ADF Code Corner

75. How-to select multiple parent table rows and synchronize a detail table with the combined result

twitter.com/adfcodecorner

Abstract:

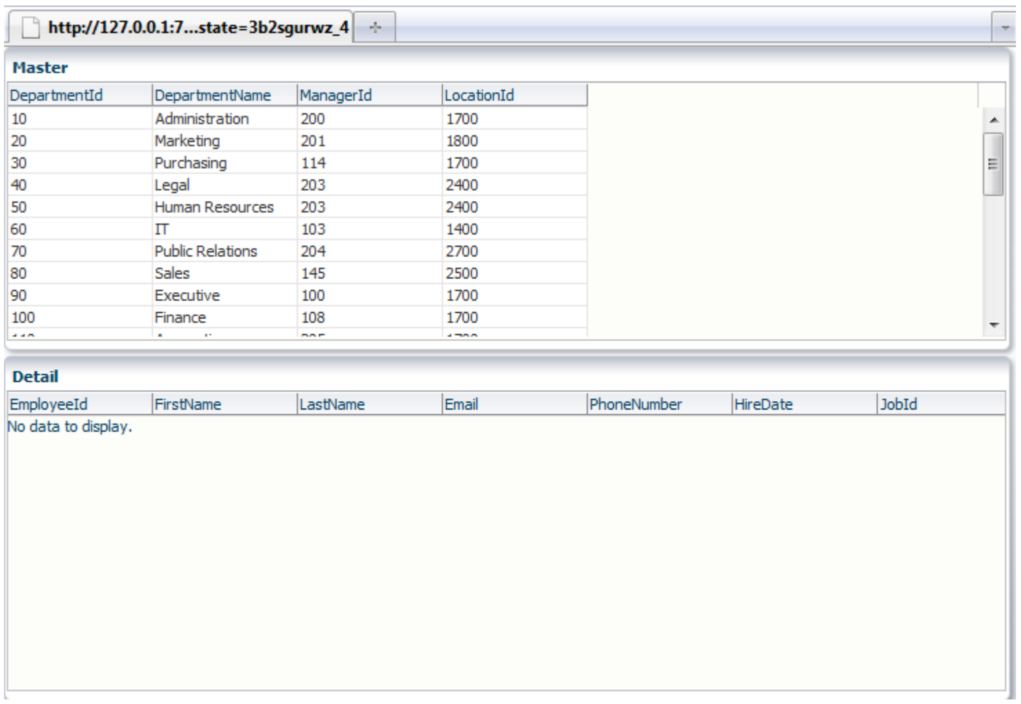
Using dependent View Objects in ADF Business Components makes it easy to create a master-detail behavior between two ADF bound ADF Faces tables. However, things are different if the parent table allows multiple row selection and the detail table is supposed to show the combined details for the selected rows. The use case of multiple parent rows is covered in this article using bind variables.

Author: Frank Nimphius, Oracle Corporation
twitter.com/fnimphiu
28-FEB-2011

Introduction

Working with multi row select tables already is an advanced topic in Oracle ADF. Having such a table triggering a detail table to show combined detail records is even more advanced, though a requirement frequently posted to the Oracle JDeveloper forum on OTN. The images below show the runtime behavior of the sample epxlained in this article. The Oracle JDeveloper 11.1.1.4 sample workspace can be downloaded from the ADF Code Corner website.

When the table initially renders, no row is selected in the parent table and therefore the detail table does not show data.



Master

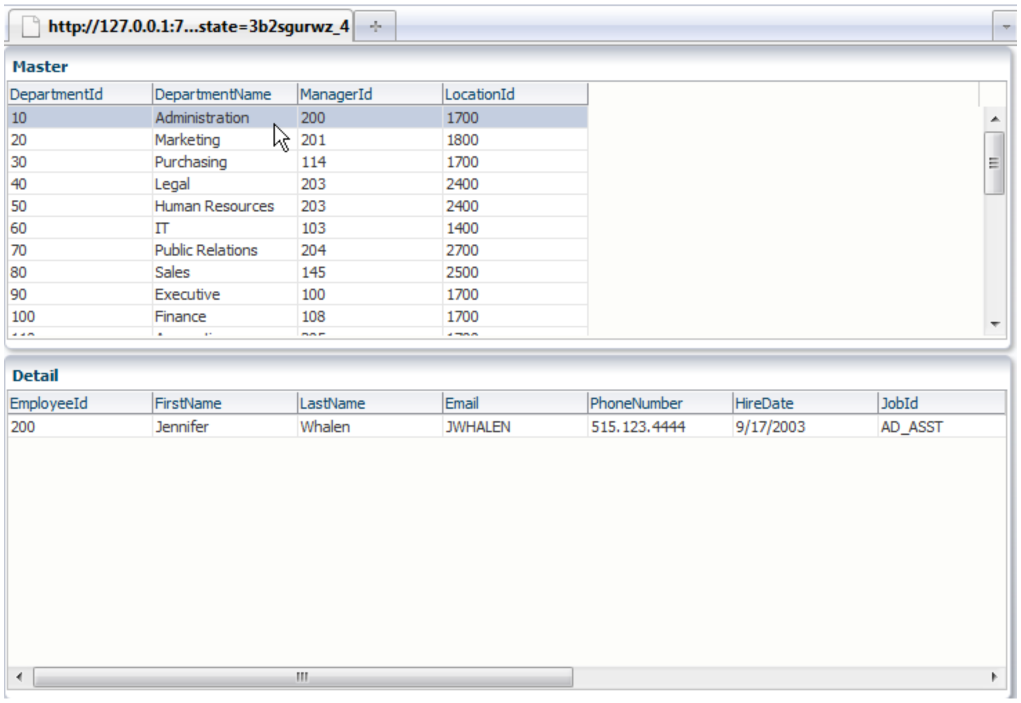
DepartmentId	DepartmentName	ManagerId	LocationId
10	Administration	200	1700
20	Marketing	201	1800
30	Purchasing	114	1700
40	Legal	203	2400
50	Human Resources	203	2400
60	IT	103	1400
70	Public Relations	204	2700
80	Sales	145	2500
90	Executive	100	1700
100	Finance	108	1700

