, the textual description has a rank of 1, the department names have a rank of 2, and employees have a rank of 3. Listing 2 illustrates the select statement and its output.

Listing 2 – Select Statement Ranking and Sorting Departments and Employees

```
SQL> select -1 state, 1 lvl, 'DEPARTMENTS' displ_val, null icn, null val
2 from dual
3 union all
4 select -1 state, 2, department_name, null, department
5 from department
6 union all
7 select -1 state, 3, last_name||', '||first_name, null, to_char(payroll_number)
8 from employee;
```

STATE	LVL	DISPL_VAL	I VAL
-1	1	DEPARTMENTS	
-1	2	CENSUS DEPT	CEN
-1	2	INTERIOR DESIGN	INT
-1	2	POLITICAL SCIEN	POL
-1	2	TRESURY DEPAR	TRF
-1	2	WELFARE BUREAU	WEL
-1	3	BUSH, GEORGE	36
-1	3	CARTER, JIMMY	34
-1	3	CLINTON, WILLIAM	37
-1	3	COOLIDGE, CALVIN	25
-1	3	EISENHOWER, DWIGHT	29
-1	3	FORD, GERALD	33
-1	3	HOOVER, HERBERT	26
-1	3	JOHNSON, ANDREW	21
-1	3	JOHNSON, LYNDON	31
-1	3	KENNEDY, JOHN	30
-1	3	NIXON, RICHARD	32
-1	3	REAGAN, RONALD	35
-1	3	ROOSEVELT, FRANKLIN	27
-1	3	TAFT, WILLIAM	23

Records are ordered by the position the select statement is within the overall statement

-1 3 TRUMAN, HAROLD

28

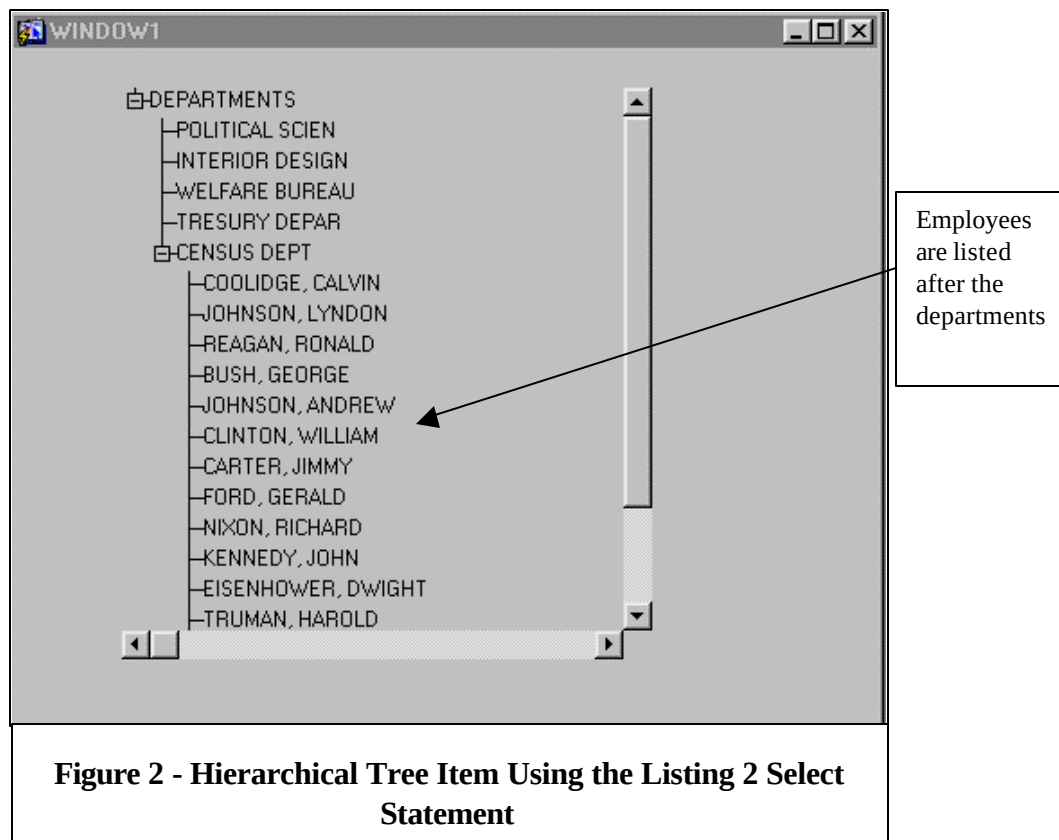
STATE	LVL	DISPL_VAL	I VAL
-1	3	WILSON, WOODROW	24

22 rows selected.

