ADF Code Corner

044. How-to restrict the list of values retrieved by a model driven LOV

Abstract:

A new feature of the Oracle ADF Business Components business layer in Oracle JDeveloper 11*g* is model driven List of Values (LOV). Using model driven LOV, the list resource is configured on the View Object attribute for which the list of values should be shown. Usecases may demand that the list of values should be filtered by the current user responsibility. For example, the sales manager for North America should not see customers from other regions when using a LOV in a new sales forecast. Others may want to use this as a security precaution in that the filter should not unveil any information that is not supposed to be seen by the authenticated user. In both cases the user is member of a role that is either a security role or a business role.

twitter.com/adfcodecorner

Author: Frank Nimphius, Oracle Corporation
twitter.com/fnimphiu
07-Feb-2010

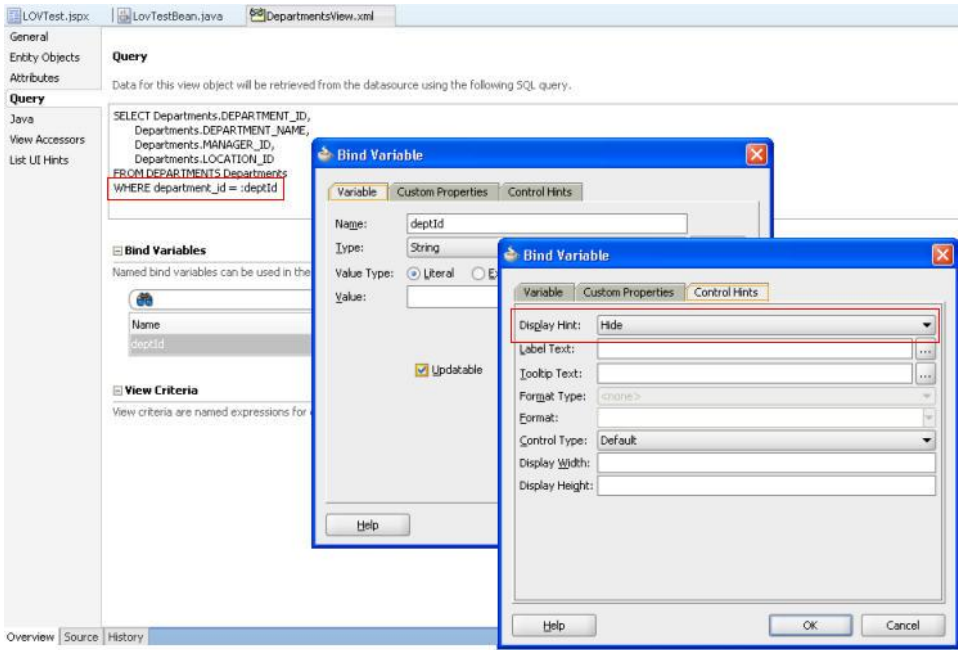
Introduction

This how-to does not explain how to create model driven LOV in ADF Business Components and other web resources exist that describe this. We start from the assumption that a model driven LOV definition is created for the "DepartmentId" attribute of the Employees View Object and that the list source points of the Departments View Object. Also we assume that the LOV UI setting is defined so an input text with LOV is used.

Bind variables

One option to implement this usecase is to create a bind variable on the Departments View Object and use it in the where clause of the query. As shown in the image below, the bind variable *deptId* is defined in the VO and added to the query where clause.

The **displays hint of the deptId variable is set so "hide",** which is important as otherwise, the variable shows as a search field in the LOV.



