R12.x Extend Oracle Applications: Building OA Framework Applications

Student Activity Guide

D61636GC10
Edition 1.0
November 2010
D69575

ORACLE

Author

Lauren Cohn

Technical Contributors and Reviewers

Mike Waddoups, Phil Cannon

Curriculum Manager

Bill Sawyer

This book was published using: Oracle Tutor

Table of Contents

Lab - Learning More About the Page 1-1
Lab - Learning More About the Page 1-3
Task 1: Login to Your E-Business Suite Instance 1-4
Task 2: About This Page Link 1-7
Solution: About This Page Link 1-13

Lab - JDeveloper and E-Business Suite Set-up 2-1
Task 1: Preparation 2-3
Task 2: Open the NX Client on Your Classroom PC 2-4
Task 3: Modify the FWKTESTER User 2-6
Task 4: Unzip Tutorial.zip 2-9
Task 5: Set Your JDeveloper Environment 2-12
Task 6: Start JDeveloper 2-13
Task 7: Get the DBC File 2-16
Task 8: Test Your JDeveloper Workspace and Project 2-17
Task 9: Set-up Your Database Connection 2-21
Task 10: Set Your Project Properties 2-27
Task 11: Test Your JDeveloper Set-up 2-32
Task 12: Optional – Personal Set-up 2-35

Lab - First OA Framework Page 3-1
Overview 3-3
Task 1: Setup your environment for OA Framework development 3-4
Task 2: Create the Model-layer Components 3-8
Step 3: Create the View-layer Components 3-16

Lab - Implementing a Query and Drill-down 4-1
Overview 4-3
Task 1: Create Your Model-layer Components 4-4
Task 2: Create Your View-layer Components 4-9
Task 3: Configure Your Search 4-12
Task 4: Add a List of Values (LOV) to Query 4-30
Task 5: Create Your Drill Down Page 4-41
Task 6: Create Your View-layer Components 4-45
Task 7: Write Your Model-layer Programmatic Elements 4-49
Task 8: Modify Your Existing View-layer Components 4-51
Task 9: Write Your Controller-layer Programmatic Elements 4-52

Lab - Implementing a Create 5-1
Part 1: Basic Create Overview 5-3
Task 1: Build Your Model-layer Components for the Create Page 5-4
Task 2: Build Your View-layer Components 5-5
Task 3: Implement the Poplist 5-12
Task 4: Write Your Model-layer Programmatic Elements 5-15
Task 5: Write Your Controller-layer Programmatic Elements 5-17
Task 6: Revise Your EmployeePG 5-22
Part 2: Validation Overview 5-24
Task 7: Handle the Back Button 5-25
Task 8: Implement Declarative Validations 5-34

Task 9: Implement Programmatic Validations 5-36
Task 10: Creating Entity-Layer Validation Objects 5-49
Part 3: Partial Page Rendering Overview 5-56
Task 11: Create Your Model-layer Components 5-57
Task 12: Configure Your View-layer Components 5-60
Task 13: Implement the Programmatic Elements 5-61

Lab - Implementing a Delete 6-1
Overview 6-3
Task 1: Build Your Model-layer Components 6-4
Task 2: Build Your View-layer Components 6-6
Task 3: Write Your Model-layer Programmatic Elements 6-9
Task 4: Write Your Controller-layer Programmatic Elements 6-11

Lab - Implementing an Update 7-1
Overview 7-3
Task 1: Initial Set-up 7-4
Task 2: Modify the EmployeePG View-layer Components 7-6
Task 3: Modify the Controller-layer Components 7-7
Task 4: Create Your View-layer Components for Multi-Page 7-12
Task 5: Create Your Controller-layer Components 7-21
Task 6 – The Final Challenge 7-30

Lab - Learning More About the Page

Chapter 1

Lab - Learning More About the Page

One of the core skills of any OA Framework developer is diagnosing and examining a running OA Framework page. As a developer, having an understanding of where the page “comes from” allows you to examine locations within the Oracle E-Business Suite Applications directory structure that may not be obvious.

OAF technology is based on an MVC framework, the controllers, models and views are not in the same locations or in the same technology. This lab will walk you through the beginning steps towards understanding how OAF pages are rendered, stored and accessed.

Task 1: Login to Your E-Business Suite Instance

In this course, you have an E-Business Suite instance that is assigned to your terminal. The E-Business Suite instance assigned to your terminal is for you and your partner - if you have one. No other students in the class will share your E-Business Suite instance.

Within this lab there might be some inconsistencies in versions or pages/patches/files since this lab was printed. Those inconsistencies are normal as your lab's instances are continually updated and patched. It is reasonable to not have an "exact match" as you go through this lab.

Every effort has been made to give the most up-to-date screen captures and examples as possible. However, If you see something that is patently incorrect, please notify your instructor.

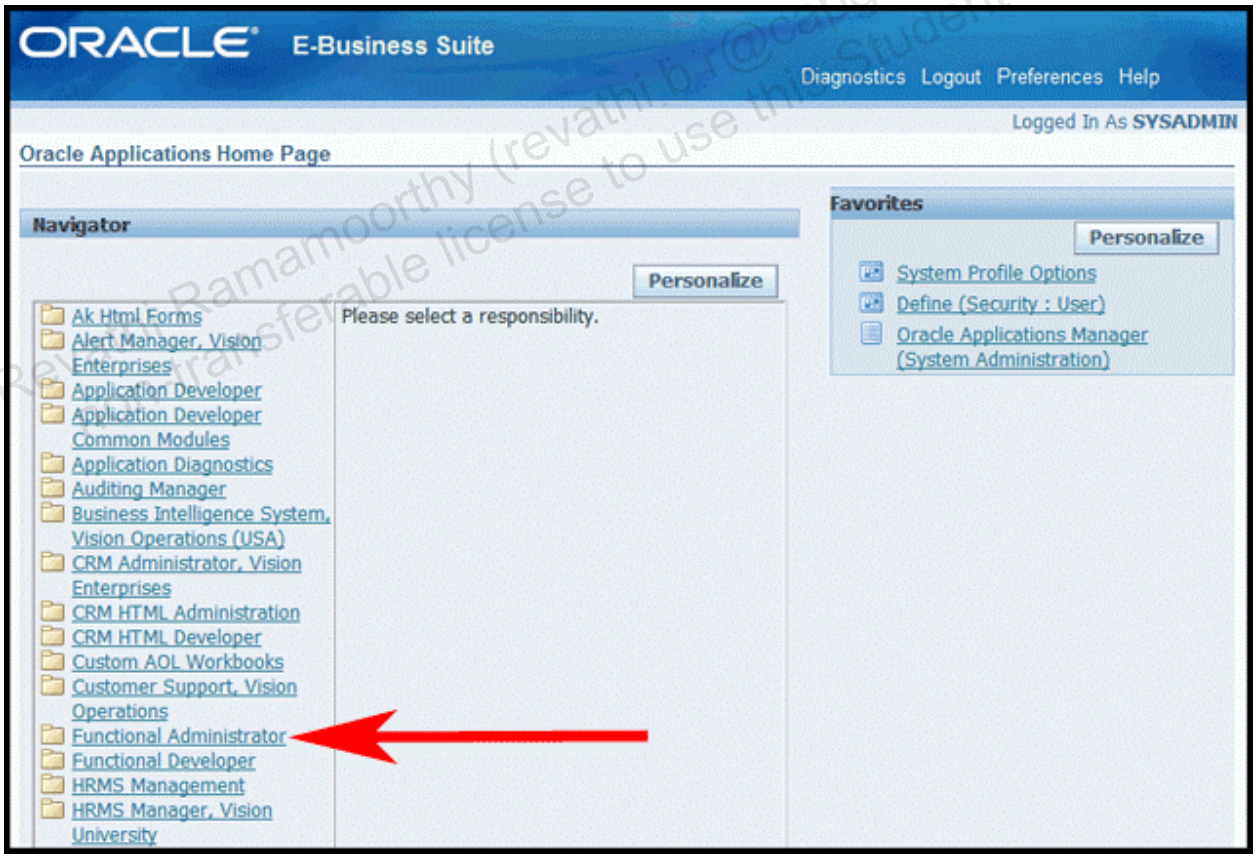
Your instructor will tell you your E-Business Suite instance URL. Write that below:

My E-Business Suite URL is: http://_____

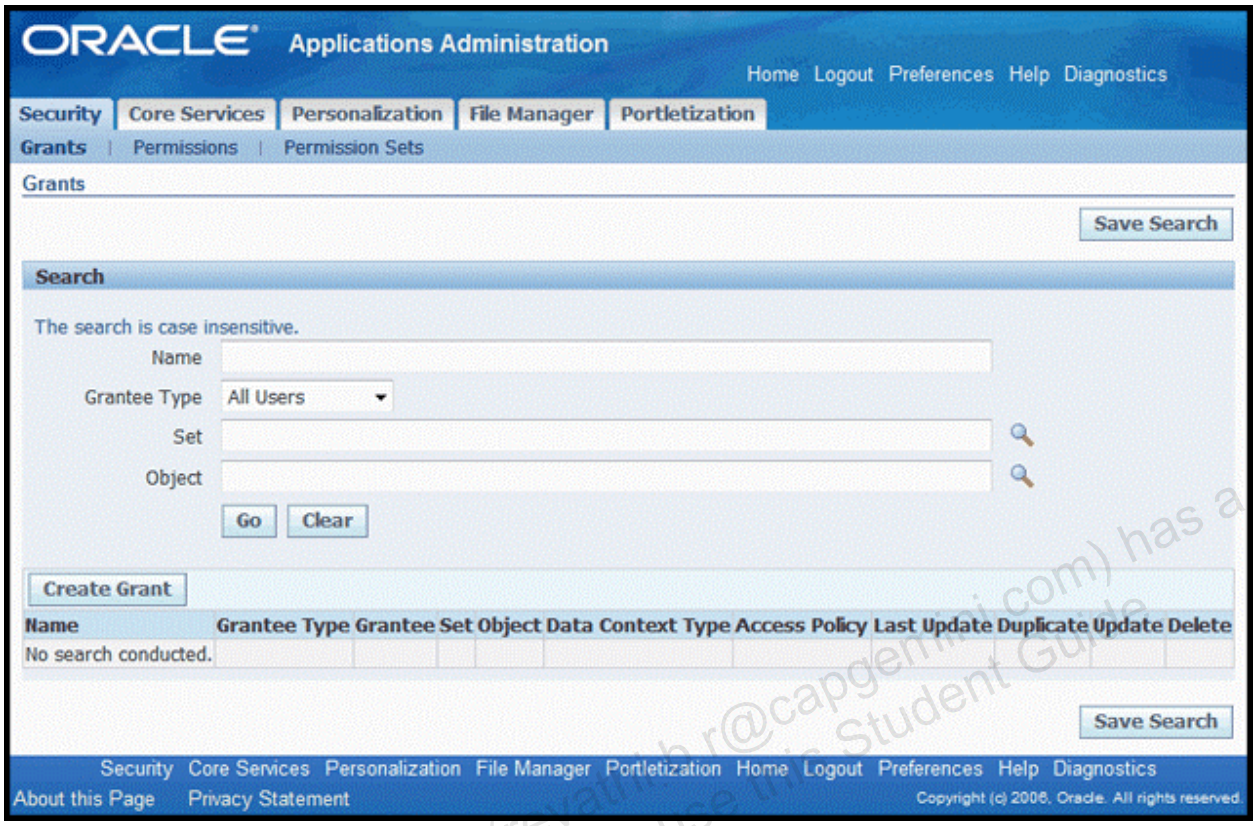
1. Open your browser.
2. Enter the URL for your E-Business Suite.
3. You will see the redirection screen which will look similar to the following:

4. Login to the E-Business Suite as SYSADMIN/SYSADMIN. Click the **Login** button.

5. Click the **Functional Administrator** responsibility from the E-Business Suite Navigator.



6. The page should look similar to the following:

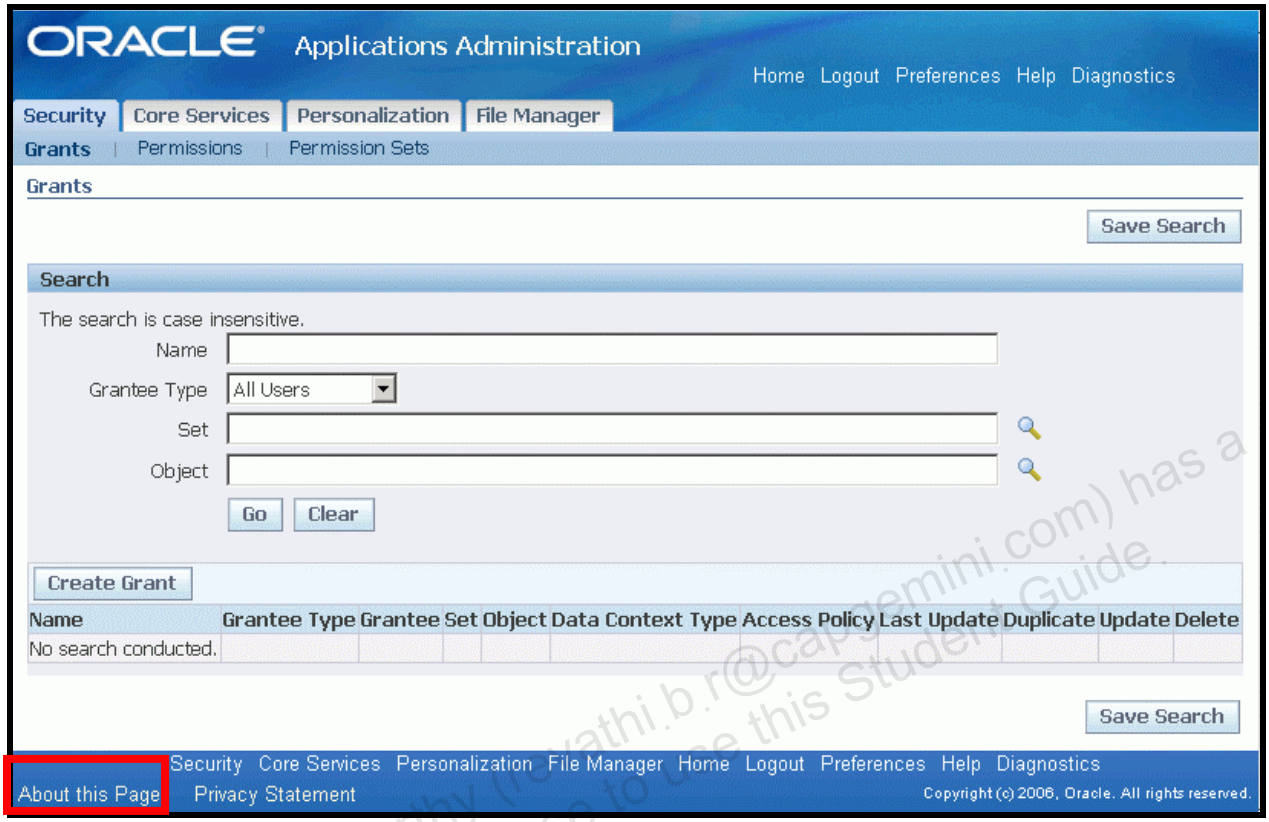


Note: Portletization is a new feature as part of Oracle E-Business Suite release 12.1.2.

See the Oracle Application Object Library Release Notes - 953916.1 on My Oracle Support (MOS) for more information on the Portletization feature.

Task 2: About This Page Link

1. Click the **About this Page** link shown on the footer (bottom) of the page.



2. The page you see should look similar to the following:

The detailed steps to answer the following questions are shown at the end of this lab. Your current knowledge about OA Framework, and a little exploration, should be enough to answer the questions.

3. What is the full name of the page? (complete path)

   _____

4. What is the version of the page?

   _____

5. What is the name of the pageLayout region (and therefore is the name of the page)?

   _____

6. What is the root AM of the page?

   _____

7. What controller does the pageLayout region use?

   _____

8. Where is the page stored?

   _____

9. How is the page stored? **(Circle one)**
   In XML only / In Java only / Both XML and Java / PL-SQL / HTML / JSP
10. Beneath the Page Definition region is a link titled, Expand All. Click the **Expand All** link.
11. List four (4) UI web beans used on the page?

    _____
    _____
    _____
    _____

12. Is there more than one controller used on the page? **(Circle one)**
    YES / NO
13. If Yes, what are the name(s) of the additional controllers?

    _____
    _____
    _____
    _____

14. What VO is being used by the UI objects?

    _____

15. Expand the **Business Component References Details** region.
16. What is the complete path to the AM?

    _____

17. Are there any other AMs used by the page? **(Circle one)**
    YES / NO
18. If Yes, list the other AMs used by the page?

    _____
    _____
    _____
    _____

19. Question 16 asked what VO was being used. What is the complete path to that VO?

    _____

20. What is the EO associated to the VO from Question 16/21? (list the full path)

    _____

21. Does this page use any flexfields? **(Circle one)**
    YES / NO
22. List the flexfield used by this page?

    _____

23. Click the **GrantsResultsVO** link to drill-down to the details of this VO.
24. What version is this VO?

    _____

25. Does this VO use an EO? **(Circle one)**
    YES / NO

26. If Yes, list the complete name of this EO?

    _____

27. Does this EO have an associated EntityExpert? **(Circle one)**
    YES / NO
28. If Yes, list the full path name of the EntityExpert?

    _____

29. List the version of the EO from Question 28?

    _____

30. List four (4) attributes that are contained in the VO?

    _____
    _____
    _____
    _____

31. Click the **Return to About Page** link.
32. Click the **Personalization** tab.
33. List the Personalizations associated with this page?

    _____
    _____
    _____
    _____

34. Click the **Page Context** tab.
35. What User Name is being used on this page?

    _____

36. What Responsibility is this page using?

    _____

37. What is the Responsibility Key being used on this page?

    _____

38. What Function is this page using?

    _____

39. Click the **Technology Components** tab.
40. What version of OA Extension is being used on this server?

    _____

41. Click the **Java Systems Properties** tab.
42. Where does APPL_TOP point to on this server? (list the complete path)

    _____

43. Click the **Profiles** tab.
44. Expand the **Logging/Diagnostics** region.
45. What is the value of the FND: Diagnostics profile option?

    _____

    Note: FND: Diagnostics is the profile option that controls whether or not you see the About this Page link. It must be set in order for you to see the link.
46. Click the **Patches** tab.

47. Have patches been applied to this E-Business Suite instance? **(Circle one)**
    YES / NO
48. If Yes, list the last patch number applied to this server?

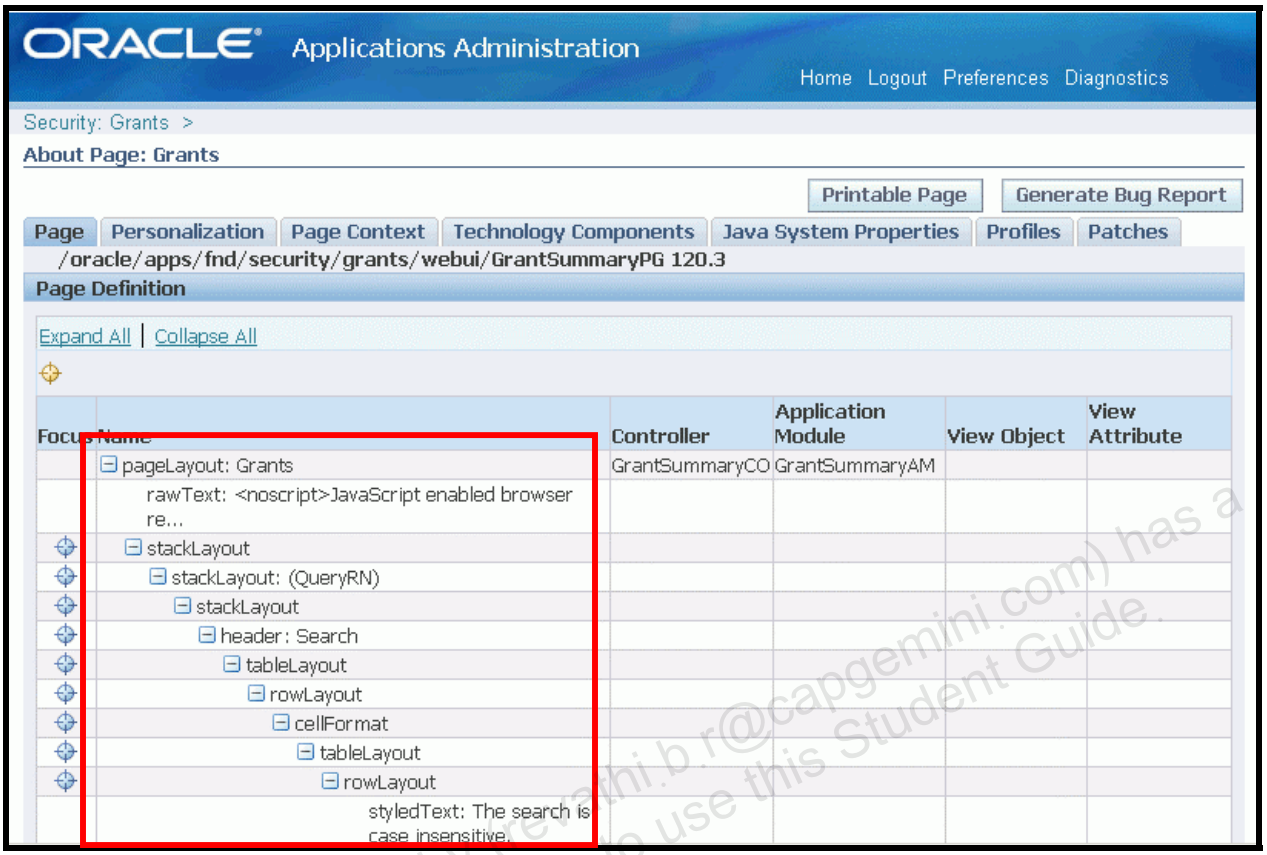
    _____

49. Click the **Return to Page: Grants** link.

Lab - Learning More About the Page

3. What is the full name of the page? (complete path)
   oracle/apps/fnd/security/grants/webui/GrantSummaryPG 120.3
4. What is the version of the page?
   120.3
5. What is the name of the pageLayout region (and therefore is the name of the page)?
   Grants
6. What is the root AM of the page?
   GrantSummaryAM
7. What controller does the pageLayout region use?
   GrantSummaryCO
8. Where is the page stored?
   In the MDS. All OA Framework pages are stored there.
9. How is the page stored?
   In XML only

10. Beneath the Page Definition region is a link titled, Expand All. Click the **Expand All** link.



11. List four (4) UI web beans used on the page?

    Any 4 of the following: stackLayout, header, tableLayout, rowLayout, cellFormat, styledText, messageTextInput, messageRadioGroup, messageLovInput, submitButton, table, messageStyledText, image, flowLayout, sortableHeader, link, pageButtonBar

12. Is there more than one controller used on the page?

    YES – As mentioned earlier. It was a trick question. Now, as you scroll down through the UI objects, you will see that a second controller is mentioned.

13. If Yes, what are the name(s) of the additional controllers?

    GrantResultsCO

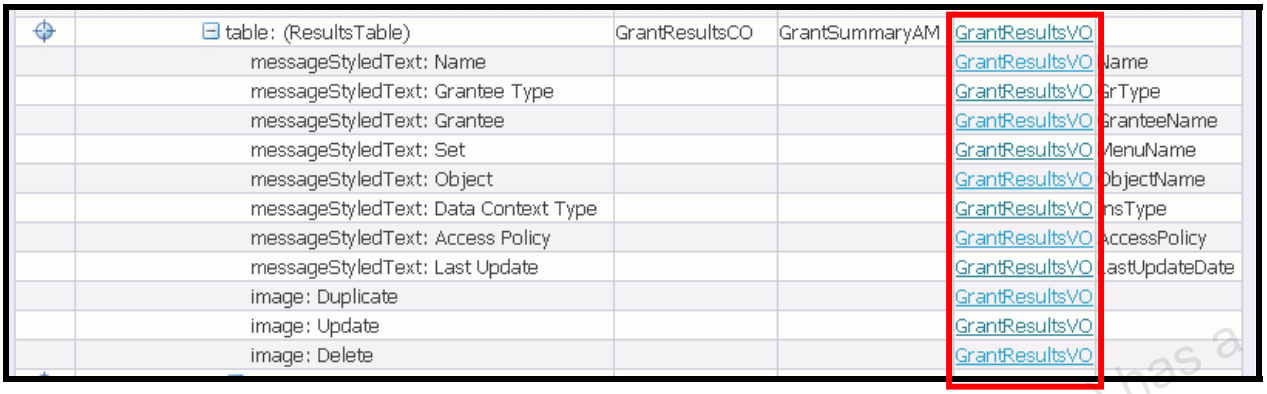
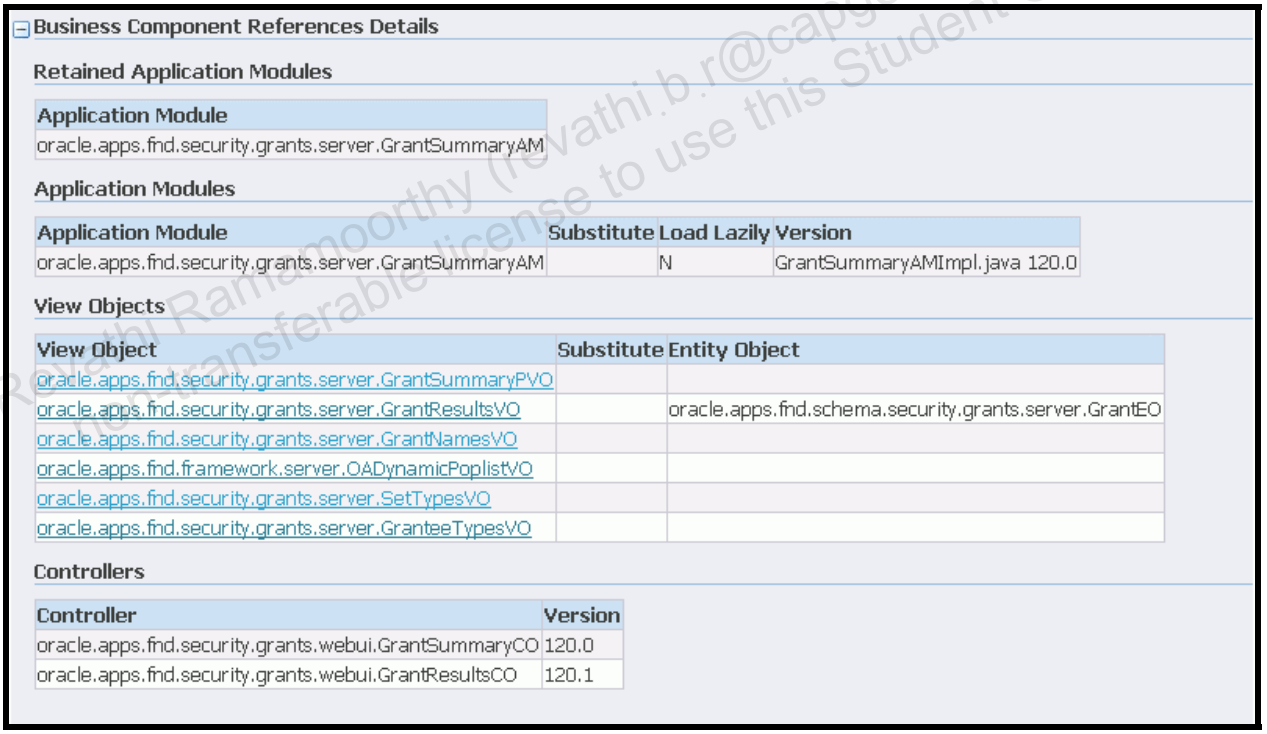


	table: (ResultsTable)	GrantResultsCO	GrantSummaryAM	GrantResultsVO
	messageStyledText: Name			GrantResultsVO
	messageStyledText: Grantee Type			GrantResultsVO
	messageStyledText: Grantee			GrantResultsVO
	messageStyledText: Set			GrantResultsVO
	messageStyledText: Object			GrantResultsVO
	messageStyledText: Data Context Type			GrantResultsVO
	messageStyledText: Access Policy			GrantResultsVO
	messageStyledText: Last Update			GrantResultsVO
	image: Duplicate			GrantResultsVO
	image: Update			GrantResultsVO
	image: Delete			GrantResultsVO

14. What VO is being used by the UI objects?

    GrantResultsVO

15. Expand the Business Component References Details region.



Business Component References Details

Retained Application Modules

Application Module
oracle.apps.fnd.security.grants.server.GrantSummaryAM

Application Modules

Application Module	Substitute	Load Lazily	Version
oracle.apps.fnd.security.grants.server.GrantSummaryAM		N	GrantSummaryAMImpl.java 120.0

View Objects

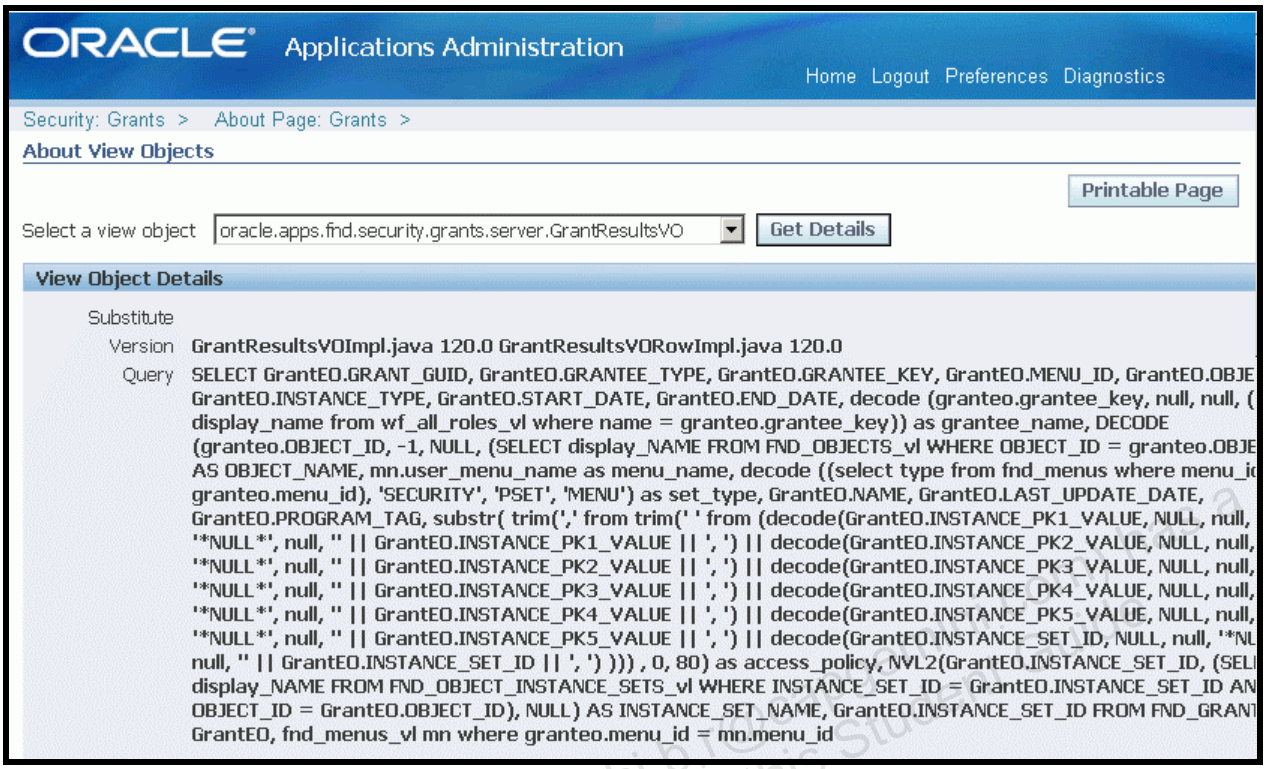
View Object	Substitute	Entity Object
oracle.apps.fnd.security.grants.server.GrantSummaryPVO		
oracle.apps.fnd.security.grants.server.GrantResultsVO		oracle.apps.fnd.schema.security.grants.server.GrantEO
oracle.apps.fnd.security.grants.server.GrantNamesVO		
oracle.apps.fnd.framework.server.OADynamicPoplistVO		
oracle.apps.fnd.security.grants.server.SetTypesVO		
oracle.apps.fnd.security.grants.server.GranteeTypesVO		

Controllers

Controller	Version
oracle.apps.fnd.security.grants.webui.GrantSummaryCO	120.0
oracle.apps.fnd.security.grants.webui.GrantResultsCO	120.1

16. What is the complete path to the AM?
    oracle.apps.fnd.security.grants.server.GrantSummaryAM
17. Are there any other AMs used by the page?
    NO
18. If Yes, list the other AMs used by the page?
    There are no other AMs. Trick question of sorts.
19. Question 16 asked what VO was being used. What is the complete path to that VO?
    oracle.apps.fnd.security.grants.server.GrantResultsVO
20. What is the EO associated to the VO from Question 16/21?
    oracle.apps.fnd.schema.security.grants.server.GrantEO
21. Does this page use any flexfields?
    NO
22. List the flexfield used by this page?
    There are no flexfields. Trick question of sorts.

23. Click the **GrantsResultsVO** link to drill-down to the details of this VO.



24. What version is this VO?

    120

25. Does this VO use an EO?

    YES – But, it is a trick question. You already answered this. But, if you scroll down on this page, you can see additional information on the EO.

26. If Yes, list the complete name of this EO?

    See Question 27. oracle.apps.fnd.schema.security.grants.server.GrantEO

27. Does this EO have an associated EntityExpert?

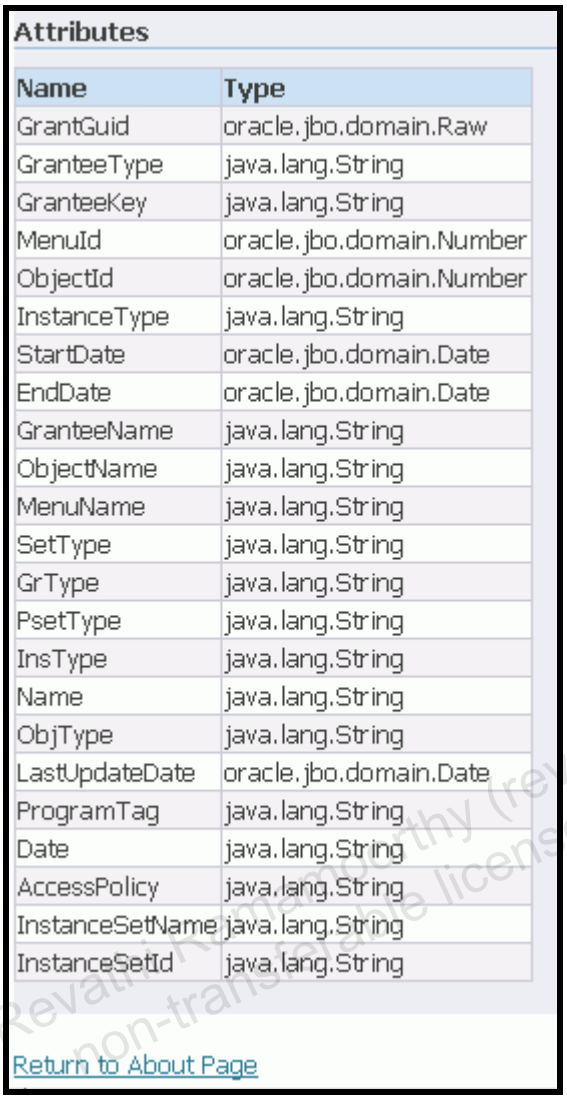
    YES

28. If Yes, list the complete name of the EntityExpert?

    oracle.apps.fnd.schema.security.grants.server.GrantEntityExpert

29. List the version of the EO from Question 28?

    120.2



Attributes

Name	Type
GrantGuid	oracle.jbo.domain.Raw
GranteeType	java.lang.String
GranteeKey	java.lang.String
MenuId	oracle.jbo.domain.Number
ObjectId	oracle.jbo.domain.Number
InstanceType	java.lang.String
StartDate	oracle.jbo.domain.Date
EndDate	oracle.jbo.domain.Date
GranteeName	java.lang.String
ObjectName	java.lang.String
MenuName	java.lang.String
SetType	java.lang.String
GrType	java.lang.String
PsetType	java.lang.String
InsType	java.lang.String
Name	java.lang.String
ObjType	java.lang.String
LastUpdateDate	oracle.jbo.domain.Date
ProgramTag	java.lang.String
Date	java.lang.String
AccessPolicy	java.lang.String
InstanceSetName	java.lang.String
InstanceSetId	java.lang.String

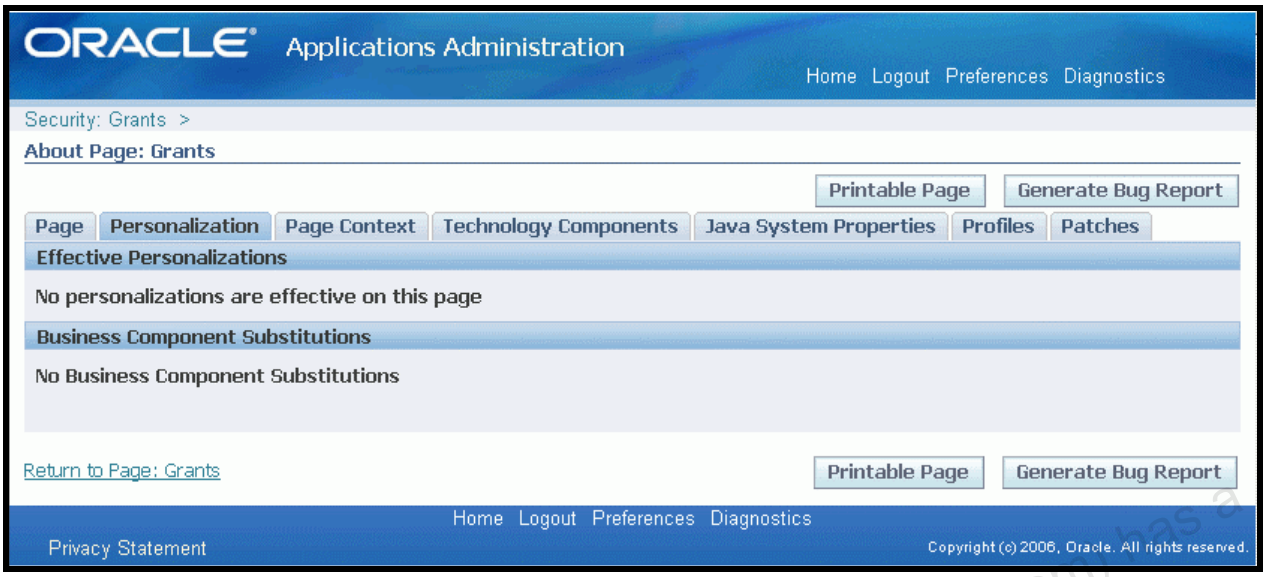
Return to About Page

30. List four (4) attributes that are contained in the VO?

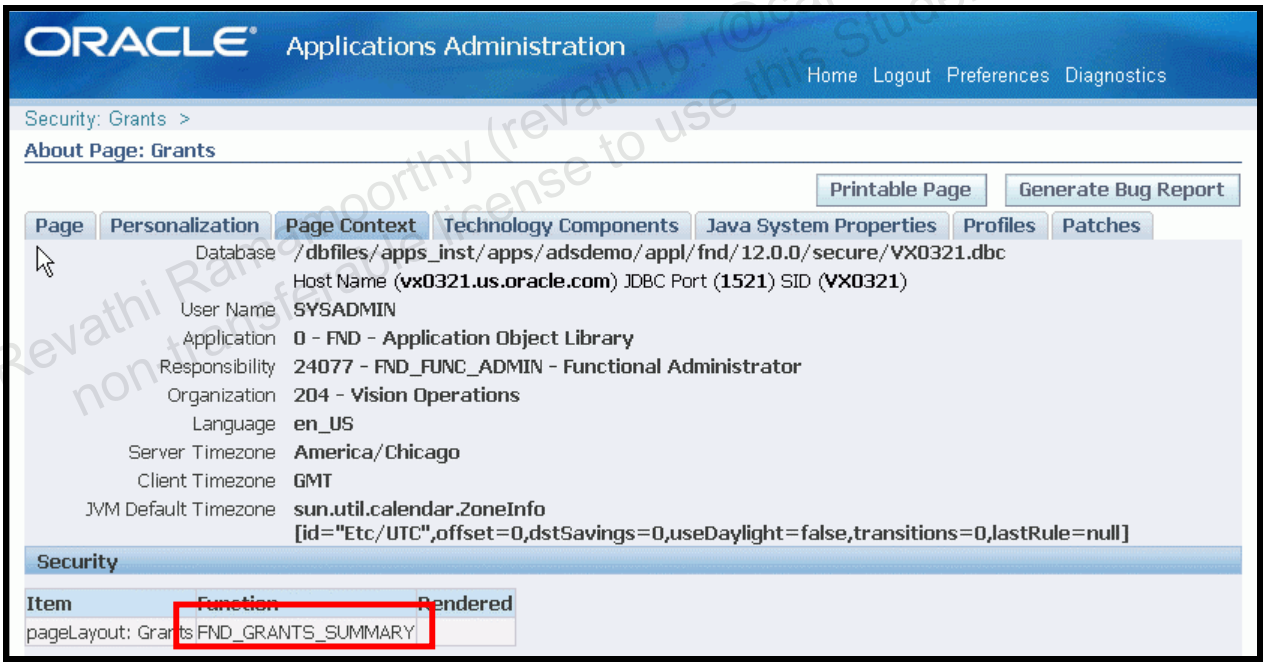
    List any 4 of the following: GrantGuid, GranteeType, GranteeKey, MenuId, ObjectId, InstanceType, StartDate, EndDate, GranteeName, ObjectName, MenuName, SetType, GrType, PsetType, InsType, Name, ObjType, LastUpdateDate, ProgramTag, Date, AccessPolicy, InstanceSetName, InstanceSetId

31. Click the **Return to About Page** link.

32. Click the **Personalization** tab.

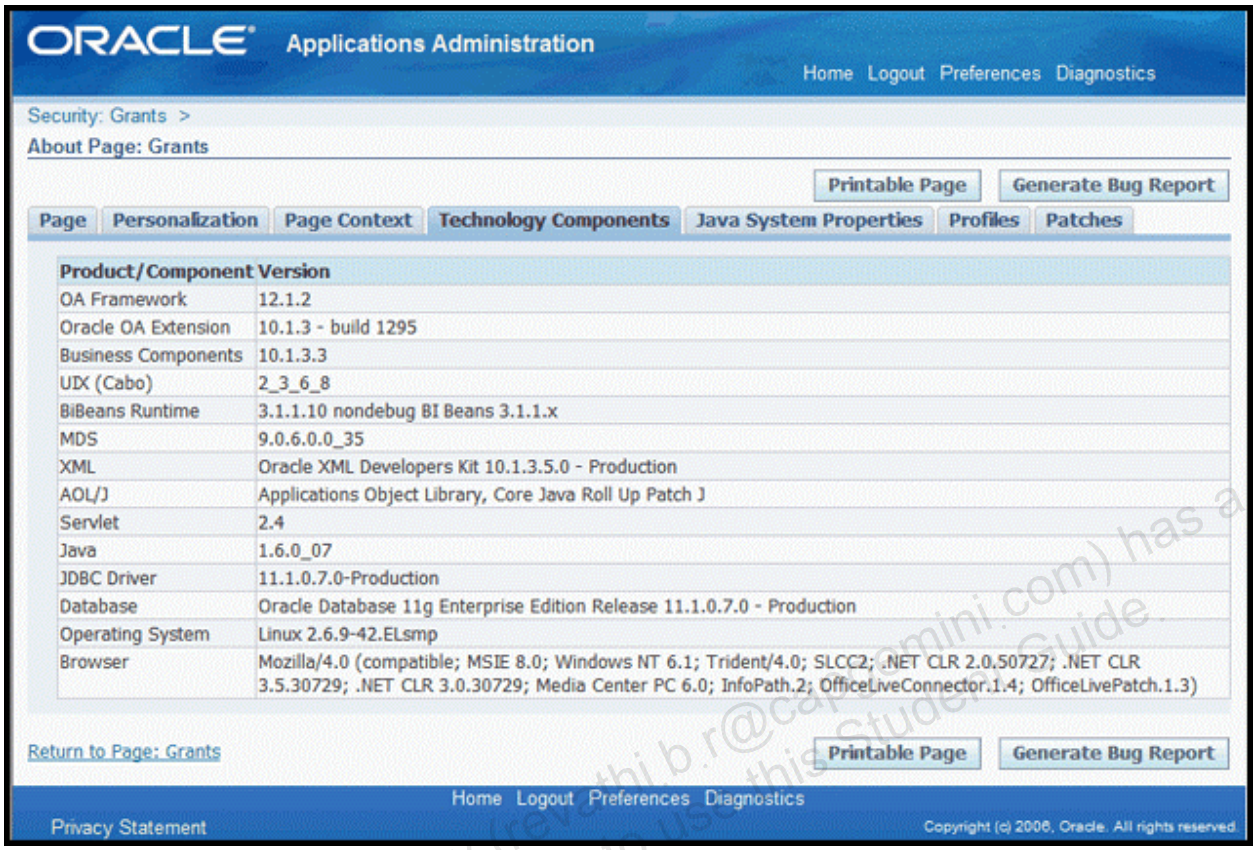


33. List the Personalizations associated with this page?
    Trick question. There are no personalizations.
34. Click the **Page Context** tab.



35. What User Name is being used on this page?
    SYSADMIN
36. What Responsibility is this page using?
    Functional Administrator
37. What is the Responsibility Key being used on this page?
    FND_FUNC_ADMIN
38. What Function is this page using?
    FND_GRANTS_SUMMARY

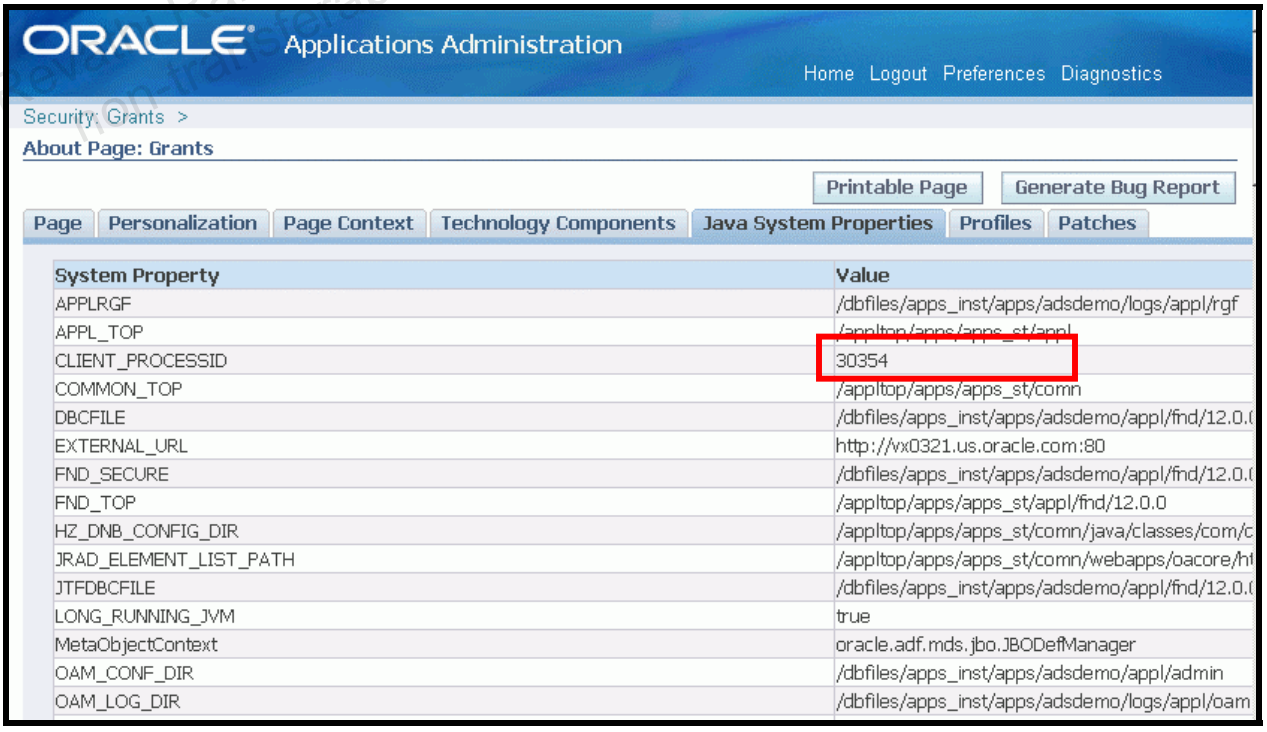
39. Click the **Technology Components** tab.



40. What version of OA Extension is being used on this server?

    10.1.3 – Build 1295

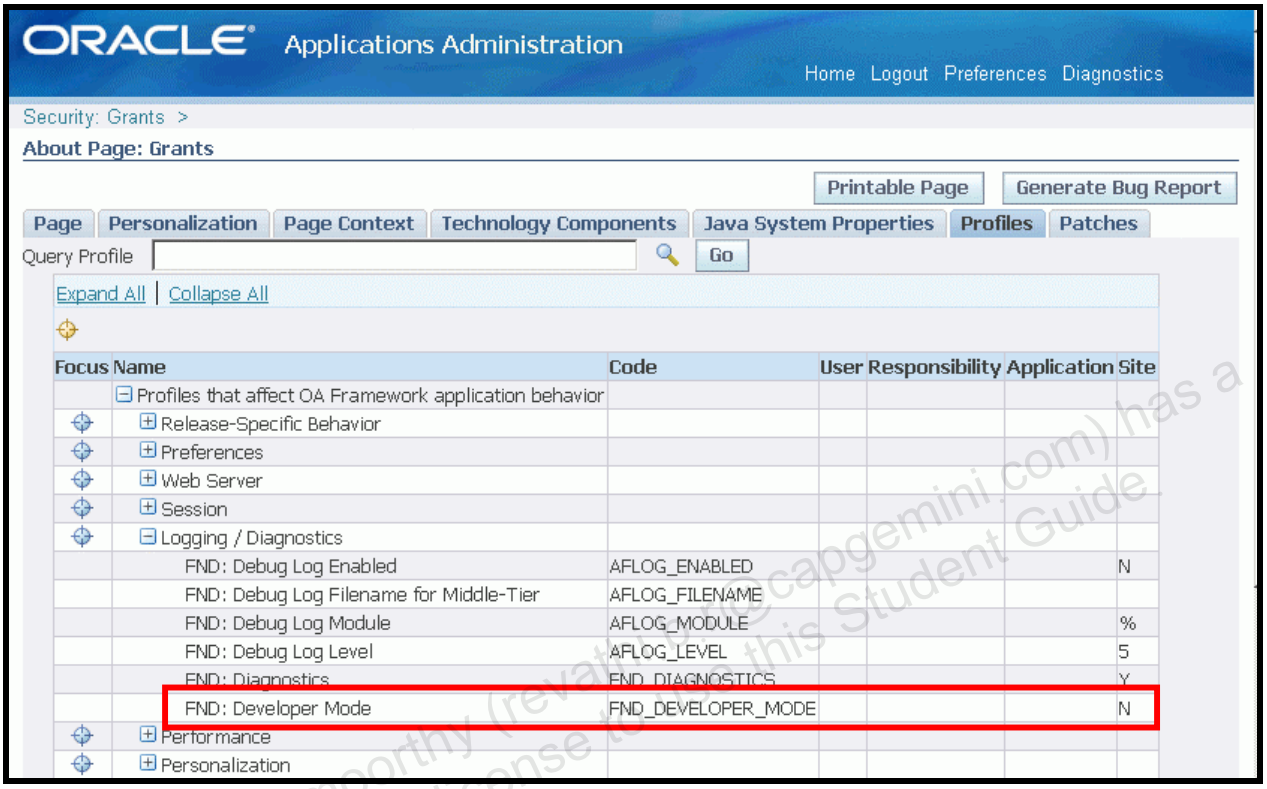
41. Click the **Java Systems Properties** tab.



42. Where does APPL_TOP point to on this server? (list the complete path)

    /appltop/apps/apps_st/appl

43. Click the **Profiles** tab.
44. Expand the **Logging/Diagnostics** region.

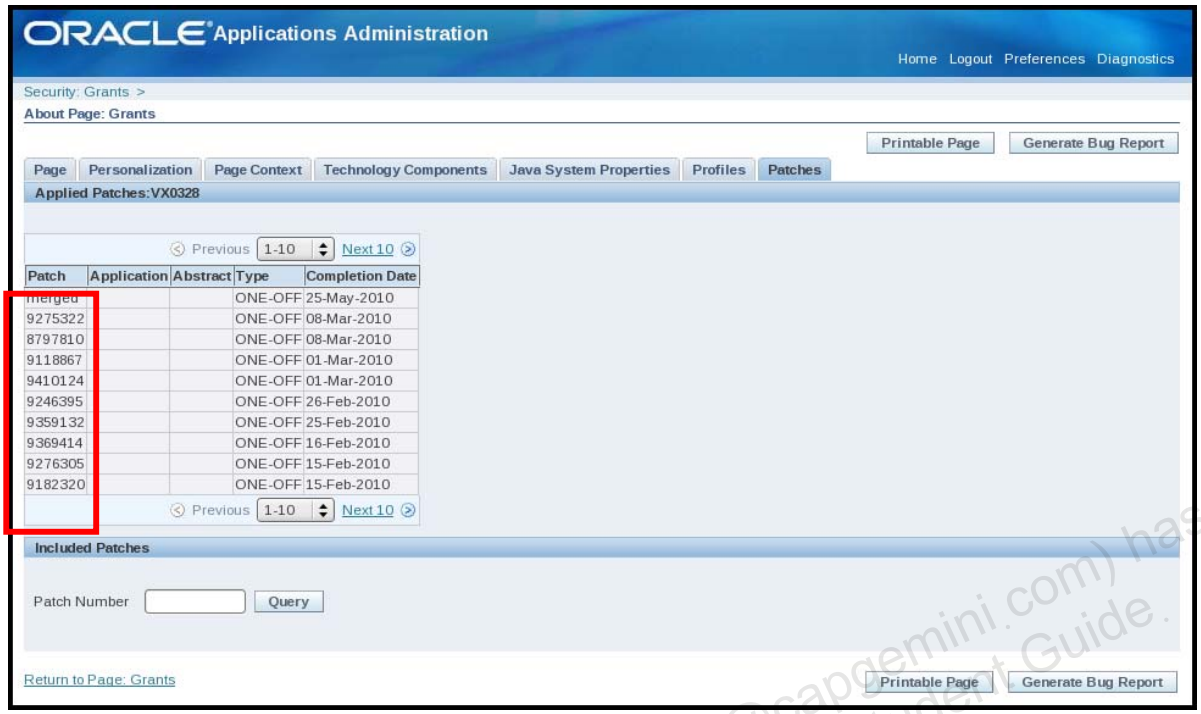


45. What is the value of the FND: Diagnostics profile option?

    Y

    Note: FND: Diagnostics is the profile option that controls whether or not you see the About this Page link. It must be set in order for you to see the link.

46. Click the **Patches** tab?



47. Have patches been applied to this E-Business Suite instance?
    YES
48. If Yes, list the last patch number applied to this server?
    Any of the patches on the left side under "Patch".
49. Click the **Return to Page: Grants** link.

You are finished with this Lab.

Lab - JDeveloper and E-Business Suite Set-up

Chapter 2

Task 1: Preparation

This lab takes you through setting up the user for your course.

Throughout this course, you will be asked to designate your name, or some other unique identifier. You can use your initials. You can use your classroom terminal number. The only requirement is that it is unique to you. Once you have clicked that designation, use it consistently throughout the remainder of the course.

Important Information:

You need to get the following from your instructor:

Your Host: _____

Your SID: _____

Your EBS Login: http://_____

Default E-Business Suite Logins:

SYSADMIN/SYSADMIN

FWKTESTER/WELCOME

Default Linux Machine Logins: (case-sensitive)

nxuser1/oracle

root/framework

applmgr/oracle

Predefined Linux Directories:

`/home/nxuser1/solutions`	- contains incremental lab solutions.
`/home/nxuser1/jdevhome`	- should be empty until the first time JDeveloper is run. If it is not empty, there may be a “system” directory pre-installed.

