

ADF Code Corner

032. How-to create a tree table from a single View Object and how to access selected rows

ORACLE®
CODE CORNER



twitter.com/adfcodecorner

Abstract:

A frequent question about ADF bound ADF Faces views is how to show a hierarchical structure of data based on a single recursive view object. A table structure that is a good candidate for such a view object is the employees table in the Oracle HR sample schema. The employees table has a self reference defined between the MANAGER_ID column and the EMPLOYEE_ID column to link managers to their directs. Beside of answering the question of how to create a tree table structure based on such a data model, a second question we answer in this article is how to access the selected row data in the tree table and how to work with the underlying binding layer.

Author:

Frank Nimphius, Oracle Corporation
twitter.com/fnimphiu
05-MAY-2010

Oracle ADF Code Corner is a loose blog-style series of how-to documents that provide solutions to real world coding problems.

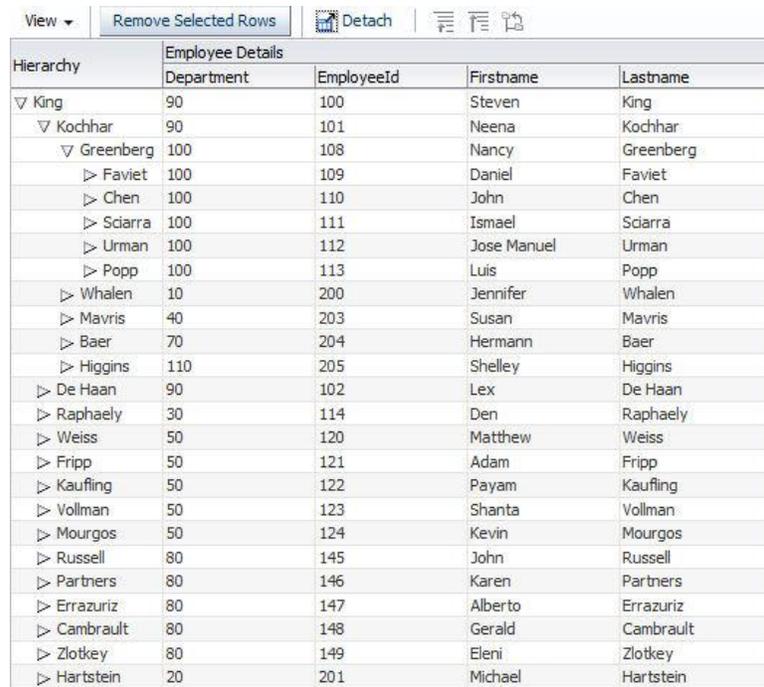
Disclaimer: All samples are provided as is with no guarantee for future upgrades or error correction. No support can be given through Oracle customer support.

Please post questions or report problems related to the samples in this series on the OTN forum for Oracle JDeveloper: <http://forums.oracle.com/forums/forum.jspa?forumID=83>

Introduction

The image below shows a tree table that is based on a single view object that reads data from the employees table in the Oracle HR sample schema. The tree table is configured so that users can select one or multiple rows to delete them by pressing the "Remove Selected Rows" command button.

The delete operation is an example of how to access the tree table data and the underlying ADF binding layer and can be easily modified by you to perform data manipulation on the selected rows instead.