Detail

EmployeeId	FirstName	LastName	Email	PhoneNumber	HireDate	JobId

No data to display.

When the user selects a parent row, the detail table is refreshed and shows the detail data for the selected parent row.



http://127.0.0.1:7...state=3b2sgurwz_4

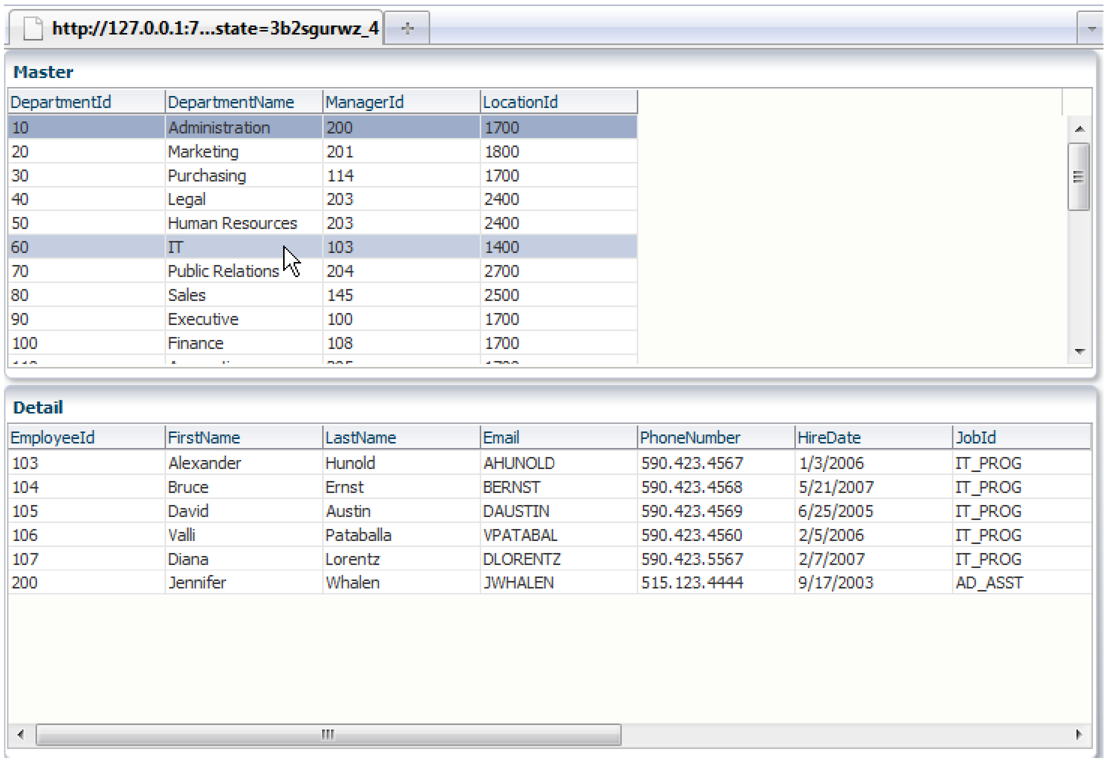
Master

DepartmentId	DepartmentName	ManagerId	LocationId
10	Administration	200	1700
20	Marketing	201	1800
30	Purchasing	114	1700
40	Legal	203	2400
50	Human Resources	203	2400
60	IT	103	1400
70	Public Relations	204	2700
80	Sales	145	2500
90	Executive	100	1700
100	Finance	108	1700

Detail

EmployeeId	FirstName	LastName	Email	PhoneNumber	HireDate	JobId
200	Jennifer	Whalen	JWHALEN	515.123.4444	9/17/2003	AD_ASST

Selecting another row by pressing the ctrl-key and clicking into another table row also refreshes the detail table, this time however showing the combined records for both selected parent rows.



http://127.0.0.1:7...state=3b2sgurwz_4

Master

DepartmentId	DepartmentName	ManagerId	LocationId
10	Administration	200	1700
20	Marketing	201	1800
30	Purchasing	114	1700
40	Legal	203	2400
50	Human Resources	203	2400
60	IT	103	1400
70	Public Relations	204	2700
80	Sales	145	2500
90	Executive	100	1700
100	Finance	108	1700

Detail

EmployeeId	FirstName	LastName	Email	PhoneNumber	HireDate	JobId
103	Alexander	Hunold	AHUNOLD	590.423.4567	1/3/2006	IT_PROG
104	Bruce	Ernst	BERNST	590.423.4568	5/21/2007	IT_PROG
105	David	Austin	DAUSTIN	590.423.4569	6/25/2005	IT_PROG
106	Valli	Pataballa	VPATABAL	590.423.4560	2/5/2006	IT_PROG
107	Diana	Lorentz	DLORENTZ	590.423.5567	2/7/2007	IT_PROG
200	Jennifer	Whalen	JWHALEN	515.123.4444	9/17/2003	AD_ASST

The easiest way to implement this behavior is to dynamically create and depend a where clause to the View Object querying the detail data. This however is only the second best option from a database performance perspective and instead bind variables shall be used.