Task 2: Open the NX Client on Your Classroom PC

1. Your desktop should appear similar to the following:

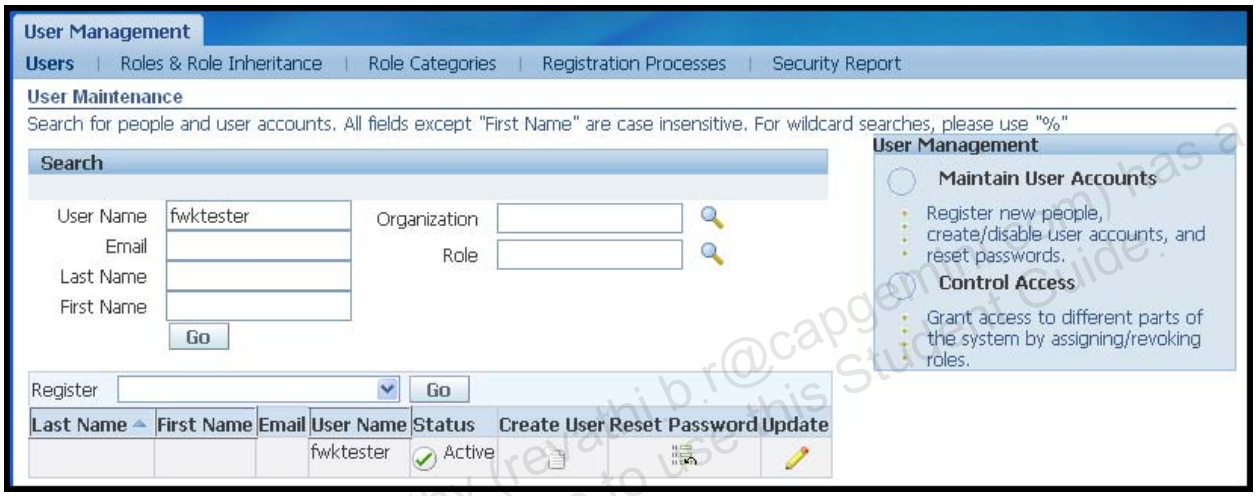
2. Double-click the Session icon (**vx0321**, as an example).

3. The Login user default is **nxuser1**, and that is correct. The password is **oracle**. Click the **Login** button to login to your server. When the NX Client finally logs in, you will see a screen similar to the following:

Task 3: Modify the FWKTESTER User

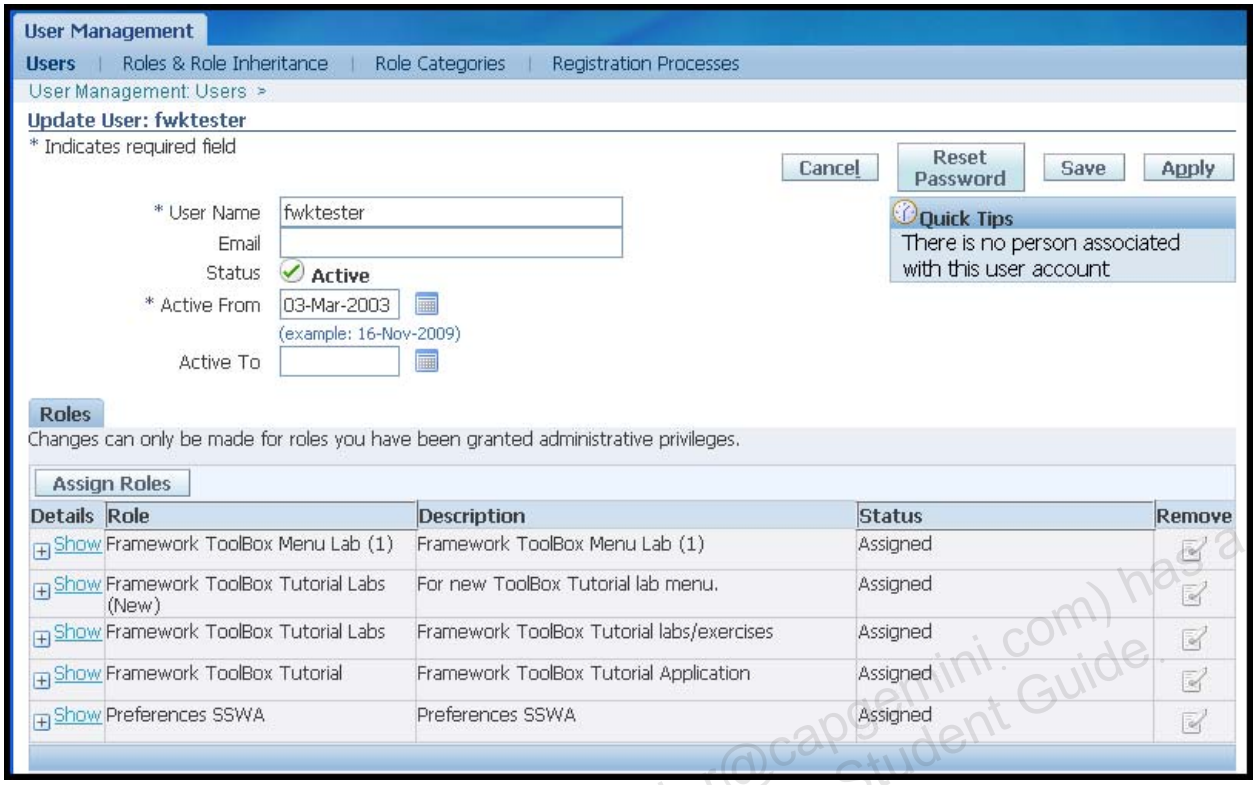
You need to modify the password for the FWKTESTER account your server. The FWKTESTER user may already be set as part of the 12.1.2 builds, but it is good practice to verify and re-set. If the roles are set, you do not have to re-set them.

1. Login to the E-Business Suite as **SYSADMIN**/**SYSADMIN**
2. From the Personal Home Page, click the **User Management** responsibility
3. From the User Management menu, click **Users**.
4. When the User Maintenance page opens, enter **FWKTESTER** in the **User Name** field.
5. Click the **Go** button located under the First Name field. This will execute the query to return the user FWKTESTER information to the results table.



6. Click the **Reset Password** icon in the results table.
7. In the Password field enter **ORACLE** and repeat the password in the **Confirm Password** field.

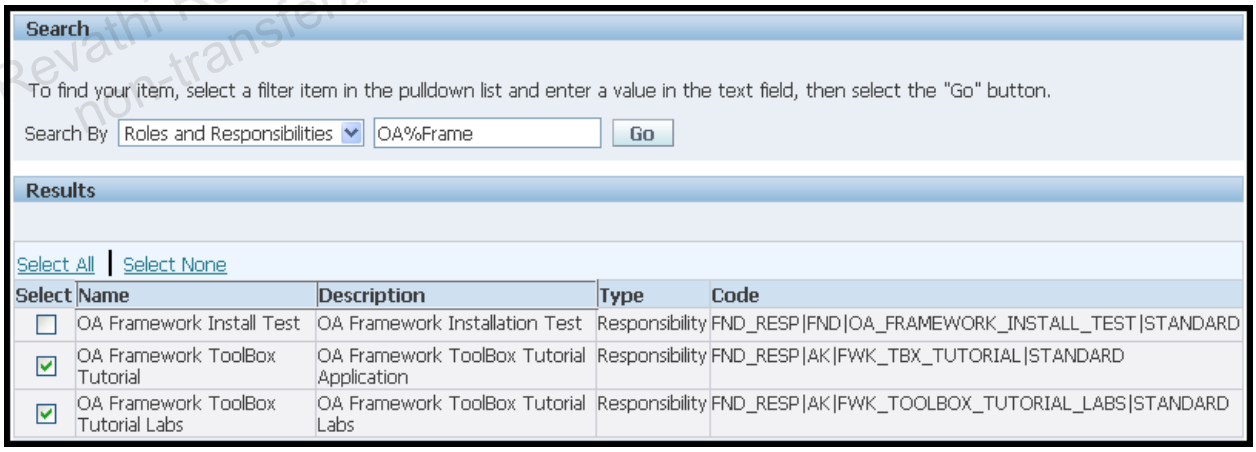
   Note: ORACLE can be lowercase, uppercase, or mixed case, just remember what you entered.
8. Click the **Submit** button to process the password reset. Upon confirmation you will be redirected back to the User Maintenance page.
9. Enter **FWKTESTER** in the **User Name** field, and click the **Go** button.
10. Click the **Update** pencil icon.



11. On the Update User page, click the **Assign Roles** button.
12. In the Search popup, check that the **Search By** criteria is set to **Roles and Responsibilities**. Enter **OA%Frame** in the **Search By** field, and click the **Go** button.
13. Click the checkbox in the results table for the following two roles:

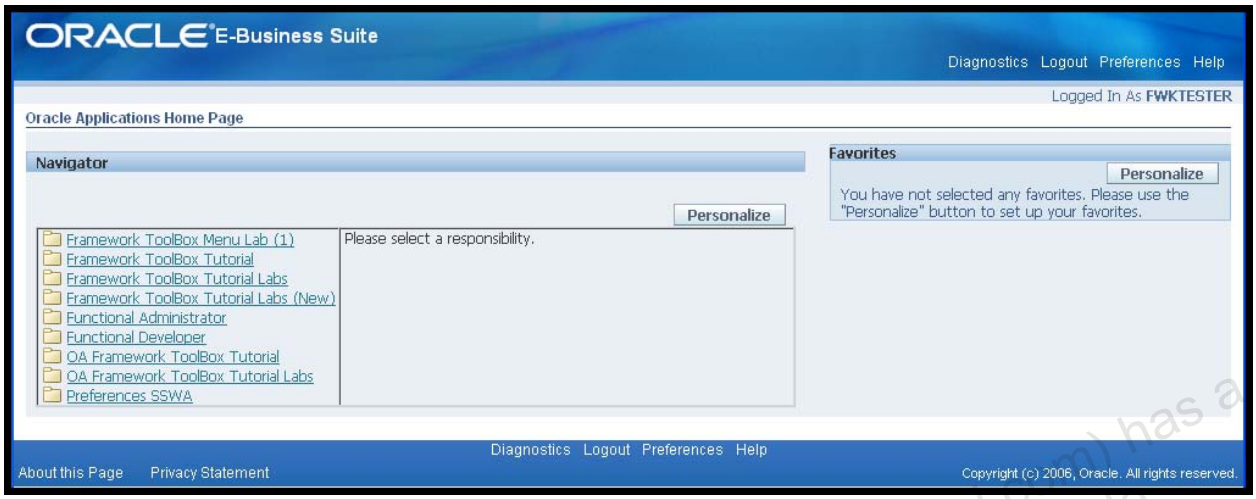
    OA Framework ToolBox Tutorial Labs

    OA Framework ToolBox Tutorial



14. Click the **Select** button on the Results table.
15. The roles will be added to the FWKTESTER user. You must enter a justification for the assignment for each role to the user. Enter a statement such as **Assigned for class project**.
16. Click the **Apply** button to save your changes.
17. Click **Logout** to logout of the E-Business Suite instance.
18. Login to the E-Business Suite instance as **FWKTESTER**/**ORACLE**.

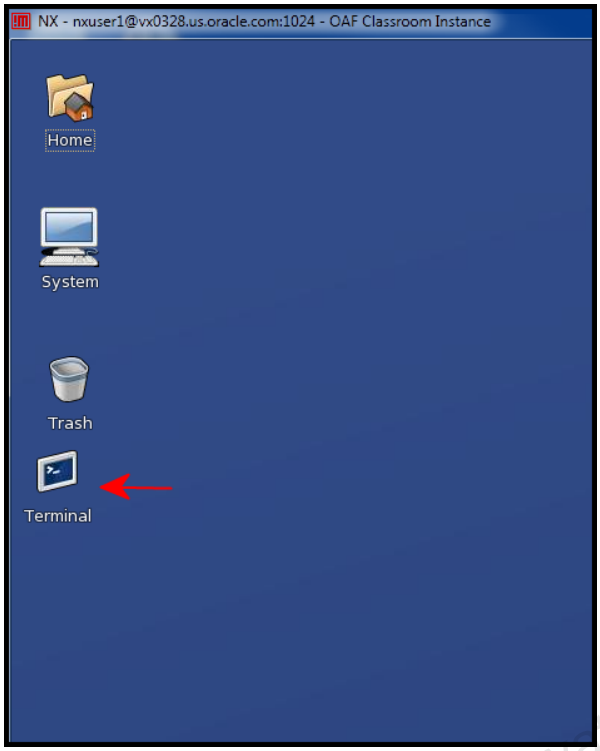
19. You are logging in to the FWKTESTER responsibility for the first time (after you reset the password). The E-Business Suite will require you to change the password. Change the password to **FWKDEV**. Remember how you set the password.
20. You go to the E-Business Suite home page, which should look similar to the following:

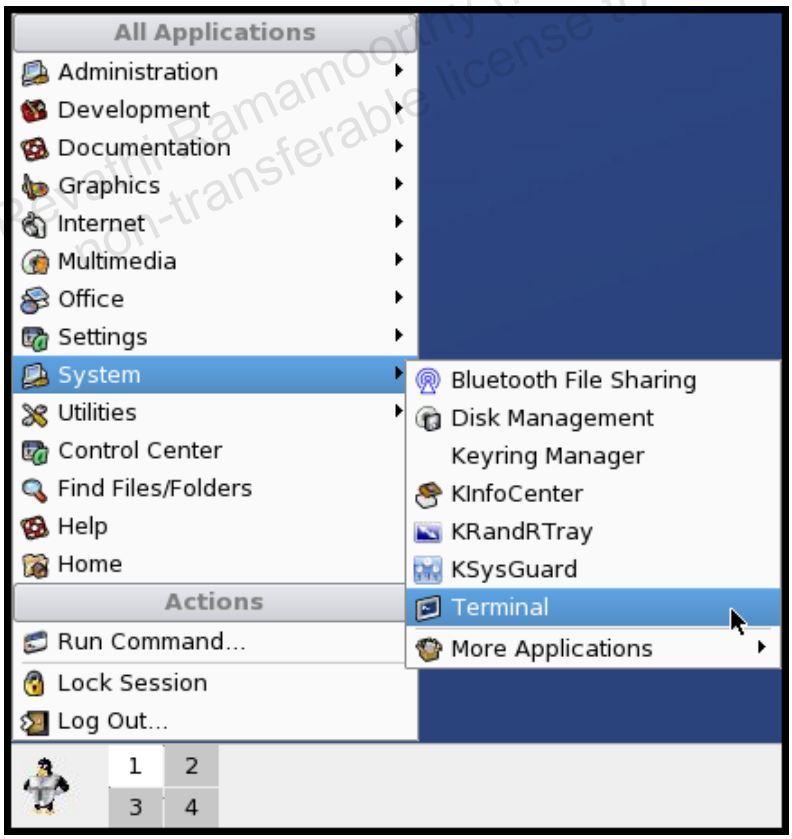


21. Click **Logout**.

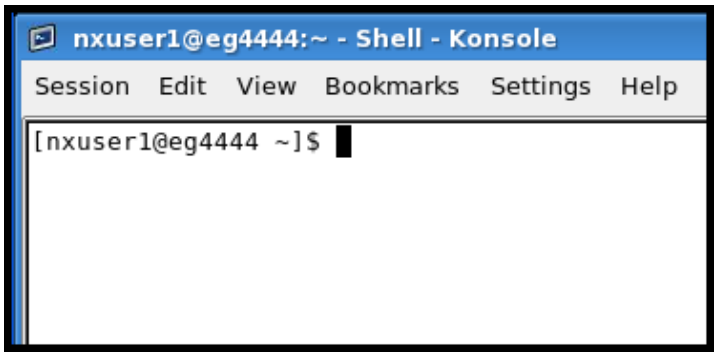
Task 4: Unzip Tutorial.zip

1. Open a terminal window. Click the **Terminal** icon or use the toolbar



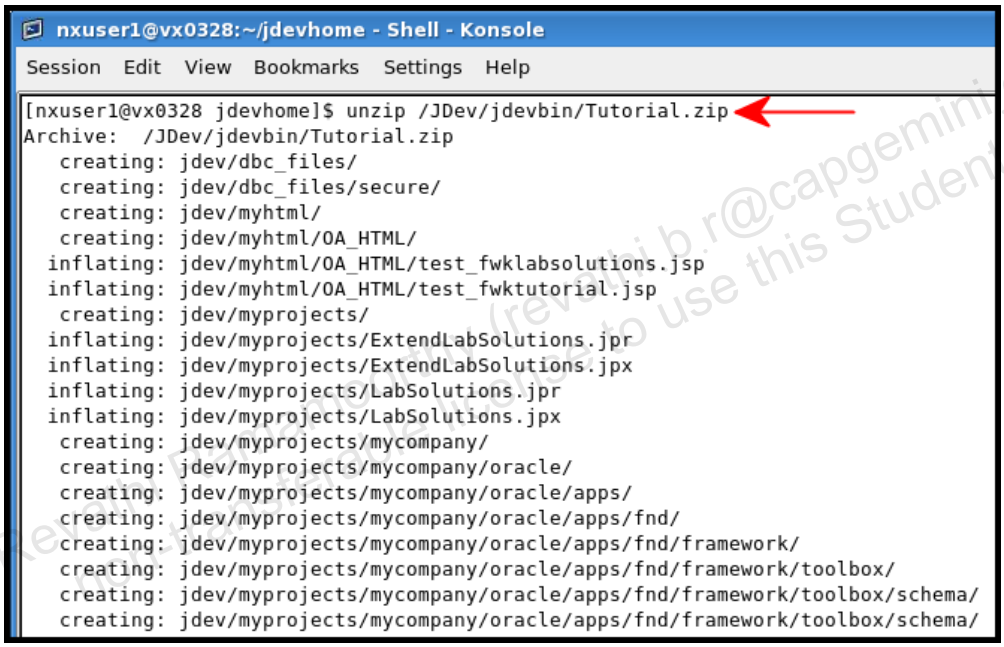


2. A terminal window will open, and it will look similar to the following:



3. Your base path will be set to `/home/nxuser1`. At the command prompt, type the following commands:

```
[nxuser1@vx0328 ~]$ cd jdevhome
[nxuser1@vx0328 jdevhome]$ unzip /JDev/jdevbin/Tutorial.zip
```



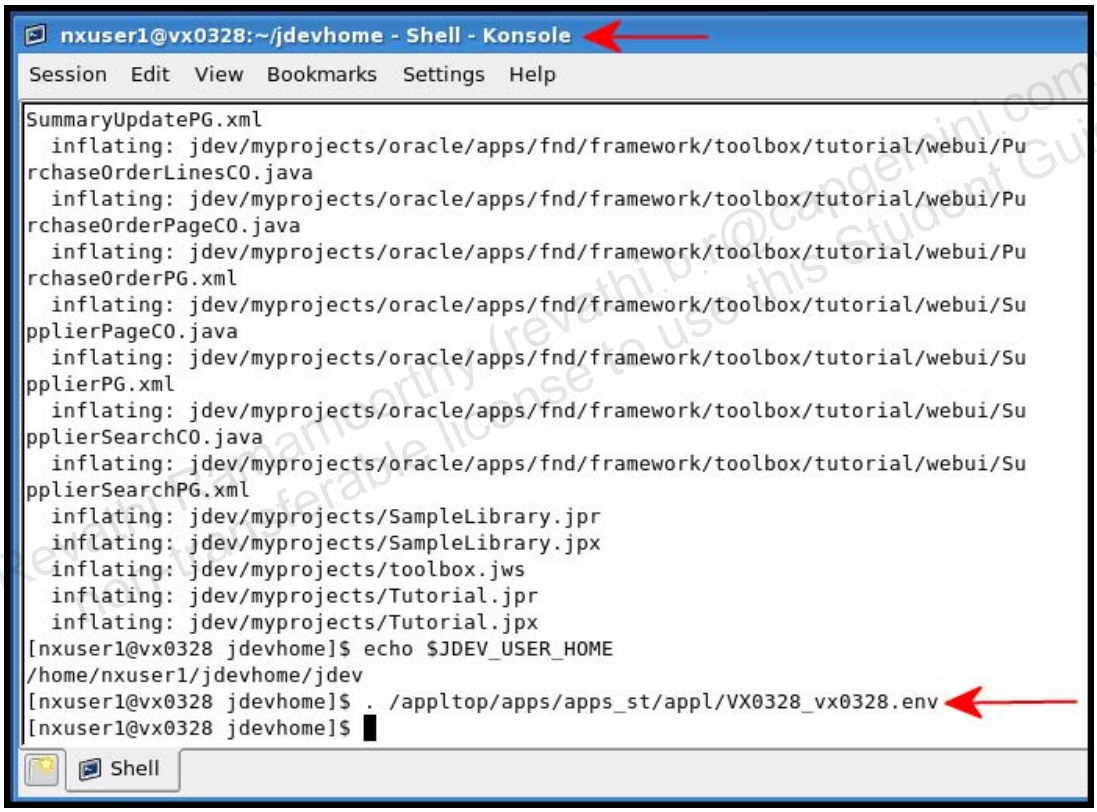
Task 5: Set Your JDeveloper Environment

1. Continuing in the same terminal window, type the following command:
   `echo $JDEV_USER_HOME`
2. If the return value is null, type the following command:
   `export JDEV_USER_HOME=/home/nxuser1/jdevhome/jdev`
3. You can also set your E-Business Suite environment at this point. In the terminal window, input the following command (case sensitive):
   `. /appltop/apps/apps_st/appl/`*`<SID>_<servername>`***`.env`**

   Note: See Task 1 for the value to substitute for <*SID*>.

   Example: `. /appltop/apps/apps_st/appl/VX0328_vx0328.env`

   Note: While difficult to discern in print, there is a <space> in the command shown between the . (period) and the `/appltop` directory path. Also, your ENV file may be different than the one shown below.



Task 6: Start JDeveloper

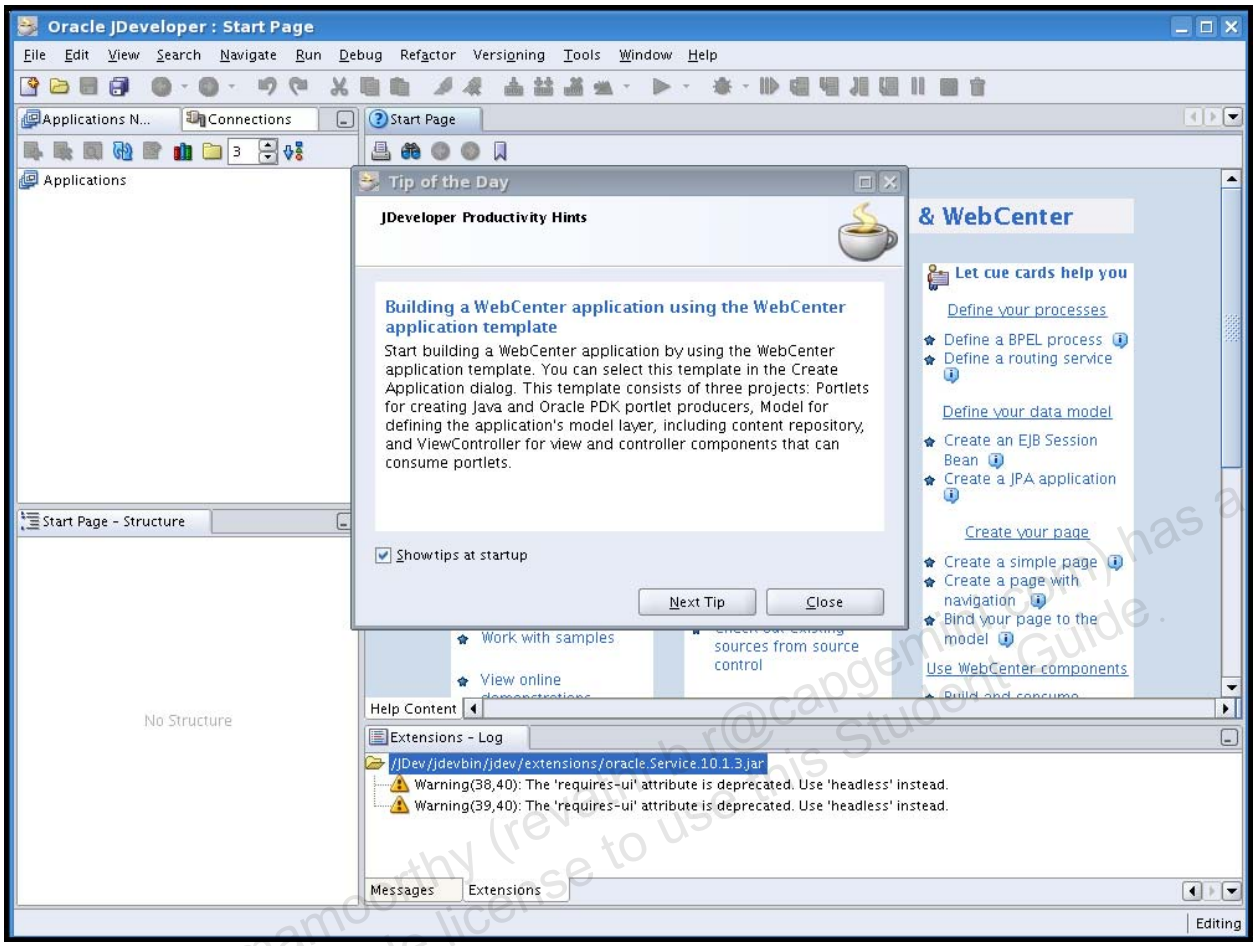
1. Continuing in the same terminal window, type the following command:

   ```
   /JDev/jdevbin/jdev/bin/jdev&
   ```

   Note: It is important to include the ampersand (&) at the end of the command to allow your terminal window to continue to be used.

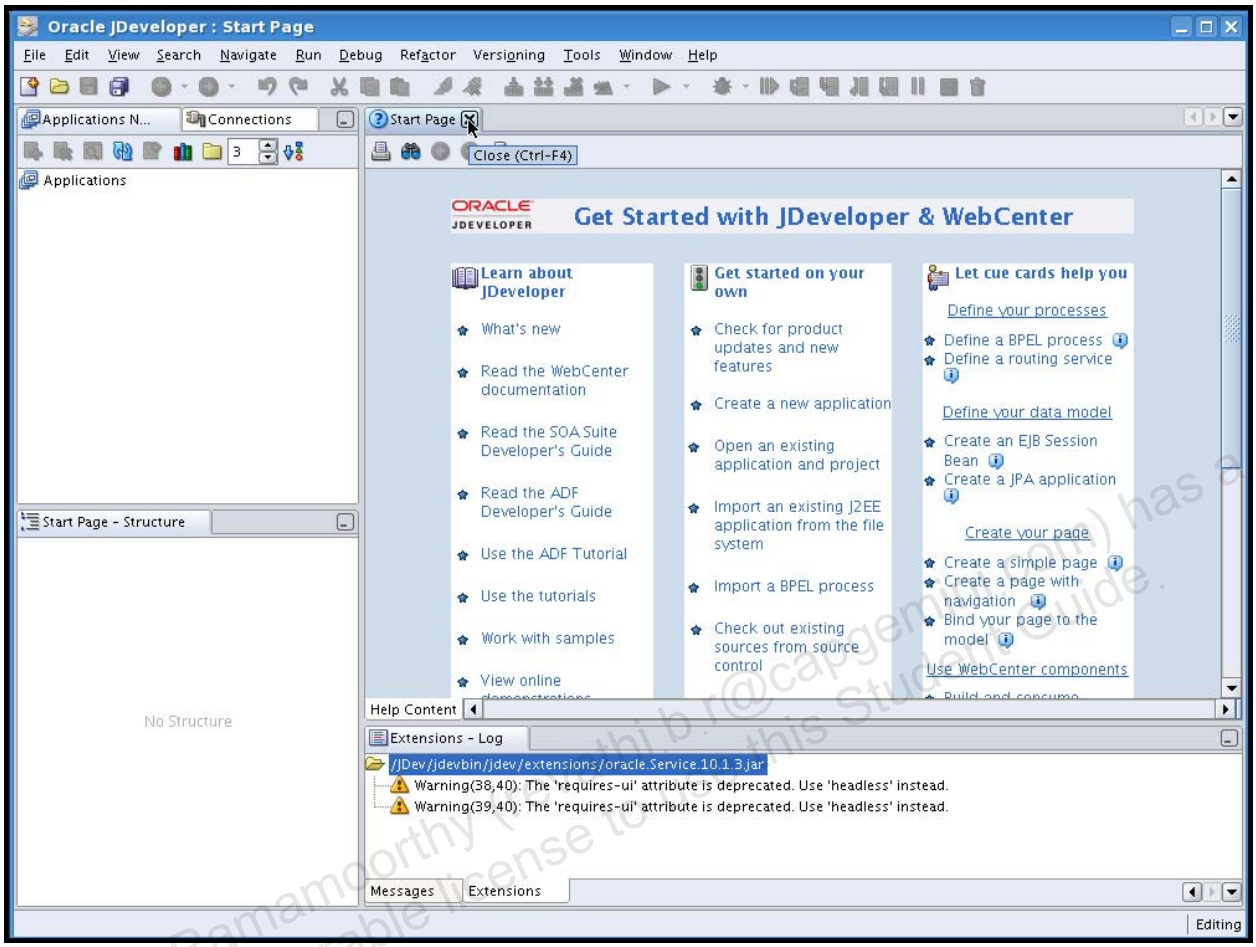
2. If the migration window appears, click the **No** button. Clicking **Yes** will overwrite any files that are present. The very first time JDeveloper runs, you will see the window. In those cases, there is no problem with clicking Yes.

3. JDeveloper 10g with OA Extension will load, and it will appear similar to the following:



4. Uncheck the **Show tips at startup** checkbox.
5. Click the **Close** button.

6. Move your cursor over the **Start Page** tab, and the **[X] (Close)** icon will appear. Click the **[X]** icon to close the Start Page.



7. Minimize JDeveloper.

Task 7: Get the DBC File

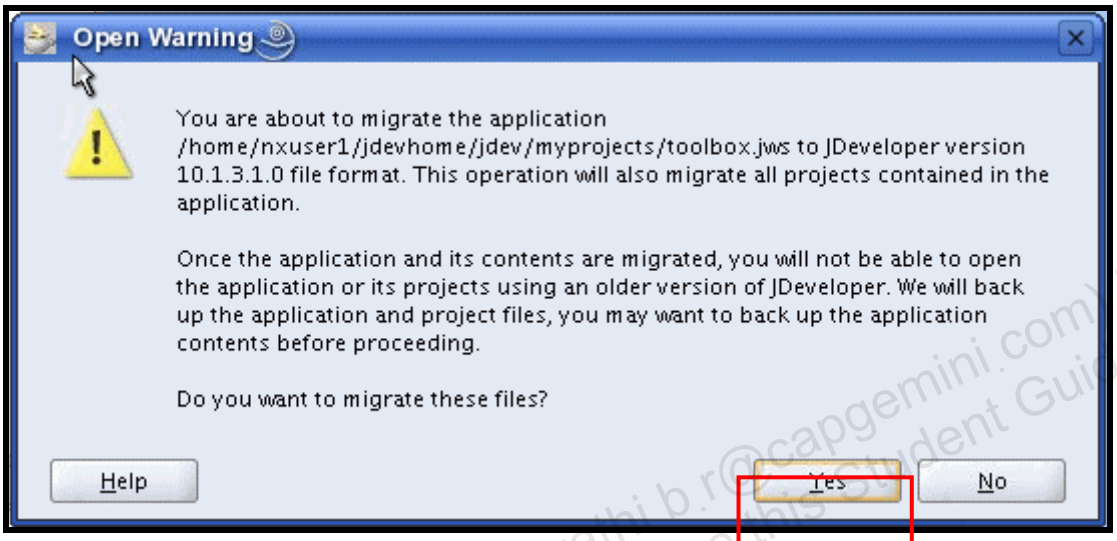
1. In the terminal window, type the following commands:

```
cd $JDEV_USER_HOME/dbc_files/secure
su root
Password: framework
cp $INST_TOP/appl/fnd/12.0.0/secure/<SID>.dbc ./
chown nxuser1:oinstall <SID>.dbc
exit
```

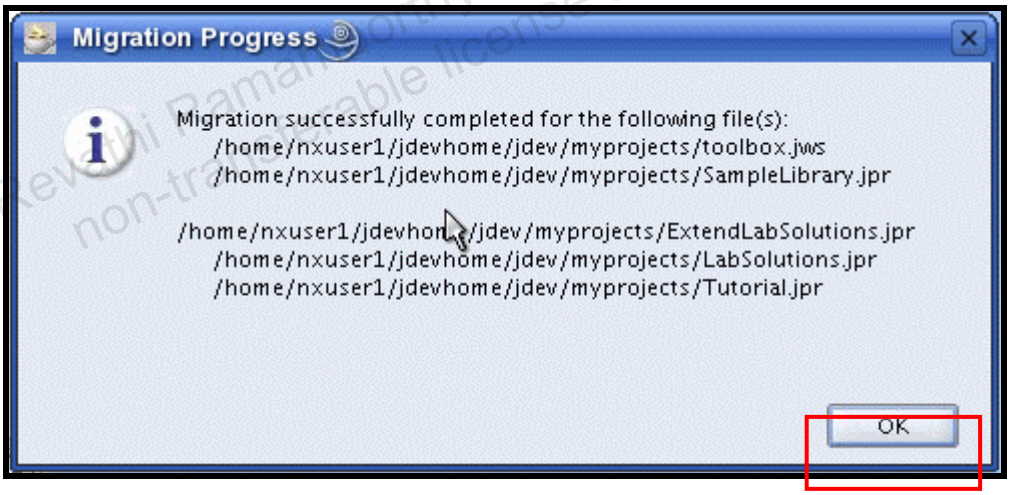
Note: See Task 1 for the value to substitute for <*SID*>.

Task 8: Test Your JDeveloper Workspace and Project

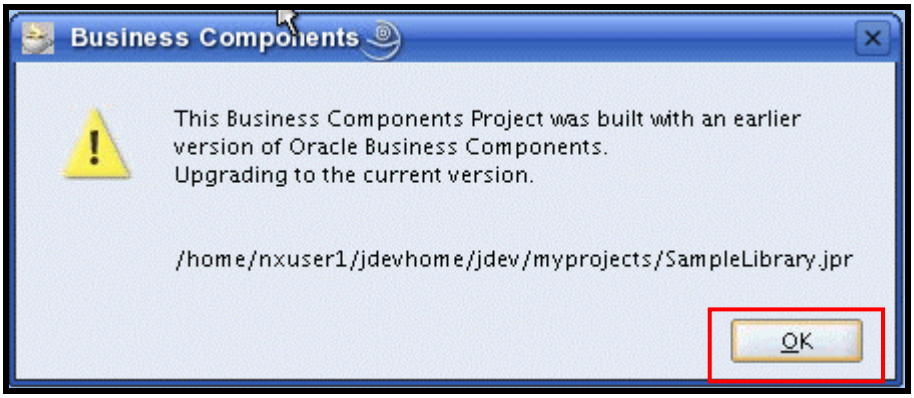
1. Maximize JDeveloper.
2. From within JDeveloper, click (Menu) File > Open.
3. Double-click the **myprojects** folder icon to drill-down to that directory.
4. Double-click the **toolbox.jws** file icon to open that specific workspace.
5. When you open the workspace, you may encounter the Migration Warning window. If you do, click the **Yes** button to continue.



6. If you encountered the Migration Warning, when the migration is complete you will see the Migration Progress window. Click the **OK** button to continue.

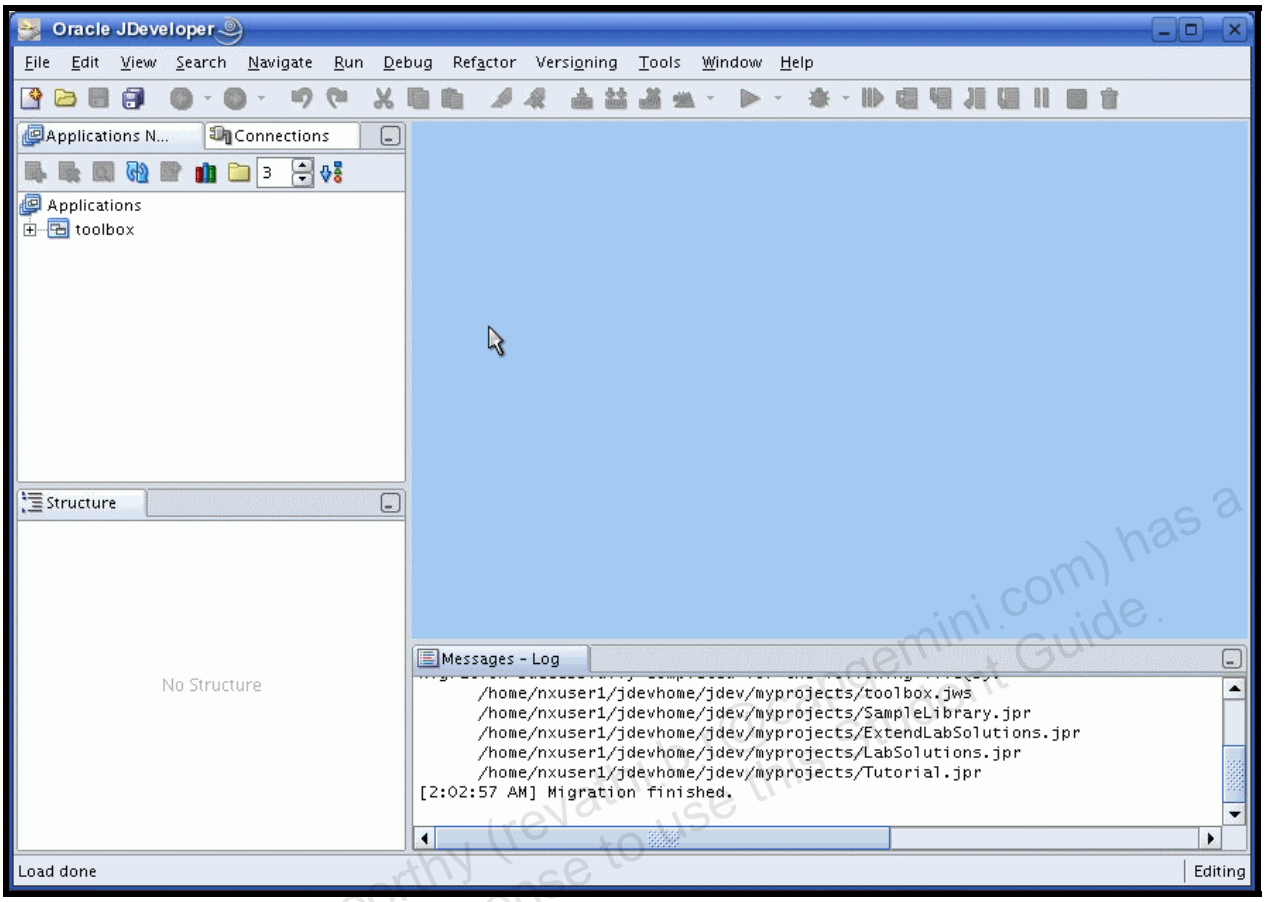


7. You may encounter the Business Components Upgrade window. If you do, click the **OK** button to continue.

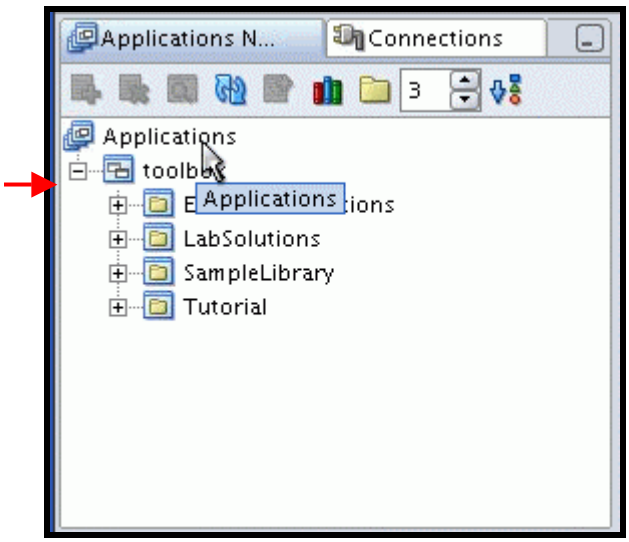


8. If you encountered the Business Components Upgrade window, when the upgrade is complete you will encounter the Connections warning. This is expected behavior. You have not yet defined a database connection for JDeveloper to test your BC4J objects. Defining this database connection will be the next step you perform. Click the **OK** button to continue.

9. JDeveloper will open the toolbox workspace. It will appear similar to the following:



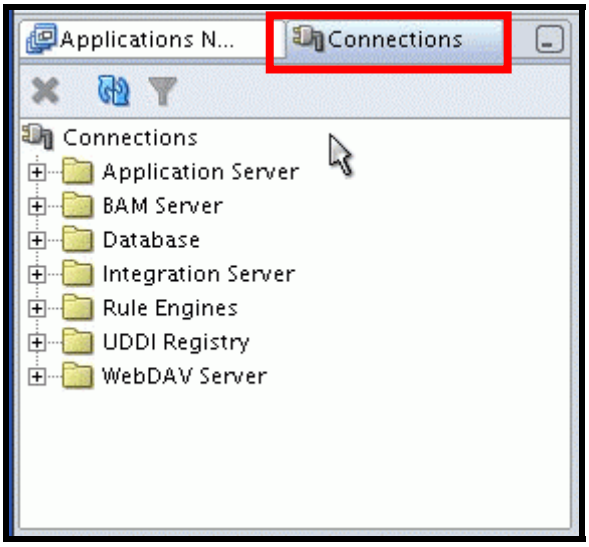
10. Expand the **toolbox** workspace. Click the **[+]** icon beside **toolbox** in the Applications Navigator window.



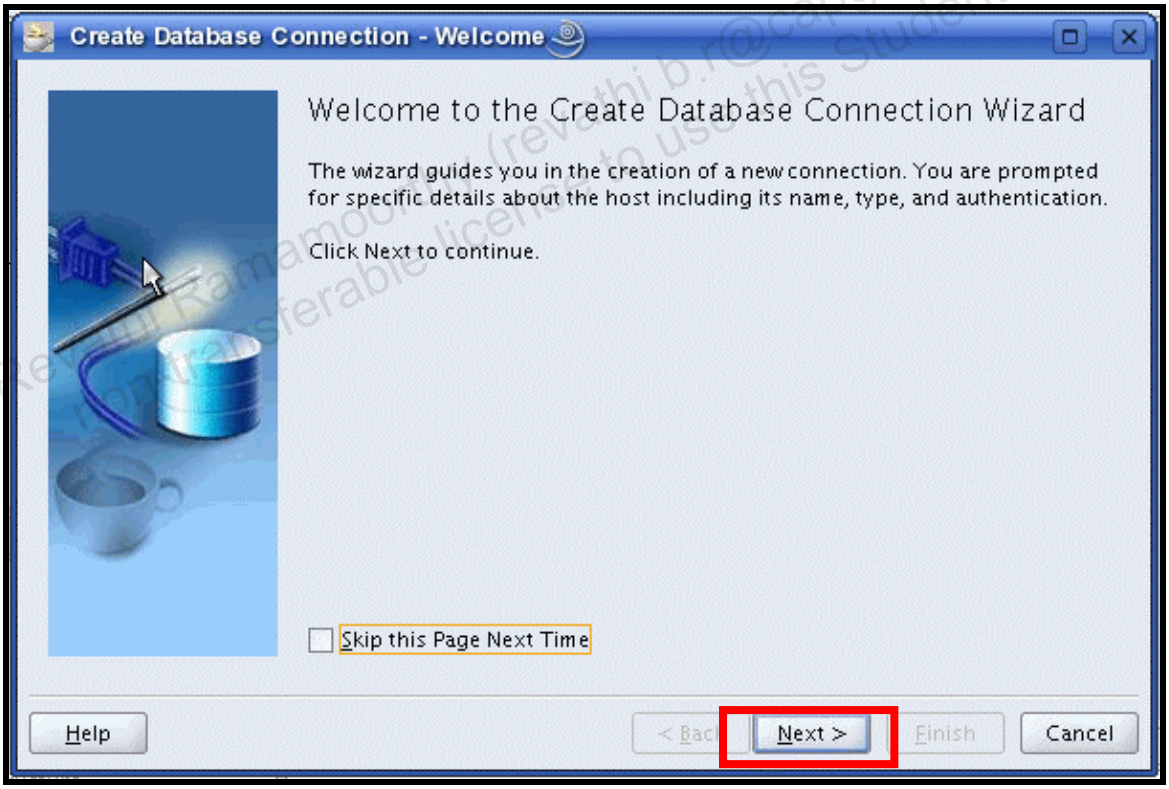
Note: Everything in the **toolbox** is legacy solutions and samples. These solutions and samples are included for information only. **LabSolutions** in the **toolbox** workspace is NOT the lab solutions for this course. Copying and pasting anything from **LabSolutions** will corrupt your work in this course.

Task 9: Set-up Your Database Connection

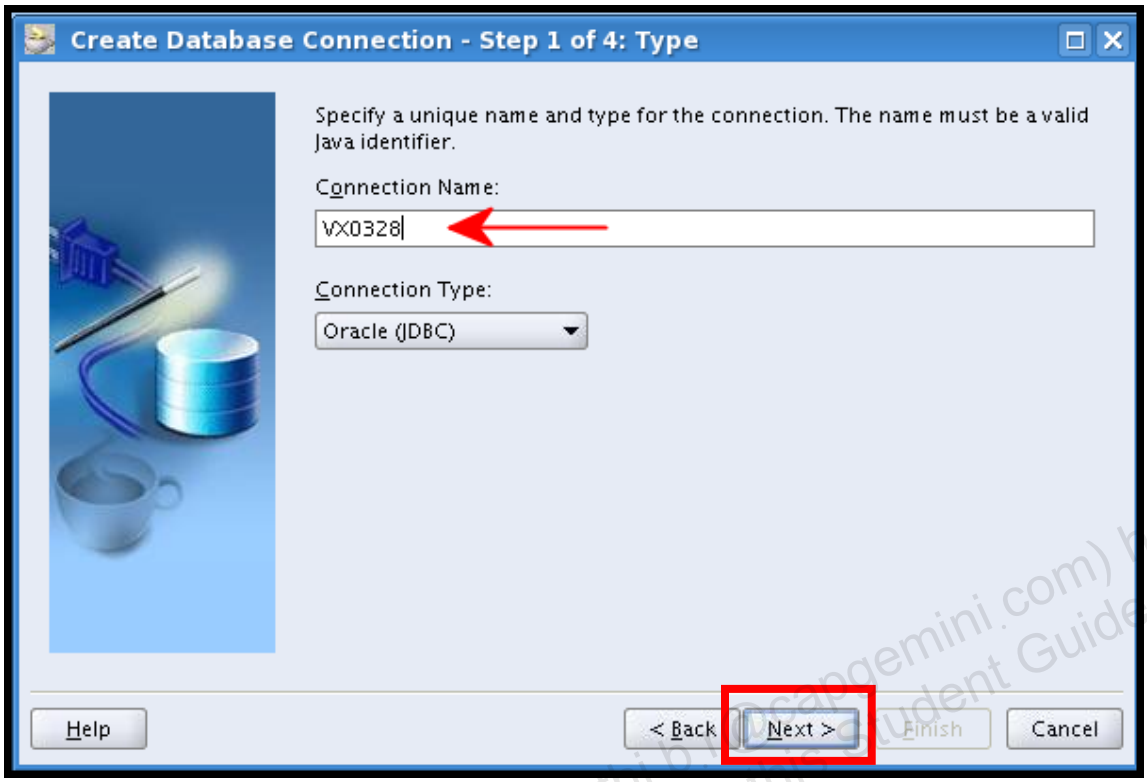
1. Click the **Connections** tab.



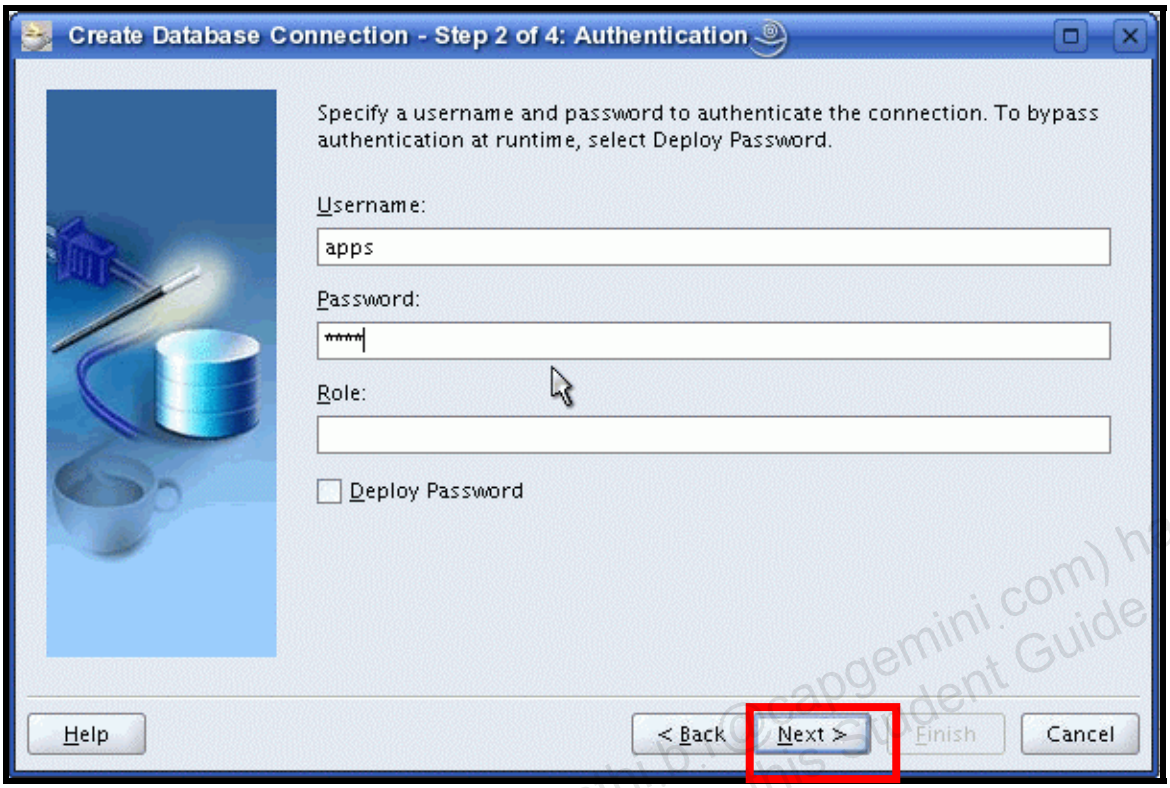
2. Right-click the **Database** folder, and click **New Database Connection**.
3. The Create Database Connection wizard will open. Click the **Next** button to continue.



4. Set the Connection name to your SID.

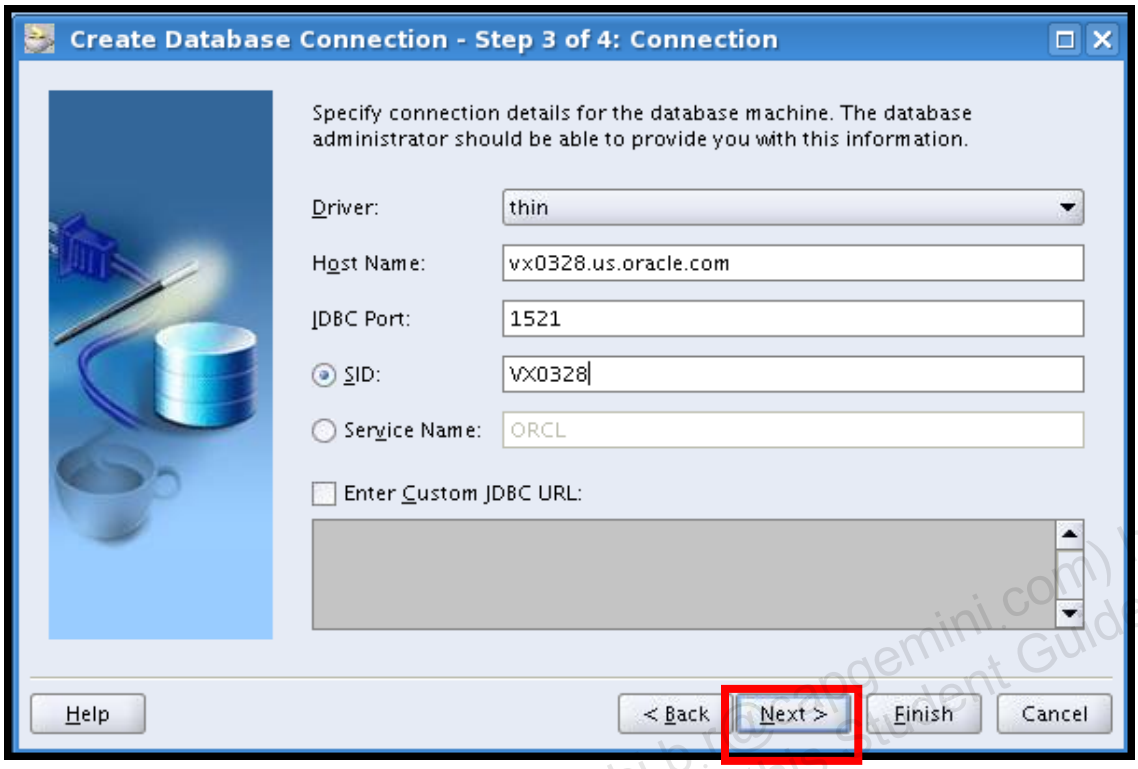


5. Click the **Next** button.



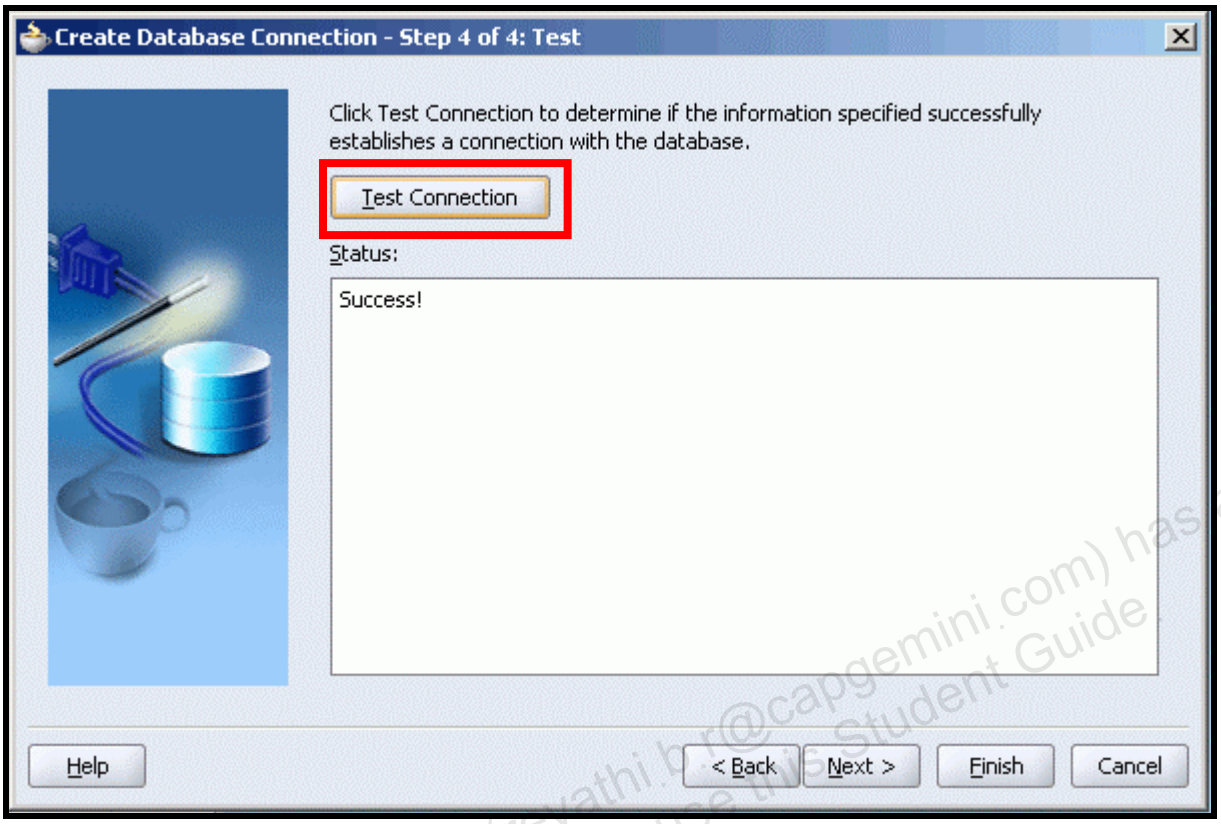
6. Set the **Username** to **apps**.
7. Set the **Password** to **apps**.

8. Click the **Next** button.



9. Set the **Host Name** to your host name.
10. Set the **SID** to your SID.
11. Click the **Next** button.

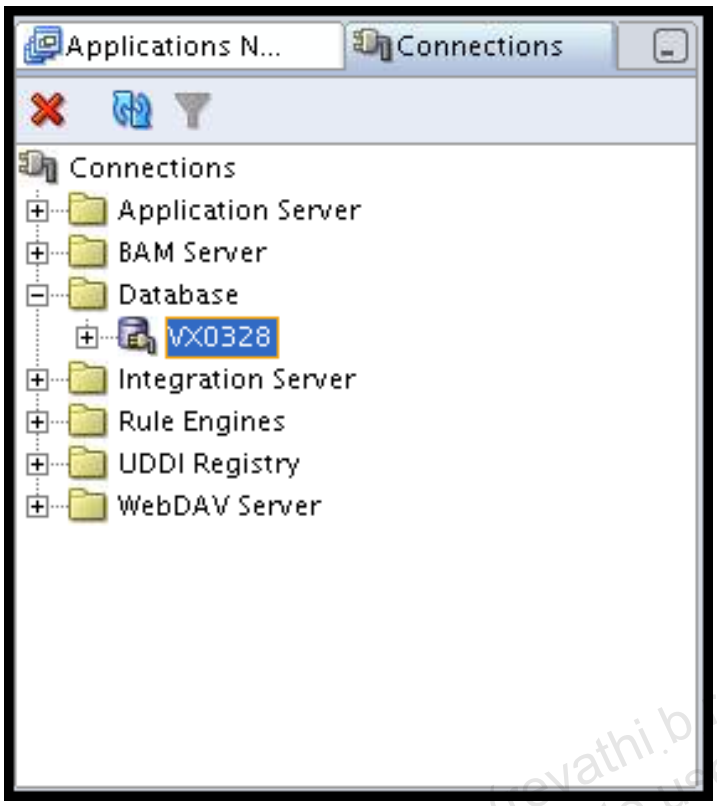
12. Click the **Test Connection** button.