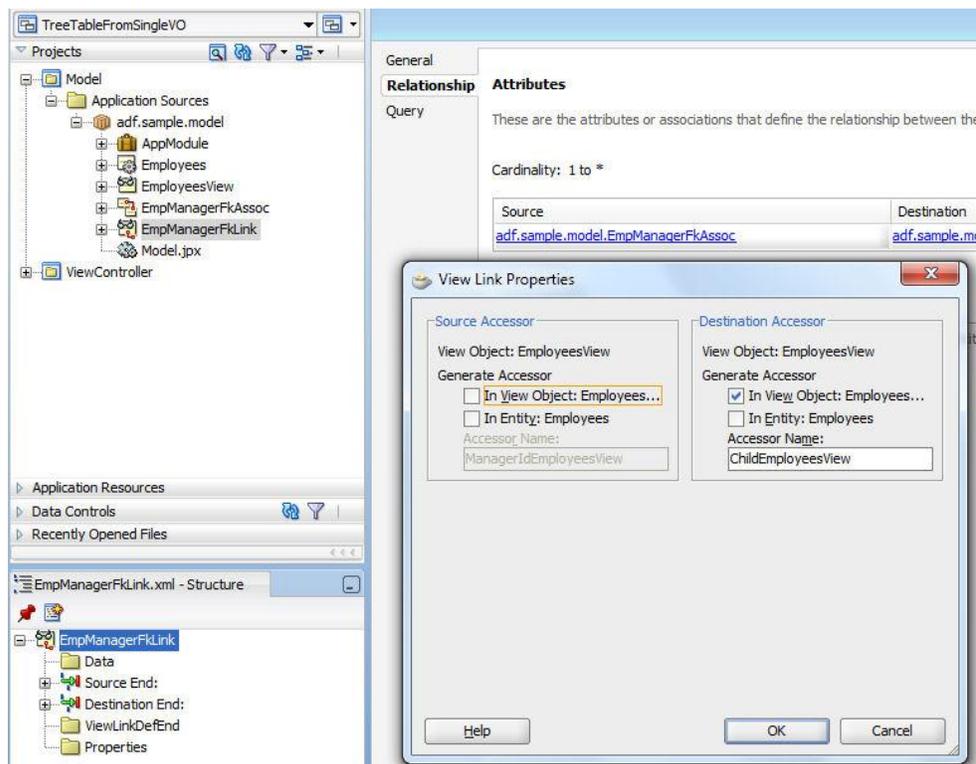
Hierarchy	Department	EmployeeId	Firstname	Lastname
King	90	100	Steven	King
Kochhar	90	101	Neena	Kochhar
Greenberg	100	108	Nancy	Greenberg
Faviet	100	109	Daniel	Faviet
Chen	100	110	John	Chen
Sciarra	100	111	Ismael	Sciarra
Urman	100	112	Jose Manuel	Urman
Popp	100	113	Luis	Popp
Whalen	10	200	Jennifer	Whalen
Mavris	40	203	Susan	Mavris
Baer	70	204	Hermann	Baer
Higgins	110	205	Shelley	Higgins
De Haan	90	102	Lex	De Haan
Raphaely	30	114	Den	Raphaely
Weiss	50	120	Matthew	Weiss
Fripp	50	121	Adam	Fripp
Kaufing	50	122	Payam	Kaufing
Vollman	50	123	Shanta	Vollman
Mourgos	50	124	Kevin	Mourgos
Russell	80	145	John	Russell
Partners	80	146	Karen	Partners
Errazuriz	80	147	Alberto	Errazuriz
Cambraut	80	148	Gerald	Cambraut
Zlotkey	80	149	Eleni	Zlotkey
Hartstein	20	201	Michael	Hartstein