Building the Model

Adding bind variables directly to the SQL *IN* operation of a View Object query is not possible. Instead you need to create a PLSQL function that you pass the bind variable to and that parses the variable's value to a comma separated list of arguments. Just passing a comma separated list of values in the bind variable causes a number format exception as the bind variable value cannot be converted into a number.

I found the PLSQL function below in a, "*Ask Tom*" forum post by Tom Kyte and used it as Tom provided it there: http://asktom.oracle.com/pls/apex/f?p=100:1:0

The

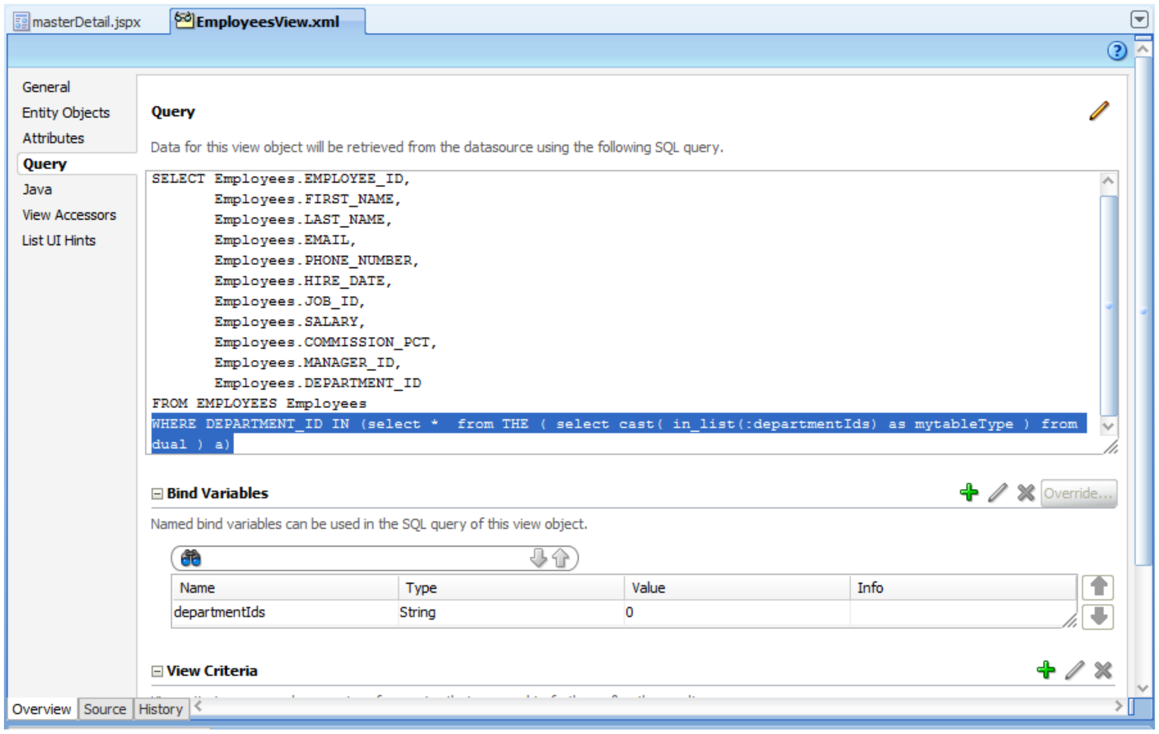
```
CREATE OR REPLACE type myTableType
AS
  TABLE OF NUMBER;
/

CREATE OR REPLACE FUNCTION in_list(p_string IN VARCHAR2 )
  RETURN myTableType

AS
  l_string LONG DEFAULT p_string || ',';
  l_data myTableType := myTableType();
  n NUMBER;
BEGIN
  LOOP
    EXIT
  WHEN l_string IS NULL;
    n            := instr( l_string, ',' );
    l_data.extend;
    l_data(l_data.count) := ltrim( rtrim( SUBSTR(l_string, 1, n-1 )));
    l_string             := SUBSTR( l_string, n                +1 );
  END LOOP;
  RETURN l_data;
END;
/
```

The stored procedure takes a string argument and returns a parsed argument chain that separates its members by a comma. The PLSQL function is provided with the Sample download on ADF Code Corner and needs to be installed into the HR schema.