Note: You should see a **Success!** message in the Status: window. If you do not, click the **Back** button. Resolve any errors in the settings, and return to the test screen and click the **Test Connection** button. Do not click the **Finish** button until you receive the **Success!** message.

13. If the test was successful, click the **Finish** button.

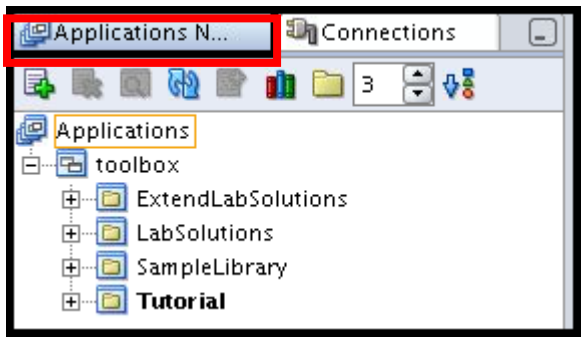
14. When you exit the Create Database Connection wizard, you will return to the Connections panel. It will appear as follows:



Note: You have two connections. The connection you just created (**VX0328**, as an example) and **fwk12dev**. The latter is a default connection provided in JDeveloper with the OA Extension. It will not work. You can delete this connection if you like. Otherwise, just ignore it. In the image above, it has been deleted.

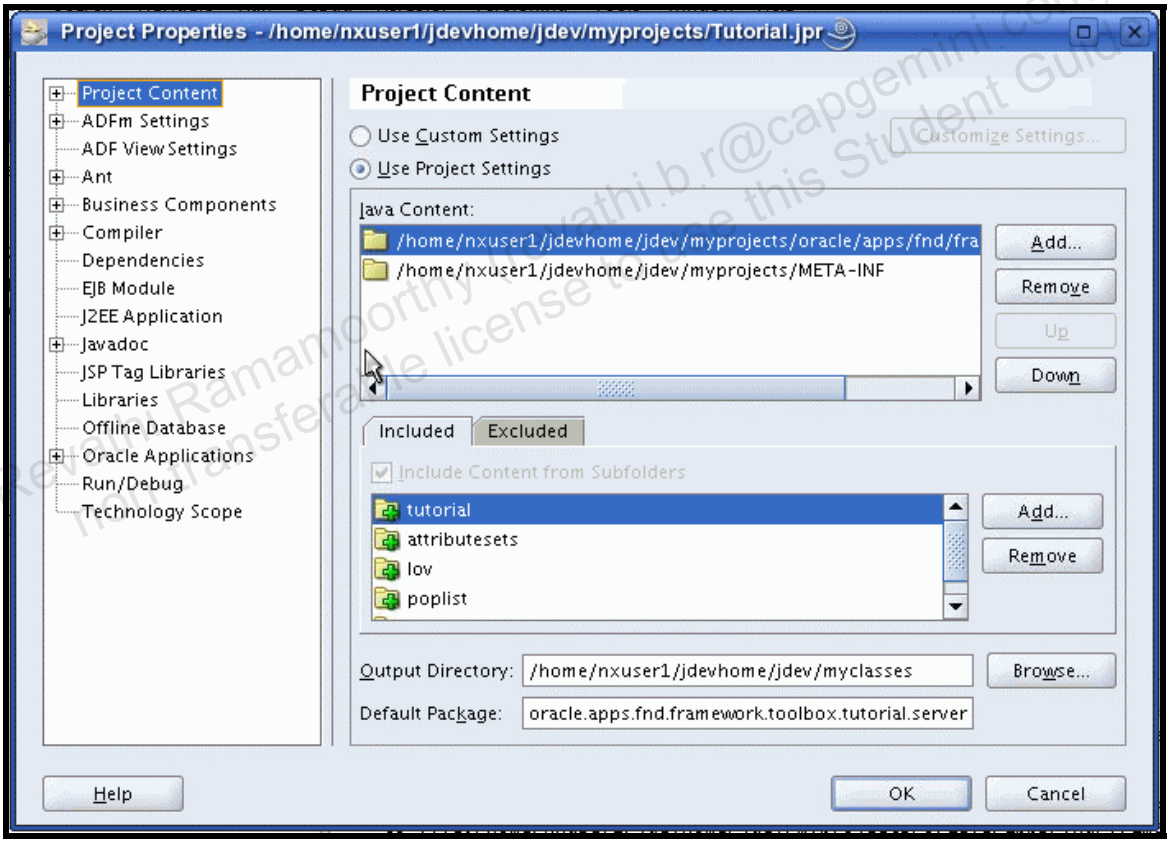
Task 10: Set Your Project Properties

1. Click the **Applications Navigator** tab.

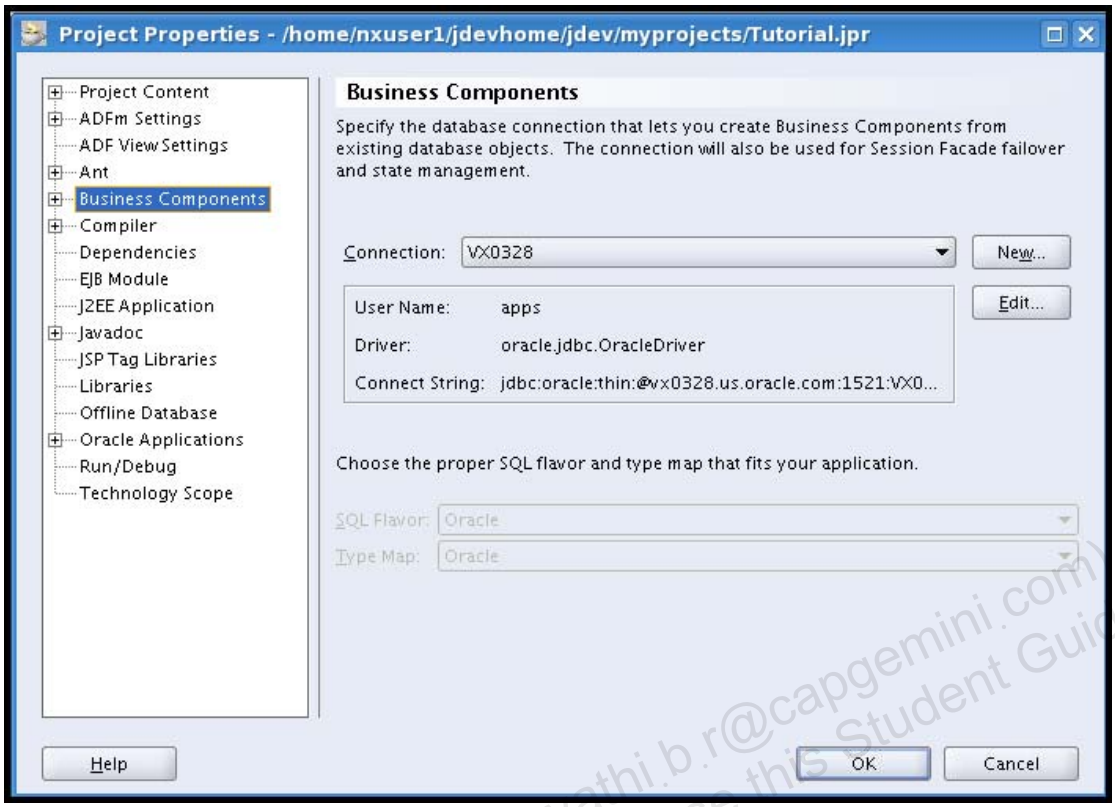


2. Right-click the **Tutorial** project, and click **Project Properties**. This will open the Project Properties window, which appears similar to the following:

   Note: If migration windows -- similar to the Business Components windows pictured in Task 8 appear -- click the **OK** button. You can safely ignore any warning messages; you may need to re-open the Project Properties window.

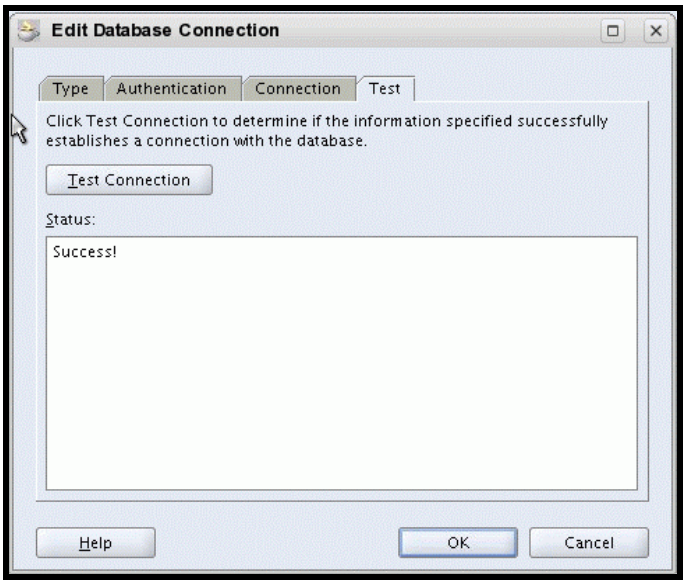


3. Click the **Business Components** category. It will appear similar to the following:



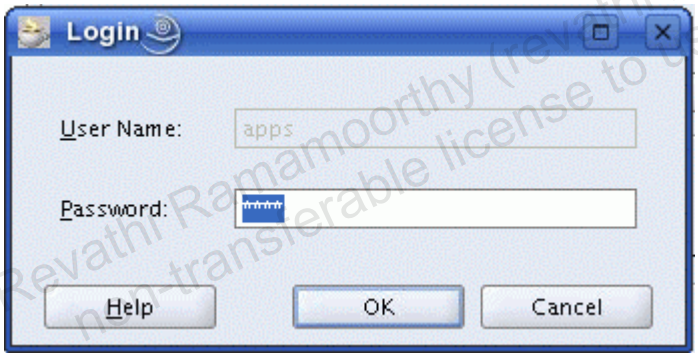
4. Click your Connection (for example, **VX0328** from the poplist).
5. Click the **Edit…** button.
6. When the Edit Database Connection window opens, click the **Test** tab.

7. Click the **Test Connection** button. You should see the **Success!** message.

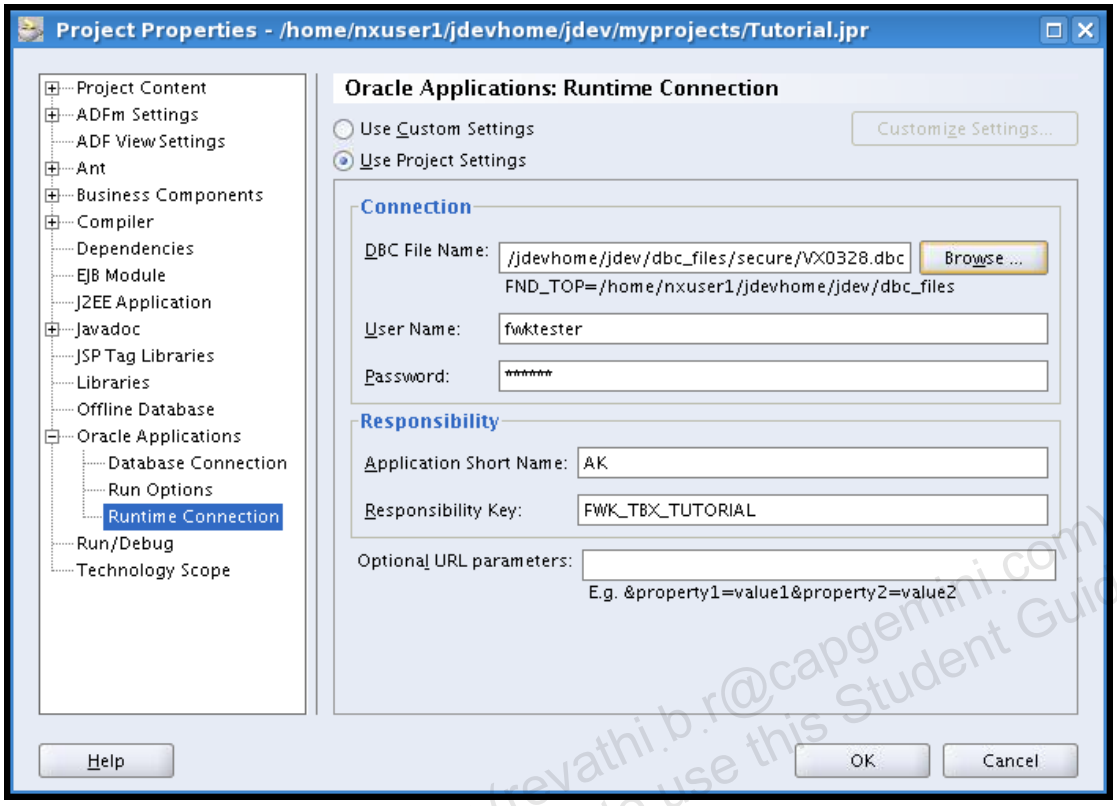


8. Click the **OK** button.
9. Find and expand the **Oracle Applications** category by clicking the **[+]** icon.

   Note: When you expand this category for the first time, you may get the following Login window. Accept the defaults -- **User Name** is **apps**, and **Password** is **apps**. Click the **OK** button.



10. Click the **Runtime Connection** subcategory. Your screen should look similar to the following:



11. The DBC File Name field is initially blank, or it will contain a path that is incorrect. You can either input the path directly, or you can browse for the specific DBC file. The field should contain:

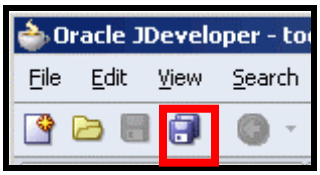
    `/home/nxuser1/jdevhome/jdev/dbc_files/secure/`*`<SID>`***`.dbc`**

12. The **User Name** is **FWKTESTER**.
13. The **Password** (case-sensitive) is **FWKDEV**.
14. The **Application Short Name** is **AK**.
15. The Responsibility Key is **FWK_TBX_TUTORIAL**.

    Note: A full discussion of E-Business Suite security is beyond the scope of this course. At this point, it is enough to know two things. One, the values will work. Two, the Application Short Name and Responsibility Key could be any value provided that the User Name being referenced has those values in its definition.

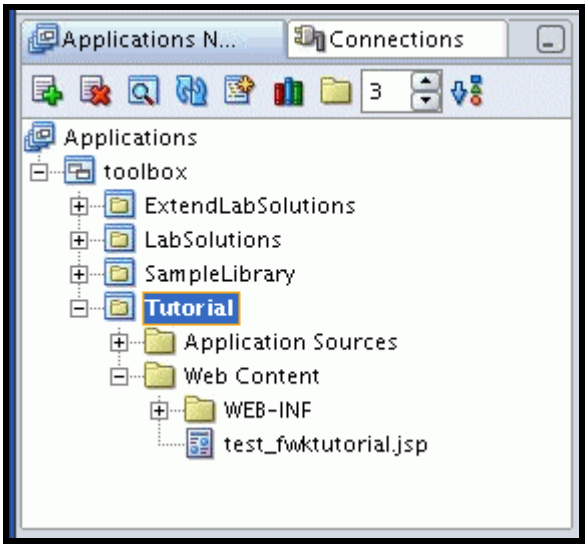
16. Click the **OK** button.

17. Click the **Save All** icon, or use **(Menu) File > Save All**, to save your work.

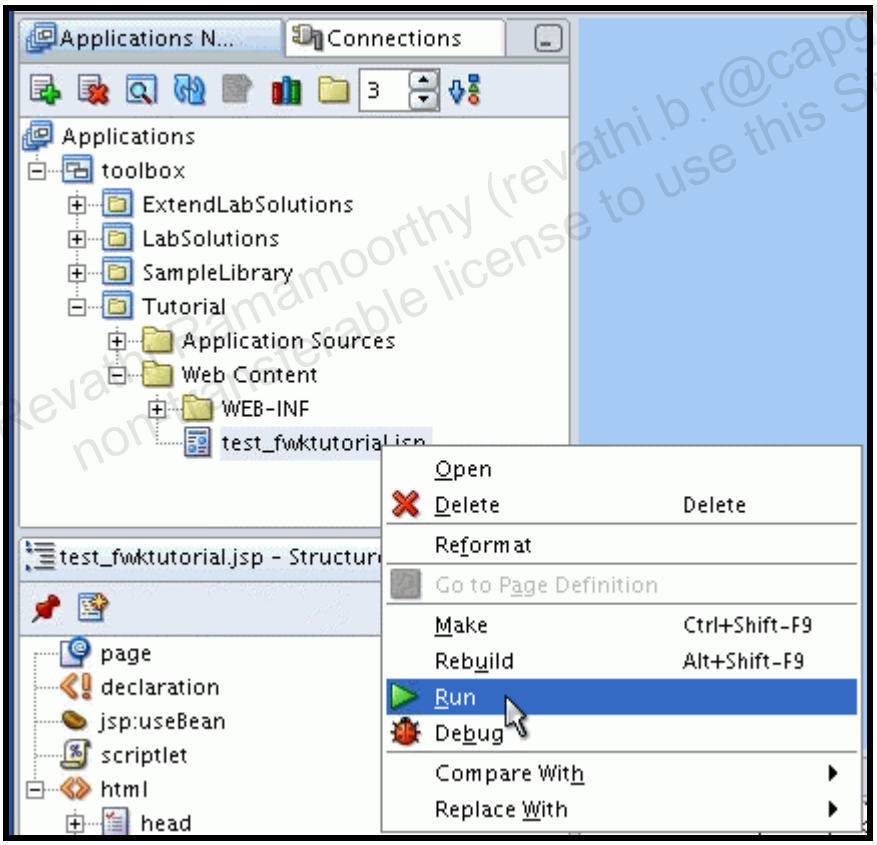


Task 11: Test Your JDeveloper Set-up

1. Expand the **Tutorial** project, then the **Web Content** folder by clicking the **[+]** icons.



2. Right-click the **test_fwktutorial.jsp** file, and choose **Run**.

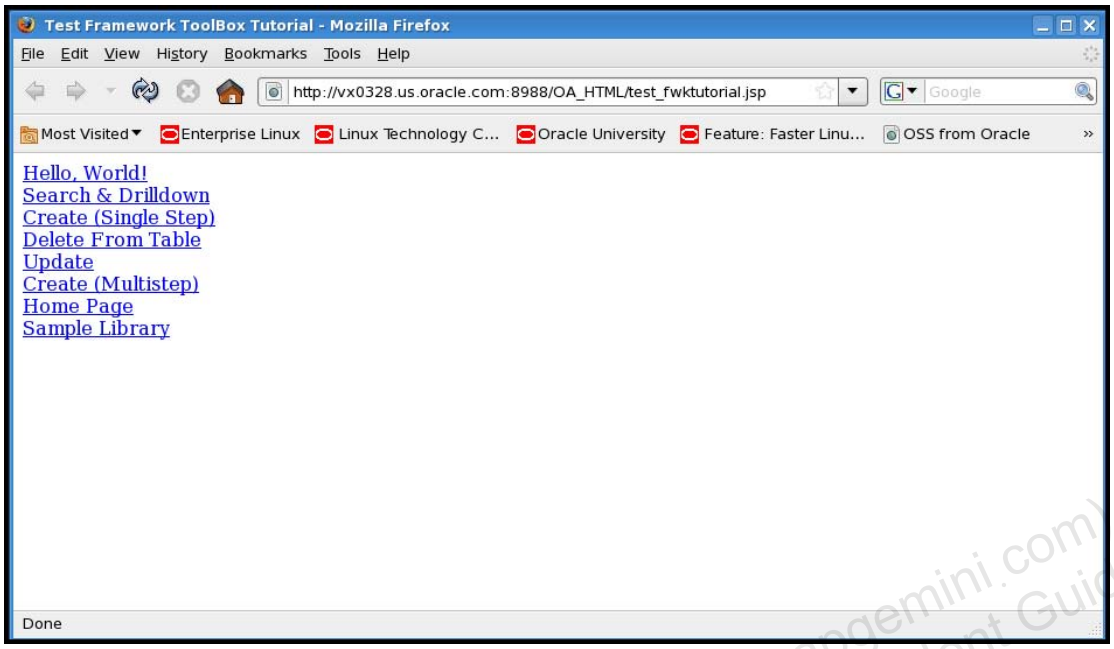


Note: If a migration/upgrade windows -- similar to the Business Components windows pictured in Task 8 appear -- click the **OK** button

Note: The first time any page runs in JDeveloper, it takes longer. Several files need to be compiled, and numerous files need to be deployed to allow JDeveloper with OA Extension to operate properly. After the first run, subsequent runs should complete much faster.

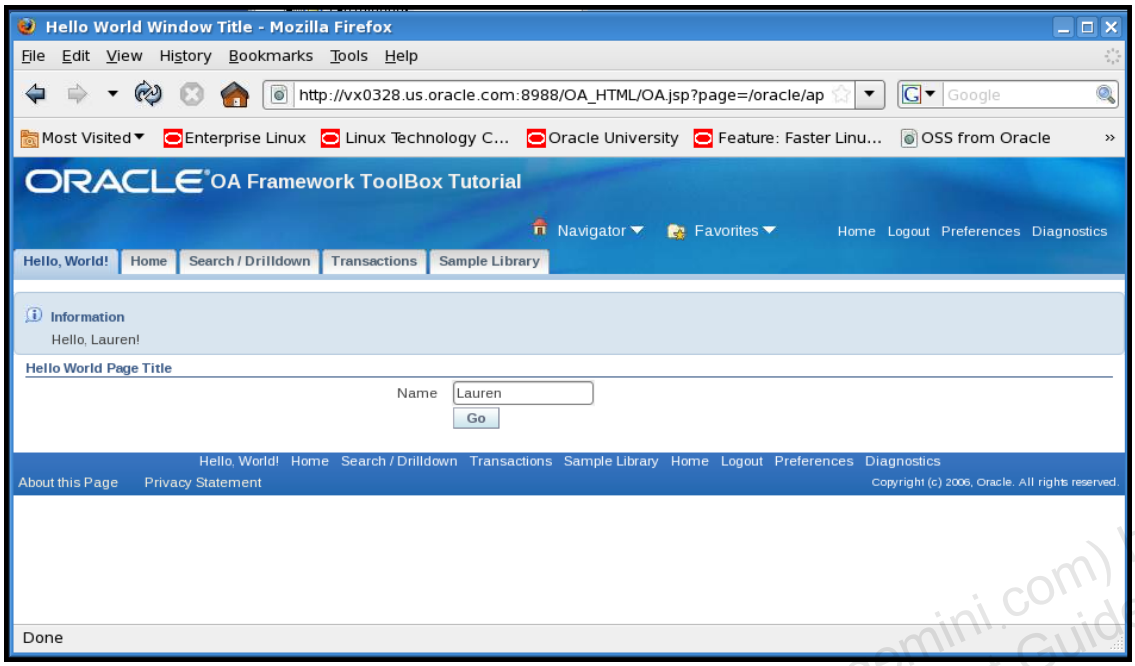
Note: You can safely ignore any warning messages in JDeveloper's compiler log area.

3. A browser window should appear with a list of links. Click the **Hello, World!** link.



4. When you see a screen similar to the following, you know that you have correctly installed JDeveloper 10g with OA Extension, and you have configured it to work properly with your E-Business Suite instance.

5. Click the **Hello, World!** tab.



6. Enter data into the **Name** field, and click the **Go** button to observe the functionality. You can also examine the contents of the other pages. Close the browser when done.

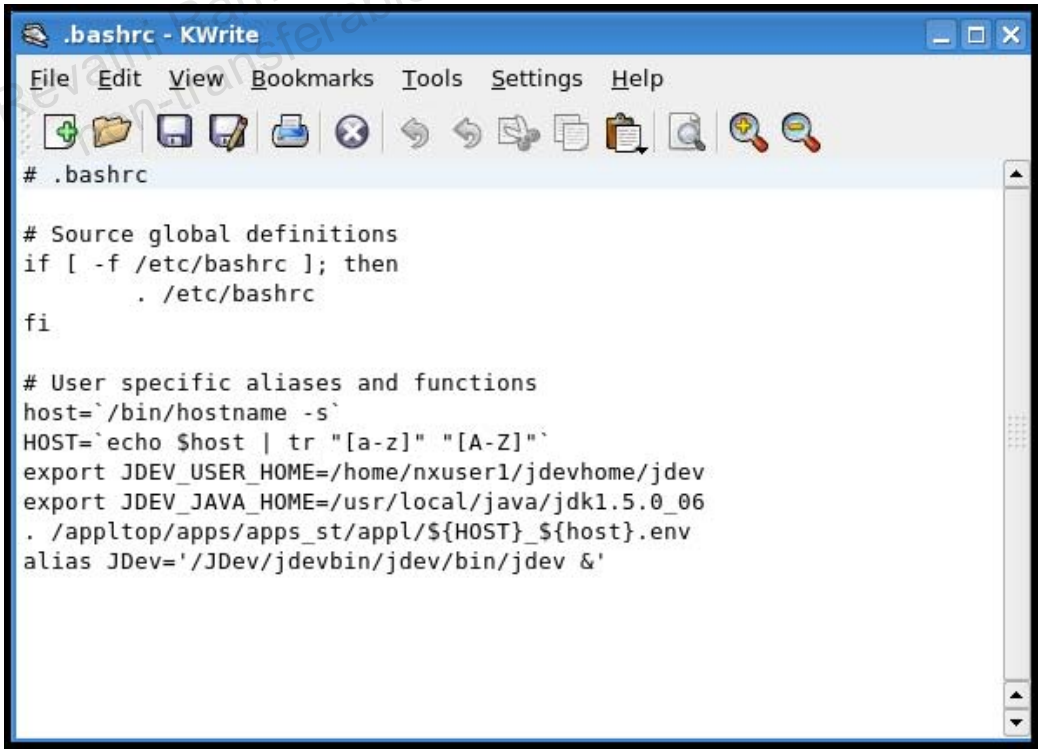
Task 12: Optional – Personal Set-up

The following should be pre-configured. The task that follows is optional, but it will significantly add to your development productivity. These changes are strongly recommended if they have not already been set.

You will set your user account so that when a terminal window is opened, the JDEV_USER_HOME environment variable is set. You will also set the E-Business Suite environment variables, and create an alias, called **JDev**, that will give you a shorthand means of invoking JDeveloper.

1. Open a new terminal window, and type the following commands:
   `kwrite .bashrc`
2. Once the file is open, verify that your .bashrc contains the following lines. If not, add the lines, then save and exit:

```
# .bashrc
# Source global definitions
if [ -f /etc/bashrc ]; then
. /etc/bashrc
fi

# User specific aliases and functions
host=`/bin/hostname -s`
HOST=`echo $host | tr "[a-z]" "[A-Z]"`
export JDEV_USER_HOME=/home/nxuser1/jdevhome/jdev
export JDEV_JAVA_HOME=/usr/local/java/jdk1.5.0_06
. /appltop/apps/apps_st/appl/${HOST}_${host}.env
alias JDev='/JDev/jdevbin/jdev/bin/jdev &'
```



3. Close the terminal window, and open a new terminal window. Your environment will automatically be set, and you can type **JDev** to start-up JDeveloper.

Lab - First OA Framework Page

Chapter 3

Overview

This lab uses Oracle JDeveloper 10g with OA Extension to create a very simple page. The lab has minimal explanation and as few steps as possible. It takes approximately 1 hour to complete with fast connections and minimal load on the instance. Your time will probably be longer. The screen captures were accurate when this lab was created, however the screen captures may not be exactly what you see when you perform these labs. After completing this lab, you should have:

- Created a JDeveloper with OA Extension workspace and project.
- Configured a project and enable Developer Mode testing and diagnostics.
- Used JDeveloper 10g with OA Extension to create a very simple page.

Task 1: Setup your environment for OA Framework development

Step 1: Create the new workspace and project.

1. Right-click **Applications** in the Applications Navigator pane.
2. Click **New OA Workspace**.
3. Enter the **File Name** as **ClassLabs.jws**.
4. **Directory Name** should be `JDEV_USER_HOME/myprojects`.

   Note: This is the default. So, you only need confirm that it is set as expected.
5. Check the **Add a New OA Project** checkbox.
6. Click the **OK** button.
7. Click the **Next** button when the Oracle Applications Project Wizard – Welcome page opens.
8. Enter the **Project Name** as **ClassProject**.
9. Enter the **Directory Name** as `JDEV_USER_HOME/myprojects`.

   Note: This is the default. So, you only need confirm that it is set as expected.
10. Enter the **Default Package** as <*student#*>.**oracle.apps.ak.first**.

    Note: The <*student#*> notation represents a substitution token. Whenever this is seen please replace it with the word "student" followed by your assigned number.
11. Click the **Next** button.
12. At the Database Connection window (Wizard Step 2 of 3), click the **Next** button to bypass the options.
13. At the Runtime Connection window, either input the path directly, or **Browse** for the specific DBC file. The field should contain:

    `/home/nxuser1/jdevhome/jdev/dbc_files/secure/`*`<SID>`*`.dbc`**
14. Enter the **User Name** as **FWKTESTER**.
15. Enter the **Password** (case-sensitive) as **FWKDEV**.
16. Enter or verify the **Application Short Name** is **AK**.
17. Enter or verify the **Responsibility Key** is **FWK_TBX_TUTORIAL**.
18. Click the **Next** button.
19. Click the **Finish** button.
20. Click the **Save All** icon, to save your progress. 

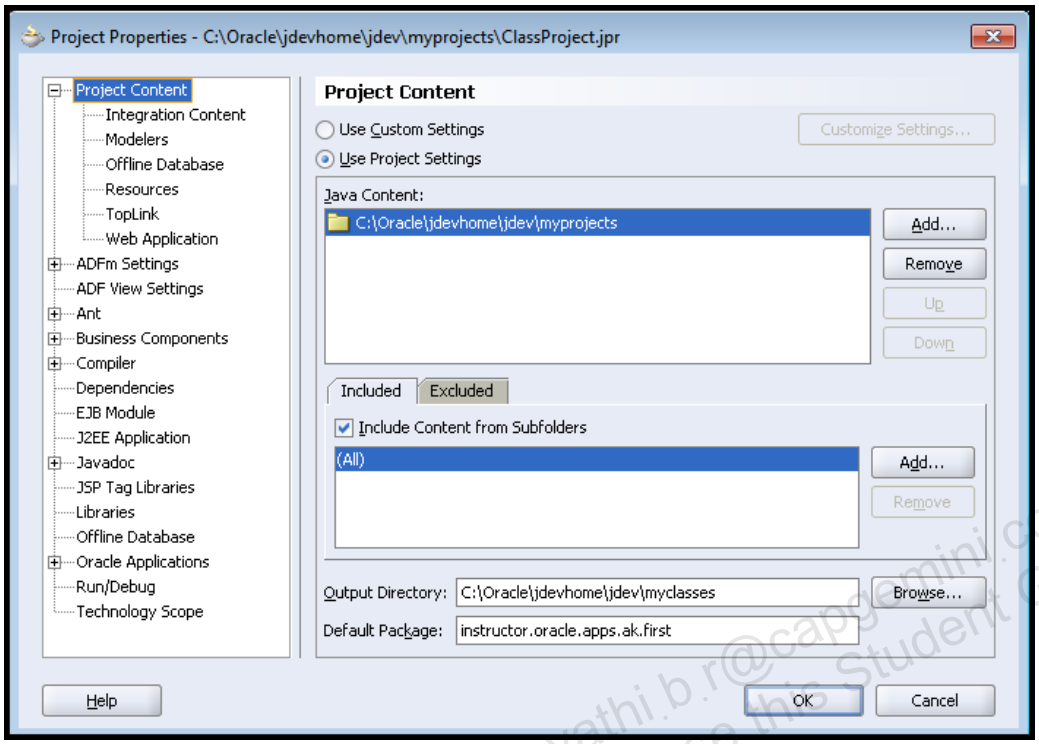
Step 2: Set your project's Run Options.

1. Click the **Applications Navigator** tab.
2. Double-click **ClassProject** to open the Project Properties window.

   Note: You can also right-click the project, and select Project Properties from the context window.
3. Expand the **Oracle Applications** category
4. Click **Run Options**.
5. Shuttle **OADiagnostics** from Available Options to Selected Options using the **[>]** button. You should have OADiagnostic and OADeveloper mode as selected options.

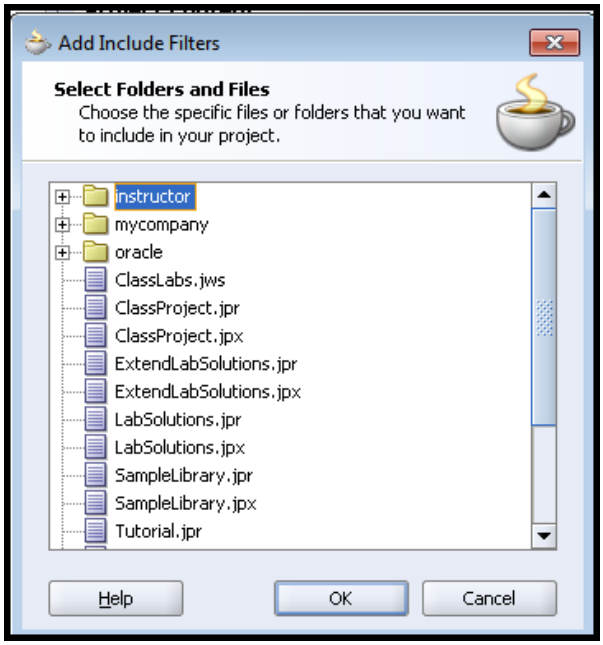
6. While still within the project properties, under Oracle Applications, under Runtime Options, you can modify or change the runtime parameters that you set when the wizard for the new OA project ran. Verify that the DBC file and other parameters are the same.

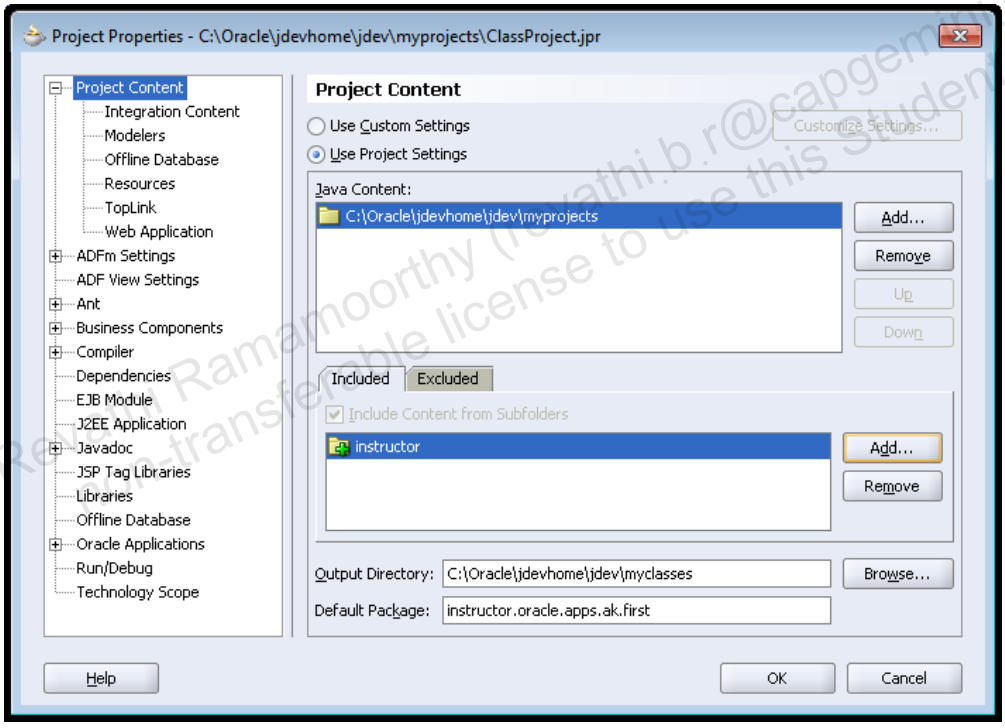
7. Another option is to restrict the packages you can see in your project. JDeveloper sees all packages in the myprojects directory until you limit the levels. In the Project Content, you will see the following screen.



By including only your student folder, you can limit the files that you see and you are working on. This makes it easier to keep track of the specific content that you want in your project. You can also exclude content using the other tab. Since you have not excluded any files, they are still available.

Note: There is a slight difference in the Windows version of JDeveloper and the Linux version. In the Windows version, you can include your files at this point, as in the screen captures below. In the Linux version, you have to create an object before there is a directory structure.





8. Click the **OK** button when you have finished setting the project properties.
9. Save your process by clicking the **Save All** icon.

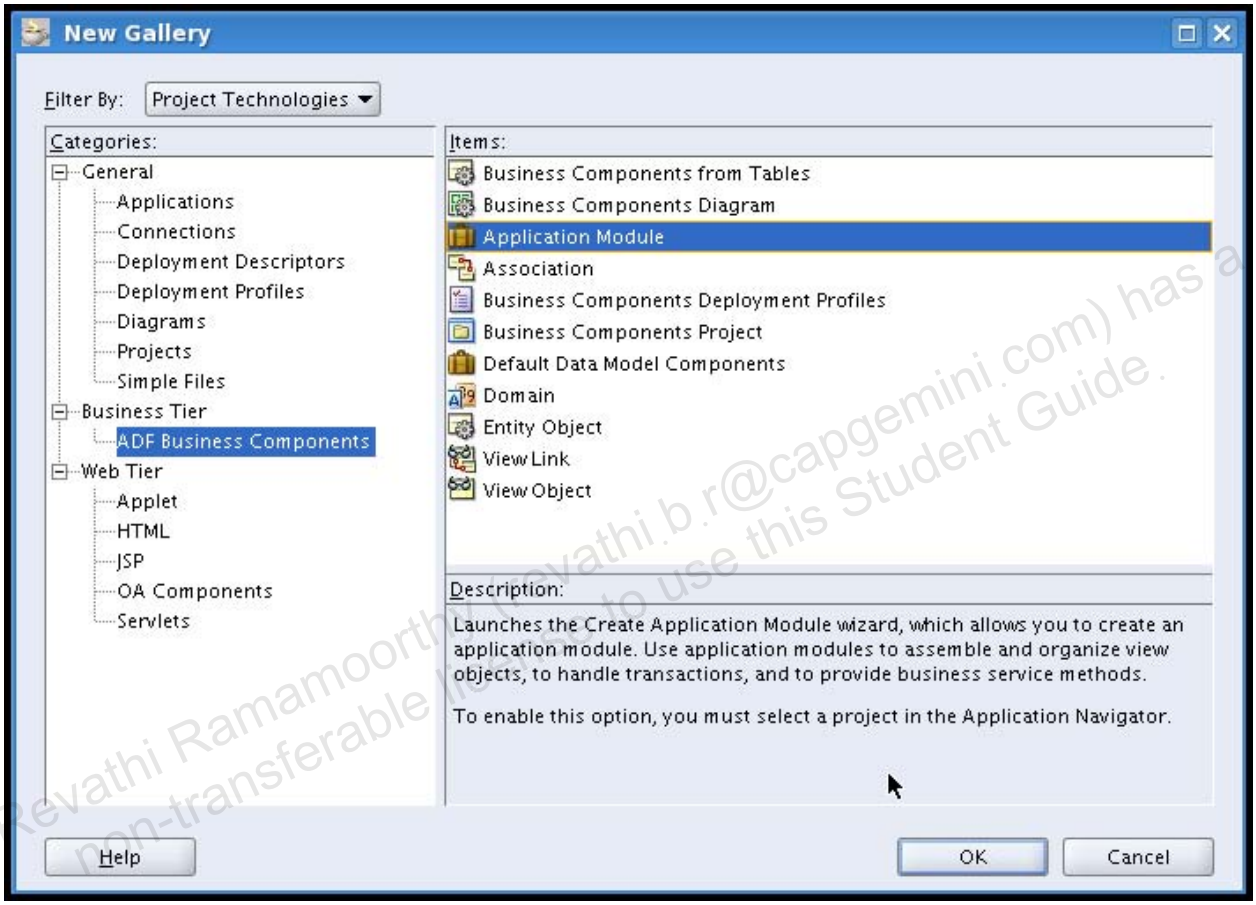
   Note: As a general rule, you should save your work after each Task # or every 20 minutes. Keep in mind that JDeveloper has extensive undo capabilities that are reset when you save.

Task 2: Create the Model-layer Components

Step 1: Create your Application Module (AM)

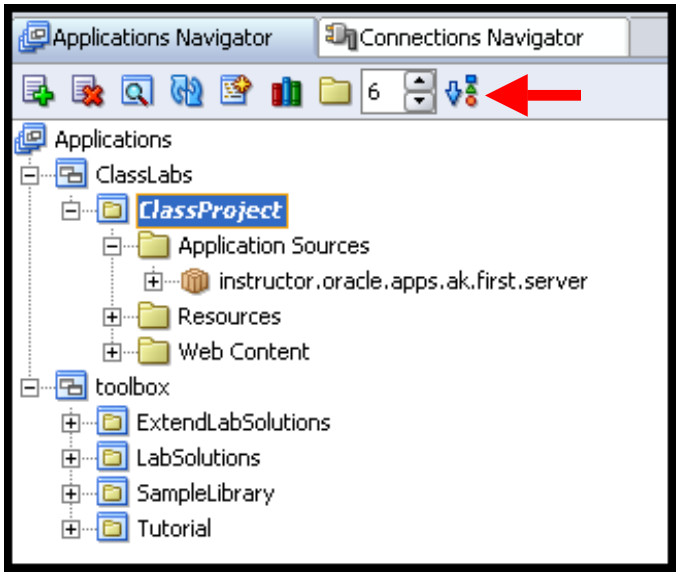
Every OA Framework page requires a connection to the E-Business Suite database. The BC4J component that holds the connection and transaction is called an Application Module (AM).

1. Right-click the **ClassProject** project, and choose **New…** from the context menu.
2. Expand the **Business Tier** category and click **ADF Business Components**.

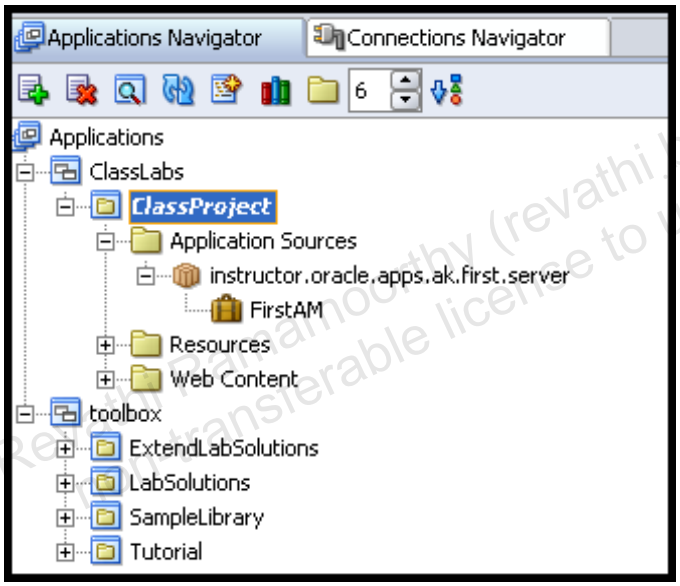


3. Click **Application Module,** and click the **OK** button to open the wizard.
4. In the Welcome window, click the **Next** button.
5. Set the Package to `<student#>.oracle.apps.ak.first.server`

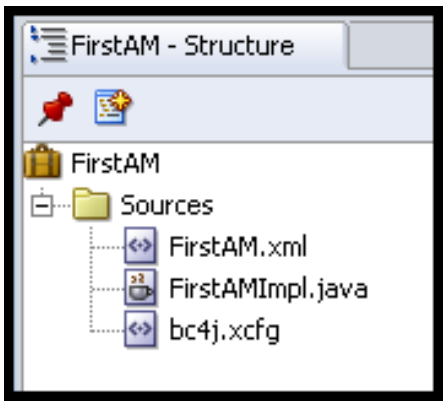
   Note: Make certain the package name is set properly. Carefully read any defaults that may have been entered automatically.
6. Set the **Name** to **FirstAM**.
7. Click the **Next** button.
8. Accept all the remaining defaults, and click the **Finish** button.
9. Click the **Save All** icon.
10. Expand the **Application Sources**.
11. Expand the **<student#>.oracle.apps.ak.first.server** package.
12. If you do not see this package in the Applications Navigator, you need to change the Level display to at least 6.



Note: When you expand <student#>.oracle.apps.ak.first.server, you will see your newly created FirstAM.



Note: FirstAM created three (3) files. They can be seen in the Sources folder of the Structure panel. Those files are



`FirstAM.xml` – the declarative (XML-based) components of the AM.

`FirstAMImpl.java` – the programmatic (Java-based) components of the AM.

`bc4j.xcfg` – the XML configuration which lists all the BC4J components in your project.

13. Save your work.

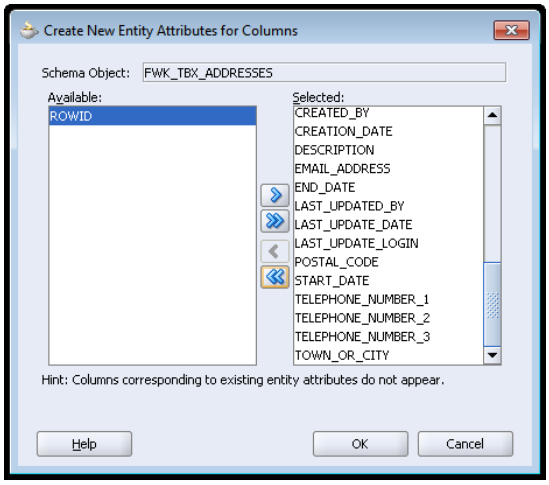
Step 2: Create your Entity Object (EO)

You only need an Entity Object (EO) in an OA Framework page if that page is going to do an Insert, Update, or Delete. A simple View Object (VO) is adequate if you only need to "select" and display data. Regardless, it is good practice to create EOs and build VOs from them. You are going to create the following EO as a building block for what is to come later in the labs.

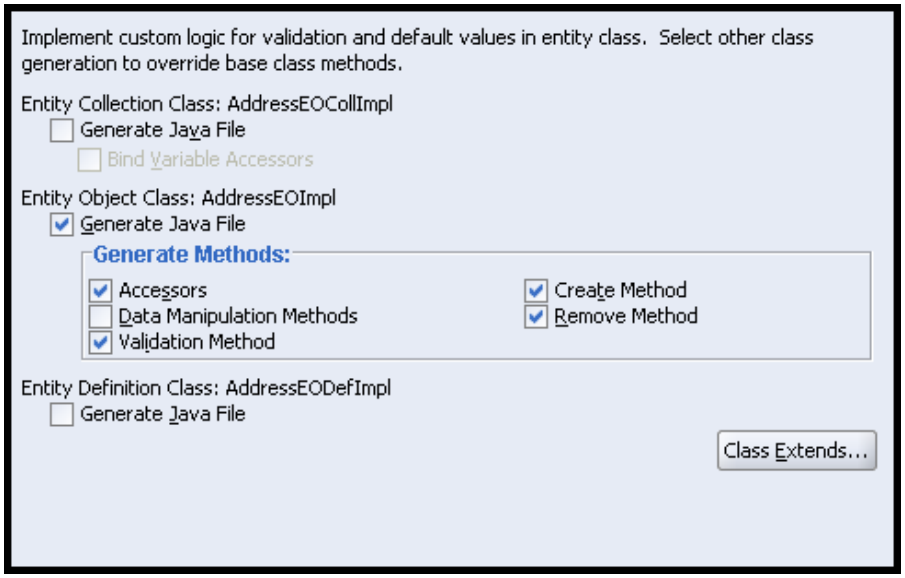
Tip: Most of the ADF Business Component wizards allow resizing the wizard window even though there may not be visible resize controls. This is helpful for viewing long values and SQL statements.

1. Right-click the **ClassProject** project, and choose **New…** from the context menu.
2. Expand the **Business Tier** category and click **ADF Business Components**.
3. Click **Entity Object**, and click the **OK** button to open the wizard.
4. In the Welcome window, click the **Next** button.
5. In the Entity Object section, set the **Name** to **AddressEO**.
6. Set the Package to **`<student#>.oracle.apps.ak.schema.server`**

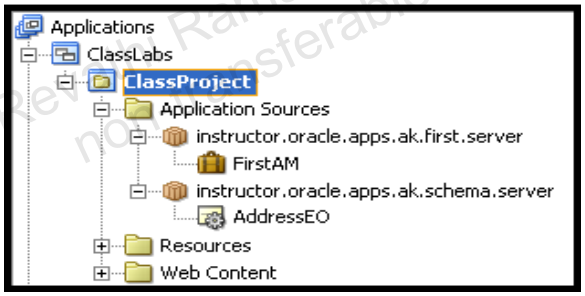
   Note: The default package name is not correct. You will have to enter this package name manually.
7. In the Database Objects section, in the **Schema Object** field, type the value, **FWK_TBX_ADDRESSES**.
8. **Tables** should already be checked, and click the **Next** button.
9. In step 2 of 5 of the EO Wizard, the Attributes window, the Entity Attributes should be populated. However, the JDeveloper EO wizard is going over a JDBC connection to query the database for the tables and can be very slow. Technically, the FWK_TBX_ADDRESSES is a synonym in the schema.
10. If you do not see the Entity Attributes, do this. Click the **New From Table…** button. **Wait**.
11. Shuttle all of the columns from Available to Selected using the **[>>]** button.
12. Shuttle ROWID back to Available using the **[<]** button.



13. Click the **OK** button.
14. Click the **Next** button until you reach the Java window



15. Check the following checkboxes:
    Entity Object Class: - Generate Java File
    Accessors
    Validation Method
    Create Method
    Remove Method
16. Click the **Next** button until you reach the Finish window.
17. Click the **Finish** button.
18. Save your work.
19. You should see the following in the Applications Navigator



Step 3: Create your View Object (VO)

According to Oracle Applications Development standards and good programming practice, your VO should include only the attributes (columns) that are going to appear in the UI. You will create a VO, based on **AddressEO**, your previously created EO, that includes specific columns from the entity (table).

1. Right-click the **ClassProject** project, and choose **New** from the context menu.
2. Click **Business Tier** > **ADF Business Components** > **View Object**.
3. Click the **OK** button to open the Create View Object wizard.
4. Click the **Next** button at the Create View Object – Welcome page.
5. Set the Package to `<student#>.oracle.apps.ak.first.server`.
6. Set the **Name** to **AddressVO**.

7. Select the **Rows Populated by a SQL Query with:** option.
8. Select the **Updatable Access through Entity Objects** option.
9. Click the **Next** button on Step 1 of 7.
10. Click the **<student#>.oracle.apps.ak.schema.server** package to expand it.

    Note: You can resize the wizard window by dragging it.
11. Click the **AddressEO**, and shuttle it to the Selected pane.
12. Click the **Next** button on Step 2 of 7.
13. Click the following fields, and shuttle them to the Selected pane:

    AddressId

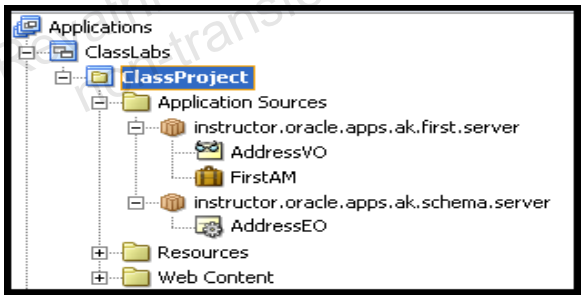
    AddressLine1

    AddressName

    TownOrCity

    Note: The Windows version of JDeveloper may also create a **RowID** in the Selected pane. If this occurs, use the Up and Down arrows in the Selected pane to order them as AddressId, AddressName, AddressLine1, TownOrCity, and RowID.
14. Click the **Next** button until Step 7 of 7.
15. On Step 7 of 7, uncheck all the pre-checked checkboxes.
16. Check the **View Row Class: AddressVORowImpl** checkboxes for **Generate Java File** and **Accessors**.

    Note: You select these as part of adhering to Oracle Applications Development standards that allow extensibility.
17. Click the **Next** button on Step 7 of 7.
18. Click the **Finish** button.
19. Save your work.
20. You should have the AddressVO as part of the `<student#>.oracle.apps.ak.first.server`.

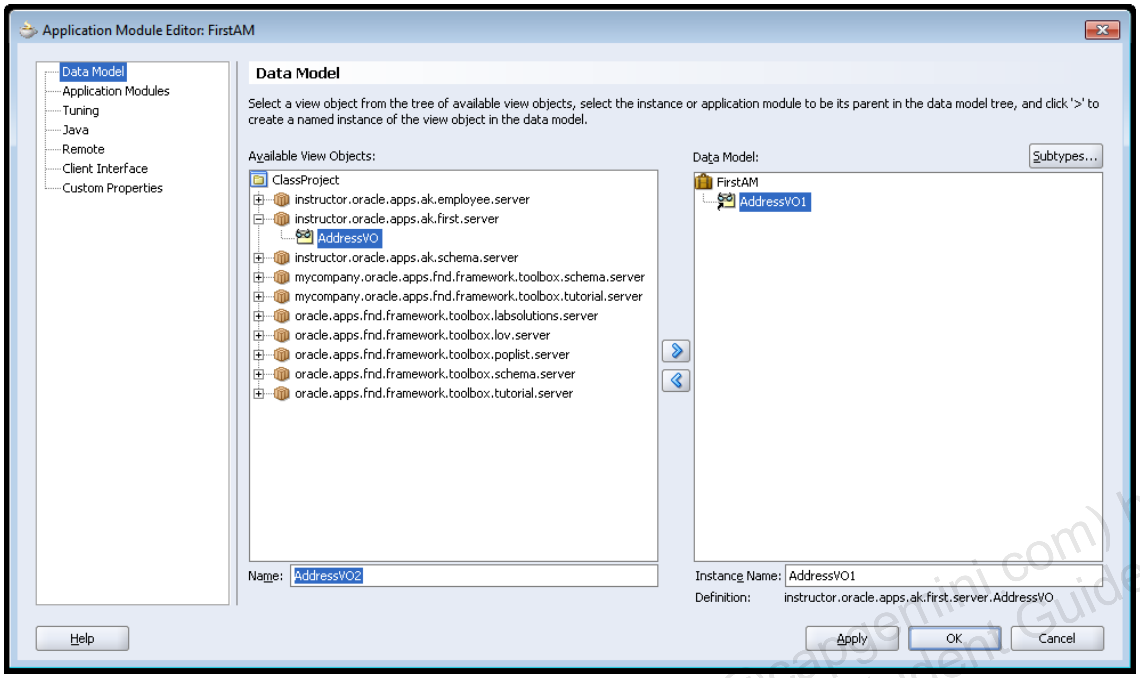


Step 4: Add your VO to your AM

All VOs must be contained within an AM. In this lab, your AM is the FirstAM. You will add the AddressVO to the FirstAM as follows;

1. Click **Applications Sources** > **<student#>.oracle.apps.ak.first.server** to expand it.
2. Right-click **FirstAM** > **edit FirstAM** to open the Application Module Editor.
3. Click the **Data Model** category (if not already selected by default).

4. Click **AddressVO** from the Available View Objects pane, and shuttle it to the Data Model pane.

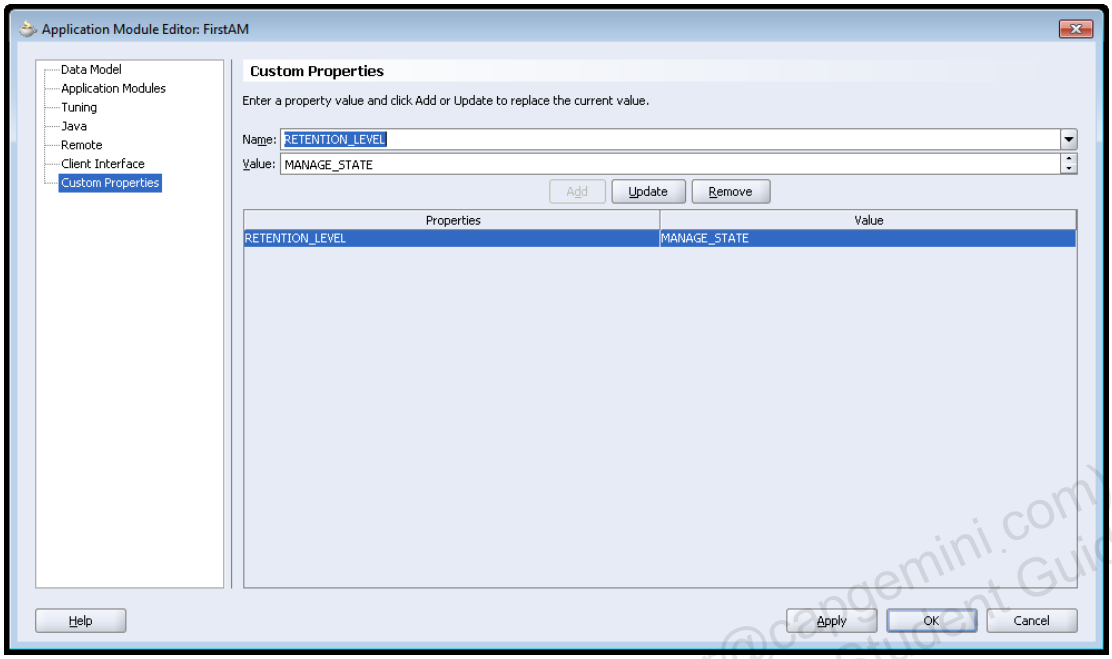


Note: When shuttled, AddressVO becomes AddressVO1. This is expected behavior. You are taking the VO from a definition to an instance of that definition.

