

ADF Code Corner

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

ORACLE
CODE CORNER



twitter.com/adfcodecorner

Abstract:

A new feature of the Oracle ADF Business Components business layer in Oracle JDeveloper 11g 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.

Author:

Frank Nimphius, Oracle Corporation
twitter.com/fnimphiu
07-Feb-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>

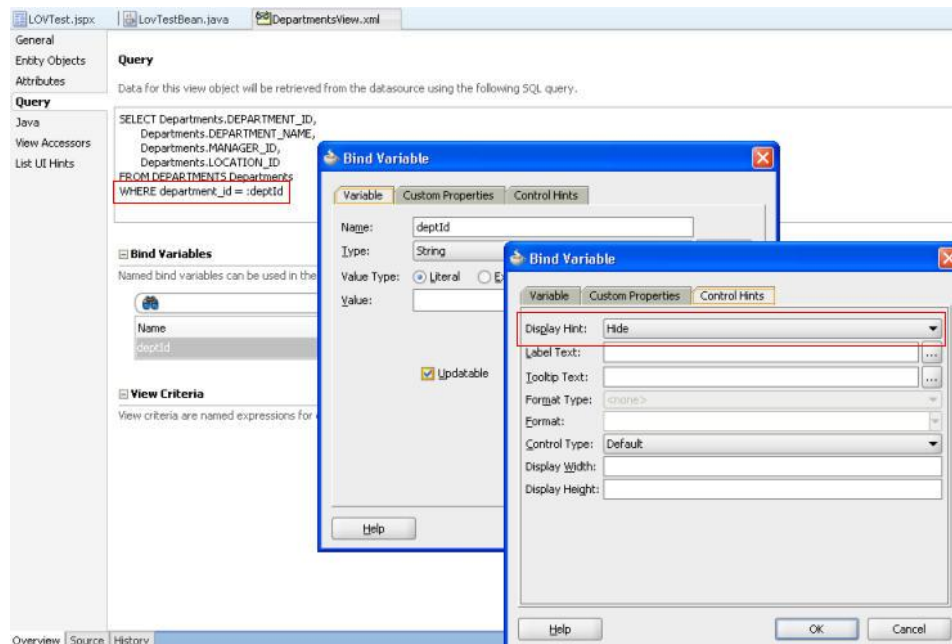
Introduction

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.