SQL>

End listing

The above statement can be used in the Hierarchical Tree item since it has all of the proper components. However it will not produce the desired results since the data is not sorted properly. Figure 2 illustrates a Hierarchical Tree item on a form using the select statement from Listing 2.



As you can see, records are displayed in the Hierarchical Tree exactly as retrieved by the select statement. In our original example, the employee for each department should follow the proper department name. The developer needs to find a way to sort the records properly. Level will not work as a sort criterion. The ordering task is further complicated in that only five expressions can be included in the select statement used within the property.

This dilemma is solved in the following manner:

1. Add a sixth expression to the query.
2. The value in the sixth record will be used to determine the proper sorting. In regards to our example, the sixth expression in the first select statement should have a value of 'A' since these records relate to the highest node. This will cause that record to be sorted to the top of the list. The sixth expression in the second clause should be the value of Department from the Department table. The value used in the third clause should be a concatenation of the Fk_department, last_name, and first_name columns in the Employee table. This will cause the employee records to be sorted within the proper department records.
3. Create a view of the select statement created in step 2.
4. Use the view in the Data Query property item. Use the first five expressions in Select clause and the sixth expression in the Order By clause.

Listing 3 illustrates the Select statement/ view with the sixth expression, the Select statement used in the Data Query property, and the results of the query.

Listing 3 Select Statement/View Uses to Properly Order Employee Data

```
SQL> create or replace view department_employees as
2 select 1 state, 1 lvl, 'DEPARTMENTS' displ_val, null icn, null val, 'A' sort_value
3 from dual
4 union all
5 select 1 state, 2, department_name, null, department, department
6 from department
7 union all
8 select 1 state, 3, last_name||', '||first_name, null, to_char(payroll_number),
9   fk_department||last_name||first_name
10 from employee;
```

View created.

```
SQL> select state, lvl, displ_val, icn, val
2 from department_employees
3 order by sort_value
4 ;
```

STATE	LVL	DISPL_VAL	I VAL
1	1	DEPARTMENTS	
1	2	CENSUS DEPT	CEN
1	2	INTERIOR DESIGN	INT
1	3	BUSH, GEORGE	36
1	3	COOLIDGE, CALVIN	25
1	3	EISENHOWER, DWIGHT	29
1	3	FORD, GERALD	33

View with
the sixth
expression

Data query
property
statement

Proper
sorting

1	3	TRUMAN, HAROLD	28
1	2	POLITICAL SCIEN	POL
1	3	CLINTON, WILLIAM	37
1	3	JOHNSON, ANDREW	21
1	3	JOHNSON, LYNDON	31
1	3	KENNEDY, JOHN	30
1	3	NIXON, RICHARD	32
1	3	ROOSEVELT, FRANKLIN	27
1	3	WILSON, WOODROW	24
1	2	TRESURY DEPAR	TRF
1	2	WELFARE BUREAU	WEL
1	3	CARTER, JIMMY	34
1	3	HOOVER, HERBERT	26
1	3	REAGAN, RONALD	35

STATE	LVL	DISPL_VAL	I VAL
1	3	TAFT, WILLIAM	23

22 rows selected.

SQL>

End Listing

The select statement shown in Listing 3 produced the results in Figure 1. This technique should allow you to easily assemble and sort data from many tables when a recursive relationship does not exist.

Creating the Hierarchical Item on the Form

Creating the SQL to populate the Hierarchical Item is the hardest part of the procedure to create the item. The following steps can be followed to complete the task:

1. Create a non-database table block. The tree must be the only item on the block, and the block's 'Number of Records Displayed' property should be set to 1.
2. Open the Layout Editor.
3. Click the Hierarchical tool (displayed on the right).
4. Draw/size the Hierarchical item by clicking and dragging.
5. Set the properties of the item, especially the Data Query property.
6. Create a trigger to populate the tree item.



A common trigger is a Pre-form trigger. The following is a sample trigger that can be used:

Ftree.populate_tree('block_name.tree_item_name');

The form can now be executed and the tree populated. FTREE is the name of a built-in package that contains several procedures that can be used with trees. They are listed later in the article.

Hierarchical Tree Item Specific Properties

Hierarchical tree items contain several properties that are peculiar to the item. These properties along with a description are contained in Table 1.