5. While still in the editor, click the **Custom Properties** category. There will an entry on the **Name** line as Description. Overwrite that entry.
6. Set the **Name** to **RETENTION_LEVEL**.
7. Set the **Value** to **MANAGE_STATE**.

8. Click the **Add** (no longer grayed out) button to add this custom property.

   Note: While not critical at this point, this enables passivation on an OA Framework page. All AMs should be set this way.

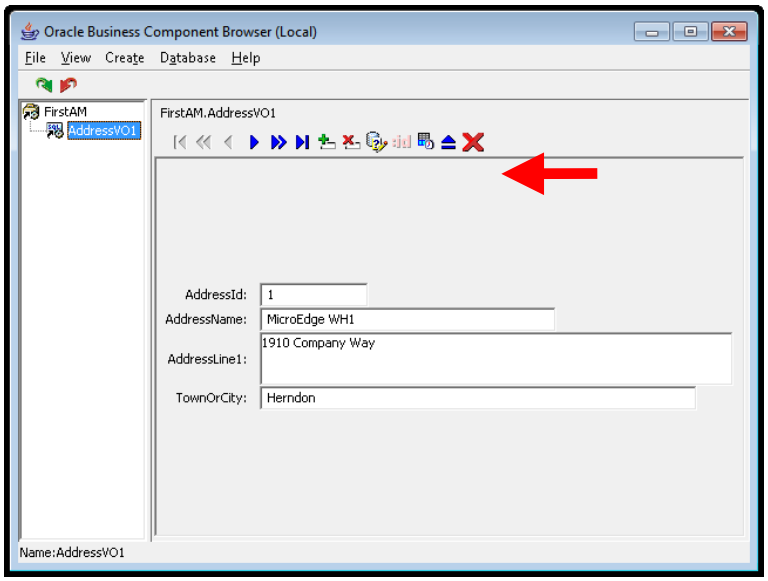


9. Click the **Apply** button.
10. Click the **OK** button.
11. Save your work.

Step 5: Test your BC4J objects

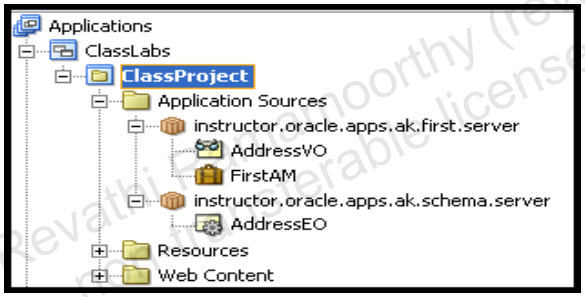
1. Right-click your **FirstAM**, and choose **Test** from the context menu.
2. This opens the Oracle Business Component Browser. Click the **Connect** button.
3. Double-click **AddressVO1** in the BC4J browser.
4. This will query the database, and return the records associated with your VO (which is just a SQL SELECT).

5. You can use the navigation keys in the BC4J browser to explore the records.



Note: Do not create new records here. Why? Because your VO only contains four (4) fields. There are other data in the record that is required for this table. If you insert a record, you will corrupt the FWK_TBX_ADDRESSES table.

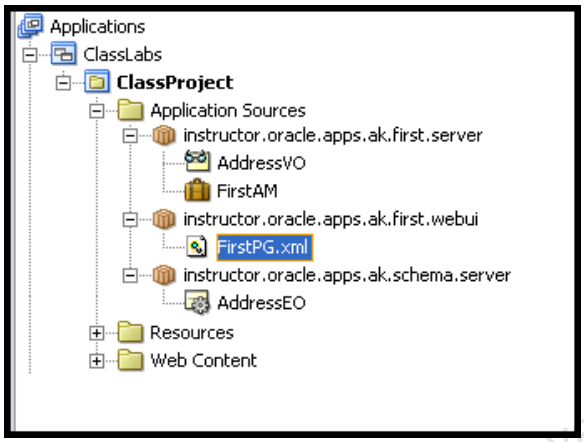
6. Close the BC4J browser by selecting the **[X]** at the top of the window.
7. Save your work
8. Your ClassProject should look as follows:

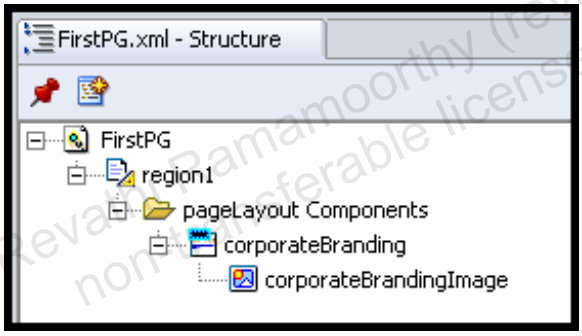


Step 3: Create the View-layer Components

Step 1: Create your page (PG)

1. Right-click the **ClassProject** project, and choose **New** from the context menu.
2. Select **Web Tier** > **OA Components** > **Page**.
3. Set the **Name** to **FirstPG**.
4. Set the **Package** to `<student#>.oracle.apps.ak.first.webui`.
5. Click the **OK** button.
6. **FirstPG** is created. It is selected in the Applications Navigator panel, and it is shown in detail in the Structure panel. Additionally, the Property Inspector opens.





7. In the Structure panel, select **region1**. Use the Property Inspector to set the following properties for this region:

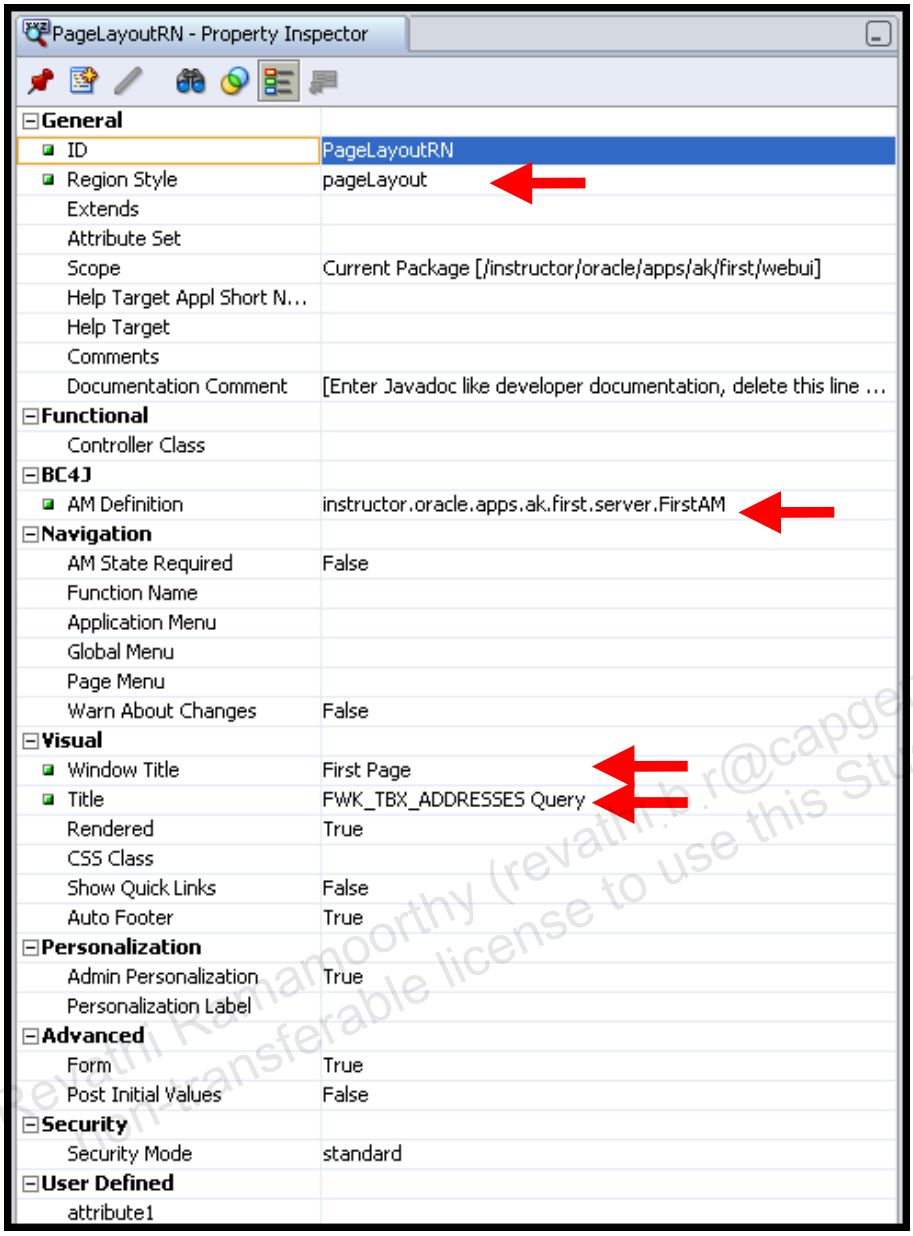
   ID: **PageLayoutRN**

   AM Definition: `<student#>.oracle.apps.ak.first.server.FirstAM`

   Note: You can click the **...** to browse for your FirstAM.

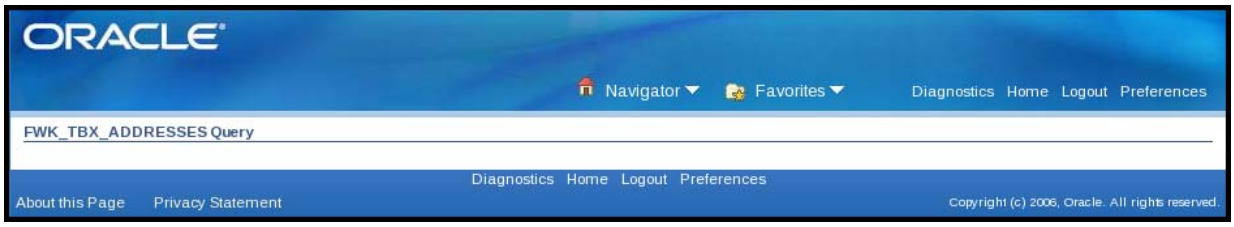
   Window Title: **First Page**

   Title: **FWK_TBX_ADDRESSES Query**



8. Save your work.

9. Test your work. Right-click **FirstPG.xml** in the Applications Navigator or Structure panel, and choose **Run** from the context menu. Your page should appear as follows:



10. While this isn't much, it is your first OA Framework page and it runs. Close the browser window. You will now modify this page to make it do something.

Step 2: Create your query region

A very common object in OA Framework pages within the E-Business Suite is a query page or region. A special UI component, the query bean, has been created to handle this task. A query bean consists of two regions, the query region and the results table.

1. Click **FirstPG** in the Structure panel.
2. Right-click **PageLayoutRN**, and choose **New > Region** from the context menu.
3. Once the new region is created, use the Property Inspector to change the properties of the newly-created region1 as follows:

   ID: **QueryRN**

   Region Style: **query**

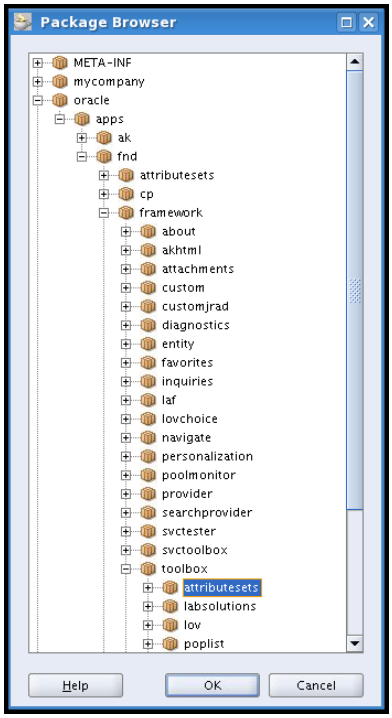
   Construction Mode: **resultsBasedSearch**

   Include Simple Panel: **True**

   Include Views Panel: **True**

   Include Advanced Panel: **True**
4. Save your work.
5. Right-click the **QueryRN** in the Structure panel and select **New > Region Using Wizard** to open the Create Region wizard.
6. Click the **Next** button at the Welcome page.
7. Select the **`<student#>.oracle.apps.ak.first.server.FirstAM`** from the drop-down list of Application Modules.
8. Click **AddressesVO1** in the Available View Usages.
9. Click the **Next** button.
10. Set the **Region ID** to **ResultsRN**.
11. Set the **Region Style** to **table** from the drop-down list.
12. Click the **Next** button.
13. Shuttle all the attributes from Available View Attributes to Selected View Attributes using the **[>>]** button.
14. Click the **Next** button.
15. On Step 4 of 4: Region Items, click the **Style** field for **AddressId**, and click **messageStyledText** from the drop-down list.
16. Click the **Attribute Set** field for the **AddressName** attribute.
17. Click the **Search** (**flashlight**) icon.
18. Select the **Entire MDS XML Path** option.
19. Click the **Browse** button.

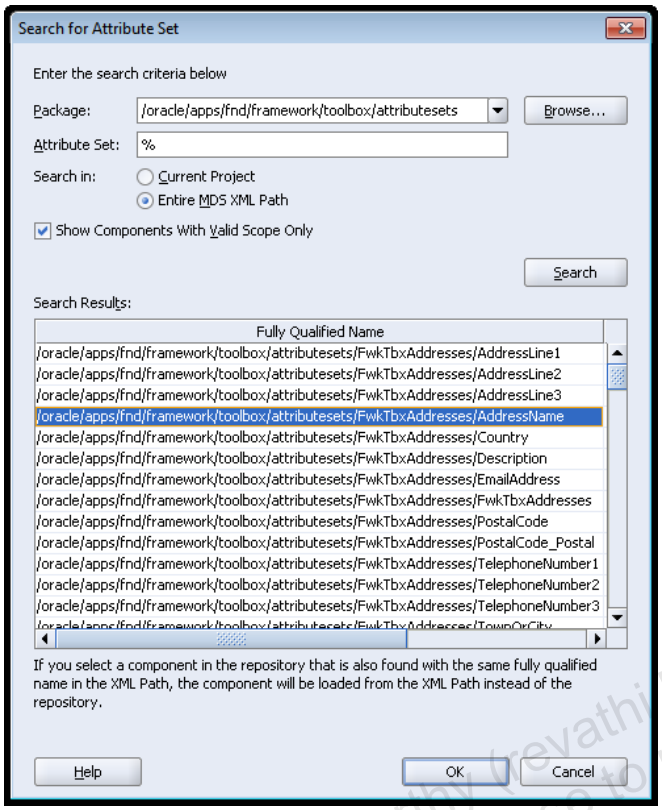
20. Using the Package Browser, click the **[+]** icon expanding down to oracle > apps > fnd > framework > toolbox > attributesets.
21. Click the **OK** button.



22. Click the **Search** button.

    Note: This lists all the associated attribute sets. Some of the attribute sets are named, FwkTbxAddresses, and have been created specifically for columns in the FWK_TBX_ADDRESSES table. You will need to expand the Fully Qualified Name column to see the complete names.

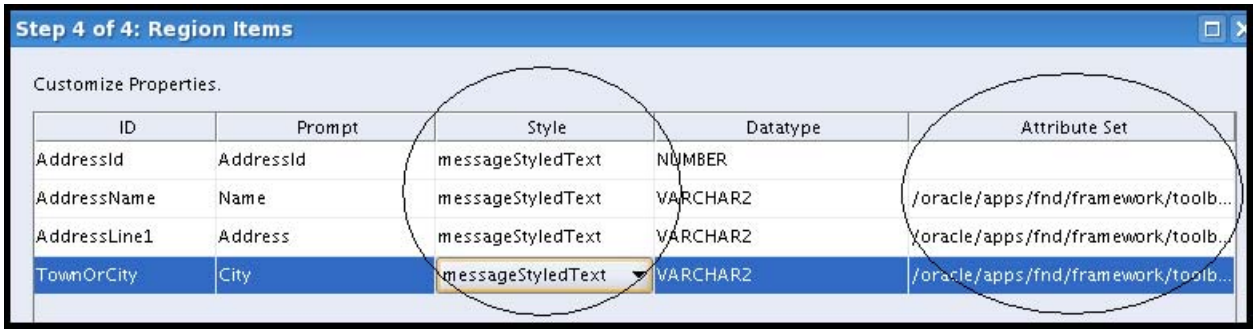
23. Click the `/oracle/apps/fnd/framework/toolbox/attributesets/FwkTbxAddresses/AddressName` attribute set.



24. Click the **OK** button to return.
25. Click the **Style** field for AddressName, and click **messageStyledText** from the drop-down list.
26. Repeat steps 16-25 for the AddressLine1 and TownOrCity attributes, setting the proper attribute set for each. The values are as follows:

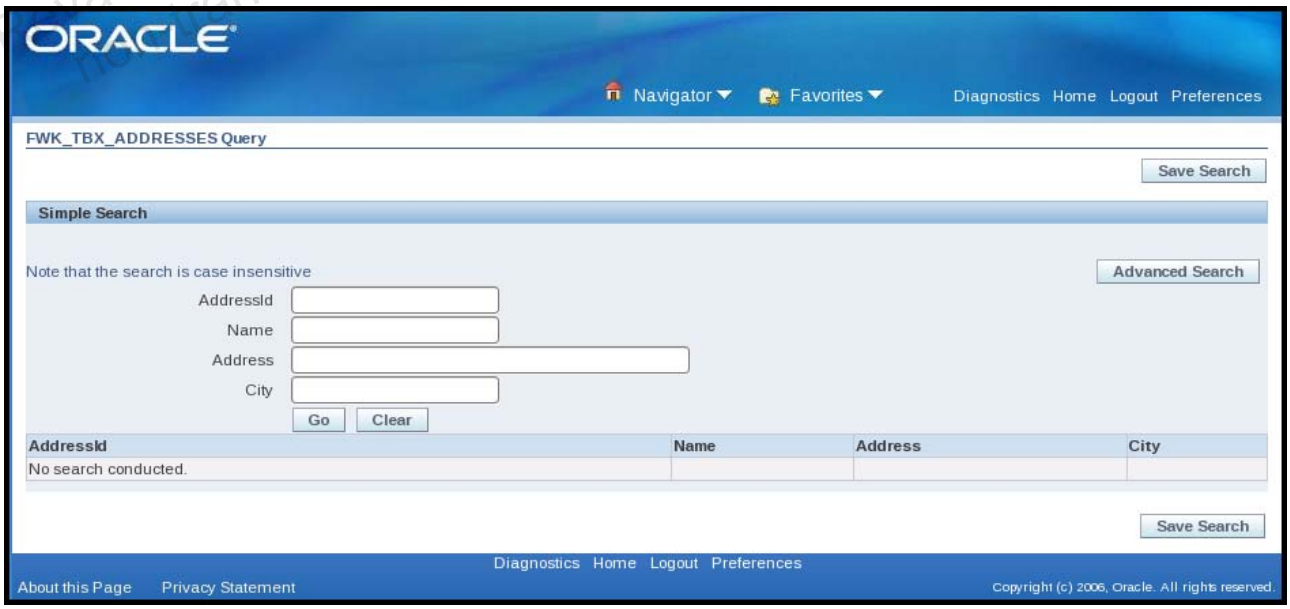
    `/oracle/apps/fnd/framework/toolbox/attributesets/FwkTbxAddresses/AddressLine1`

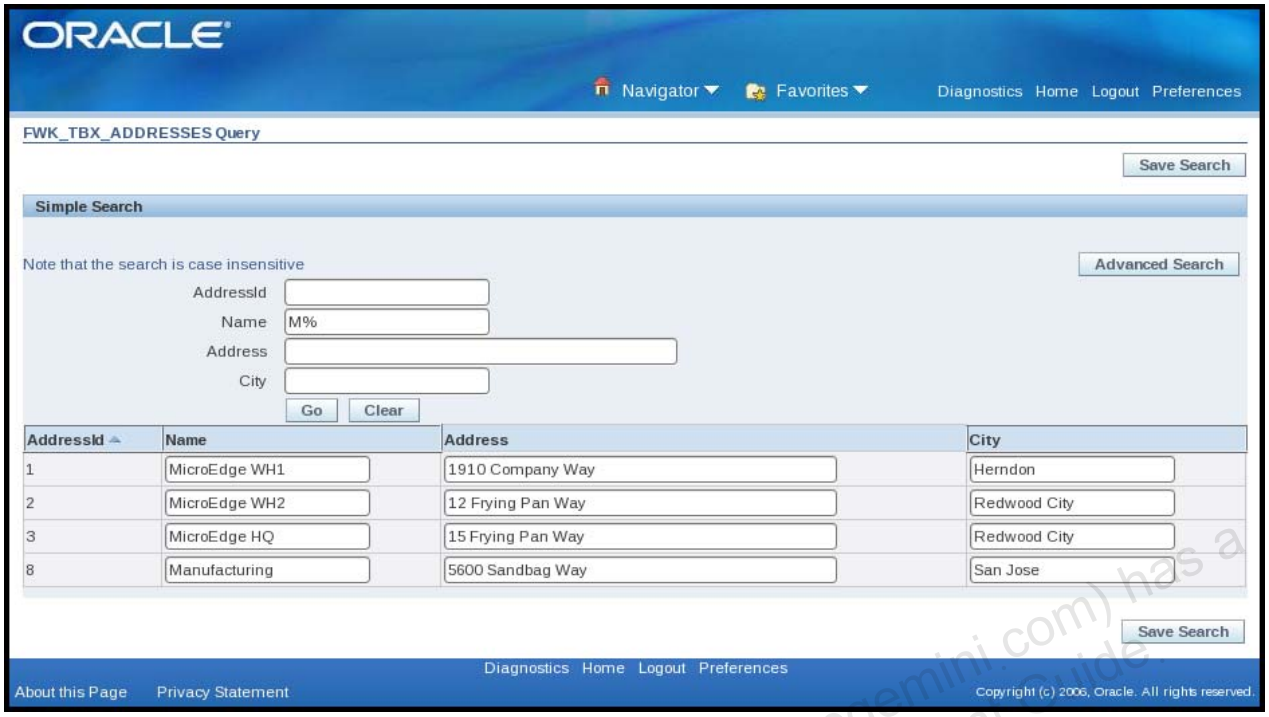
    `/oracle/apps/fnd/framework/toolbox/attributesets/FwkTbxAddresses/TownOrCity`



27. Click the **Next** button.
28. Click the **Finish** button to create your results table.
29. Save your work.

30. Click **ResultsRN**, and set the following properties:
    Additional Text: **Addresses Table**
    Width: **100**%
    User Personalization: **True**
31. In the Structure panel, click **ResultsRN** to expand it.
32. Click **AddressId,** and set the following properties:
    Search Allowed: **True**
    Sort Allowed: **yes**
    Initial Sort Sequence: **first**
    Selective Search Criteria: **True**
    User Personalization: **True**
33. Click **AddressName**, and set the following properties:
    Search Allowed: **True**
    Sort Allowed: **yes**
    Selective Search Criteria: **True**
    User Personalization: **True**
34. Click **AddressLine1**, and set the following properties:
    Search Allowed: **True**
    User Personalization: **True**
35. Click **TownOrCity**, and set the following properties:
    Search Allowed: **True**
    Sort Allowed: **yes**
    User Personalization: **True**
36. Save your work.
37. Right-click **FirstPG.xml** in the Applications Navigator or Structure panel, and choose **Run** from the context menu. Your page should appear as follows:



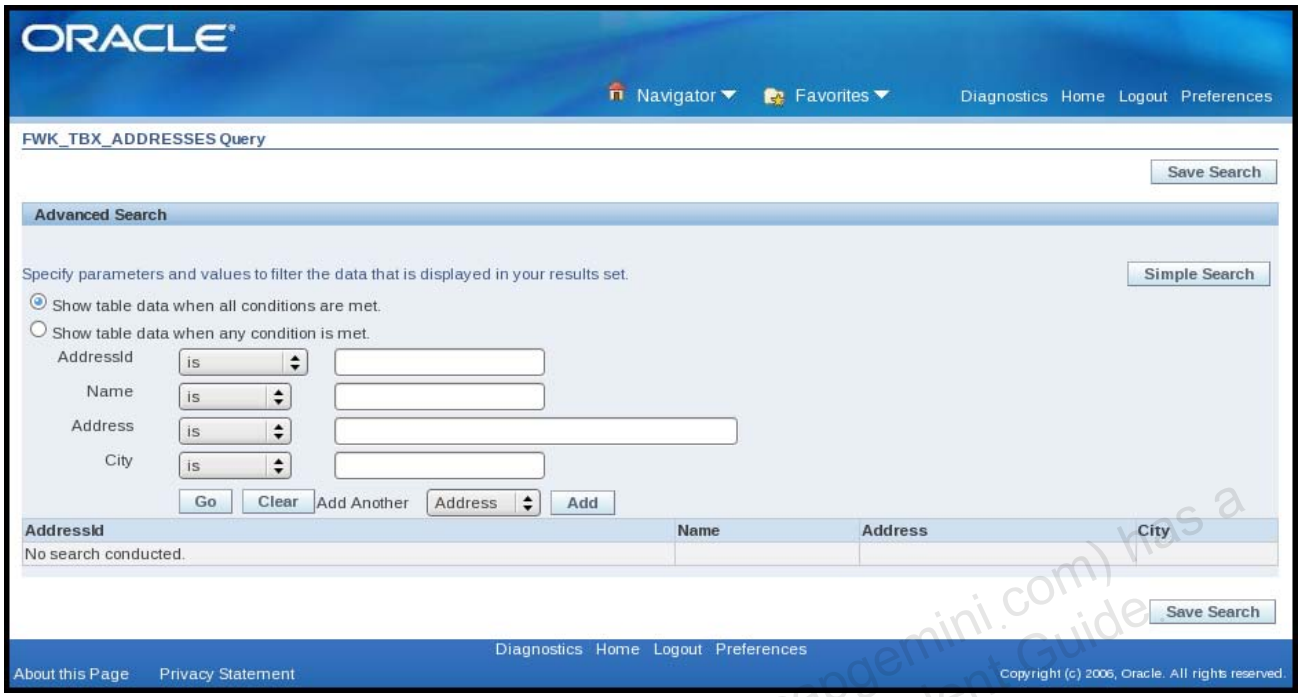


38. Explore the page.

    Note: You can sort on the AddressId and Name columns by clicking the column heading.

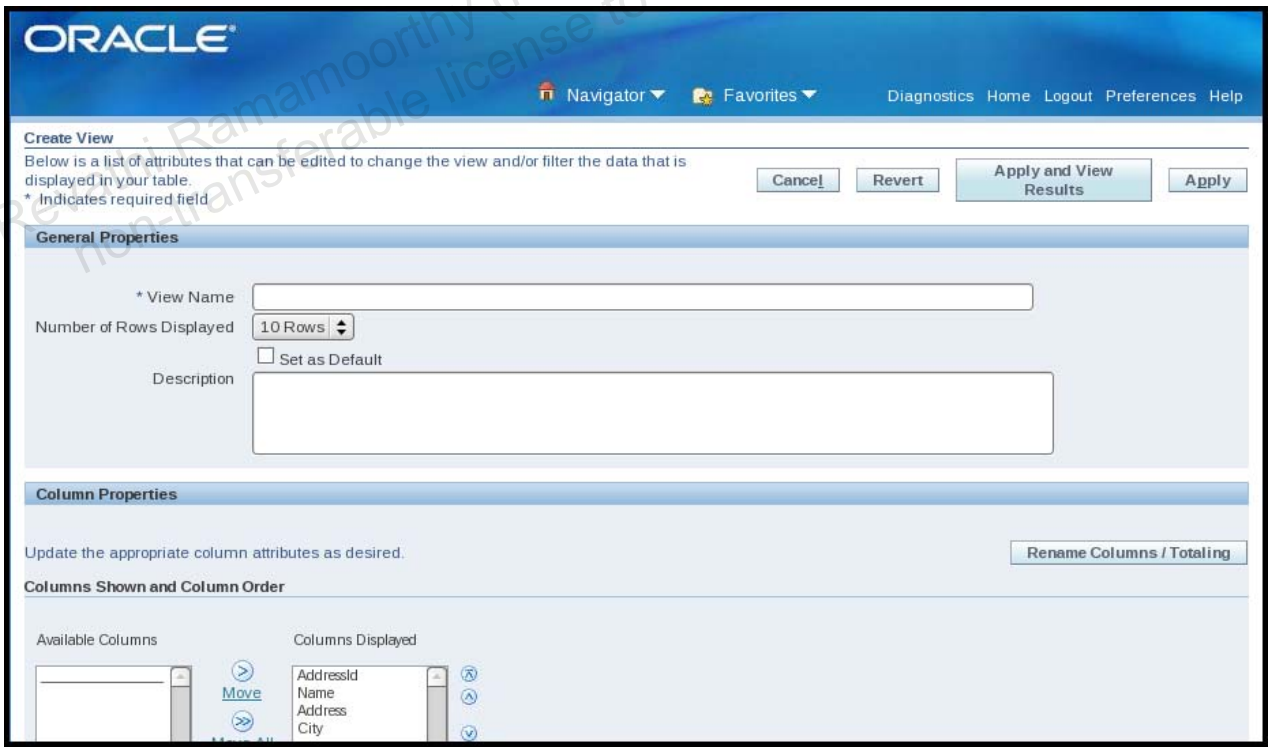
    Note: If you click on the column heading a second time, it will change the sorting from ascending to descending or descending to ascending.

39. Click the **Advanced Search** button.



Note: The **Save Search** button allows you to save this search. This button appears because you set the User Personalization property on the ResultsRN region (and its children items) to True.

40. For advanced challenge, create a User Personalization on your query page.



This is a powerful technique. You can use this as a template to create a query page for any table within an E-Business Suite instance, assuming that the E-Business Suite user and password you are using for your project has the proper access.

Where's the Controller for this region? The Query Bean encapsulates the controller within the UI component. There is no need to write an additional controller. Only write a controller when you are required to do so.

Congratulations! You have now completed your first OA Framework page.

Lab - Implementing a Query and Drill-down

Chapter 4

Overview

For this lab, you will be implementing an employee query and a drill-down to an employee details page. This is the foundation for future enhancements to the application you are building. Eventually, you will add the ability to create, update and delete employees as part of your application.

Task 1: Create Your Model-layer Components

In this task, you will create the initial standards-compliant packages and objects that make up the model-layer in your Employees application.

Note: This lab assumes you have completed Lab 3: First OA Framework Page. This lab builds on objects created during Lab 3. If you have not yet completed that Lab 3, you need to complete the steps outlined in Lab 3 before attempting the steps in this lab.

Step 1: Create Your Employees Application Module (AM)

Create a new Application Module (AM) to be used as the root UI application module for your page.

1. Right-click your **ClassProject** project, and click **New** from the context menu.
2. Click the **Business Tier** > **ADF Business Components** > **Application Module** from the New Gallery to open the Create Application Module wizard.
3. Set the **Package** to `<student#>.oracle.apps.ak.employee.server`.
4. Set the **Name** to **EmployeesAM**.
5. Click the **Next** button.
6. Click the **Finish** button to accept the remaining defaults.
7. Click the **<student#>.oracle.apps.ak.employee.server** package to expand it in the Applications Navigator.
8. Double-click **EmployeesAM** to open the Application Module Editor.
9. Click **Custom Properties** from the categories panel.
10. Set the **Name** to **RETENTION_LEVEL**.
11. Set the **Value** to **MANAGE_STATE**.
12. Click the **Add** button.
13. Click the **Apply**.
14. Click the **OK** button.
15. Save your work.

Step 2: Create Your Employee Entity Object (EO)

Create an entity object (EO) for the FWK_TBX_EMPLOYEES table in the OA Framework ToolBox Tutorial schema.

Tip: Most of the ADF Business Component wizards allow you to resize the wizard window, even though there may not be visible resize controls. This is helpful for viewing long values and SQL statements.

1. Right-click your **ClassProject** project, and click **New** from the context menu.
2. Click the **Business Tier** > **ADF Business Components** > **Entity Object** from the New Gallery to open the Create Entity Object wizard.
3. Set the **Name** to **EmployeeEO**.
4. Set the **Package** to `<student#>.oracle.apps.ak.schema.server`.
5. Set the **Schema Object** to **FWK_TBX_EMPLOYEES**.

   Note: Do not browse for the schema object. Because of the number of objects in the schema, this database query can take a significant amount of time to return with the result.
6. Click the **Next** button.

7. If the columns of the FWK_TBX_EMPLOYEES table do not show up in the Entity Attributes pane, click the **New from Table …** button. Then click the **[>>]** button to move all the columns to the Selected pane. Remove the column named **RowId** from the Selected pane. Click the **OK** button.

   Note: You must remove the RowId column from the Selected columns. If you do not, you will be unable to create your VO later in the lab unless you include the RowId in your select statement.
8. Click the **Next** button.
9. Click the **Next** button.
10. Check the following checkboxes:

    Entity Object Class: EmployeeEOImpl: Generate Java File

    Accessors

    Validation Method

    Create Method

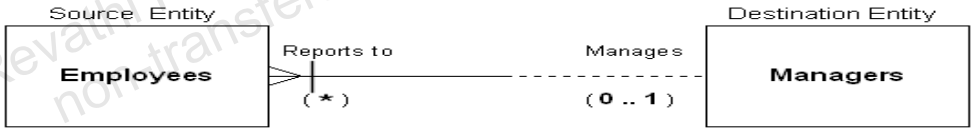
    Remove Method
11. Click the **Finish** button to accept the remaining defaults.
12. Save your work.

Step 3: Create Your Association Object (AO)

You are now going to create an employee-to-manager Association Object (AO) to create a self-join condition on the **FWK_TBX_EMPLOYEES** table. The business rules are:

- An employee must have a manager only if their assigned `POSITION_CODE != "PRESIDENT"`. For all other positions, a manager is required.
- An employee may have, at most, 1 manager.
- A manager can have many employees.

From an entity modeling standpoint, this relationship can be expressed as follows:



In the diagram above, a manager can have any number of employees (indicated by the crow's foot symbol next to the Employees entity). An employee can have at most one, possibly null manager if the employee is a manager (indicated by the straight dotted line next to the Managers entity). In object-oriented modeling terms, the crow's foot symbol is analogous to the ***** annotation, and the single dotted line is represented by the **0..1** annotation. In SQL terms, this relationship is expressed as an outer join.

1. Right-click your **ClassProject** project, and click **New** from the context menu.
2. Click the **Business Tier** > **ADF Business Components** > **Association** from the New Gallery to open the Create Association wizard. If the Welcome screen appears, click the **Next** button.
3. Set the **Package** to `<student#>.oracle.apps.ak.schema.server`.
4. Set the **Name** to **EmpToMgrAO**.
5. Click the **Next** button.
6. Set the **Cardinality** to *** to 0..1**
7. Click the **<student#>.oracle.apps.ak.schema.server** to expand it.

8. Click the **EmployeeEO** under the Select Source Attribute to expand it.
9. Set the **Select Source Attribute** to **ManagerID**.
10. Click the **<student#>.oracle.apps.ak.schema.server** under Select Destination Attribute to expand it.
11. Click the **EmployeeEO** under the Select Destination Attribute to expand it.
12. Set the **Select Destination Attribute** to **EmployeeID**.
13. Click the **Add** button.
14. Click the **Next** button.
15. Click the **Next** button on Step 3 of 4.
16. Click the **Finish** button.
17. Save your work.

Step 4: Create Your View Object (VO)

Create a summary-level view object including only those attributes that you need for the Employees search results table. This view object is the main view object for your Search page. It should leverage the EmployeeEO and EmpToMgrAO you just created.

1. Right-click your **ClassProject** project, and click **New** from the context menu.
2. Click the **Business Tier** > **ADF Business Components** > **View Object** from the New Gallery to open the View Object wizard. If the Welcome screen appears, click the **Next** button.
3. Set the **Package** to `<student#>.oracle.apps.ak.employee.server`.
4. Set the **Name** to **AllEmployeesVO**.
5. Select the **Updateable Access through Entity Objects** option.
6. Click the **Next** button.
7. Click the **<student#>.oracle.apps.ak.schema.server** package in the Available pane, and click the **EmployeeEO**. Use the **[>]** button to shuttle the EO to the Selected pane.
8. Click the **[>]** button and shuttle a second copy of the **EmployeeEO** to the Selected pane.
9. Verify that the **Association** field for the **EmployeeEO1** (second copy) is set to **EmpToMgrAO.ManagerIdEmployeeEO**. Verify that the Alias is set to **EmployeeEO1**.
10. Check the **Reference** checkbox for the Association.

    Note: The Reference checkbox indicates if you want the information from the entity object to be treated as reference information in this view object, with automatic lookup. Attribute values are dynamically fetched from the entity cache when a controlling key attribute changes. In addition to lookup, a reference EO does not allow DML activity. By setting the second EO, EmployeeEO1, to reference, that EO does not support DML. This will prevent a manager record from being accidentally changed. EmployeeEO supports DML, and data can be changed.
11. Click the **Next** button.

12. In the **Attributes** page, click the following attributes from the **Available** list and shuttle them to the **Selected** list in the following order:

    From **Employee**EO:

    EmployeeId

    FullName

    EmailAddress

    From EmployeeEO1:

    EmployeeId (JDeveloper assigns this attribute the default name of EmployeeId1)

    FullName (JDeveloper assigns this attribute the default name of FullName1)

    EmailAddress (JDeveloper assigns this attribute the default name of EmailAddress1)
13. Select the Next button.
14. In Step 4 of the wizard in the Attributes Setting page, click the **Select Attribute** field drop-down, and click **FullName**.
15. Set the **Attribute Name** to **EmployeeName**.
16. Set the **Query Column Alias** to **EMPLOYEE_NAME**.
17. Click the **Select Attribute** field drop-down, and click **EmailAddress**.
18. Set the **Attribute Name** to **EmployeeEmail**.
19. Set the **Query Column Alias** to **EMPLOYEE_EMAIL**.
20. Click the **Select Attribute** field drop-down, and click **EmployeeId1**.
21. Set the **Attribute Name** to **ManagerId**.
22. Set the **Query Column Alias** to **MANAGER_ID**.
23. Click the **Select Attribute** field drop-down, and click **FullName1**.
24. Set the **Attribute Name** to **ManagerName**.
25. Set the **Query Column Alias** to **MANAGER_NAME**.
26. Click the **Select Attribute** field drop-down, and click **EmailAddress1**.
27. Set the **Attribute Name** to **ManagerEmail**.
28. Set the **Query Column Alias** to **MANAGER_EMAIL**.
29. Click the **Next** button.
30. Check the **Expert Mode** checkbox.

    Note: This will allow you to edit the generated query.

31. In the **Query Statement** text box modify the query so it looks like the following.

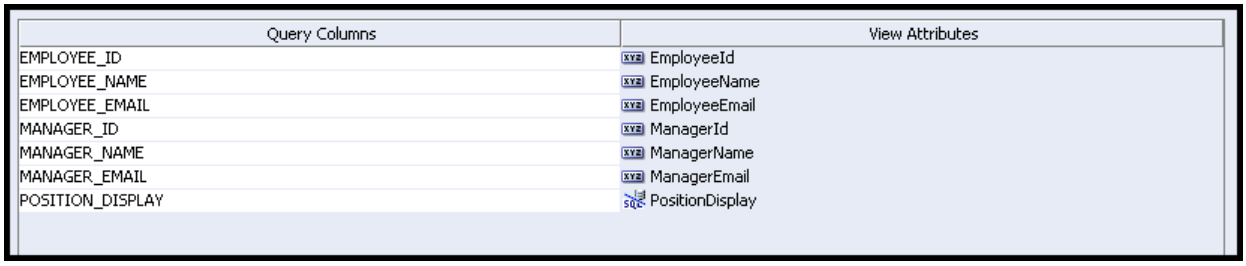
    Note: You must add the outer join **(+)** operator to the `MANAGER_ID / EMPLOYEE_ID` join as this is not automatically derived from the association. You must join to the FWK_TBX_LOOKUP_CODES_VL view to obtain the display name for the employee's assigned position.

```
SELECT EmployeeEO.EMPLOYEE_ID,
       EmployeeEO.FULL_NAME AS EMPLOYEE_NAME,
       EmployeeEO.EMAIL_ADDRESS AS EMPLOYEE_EMAIL,
       EmployeeEO1.EMPLOYEE_ID AS MANAGER_ID,
       EmployeeEO1.FULL_NAME AS MANAGER_NAME,
       EmployeeEO1.EMAIL_ADDRESS AS MANAGER_EMAIL,
       FwkLookupCode.MEANING AS POSITION_DISPLAY
FROM   FWK_TBX_EMPLOYEES EmployeeEO,
       FWK_TBX_EMPLOYEES EmployeeEO1,
       FWK_TBX_LOOKUP_CODES_VL FwkLookupCode
WHERE  EmployeeEO.POSITION_CODE = FwkLookupCode.LOOKUP_CODE
AND FwkLookupCode.LOOKUP_TYPE = 'FWK_TBX_POSITIONS'
```

    Note: The differences between the generated query and this query are:

    - `FWK_TBX_LOOKUP_CODES_VL FwkLookupCode` was added to the `FROM` clause.
    - `EmployeeEO.EMPLOYEE_ID = FwkTbxEmployeesEO1.MANAGER_ID` was removed from the `WHERE` clause.
    - `AND EmployeeEO.POSITION_CODE = FwkLookupCode.LOOKUP_CODE` was added to the `WHERE` clause.
    - `AND FwkLookupCode.LOOKUP_TYPE = 'FWK_TBX_POSITIONS'` was added to the `WHERE` clause.

32. When you are finished with your editing, click the Test button to ensure your syntax is correct.
33. Click the Next button.
34. Verify that the Attribute Mappings to ensure that all of the Query Columns match up to the proper View Attributes. This is a critical check when you create a VO in Expert Mode.



Query Columns	View Attributes
EMPLOYEE_ID	EmployeeId
EMPLOYEE_NAME	EmployeeName
EMPLOYEE_EMAIL	EmployeeEmail
MANAGER_ID	ManagerId
MANAGER_NAME	ManagerName
MANAGER_EMAIL	ManagerEmail
POSITION_DISPLAY	PositionDisplay

35. Click the **Next** button.
36. Click the **Next** button on Step 7 of 8.

    Note: The Bind Variables page (Step 7) allows you to define named bind variables. You will not need those for your VO.

37. Uncheck the **View Object Class: AllEmployeesVOImpl: Generate Java File** checkbox.

38. Check the **View Row Class: AllEmployeesVORowImpl: Generate Java File**, and verify that **Accessors** is checked.
39. Click the **Finish** button.
40. Save your work.

Step 5: Add Your AllEmployeesVO View Object to Your EmployeesAM Application Module

View objects can be used only within the context of a containing application module. Before you can use the AllEmployeesVO in your page, you must add it to the EmployeesAM Application Module.

1. Double-click **EmployeesAM** to open the Application Module Editor.
2. Click **Data Model** from the categories pane.
3. Shuttle **AllEmployeesVO** from the Available View Objects to the Data Model pane.

   Note: When you shuttle AllEmployeesVO to the Data Model, it is automatically assigned an Instance Name of AllEmployeesVO1. This is expected and correct behavior.
4. Click the **OK** button.
5. Save your work.

Task 2: Create Your View-layer Components

Step 1: Create Your EmployeePG Page

1. Right-click your **ClassProject** project, and click **New** from the context menu.
2. Click the **Web Tier** > **OA Components** > **Page** from the New Gallery.
3. Set the **Name** to **EmployeePG**.
4. Set the **Package** to `<student#>.oracle.apps.ak.employee.webui`.
5. Click the **OK** button.

Step 2: Modify the pageLayout Region

1. Click the **EmployeePG** in the Applications Navigator panel.
2. In the Structure panel, select the **region1** item.
3. In the property inspector, change the following properties:

   ID = **PageLayoutRN**

   AM Definition = `<student#>.oracle.apps.ak.employee.server.EmployeesAM`

   Windows Title = **<Your Name> Query**

   Title = **Employees: <Your Name>**

   Note: Be sure that the entries stick by using the enter key on your keyboard after setting the appropriate value.

Step 3: Add a Product Branding Image

Each Oracle Applications page requires a product branding image.

1. Click your **PageLayoutRN** in the **Structure** panel.
2. Right-click it, and click New **>** productBranding from the context menu.
3. JDeveloper creates a **pageLayoutComponents** folder containing a **productBranding** image item (named item1). Click this item and set the following properties:

   ID = **ProdBrand**

   Image URI = `FNDTAPPBRAND.gif`

   Additional Text = **Employee Query Page**

   Note: Additional Text is added to provide accessibility to the application, and will be set by standards on all Oracle E-Business Suite pages.

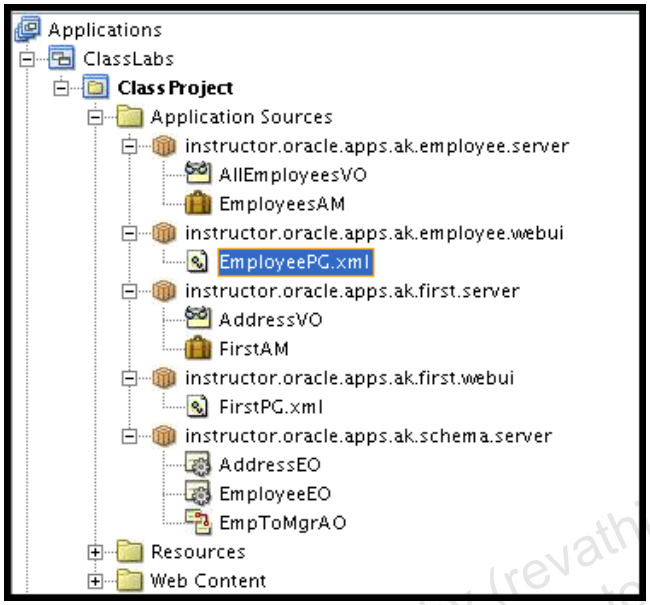
Step 4: Create the Page Instruction Text

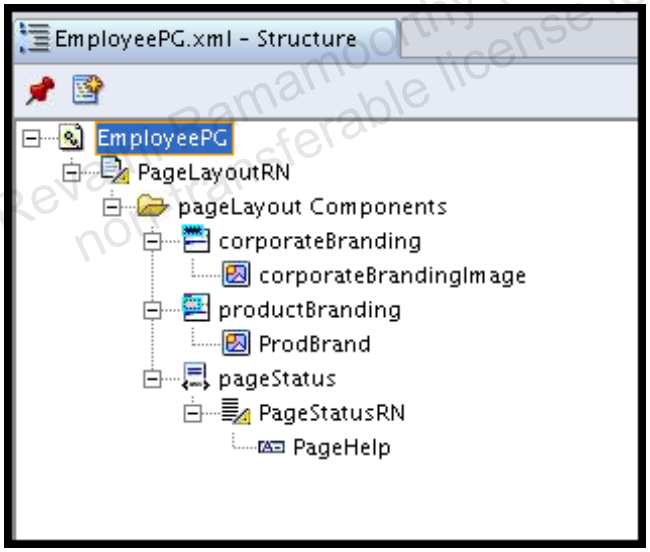
In this step, you are using an Oracle Applications Message Dictionary message to ensure that the text is translatable. Furthermore, all the properties that you specify in JDeveloper that can be seen by the user are translatable.

1. Click the **PageLayoutRN** in the **Structure** panel.
2. Right-click it, and click **New** > **pageStatus** from the context menu.

   Note: Any instruction text or other content added to the pageStatus will render in parallel with any page-level action or navigation buttons that you add.
3. JDeveloper automatically creates a flowLayout region for you beneath the pageStatus component. Click the new **flowLayout** region, and set its **ID** to **PageStatusRN**.
4. Click **PageStatusRN**.

5. Right-click it, and click New **>** Item. Set this item's properties as follows:
   ID = **PageHelp**
   Item Style = **staticStyledText**
   CSS Class = **OraInstructionText**
   Message Appl Short Name = **AK**
   Message Name = **FWK_TBX_T_PAGE_GENERAL**
   Note: Be sure that the entries “stick” by using the enter key on your keyboard





Step 5: Test Your Work

1. Save your work.
2. Click your page (in either the **Structure** panel or the Applications **Navigator** panel).
3. Right-click your page, and click Run from the context menu.
4. Your results should appear as follows:

Task 3: Configure Your Search

In this section, you will create a more complex search region than you did earlier in Lab 3: First OA Framework Page. The search method you are going to use is called an auto customization criteria search. Having an auto customization criteria search will allow you more capabilities than with the simpler results-based search as was performed in the Lab 3: First OA Framework Page.

Tip: See the Search topic in Chapter 4 of the OA Framework Developer's Guide for additional information about implementing searching in your pages. This is a broad topic, and you have just scratching the surface of what you can do in this lab.

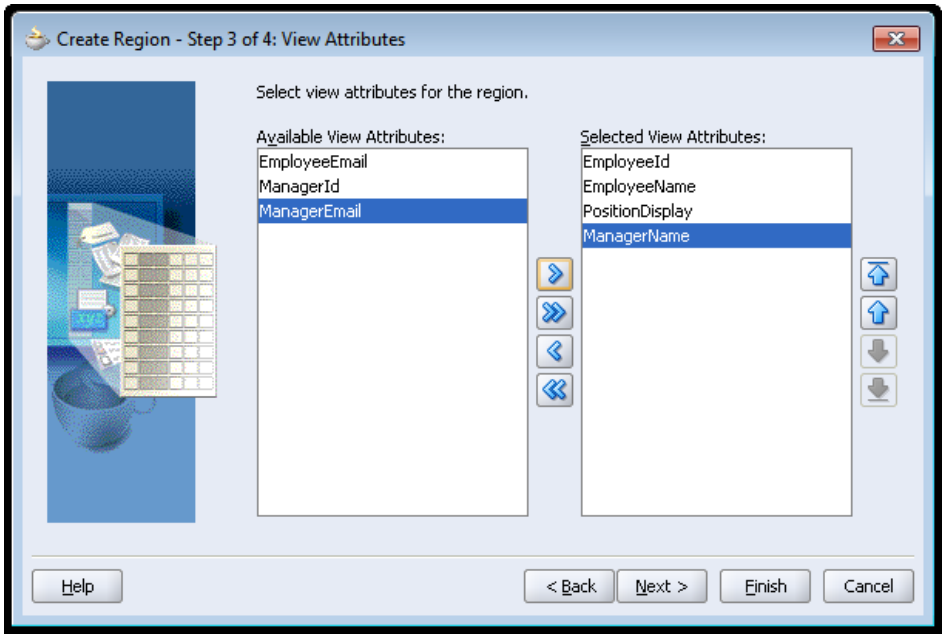
Construction Mode	Region Construction Impact	Search Execution Impact
resultsBasedSearch	OA Framework automatically renders both the Simple and Advanced search regions based on the designated queryable items in the associated table or HGrid. **Note:** The search regions automatically include both a **Go** and a **Clear** button.	OA Framework automatically executes the underlying search when the user selects the **Go** button. If the underlying view object is "dirty" (it has pending changes), OA Framework displays an error message instead of executing the query.
autoCustomizationCriteria	OA Framework automatically renders both the Simple and Advanced search regions based on the corresponding Simple search and Advanced search regions that you define and specify as named children of the query region. **Note:** The search regions automatically include a **Go** button. In addition, the Advanced search region includes a **Clear** button.	OA Framework automatically executes the underlying search when the user selects the **Go** button. However, developers *must* explicitly define mappings between items in the Search panel and items in the table/HGrid region. As in the resultsBasedSearch case, OA Framework displays an error message instead of executing the query if the underlying view object has pending changes.
none	The Search regions are rendered based on the Simple Search and Advanced Search regions that you define and specify as named children of the query region. **Note:** You must implement your own **Go** button in this mode.	The underlying search must be executed by the developer.

Step 1: Add a Query Region to Your EmployeePG Page

1. Select the **PageLayoutRN** in the **Structure** panel.
2. Right-click it, and click New **>** Region from the context menu.
3. Click the new region (**region1**), and set its properties as follows:
   ID = **QueryRN**
   Region Style = **query**
   Construction Mode = **autoCustomizationCriteria**
   Include Simple Panel = **true**
   Include Views Panel = **true**

Step 2: Add a Results Region to Your EmployeePG Page

1. Click the **QueryRN** in the **Structure** panel.
2. Right-click it, and click New **>** Region **Using Wizard** from the context menu. If the Welcome page appears, click the Next button.
3. Set the **Application Module** to your `<student#>.oracle.apps.ak.employee.server.EmployeesAM`.
4. Do not check the **Use this as Application Module Definition for the region** checkbox.
5. Click the **AllEmployeesVO1** from the Available View Usages.
6. Click the **Next** button.
7. Set the **Region ID** to **ResultsRN**.
8. Set the **Region Style** to **table**.
9. Click the **Next** button.
10. Shuttle the following attributes from the **Available View Attributes** list to the **Selected View Attributes** list in this order:
    EmployeeId
    EmployeeName
    PositionDisplay
    ManagerName
    Note: If needed, use the Up and Down arrow buttons in the Selected View Attributes pane to move the attributes into the correct order.



11. Click the **Next** button.
12. In the Region Items table, set the **Attribute Set** column of the items as follows:

    Note: While it is possible to use the lookups associated with the field to set the Attribute Sets, JDeveloper uses a non-intuitive interface for doing this. It is easier to simply type the value into the field.

    EmployeeId = **`/oracle/apps/fnd/framework/toolbox/attributesets/FwkTbxEmployees/EmployeeId`**

    EmployeeName = **`/oracle/apps/fnd/framework/toolbox/attributesets/FwkTbxEmployees/FullName`**

    PositionDisplay = **`/oracle/apps/fnd/framework/toolbox/attributesets/FwkTbxEmployees/Position`**

    ManagerName = **`/oracle/apps/fnd/framework/toolbox/attributesets/FwkTbxEmployees/FullName_Manager`**

    Tip: You can use the Search capability to help find your attribute sets. Type `/oracle/apps/fnd/framework/toolbox/attributesets/` (or a shorter path such as `/oracle/apps/fnd/framework`) into the **Package** field in the Search window. Check Search in: Entire MDS XML Path. Click the Search button (not the **Browse** button).
13. Set each and every item in the **Region Items** table to Style = **messageStyledText**.
14. Click the **Finish** button.

15. Click the **ResultsRN** region in the structure panel, and set the following properties:

    Additional Text = **Employees Table**

    Width = **100%**

    Note: It is critical that you include the % (percent) symbol. If you enter 100 without the % symbol, the table will only be 100 pixels wide. By using 100%, you are telling OA Framework to use all of the available width.

    User Personalizations = **True**

Step 3: Set the ResultsRN's Items Properties

1. For the **EmployeeId** item, set the following properties:

   Search Allowed = **True**

   Sort Allowed = **ascending**

   Initial Sort Sequence= **first**

   Selective Search Criteria = **True**

   Note: Setting the Selective Search Criteria property to True designates that the item will be used as a user-specified search value. This is done to ensure a performant search. At least one item must be set to True for this value. This prevents a "Wild Card" search when set to True.

   User Personalization = **True**

   Tip: For more information on the Sort Allowed and Initial Sort Sequence properties, refer to the Developer's Guide topic on Tables > Sorting. For more information on the Search Allowed and Selective Search Criteria properties, refer to the Developer's Guide topics on Search > Declarative Implementation: Results Based Search and Search > Declarative Implementation: Auto Customization Criteria.
2. For the **EmployeeName** item, set the following properties:

   Search Allowed = **True**

   Selective Search Criteria = **True**

   User Personalization = **True**
3. For the **PositionDisplay** item, set the following properties:

   Search Allowed = **True**

   User Personalization = **True**
4. For the **ManagerName** item, set the following properties:

   Search Allowed = **True**

   Destination URI = **mailto:{@ManagerEmail}**

   User Personalization = **True**

Step 4: Add a Simple Search Region to Your Query

1. Click the **QueryRN** in the **Structure** panel.
2. Right-click it, and click New **>** simpleSearchPanel from the context menu. A header region (region2) and a messageComponentLayout region (region1) will be created automatically.
3. Click the header region (**region2**), and set its **ID** to **SimpleSearchRN**.
4. Click the default messageComponentLayout region created under the **simpleSearchPanel** folder (**region1**), and set its **ID** to **CustomSimpleSearchRN**.

Step 5: Create an Employee Name Search Item for Your Query.