With this pre-requisite in place, the query for the View Object showing the detail rows can be constructed with a *WHERE* clause containing the function call as

```
DEPARTMENT_ID IN (select * from THE(select cast(
in_list(:departmentIds) as mytableType ) from dual ) a)
```

departmentIds is of type String and defined as a bind variable for the *EmployeesView* View Object. Bind variables are added to the query with a leading colon – ':' – but in addition need to be explicitly created by pressing the green plus icon next to the *Bind Variables* section. The *departmentsIds* bind variable definition used in the sample is shown in the image below.



```
SELECT Employees.EMPLOYEE_ID,
       Employees.FIRST_NAME,
       Employees.LAST_NAME,
       Employees.EMAIL,
       Employees.PHONE_NUMBER,
       Employees.HIRE_DATE,
       Employees.JOB_ID,
       Employees.SALARY,
       Employees.COMMISSION_PCT,
       Employees.MANAGER_ID,
       Employees.DEPARTMENT_ID
FROM EMPLOYEES Employees
WHERE DEPARTMENT_ID IN (select *  from THE ( select cast( in_list(:departmentIds) as mytableType ) from
dual ) a)
```

Bind Variables
Override...
Named bind variables can be used in the SQL query of this view object.

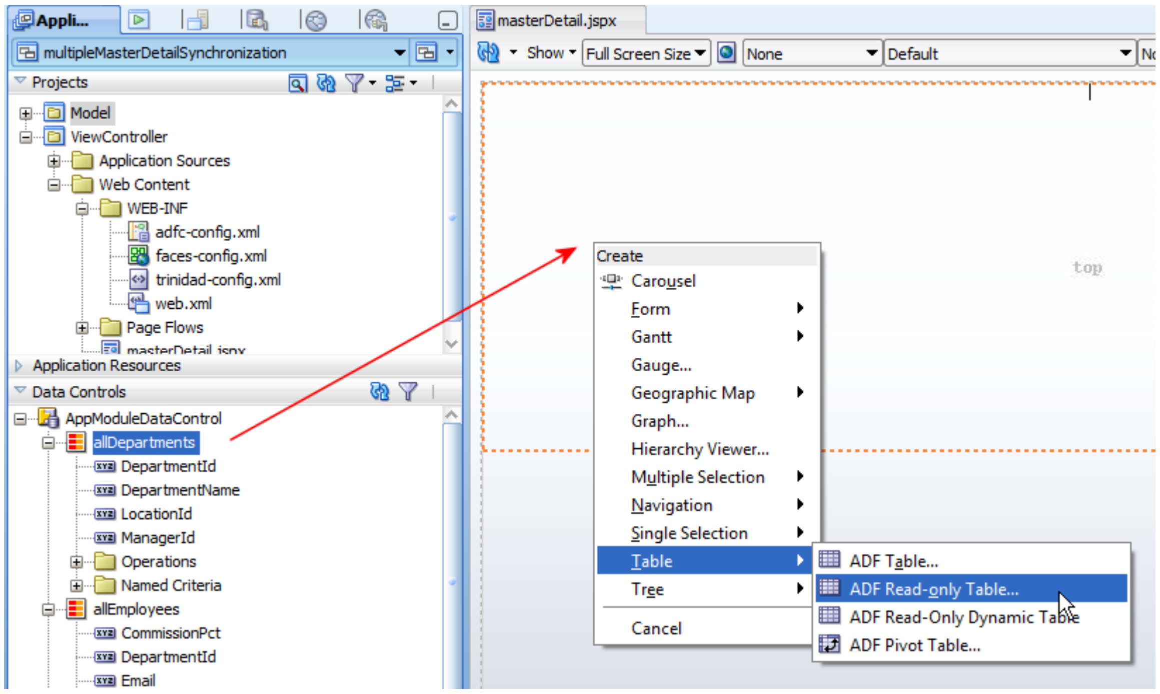
Name	Type	Value	Info
departmentIds	String	0	

Note the *EmployeesView* View Object instance later used in the View layer is not dependent from the *DepartmentsView* parent View Object, meaning that we don't synchronize the parent View Object multiple row selection with the detail employee data through a View Links.

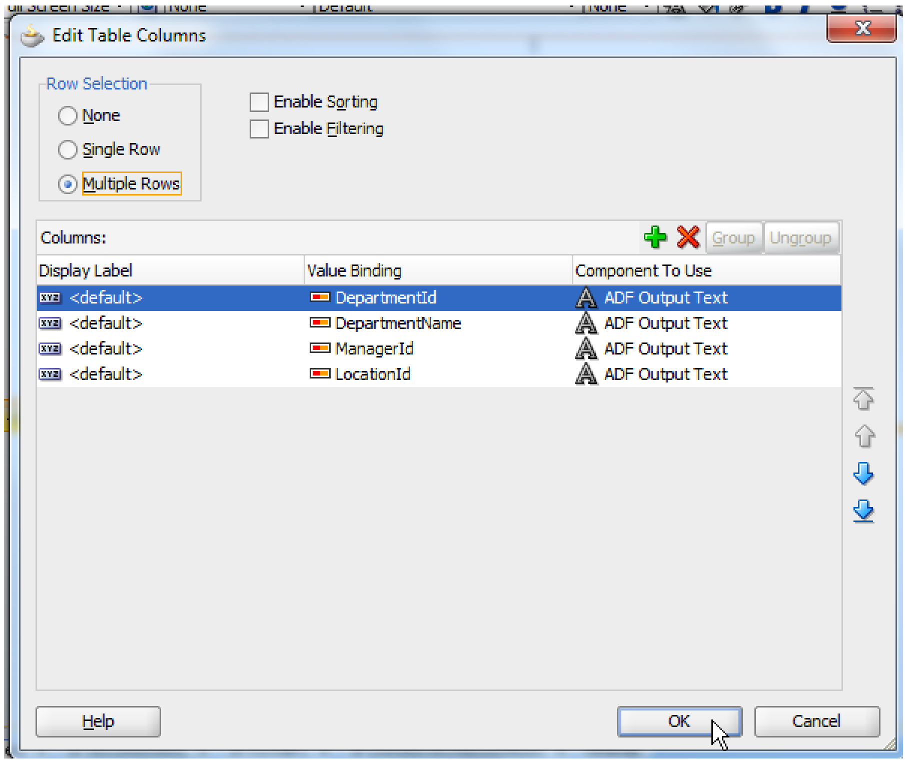
Building the View

With the ADDF Business Components model in place, it is time to build the View Controller layer. If you started your application development using the Fusion Application template, then a View Controller project already exists. Otherwise create a new project and configure its technology scope for ADF Faces and Oracle Page Flow.