Creating the ADF Business Component model

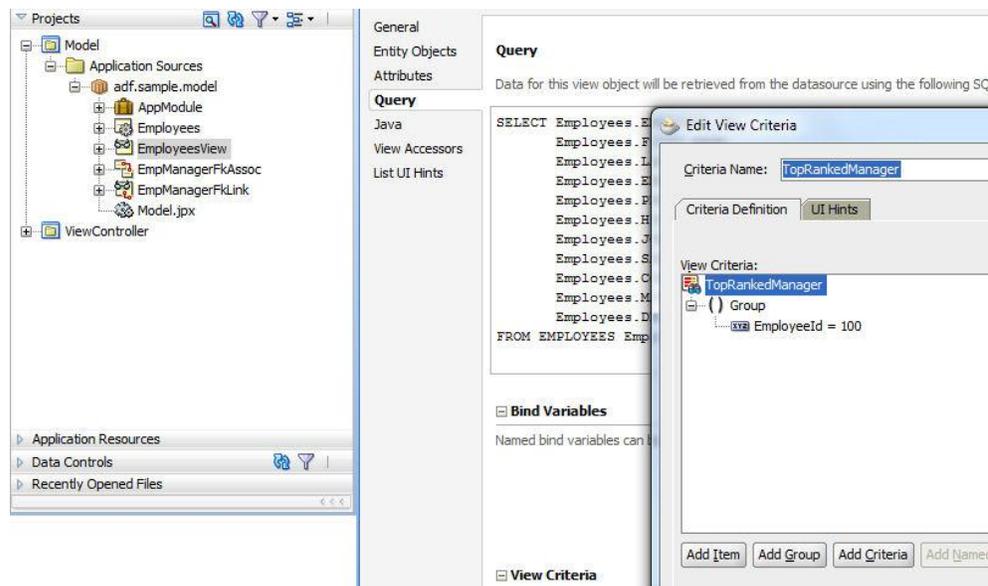
There is nothing special about creating the ADF Business Components model and you start with the "Create ADF Business Components from Table" wizard in the Oracle JDeveloper **NEW** Gallery. From the available tables, choose the employees table and finish the wizard so that an entity object and a view object are created.

The image below shows the structure of such a model project and you can see that the "EmpManager" entity association and view link are automatically created based on the database constraint that is found.

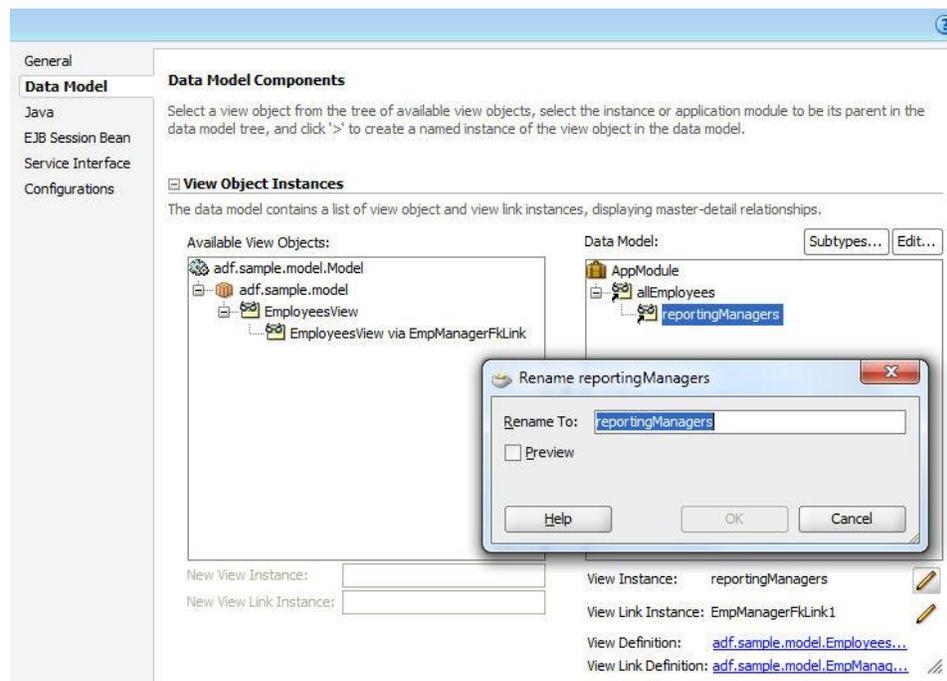
The changes applied to the model in this example are i) the non-default name of the view link accessor as shown in the image below and ii) the creation of a view criteria that queries the top level row, similar to what CONNECT BY PRIOR ... START WITH would do in SQL. The image below shows how to change the view accessor name for the destination, the recursive child reference. This becomes handy later when building the tree table to identify the child node rule.