1. Click **CustomSimpleSearchRN** region in the **Structure** panel.
2. Right-click it, and click New **>** messageTextInput from the context menu.
3. Click this item, and set its properties as follows:

   ID = **SearchEmpName**

   Attribute Set = **`/oracle/apps/fnd/framework/toolbox/attributesets/FwkTbxEmployees/FullName`**

   Selective Search Criteria = **True**

   Maximum Length = **240**

   CSS Class = **OraFieldText**

Step 6: Create an Employee Number Search Item for Your Query

1. Click **CustomSimpleSearchRN** region in the **Structure** panel.
2. Right-click it, and click New **>** messageTextInput from the context menu.
3. Click this item, and set its properties as follows:

   ID = **SearchEmpNum**

   Attribute Set = **`/oracle/apps/fnd/framework/toolbox/attributesets/FwkTbxEmployees/EmployeeId`**

   Selective Search Criteria = **True**

   Data Type = **NUMBER**

   Maximum Length = **15**

   CSS Class = **OraFieldText**

Step 7: Create Search Mappings for Your Query

OA Framework uses the mappings that you define between your custom search items and columns in the ResultsRN to automatically handle the query when the user selects the Go button.

1. Click the query Components folder in the **Structure** panel.
2. Right-click it, and click New **>** simpleSearchMappings from the context menu.
3. Click the default mapping created under the **simpleSearchMappings (**queryCriteriaMap1**),** and set its properties to the following:

   ID = **EmpNameMap**

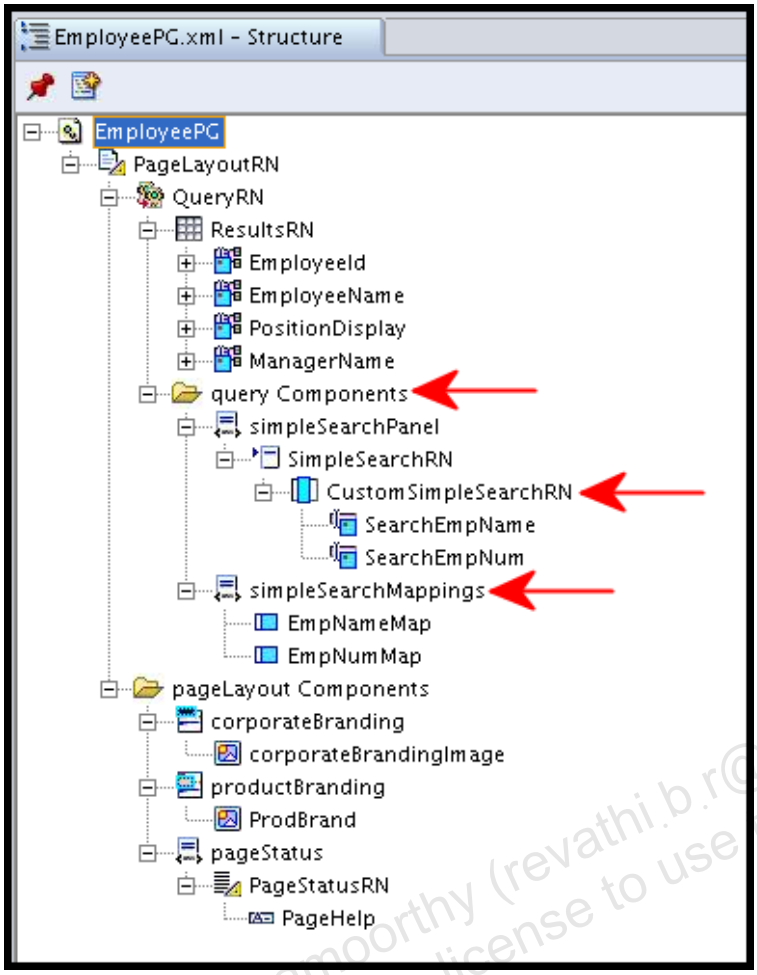
   Search Item = **SearchEmpName**

   Results Item = **EmployeeName**
4. Click the simpleSearchMappings folder in the **Structure** panel.
5. Right-click it, and click New **>** queryCriteriaMap from the context menu.
6. Click the default mapping, **queryCriteriaMap1**, (it has the same name as in the above step because you renamed the above step) created under the **simpleSearchMappings,** and set its properties to the following:

   ID = **EmpNumMap**

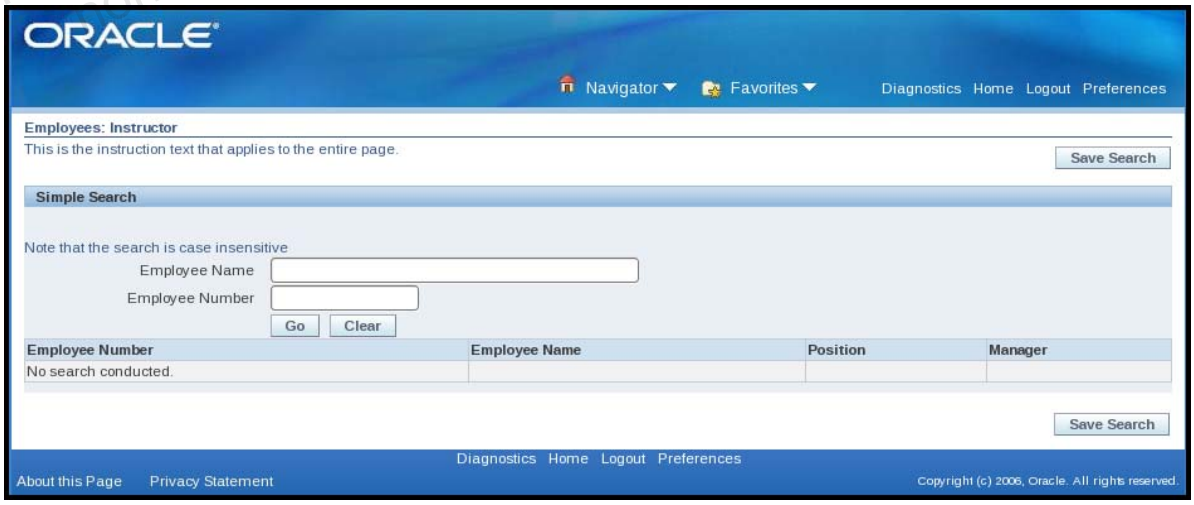
   Search Item = **SearchEmpNum**

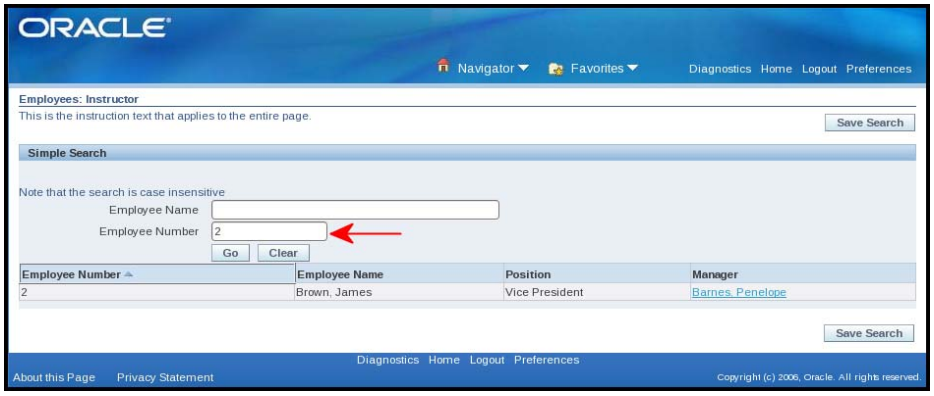
   Results Item = **EmployeeId**

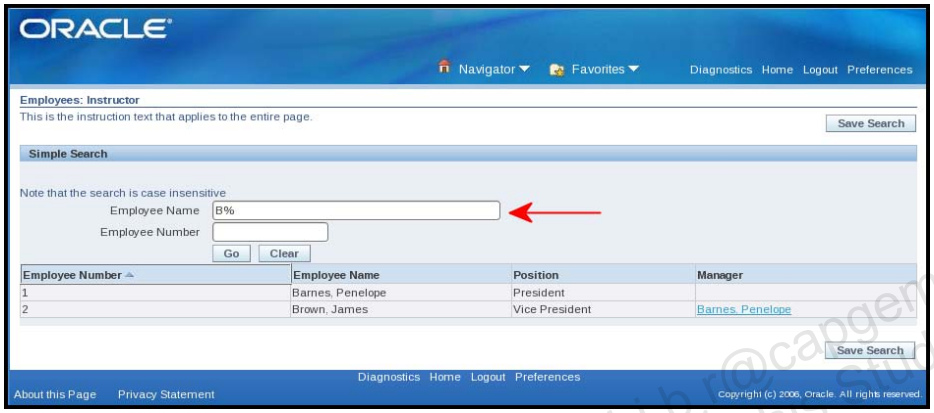


Step 8: Test Your Work

1. Save your work
2. Run your page. Your results should appear as follows:







Adding a Popup

Popups are an enhanced feature in Oracle E-Business suite 12.1.2 and OA Framework 12.1.2.

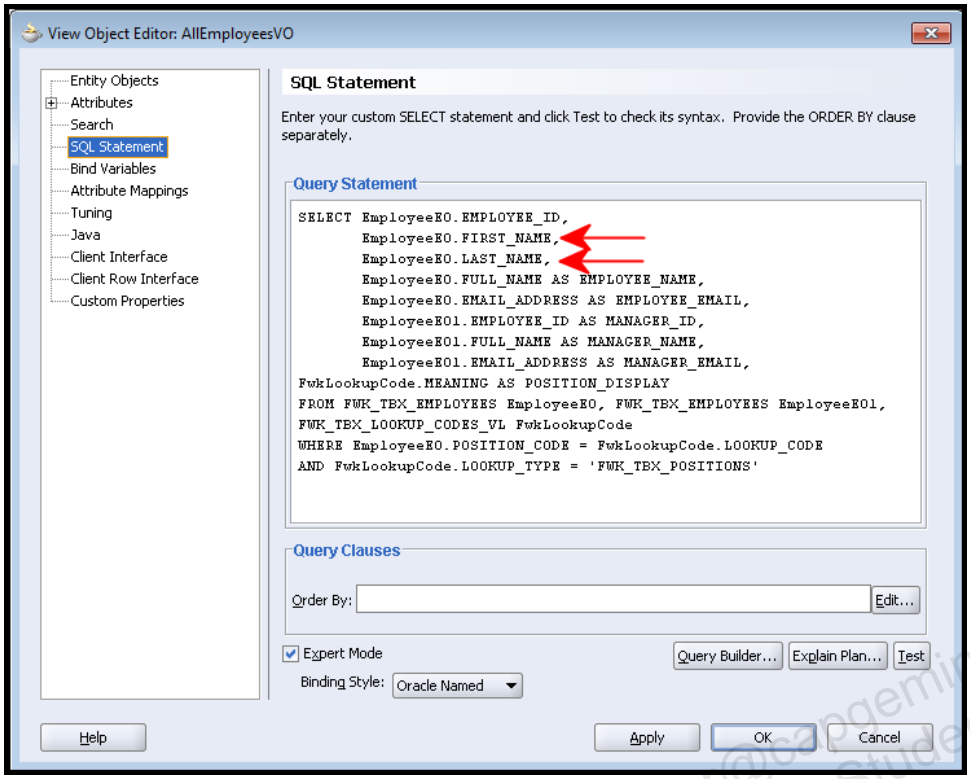
In the FWK_TBX_EMPLOYEES table, first and last name are stored as separate entities. In the example previously, the Full Name is returned. You are going to create a popup that displays the first and last name of an employee when you hover over the employee number in the results table.

Step 9: Add new Attributes to the existing AllEmployeesVO

Your first step is to add attributes to the AllEmployeesVO. This will add the FirstName and LastName into the VO, and make them available.

1. Right-click the **AllEmployeesVO** in the application navigator, and click **edit**.
2. Click on the **SQL Statement – Query Statement**. Add the following entries to the query:

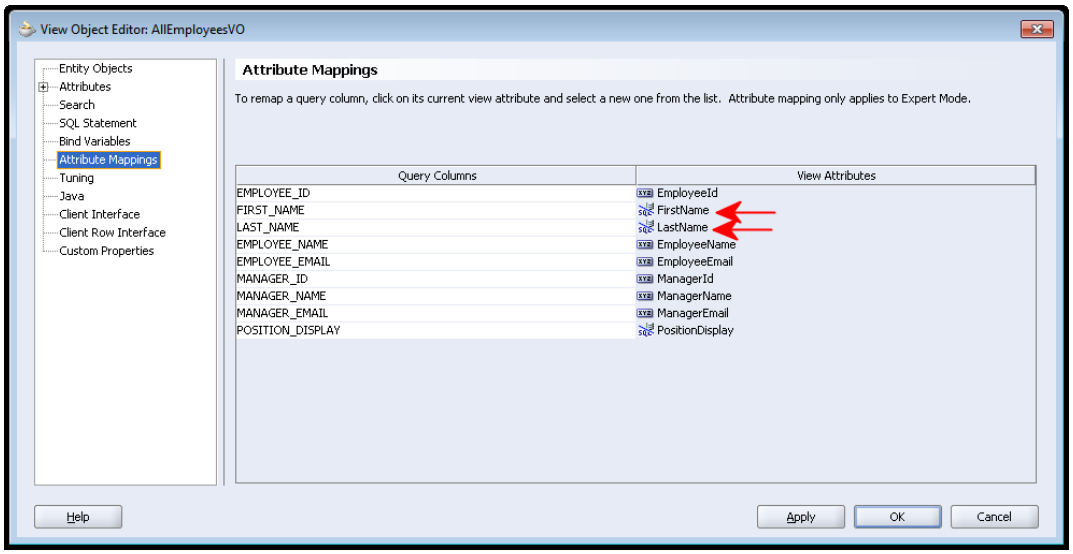
```
EmployeeEO.FIRST_NAME,
EmployeeEO.LAST_NAME,
```

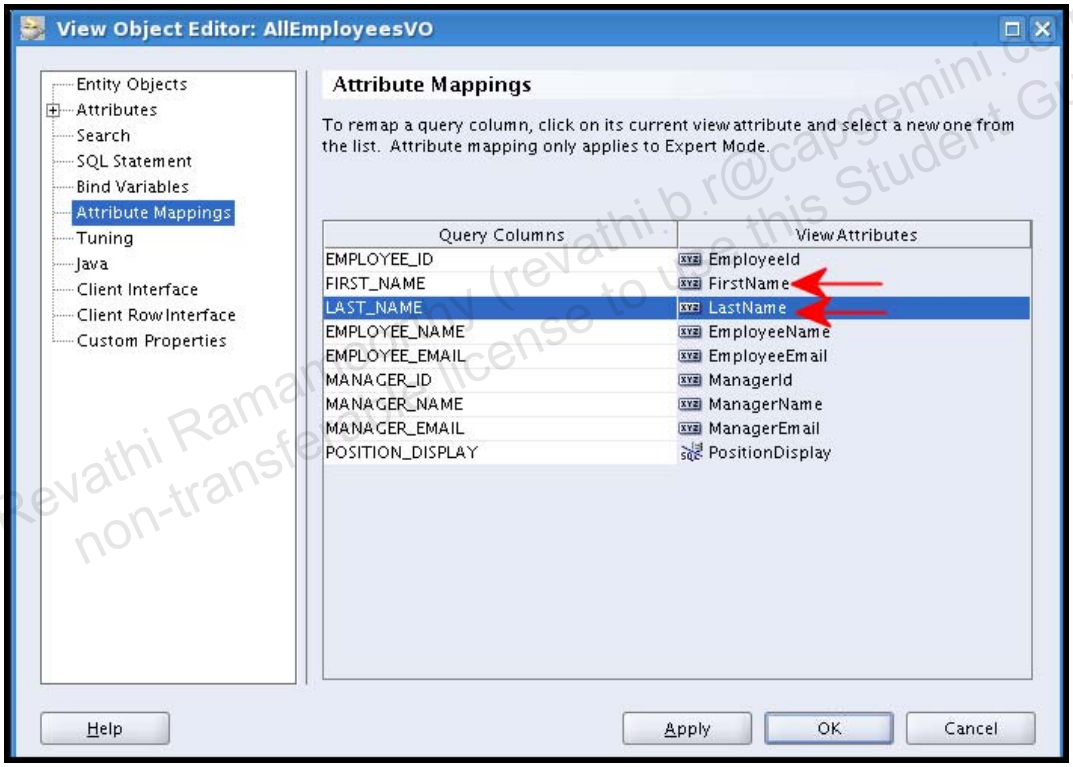


3. Click the **Apply** button. Do not click the **OK** button.
4. While still in edit mode, click **Attribute Mappings**.

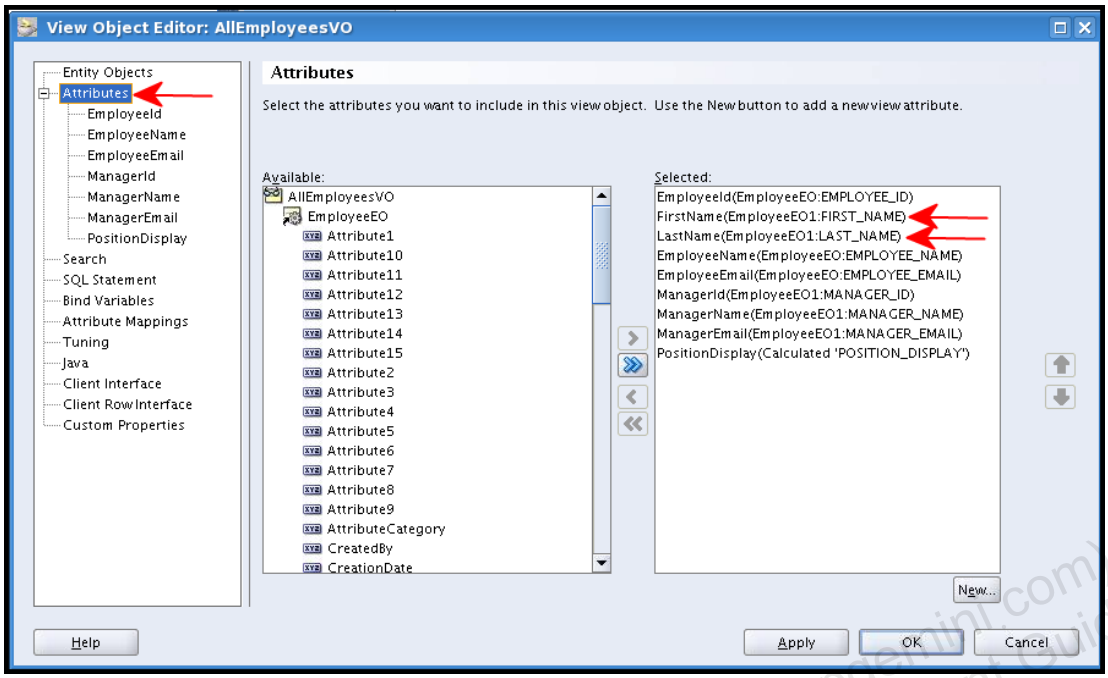
   Note: Because you added the attributes via SQL, the view attributes are SQL transients at this point. Change the view attributes to reflect the entity attributes in the underlying EO and match them.

5. Click the **Apply** button when you have selected the correct View Attributes





6. Verify the View Attributes. Click on **Attributes**, and verify the following:



7. Click the OK button when you have verified that the attributes are set correctly.

Step 10: Create a Stand Alone Region

You will create a stand-alone region that is going to contain the attributes you added to the AllEmployeesVO as well as the EmployeeId

- **EmployeeId**
- **FirstName**
- **LastName**

1. Right-click the **<student#>.oracle.apps.ak.employee.webui** package in the Application Navigator.
2. Click **New** > **OA Components** > **Region**.
3. Click the **OK** button
4. Set the **Name** to **FirstLastNameRN**.
5. Set the **Package** to `<student#>.oracle.apps.ak.employee.webui`.
6. Set the **Style** to **messageComponentLayout**.
7. Click the **OK** button.

Step 11: Set the properties on FirstLastNameRN

1. Click the **FirstLastNameRN** in the structure pane, and set the properties as follows:
   AM Definition = `<student#>.oracle.apps.ak.employee.server.EmployeesAM`
   Width = **100%**
   Rows = **3**
   Columns = **1**

Step 12: Create the Display Items

1. Right-click the **FirstLastNameRN** in the structure pane, and click **New** > **messageStyledText**. Repeat this action two more times.

   Note: In the Windows version of JDeveloper you will have **item** as your choice.
2. Set and verify the following properties on **item1:**

   ID = **EmployeeNumber**

   Item Style = **messageStyledText**

   Attribute Set = **`/oracle/apps/fnd/framework/toolbox/attributesets/FwkTbxEmployees/EmployeeId`**

   Data Type = **Number**

   View Instance = **AllEmployeesVO1**

   View Attribute = **EmployeeId**

   CSS Class = **OraFieldText**
3. Set and verify the following properties on **item2**:

   ID = **FirstName**

   Item Style = **messageStyledText**

   Attribute Set = **`/oracle/apps/fnd/framework/toolbox/attributesets/FwkTbxEmployees/FirstName`**

   Data Type = **VARCHAR2**

   View Instance = **AllEmployeesVO1**

   View Attribute = **FirstName**

   CSS Class = **OraFieldText**
4. Set and verify the following properties on **item3**:

   ID = **LastName**

   Item Style = **messageStyledText**

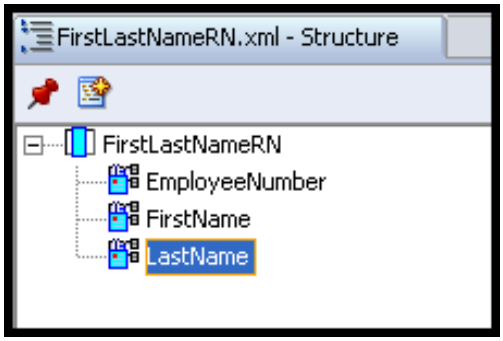
   Attribute Set = **`/oracle/apps/fnd/framework/toolbox/attributesets/FwkTbxEmployees/LastName`**

   Data Type = **VARCHAR2**

   View Instance = **AllEmployeesVO1**

   View Attribute = **LastName**

   CSS Class = **OraFieldText**

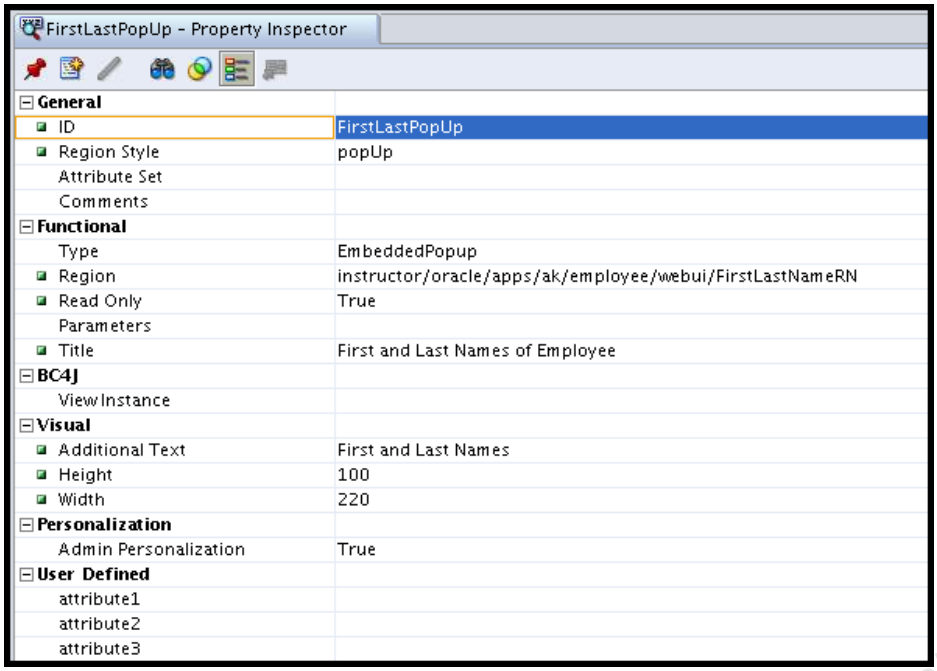


Step 13: Create the Popup Region

You want to display the Popup when you hover over the EmployeeId. To do that, you are going to create a new region as part of the QueryRN and ResultsRN in the EmployeePG. When you created the query, the query mechanism automatically fetches the underlying data and has it ready. By putting an Embedded Popup in this region, you make use of the fetch as part of the original AllEmployeesVO.

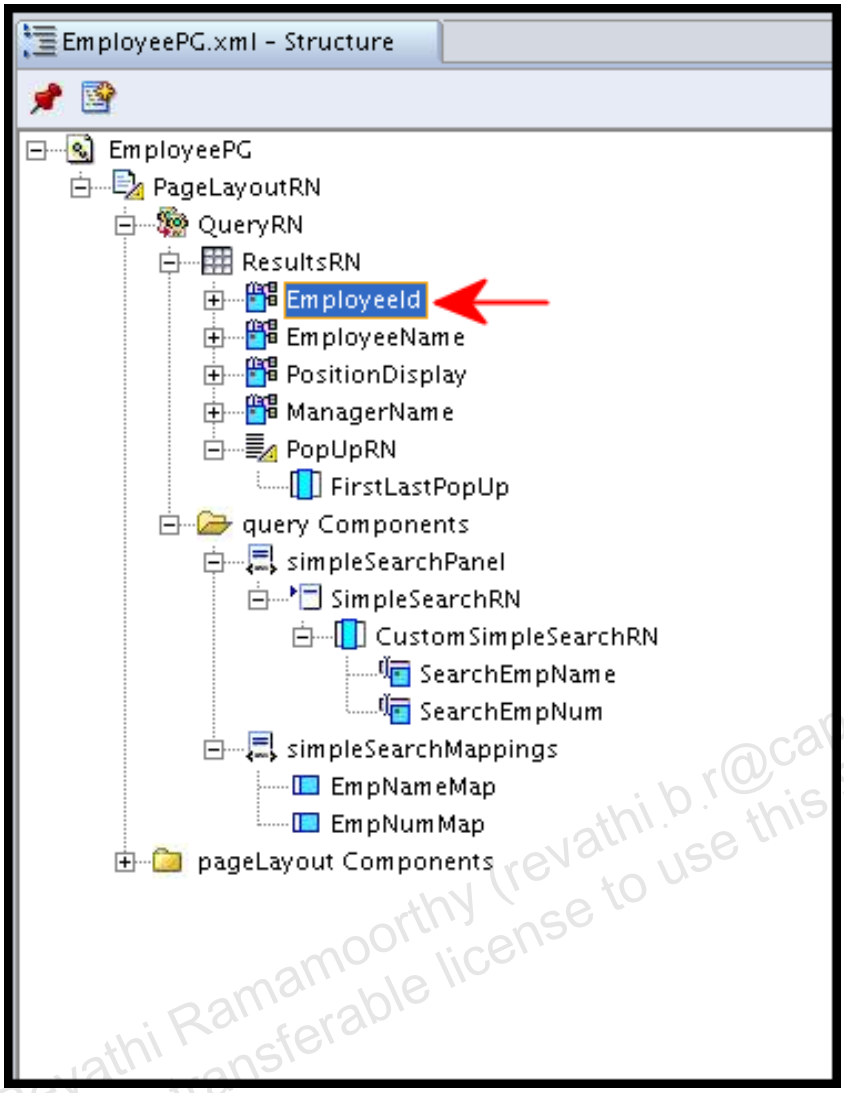
1. Click the **EmployeePG.xml** in the Applications Navigator panel, and then click the structure pane.
2. Right-click the **ResultsRN**, and click **New** > **flowLayout**.
3. Set the **ID** to **PopUpRN**.
4. Right-click the **PopUpRN**, and click **New** > **Region**.
5. Set **region1's ID** as **FirstLastPopUp**.
6. Set the **Style** as **popUp**.

   Note: EmbeddedPopup is the default styles, for more information on Popups, see the Oracle Application Framework Developer's Guide 12.1.2 or higher.
7. In the Property Inspector, within the Functional region, set the Region property to the stand alone region you created
   `<student#>/oracle/apps/ak/employee/webui/FirstLastNameRN`
8. Set **Read Only** to **True**.
9. Set **Title** to **First and Last Names of Employee**.
10. Set **Additional Text** to **First and Last Names**.
11. Set **Height** to **100**.
12. Set **Width** to **220**.

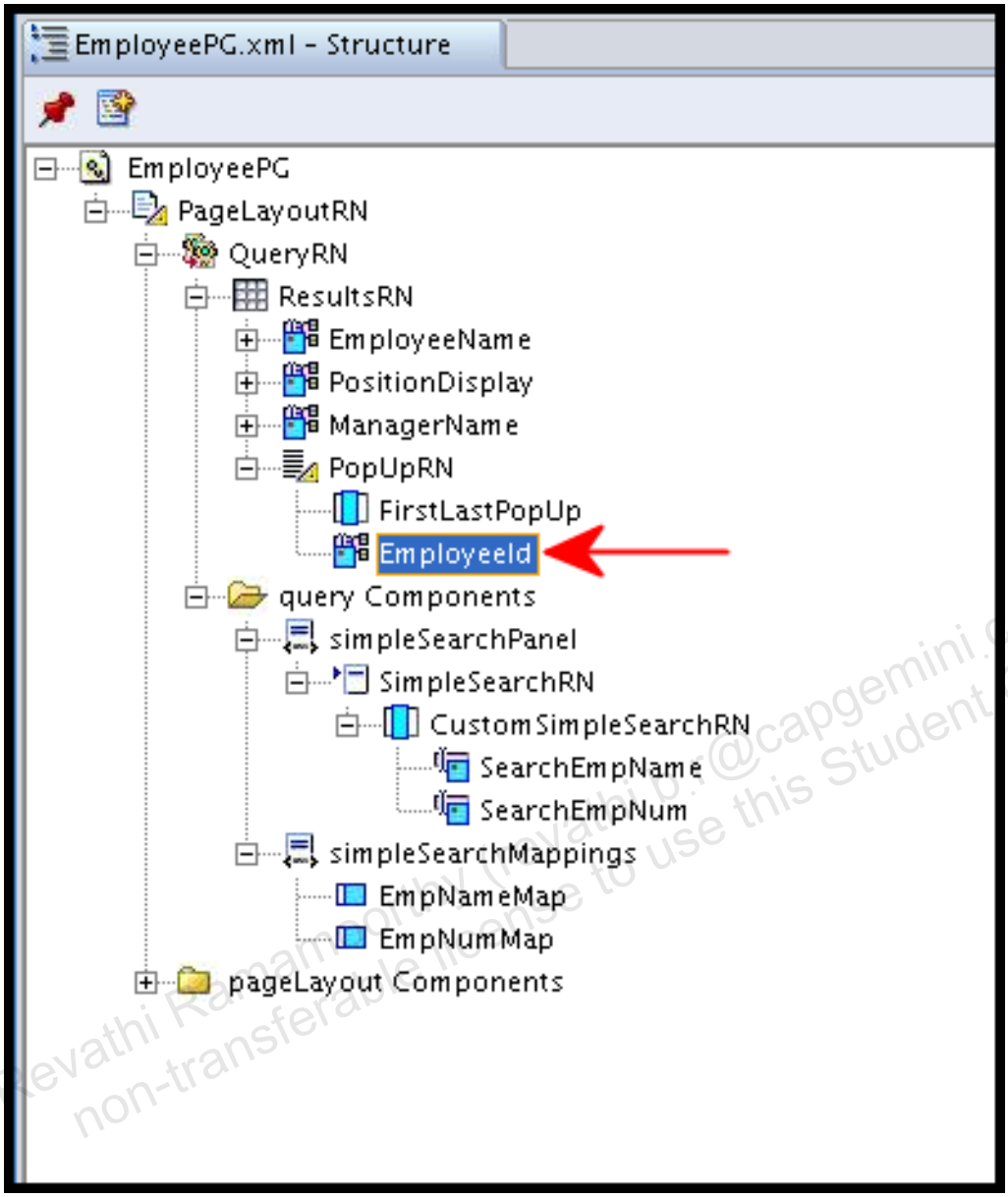


13. This is the EmployeePG structure as this point.

    Note: The PopUpRN is a sibling to the other columns that appear as part of the ResultsRN.



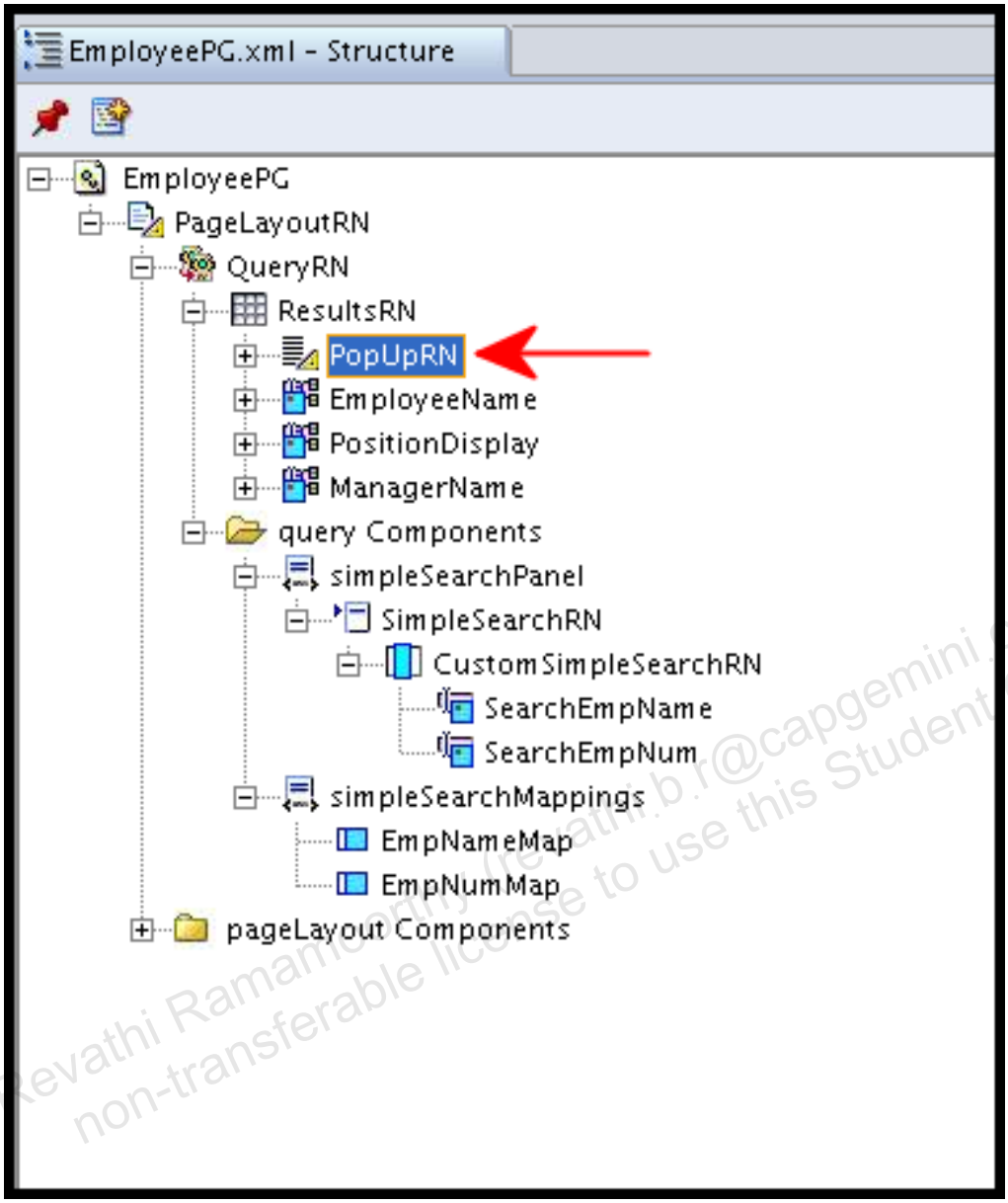
14. Since the PopUpRN is part of the ResultsRN, there will be an additional table column that will appear blank if you do not change it. To clean up this extraneous table column in the results table, drag and drop the **employeeId** into the PopUpRN as shown below.
15. When you drag the **EmployeeId**, you will be given options to treat it as a Child or Sibling. Click the **Child** button.



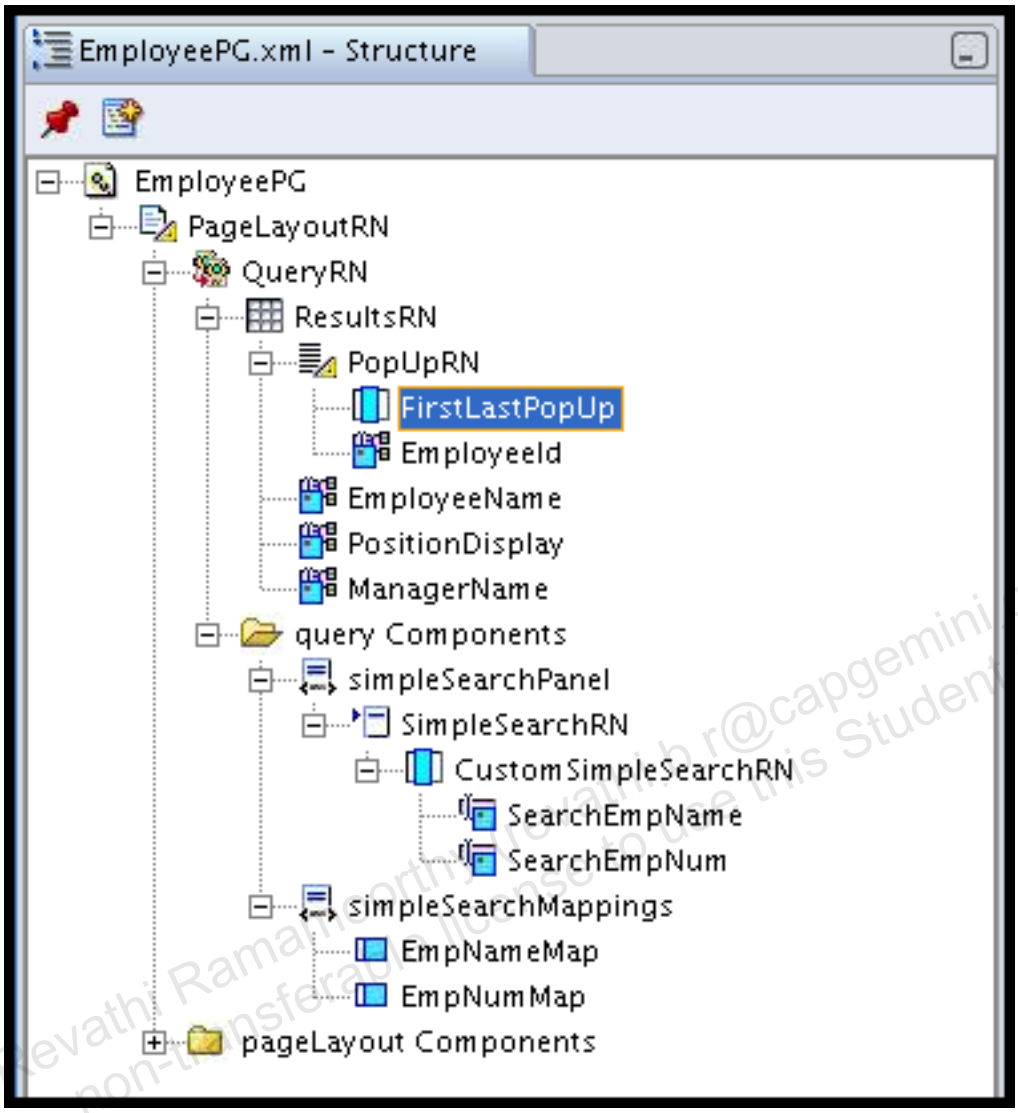
16. When you drag the **EmployeeId** into the PopUpRN, set the PopUpRN's **Visual Prompt** to **Employee Number**.

    Note: If you do not, then there will be a blank column that contains the employee number. This happens because the EmployeeId becomes a child of the PopUpRN. Since the PopUpRN is within a table, there will be a blank heading if you do not set the Visual Prompt property.

17. Click the **[-]** icon to collapse the PopUpRn, and drag it up to the top of the ResultsRN display attributes.

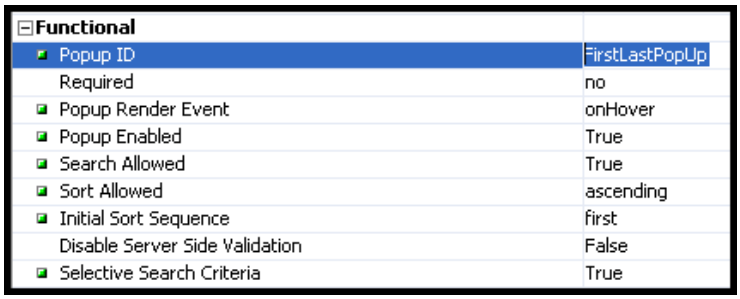


18. Your final EmployeePG structure will look like the below image.



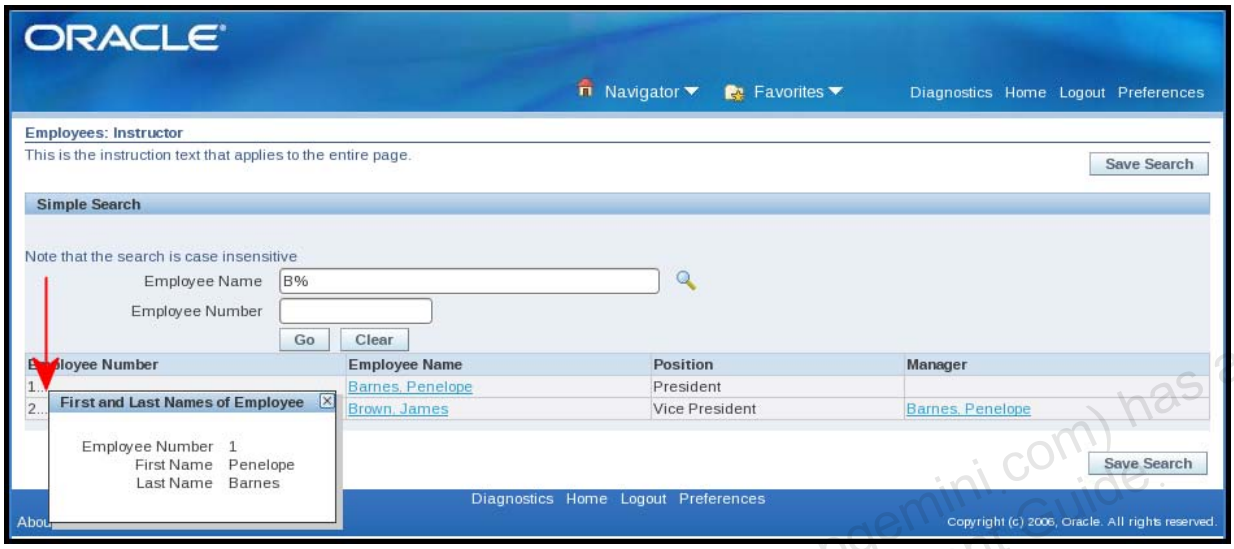
Step 14: Enable the Popup

1. Click the **EmployeePG**.
2. Set the following properties on **EmployeeId**:
   Popup ID = **FirstLastPopUp**
   Popup Render Event = **onHover**
   Popup Enabled = **True**



3. Test your page. Run **EmployeePG**.

4. When the page renders, notice there is now an ellipse on the EmployeeId field. If you hover your mouse over the ellipse, the popup will render. Notice in the image below that there is the correct number of columns in the rendered page. If you had not dragged the EmployeeId region under the PopUpRN region, you would have an additional blank column.



Task 4: Add a List of Values (LOV) to Query

In this section, you will create a reusable list of values (LOV) that you will then configure for use in the EmployeePG page (you will use this same LOV later in the Create page). A reusable, sharable LOV has both Model-layer and View-layer components.

Step 1: Create Your LOV Application Module

Related LOV view objects should be grouped into a common application module. For example, any LOV view objects that you create for this class, that is, the Employees application you are currently working on, should be included in one application module.

1. Right-click your **ClassProject** project, and select **New** from the context menu.
2. Click the **Business Tier** > **ADF Business Components** > **Application Module** from the New Gallery to open the Create Application Module wizard.
3. Set the **Package** to `<student#>.oracle.apps.ak.lov.server`.
4. Set the **Name** to **EmpNameLOVAM**.
5. Click the **Next** button until you arrive at step 4 of 4: Java.
6. Uncheck the **Generate Java Files(s)** checkbox.
7. Click the **Finish** button.
8. Save your work.

Step 2: Create Your LOV View Object

1. Click the **<student#>.oracle.apps.ak.lov.server** package in the Applications **Navigator panel.**
2. Right-click it, and click **New View Object...** from the context menu to open the Create View Object wizard. If the Welcome page appears, click the **Next** button.
3. Set the **Name** to **EmpNameLOVVO.**
4. Select the **Read-only Access** option.
5. Click the **Next** button.

6. Enter the following query into the **Query Statement** text field.

   Note: This LOV is querying active employees, and for the sake of simplicity, the business rule assumes that the presence of any `end_date` -- regardless of what date is relative to sysdate -- indicates that the employee is inactive. Before you complete you Query Statement, click the **Test** button to ensure your syntax is correct.

   ```
   SELECT FULL_NAME AS EMPLOYEE_NAME,
   EMPLOYEE_ID AS EMPLOYEE_NUMBER,
   EMAIL_ADDRESS
   FROM FWK_TBX_EMPLOYEES
   WHERE END_DATE IS NULL
   ```

7. Click the **Next** button until you arrive at Step 7 of 7:Java.
8. On the **Java** page, set the following:

   Uncheck the **View Object Class: EmpNameLOVVOImpl**: **Generate Java File** checkbox.

   Check the View Row Class: EmpNameLOVVORowImpl: **Generate Java File** checkbox.

   Note: Accessors are selected automatically.
9. Click the **Finish** button.

   Note: You just finished creating a VO without an underlying EO. The reason is that a Look Ahead View – LOV - just needs data to display and select from without updating the data in the underlying table.

Step 3: Add the EmpNameLOVVO to the EmpNameLOVAM

View objects can be used only within the context of a containing application module. Before you can use the EmpNameLOVVO in your LOV, you must add it to the EmpNameLOVAM application module.

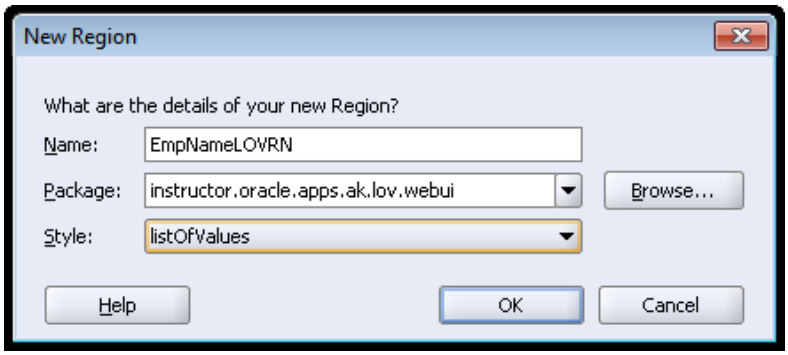
1. Double-click the **EmpNameLOVAM** in the Applications Navigator panel to open the Application Module Editor.
2. Click **Data Model** from the categories pane.
3. Shuttle **EmpNameLOVVO** from Available View Object pane to the Data Model pane. When shuttled over to the data model, you should see EmpNameLOVVO1 under the EmpNameLOVAM.
4. Click the **Apply** button.
5. Click the **OK** button.

Step 4: Create Your LOV Region

Since the LOV you are creating can be used in many different pages, it must be created as a shared, standalone region. When you created the PopUp in the previous steps, you already created such a “Stand Alone” region. The PopUp wasn’t a true stand alone region as it was designed to be used in a results table. What makes the “Stand Alone” region you are going to create now, is that it will not share an AM or transaction state, but will be truly “Stand Alone” and reusable.

1. Click the **ClassProject** project in the Applications Navigator.
2. Right-click it, and click **New** from the context menu.
3. Click **Web Tier** > **OA Components** > **Region** from the Items pane.
4. Set the **Name** to **EmpNameLOVRN**.

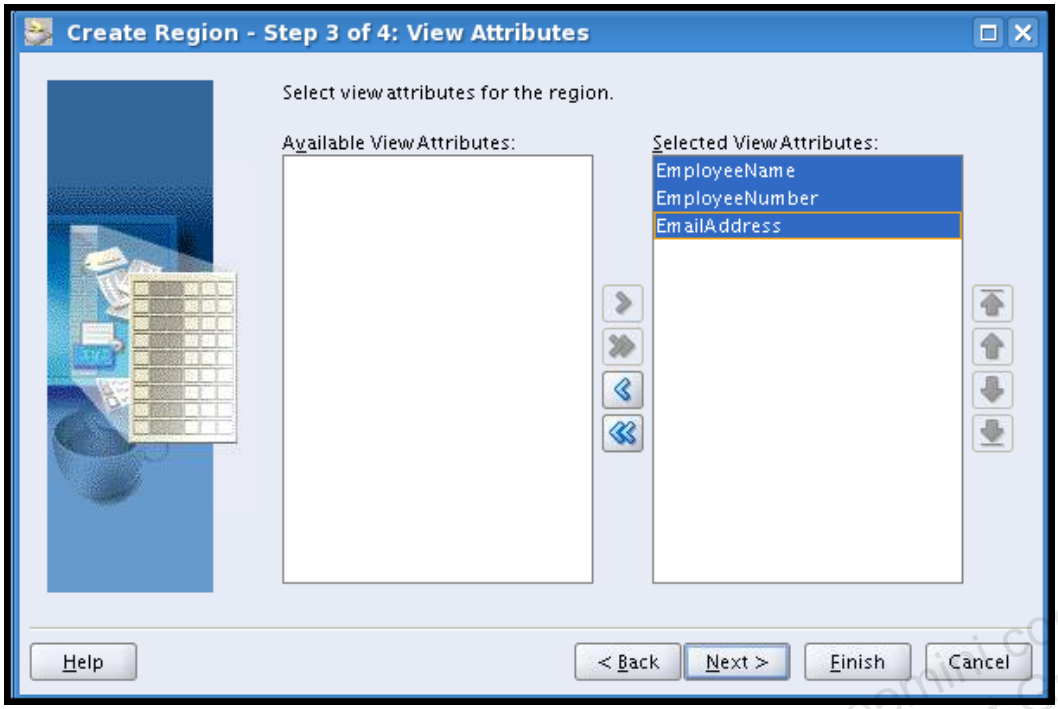
5. Set the **Package** to `<student#>.oracle.apps.ak.lov.webui`.
6. Set the **Style** to **listOfValues**.
7. Click the **OK** button.



8. Click **EmpNameLOVRN** in the **Structure** panel, and set the properties as follows:
   Scope = **Public**
   Advanced Search Allowed = **True**
   AM Definition = **<student#>.oracle.apps.ak.lov.server.EmpNameLOVAM**.

Step 5: Add a Table to Your EmpNameLOVRN LOV

1. Click the **EmpNameLOVRN** in the **Structure** panel.
2. Right-click it, and click **New > table Using Wizard** from the context menu. If the Welcome page appears, click the **Next** button.
3. Set the **Application Module** drop-down to your **EmpNameLOVAM**.
4. Do not check the **Use this as Application Module Definition for the region** checkbox.
5. Click **EmpNameLOVVO1** from the **Available View Usages**.
6. Click the **Next** button.
7. Set the **Region ID** to **EmpNameResultsRN**.
8. Set the **Region Style** to **table**.
9. Click the **Next** button.
10. Shuttle all of the Available View Attributes to Selected View Attributes using the **[>>]** button.

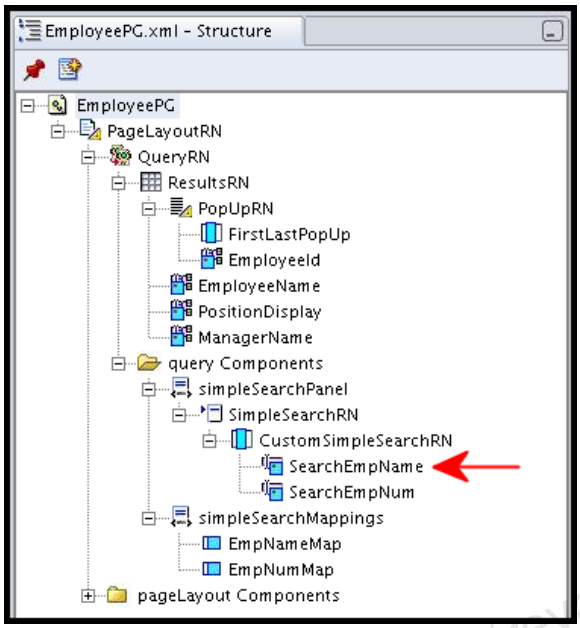


11. Click the **Next** button.
12. In the Region Items table, set the **Attribute Set** column of the items as follows:
    EmployeeName = **`/oracle/apps/fnd/framework/toolbox/attributesets/FwkTbxEmployees/FullName`**
    EmployeeNumber = **`/oracle/apps/fnd/framework/toolbox/attributesets/FwkTbxEmployees/EmployeeId`**
    EmailAddress = **`/oracle/apps/fnd/framework/toolbox/attributesets/FwkTbxEmployees/EmailAddress`**
13. Click the **Finish** button.
14. Click the **EmpNameResultsRN** in the Structure panel to expand it.
15. Click the **EmpNameResultsRN** region, and set the **Additional Text** property to **Employee Name List of Values**.
16. Click the **EmployeeName** item, and set the properties as follows:
    Search Allowed = **True**
    Selective Search Criteria = **True**
    CSS Class: **OraFieldText**
17. Click the **EmployeeNumber** item, and set the properties as follows:
    Search Allowed = **True**
    Selective Search Criteria = **True**
    CSS Class: **OraFieldText**
18. Click the **EmailAddress** item, and set the properties as follows:
    CSS Class: **OraFieldText**
19. Save your work.

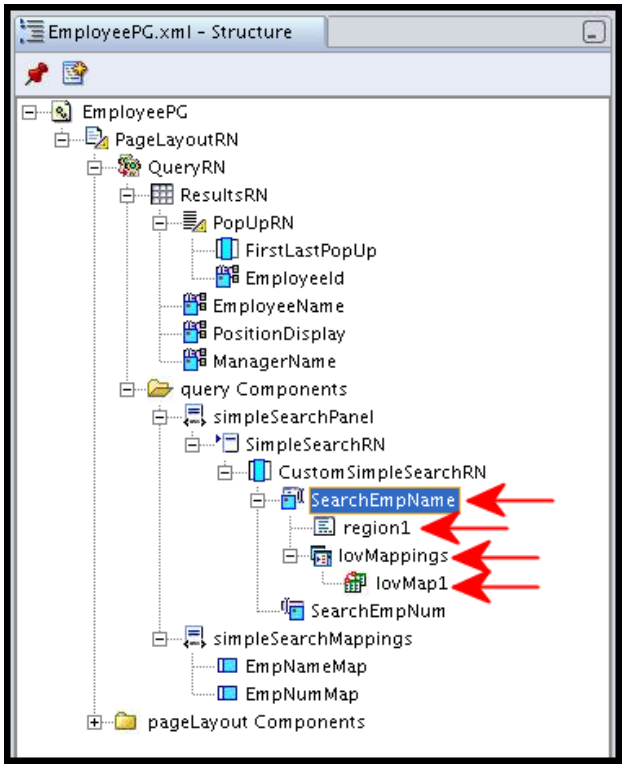
Step 6: Modify the EmployeePG page to use the EmpNameLOVRN LOV

When you first created the SearchEmpName item in your query, you set its style to messageTextInput (a standard text entry field). Now, you will configure it as an LOV.

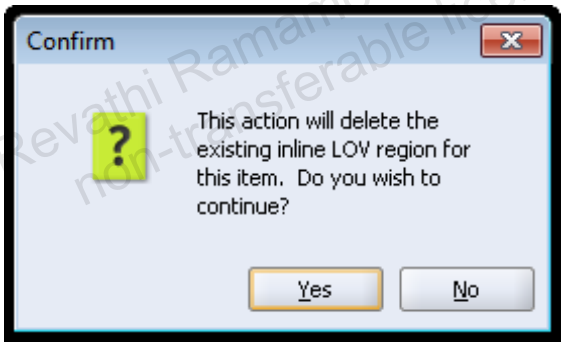
1. Click the **EmployeePG** in the Applications Navigator panel.
2. Click and drill-down to **SearchEmpName** in the **query Components** from the Structure panel.