Query

Data for this view object will be retrieved from the datasource using the following SQL query.

```
SELECT Departments.DEPARTMENT_ID,
    Departments.DEPARTMENT_NAME,
    Departments.MANAGER_ID,
    Departments.LOCATION_ID
FROM DEPARTMENTS Departments
WHERE department_id = :deptId
```

Bind Variable

Name: deptId
Type: String
Value Type: [✓ Literal]
Value:
[✓ Updatable]

Bind Variable

Display Hint: Hide
Label Text:
Tooltip Text:
Format Type: <none>
Format:
Control Type: Default
Display Width:
Display Height:

The ADF Faces RC input form is created by dragging the Employees View object to the JSF page, choosing **Forms | ADF Form** from the popup menu after releasing the drag operation. The DepartmentId attribute automatically is rendered as a LOV component that shows its value in a text field. This is the beauty of model driven list of values, which I think is a cool feature added in this release. At least, it demos well ;-)

The page code for the list of value component is shown below

```
<af:inputListOfValues id="departmentIdId"
     popupTitle="Search and Select:
                 #{bindings.DepartmentId.hints.label}"
     value="#{bindings.DepartmentId.inputValue}"
     label="#{bindings.DepartmentId.hints.label}"
     model="#{bindings.DepartmentId.listOfValuesModel}"
     required="#{bindings.DepartmentId.hints.mandatory}"
     columns="#{bindings.DepartmentId.hints.displayWidth}"
     shortDesc="#{bindings.DepartmentId.hints.tooltip}"
     launchPopupListener="#{LovTestBean.onLovLaunch}">
   <f:validator binding="#{bindings.DepartmentId.validator}"/>
   <af:convertNumber groupingUsed="false"
           pattern="#{bindings.DepartmentId.format}"/>
</af:inputListOfValues>
```

Well spotted ! The af:inputListOfValues component has a launchPopupListener defined, and its this listener that sets the query filter for us. The managed bean code is shown below and sets a - in this example - hard coded value to the bind variable defined in the View object

```
 public void onLovLaunch(LaunchPopupEvent launchPopupEvent) {
    BindingContext bctx = BindingContext.getCurrent();
    BindingContainer bindings = bctx.getCurrentBindingsEntry();
    FacesCtrlLOVBinding lov =
              (FacesCtrlLOVBinding)bindings.get("DepartmentId");
    lov.getListIterBinding().getViewObject()
                       .setNamedWhereClauseParam("deptId","60");
 }
```