You create named view criteria from the view object editor (its located in the Query category). The view criteria in this example is called "TopRankedManager" and defines a where clause that only shows the employee with the employee_id 100, which in the employees table happens to be Steven King, the president. To make this view criteria usage a more realistic use case for a production environment you may decide to filter the table rows by the job_id, in case there are more or changing presidents (retirement is a good reason for a president to leave) in an organization. Also use a bind variable to dynamically set the filter criteria value, which allows you to further filter the returned row set based on the context the tree table is shown in.



When building the ADF Business Component model, it is good practice to change the default name of the view object instance to a name closer related to what the view object instance is used for. I like to see this naming change as a contract the business service developer enforces between him or her and the application developer. The more meaningful names you can come up with the easier it will be for the application developer to stay away from confusion. This is especially important when view object instances are filtered by a view criteria, thus permanently return a different result set than the view object definition has defined.

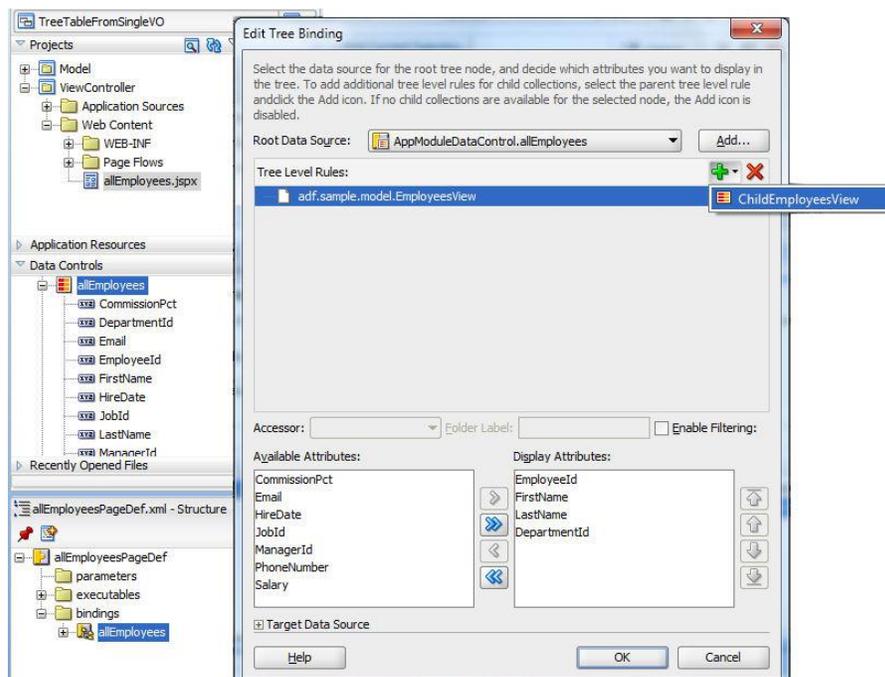


To filter a view object instance with a named view criteria, select the View Object instance (for example the allEmployees instance) and press the Edit button. The opened dialog shows you all the view criteria that are defined for the view object and you select the one to permanently apply to the view object instance. Note that this is applied to the view object instance, not the view object definition itself. According to my suggestion, you would then rename the instance from "allEmployees" to for example "topRankedEmployees" for the use case in which the view object only returns records of employees in the job role of "president". To rename a view object instance, select the view object entry on the Data Mode side of the shuttle control and press the pencil icon next to the "View Instance" label.

Building the Tree Table

The ADF Business Component data model is displayed in the Oracle JDeveloper Data Controls panel from where you drag the "allEmployees" View Object instance (or the "topRankedEmployees" instance if you followed my renaming hint) as a tree table to the page. You then select the attributes you want to see in the tree table from the list of "Available Attributes". Next you press the "green plus" icon to create the tree level rule for the child nodes. Note that the context menu shows an entry "ChildEmployeeView", which is the name specified earlier as the ViewLink accessor name. The attributes selected for the child tree level are the same as the attributes chosen for the root node.