Table 1 Hierarchical Tree Specific Properties

Property Name	Description
Allow Empty Branches	This property determines whether nodes that contain children will display the collapsed/expanded symbol. The default value of 'Yes' will show the symbol on nodes that do not contain children. A value of 'No' will suppress them.
Multi-Selection	This property determines whether the user can select multiple items while holding down the shift button. A value of 'No' will disable this function. A value of 'Yes' will allow multi-selection.
Show Lines	This property determines whether a line links the Tree nodes. Set the value to 'Yes' to display the lines, and 'No' to suppress the lines.
Show Symbols	This property determines whether the nodes can be expanded and collapsed. A value of 'No' will disable the functionality and a value of 'Yes' will enable it. When the 'No' value is selected, the tree will be fully expanded.
Record Group	This property contains the name of the Record Group to be used to supply data to the tree. This property should not be filled in if the Data Query property is populated.
Data Query	This property contains a select statement that is used to supply data to the tree. This property should not be filled in if the Record Group property is populated.

Populating Other Blocks or Forms Based Upon the Selected Node

It is possible to populate forms or other blocks based upon the selected node. Figure 3 illustrates this. The form containing the Tree item was modified. A database block called 'Employee' was placed on the form. This block is based upon the Employee table, the same table as the third select statement in the statement used in the Tree's Data Query property. This statement uses Payroll_number as the data value or fifth expression. Thus, the data value for this node can be used as the primary key for identifying the record and populating the new Employee block.

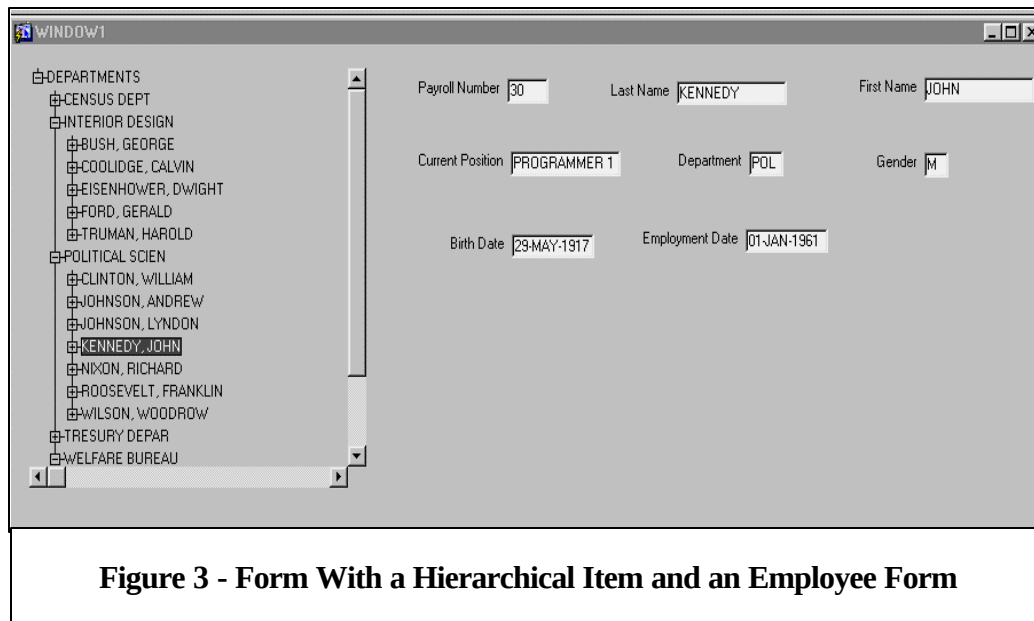


Figure 3 - Form With a Hierarchical Item and an Employee Form

A block level 'When-tree-node-selected' trigger is used to populate the Employee block. Each time a node is selected, the trigger will fire. The trigger is placed on the same block as the tree item. Listing 4 is a sample trigger. This trigger uses the `Ftree.get_tree_node_property` function to return selected values from the tree item.

Each time the trigger is fired, it first determines whether the selected node was for an Employee record. If it were not, a message is displayed informing the user an incorrect node was selected. If a valid node is selected, the executable within the If statement is launched and the Employee block is populated based upon the value supplied from the `Ftree.get_tree_node_property` function.

The tree values or parameter in the `Ftree.get_tree_node_property` function are:

1. `Tree_control.tree4` - The block and Tree item names.
2. `.:System.trigger_node` - System value containing the value of the currently selected node.
3. `Ftree.node_depth` - The rank or depth property.