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);
}

```

The screenshot shows a web form for an employee record. The form fields are as follows:

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

Navigation buttons: First, Previous, Next, Last, Submit.

The 'Search and Select: DepartmentId' dialog box is open, showing the following search criteria:

- Match: All Any
- DepartmentId: (dropdown)
- DepartmentName: (text input)
- ManagerId: (dropdown)
- LocationId: (dropdown)

Buttons: Search, Reset, OK, Cancel.

The search results table is as follows:

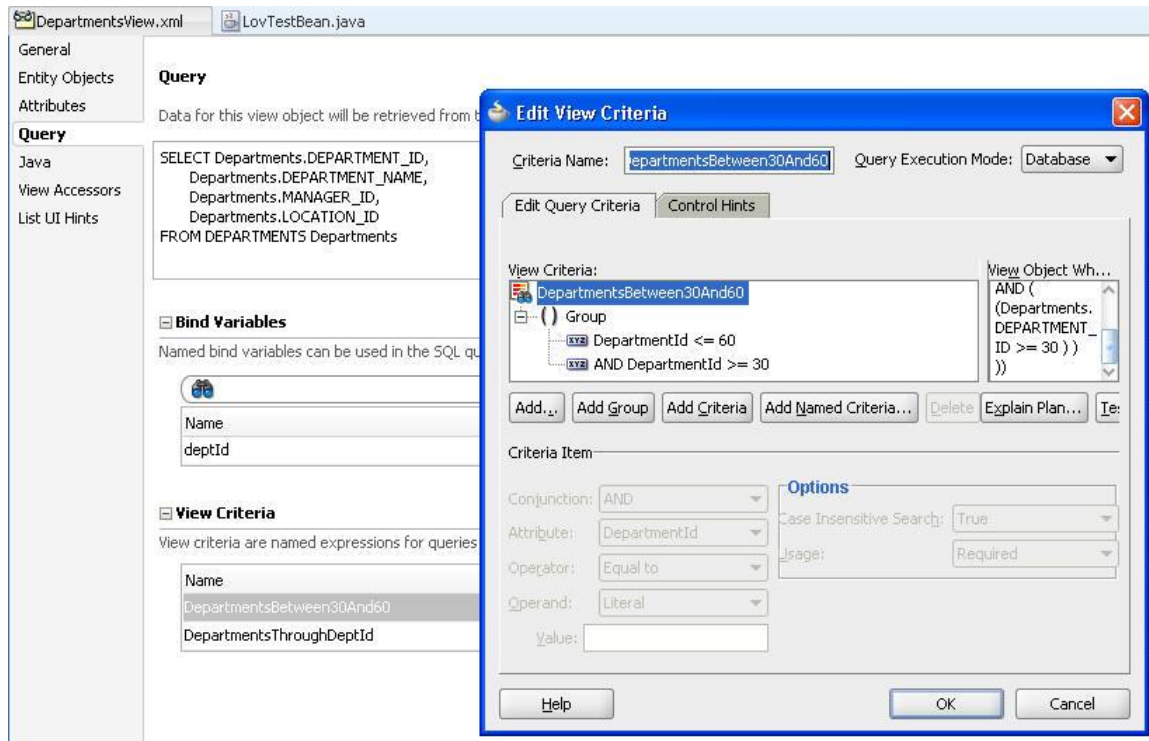
DepartmentName
Purchasing
Human Resource
Shippings
IT

Using named ViewCriteria

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 Criteria are one-time where clauses added to a query. In JDeveloper 11g, 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.

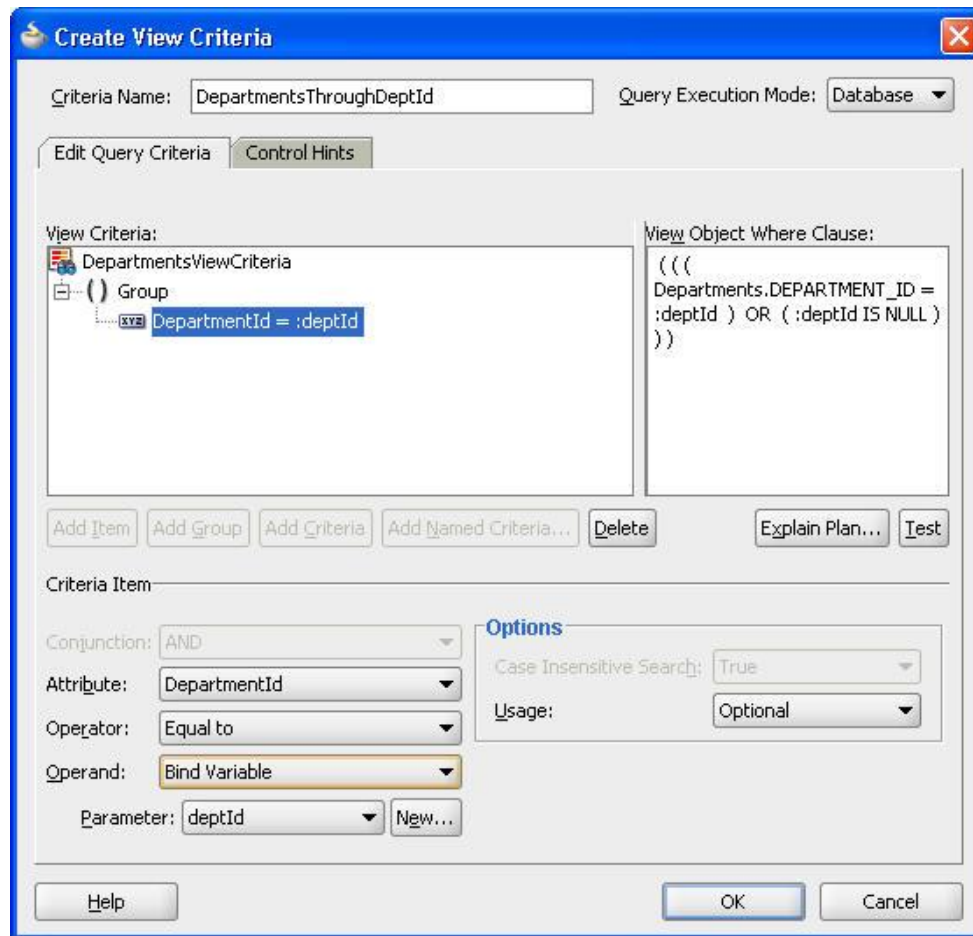
Lets create the same two samples using named View Criteria on the Departments View object First, query for departments between 30 and 60.



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. To use a ViewCriteria that references a bind variable in its query



... define the `launchPopupListener` as follows

```
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");
    ViewCriteriaManager vcm =
        lov.getListIterBinding().getViewObject().getViewCriteriaManager();
    ViewCriteria vc = vcm.getViewCriteria("DepartmentsThroughDeptId");
    lov.getListIterBinding().getViewObject().applyViewCriteria(vc);
}
```