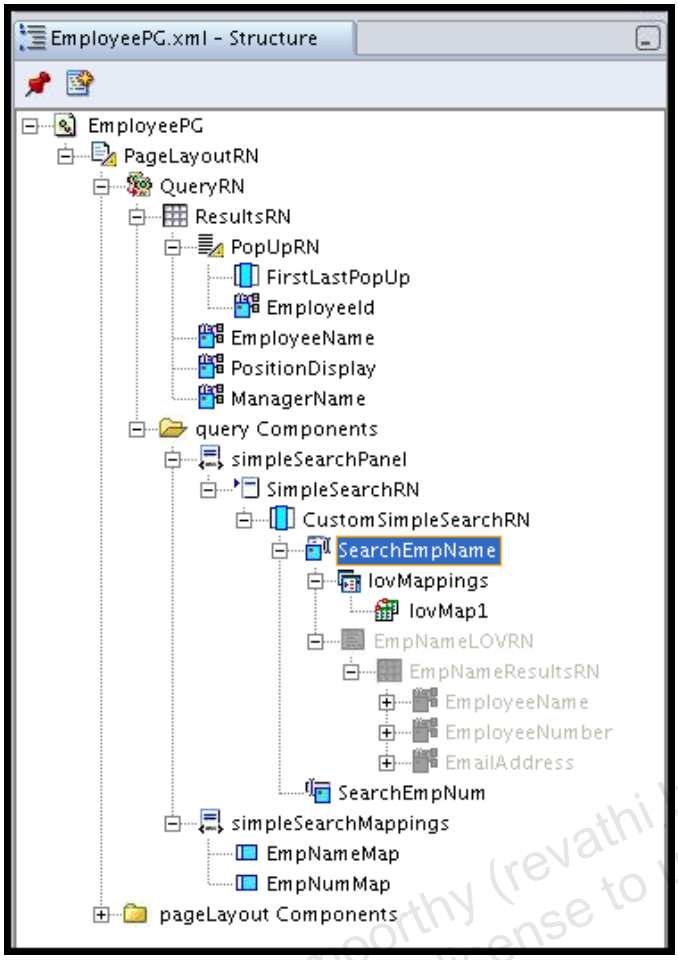


3. Change the **Item Style** to **messageLovInput**.
4. JDeveloper automatically creates an inline LOV region (Region Style listOfValues) and a default LOV mapping. You are not going to use the defaults. You are going to use your newly created LOV.



5. Click the **SearchEmpName** in the structure, and set the **External LOV** in the property inspector to **`<student#>/oracle/apps/ak/lov/webui/EmpNameLOVRN`**.
6. When you set the External LOV property, a confirmation message appears. The message states, “This action will delete the existing inline LOV region for this item. Do you wish to continue?” Click the **Yes** button.





Note: The EmpNameLOVRN is now added to the EmployeePG, but is grayed out. The LOV is an External LOV, and is not contained within the EmployeePG. To edit the EmpNameLOVRN, you have to change the master version of it that is external to the EmployeePG. You can edit the mappings, because they are part of the page.

Step 7: Define Your LOV Mappings in the EmpNameLOVRN

Create mappings between base page items and LOV items. The mappings identify the data input/output relationships between base page items and LOV items.

1. In the structure pane (you are still in the EmployeePG page), click the default LOV mapping with the ID of **lovMap1**, and set the following properties:

   Note: To find the lovMap1 item, expand the SearchEmpName item, and then expand the lovMappings item under SearchEmpName. lovMap1 will be found there.

   LOV Region Item = **EmployeeName**

   Return Item = **SearchEmpName**

   Criteria Item = **SearchEmpName**

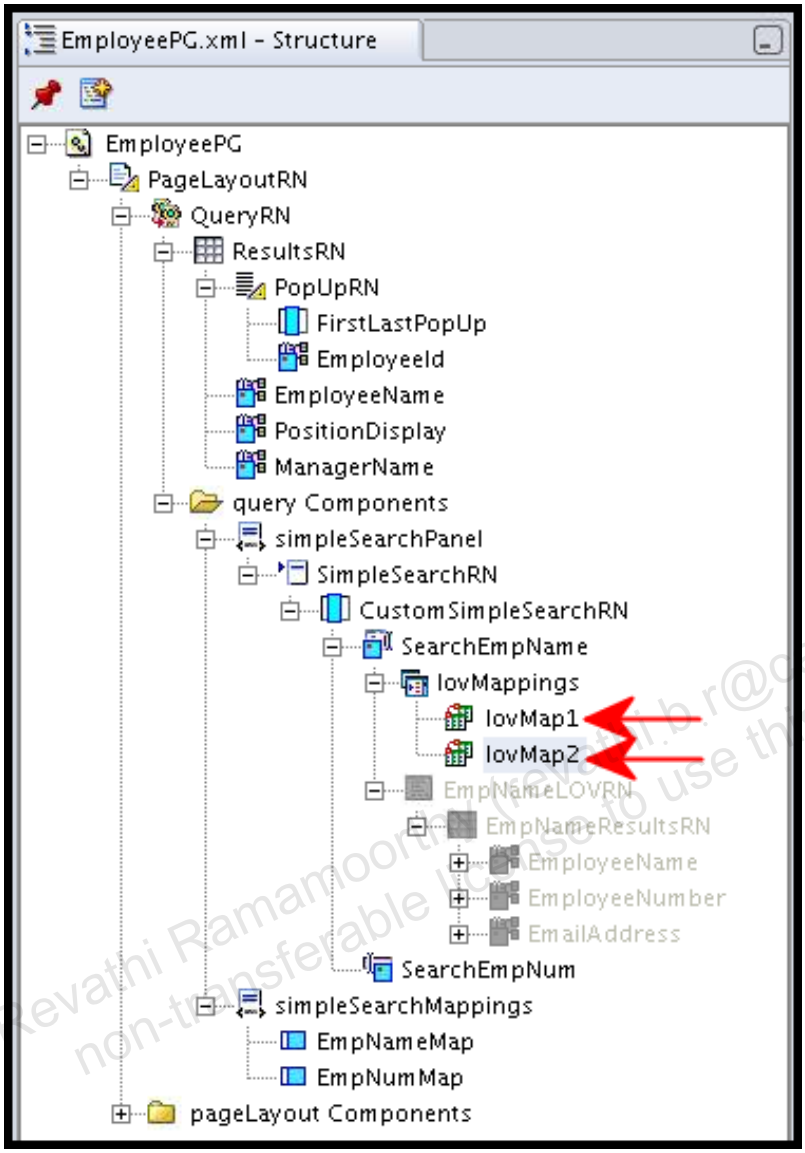
   Disable Validation = **True** (only appears in the Windows version of JDeveloper)

   Note: A Return Item is the attribute returned by the LOV. A Criteria Item is the attribute passed to the LOV, and is the basis for the LOV's query (select).
2. Right-click **lovMappings**, and select **New** > **lovMap** from the context menu.
3. Click the default LOV mapping, **lovMap2**, and set the following properties:

   LOV Region Item = **EmployeeNumber**

Return Item = **SearchEmpNum**

Criteria Item = **SearchEmpNum**

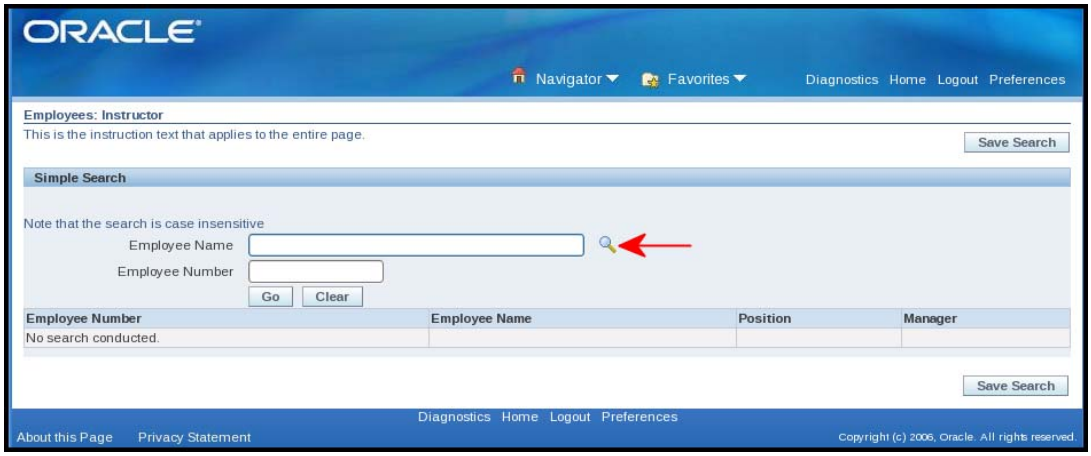


Step 8: Test Your Work

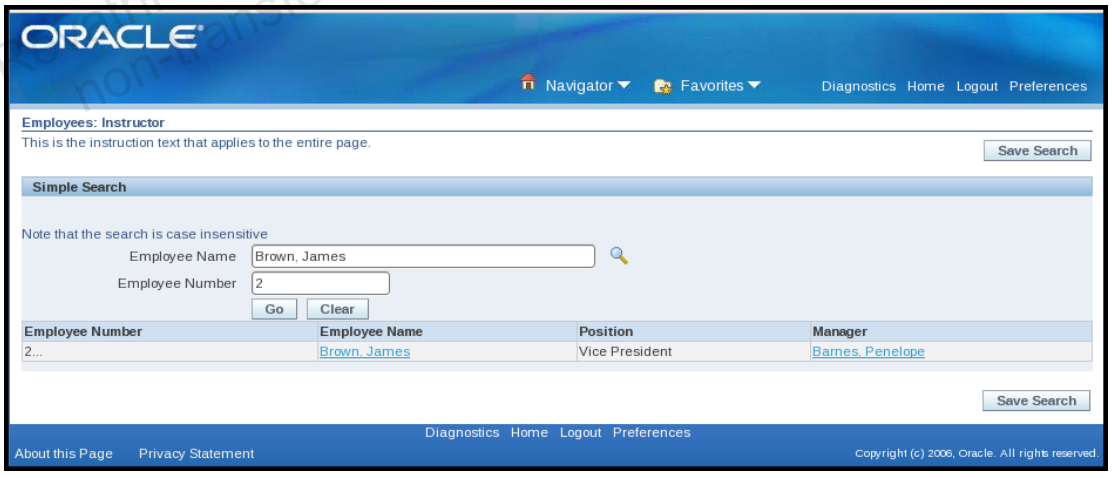
1. Save your work
2. Run your page. Your results should look as follows with the LOV icon (Magnifying Glass)

Note: Turn off the PopUp blocker if your browser has one. You must turn it off. Setting it to **Allow PopUps from this site** might cause Null Pointer Exceptions on your page.

Note: Remember that the Query Statement that is part of the underlying EmpNameLOVVO has `END_DATE IS NULL`.



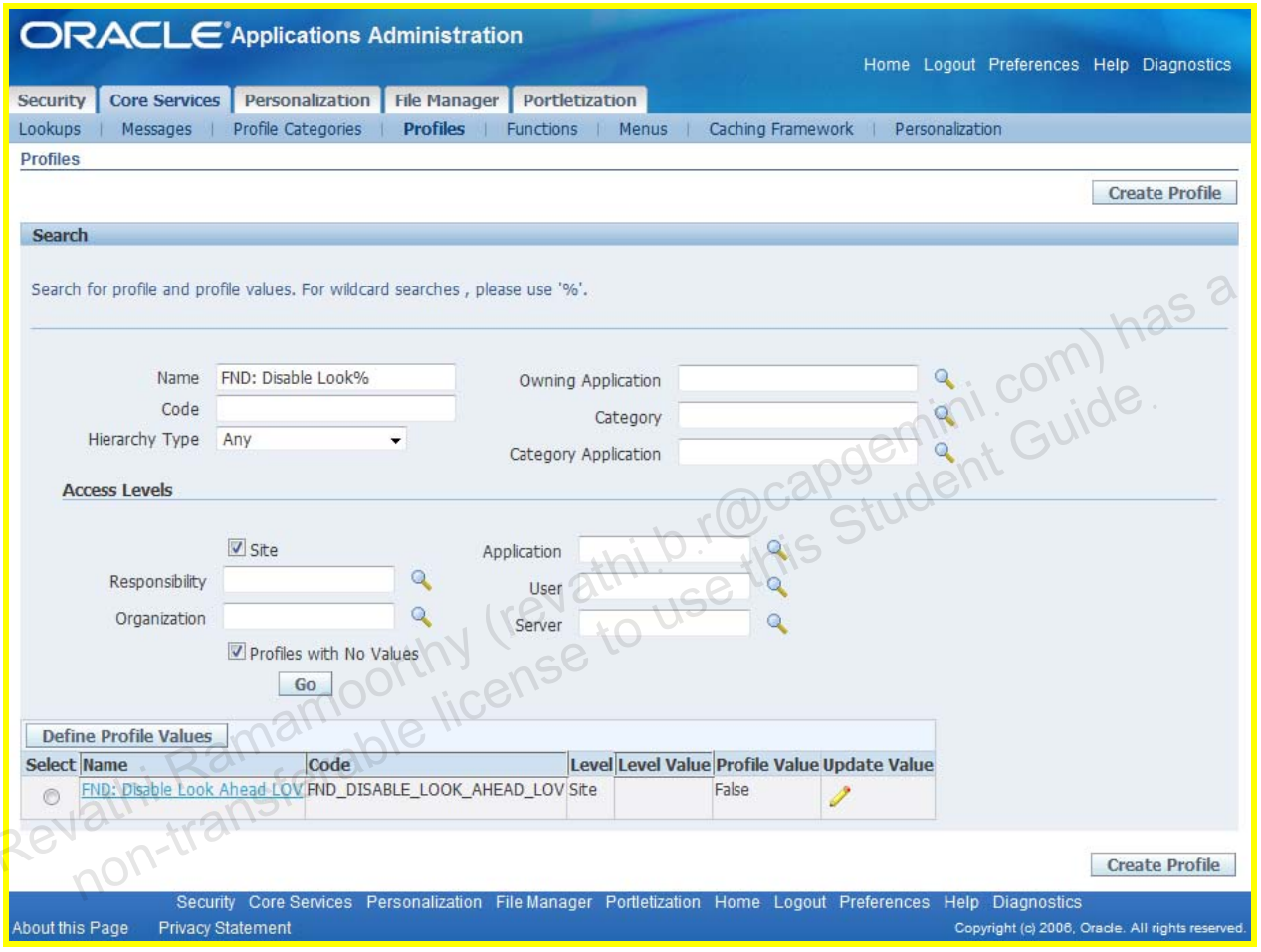
When you click on the magnifying glass the below page appears.



Step 9: Enable the Look Ahead LOV

Part of Oracle E-Business Suite 12.1.2 / OA Framework 12.1.2 is a new feature that has been created to replace the LOV window. The LOV window is still functional. A popup with the LOV choices can be enabled via an E-Business Suite system profile option, and by setting additional properties on the LOV item.

1. Log into your E-Business Suite instance via SYSADMIN/SYSADMIN.
2. In the Application's Navigator, click the **Functional Administrator** responsibility.
3. Click the **Core Services** tab.
4. Click the **Profiles** subtab.
5. In the Search, enter the **Name** as **FND: Disable Look%**.
6. Click the **Go** button.



7. Click the **Update Value** (**pencil**) icon.
8. Verify that **FND: Disable Look Ahead LOV** is set to **False**.
   Note: It should already be set as False in an OU course instance.
9. Change the **Name** to **FND: Minimum Char%**.
10. Click the **Go** button.
11. Click the **Update Value** (**pencil**) icon.
12. Verify that **FND: Minimum Characters for Look Ahead** is set to a value of to 1.
    Note: The default is 3. You do not have a large dataset as part of the FWK_TBX_EMPLOYEES table.
13. You can logout of the E-Business Suite instance.
14. Return to JDeveloper.
15. Click the **EmployeePG** page.

16. Drill-down to the **SearchEmpName** as part of the query Components of that page, and set the following properties:

    Look Ahead Enabled = **True**

    Look Ahead Search Type = **startsWith**

    Look Ahead Records Displayed = **5**

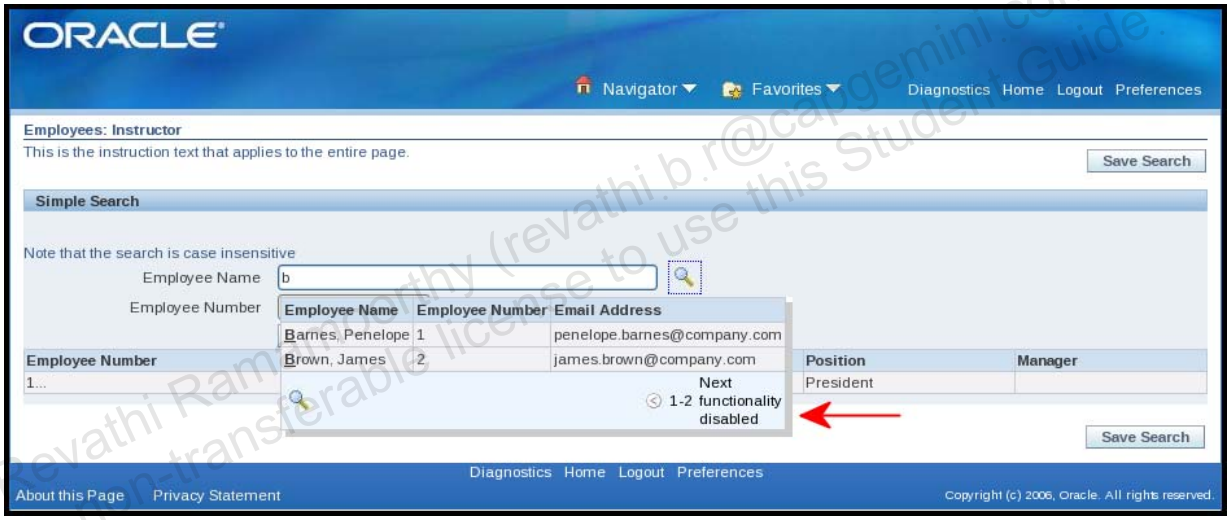
    Look Ahead Selection Event Enabled = **True**

    Minimum Characters For Look Ahead = **1**

    Note: It is blank in the properties, and can be set to a value greater than the profile option to override it. You are only using **1** for your look ahead because there are so few names in the FWK_TBX_EMPOYEES table.

    Additional Text = **Employee Name**

17. Save your work.
18. Test your page with the LOV look ahead enabled.
19. As you start to type, the record search activates as you type in real-time. A window with the results of the criteria that you defined as part of the setup for the EmpNameLOVVO appears.



    Note: Clicking on the magnifying glass will activate the separate LOV window. That functionality continues to work. Once the LOV record is selected, the earlier PopUp functionality will work.

20. In the image above, you can see **Next Functionality disabled** with a single navigation icon displayed. There are only two records (people) in the base dataset that start with B. The Look Ahead LOV is aware that there are no next records, and disables the navigation functionality appropriately.

Task 5: Create Your Drill Down Page

Step 1: Create Your Application Module (AM)

By standards, an AM represents a transaction. Since the query is a transaction, and drilling down to employee details is another transaction, you should adhere to standards and good programming practice by creating another AM.

1. Right-click your **ClassProject** project, and click **New** from the context menu.
2. Click the **Business Tier** > **ADF Business Components** > **Application Module** from the New Gallery to open the Create Application Module wizard.
3. Set the **Package** to `<student#>.oracle.apps.ak.employee.server`.
4. Set the **Name** to **EmpDetailsAM**.
5. Click the **Next** button.
6. You can accept the remaining defaults. Click the **Finish** button.

   Note: One of the defaults when you create an AM is to generate the associated java files. EmpDetailsAMImpl.java was created when you accepted the defaults. You will be adding code to this java class at a later point in the lab.
7. Open the **<student#>.oracle.apps.ak.employee.server** package in the Applications Navigator.
8. Double-click **EmpDetailsAM** to open the Application Module Editor.
9. Click **Custom Properties** from the categories panel.
10. Set the **Name** to **RETENTION_LEVEL**.
11. Set the **Value** to **MANAGE_STATE**.
12. Click the **Add** button.
13. Click the **Apply**.
14. Click the **OK** button.
15. Save your work.

Step 2: Create Your EmployeeDetailsVO View Object (VO)

The EmployeeDetailsVO will be a detail-level view object including all the attributes that you need for the Employees detail page that you will create.

1. Right-click your **ClassProject** project, and click **New** from the context menu.
2. Click the **Business Tier** > **ADF Business Components** > **View Object** from the New Gallery to open the View Object wizard. If the Welcome screen appears, click the **Next** button.
3. Set the **Package** to `<student#>.oracle.apps.ak.employee.server`.
4. Set the **Name** to **EmployeeDetailsVO**.
5. Select the **Updateable Access through Entity Objects** option.
6. Click the **Next** button.
7. Open the **<student#>.oracle.apps.ak.schema.server** package in the Available pane, and click the **EmployeeEO**. Use the **[>]** button to shuttle **EmployeeEO** to the Selected pane.
8. Use the **[>]** button and shuttle a **second** copy of **EmployeeEO** to the Selected pane.
9. Verify that the **Association** field for the second copy is set to **EmpToMgrAO.ManagerIdEmployeesEO**.

10. Verify that the **Alias** is set to **EmployeesEO1**.
11. Check the **Updatable** checkbox for FwkTbxEmployeesEO1.
12. Check the **Reference** checkbox for FwkTbxEmployeesEO1, if not already checked.
13. Click the **Next** button.
14. In the Attributes page, click the following attributes from the Available list and shuttle them to the Selected list in the following order:

    From **EmployeesEO**:

    EmployeeId
    FirstName
    LastName
    FullName
    EmailAddress
    ManagerId
    PositionCode
    Salary
    StartDate
    EndDate

    From **EmployeesEO1**:

    EmployeeId (JDeveloper assigns this attribute the default name of EmployeeId1)
    FullName (JDeveloper assigns this attribute the default name of FullName1)
    EmailAddress (JDeveloper assigns this attribute the default name of EmailAddress1)
15. Click the **Next** button.
16. On Step 4 of 7: The Attributes Setting page, click the **Select Attribute** field drop-down. Click **FullName**, and set the following:

    Attribute Name = **EmployeeName**

    Query Column Alias = **EMPLOYEE_NAME**
17. Click the Select Attribute field drop-down. Click **EmailAddress**, and set the following:

    Attribute Name = **EmployeeEmail**

    Query Column Alias = **EMPLOYEE_EMAIL**
18. Click the Select Attribute field drop-down. Click **FullName1**, and set the following:

    Attribute Name = **ManagerName**

    Query Column Alias = **MANAGER_NAME**

    Select the **Updatable: Always** option.
19. Click the Select Attribute field drop-down. Click **EmailAddress1**, and set the following:

    Attribute Name = ManagerEmail

    Query Column Alias = MANAGER_EMAIL

    Select the **Updatable: Always** option.
20. Select the **Next** button.
21. Check the **Expert Mode** checkbox. This will allow you to edit the generated query. In the **Query Statement** text box modify the query so it looks like the following:

```
SELECT EmployeeEO.EMPLOYEE_ID,
       EmployeeEO.FIRST_NAME,
```

```
        EmployeeEO.LAST_NAME,
        EmployeeEO.FULL_NAME AS EMPLOYEE_NAME,
        EmployeeEO.EMAIL_ADDRESS AS EMPLOYEE_EMAIL,
        EmployeeEO.MANAGER_ID,
        EmployeeEO.POSITION_CODE,
        EmployeeEO.SALARY,
        EmployeeEO.START_DATE,
        EmployeeEO.END_DATE,
        EmployeeEO1.EMPLOYEE_ID AS EMPLOYEE_ID1,
        EmployeeEO1.FULL_NAME AS MANAGER_NAME,
        EmployeeEO1.EMAIL_ADDRESS AS MANAGER_EMAIL,
        FwkLookupCode.MEANING AS POSITION_DISPLAY
FROM    FWK_TBX_EMPLOYEES EmployeeEO,
        FWK_TBX_EMPLOYEES EmployeeEO1,
        FWK_TBX_LOOKUP_CODES_VL FwkLookupCode
WHERE   EmployeeEO.POSITION_CODE = FwkLookupCode.LOOKUP_CODE
AND FwkLookupCode.LOOKUP_TYPE = 'FWK_TBX_POSITIONS'
```

Note: The differences between the generated query and this query are:

- `FWK_TBX_LOOKUP_CODES_VL FwkLookupCode` was added to the `FROM` clause.
- `EmployeeEO.EMPLOYEE_ID = FwkTbxEmployeesEO1.MANAGER_ID` was removed from the `WHERE` clause.
- `EmployeeEO.MANAGER_ID = FwkTbxEmployeesEO1.EMPLOYEE_ID (+)` was added to the `WHERE` clause.
- `AND EmployeeEO.POSITION_CODE = FwkLookupCode.LOOKUP_CODE` was added to the `WHERE` clause.
- `AND FwkLookupCode.LOOKUP_TYPE = 'FWK_TBX_POSITIONS'` was added to the `WHERE` clause.

22. When you are finished with your editing, select the **Test** button to ensure your syntax is correct.
23. Click the **Next** button.
24. Check and verify the **Attribute Mappings** to ensure that all of the Query Columns match up to the proper View Attributes. This is a critical check when you create a VO in Expert Mode.
25. Click the **Next** button.
26. Click the **Next** button on Step 7 of 8.
27. Check the **View Object Class: EmployeeDetailsVOImpl: Generate Java File** checkbox.
28. Check the **View Row Class: EmployeeDetailsVORowImpl: Generate Java File** and **Accessors** checkboxes.
29. Click the **Finish** button.
30. Save your work.

Step 3: Add Your View Object to Your Application Module

1. Double-click **EmpDetailsAM** to open it in the Application Module Editor.
2. Click **Data Model** from the categories pane.
3. Shuttle **EmployeeDetailsVO** from the Available View Objects to the Data Model pane.
4. Click the **OK** button.
5. Save your work.

Task 6: Create Your View-layer Components

Step 1: Create Your EmpDetailsPG Page

1. Right-click your **ClassProject** project, and click **New** from the context menu.
2. Click the **Web Tier** > **OA Components** > **Page** from the New Gallery.
3. Set the **Name** to **EmpDetailsPG**.
4. Set the **Package** to `<student#>. oracle.apps.ak.employee.webui`.
   Note: You may have to browse for the correct package.
5. Click the **OK** button.

Step 2: Modify the pageLayout Region

1. Click the **EmpDetailsPG** in the Applications Navigator panel.
2. In the Structure panel, click the **region1** item.
3. In the property inspector, change the following properties:
   ID = **PageLayoutRN**
   AM Definition = `<student#>.oracle.apps.ak.employee.server.EmpDetailsAM`
   Windows Title = **<Your Name> Drill-down Details**
   Title = **<Your Name> Employees Details:**

Step 3: Add a Product Branding Image

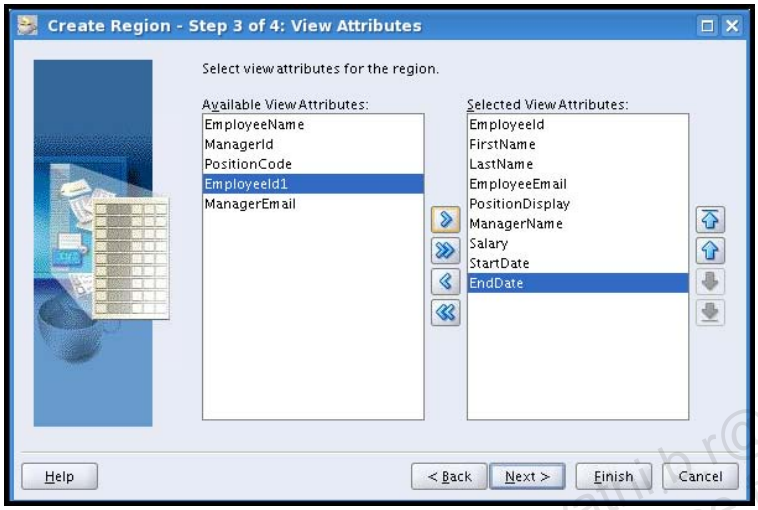
Each Oracle Applications page requires a product branding image.

1. Select your **PageLayoutRN** in the Structure panel.
2. Right-click it, and click **New** > **productBranding** from the context menu.
3. JDeveloper creates a pageLayoutComponents folder containing a productBranding image item (named **item1**). Click **item1**, and set the following properties:
   ID = **ProdBrand**
   Image URI = `FNDTAPPBRAND.gif`
   Additional Text = **Employee Details Page**

Step 4: Create Your Main Content Region

1. Click your **PageLayoutRN**.
2. Right-click it, and click **New** > **Region Using Wizard**. If the Welcome page appears, click the **Next** button.
3. Set the **Application Module** drop-down to your **EmpDetailsAM**.
4. Do not check the **Use this as Application Module Definition for the region** checkbox.
5. Click **EmployeeDetailsVO1** from the Available View Usages.
6. Click the **Next** button.
7. Set the **Region ID** to **MainRN**.
8. Set the **Region Style** to **defaultSingleColumn**.
9. Click the **Next** button.
10. Shuttle the following attributes in the listed order to the Selected View Attributes pane.
    EmployeeId

FirstName
LastName
EmployeeEmail
PositionDisplay
ManagerName
Salary
StartDate
EndDate



11. Click the **Next** button.
12. In the **Region Items** table, set the **Attribute Set** column of the items as follows:

    EmployeeId = **`/oracle/apps/fnd/framework/toolbox/attributesets/FwkTbxEmployees/EmployeeId_Number`**

    FirstName = **`/oracle/apps/fnd/framework/toolbox/attributesets/FwkTbxEmployees/FirstName`**

    LastName = **`/oracle/apps/fnd/framework/toolbox/attributesets/FwkTbxEmployees/LastName`**

    EmployeeEmail = **`/oracle/apps/fnd/framework/toolbox/attributesets/FwkTbxEmployees/EmailAddress`**

    PositionDisplay = **`/oracle/apps/fnd/framework/toolbox/attributesets/FwkTbxEmployees/Position`**

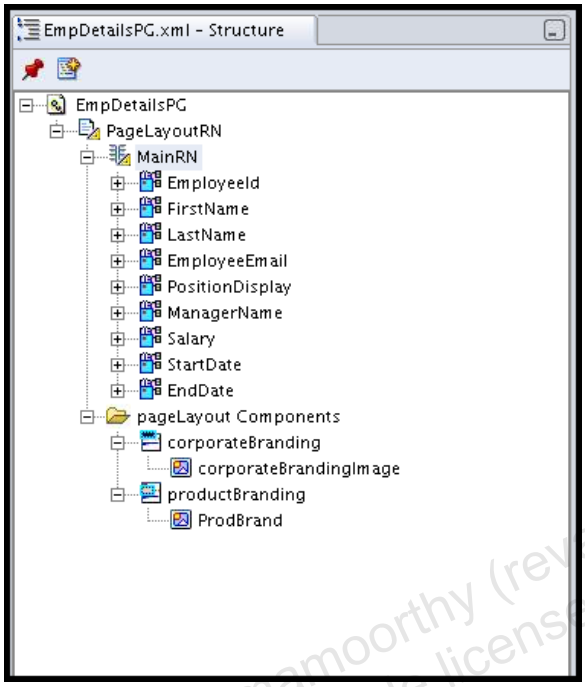
    ManagerName = **`/oracle/apps/fnd/framework/toolbox/attributesets/FwkTbxEmployees/FullName_Manager`**

    Salary = **`/oracle/apps/fnd/framework/toolbox/attributesets/FwkTbxEmployees/Salary`**

StartDate =
`/oracle/apps/fnd/framework/toolbox/attributesets/FwkTbxEmployees/StartDate`

EndDate =
`/oracle/apps/fnd/framework/toolbox/attributesets/FwkTbxEmployees/EndDate`

13. Set each and every item in the **Region Items** table to **Style** = **messageStyledText**.
14. Click the **Finish** button.



15. Click the **MainRN** region in the structure panel, and set the following properties:
    Text = **Employees Details**
    Hide Header = **True**

Step 5: Set Your EmpDetails Page's Items Properties

Expand the MainRN, and set the item properties as indicated.

1. For the **EmployeeId**, **FirstName**, **LastName**, **PositionDisplay**, **Salary**, **StartDate**, and **EndDate** items, set the following property:
   CSS Class = **OraDataText**
2. For the **EmployeeEmail** item, set the following properties:
   Destination URI = **mailto:{@EmployeeEmail}**
   CSS Class = **OraLinkText**
3. For the ManagerName item, set the following properties:
   Destination URI = **mailto:{@ManagerEmail}**
   CSS Class = **OraLinkText**

Step 6: Modify the MainRN's Style

Select the MainRN, and set the item properties as indicated.

1. Change the **Region Style** to **messageComponentLayout**.

Note: The reason to change this is to allow UIX to render the page appropriately. There are many default components in OAF, and changing the style affects how and what children can be built in the item's hierarchy.

2. The dialog window will appear with the message, "This component's child(ren) could potentially be removed by this action. Do you wish to continue?" Click the **Yes** button.

   Note: Nothing will change except for the style. Do not worry about this message in this instance.

Step 7: Add a Return Link

1. Right-click your **PageLayoutRN**, and click **New** > **returnNavigation**.
2. Click the **returnNavigationLink** item that was just created, and set the properties as follows:

   ID = **ReturnLink**

   Destination URI = **`OA.jsp?page=/<student#>/oracle/apps/ak/employee/webui/EmployeePG`**

   Text = **Return to the Employee Page**
3. Save your work.

Task 7: Write Your Model-layer Programmatic Elements

Step 1: Add an initQuery() Method to the EmployeeDetailsVO

1. Click on your **EmployeeDetailsVO** in the Applications Navigator panel.
2. In the Structure panel, double-click on the **`EmployeeDetailsVOImpl.java`** file in the Sources folder. This will open the Java editor.
   Note: There is a skeleton with an empty public method.
3. In your import statements, add the following:
```
import oracle.jbo.domain.Number;
import oracle.apps.fnd.framework.OAException;
```
4. In your class definition, add a new method, called initQuery(), as follows:
```
public void initQuery(String employeeNumber) {
  if ((employeeNumber != null) &&
     (!("".equals(employeeNumber.trim())))) {
    // Do the following conversion for type consistency.
    Number empNum = null;
    try {
          empNum = new Number(employeeNumber);
        } catch (Exception e) {
          throw new OAException
            ("AK","FWK_TBX_INVALID_EMP_NUMBER");
        }
    setWhereClause("EMPLOYEE_ID = :1");
    setWhereClauseParams(null); // Always reset
    setWhereClauseParam(0, empNum);
    executeQuery();
  }
} // end initQuery()
```
5. Save your work.
6. Close the EmployeeDetailsVOImpl.java.

Step 2: Add an initDetails() Method to Your EmpDetailsAM

1. Click on your **EmpDetailsAM** in the Applications Navigator panel.
2. In the Structure panel, double-click on the **`EmpDetailsAMImpl.java`** file in the Sources folder. This will open the Java editor.
3. In your import statements, add the following:
```
import oracle.apps.fnd.framework.OAException;
import oracle.apps.fnd.common.MessageToken;
```
4. In your class definition, add a new method, called initDetails(), as follows:
```
public void initDetails(String employeeNumber) {
  EmployeeDetailsVOImpl vo = getEmployeeDetailsVO1();
  if (vo == null) {
    MessageToken[] errTokens =
```

```
        { new MessageToken("OBJECT_NAME", "EmployeeDetailsVO1") };
          throw new OAException
            ("AK", "FWK_TBX_OBJECT_NOT_FOUND", errTokens);
        }
        vo.initQuery(employeeNumber);
      } // end initDetails()
```

5. Save your work.
6. Close EmpDetailsAMImpl.java.

Task 8: Modify Your Existing View-layer Components

In addition to creating new View-layer components, which you did in Task 6, you also need to modify the existing EmployeePG to implement the drill-down.

Step 1: Modify Your EmployeeName Item as a Link

1. Click your **EmployeePG** in the Applications Navigator.
2. In the **ResultsRN**, click **EmployeeName** and set the following:

   Destination URI =

   ```
   OA.jsp?page=/<student#>/oracle/apps/ak/employee/webui/EmpDetailsPG
   &employeeNumber={@EmployeeId}
   &employeeName={@EmployeeName}
   &retainAM=N&addBreadCrumb=Y
   ```

3. Save your work.

Task 9: Write Your Controller-layer Programmatic Elements

For the first time, you need to create a controller on a component. Your controller is going to do two things. One, it is going to initialize the EmployeeDetailsVO Query by calling the EmpDetailsAM's initDetails() method. Two, it is going to programmatically change the page title text to something that is context sensitive.

Step 1: Create Your Controller

1. Right-click your **PageLayoutRN** of your EmpDetailsPG, and click **Set New Controller…**.
2. Set the **Package Name** to `<student#>.oracle.apps.ak.employee.webui`.

   Note: The default name that populates the Package Name field repeats webui twice, and needs to be changed.
3. Set the **Class Name** to **EmpDetailsCO**.
4. Click the **OK** button. The EmpDetailsCO.java code immediately opens in JDeveloper.

Step 2 Add Controller Logic to Initialize the EmployeeDetailsVO.

1. In your import statements of your EmpDetailsCO.java, add the following:

```
import java.io.Serializable;
import oracle.apps.fnd.framework.OAApplicationModule;
```

2. In your class definition, add code to the existing processRequest() method as follows:

   Note: In the example below, some of the lines of code will already be created.

```
public void processRequest(OAPageContext pageContext, OAWebBean
webBean) {
  // Always call this first.
  super.processRequest(pageContext, webBean);
  // Get the employeeNumber parameter from the URL
  String employeeNumber =
pageContext.getParameter("employeeNumber");
  // Now initialize the query for our single employee
  // with all of its details.
  OAApplicationModule am =
pageContext.getApplicationModule(webBean);
  Serializable[] parameters = { employeeNumber };
  am.invokeMethod("initDetails", parameters);
}
```

3. Save your work.

Step 3: Add Controller Logic to Set Page Title Text.

1. In your import statements of your EmpDetailsCO.java, add the following:

```
import oracle.apps.fnd.common.MessageToken;
import
oracle.apps.fnd.framework.webui.beans.layout.OAPageLayoutBean;
```

   Note: Per the Oracle BLAF UI Guidelines on Header Components, you need to display the current selected employee name in the page title. To do this, add the following code to your controller's processRequest() method.

   Note: This assumes a message (FWK_TBX_T_EMP_HEADER_TEXT) has already been created in the database with the following content: **Employee: &EMP_NAME**.

2. In your class definition, add code to the existing **processRequest()** method, directly following the code you added in Step 2, as follows:

```
// Always use a translated value from Message Dictionary when
setting
// strings in your controllers.
// Instantiate an array of message tokens and set the value for the
// EMP_NAME token.
String employeeName = pageContext.getParameter("employeeName");
MessageToken[] tokens = { new MessageToken("EMP_NAME", employeeName)
};
// Now, get the translated message text including the token value.
String pageHeaderText =
pageContext.getMessage("AK", "FWK_TBX_T_EMP_HEADER_TEXT", tokens);
// Set the employee-specific page title (which also appears in
// the breadcrumbs).  Note that we know this controller is
// associated with the pageLayout region, which is why we cast the
// webBean to an OAPageLayoutBean before calling setTitle.
((OAPageLayoutBean)webBean).setTitle(pageHeaderText);
```

3. Save your work.
4. Close EmpDetailsCO.java.

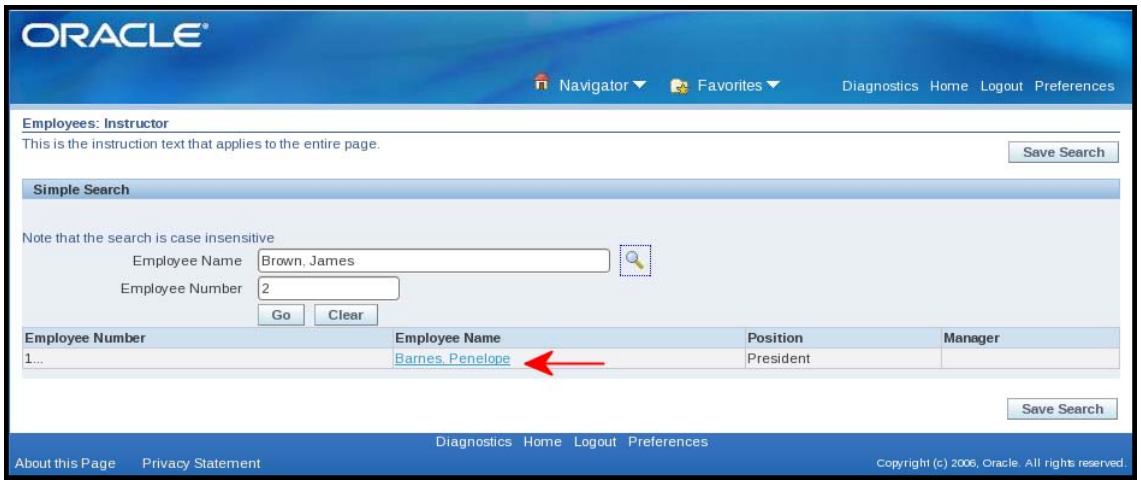
Step 4: Compile Your Project

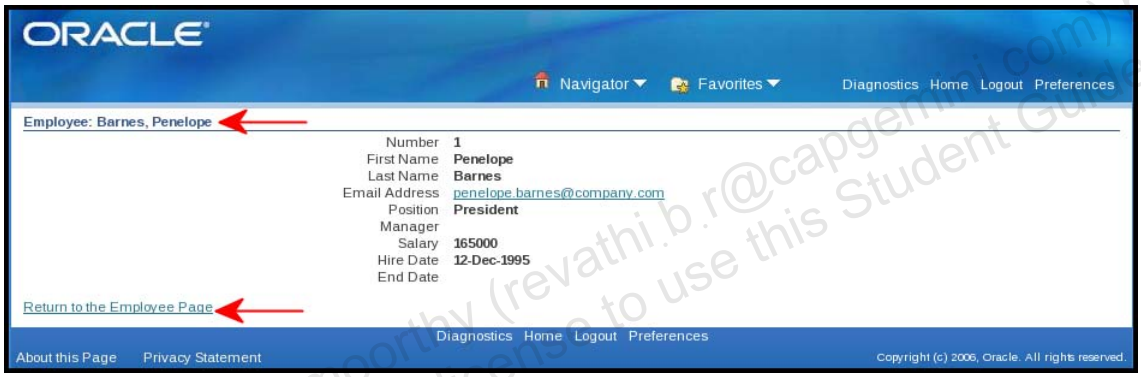
1. Right-click your **ClassProject** project in the Applications Navigator panel, and click **Rebuild**.
2. You can ignore any warnings. You must correct any errors.

Step 5: Test Your Work.

1. Save your work.
2. Run your **EmployeePG**.

   Note: Your EmployeePG allows you to drill-down to details. You cannot run the EmpDetailsPG directly. To test your EmpDetailsPG, you will need to click on the link provided in the EmployeePG results table.





Note: There are some caveats if the employeeName does not render on the Header. Make sure your bind parameters for the EmployeePG are set correctly on the Destination URI. Make sure the following...

```
OA.jsp?page=<student#>/oracle/apps/ak/employee/webui/EmpDetail
sPG&employeeNumber={@EmployeeId}&employeeName={@EmployeeName}&
retainAM=N&addBreadCrumb=Y
```

Note: The above code matches the code in the controller in the **EmpDetailsCO**.

Lab - Implementing a Create

Chapter 5

Part 1: Basic Create Overview

For this lab, you will implement the basic components of an Employee Create page. This lab assumes, and relies upon, you having completed both the Lab: First OA Framework Page and Lab: Implementing a Query Page and Drill-down Page. If you have not yet finished those labs, please finish them before working on this lab.

After completing this lab, you will have:

- Added a global table button.
- Added an **Indicates Required Field** region to a page, and designate selected fields as being required.
- Created a poplist.
- Added page-level action/navigation buttons.
- Coded a programmatic JSP Forward from one page to another.
- Enabled **Warn About Changes** checking to help the user avoid inadvertently losing work.
- Displayed a confirmation dialog message.
- Implemented typical entity object and entity expert attribute and row-level business logic, including throwing attribute and row-level validation exceptions.
- Created a new row in an entity object based view object.
- Handled an Apply button press, which implements the commit.

Task 1: Build Your Model-layer Components for the Create Page

Step 1: Create Your EmpCreateAM Application Module (AM)

By standards, an AM represents a transaction. Since the query employees is a transaction, and drilling down to employee details is another transaction, you should adhere to standards and good programming practice by creating yet another AM for creating employees.

1. Right-click your **ClassProject** project, and click **New** from the context menu.
2. Click the **Business Tier** > **ADF Business Components** > **Application Module** from the New Gallery to open the Create Application Module wizard.
3. Set the **Package** to `<student#>.oracle.apps.ak.employee.server`.
4. Set the **Name** to **EmpCreateAM**.
5. Click the **Next** button.
6. You can accept the remaining defaults. Click the **Finish** button.
7. Open the **<student#>.oracle.apps.ak.employee.server** package in the Applications Navigator.
8. Double-click **EmpCreateAM** to open the Application Module Editor.
9. Click **Custom Properties** from the categories pane.
10. Set the **Name** to **RETENTION_LEVEL**.
11. Set the **Value** to **MANAGE_STATE**.
12. Click the **Add** button.
13. Click **Data Model** from the categories pane.
14. Shuttle **EmployeeDetailsVO** from the Available View Objects to the Data Model pane.

    Note: It is perfectly acceptable to reuse a VO, if and when it makes sense to do so. But, your Application Modules should still be restricted to a single transaction. In this case, the transaction is create/insert.
15. Click the **Apply**. Click the **OK** button.
16. Save your work.

Task 2: Build Your View-layer Components

Step 1: Create Your EmpCreatePG Page

1. Right-click your **ClassProject** project, and click **New** from the context menu.
2. Click the **Web Tier** > **OA Components** > **Page** from the New Gallery.
3. Set the **Name** to **EmpCreatePG**.
4. Set the **Package** to `<student#>.oracle.apps.ak.employee.webui`.
5. Click the **OK** button.

Step 2: Modify the pageLayout Region

1. Click the **EmpCreatePG** in the Applications Navigator panel.
2. In the Structure panel, click the **region1** item. In the property inspector, change the following properties:

   ID = **PageLayoutRN**

   AM Definition = `<student#>.oracle.apps.ak.employee.server.EmpCreateAM`

   Warn About Changes = **True**

   Windows Title = **<Your Name> Create**

   Title = **Create Employee: <Your Name>**

   Note: Setting the `Warn About Changes` property to True ensures that users are warned when they try to navigate from this data entry page with pending changes (the page's Cancel button is an exception, as you will see later in this lab).

Step 3: Add a Product Branding Image

Each Oracle Applications page requires a product branding image.

1. Click your **PageLayoutRN** in the Structure panel.
2. Right-click it, and click **New** > **productBranding** from the context menu.
3. JDeveloper creates a pageLayoutComponents folder containing a productBranding image item (named **item1**). Click **item1**, and set the following properties:

   ID = **ProdBrand**

   Image URI = `FNDTAPPBRAND.gif`

   Additional Text = **Employee Create Page**

   Note: Additional Text is added to provide accessibility to the application, and will be set by standards on all Oracle E-Business Suite pages.

Step 4: Add Your Buttons

Per the BLAF UI guidelines, all page-level buttons render twice on the page: below the page title, and below the menu. However, you only need to add them once using a special pageButtonBar region.

1. Click your **PageLayoutRN** in the **Structure** panel.
2. Right-click it, and click **New** > **Region** from the context menu. Set the following properties for this region:

   ID = **PageButtons**

   Region Style = **pageButtonBar**

3. Right-click **PageButtons** in the Structure panel, and click **New** > **Item** from the context menu. Set the following properties for this item:

   ID = **Cancel**

   Item Style = **submitButton**

   Attribute Set = **`/oracle/apps/fnd/attributesets/Buttons/Cancel`**

   Note: E-Business Suite provides attribute sets for common objects, like Apply and Cancel buttons. You promote good programming practice, standards compliance, and ease of translation, it is recommended that you use this attribute sets.

   Disable Server Side Validation = **True**

   Disable Client Side Validation = **True**

   Note: You set both of these properties to True to allow the user to leave the page by clicking the button without encountering a validation error.
4. Right-click **PageButtons** in the Structure panel, and click **New** > **Item** from the context menu. Set the following properties for this item:

   ID = **Apply**

   Item Style = **submitButton**

   Attribute Set = **`/oracle/apps/fnd/attributesets/Buttons/Apply`**

Step 5: Create the Main Content Region

To achieve the correct indentation of our display fields, you are going to add a defaultSingleColumn region to your PageLayoutRN on the EmpCreatePG. Since each of the items in this region will bind to the EmployeeDetailsVO1 view instance, you will use the region wizard to quickly create this.

1. In the EmpCreatePG, click the **PageLayoutRN** in the Structure panel. Right-click it, and click **New** > **Region Using Wizard** from the context menu.
2. Set the **Application Module** drop-down to **`<student#>.oracle.apps.ak.employee.server.EmpCreateAM`**.
3. Do not check the **Use this as Application Module Definition for the region** checkbox.
4. Click the **EmployeeDetailsVO1** from **Available View Usages**.
5. Click the **Next** button.
6. Set the **Region ID** to **MainRN**.
7. Set the **Region Style** to **defaultSingleColumn**.
8. Click the **Next** button.
9. Shuttle the attributes to the Selected View Attributes in the order as follows:

   EmployeeId
   FirstName
   LastName
   EmployeeEmail
   PositionCode
   ManagerName
   Salary
   StartDate
   EndDate
   ManagerId

10. Click the **Next** button.
11. In the Region Items table, set the **Attribute Set** property of the items as follows:
    EmployeeId =
    /oracle/apps/fnd/framework/toolbox/attributesets/FwkTbxEmployees/EmployeeId_Number
    FirstName =
    /oracle/apps/fnd/framework/toolbox/attributesets/FwkTbxEmployees/FirstName
    LastName =
    /oracle/apps/fnd/framework/toolbox/attributesets/FwkTbxEmployees/LastName
    EmployeeEmail =
    /oracle/apps/fnd/framework/toolbox/attributesets/FwkTbxEmployees/EmailAddress
    PositionCode =
    /oracle/apps/fnd/framework/toolbox/attributesets/FwkTbxEmployees/Position
    ManagerName =
    /oracle/apps/fnd/framework/toolbox/attributesets/FwkTbxEmployees/FullName_Manager
    Salary =
    /oracle/apps/fnd/framework/toolbox/attributesets/FwkTbxEmployees/Salary
    StartDate =
    /oracle/apps/fnd/framework/toolbox/attributesets/FwkTbxEmployees/StartDate
    EndDate =
    /oracle/apps/fnd/framework/toolbox/attributesets/FwkTbxEmployees/EndDate
12. Set the **Style** for ManagerName to **messageLovInput**.
13. Click the **Finish** button.
    Note: You did not set any attributes for the ManagerId.

Step 6: Set Your MainRN's Items Properties

1. For the **EmployeeId** item, set the following properties:
   CSS Class = **OraDataText**
2. For the **FirstName** and **LastName** items, set the following properties:
   Required = **yes**
   CSS Class = **OraFieldText**
3. For the EmployeeEmail item, set the following properties:
   CSS Class = **OraFieldText**
4. For the PositionCode item, set the following properties:
   CSS Class = **OraFieldText**

5. For the ManagerName item, set the following properties:

   External LOV = **`/<student#> /oracle/apps/ak/lov/webui/EmpNameLOVRN`**

   Note: Click the **Yes** button when the confirm dialog pops up.

   CSS Class = **OraFieldText**
6. When you set the ManagerName's External LOV property, the lovMappings will automatically appear. Expand the **ManagerName** item in the Structure panel to set the mappings.
7. Set lovMap1's properties as follows:

   LOV Region Item = **EmployeeName**

   Return Item = **ManagerName**

   Criteria Item = **ManagerName**
8. Right-click **lovMappings**, click **New** > **lovMap**. Set lovMap2's properties as follows:

   LOV Region Item = **EmployeeNumber**

   Return Item = **ManagerId**

   Criteria Item = **ManagerId**
9. For the Salary item, set the following properties:

   Required = **yes**

   Initial Value = **68320**

   Note: This illustrates one method of setting an initial value in OA Framework pages. FND_OA_ENABLE_DEFAULTS should be set to **Yes** for the defaults to work. If the default is not set, have your instructor set it for this course.

   CSS Class = **OraFieldText**
10. For the StartDate item, set the following properties:

    Required = **yes**

    CSS Class = **OraFieldText**

    Tip Type = **dateFormat**

    Note: this displays a standard date format hint below the field based on the current user's locale.
11. For the EndDate item, set the following properties:

    CSS Class = **OraFieldText**

    Tip Type = **dateFormat**

    Note: this displays a standard date format hint below the field based on the current user's locale.

Step 7: Extend a Region

The **Indicates Required Field** region is used to provide a hint to the user that the fields with the asterisk (*) are required fields. This region should render on the same line as the page-level action/navigation buttons. To achieve this, always add page-level keys to a pageStatus component.

1. Click the PageLayoutRN.
2. Right-click it, and click **New** > **pageStatus**.
3. Under the pageLayout Components, under pageStatus, click **region1**, and set its **ID** properties to **PageStatusRN**.
4. Right-click **PageStatusRN**, and click **New** > **Region**.
5. Click **region1**, and set its properties as follows:

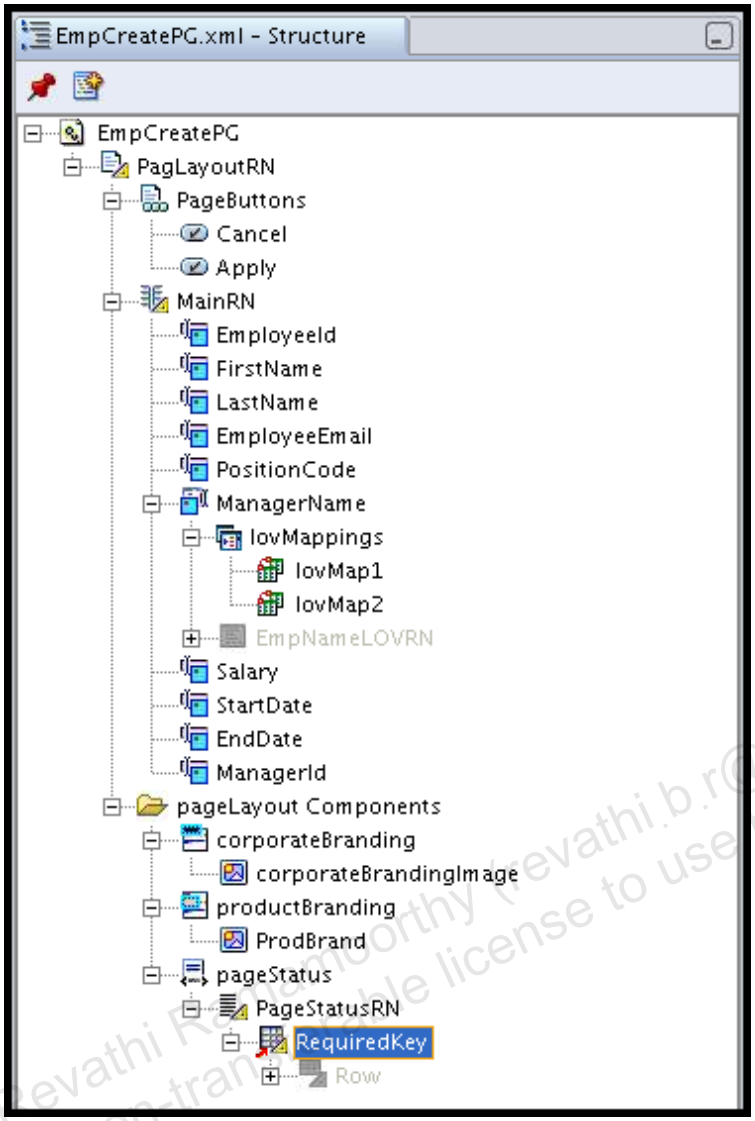
   ID = **RequiredKey**

   Extends = **`/oracle/apps/fnd/framework/webui/OAReqFieldDescRG`**

   Note: For the Extends property, uncheck the Show Components With Valid Scope Only checkbox in the dialog window for this case. At other times you may want to restrict the choices to components with a valid scope, such as in your particular application.

   Note: For additional information about configuring content to render in parallel with page-level action/navigation buttons, see Page Stamps in the OA Framework Developer's Guide.