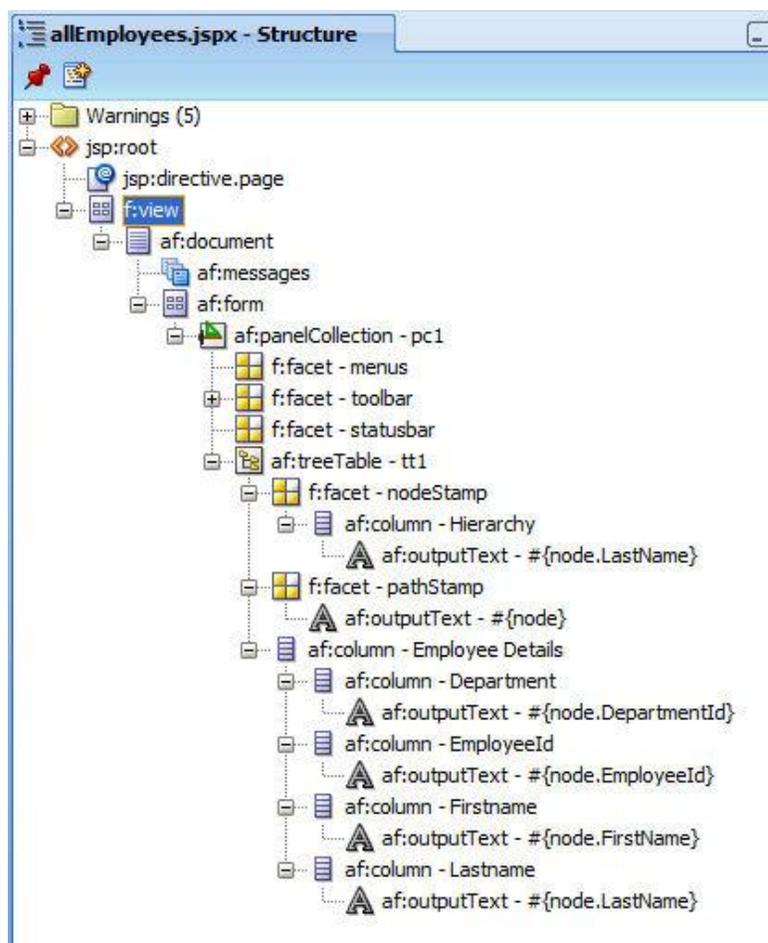
Any change in the selected attributes for the "ChildEmployeesView" are also reflected in the root node because both are based on the same view object. When you created the "ChildEmployeeView" tree node rule, then you see that the name "ChildEmployeeView" is shown in brackets next to "Tree Level Rules" entry, which like for the tree root node is (EmployeesView).



By default, the tree table has two facets , the **nodeStamp** facet and the **pathStamp** facet. The nodeStamp facet is the one that renders the tree table. However, to create a tree table, additional columns need to be built as shown in the image below.

When rendering the tree table, the **node** variable is used to iterate over the individual rows. The node variable is EL accessible and provides access to the attributes like FirstName, LastName etc. Only those attributes that were selected when defining the ADF tree binding can be accessed at runtime.

Note: The pathStamp facet is used to navigate within a tree table, for example to select a tree branch and temporarily make it the top level node. If the current root node is not the top level node in a tree, then the path stamp can be used from the context menu to reset the original tree view. A full discussion of using the path stamp is beyond the scope of this paper



Shown in the image above, the columns are created outside of the nodeStamp and pathStamp facets. They contain a component to print the cell content - an af:outputText in the example above- referencing the node variable.