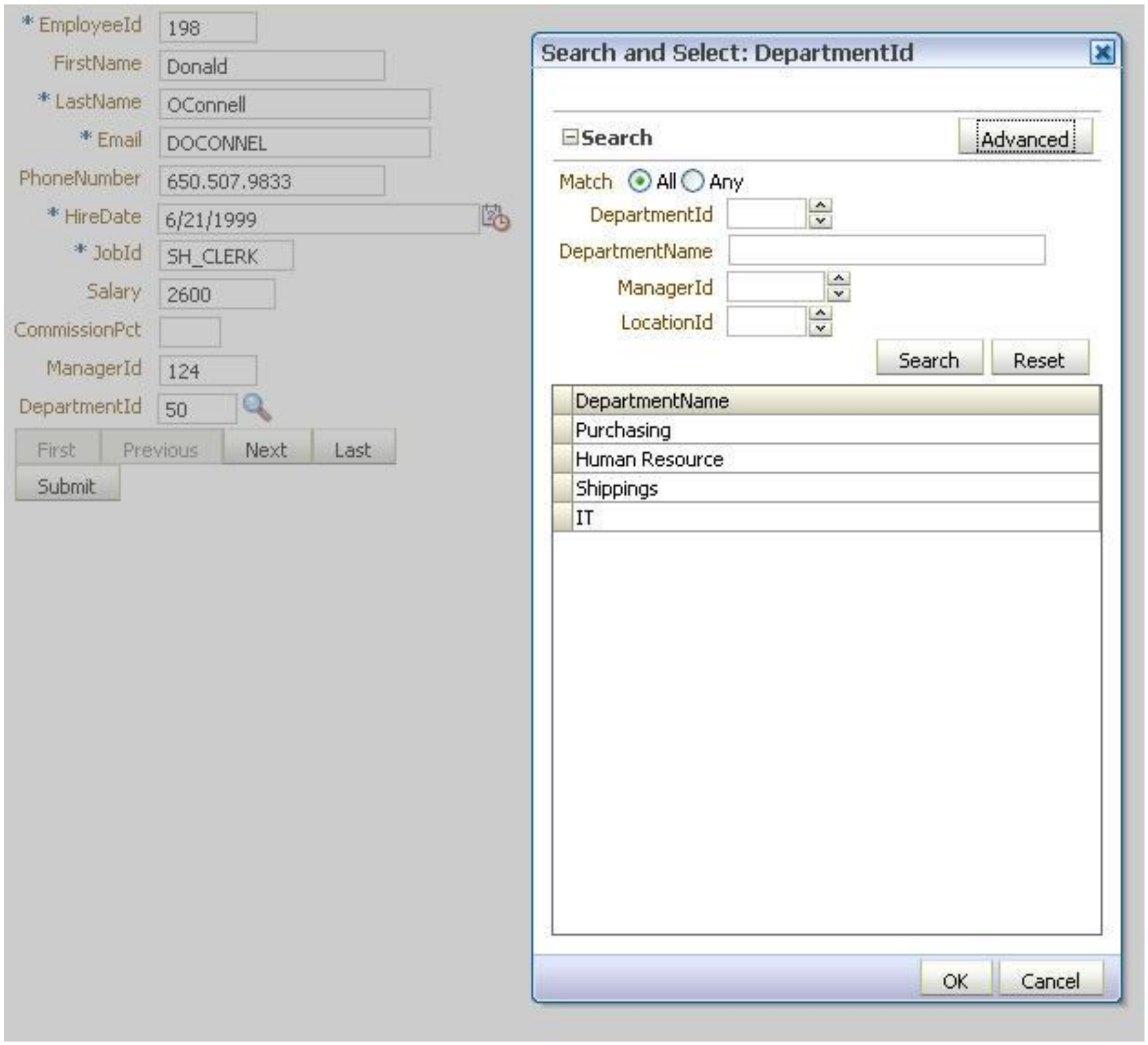
Using this listener, restricts all values shown in the LOV to those that at least have a department id of 60, which in the case of the Department View Object is one entry, but you get the idea.

Note that "DepartmentId" in the code line `(FacesCtrlLOVBinding)bindings.get( "DepartmentId");` references the attribute binding of the input list of value component in the pageDef file.

Using appending a where clause

When a bind variable cannot be used, or if the where clause needs to be complex then appending the where clause from the launchPopupListener is an option to use. So modifying the managed bean to the code sample shown below will list all entries for the departments 30,40,50 and 60 if the LOV is queried for all records.

```
public void onLovLaunch(LaunchPopupEvent launchPopupEvent) {
     BindingContext bctx = BindingContext.getCurrent();
     BindingContainer bindings = bctx.getCurrentBindingsEntry();
     FacesCtrlLOVBinding lov =
          (FacesCtrlLOVBinding)bindings.get("DepartmentId");
     String wcl = "department_id in (30,40,50,60)";
     lov.getListIterBinding().getViewObject().setWhereClause(wcl);
}
```



* EmployeeId	198
FirstName	Donald
* LastName	OConnell
* Email	DOCONNEL
PhoneNumber	650.507.9833
* HireDate	6/21/1999
* JobId	SH_CLERK
Salary	2600
CommissionPct	
ManagerId	124
DepartmentId	50