First, you create a page using a two column *Quick Layout*. You then drag the *DepartmentsView* View Object from the Data Controls panel and drop it as a table to the page. Note that in the example, I renamed the View Object instance from *DepartmentsView1* to *allDepartments*. You can do the same in the Application Module editor, selecting the *Data Model* panel. Right mouse click onto the View Object instance to rename and choose *Rename* from the context menu.



In the table configuration dialog, select *Multiple Row* and press Ok. Optionally you may choose the sort and filter option as well, if your use case requires users to be able to sort and filter query data at runtime.


Edit Table Columns

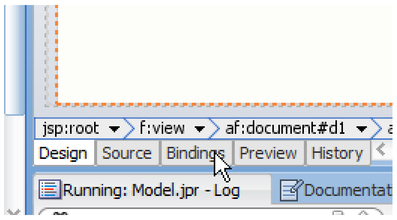
Row Selection: None / Single Row / [✓ Multiple Rows]

Enable Sorting [ ]
Enable Filtering [ ]

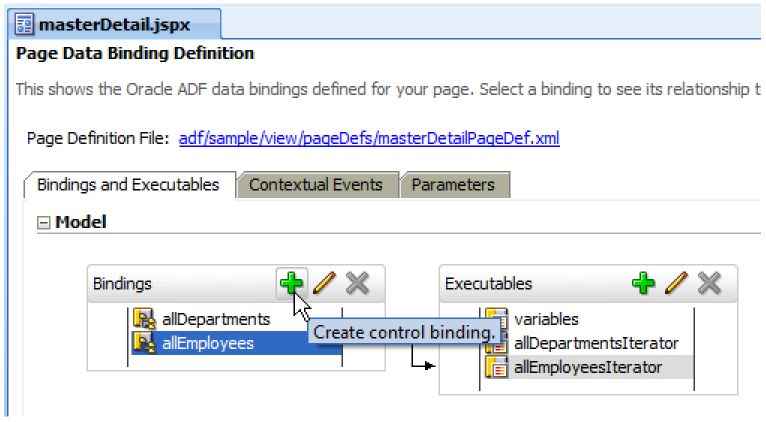
Columns:

Display Label	Value Binding	Component To Use
<default>	DepartmentId	ADF Output Text
<default>	DepartmentName	ADF Output Text
<default>	ManagerId	ADF Output Text
<default>	LocationId	ADF Output Text

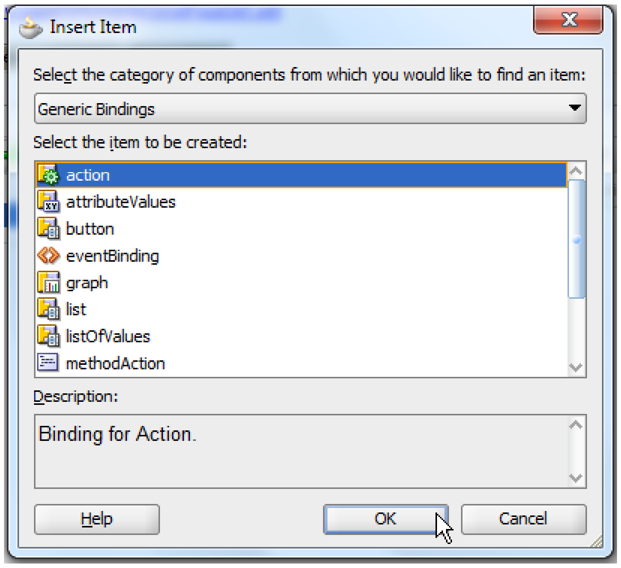
Next step is to switch to the ADF binding editor. For this you click onto the *Bindings* tab at the bottom of the visual page editor.