Your backstage access to the ADF binding

As in many articles published on ADF Code Corner, we encourage you to not hard code names of PageDef entries in your managed beans, unless absolutely required. As you can see in the Java method referenced from the "Remove Selected Rows" button shown in the image on top of this article, the ADF binding layer can be accessed from the ADF Faces component model so that the component and the knowledge about the component becomes a generic "backstage access" to the ADF binding layer. All you need to know about the managed bean access to the binding layer and the tree table row data is added as code comments below.

```
import java.util.Iterator;
import java.util.List;

import javax.faces.event.ActionEvent;

import oracle.adf.view.rich.component.rich.data.RichTreeTable;
import oracle.adf.view.rich.context.AdfFacesContext;

import oracle.jbo.Row;
import oracle.jbo.uicli.binding.JUCtrlHierNodeBinding;

import org.apache.myfaces.trinidad.model.RowKeySet;

public class AllEmployeesBean {
    private RichTreeTable treeTable1;

    public AllEmployeesBean() {
    }

    public void onPrintSelection(ActionEvent actionEvent) {
        //get treeTable component instance from this bean. This instance is
        // created when configuring a component EL binding to the managed
        //bean using the "binding" property of the tree table component
        RichTreeTable treeTable = this.getTreeTable1();
        //get all selected row keys
        RowKeySet rks = treeTable.getSelectedRowKeys();
        Iterator keys = rks.iterator();
        //if the treeTable is configured to support single selection or
        //multiple selection doesn't matter as the routine below works with
        //all cases
        while(keys.hasNext()){
            //the treeTable path is defined as a List of keys
            List key = (List) keys.next();
            //set the treeTable current row to the row defined by the key
            treeTable.setRowKey(key);
            //Using ADF, a treeTable row is represented by the
            //JUCtrlHierNodeBinding class, which is an ADF Faces binding
            //object that wraps the row of the ADF model
            JUCtrlHierNodeBinding node =
                (JUCtrlHierNodeBinding) treeTable.getRowData();
            //The row in the model, when using ADF BC, is of type
            //oracle.jbo.Row
            Row rw = node.getRow();
```

```
//Lets read and print from the row ...

System.out.println("Selected: "+rw.getAttribute("FirstName")+
                  "+rw.getAttribute("LastName"));

// ... before deleting it
rw.remove();

//Note: With the handle to the JUCtrlHierNodeBinding, you could
//access the binding layer, for example the JUCtrlHierBinding
//object of the tree binding definition or DCIteratorBinding of
//the iterator binding. Its your backstage access to the runtime
//object representing the PageDef file
}
//ppr the tree
AdfFacesContext.getCurrentInstance().addPartialTarget(treeTable);

//Ahhh, yes - if you don't commit the data removal then the data
//is not permanently deleted but just from the ADF collection
//displayed in the treeTable. To commit the change, drag and drop the
//Commit operation from the data control as a button and press it
//after deleting the rows, or call the action from here - as
//mentioned you have your backstage access.
}

public void setTreeTable1(RichTreeTable treeTable1) {
    this.treeTable1 = treeTable1;
}

public RichTreeTable getTreeTable1() {
    return treeTable1;
}
}
```

Download Sample

You can **download the sample workspace from ADF Code Corner:**

<http://www.oracle.com/technetwork/developer-tools/adf/learnmore/index-101235.html>

The sample is written with JDeveloper 11g R1 PS2 but is supposed to work with any previous release of JDeveloper 11g. Before running the sample, change the database connection used by the ADF Business Component model to point to a database of yours that has the HR schema enabled.

RELATED DOCUMENTATION

<input type="checkbox"/>	Tree Table tag - http://download.oracle.com/docs/cd/E15523_01/apirefs.1111/e12419/tagdoc/af_treeTable.html
<input type="checkbox"/>	PanelCollection tag - http://download.oracle.com/docs/cd/E15523_01/apirefs.1111/e12419/tagdoc/af_panelCollectio

	n.html
<input type="checkbox"/>	Oracle Fusion Developer Guide – McGraw Hill Oracle Press, Frank Nimphius, Lynn Munsinger http://www.mhprofessional.com/product.php?cat=112&isbn=0071622543