Listing 4 – Sample When-tree-node-selected Trigger

```

declare
    rank    number := 0;
begin
    rank := ftree.get_tree_node_property('tree_control.tree4',
        :system.trigger_node, ftree.node_depth);
    if rank = 3 then
        go_block ('employee');
        set_block_property ('employee', default_where,
            'payroll_number='||ftree.get_tree_node_property('tree_control.tree4',
                :system.trigger_node, ftree.node_value));
        execute_query;
    else
        message ('Please select an employee node');
    end if;
end;

```

End listing

Built-in Triggers

Table 2 contains the Ftree package built-ins that can be used with Hierarchical trees.

Table 2 – Built-in Triggers for Tree Items

Name	Description
Get_tree_property	Retrieves a specific tree property for a target node.
Set_tree_property	Sets a specific tree property for a target node.
Get_tree_selection	Returns the values for selected nodes.
Set_tree_selection	Selects or highlights the target node.
Get_tree_node_property	Retrieves a property for a target node.
Set_tree_node_property	Sets a property for a target node
Get_tree_node_parent	Determines the id of the parent of the target node.
Find_tree_node	Locates the next node that has a data value that matches the target value.
Populate_tree	Launches the query and places the results into the tree.
Add_tree_data	Adds a new set of data under the identified node.
Add_tree_node	Adds a new node to the tree under the identified node.

Populate_group_from_tree	Places the data within the tree into an existing record group.
Delete_tree_node	Removes a node from the tree.

Exercises

Create a Hierarchical Tree item for Departments and their employees. Be sure that the first level displays the word 'Departments'

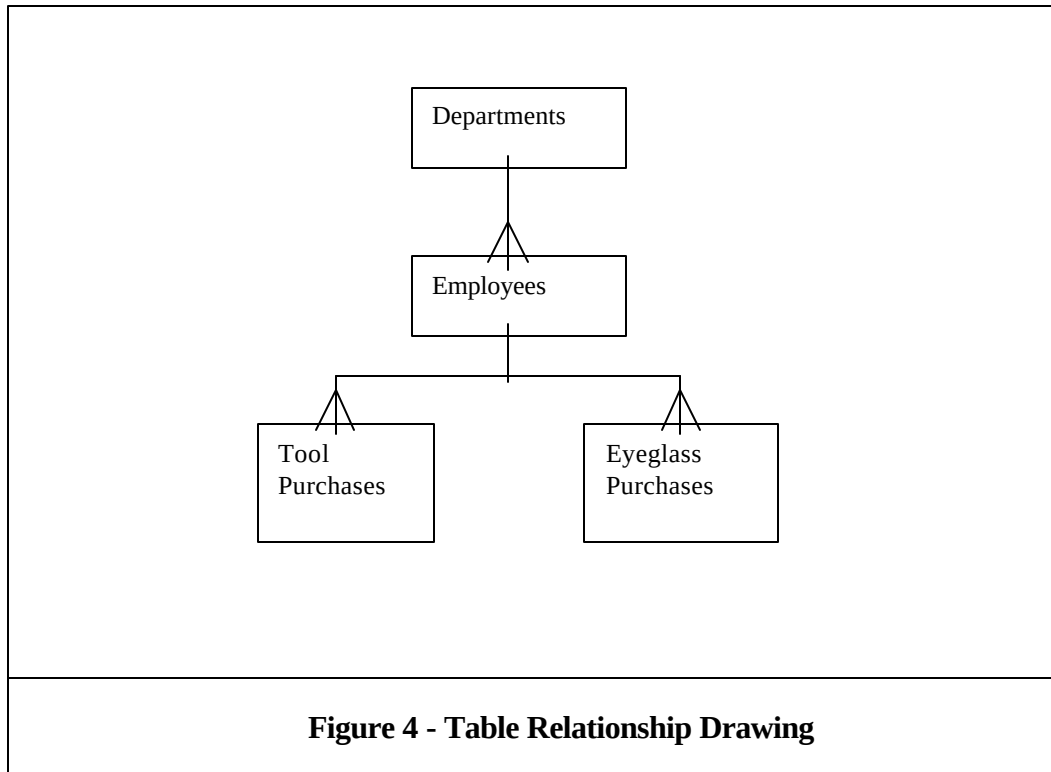
Create a second block on the form created in 1. This block will contain information about the selected employee. Create a trigger that populates this block when an employee is selected on the tree.