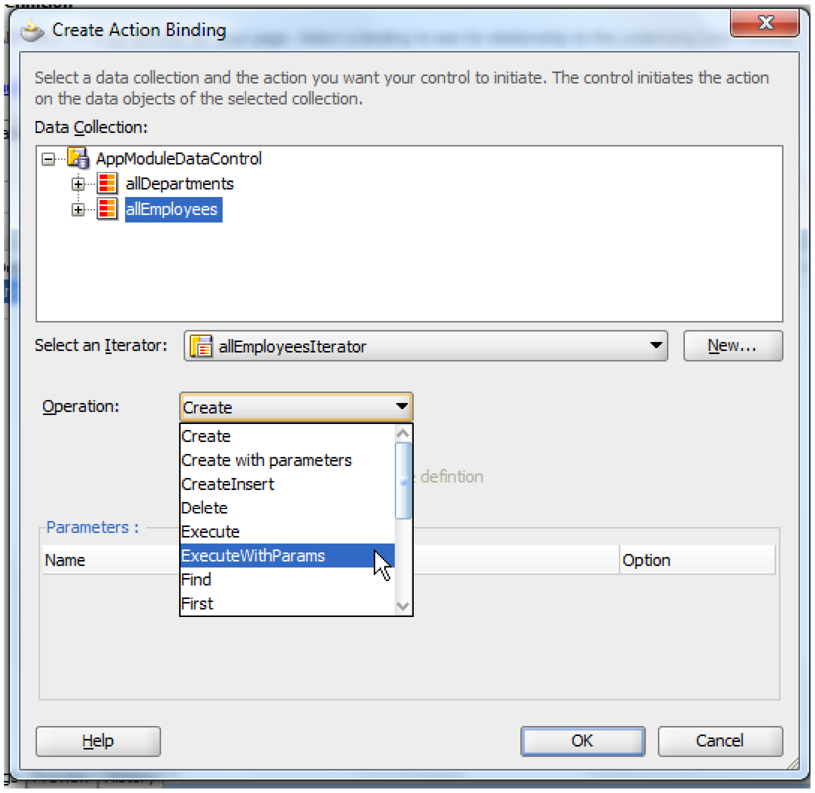
In the editor, you create an action binding for the *ExecuteWithParams* operation of the *EmployeesView* View Object. This allows you to pass the bind variable values and execute the View Object query.


masterDetail.jspx
Page Data Binding Definition
This shows the Oracle ADF data bindings defined for your page. Select a binding to see its relationship t
Page Definition File: adf/sample/view/pageDefs/masterDetailPageDef.xml
Bindings and Executables
Contextual Events
Parameters
Model
Bindings
allDepartments
allEmployees
Create control binding.
Executables
variables
allDepartmentsIterator
allEmployeesIterator

Pressing the green plus icon, the *Insert Item* dialog opens.



Choose *action* and click ok to create the metadata in the PageDef binding file. In the *Create Action Binding* dialog, select the *allEmployees* View Object instance (also renamed from *EmployeesView1*) and choose *ExecuteWithParams*.


Create Action Binding

Select a data collection and the action you want your control to initiate. The control initiates the action on the data objects of the selected collection.

Data Collection:

- AppModuleDataControl
  - allDepartments
  - allEmployees

Select an Iterator: allEmployeesIterator

Operation: Create

You don't need to provide a value to the *ExecuteWithParams* argument, which is the *departmentIds* bind variable name. This value will be added programmatically from a managed bean in response to a row selection in the table.

To create the **detail table**, drag the *allEmployees* View Object from the Data Control panel and drop it as a read-only table onto the ADF Faces page. The configuration of the table doesn't matter for this example as it is only used for displaying detail data.

Select the employee table in the Structure Window and open the Property Inspector [ctrl+shift+I]. Click the arrow icon at the end of the *Binding* property to create a managed bean reference for the table. This reference allows to partial refresh the table in response to a parent row selection in the *allDepartments* table.

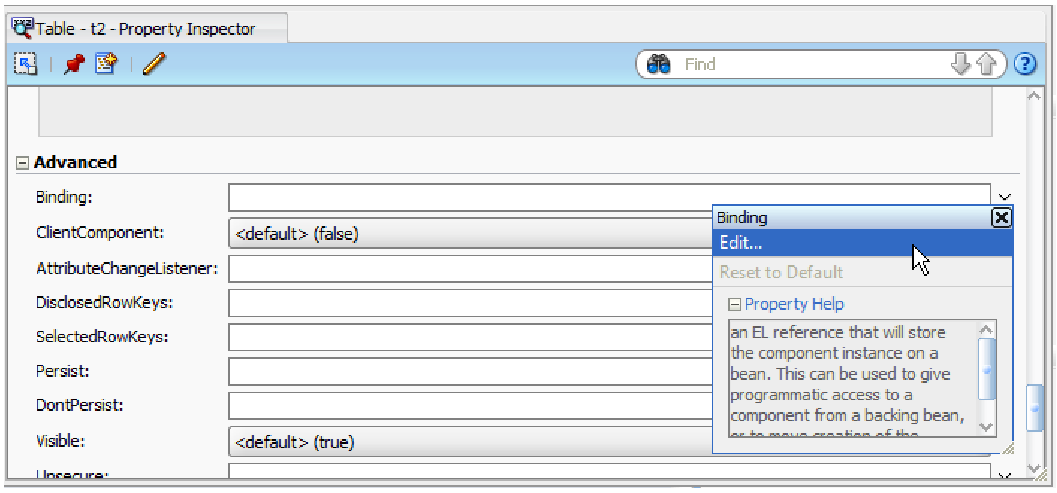
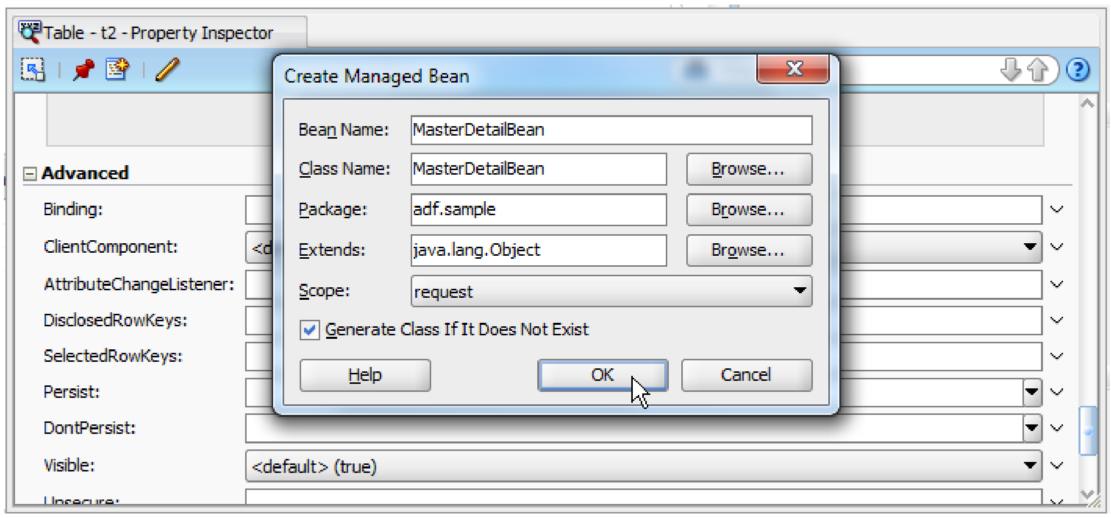