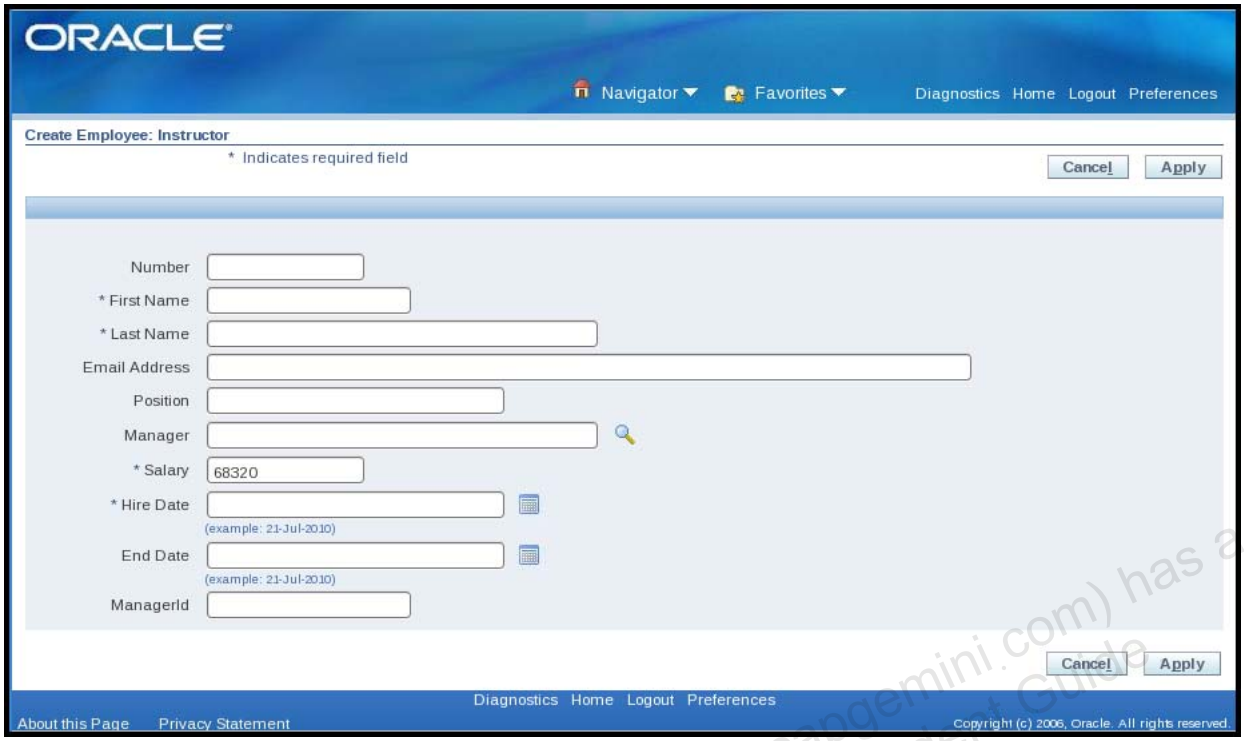
   Width = **100%**



Step 8: Test Your Work

1. Save your work
2. Run your **EmpCreatePG**.

   Note: It is possible to run the EmpCreatePG directly. However, some of the functionality is not enabled at this checkpoint. You will see the following behavior:
3. You will not see the employee number. You will see an empty input field instead.
4. Position will be an input field, not a poplist.
5. You will not be able to use the Manager LOV or the Apply button. If you try, you will see several error messages that say **No current row for View (EmployeeFullVO1)**. Once you add the appropriate logic later in this lab, this message will no longer appear.
6. You will not be able to do anything else in the Employees page because you have not yet added any logic.



Task 3: Implement the Poplist

The EmpCreatePG page will include a poplist for choosing the employee's position. You will create a view object to be used for the poplist data.

To learn more about poplists (and other standard basic components like text fields, checkboxes, radio buttons, list boxes and so forth), see Chapter 4 in the OA Framework Developer's Guide.

Step 1: Create a Poplist View Object

1. Right-click **ClassProject** , click **New** > **Business Tier** > **ADF Business Components** > **View Object**, and click the **OK** button.
2. Set the **Package** to **`<student#>.oracle.apps.ak.poplist.server`**.
3. Set the **Name** to **PositionsVO**.
4. Select the **Read-only Access** option.
5. Click the **Next** button.
6. Enter the following query:

```
SELECT MEANING, LOOKUP_CODE
FROM FWK_TBX_LOOKUP_CODES_VL
WHERE LOOKUP_TYPE = 'FWK_TBX_POSITIONS'
```

7. Click the **Test** button to validate your SELECT syntax.
8. Click the **Next** button until you are on the Java step.
9. Uncheck the **View Object Class: PositionsVOImpl: Generate Java File** checkbox.
10. Check the **View Row Class: PositionsVORowImpl: Generate Java File** and **Accessors** checkboxes.
11. Click the **Finish** button.

Step 2: Add the PositionsVO to the EmpCreateAM

1. Add the **PositionsVO** view object to the **EmpCreateAM**.

   Note: This is the first real example of the modifications that will slowly happen over the duration of the labs for this course. As the labs progress, simple processes, with which you should be familiar, will be explained briefly. If you need detailed explanations on the process, you will have to look at earlier labs and lab steps.

Step 3: Convert PositionCode Item to a Poplist

1. Click the **EmpCreatePG**.
2. For the **PositionCode** item, set the properties as follows:

   Item Style = **messageChoice**

   Required = **yes**

   Picklist View Definition = **`<student#>.oracle.apps.ak.poplist.server.PositionsVO`**

   Picklist Display Attribute = **Meaning**

   Picklist Value Attribute = **LookupCode**

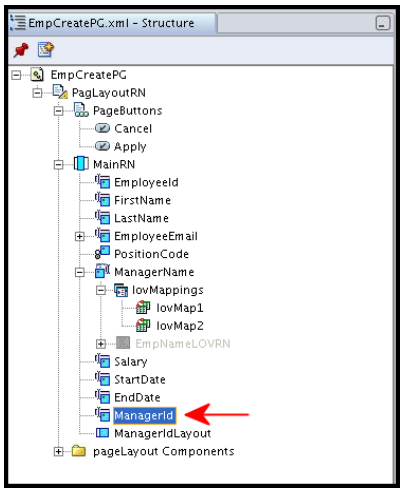
   CSS Class = **OraFieldText**

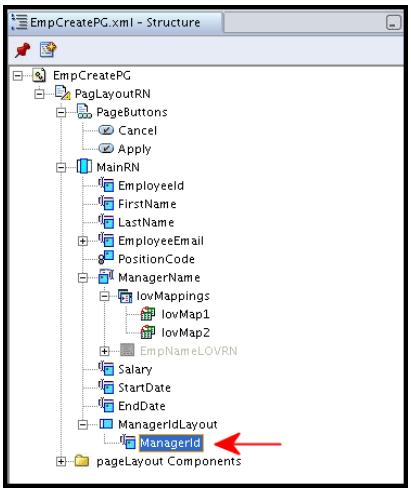
Step 4: Convert the MainRN in EmpCreatePG

As you did in the Drilldown to Details lab, change the MainRN `Region Style` to messageComponentLayout.

1. Click the **MainRN**, set its **Region Style** to **messageComponentLayout** from defaultSingleColumn. At the warning message, click the **Yes** button.
2. Right-click the **MainRN**, and click **New** > **messageLayout**. Set the **ID** property of messageLayout1 to **ManagerIdLayout**.
3. Click the **ManagerId** item, and drag and drop it into the ManagerIdLayout.

   Note: You have to create this extra region for the ManagerId item since you can add only message* items directly to a messageComponentLayout region, and for the LOV configuration, this hidden developer key needs to be a formValue item. To add any non-message* item to a messageComponentLayout region, add a messageLayout region first as shown here. To learn more about working with the messageComponentLayout region, see Page Layout (How to Place Content) in the Oracle Application Framework Developer's Guide 12.1.2 or higher.

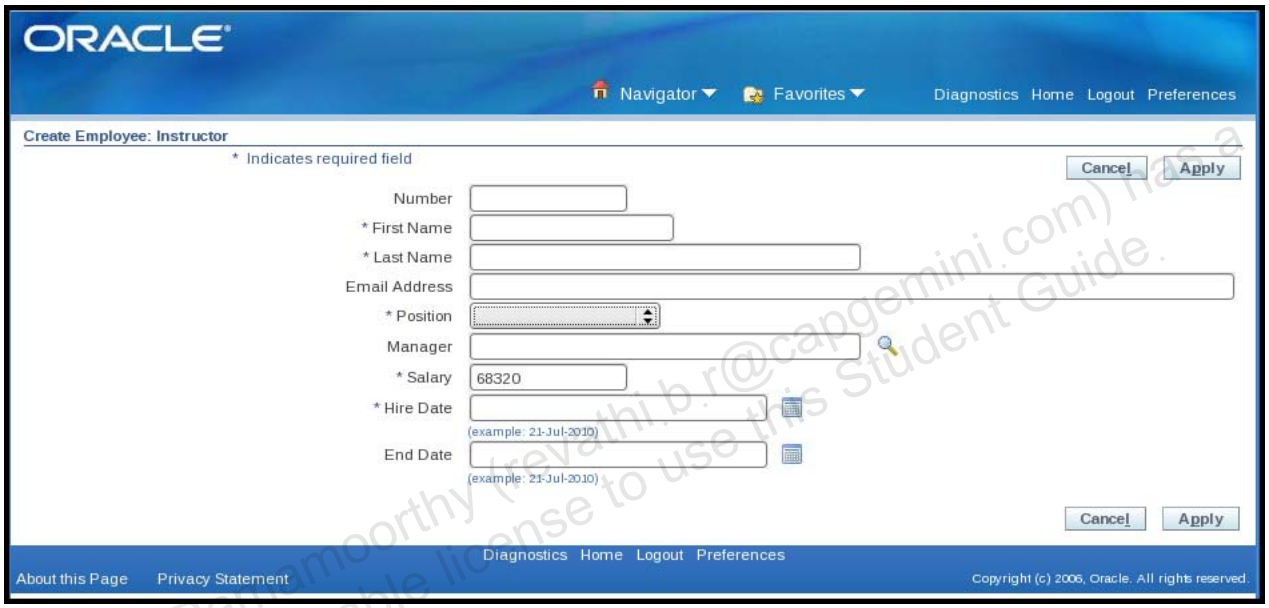


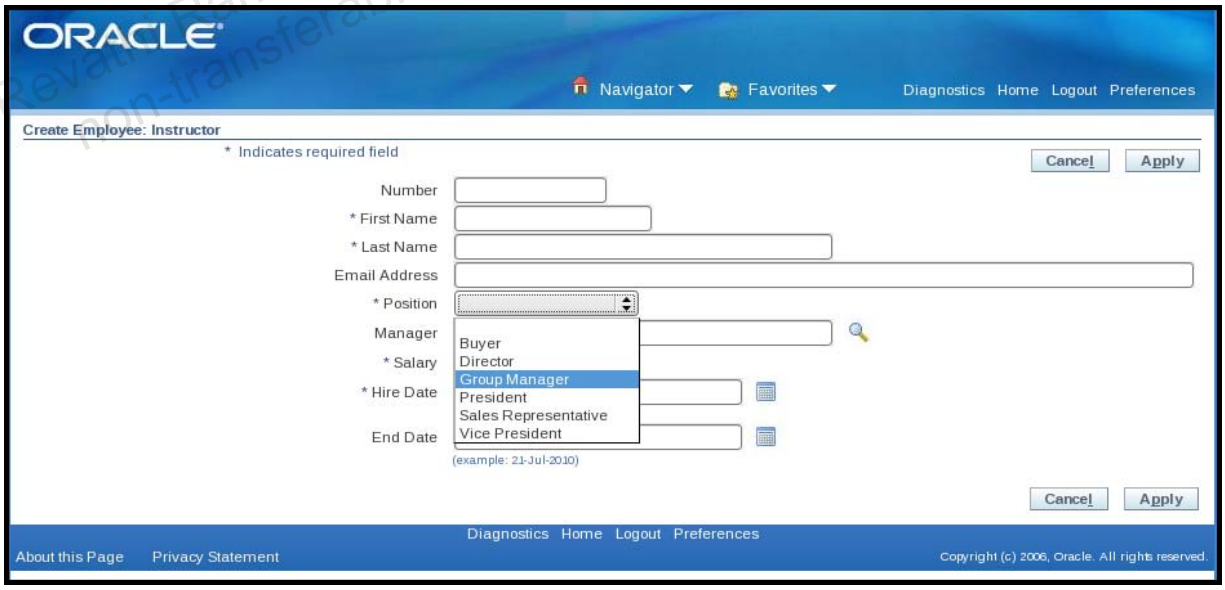


4. Click the **ManagerId** item, and set its **Item Style** to **formValue**.

Step 5: Test Your Work

1. Save your work.
2. Run the **EmpCreatePG**. Here is a list of expected behaviors:
   - You will not see the employee number. You will see an empty input field instead.
   - You should see Position as a poplist, and you should be able to click a value.
   - You will not be able to do anything else in the EmpCreatePG page because you have not yet added any logic.
   - ManagerId is hidden.





Task 4: Write Your Model-layer Programmatic Elements

You are going to implement some basic logic into the EmpCreatePG page that allows creation and saving of an employee without any validation. You will add validation and partial-page refresh code later.

Step 1: Add a createEmployee( ) Method to Your EmpCreateAM

Before you can fill out your EmpCreatePG page for a new employee, you must create a view object row. This view object row will receive page values once the user initiates a form submit (as by clicking the **Apply** button), which sends the new employee information to the middle tier.

1. Add the following code, including the import statements and the method code, to your **`EmpCreateAMImpl.java`** file.

```
import oracle.jbo.Row;
import oracle.apps.fnd.framework.OAViewObject;
...

public void createEmployee() {
    OAViewObject vo = (OAViewObject)getEmployeeDetailsVO1();

    // Per the coding standards, this is the proper way to
    // initialize a VO that is used for both inserts and
    // queries.  See View Objects in Detail in the Developer's
    // Guide for additional information.
    if (!vo.isPreparedForExecution()) {
        vo.executeQuery();
    }

    Row row = vo.createRow();
    vo.insertRow(row);
    // Required per OA Framework Model Coding Standard M69
    row.setNewRowState(Row.STATUS_INITIALIZED);
} // end createEmployee()
```

Step 2: Add an apply( ) Method to Your EmpCreateAM

1. Add the following code, including the import statements and the method code, to your **`EmpCreateAMImpl.java`** file.

```
import oracle.jbo.Transaction;
...

public void apply(){
   getTransaction().commit();
} // end apply()
```

```
public void createEmployee() {
    OAViewObject vo = (OAViewObject)getEmployeeDetailsVO1();

    // Per the coding standards, this is the proper way to
    // initialize a VO that is used for both inserts and
    // queries.  See View Objects in Detail in the Developer's
    // Guide for additional information.
    if (!vo.isPreparedForExecution()) {
        vo.executeQuery();
    }

    Row row = vo.createRow();
    vo.insertRow(row);
    // Required per OA Framework Model Coding Standard M69
    row.setNewRowState(Row.STATUS_INITIALIZED);
} // end createEmployee()

public void apply() {
    getTransaction().commit();
} // end apply()
```

Task 5: Write Your Controller-layer Programmatic Elements

For now, ignore the Cancel button. You will add logic for the Cancel button later.

Step 1: Create Your Controller

1. Right-click the **PageLayoutRN** region in the EmpCreatePG, and click Set **New Controller ...** from the context menu.
2. In the **Package Name** field enter **`<student#>.oracle.apps.ak.employee.webui`**.

   Note: remember that the default name puts webui in the name twice.
3. In the **Name** field enter **Emp**CreateCO.
4. Click the **OK** button.

Step 2: Add processRequest() Code to Your Controller

1. Add the following code, including the import statements and the processRequest() method code, to your EmpCreateCO.java file.

   Note: Some of the below code will already be there.

```
import oracle.apps.fnd.framework.OAApplicationModule;
...

public void processRequest(OAPageContext pageContext, OAWebBean
webBean) {
    // Always call this first.
    super.processRequest(pageContext, webBean);
    OAApplicationModule am =
pageContext.getApplicationModule(webBean);

    if (!pageContext.isFormSubmission()) {
        am.invokeMethod("createEmployee", null);
    }
} // end processRequest()
```

Step 3: Add processFormRequest() Code to Your Controller

When the user clicks the Apply button, you need to commit and navigate back to the EmployeePG page where you will display a confirmation message.

1. Add the following code, including the import statements and the processFormRequest() method code, to your EmpCreateCO.java file.

   Note: Some of the below code will already be there.

```
import oracle.jbo.domain.Number;
import oracle.apps.fnd.common.MessageToken;
import oracle.apps.fnd.framework.OAException;
import oracle.apps.fnd.framework.OAViewObject;
import oracle.apps.fnd.framework.webui.OAWebBeanConstants;
import oracle.apps.fnd.framework.webui.OADialogPage;

...



public void processFormRequest(OAPageContext pageContext,
                               OAWebBean webBean) {
    // Always call this first.
    super.processFormRequest(pageContext, webBean);
    OAApplicationModule am =
pageContext.getApplicationModule(webBean);

    // Pressing the "Apply" button means the transaction should
be
    // validated and committed.
    if (pageContext.getParameter("Apply") != null) {
        // Generally in the labs, we've illustrated
        // all BC4J interactions (except for the AMs).
        // Here, we deal with the VO directly so
        // the reasons why we're getting values from the VO,
        // not the request, make sense in context.
        OAViewObject vo =
(OAViewObject)am.findViewObject("EmployeeDetailsVO1");

        // Note  we have to get this value from the VO because
the
        // EO will assemble it during its validation cycle.
        // For performance reasons, we should generally be
calling
        // getEmployeeName() on the DetailsVORowImpl object, but
        // don't do this on the client so we're illustrating
        // the interface-appropriate call. If we implemented
this
        // code in AM where it belongs, we would use the other
        // approach.
        String employeeName =
            (String)vo.getCurrentRow().getAttribute("FirstName")
+ " "
```

```
                +
(String)vo.getCurrentRow().getAttribute("LastName");
        // We need to get a String so we can pass it to the
        // MessageToken array below. Note that we are getting
this
        // value from the VO (we could also get it from the Bean
        // shown in the Drilldown to Details lab) because the
item
        // style is messageStyledText, so the value isn't put on
        // request like a messaqeTextInput value is.
        Number employeeNumber =

(Number)vo.getCurrentRow().getAttribute("EmployeeId");
        String employeeNum =
String.valueOf(employeeNumber.intValue());

        MessageToken[] tokens =
        { new MessageToken("EMP_NAME", employeeName),
          new MessageToken("EMP_NUMBER", employeeNum) };
        OAException confirmMessage =
            new OAException("AK",
"FWK_TBX_T_EMP_CREATE_CONFIRM", tokens,
                            OAException.CONFIRMATION, null);
        am.invokeMethod("apply");

        OADialogPage dialogPage =
            new OADialogPage(OAException.CONFIRMATION,
confirmMessage,
                                null,
"OA.jsp?page=/<student#>/oracle/apps/ak/employee/webui/EmployeeP
G",

"OA.jsp?page=/<student#>/oracle/apps/ak/employee/webui/EmpCreate
PG");
        // We set this value so the code that handles this
button
        // press is descriptive.
        dialogPage.setOkButtonItemName("CreateOKButton");
        dialogPage.setOkButtonLabel("To Employee Query Page");
        dialogPage.setNoButtonLabel("Add Another Employee");
        dialogPage.setPostToCallingPage(true);

        pageContext.redirectToDialogPage(dialogPage);
    }
}
```

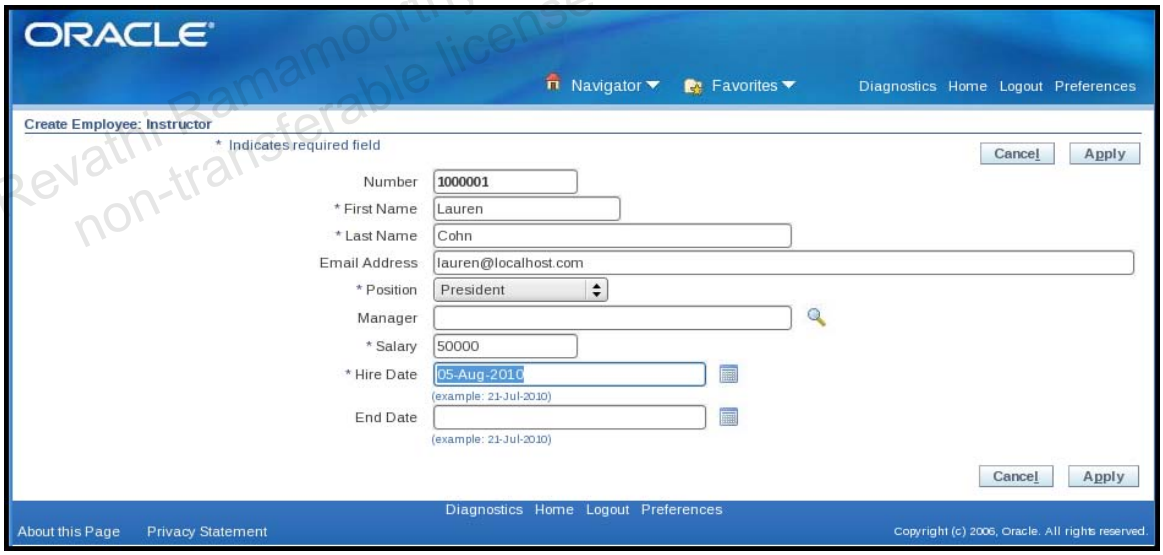
Step 4: Test Your Work

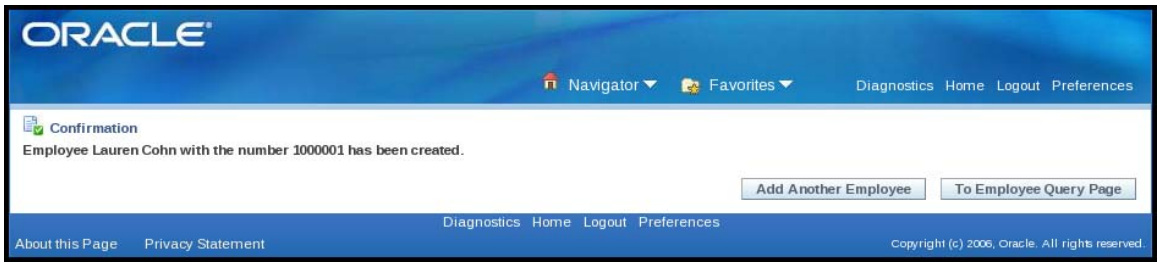
1. Save your work.
2. Rebuild/compile the **ClassProject** .
3. Run the **EmpCreatePG**.
4. Enter an employee, choosing any employee number you want then click the **Apply** button. For now, choose employee numbers greater than 10000. Later, you will populate the field from a sequence. Small employee numbers might conflict with the sequence-generated employee numbers.

   Note: that because the employee number must be unique, you may get errors that the value needs to be unique.
5. The confirmation message should display in a dialog window with buttons to return to the EmpCreatePG or continue on to the EmployeePG.
6. Write the Number of your Employee Down.
7. Search for your new employee by typing in the new employee number and clicking the **Go** button.

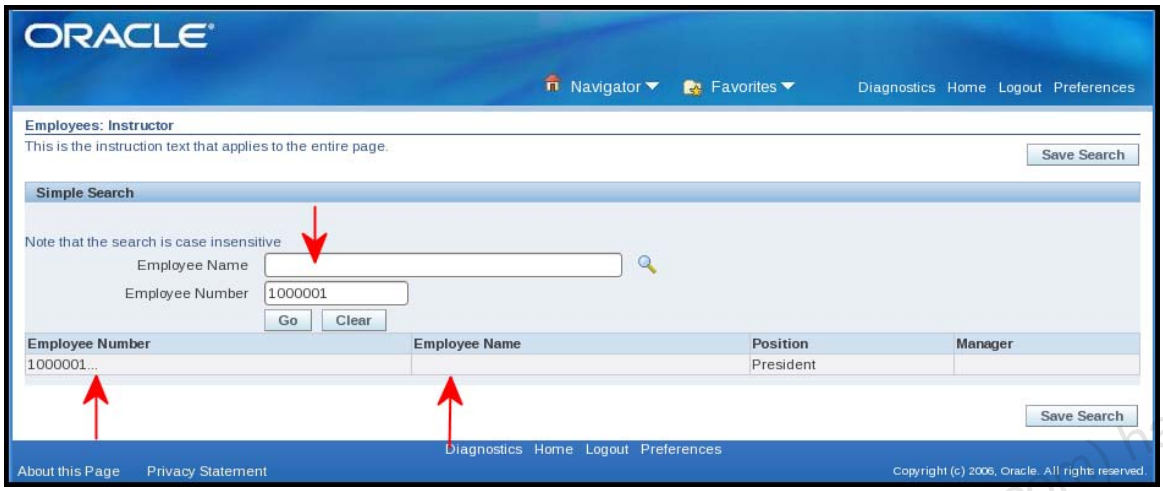
   Note: you will not see the employee name because combining the first name and last name into the employee name happens in the validation you have not created. However, you will see the employee number and the position, and if you assigned a manager name that will show as well.

   Note: If include an End Date, the SQL query that is part of EmpNameLOVVO will not return the name so leave that blank for now.

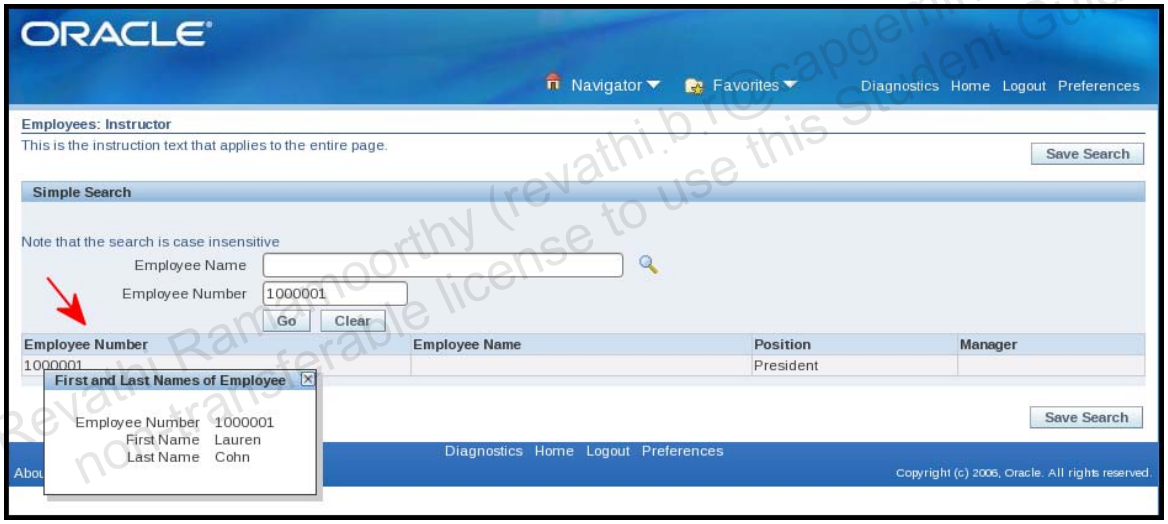




8. Employee Name and the Look Ahead LOV will not show the names at this point because the First and Last name of the employee you just created has not been concatenated (FullName) at this point. However, if you enter the number, the PopUp should work fine showing the first and last names of your employee.



Note: The Employee Number Popup still works



Task 6: Revise Your EmployeePG

Step 1: Add a create Button to Your EmployeePG

To access the EmpCreatePG page from the EmployeePG page, revise the EmployeePG page, to include a Create Employee button above the results table. To do this, add a tableAction component to the ResultsRN table that you created in the EmployeePG.

1. Right-click the **ResultsRN** table region of the EmployeePG in the Structure pane, and click **New** > **tableActions** from the context menu.
2. JDeveloper automatically creates a table components and tableActions to the ResultsRN as well as a flowLayout region for you. Change the flowLayout region's `ID` to **ButtonLayout**.
3. Right-click the **ButtonLayout** region, and click **New** > **Item**. Set this item's properties as follows:

   ID = **Create**

   Item Style = **submitButton**

   Attribute Set = **`/oracle/apps/fnd/framework/toolbox/attributesets/FwkTbxEmployees/CreateEmployee`**

   Note: There are other attributes sets that can be used for this button. However, this is a pre-configured attribute, and there are additional settings done for you.
4. Save your work.

Step 2: Add a Controller to Handle the Button Press

When the user clicks the Create Employee submit button, you need the processFormRequest() method to determine that this button has been pressed and navigate the user to the EmpCreatePG.

1. Right-click the **ResultsRN** table region of the EmployeePG, and click **Set New Controller …** from the context menu.
2. Set the **Package** Name to **`<student#>.oracle.apps.ak.employee.webui`**.

   Note: Beware the double webui package name.
3. Set the **Name** to **EmpResultsCO**.
4. Click the **OK** button.

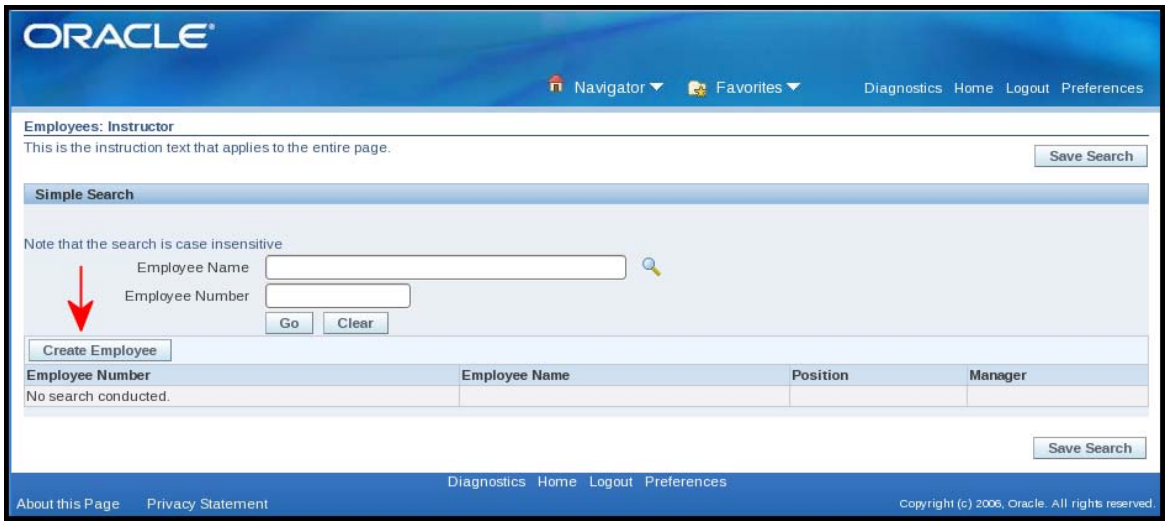
5. Enter the following import and java code for your **`EmpResultsCO.java`** file.

```
import oracle.apps.fnd.framework.webui.OAWebBeanConstants;
...

public void processFormRequest(OAPageContext pageContext,
OAWebBean webBean) {
  super.processRequest(pageContext, webBean);
  if (pageContext.getParameter("Create") != null) {

pageContext.setForwardURL("OA.jsp?page=/<student#>/oracle/apps/a
k/employee/webui/EmpCreatePG",
                          null,
                          OAWebBeanConstants.KEEP_MENU_CONTEXT,
                          null, null, false,

OAWebBeanConstants.ADD_BREAD_CRUMB_YES,
                          OAWebBeanConstants.IGNORE_MESSAGES);
  }
}
```

Step 3: Test your work

1. Save your work.
2. Rebuild **ClassProject** .
3. Run the **EmployeePG**.
4. Click the **Create Employee** button, and verify that it takes you to your Create Employee page.

   Note: Once at the Create Employee page, you cannot return to the EmployeePG page because you have not programmed the logic to do so. Because you are building a transaction, you do not want to add default page navigation.



Part 2: Validation Overview

For this lab, you will implement validations on the Employee Create page. This lab assumes, and relies upon, you having completed Part 1 of this lab. If you have not yet finished Part 1 of the lab, please do so before continuing with this lab.

After completing this lab, you will have:

- Implemented typical entity object and entity expert attribute and row-level business logic, including throwing attribute and row-level validation exceptions.
- Used validation view objects and validation application modules in your business logic.
- Ensured your transaction flow is browser Back button safe.

Task 7: Handle the Back Button

Step 1: Make Your Page Handle the Back Button

The EmpCreatePG requires logic to ensure that the user does not encounter errors due to navigation using the browser Back button. For example, it is important to ensure that there are no partially created EmployeesEO entity objects in the middle-tier cache when the user tries to create a new employee. While the solution prevents page access after invalid Back button navigation, the code below allows the page to be properly rebuilt if the transaction is passivated.

1. Add the following code to the end of your **EmpCreateCO's** `processRequest()` method.

```
import oracle.apps.fnd.framework.webui.TransactionUnitHelper;

...

// If isBackNavigationFired = false, we're here after a valid
// navigation (the user selected the Create Empoyee button) and
// we should proceed normally and initialize a new employee.

if (!pageContext.isBackNavigationFired(false)) {
  // We indicate that we are starting the create transaction
  // (this  is used to ensure correct Back button behavior).
  TransactionUnitHelper.startTransactionUnit(pageContext,
                                             "empCreateTxn");
  // This test ensures that we don't try to create a new
  // employee if we had a JVM failover, or if a recyled
  // application module is activated after passivation.
  // If these things happen, BC4J will be able to find
  // the row that you created so the user can resume work.
} else {
  if
(!TransactionUnitHelper.isTransactionUnitInProgress(pageContext,

"empCreateTxn",
                                                    true)) {
    // We got here through some use of the browser "Back"
    // button, so we want to display a stale data error and
    // disallow access to the page. If this were a real
    // application, we would probably display a more
    // context-specific message telling the user she can't
    // use the browser "Back" button and the "Create" page.
    // Instead, we wanted to illustrate how to display the
    // Applications standard NAVIGATION ERROR message.
```

```
    OADialogPage dialogPage = new
OADialogPage(NAVIGATION_ERROR);
    pageContext.redirectToDialogPage(dialogPage);
  }
}
```

Step 2: Add rollbackEmployee( ) Method to Your EmpCreateAM

1. Add the following method to your **EmpCreate`AMImpl`** class for rolling back the database and the middle tier. This method will later be called by the Emp`ResultsCO` and Emp`CreateCO` controllers.

   Note: You call `getTransaction()` here to use an `oracle.jbo.Transaction` instead of calling `getOADBTransaction()` to get an OAF subclass `oracle.apps.fnd.framework.server.OADBTransaction` because the superclass has the behavior you need, and as a rule, you should always instantiate the class that includes the behavior you want to use. Avoid instantiating subclasses if the additional behavior is not required.

```
/*
*************************************************************
*
* Executes a rollback including the database and the middle
tier.
*************************************************************
*
*/

public void rollbackEmployee() {
  Transaction txn = getTransaction();
  // This small optimization ensures you don't perform a
rollback
  // if you don't have to.
  if (txn.isDirty()) {
    txn.rollback();
  }
} // end rollbackEmployee()
```

Step 3: Make Your EmpResultsCO Handle the Back Button

1. To account for various Back button navigation patterns between the EmpCreatePG and the EmployeePG, add the following logic to the **Emp`ResultsCO`** controller's `processRequest()` method. This ensures that any unfinished employee objects are cleared from the middle tier cache when the user navigates to EmployeePG using the browser Back button and then tries to create a new employee.

   Tip: make sure you are adding this logic to the correct controller: Emp`ResultsCO`.

Note: When you are ready to start building your own applications, you can learn more about coding for the browser Back button in the OA Framework Developer's Guide advanced topic Supporting the Browser Back Button. You can learn about passivation in the OA Framework State Management (Passivation) topic.

```
import oracle.apps.fnd.framework.webui.TransactionUnitHelper;
import oracle.apps.fnd.framework.OAApplicationModule;

...

public void processRequest(OAPageContext pageContext, OAWebBean
webBean) {
  super.processRequest(pageContext, webBean);
  OAApplicationModule am =
pageContext.getApplicationModule(webBean);
  // The following checks if the user navigated back to this
  // page without taking actions that cleared an "in
transaction"
  // indicator. If so, rollback any abandoned changes to
  // ensure they aren't left lingering in the BC4J cache to
cause
  // problems with subsequent transactions.  For example, if the
  // user navigates to the Create Employee page where you start
a
  // "create" transaction unit, then navigates back to this page
  // using the browser Back button and selects the Create
Employee
  // button again, OA Framework detects this Back button nav.
And
  // steps through processRequest() so code is executed before
  // you try to create another new employee.
  if
(TransactionUnitHelper.isTransactionUnitInProgress(pageContext,
        "empCreateTxn", false))
    {
    am.invokeMethod("rollbackEmployee");
    TransactionUnitHelper.endTransactionUnit(pageContext,
                                             "empCreateTxn");
  }
}
```

Step 4: Handle the Cancel Button

When the user clicks the Cancel button on the EmpCreatePG, you need to rollback the transaction and navigate back to the EmployeePG page (the rollback is required per the Back button guidelines in this particular use case).

1. Add the following code to the end of the `processFormRequest` method in the **EmpCreateCO** controller to handle the Cancel button selection.

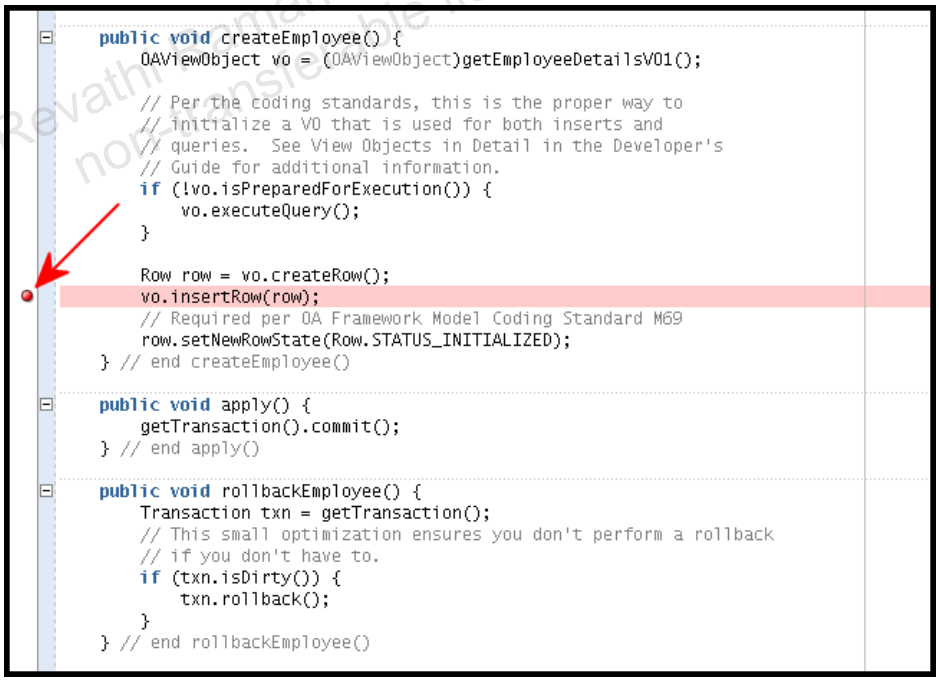
   Tip: make sure you are adding this logic to the correct controller, **EmpCreateCO**.

```
if (pageContext.getParameter("Cancel") != null) {
  am.invokeMethod("rollbackEmployee");
  // Indicate that the Create transaction is complete.
  TransactionUnitHelper.endTransactionUnit(pageContext,
                                           "empCreateTxn");

pageContext.forwardImmediately("OA.jsp?page=/<student#>/oracle/a
pps/ak/employee/webui/EmployeePG",
                               null,

OAWebBeanConstants.KEEP_MENU_CONTEXT,
                               null, null, false,

OAWebBeanConstants.ADD_BREAD_CRUMB_NO);
}
```

Step 5: Test Your Work and a little debugging.

1. Save your work.
2. Rebuild **ClassProject** .
3. Open **`EmpCreateAMImpl.java`**, and add a breakpoint to the **first executable line** of code in the **createEmployee()** method.

   Note: To add a breakpoint to the code, click the left column of the line of code in the code editor window. When you click, a red dot (breakpoint) appears. It is a toggle. So, if you click again, the breakpoint will be removed.



```
public void createEmployee() {
    OAViewObject vo = (OAViewObject)getEmployeeDetailsVO1();

    // Per the coding standards, this is the proper way to
    // initialize a VO that is used for both inserts and
    // queries.  See View Objects in Detail in the Developer's
    // Guide for additional information.
    if (!vo.isPreparedForExecution()) {
        vo.executeQuery();
    }

    Row row = vo.createRow();
    vo.insertRow(row);
    // Required per OA Framework Model Coding Standard M69
    row.setNewRowState(Row.STATUS_INITIALIZED);
} // end createEmployee()

public void apply() {
    getTransaction().commit();
} // end apply()

public void rollbackEmployee() {
    Transaction txn = getTransaction();
    // This small optimization ensures you don't perform a rollback
    // if you don't have to.
    if (txn.isDirty()) {
        txn.rollback();
    }
} // end rollbackEmployee()
```

4. Add a breakpoint to the **vo.insertRow(row)** line of the **createEmployee()** method.
5. Add a breakpoint to the **first executable line** of the **rollbackEmployee()** method.

```
        Row row = vo.createRow();
        vo.insertRow(row);
        // Required per OA Framework Model Coding Standard M69
        row.setNewRowState(Row.STATUS_INITIALIZED);
    } // end createEmployee()

    public void apply() {
        getTransaction().commit();
    } // end apply()

    public void rollbackEmployee() {
        Transaction txn = getTransaction();
        // This small optimization ensures you don't perform a rollback
        // if you don't have to.
        if (txn.isDirty()) {
            txn.rollback();
        }
    } // end rollbackEmployee()
```

6. Open **`EmployeesEOImpl.java`**, and add a breakpoint to the **first executable line** of code in the **create()** method.

```
     */
    public void create(AttributeList attributeList) {
        super.create(attributeList);
    }
```

7. Open **`EmpCreateCO.java`**, and add a breakpoint to the **first executable line** of code in the **processRequest()** method.

```
public void processRequest(OAPageContext pageContext, OAWebBean webBean)
{
  super.processRequest(pageContext, webBean);
  OAApplicationModule am = pageContext.getApplicationModule(webBean);
```

8. Add a breakpoint to the **first executable line** of code in the **processFormRequest()** method.

```
public void processFormRequest(OAPageContext pageContext, OAWebBean webBean)
{
  super.processFormRequest(pageContext, webBean);
    OAApplicationModule am = pageContext.getApplicationModule(webBean);
```

9. Open **`EmpResultsCO.java`**, and add a breakpoint to the **first executable line** of code in the **processRequest()** method.

```
public void processRequest(OAPageContext pageContext, OAWebBean webBean)
{
  super.processRequest(pageContext, webBean);
    OAApplicationModule am = pageContext.getApplicationModule(webBean);
```

10. Step through your code using the debugger to examine what happens during the page running and how the page handled back buttons. Right-click the **EmployeePG**, and click **Debug** from the context menu.

    Note: Your instructor may ask you to skip debugging as performance and classroom limitations may take all available network resources.

11. The first break that you encounter is when the EmployeePG renders and the EmpResultsCO is executed because it is the first controller that is part of the ResultsRN, and fires as part of the ProcessRequest. Press **(F9)** or click the **continue** icon.

12. On the EmployeePG, click the **Create Employee** button.

13. When you click the Create Employee button, the next breakpoint in the EmpCreateCO fires and the page searches for the pageContext.
14. Press **(F9)** or click the **continue** icon.
15. The next break you encounter is in the EmployeeEOImpl as the data is queried and the create attribute list method fires.
16. Press **(F9)** or click the **continue** icon.
17. The next break occurs in the EmpCreateAMimpl as the VO and the rows are built from the attributes taken from the previous step in the EO.
18. Press **(F9)** or click the **continue** icon.
19. The next break checks to see about the transaction state.
20. Press **(F9)** or click the **continue** icon.
21. The EmpCreatePG renders with the correct rows and attributes.
22. Enter a new Employee using a high number again. When you enter your data for the new employee and then click the **Apply** button.

    Note: The page runs through the breakpoints again.
23. Return to the EmployeePG page by clicking the browser's **back** button.

24. No processRequest() method fires, because your page is not aware that you are using the back button on the browser on the EmployeePG.

    Note: You can stop execution of the page at this point. While the code was not essential at this point, it was a good introduction and a good debugging exercise.
25. To clear or reset the breakpoint(s), click on the breakpoint.

Task 8: Implement Declarative Validations

Simple business rules should, with rare exception, be implemented at the entity object (EO) level. All rules can be implemented programmatically (i.e., in Java). Some, but not all, of the rules can be implemented declaratively (i.e., via XML). When you have a choice between declarative and programmatic validation, the declarative implementation is preferred.

The table below is for information purposes only. The steps that follow this table are the precise steps to implement the validations outlined in this summary table.

Entity Attribute	Business Rules for Validation
EmployeeId	• Defaults when an Employee entity object is created using the FWK_TBX_EMPLOYEES_S sequence to generate your unique identifier. • Must be unique across the system. • Required; cannot be null. • Cannot be updated on a committed row.
FirstName	• Required; cannot be null
LastName	• Required; cannot be null
FullName	• Must be built programmatically by concatenating the LAST_NAME, FIRST_NAME (example: Smith, Joe) • Must be updated whenever FIRST_NAME or LAST_NAME changes.
ManagerId	• Optional for anyone with the positionCode of PRESIDENT; required for everyone else • Must refer to an active employee (valid means the employee is defined in the employees table and end_date is null).
PositionCode	• Must be a valid position (valid means it exists in the FWK_TBX_LOOKUP_CODES_B table)
StartDate	• Defaults to SYSDATE when an Employee entity object is created • Must be >= sysdate • Required; cannot be null • Cannot be changed on a committed row
EndDate	• If specified, must be >= START_DATE • Must be >= sysdate

Salary	• Required; cannot be null • Must be > 0

Step 1: Set Your Declarative Validations

To implement the business rules in the prior table, you are going to set the properties in the EmployeeEO so that declarative validation occurs.

1. Double-click the **EmployeeEO** in the Navigator to open the Entity Object Editor.
2. Expand the **Attributes** category.
3. Click the **EmployeeId** attribute, and select the **Updatable: While New** option. Click the **Apply** button.
4. Click the **FirstName** attribute, and check the **Mandatory** checkbox. Click the **Apply** button.
5. Click the **LastName** attribute, and check the **Mandatory** checkbox. Click the **Apply** button.
6. Click the **Salary** attribute, and check the **Mandatory** checkbox. Click the **Apply** button.
7. Click the **StartDate** attribute, and check the **Mandatory** checkbox and select the **Updatable: While New** option.
8. Click the **Apply** button.
9. Click the **OK** button.
10. Save your work.
11. The declarative validation you just set, should not change the current behavior of your page, so you do not need to test anything now.

Task 9: Implement Programmatic Validations

Note: If possible, you are encouraged to cut-n-paste the validation logic introduced in the next several tasks from the solution text provided by your instructor. It is quite time-consuming to type this in directly, the risk of mistakes is higher, and you might not include the comments that will help explain what is happening as you debug the code.

If the solution is not available in an electronic form on your classroom desktop, take your time and type slowly and accurately.

Step 1: Add Import Statements

1. Add the following import statements to your **`EmployeeEOImpl`** class. These will be used in the various business logic steps that follow.

```
import oracle.apps.fnd.framework.OAAttrValException;
import oracle.apps.fnd.framework.OARowValException;
import oracle.apps.fnd.framework.server.OADBTransaction;
import oracle.apps.fnd.framework.OAException;
import oracle.jbo.server.EntityImpl;
```

Note: Within the context of the entity object, you use an `OADBTransaction` (unlike the superclass `Transaction` that you used in our application module) because the `OAEntityImpl` superclass provides a convenience `getOADBTransaction()` method for us to use in our `EmployeeEOImpl` subclass.

Step 2: Validate and Set the EmployeeId in EmployeeEOImpl

1. Add the following validation logic to the **`setEmployeeId()`** method in the **`EmployeeEOImpl`** class. This verifies that the EmployeeId is unique (it checks both the entity cache and the database), and throws an attribute-level validation error if it is not.

   Note: This `setEmployeeId()` method will be called by defaulting logic in the `create()` method (which you will add soon). The call from the `create()` method will pass a sequence number argument. Then `setEmployeeId()` will validate that the employee number is unique, and then set the employee number value on the entity object by calling `setAttributeInternal()`.

```
public void setEmployeeId(Number value) {
  // Because of the declarative validation that you specified
for
  // this attribute, BC4J validates that this can be updated
only
  // on a new line, and that this mandatory attribute is not
null.
  // This code adds the additional check of only allowing an
  // update if the value is null to prevent changes while the
  // object is in memory.
  if (getEmployeeId() != null) {
```

```
      throw new OAAttrValException(OAException.TYP_ENTITY_OBJECT,
                                   getEntityDef().getFullName(),
                                   getPrimaryKey(), "EmployeeId",
                                   value, "AK",
                                   "FWK_TBX_T_EMP_ID_NO_UPDATE");
    }
    if (value != null) {
      // Employee id must be unique. To verify this, you must
      // check both the entity cache and the database. In this
      // case, it's appropriate to use findByPrimaryKey()
      // because you're unlikely to get a match, and are
      // therefore unlikely to pull a bunch of large objects
      // into memory. Note that findByPrimaryKey() is
      // guaranteed to check all employees. First it checks
      // the entity cache, then it checks the database.
      OADBTransaction transaction = getOADBTransaction();
      Object[] employeeKey = { value };
      EntityDefImpl empDefinition =
EmployeeEOImpl.getDefinitionObject();
      EmployeeEOImpl employee =
(EmployeeEOImpl)empDefinition.findByPrimaryKey(transaction,
                                     new Key(employeeKey));
      if (employee != null) {
        throw new
OAAttrValException(OAException.TYP_ENTITY_OBJECT,
                                   getEntityDef().getFullName(),
                                   getPrimaryKey(),
"EmployeeId",
                                   value, "AK",
                                   "FWK_TBX_T_EMP_ID_UNIQUE");
      }
    }
    // Note that this is the point at which the value is actually
    // set on the EO cache (during the scope of
setAttributeInternal
    // processing).  If you don't make this call after you perform
    // your validation, your value will not be set correctly.
    // Also, any declarative validation that you define for this
    // attribute is executed within this method.
    setAttributeInternal(EMPLOYEEID, value);
  }
```

Step 3: Set Your create() Method in EmployeeEOImpl.

1. In this step, you set the default value (EmployeeId) for your EmpCreatePG page by overriding the `create()` method in your `EmployeeEOImpl` class. The `create()` method is called when your `EmployeeEOImpl` class is instantiated. Any programmatic defaulting logic should be added to this method. In this case, you are setting an EmployeeId based on a database sequence value.

   Note: You call the `set<AttributeName>()` methods and pass the values you want to set. This ensures that all attribute-level validation is performed.

```
public void create(AttributeList attributeList) {
  super.create(attributeList);

  OADBTransaction transaction = getOADBTransaction();
  Number employeeId =
transaction.getSequenceValue("FWK_TBX_EMPLOYEES_S");
  setEmployeeId(employeeId);
}
```

Step 4: Make a Minor UI Change

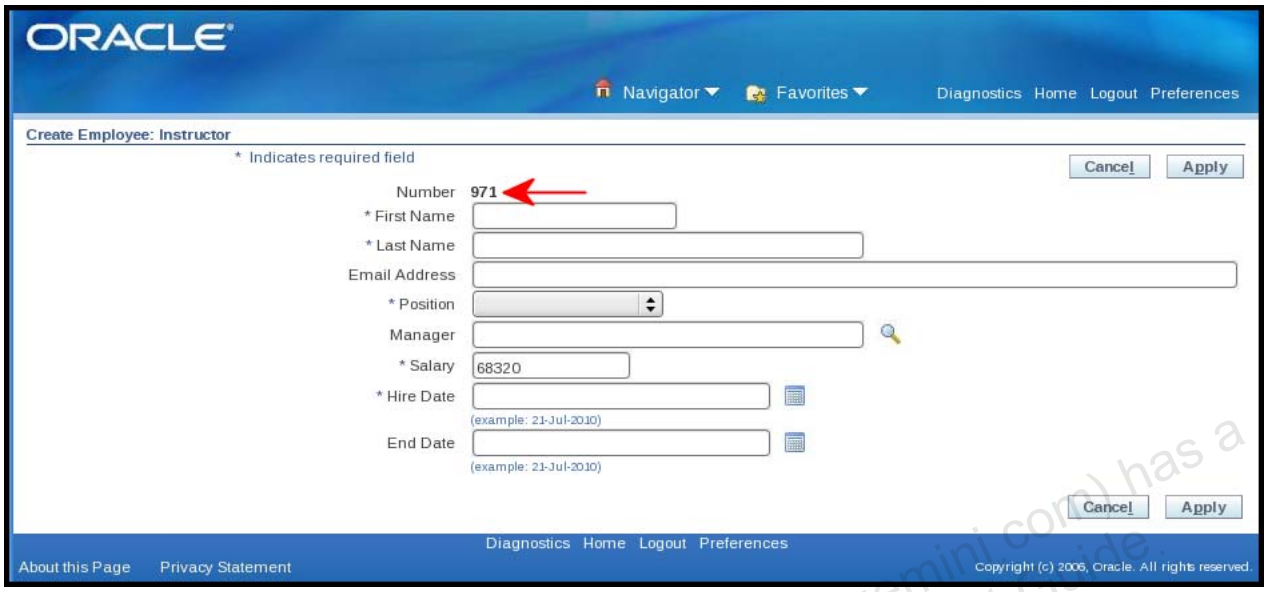
Normally, making programmatic validation changes would not imply any connection to your UI element. However, the EmpCreatePG has an EmployeeId item that has, until now, been an messageTextInput style item. Now that the EmployeeId is being derived from a sequence and an EO-based validation, you should change the item to be display-only.

2. Click the **EmployeeId** item on your **EmpCreatePG**.
3. Change the **Item Style** to **messageStyledText**.

Step 5: Test Your Work

1. Save your work.
2. Rebuild **ClassProject** .
3. Run the **EmployeePG**.

4. Click the **Create Employee** button. When the EmployeePG loads, your EmployeeId Number should now appear automatically. You should also see the asterisks for mandatory fields.



Step 6: Set the StartDate

1. Add the following setter logic to the **`setStartDate()`** method in the **`EmployeeEOImpl`** class. In this case, you are delegating to a `validateStartDate()` method because you ultimately want to perform the start date validation again at the entity level, so you want to be able to exercise that validation independent of the attribute setting. Once `validateStartDate()` performs the actual validation, `setStartDate()` will set the date on the entity object by calling `setAttributeInternal()`.

```
public void setStartDate(Date value) {
  validateStartDate(value);
  setAttributeInternal(STARTDATE, value);
}
```

Note: the above code will not compile until you add the method in the next step.

Step 7: Validate the StartDate

1. Create a **`validateStartDate()`** method, and add this code to it. It is protected because the `EmployeeEOImpl` class is the only expected caller, but you want to ensure that subclasses can also leverage this. If you marked it as private, subclasses could not call it. This method demonstrates correct `oracle.jbo.domain.Date` comparison handling.

   Tip: Since the entity object classes can be large, it is helpful to co-locate related code. For example, you should add the following method immediately after the **`setStartDate()`** method.

```
protected void validateStartDate(Date value) {
  // BC4J ensures that this mandatory attrib has a non-null
value.
  if (value != null) {
    OADBTransaction transaction = getOADBTransaction();
    // Note you truncate the values to allow for the possibility
    // that we're trying to set them to be the same day. Calling
    // dateValue( ) does not include time. If you want the
    // time element call timestampValue(). Finally, you cannot
    // compare oracle.jbo.domain.Date objects directly. Instead,
    // convert the value to a long as shown.
    long sysdate =
transaction.getCurrentDBDate().dateValue().getTime();
    long startDate = value.dateValue().getTime();
    if (startDate < sysdate) {
      throw new
OAAttrValException(OAException.TYP_ENTITY_OBJECT,
                                    getEntityDef().getFullName(),
                                    getPrimaryKey(),
                                    "StartDate",
                                    value,
                                    "AK",
                                    "FWK_TBX_T_START_DATE_PAST");
    }
  }
}
```

Step 8: Set the StartDate During Create

In this step, you set the default value for StartDate for your EmpCreatePG page by adding to your `create()` method in your `EmployeeEOImpl` class. The `create()` method is called when your `EmployeeEOImpl` class is instantiated. Any programmatic defaulting logic should be added to this method. In this case, you are setting the start date to the database `SYSDATE`.

Note: You call the `set<AttributeName>()` methods and pass them the values you want to set. This ensures that all attribute-level validation is performed.

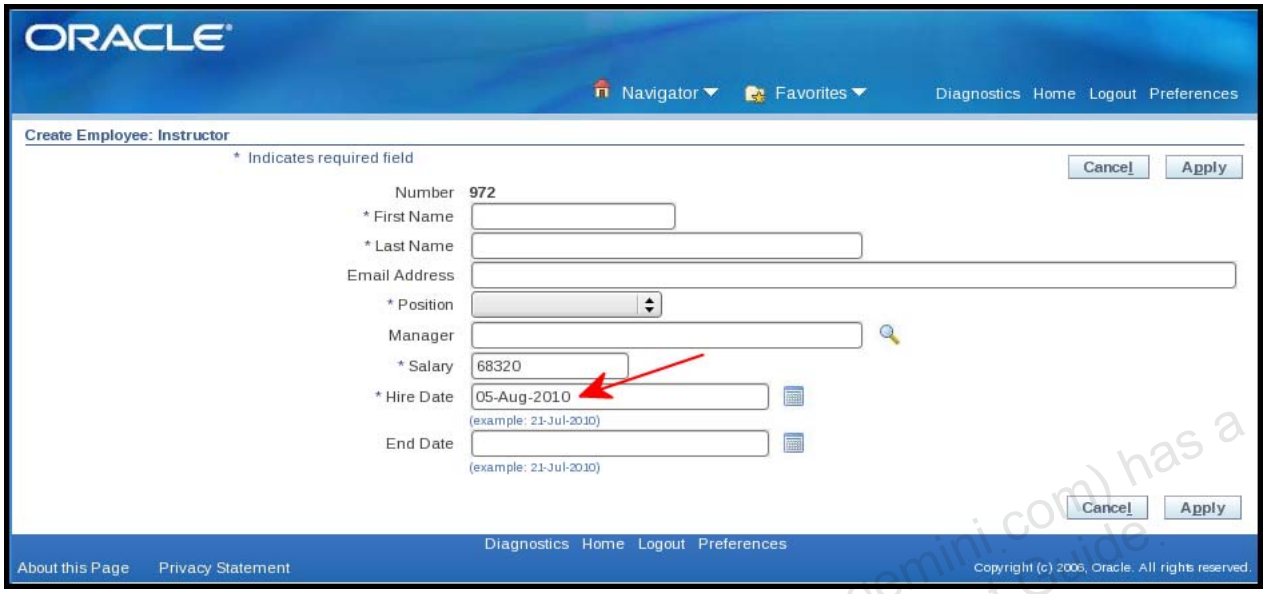
1. Add the following **setStartDate()** call right after the `setEmployeeId(employeeId);` line in the **create()** method.

```
// Start date should be set to sysdate
setStartDate(transaction.getCurrentDBDate());
```

Step 9: Test Your Work

2. Save your work.

3. Rebuild **ClassProject** .
4. Run the **EmployeePG**, and click the **Create Employee** button. You results should appear similar to the following:



Step 10: Validate and Set the FirstName

Now, you are going to implement the business rules to handle changes to the `FirstName` and `LastName` attributes. In each method, you check to see if the new value differs from the old value, and if it does, you combine the new name values into a `FullName` value in the form of LastName, FirstName, and then call the `FullName` setter. Since you do not need any additional validation in the `setFullName()` method, you do not need to add any logic to it.