Modify the Hierarchical Tree to show an Employee Tool and Eyeglass Purchases. The Hierarchy should contain the labels 'Tools' and 'Eyeglasses'.

Authors Note: The database scripts needed to perform the exercises or any of the examples can be downloaded from www.oracle-trainer.com on the Oracle Internals page.

Populating Oracle Form Hierarchies Incrementally At Run Time

Much of the data in databases is contained in grandfather, father, and child relations. For example, a company may have a number of departments. Each department has employees, and each employee can have eyeglass and tool purchases. This relationship can be seen in Figure 4.



A Hierarchical item is an excellent tool to explore the records for a particular instance (i.e. department, employee). Using a hierarchy for the database depicted in Figure 1, at the highest level, the hierarchy will show a list of departments. Expanding a particular Department node will display the department's employees. Expanding the Employee node will cause the tool and eyeglass purchases to be displayed. This type of procedure is called "Drilling Down" and can be employed with almost every database.

Problem With Hierarchical Items

The previous section described how to create and populate a Hierarchical item and associate corresponding data block with the selected node or record. However, the section did not mention one serious problem with the Hierarchical item. This problem is the amount of data that can be potentially displayed in the item. A Hierarchical item is similar to an LOV. It accepts the results of a Select statement. If the initial Select statement retrieves a large amount of records, Forms must load all of these records into memory. This can take a lot of

time, wasting your system's resources as well as the user's time. The technique discussed in the previous article discussed how to load the hierarchy at one time.

If your Hierarchical item can contain a large number of records, a better way is to initially load as little data into the Hierarchical item as is necessary. As the user begins to drill down, add the additional data to the Hierarchical item at that time. This will minimize the data contained in memory and the time needed to initially display the form. In my previous article, I explained how to load all of the possible data into the Hierarchical item when the form is displayed using a Pre-form trigger. In this article, I will attempt to show you how to minimize the initial load and populate the Hierarchical item incrementally. This will make the item a much more practical tool for your real world databases.

Designing the Hierarchy

The first step is to design the hierarchy and the Select statement that will populate the hierarchy. In the previous section I described how to create the hierarchy using a script containing several Select statements combined with Union set operators. This new approach requires this Select statement to be broken down into numerous individual Select statements or Record groups. Each individual level in the hierarchy requires its own statement. The Pre-form trigger launches the Grandfather level Select statement. The other Select statements are launched when specific nodes are selected or activated.

To illustrate, assume you design the hierarchy displayed in Figure 5.

This hierarchy has the following characteristics:

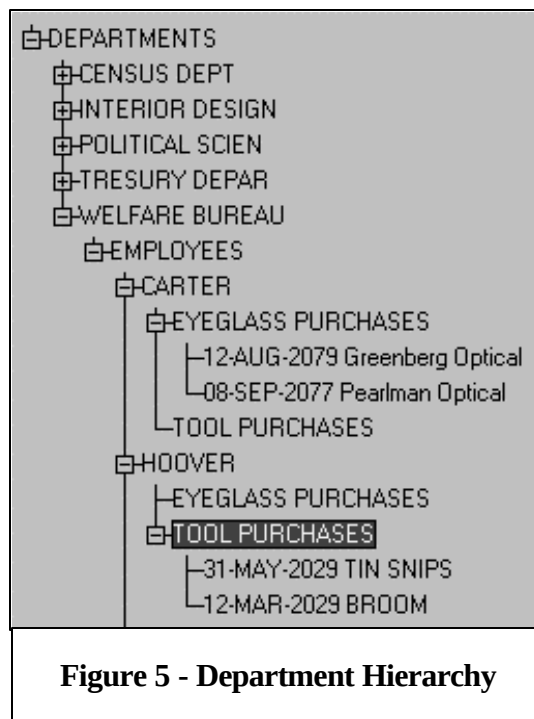


Figure 5 - Department Hierarchy

The top node is the text value "Departments".

Below the "Departments" node will be a node for each department displaying the name of the department.

Expanding the department node will cause a node containing the text value "Employees" to appear.

The Pre-form trigger will populate each of the above three nodes.