Table - t2 - Property Inspector

Advanced

Binding:	
ClientComponent:	<default> (false)
AttributeChangeListener:	
DisclosedRowKeys:	
SelectedRowKeys:	
Persist:	
DontPersist:	
Visible:	<default> (true)

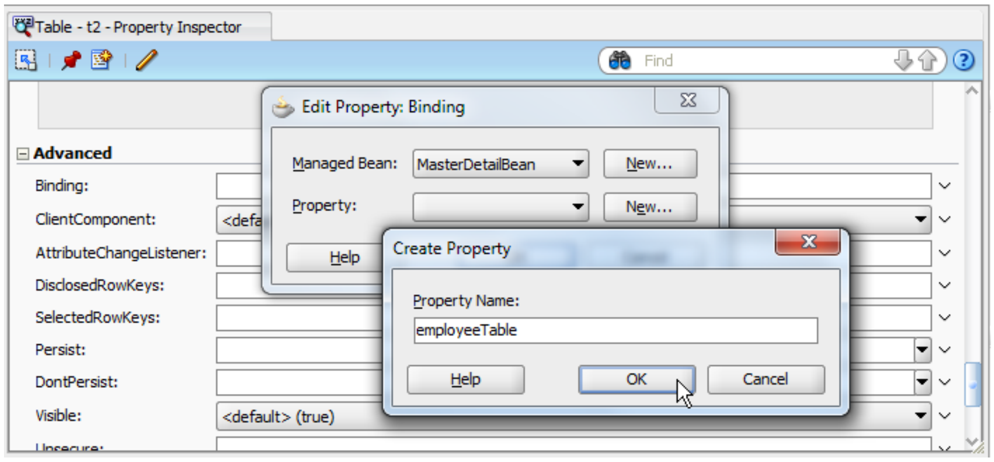
Property Help: an EL reference that will store the component instance on a bean. This can be used to give programmatic access to a component from a backing bean, Create a managed bean, which in this case is created in the `adfc-config.xml` file.



Create Managed Bean

Bean Name: MasterDetailBean
Class Name: MasterDetailBean
Package: adf.sample
Extends: java.lang.Object
Scope: request
[✓] Generate Class If It Does Not Exist

Define a name for the property referencing the table. This property will be created as a private variable in the managed bean with associated setter/getter methods created.



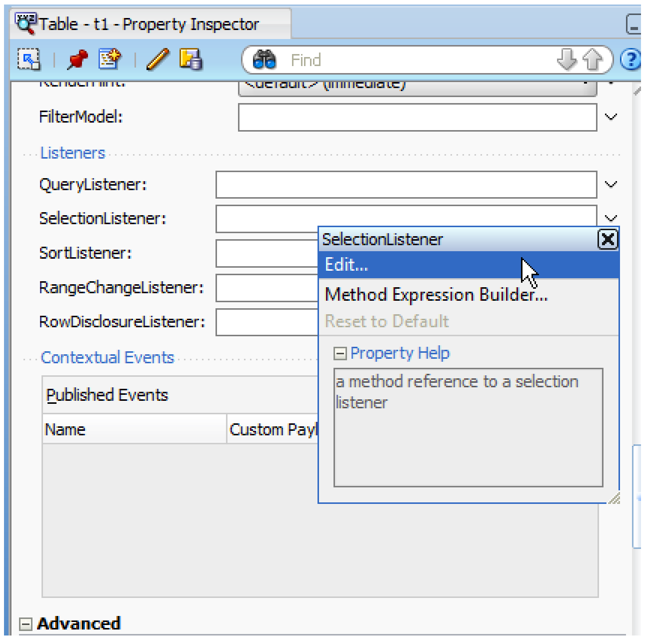
Edit Property: Binding

Managed Bean: MasterDetailBean
Property:

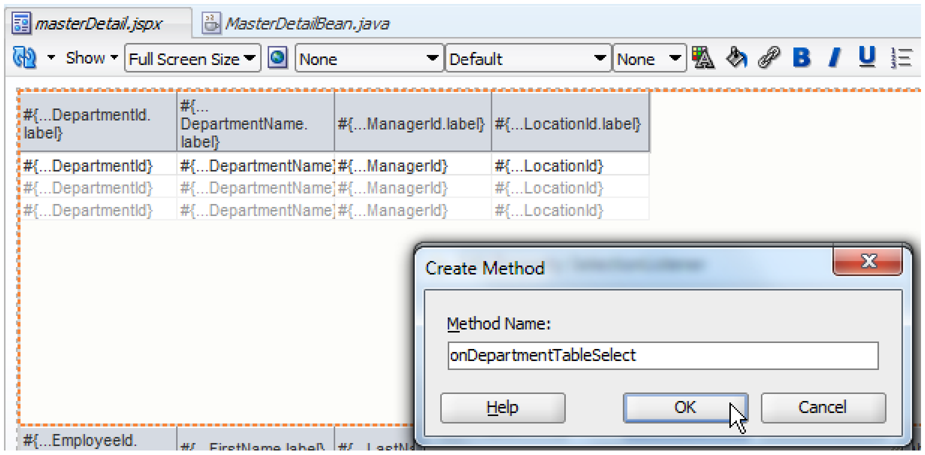
Create Property

Property Name: employeeTable

Select the **allDepartments** table in the Structure Window and browse for the *Selection Listener* property. Press the arrow icon to the right of the field to define a selection handler method. Any existing value of the *Select Listener* property can be ignored as it is not needed for this use case. Click the *Edit* option in the opened context menu.



In the opened dialog, define a method name for the selection event listener. The selection event listener is notified about row selection in the *allDepartments* table.



The managed bean code below is used in the sample to query the *allEmployees* table to show detail data for the selected table rows in the parent table, the *allDepartments* table.

```
import java.util.Iterator;
import java.util.List;
import oracle.adf.model.BindingContext;
import oracle.adf.view.rich.component.rich.data.RichTable;
import oracle.adf.view.rich.context.AdfFacesContext;
import oracle.binding.BindingContainer;
import oracle.binding.OperationBinding;
import oracle.jbo.Row;
import oracle.jbo.domain.Number;
import oracle.jbo.uicli.binding.JUCtrlHierNodeBinding;
import org.apache.myfaces.trinidad.event.SelectionEvent;
import org.apache.myfaces.trinidad.model.RowKeySet;


public class MasterDetailBean {
  //reference to the employees table used for PPR
  private RichTable employeeTable;

  public MasterDetailBean(){}

  public void setEmployeeTable(RichTable employeeTable) {
    this.employeeTable = employeeTable;
  }
  public RichTable getEmployeeTable() {
     return employeeTable;
  }

  //Select listener defined for the departments table. The selected
  //row keys are read from the table reference. For each row, the
  //department ID value is read and added to a string buffer that
  //builds a comma separated list of departmentIds. At the end, this
  //String is passed as an argument to the ExecuteWithParams action
  //binding
  public void onDepartmentTableSelect(SelectionEvent selectionEvent) {
    //variable to hold the string containing all selected row's
    //departmentId value
    StringBuffer departmentIds = new StringBuffer();
    //get access to the master table to read selected row keys
    RichTable rt = (RichTable) selectionEvent.getSource();
    RowKeySet rks = rt.getSelectedRowKeys();
    Iterator selectedRowsIterator =  rks.iterator();
    //memorize the current row key to set it back at the end
    List currentRowKey = (List) rt.getRowKey();
    //for each selected master row, determine the departmentId
    while(selectedRowsIterator.hasNext()){
      List rowKey = (List) selectedRowsIterator.next();
      //each value is ended with a comma
      if(departmentIds.length() > 0){
         departmentIds.append(",");
       }
       //make the row current
       rt.setRowKey(rowKey);
       JUCtrlHierNodeBinding wrappedRow =
                        (JUCtrlHierNodeBinding) rt.getRowData();
```

```
        Row rw = wrappedRow.getRow();
        Number departmentId =
                        (Number) rw.getAttribute("DepartmentId");
        departmentIds.append(departmentId.stringValue());
      }
     //set row currency back
     rt.setRowKey(currentRowKey);
     //execute query on detail table
     BindingContext bctx = BindingContext.getCurrent();
     BindingContainer bindings = bctx.getCurrentBindingsEntry();
     OperationBinding executeWithParams =
                 bindings.getOperationBinding("ExecuteWithParams");
     executeWithParams.getParamsMap().put("departmentIds",
                                         departmentIds.toString());
     executeWithParams.execute();
     //refresh detail table
     AdfFacesContext adfFacesContext =
                                AdfFacesContext.getCurrentInstance();
     adfFacesContext.addPartialTarget(employeeTable);
   }
}
```

In summary: The multi master row – detail behavior is implemented using independent View Object instances. The View Object displaying the detail data uses a bind variable in its *WHERE* clause to filter its data based on the selected parent rows. Because bind variables cannot have character delimited strings if they represent numbers, a stored PLSQL function is used to parse the String data and return a valid *IN* function argument.

Sample download

An Oracle JDeveloper 11.1.1.4 sample application that is based on the Oracle HR schema can be downloaded from the ADF Code Corner website. The SQL script to create the PLSQL function that parses the *IN* argument is provided. You can download the sample ZIP file from sample #75 here:

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

RELATED DOCOMENTATION

⊠	
⊠	
⊠	