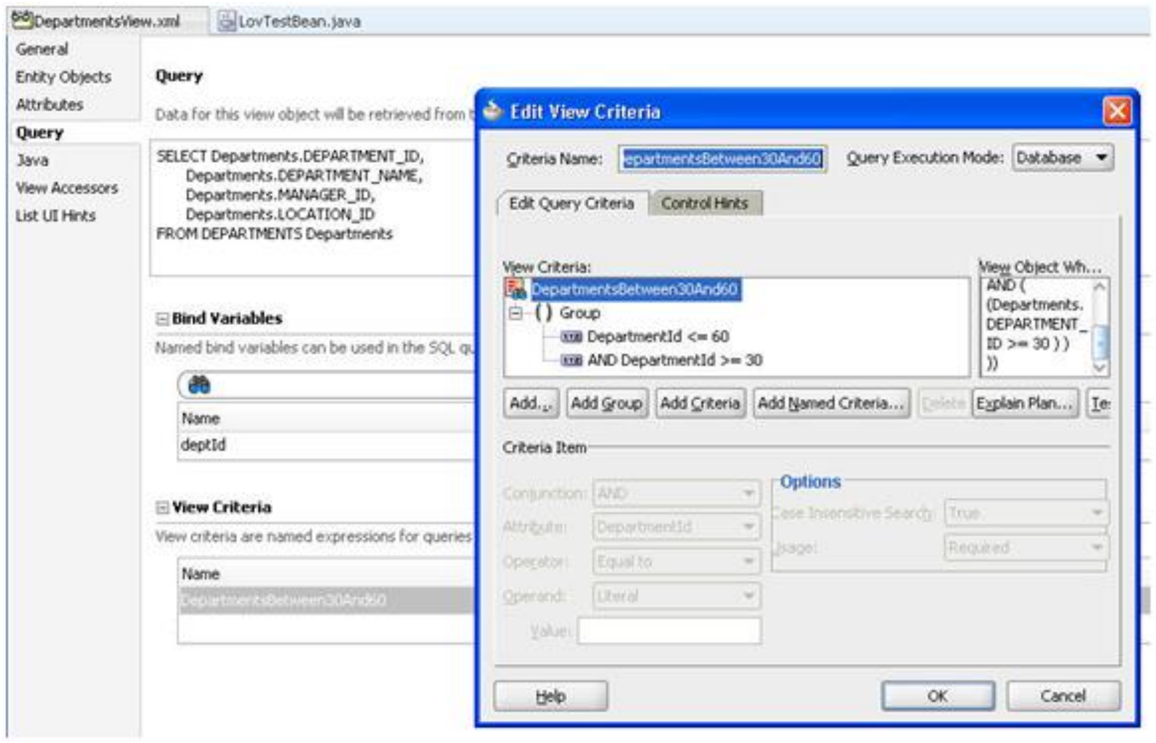
DepartmentName
Purchasing
Human Resource
Shippings
IT

Using named ViewCriterias

Now that I showed you all that works, let me show you what I like the best and think probably is best practices too.

View Criterias are one-time where clauses added to a query. In JDeveloper 11*g*, you can define named view criterias that so that all developers that need to append a where clause can use a consistent definition, making it easier to enforce best practices and audit application business rules. ViewCriteria can be created with or without bind variables, where the bind variables used can be defined as "optional" - or not required - which means that the View object could be used without populating the bind variable, solving the limitation that exist when hard coding the variable usage into the View object query where clause.

Lets create the same two samples using View Criteria on the Departments View object First, query for departments between 30 and 60.



```
SELECT Departments.DEPARTMENT_ID,
       Departments.DEPARTMENT_NAME,
       Departments.MANAGER_ID,
       Departments.LOCATION_ID
FROM DEPARTMENTS Departments
```

Name: deptId

View Criteria Name: DepartmentsBetween30And60

Criteria Name: DepartmentsBetween30And60   Query Execution Mode: Database

```
DepartmentsBetween30And60
  () Group
      DepartmentId <= 60
      AND DepartmentId >= 30
```

```
AND (
(Departments.
DEPARTMENT_
ID >= 30 ) )
))
```

Conjunction: AND
Attribute: DepartmentId
Operator: Equal to
Operand: Literal
Value:
Case Insensitive Search: True
Usage: Required