- Double-clicking the “Employees” node will activate the node and cause the department’s employees to be displayed under the node. If the department does not have employees, the node will be collapsed.
- Each of the child nodes under the Employees node will contain the name of an employee.
- Below each of the employee nodes will be two child nodes at the same level. The first will contain the text value “Eyeglass Purchases” and the second will be “Tool Purchases”.
- Double-clicking the “Eyeglass Purchases” node will active the node and causes the employee’s eyeglass purchases to be displayed. If the employee does not have eyeglass purchases, the node will be collapsed.
- Double-clicking the “Tool Purchases” node will active the node and causes the employee’s tool purchases to be displayed. If the employee does not have tool purchases, the node will be collapsed.

Views, Select Statements, and Record Groups

In my previous section, I suggested that a View be created for the Select statements used in the hierarchy. This technique allows the developer to sort the records properly. Following this technique means four Views will need to be created. A View is needed to retrieve the records for each of the events affecting the item (i.e. Launching the form, Double-clicking the Employees Node, Double-clicking the Eyeglass Purchases Node, Double-clicking the Tool Purchases node).

Listing 5 contains the View for the first Select statement that initially populates the Hierarchical item via the Pre-form trigger. This View loads the text value node “Departments”, the department name node, and the “Employees” text value node into the item.

Listing 5 – View Used in the Pre-form Trigger to Add the Departments

```
create or replace view dept_list as
select 1 state, 1 depth, 'DEPARTMENTS' displ_value, null icn, null pk, 'A' sort_value
from dual
union
select -1 state, 2, department_name, null, department, department
from department
union
select -1 state, 3, 'EMPLOYEES', null, department, department||'A'
from department;
```

End Listing

The next View will be used in the When-node-activated trigger to populate the employees of the selected department. This View is contained in Listing 6. You might notice two things:

This View is using the same depth values as in Listing 5. Forms will not put these nodes at the same level as the nodes created by the Listing 5 View. It will put the employee nodes one level lower than the selected node and the two expression nodes two levels lower.

Each of the records has the department number as the last field. This is necessary in order to select only the records from the View that match the selected department node.

You will see these same two features in each of the remaining Views:

Listing 6 – View Used in the When-node-activated trigger to Populate Employees

```
create or replace view employee_list as
select 1 state, 1 depth, last_name displ_value, null icn,
      payroll_number pk, last_name||first_name sort_value, fk_department
from employee
union
select -1, 2, 'EYEGGLASS PURCHASES', null,
      payroll_number, last_name||first_name||'Y', fk_department
from employee
union
select -1, 2, 'TOOL PURCHASES', null,
      payroll_number, last_name||first_name||'Z', fk_department
from employee;
```

Depth Values

Department
ID used for
selection.

End listing

The third View will be used in the When-node-activated trigger to populate the hierarchy with the eyeglass purchases for the selected employee. This View is illustrated in Listing 7.

Listing 7 – View Used in the When-node-activated trigger to Populate an Employee's Eyeglass Purchases

```
create or replace view eyeglass_list as
select 1 state, 1 depth, to_char(purchase_date,
      'DD-MON-YYYY')||' '||optician displ_value,
      null icn, fk_payroll_number pk,
      to_char(purchase_date,'YYYY-MON-DD') sort_value
from glasses;
```

End Listing

The fourth View will be used in the When-node-activated trigger to populate the hierarchy with the tool purchases for the selected employee. This View is illustrated in Listing 8.

Listing 8 – View Used in the When-node-activated trigger to Populate and Employee's Tool Purchases.

```
create or replace view tool_purchase_list as
select 1 state, 1 depth, to_char(purchase_date,
    'DD-MON-YYYY')||' '||tool_name displ_value,
    null icn, fk_payroll_number pk,
    to_char(purchase_date,'YYYY-MON-DD') sort_value
from emp_tools;
```

End Listing

Record Groups

After creating the Views, the form can be created. A Hierarchical item can be created along with a separate non-database block. Two different built-in sub programs are available to populate the Hierarchical item. These are:

- **Ftree.populate_tree** This built-in will be used to initially populate the tree. It will reside in the Pre-form trigger.
- **Ftree.Add_Tree_Data** This built-in will be used to populate individual branches. It will reside in the When-node-activated trigger.

Each of the built-in subprograms can use a Select statement or a Record group based upon the Views to retrieve the needed records. The Ftree.populate_tree trigger can use the Data Query property or the Record Group property of the Hierarchical item. The Add_tree_data built-in has a parameter for a text string that contains the Select statement or a text string using the name of a Record group contained within the form. **It is best if a Record group is used**

The reason it is better to use a Record group is because the Select statements used within the form require a Where clause and a bind variable as an argument. For example, the statement retrieving records from the Tool_purchase_list View will only want to retrieve records for the employee selected on the hierarchy. It is much easier to reference a bind variable contained on the form from a Record group than as a text literal in a parameter list.

Listing 9 illustrates the Select statements that comprise the four Record groups used in the example form. Notice that the latter three statement contain the bind variable :holder.val_char. This argument determines the records that will be selected by the latter three Record groups. It is a character type of item that is located on a non-database block on the form.

Listing 9 – Record Group Select Statements

/ Record Group (DEPT) Statement That Initially Populates the Hierarchy Using the Pre-form Trigger*/*

```
select state, depth, displ_value, ICN, pk
from dept_list
order by SORT_value
```

*/*Record Group (EMPS) Statement That Populates the Employees Using the When-node activated Trigger*/*

```
select state, depth, displ_value, icn, pk
from employee_list
where fk_department = :holder.val_char
order by sort_value
```

Bind Variables

*/*Record Group (GLASSES) Statement That Populates the Eyeglass Purchases Using the When-node-activated Trigger*/*

```
select state, depth, displ_value, icn, pk
from eyeglass_list
where pk = to_number(:holder.val_char)
order by sort_value desc
```

*/*Record Group (TOOLS) Statement That Populates the Tool Purchases Using the When-node-activated Trigger*/*

```
select state, depth, displ_value, icn, pk
from tool_purchase_list
where pk = to_number(:holder.val_char)
order by sort_value desc
```

End Listing

Triggers

After creating the Record groups, populate the Hierarchical item's Record Group property with "DEPT", the name of the initial Record group. A Pre-form trigger can then be created. This trigger is displayed in Listing 10. The parameter within the built-in is the name of the Hierarchical item and the Database block it resides on.

Listing 10 – Pre-form Trigger That Initially Populates the Hierarchical Item

```
ftree.populate_tree('block2.tree4');
```

End Listing

The next step is to create the When-node-activated trigger that adds children records to the activated node. This type of trigger is fired when a node item is double-clicked. A When-node-selected trigger or a When-node-expanded trigger could also be used. I prefer to use the When-node-selected, which fires when a node is clicked once, to execute a script that populates a corresponding data block. I also prefer to have the user double-click a node to add children records rather than expanding the node. Any of these triggers can be successfully used depending upon your design style.

Regardless of the type of trigger, the PL/SQL within the trigger is the same. The script (see Listing 11) must perform the following functions:

- Identify the node that was activated. This determines the Record group to be used.
- Populate the bind variable with the primary key for the selected node (i.e. department id, payroll_number).
- Ensure that the children for a node are only added once.
- Make sure the children nodes appear in an expanded manner rather than collapsed.

Listing 11 – When-node-activated Trigger Used to Populate The Hierarchical Item Incrementally

```

DECLARE
  htree    item;           --Holds the id of the hierarchy
  current_node  ftree.node; --Holds the id of the activated node
  executed_num number;      /*Holds the value indicating success
                             or failure after a Record group
                             is populated*/
  node_val  varchar2(100);  /*Holds the value of the activated node*/
BEGIN
  /*Step 1: Populate a variable with the hierarchy id.*/

  htree := find_item('block2.tree4');

  /*Step 2: Populate a variable with the id of the activated node*/

  current_node := :system.trigger_node;

  /*Step 3: Populate a variable with the value for the record
  displayed on the activated node. Records with a value
  have never had child records added. Records with a value
  of '' have had records added*/

  node_val := ftree.get_tree_node_property(htree, current_node,
                                           ftree.node_value);

  /*Step 4: Determine whether the current node has had child records
  added. If node_val = '', then records have been added
  and the trigger will perform on more work. If a different
  value exists, child records may be available for the. The
  tasks in the if-then structure should be executed*/

  if node_val != '' then

  /*Step 5: Place the value of the activated node record into a form
  item. This value will be used in the form record