The launchPopupListener code for this is shown below:

```
public void onLovLaunch(LaunchPopupEvent launchPopupEvent) {
   BindingContext bctx = BindingContext.getCurrent();
   BindingContainer bindings = bctx.getCurrentBindingsEntry();
   FacesCtrlLOVBinding lov =
           (FacesCtrlLOVBinding)bindings.get("DepartmentId");
   ViewCriteriaManager vcm =
     lov.getListIterBinding().getViewObject().getViewCriteriaManager();
   ViewCriteria vc = vcm.getViewCriteria("DepartmentsBetween30And60");
   lov.getListIterBinding().getViewObject().applyViewCriteria(vc);
}
```

Instead of using the where clause on the View object, the listener queries the ViewCriteriaManager for the "DepartmentsBetween30and60" ViewCriteria to apply it to the VO.

However, more likely you need to filter the LOV dialog more dynamic than this. For this you can use the code shown below, which dynamically creates a view criteria and applies it to the LOV query. Using the code sample below, you can filter the LOV using all queryable attributes of the View Object you use for populating the LOV search popup.

In the sample below, the LOV is pre-filtered by the DepartmentId, which I set to 60 (however, the example is flexible)

```
public void onPopupLaunch(LaunchPopupEvent launchPopupEvent) {
  BindingContext bctx = BindingContext.getCurrent();
  BindingContainer bindings = bctx.getCurrentBindingsEntry();
  FacesCtrlLOVBinding lov =
           (FacesCtrlLOVBinding)bindings.get("DepartmentId");
  ViewCriteriaManager vcm =
     lov.getListIterBinding().getViewObject().getViewCriteriaManager();

  //make sure the view criteria is cleared
  vcm.removeViewCriteria(vcm.DFLT_VIEW_CRITERIA_NAME);
  //create a new view criteria
  ViewCriteria vc =
         new ViewCriteria(lov.getListIterBinding().getViewObject());
  //use the default view criteria name
  //"__DefaultViewCriteria__"
  vc.setName(vcm.DFLT_VIEW_CRITERIA_NAME);
  //create a view criteria row for all queryable attributes
  ViewCriteriaRow vcr = new ViewCriteriaRow(vc);
  //for this sample I set the query filter to DepartmentId 60.
  //You may determine it at runtime by reading it from a managed bean
  //or binding layer
  vcr.setAttribute("DepartmentId", 60);
  vc.addRow(vcr);

  lov.getListIterBinding().getViewObject().applyViewCriteria(vc);
}
```

If you want to use a named view criteria defined at design time instead of a dynamically created one, you can define a named view criteria with all queryable attributes (no bind variables, just the attributes you want to query when filtering the LOV view object) in the View Object editor and apply it at runtime by its name.

For example, if the named view criteria you design at runtime is named "myQueryAbleVC" then the code above needs to be slightly changed as shown below

```
public void onPopupLaunch(LaunchPopupEvent launchPopupEvent) {
  BindingContext bctx = BindingContext.getCurrent();
  BindingContainer bindings = bctx.getCurrentBindingsEntry();
  FacesCtrlLOVBinding lov =
           (FacesCtrlLOVBinding)bindings.get("DepartmentId");
  ViewCriteriaManager vcm =
     lov.getListIterBinding().getViewObject().getViewCriteriaManager();

  //make sure the view criteria is cleared
  vcm.removeViewCriteria(vcm.DFLT_VIEW_CRITERIA_NAME);
  //create a new view criteria
  ViewCriteria vc = vcm.getViewCriteria("myQueryAbleVC");
  ViewCriteriaRow vcr = new ViewCriteriaRow(vc);
  vcr.setAttribute("DepartmentId", 60);
  vc.addRow(vcr);
  lov.getListIterBinding().getViewObject().applyViewCriteria(vc);
}
```