```

```
group's Where clause to extract the proper records*/
```

```
:holder.val_char := Ftree.Get_Tree_Node_Property(htree, current_node, Ftree.node_value);
```

```
/*Step 6: Place the label of the activated node into a form item.  
This value is used to determine which Record group to  
use for extracting records*/
```

```
:holder.label := Ftree.Get_Tree_Node_Property(htree, current_node, Ftree.node_label);
```

```
/*Step 7: Populate the appropriate Record group and add the  
records to the hierarchy. The first if-then construct  
is used if the activated node is an "Employee" node.  
The second construct is used if the activated node is  
and "Eyeglass Purchases" node. The last construct is used  
if the activated node is a "Tool Purchases" node.*/
```

```
-- Beginning of the first if-then construct
```

```
if :holder.label = 'EMPLOYEES' then
```

```
/*Step 7a: Populate the Record group.*/
```

```
executed_num := populate_group('emps');
```

```
/*Step 7b: Add the contents of the Record group as children to  
the activated node*/
```

```
Ftree.Add_Tree_Data(htree, current_node, Ftree.parent_OFFSET, ftree.last_child,  
Ftree.record_group, 'emps');
```

```
-- Beginning of the second if-then construct
```

```
elseif :holder.label = 'EYEGLOSS PURCHASES' then  
executed_num := populate_group('glasses');  
Ftree.Add_Tree_Data(htree, current_node, Ftree.parent_OFFSET, ftree.last_child,  
Ftree.record_group, 'glasses');
```

```
-- Beginning of the third if-then construct
```

```
elseif :holder.label = 'TOOL PURCHASES' then  
executed_num := populate_group('tools');  
Ftree.Add_Tree_Data(htree, current_node, Ftree.parent_OFFSET, ftree.last_child,  
Ftree.record_group, 'tools');
```

end if;

```
/*Step 8: If child records are added to the hierarchy, this construct will  
change the activated node value to '. This prevents the node  
from being populated again. The construct also expands the newly  
populated node which causes the records to be displayed*/
```

```
if :holder.label in ('EMPLOYEES', 'EYEGLOSS PURCHASES', 'TOOL PURCHASES') then
```

```
/*Step 8a: Changes the activated node value to ''*/  
  
Ftree.Set_Tree_Node_Property(htree, current_node, Ftree.NODE_value, ' ');  
  
/*Step 8b: Expands the activated node displaying the newly added child nodes*/  
  
Ftree.Set_Tree_Node_Property(htree, current_node, Ftree.NODE_state,  
                             ftree.expanded_node);  
    end if;  
end if;  
end;
```

End Listing

At this point, you should be able to execute your form and the Hierarchical item will operate similar to the one shown in Figure 5. A When-node-selected trigger could be added to the form. This trigger is ideal for populating data blocks that correspond to the selected node.

Limiting the Display of the Initial Set of Nodes

In the previous example, the Select statement used in the Record group for the Pre-form trigger did not contain a Where clause. This will cause all of the grandparent records to appear in the Hierarchy. In some instances, you may not want all of these records to appear. If this layer consists of a large amount of data and you would like to limit the initial set of records, I would recommend the following:

- Delete the Pre-form trigger.
- Create another Record Group similar to the “DEPT” Record group. Add a Where clause containing a bind variable to this Record group. You may also consider using the Like operand in the where clause with a concatenated ‘%’ with the bind variable (i.e :holder.dept_val || ‘%’).
- Place the form item used as a bind variable on the form.
- Change the Pre-form trigger to a Key-execute-query trigger using the script contained in Listing 8.

Listing 8 – Key-execute Trigger Used to Initially Populate the Hierarchical Item

```
declare
  executed_num number;      /*Holds the value indicating success
                             or failure after a record group
                             is populated*/
begin
  If :holder.dept_val is null then
    executed_num := populate_group('dept');  -- Populates the record group
    Ftree.populate_tree ('block2.tree4');    --This subprogram uses the Record group
                                              --without the bind variable
  else
    /*Since the Form Item has a value, the Record group with the bind variable
    should be used. The following command changes the Record group of the
    hierarchical item.*/

    executed_num := populate_group('dept1'); -- Populates the record group

    -- Transfer the data from the record group to the hierarchical
    -- tree and cause it to display.

    ftree.set_tree_property ('block2.tree4', ftree.record_group, 'dept1');
  end if;
end;
```

End Listing

This last technique will change the way your form operates. The Hierarchical item will only be populated when the Key-exeqry (F8) button is pressed. If the user want to limit the search, a value can be placed in the :holder.dept_val form item. This will limit the number of records that will be loaded. If the user leaves this item blank, the Hierarchical item will display all of the records.

Using this technique along with the ones discussed in the previous section will allow you to develop exceptional applications that allow your users to easily move around the data in their database.

Author's Note

The scripts to install the practice database, the scripts for the Views/Record Groups, and a working form based upon this database are available on my web site (www.oracle-trainer.com)