1. Add the following code to the **`setFirstName()`** method in your **`EmployeeEOImpl`** class.

```
public void setFirstName(String value) {
  // BC4J will make sure this value is not null. You still need
to
  // check before trying to set the full name.
  if ((value != null) || (!("".equals(value.trim())))) {
    String oldFirstName = getFirstName();
    if (oldFirstName == null) {
      oldFirstName = "";
    }
    // If you have a new firsttname value, update the full name.
    if (value.compareTo(oldFirstName) != 0) {
      String lastName = getLastName();
      if (lastName == null) {
        lastName = "";
      }
      setFullName(lastName.concat(", ").concat(value));
```

```
    }
  }
  setAttributeInternal(FIRSTNAME, value);
}
```

Note: You will now have a FullName as a result of this addition.

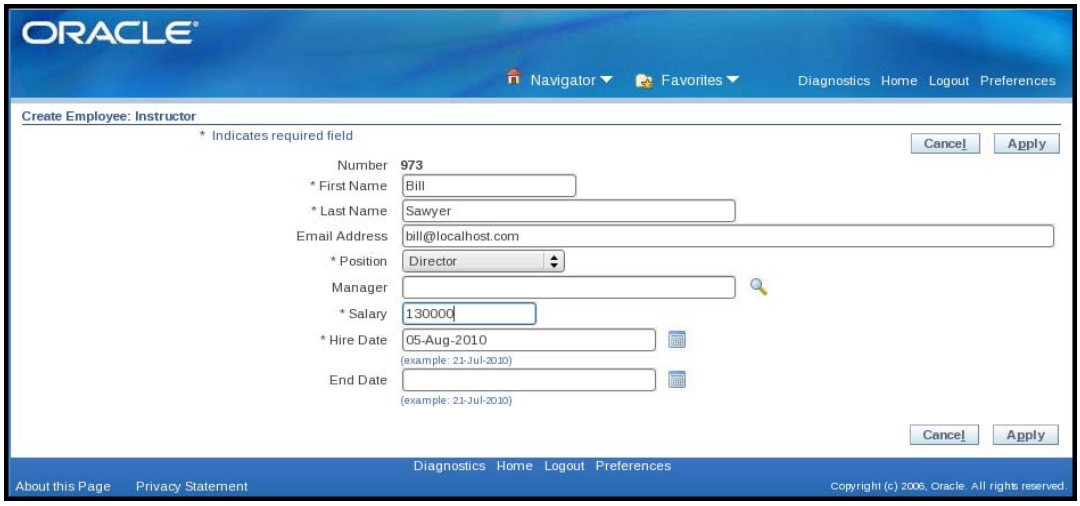
Step 11: Validate and Set the LastName

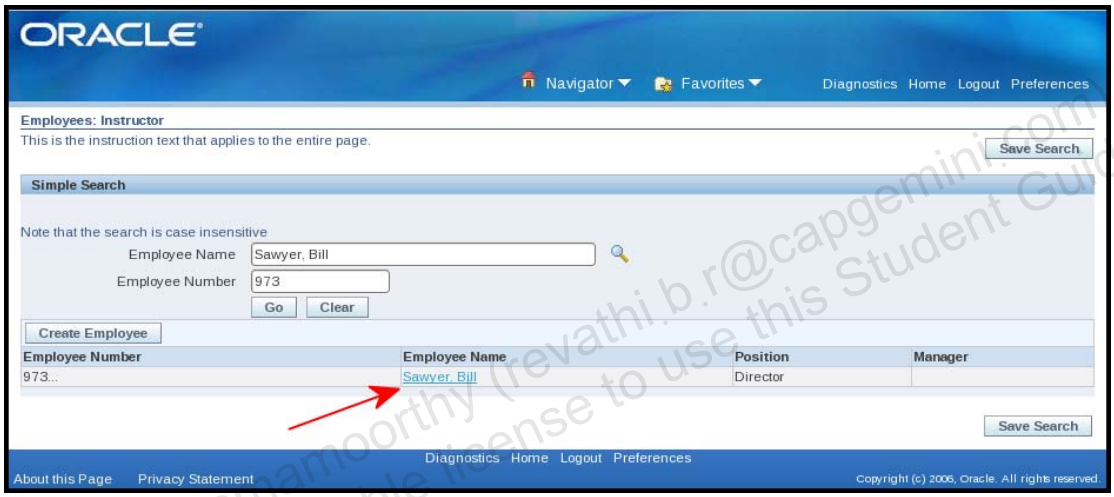
1. Add the following code to the **`setLastName()`** method in your **`EmployeeEOImpl`** class.

```
public void setLastName(String value) {
  // BC4J will make sure this value is not null. You still need
to
  // check before trying to set the full name.
  if ((value != null) || (!("".equals(value.trim())))) {
    String oldLastName = getLastName();
    if (oldLastName == null) {
      oldLastName = "";
    }
    // If you have a new lastname value, update the full name.
    if (value.compareTo(oldLastName) != 0) {
      String firstName = getFirstName();
      if (firstName == null) {
        firstName = "";
      }
      setFullName(value.concat(", ").concat(firstName));
    }
  }
  setAttributeInternal(LASTNAME, value);
}
```

Step 12: Test Your Work

1. Save your work.
2. Rebuild **ClassProject** .
3. Run the **EmployeePG**, and click the **Create Employee** button.
4. Create an employee, apply the creation, and return to the EmployeePG page. Query your newly created employee. You should now see the Employee Name handled correctly in the results table of your query page.





5. If you want to see how the new code interacts with your entire page, set the following breakpoints on the **first executable line** of these methods within the **EmployeeEOImpl** as follows:
   - `create()` **method**
   - `setFirstName()` **method**
   - `setLastName()` **method**
   - **setStartDate() method**
6. Run the **EmployeePG** in debug mode, and click the **Create Employee** button. Enter values for the fields, and click the **Apply** button.
7. As the breakpoints are encountered, verify that all of the methods are behaving as expected.

Step 13: Set the EndDate

These methods are virtually identical to the `setStartDate()` and `validateStartDate()` methods that you implemented earlier.

1. Add the following logic to the `setEndDate()` method in your `EmployeeEOImpl` class.

```
public void setEndDate(Date value) {
```

```
    validateEndDate(value);
    setAttributeInternal(ENDDATE, value);
  }
```

Step 14: Validate the EndDate

1. Add the following **validateEndDate()** method to your **EmployeeEOImpl** class.

```
  protected void validateEndDate(Date value) {
    // If a value has been set, validate it.
    if (value != null) {
      OADBTransaction transaction = getOADBTransaction();
      long sysdate =
transaction.getCurrentDBDate().dateValue().getTime();
      long endDate = value.dateValue().getTime();
      if (endDate < sysdate) {
        throw new
OAAttrValException(OAException.TYP_ENTITY_OBJECT,
                                   getEntityDef().getFullName(),
                                   getPrimaryKey(),
                                   "EndDate", value, "AK",
                                   "FWK_TBX_T_END_DATE_PAST");
      }
    }
  }
```

Step 15: Validate and Set the Salary

This method simply verifies that the salary value is > 0. It demonstrates how to correctly compare `oracle.jbo.domain.Number` values.

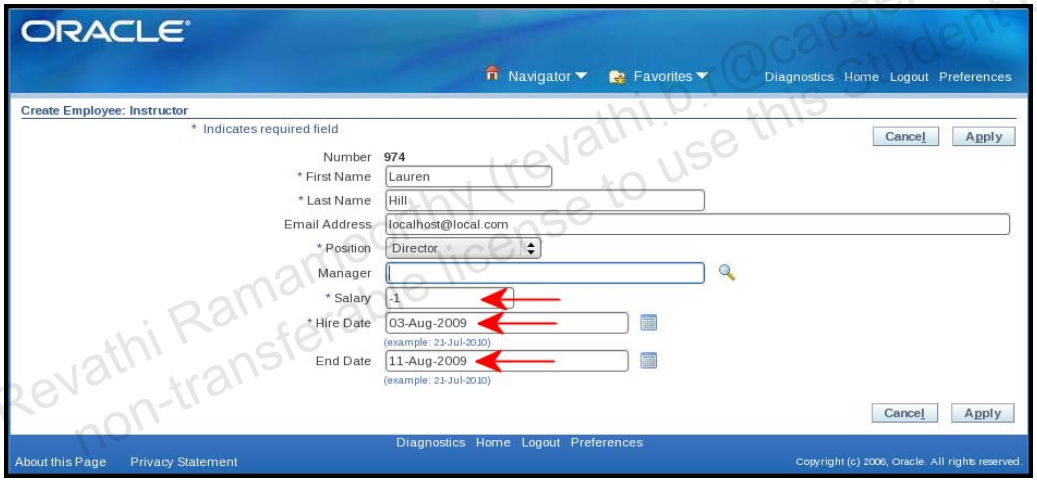
1. Add the following logic to the **setSalary()** method in your **EmployeeEOImpl** class.

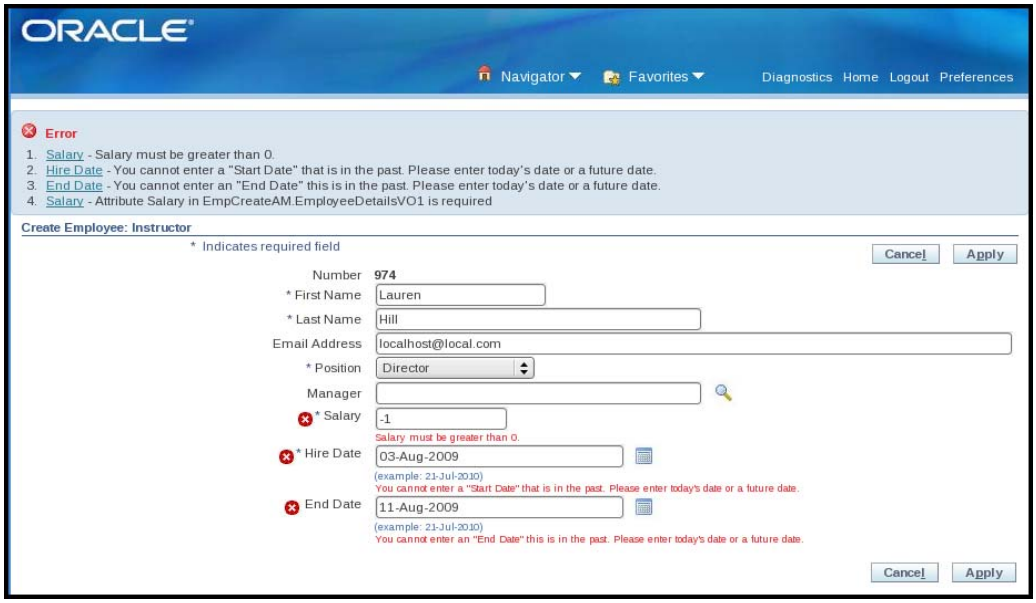
```
  public void setSalary(Number value) {
    if (value != null) {
      // Verify value is > 0
      if (value.compareTo(0) <= 0) {
        throw new
OAAttrValException(OAException.TYP_ENTITY_OBJECT,
                                   getEntityDef().getFullName(),
                                   getPrimaryKey(), "Salary",
                                   value, "AK",

"FWK_TBX_T_EMP_SALARY_REQUIRED");
```

```
        }
        setAttributeInternal(SALARY, value);
      }
    }
```

Step 16: Test Your Work

1. Save your work.
2. Rebuild **ClassProject** .
3. Run the **EmployeePG**, and click the **Create Employee** button.
4. You have added validations for EmployeeId, FirstName, LastName, StartDate, EndDate, and Salary. Of those, StartDate, EndDate, and Salary will throw exceptions if the value is not valid. EmployeeId will throw an exception, but since the value is based on a database sequence, the error should never occur.
5. Enter a StartDate and EndDate prior to the sysdate. The sysdate defaults into the StartDate field. So, simply change the value to be earlier than the default setting.
6. Enter a Salary of -1.
7. Click the **Apply** button.





Step 17: Validate and Set the Position (Cross-Attribute)

Position is unique to our validation as it has multiple conditions.

Condition 1: The position is President. If so, there is no need for a Manager to be set.

Condition 2: The position is not President, If so, both the ManagerName and the ManagerId must be set. And, the ManagerId must be a valid EmployeeId.

All cross-attribute validation must be added to the entity level `validateEntity()` method in your entity object. You cannot control the sequence in which BC4J calls individual attribute setters, so any validation that you add to a `set<AttributeName>()` method can refer only to that single attribute's value. When values are posted to the middle tier, BC4J calls each of the appropriate attribute setters, and then calls the `validateEntity()` method for any changed entities. So, the logic always executes after all your attribute setters have been called.

1. Add the following logic to the **`validateEntity()`** method in your **Employee`EOImpl`** class.

```
protected void validateEntity() {
  // Always call this first.
  super.validateEntity();
  // Check to see if the ManagerId is specified, unless the
  // employee has the Position of President; in which case, it
  // is OK for the ManagerId to be null.
  if (getManagerId() == null) {
    if (!("PRESIDENT".equals(getPositionCode()))) {
      throw new
OAAttrValException(OAException.TYP_ENTITY_OBJECT,
                                    getEntityDef().getFullName(),
                                    getPrimaryKey(),
                                    "ManagerId", getManagerId(),
                                    "AK",
```

```
"FWK_TBX_T_EMP_MANAGER_REQUIRED");
    }
  }
}
```

Step 18: Validate StartDate with EndDate (Cross-Attribute)

1. Add the following logic to the **validateEntity()** method in your **EmployeeEOImpl** class, at the end of the method.

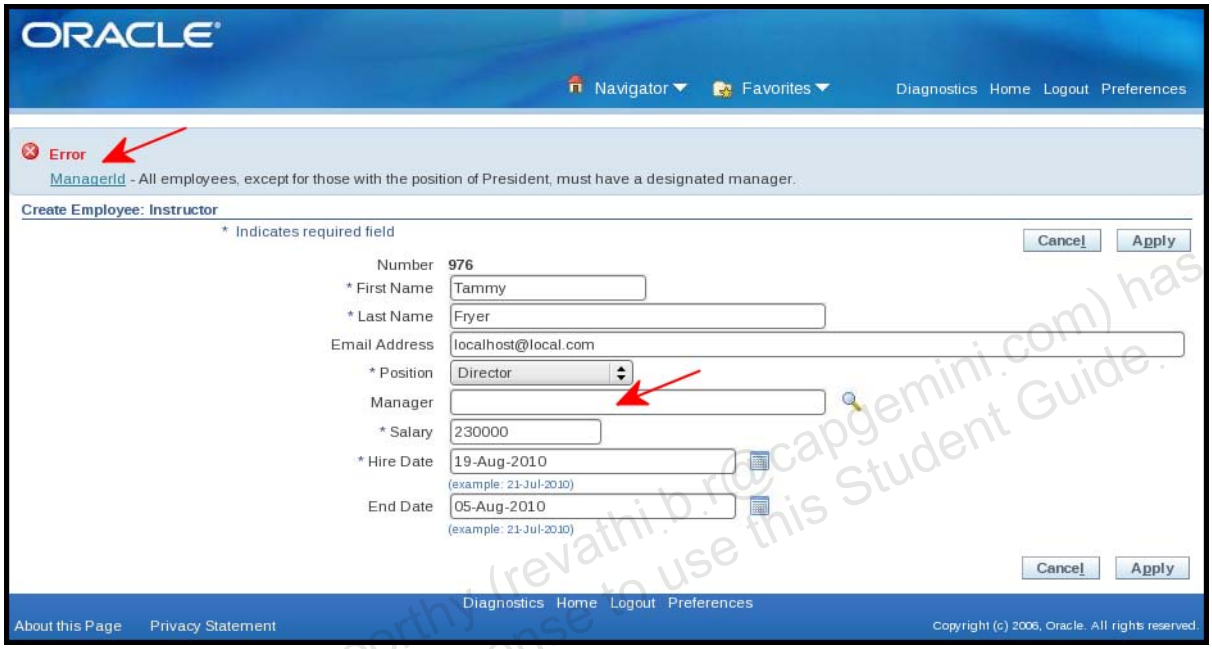
```
// Validate the StateDate and EndDate
Date startDate = getStartDate();
Date endDate = getEndDate();
// The StartDate can only be entered when object is new. If you
// call this validation when updating row, it will fail. So,
check
// the status. You validate here instead of within attribute to
// ensure both values are set before performing the
// cross-attribute test.
if (getEntityState() == STATUS_NEW) {
  validateStartDate(startDate);
}
validateEndDate(endDate);
if (endDate != null) {
  long endDateLong = endDate.dateValue().getTime();
  if (endDateLong < startDate.dateValue().getTime()) {
    throw new
OARowValException(OARowValException.TYP_ENTITY_OBJECT,
                                  getEntityDef().getFullName(),
                                  getPrimaryKey(),
                                  "AK",
                                  "FWK_TBX_T_START_END_BAD");
  }
}
```

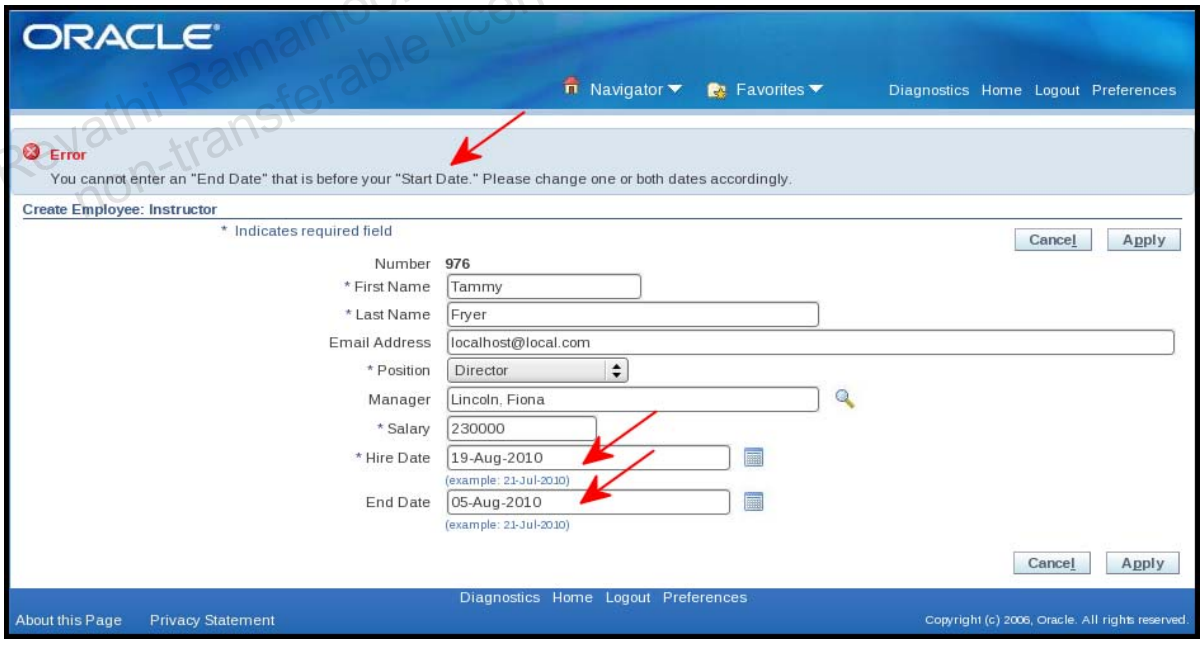
Step 19: Test Your Work

1. Save your work.
2. Rebuild **ClassProject** .
3. Run the **EmployeePG**, and click the **Create Employee** button.
4. Enter data for your test employee.
5. Click and choose a **Position** other than President.
6. Leave the **Manager** field empty. Do not select a Manager.

7. Set the **Start Date** to the sysdate (current date) plus 10 days.
8. Set the **End Date** to the sysdate (current date). By setting the Start Date to the sysdate plus 10, you enable the cross-attribute StartDate/EndDate exception to be tested.

   Note: Why does the ManagerId error throw first, and not the Start/End Date? Since the checks are both in the same method, once the exception for ManagerId is thrown, the method ceases to execute. To test and both exceptions, correct the ManagerId exception by changing the position to President, and then clicking the **Apply** button again.





Task 10: Creating Entity-Layer Validation Objects

In this task, you will create a Validation Application Module (VAM), and two Validation View Objects (VVO) to be used in `ManagerId` and `PositionCode` validation. You will also create an EntityExpert. You will use these Entity-Layer Validation Objects to accomplish more complex validations than possible with just the attribute setters or validateEntity methods. In each of these cases, you have to execute some SQL statements in order to implement the required business rules.

Step 1: Create Your Validation Application Module (VAM)

Create an application module to hold validation view objects used by your `EmployeesEO` and its entity expert. The validation application module is not used by the UI, so you create the AM in the same package as your entity object (the schema package).

1. Create a new Application module, in the package **`<student#>.oracle.apps.ak.schema.server`**.
2. Name the new application module, **Emplo**yeeVAM.
3. On the Java page (Step 4 of 4); uncheck the **Generate Java File** checkbox. You will not be adding any code to this application module.

Step 2: Associate Your EmployeeVAM Validation Application Module (VAM)

You must associate the `EmployeeVAM` with `EmployeeEO` so you can use certain convenience methods in the `Entity Expert.`

Note: You will need this in a few steps.

1. Double-click **EmployeeEO** in the Navigator.
2. Click the **Custom Properties** category.
3. Create a new property with the following Name and Value
   Note: The case must match exactly.
   Name = **VAMDef**
   Value = **`<student#>.oracle.apps.ak.schema.server.EmployeeVAM`**
4. Click the **Add** button. Click the **Apply** button.
5. Click the **OK** button.

Step 3: Create Your Position Validation View Object (VVO)

Validation View Objects (VVO) encapsulate simple validation queries that you would otherwise write directly in your entity object or expert. You create a VVO just like any other VO. The PositionVVO you are about to create, simply checks if the given position exists in the database.

1. In the **<student#>.**oracle.apps.ak.schema.server package, create a new view object called **PositionVVO**.
2. On the SQL Statement page (Step 5 of 8), set the **Query Statement** as follows:

```
SELECT LOOKUP_CODE
FROM FWK_TBX_LOOKUP_CODES_B
WHERE LOOKUP_CODE = :1
AND LOOKUP_TYPE = 'FWK_TBX_POSITIONS'
```

3. On the Java page (Step 7 of 8), check to generate both the **PositionVVOImpl** and **PositionVVORowImpl** for this VVO.

Step 4: Create Your Employee Validation View Object (VVO)

This validation view object looks for active employees. For simplicity, this is defined as any employee without an end date regardless of what the end date is.

1. In the **<student#>.oracle.apps.ak.schema.server** package, create a new view object called **EmployeeVVO**.
2. On the SQL Statement page (Step 5 of 8), set the **Query Statement** as follows:

```
SELECT EMPLOYEE_ID
FROM FWK_TBX_EMPLOYEES
WHERE EMPLOYEE_ID = :1
AND END_DATE IS NULL
```

3. On the Java page (Step 7 of 8), check to generate both the **EmployeeVVOImpl** and **EmployeeVVORowImpl** for this VVO.

Step 5: Add Your VVOs to Your VAM

4. Add **PositionVVO** to **EmployeeVAM**.
5. Add **EmployeeVVO** to **EmployeeVAM**.

Step 6: Add an initQuery( ) Method to Your VVOs

The validation view object must be able to set a where clause and execute its own query.

1. Add the following simple **initQuery()** method to the **PositionVVOImpl** class.
   Note: Be very careful to add this to the **PositionVVO** and not the PositionsVO.

```
public void initQuery(String positionCode) {
  setWhereClauseParams(null); // Always reset
  setWhereClauseParam(0, positionCode);
  executeQuery();
}
```

2. Add the following simple **initQuery()** method to the **EmployeeVVOImpl** class.
   Note: Be very careful to add this to the **EmployeeVVO**.

```
import oracle.jbo.domain.Number;
...
public void initQuery(Number employeeId) {
  setWhereClauseParams(null); // Always reset
  setWhereClauseParam(0, employeeId);
  executeQuery();
}
```

Step 7: Create Your Entity Expert

The entity expert class will handle validation of `ManagerId` and `PositionCode`. In each of these cases, the expert class delegates to the validation view objects (VVOs) to execute some SQL statements that implement the required business rules.

Entity experts are singletons (meaning there is one instance that is global to the JVM and shared by all users) that are closely affiliated with entity objects. They can be used to perform internal work on behalf of the entity object, and they can be used for lightweight operations by other classes instead of requiring multiple entity object instances.

1. Right-click **ClassProject** in the Navigator, and click **New** > **General** > **Java Class** from the New Gallery.
2. Click the **OK** button.
3. In the Create Java Class dialog window, enter the following:

   Name = **EmployeeEntityExpert**

   Package = **`<student#>.oracle.apps.ak.schema.server`**

   Extends = **`oracle.apps.fnd.framework.server.OAEntityExpert`**

   Note: the Wizard may look ahead for choices and might seem to hang. Click on any location to have the look ahead disappear.
4. Uncheck the **Generate Default Constructor** checkbox.
5. Click the **OK** button.

Step 8: Add isEmployeeActive( ) Method to Your Entity Expert

1. Add the following import **and isEmployeeActive()** method to your **`EmployeeEntityExpert`** class.

   Tip: If you fail to associate the `VAM` with the `EO`, the `findValidationViewObject()` method below will return null, and you will see an error message like, "**Could not resolve attribute set method...**" when you try to save a new employee.

```
import oracle.jbo.domain.Number;
import oracle.apps.fnd.framework.server.OAEntityExpert;
...
/*

*****************************************************************
*
 * Returns true if the given employee is currently active and
 * false if not.

*****************************************************************
*
 */
public boolean isEmployeeActive(Number employeeId) {
  boolean isActive = false;
  EmployeeVVOImpl empVO =
(EmployeeVVOImpl)findValidationViewObject("EmployeeVVO1");
```

```
    empVO.initQuery(employeeId);

    // Just doing a simple existence check. If you don't find a
    // match, return false.
    if (empVO.hasNext()) {
      isActive = true;
    }
    return isActive;
  } // end isEmployeeActive()
```

Step 9: Add isPositionValid( ) Method to Your Entity Expert

1. Add the following **isPositionValid()** method to your **`EmployeeEntityExpert`** class.

```
  /*

  ********************************************************************
  *
   * Returns true if the given position is valid.

  ********************************************************************
  *
   */
  public boolean isPositionValid(String position) {
    boolean isActive = false;
    PositionVVOImpl positionVO =
  (PositionVVOImpl)findValidationViewObject("PositionVVO1");
    positionVO.initQuery(position);

    // Just do a simple existence check. If you don't find a
    // match, return false.
    if (positionVO.hasNext()) {
      isActive = true;
    }
    return isActive;
  } // end isPositionValid()
```

Step 10: Associate Your Entity Expert with Your EO

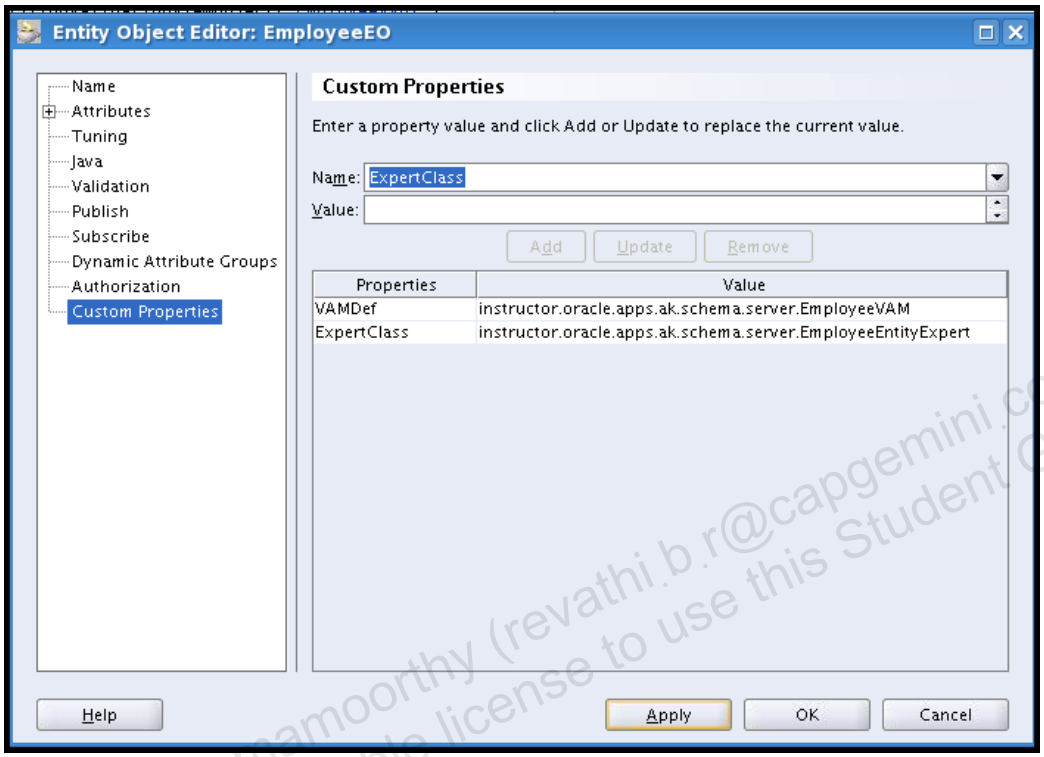
Entity expert classes are officially registered with their owning entity object.

1. Doublie-click the **EmployeeEO** to open the Entity Object Editor.
2. Click the **Custom Properties** category.
3. Create a new property with the following Name and Value (note that the case must match exactly):

Name = **ExpertClass**

Value = **`<student#>.oracle.apps.ak.schema.server.EmployeeEntityExpert`**

4. Click the **Add** button.
5. Click the **Apply** button.
6. Click the **OK** button.



Step 11: Add a Convenience Method to Your Entity Object

Since the Entity Expert is owned by your Entity Object (EO), you should create a method in the EO to allow any associated Entity Expert to be returned.

1. Add the following **EmployeeEntityExpert()** static method to your **`EmployeeEOImpl`** class.

   Note: This is a static method so it can be called without the client instantiating the `EOImpl` class.

```
/*
 * Convenience method returns the EmployeeEntityExpert.
 */
public static EmployeeEntityExpert
getEmployeeEntityExpert(OADBTransaction txn) {
  return
(EmployeeEntityExpert)txn.getExpert(EmployeeEOImpl.getDefinition
Object());
} // end getEmployeeEntityExpert()
```

Step 12: Add Validation Logic to Your setManagerId() Method

1. Add the following code to the generated **setManagerId()** method in your **EmployeeEOImpl** class. It calls the entity expert `isEmployeeActive()` method, and if this call returns false, it throws an attribute-level exception.

```
public void setManagerId(Number value) {
  if (value != null) {
    EmployeeEntityExpert expert =
getEmployeeEntityExpert(getOADBTransaction());
    if (!(expert.isEmployeeActive(value))) {
      throw new
OAAttrValException(OAException.TYP_ENTITY_OBJECT,
                                    getEntityDef().getFullName(),
                                    getPrimaryKey(),
                                    "ManagerId", value, "AK",

"FWK_TBX_T_EMP_MGR_INACTIVE");
    }
  }
  setAttributeInternal(MANAGERID, value);
} // end setManagerId()
```

Step 13: Add Validation Logic to Your setPositionCode Method

1. Add the following code to the generated **setPositionCode()** method code in your **EmployeeEOImpl** class. It calls the entity expert `isPositionValid ()` method, and if this call returns false, it throws an attribute-level exception.

```
public void setPositionCode(String value) {
  if ((value != null) || (!("".equals(value.trim())))) {
    EmployeeEntityExpert expert =
getEmployeeEntityExpert(getOADBTransaction());
    if (!(expert.isPositionValid(value))) {
      throw new
OAAttrValException(OAException.TYP_ENTITY_OBJECT,
                                    getEntityDef().getFullName(),
                                    getPrimaryKey(),
                                    "PositionCode",
                                    value, "AK",

"FWK_TBX_T_EMP_POSITION_INVALID");
    }
  }
  setAttributeInternal(POSITIONCODE, value);
```

```
    } // end setPositionCode()
```

Step 14: Test Your Work

1. Save your work.
2. Rebuild **ClassProject** .
3. Optional: Add the following breakpoints on the **first executable line** of code in **EmployeeEOImpl** and in **EmployeeEntityExpert** as follows:
   setManagerId() method
   setPositionCode() method
   isEmployeeActive() method
   isPositionValid() method
4. Debug or run **EmployeePG**.
5. Enter a First Name, Last Name, set the Position to anything other than President, click a Manager from the LOV or click from the Look Ahead, enter a Salary, and click the **Apply** button.
6. Verify that all your methods are called as expected.
7. You should still be able to create a new employee.

Part 3: Partial Page Rendering Overview

In this lab, you will modify the Create Employee page to implement a dynamic user interface using partial page rendering (PPR). If the Position poplist value is null or PRESIDENT, the Manager field does not display. If it is set to any other value, the Manager field displays and is required.

After completing this part of the lab, you will have:

- Implemented a PPR client action on a poplist.

Task 11: Create Your Model-layer Components

To enable PPR in this page, you must create an OA Framework standard applications properties view object (PVO). You will include a transient attribute for tracking the state of the `Rendered` property for the Manager field. This view object will ultimately include a single row which you will initialize and update as the user interacts with the page. You will see how the Manager field binds to this view object attribute to determine whether it should be rendered or not, and you will configure the Position poplist to fire a PPR event when the value changes.

Note: See the Dynamic User Interface topic in the OA Framework Developer's Guide for additional information about leveraging PPR in your application.

The PVO should be created in the same package as your page's AM. You should have only one PVO per application module, but it can include multiple attributes used by various pages that share the same root AM. This PVO should follow the naming convention: <AssociatedApplicationModuleName>PVO. Since you will be associating your application properties view object with your EmpCreateAM, it should be named EmpCreatePVO.

Step 1: Create Your PVO

1. In the **<student#>.oracle.apps.ak.employee.server** package, create a VO named **EmpCreatePVO**.
2. Select the **Rows Populated Programmatically, not Based on a Query** option.

3. Click the **Next** button
4. The EmpCreatePVO has no associated EOs.
5. On the Attributes page, click the **New...** button, and create an attribute with the following properties:

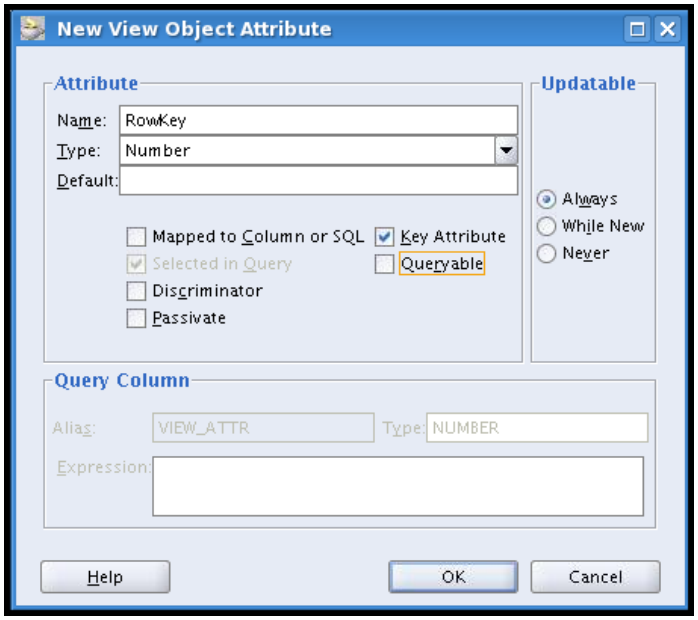
   Name = **RowKey**

   Type = **Number**

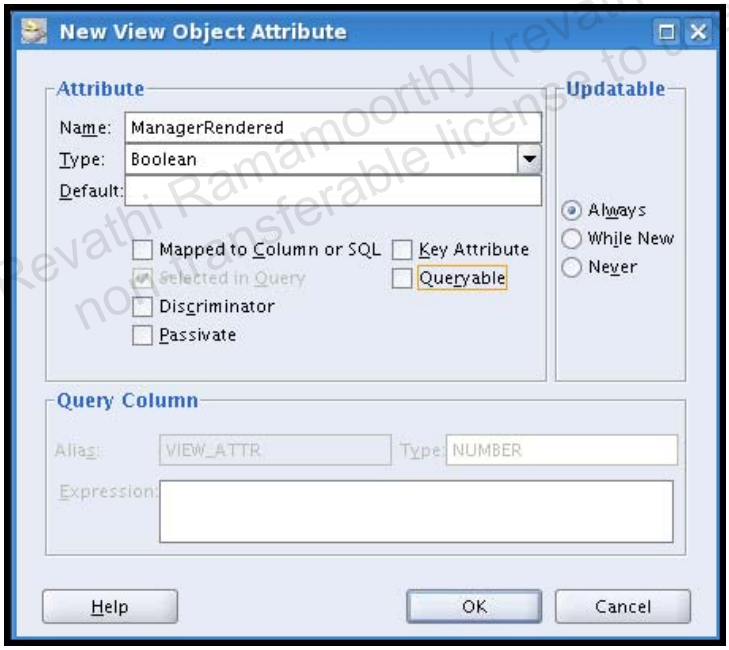
   Select the **Updateable: Always** option.

   Check the **Key Attribute** checkbox.

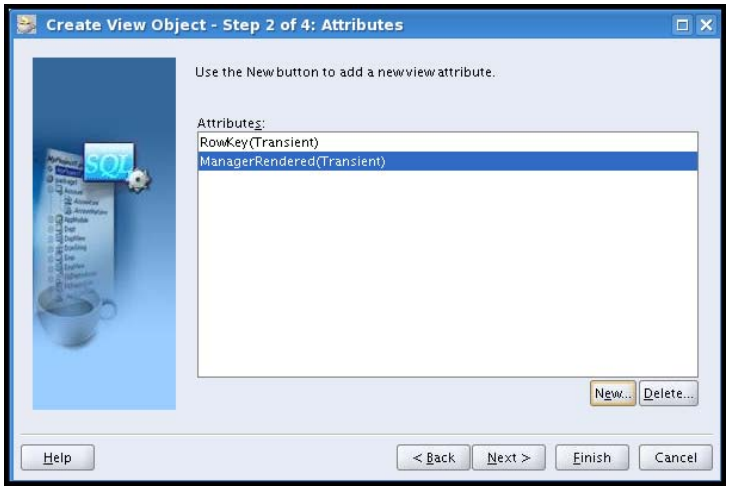
   Uncheck the **Queryable** checkbox.
6. Click the **OK** button.



7. Click the **New...** button again and create an attribute with the following properties:
   Name = **ManagerRendered**
   Type = **Boolean**
   Select the **Updateable: Always** option.
   Uncheck the **Queryable** checkbox.



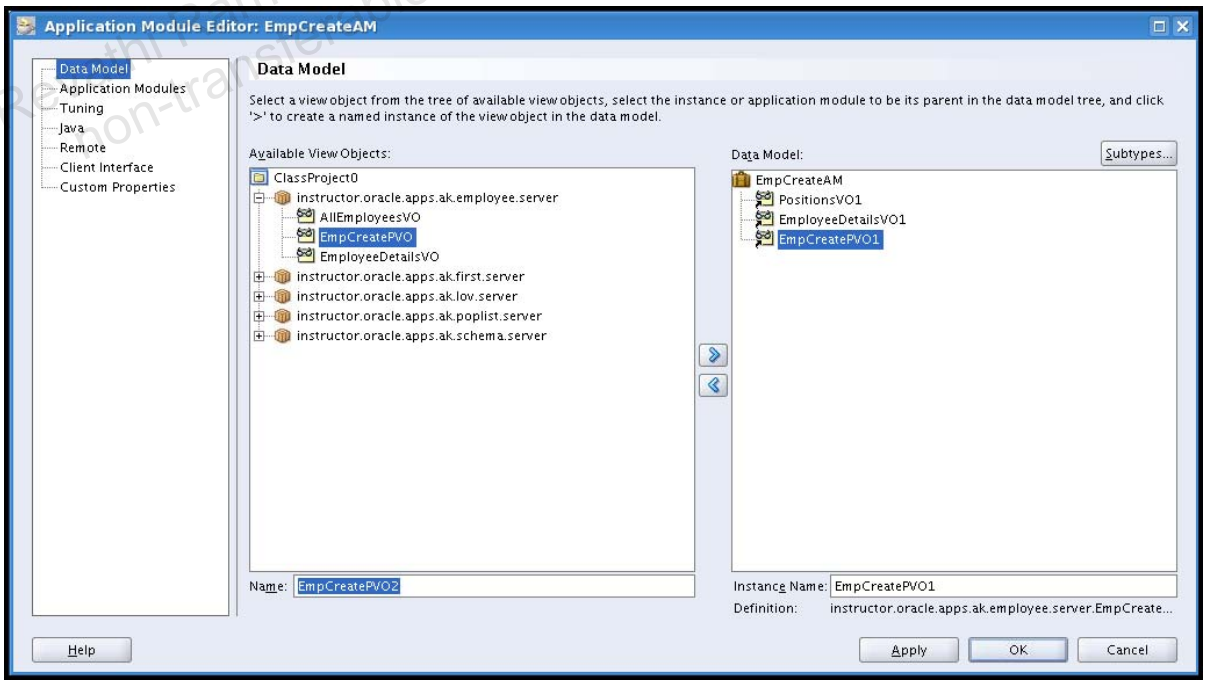
8. Click the **OK** button.



9. Click the **Next** button until you get to the Java page. Do not generate any Java files for this PVO, not even the EmpCreatePVORowImpl. A PVO is a special case of a VO. Uncheck any checkboxes in the Java step of the wizard.
10. Click the **Finish** button.
11. Double-click the **EmpCreatePVO** to open the View Object Editor.
12. Click the **Tuning** category.
13. Check the **Including All Transient Values** checkbox in the **Passivate State** category.
    Note: It is important that the transient attributes in a PVO are properly passivated.
14. Click the **Apply** button.
15. Click the **OK** button.

Step 2: Add Your EmpCreatePVO to Your EmpCreateAM

1. Add the **EmpCreate**PVO view object to **EmpCreateAM**.



Task 12: Configure Your View-layer Components

Step 1: Change Your PositionCode Item on the EmpCreatePG

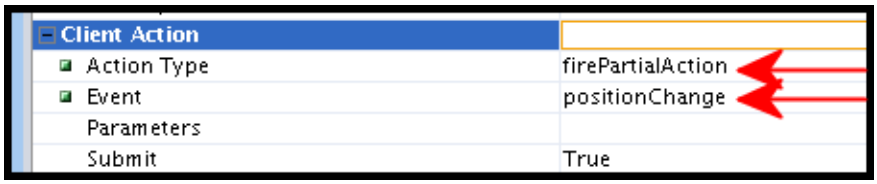
1. Set the following properties for the **PositionCode** item on your **EmpCreatePG** page.

   Client Action - Action Type = **firePartialAction**

   Enables a PPR event for this item.

   Note: There are three action types, none, firePartialAction, and fireAction. None is the default. firePartialAction is used for PPR. Why? This is being done somewhere on the form. You do not want the entire form to submit. You do not want to validate all the attributes. You simply want a subset of actions to fire, namely the PPR-related actions. This is why for PPR, you click firePartialAction as your Action Type value.

   Client Action – Event = **positionChange**



   Name of PPR event added to request when the item value changes

   Disable Server Side Validation = **True**

   Do not perform any server side validation when this item changes, and the form is submitted. For example, ignore invalid field values. Why? The VVOs and the EntityExpert are going to handle the validation.

   Disable Client Side Validation = **True**

   Do not perform any client side JavaScript validation when this item changes, and the form is submitted. For example, ignore null required field values.

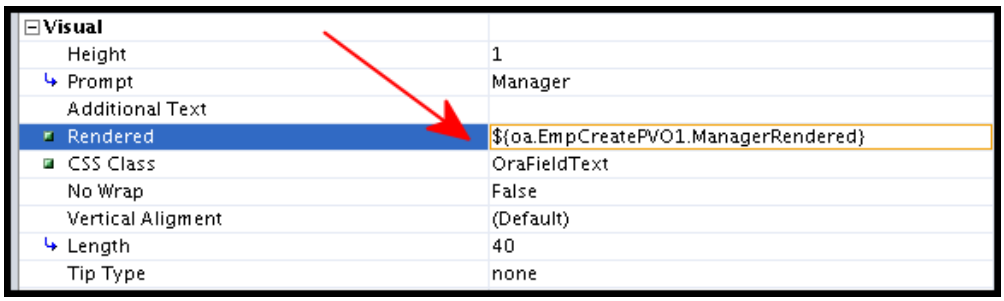
Step 2: Change Your ManagerName Item

1. Set the following properties for the ManagerName item on your EmpCreatePG page under the Visual section:

   Note: Simply type the value into the property, even though there is a True/False drop-down value. Then, press the Enter key to get the value to stick in the property. If you use the mouse to click directly out of the property after typing the value, the property resets to True. This type of expression is known as SPEL (simplest possible expression language), and is the UI syntax that enables PPR.

   Rendered = **${oa.EmpCreatePVO1.ManagerRendered}**

   At runtime, this item will render according to the value of the ManagerRendered attribute in the CreatePVO1 view object.



Task 13: Implement the Programmatic Elements

Step 1: Add to EmpCreateAM the handlePositionChange() Method

You've created the PVO, configured the PositionCode item to fire a PPR event, and configured the ManagerName item's `Rendered` property to bind to the `PVO`'s ManagerRendered attribute. Now, you need to add code to handle the PPR change events and initialize the `PVO`.

Your controller code will invoke the method whenever it's PPR event is fired.

1. Add the following code to your **EmpCreate`AMImpl`** class. This checks the value of the item's associated EmployeeDetailsVO attribute. If the value is null or PRESIDENT, the ManagerName field should not be rendered. In this case, the ManagerRendered property should be set to `Boolean.FALSE`.

```
import oracle.jbo.domain.Number;
import oracle.apps.fnd.framework.OARow;
...
/*

*************************************************************
*
 * Handles changes of the position poplist to set the
application
 * properties VO value for PPR.

*************************************************************
*
 */
public void handlePositionChangeEvent() {
  // Get the special, single-row application properties and make
  // the first (only) row current.
  OAViewObject vo =
(OAViewObject)findViewObject("EmpCreatePVO1");
  OARow row = (OARow)vo.first();

  // Get the value of the view object attribute with the
position
  // code.
  OAViewObject detailsVO =
(OAViewObject)findViewObject("EmployeeDetailsVO1");
  OARow detailsRow = (OARow)detailsVO.getCurrentRow();
  String position =
(String)detailsRow.getAttribute("PositionCode");
  if ((position == null) || ("PRESIDENT".equals(position))) {
    row.setAttribute("ManagerRendered", Boolean.FALSE);
  } else {
    row.setAttribute("ManagerRendered", Boolean.TRUE);
```

```
    }
  } // end handlePositionChangeEvent()
```

Step 2: Add an init() Method to Your EmpCreateAMImpl

1. Add the **init()** method to your **EmpCreateAMImpl**. The code ensures that your EmpCreatePVO includes a single row. It sets the RowKey primary key value, and calls the `handlePositionChangeEvent()` method to initialize the ManagerRendered property based on whatever the item's default value is.

```
/*

*******************************************************************
*
 * Initializes the transient application properties VO.

*******************************************************************
*
 */
public void init() {
  OAViewObject appPropsVO =
(OAViewObject)findViewObject("EmpCreatePVO1");
  if (appPropsVO != null) {
    // Do not reinitialize the VO unless needed. Note that
    // this method call does not try to query the database for
    // VOs with no SELECT statement and only transient
attributes.
    if (appPropsVO.getFetchedRowCount() == 0) {
      // Setting the match fetch size to 0 for an in-memory VO
      // prevents it from trying to query rows.  Calling
      // executeQuery() ensures that rows aren't lost after
      // a commit in the transaction (BC4J known issue
      // workaround).
      appPropsVO.setMaxFetchSize(0);
      appPropsVO.executeQuery();
      // You must create and insert a row in the VO before you
      // can start setting properties.
      appPropsVO.insertRow(appPropsVO.createRow());
      // Set the primary key value for this single-rwo VO.
      OARow row = (OARow)appPropsVO.first();
      row.setAttribute("RowKey", new Number(1));
    }
  }
  // Initialize the application properties VO (and the UI) based
```

```
  // on the default employee position value set on the
underlying
  // object.
  handlePositionChangeEvent();
} // end init()
```

Step 3: Call Your EmpCreateAMImpl init( ) Method from Your Controller

1. Add the following code to your **EmpCreateCO's** `processRequest()` method after the call to `am.invokeMethod("createEmployee")`. This ensures that your application properties VO is properly initialized when the page renders.

```
// Initialize the PVO for PPR.
am.invokeMethod("init");
```

Step 4: Handle the PPR Event in Your Controller

You configured the PositionCode item to perform a form submit.

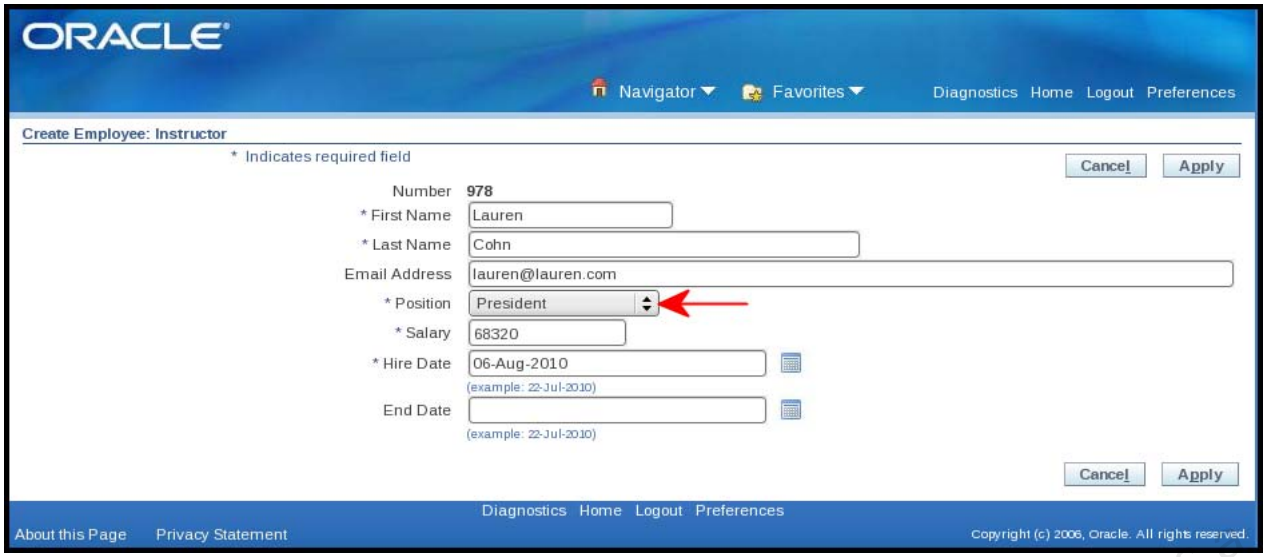
1. Add the following code to the end of your existing **EmpCreateCO's** `processFormRequest()` method to check to see if the item's PPR event has been fired. It is was fired, invoke the Create`AMImpl.handlePositionChangeEvent()` method to handle the change.

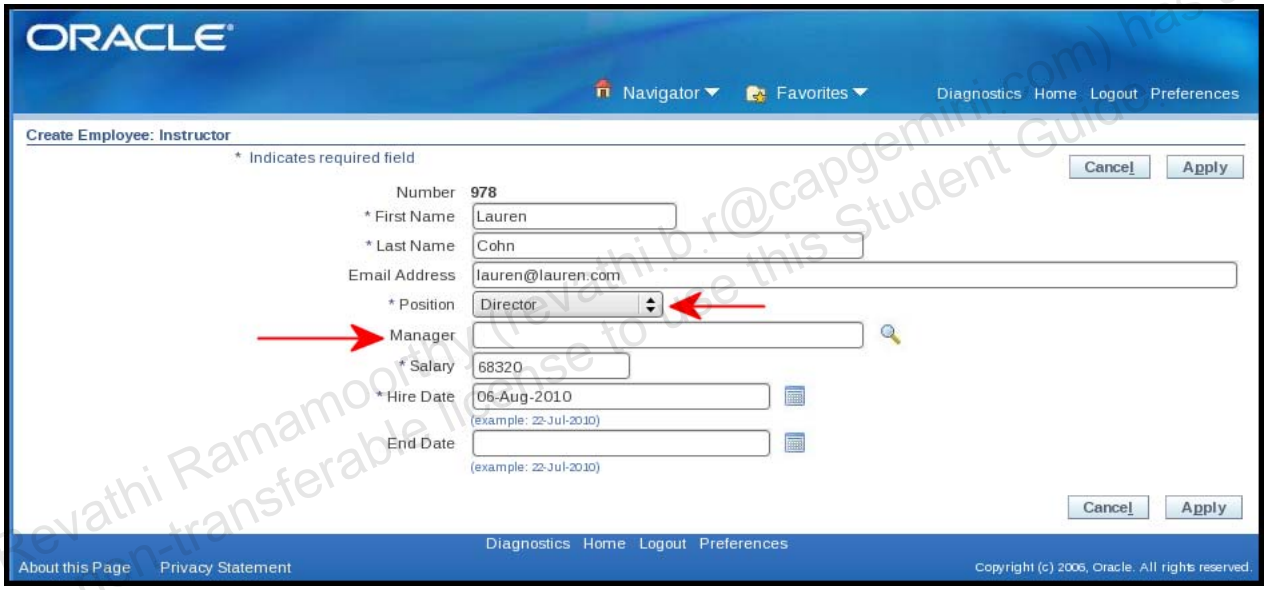
```
if
("positionChange".equals(pageContext.getParameter(OAWebBeanConst
ants.EVENT_PARAM))) {
  // The PositionCode PPR change event has fired.
  am.invokeMethod("handlePositionChangeEvent");
}
```

Step 5: Test Your Work

1. Save your work.
2. Rebuild **ClassProject** .
3. Run the **EmpCreatePG** from the EmployeePG, and test PPR by changing the Position from President to something else.

   Note: Pay attention to the changes in the UI as the PPR is fired. The Manager field will appear or disappear (Render) depending on the validation. The Validation being "If the position is President, there cannot be a Manager"





Lab - Implementing a Delete

Chapter 6

Overview

For this lab, you will be implementing the basic components of an Employee Delete page. This lab assumes, and relies upon, you having completed Lab: First OA Framework Page, Lab: Implementing a Query and Drill-down, and Lab: Implementing a Create. If you have not yet finished those labs, please do so before continuing with this lab.

After completing this lab, you should have learned how to:

- Configure an item (icon) to perform a form submit.
- Display and handle a Warning dialog message.
- Display an in-page confirmation message using a JSP "forward immediate."
- Delete a row in an entity object-based view object.
- Handle the delete action item selection.
- Iterate through view object rows looking for selected values.
- Implement a table content switcher.
- Control column alignment.

Task 1: Build Your Model-layer Components

Step 1: Add Attributes to Your AllEmployeesVO

You are implementing the delete function by adding capabilities to the existing EmployeePG. To add the needed capability, you need to add two attributes to the AllEmployeesVO, a delete switcher attribute and an employee status attribute.

The delete switcher attribute implements the following rules:

- If the employee is active, the delete trashcan icon is disabled.
- If the employee is inactive, the delete trashcan icon is enabled.

The employee status attribute implements the following rules:

- If the employee is active, a checkmark icon displays in the status column.
- If the employee is inactive, an X icon displays in the status column.

Note: For simplicity, the business rules assume that the presence of a non-null `END_DATE` implies that the employee is inactive, regardless of the `END_DATE` value.

1. Open **AllEmployeesVO** in the VO Editor.
2. Click the SQL Statement category.
3. Click the existing Query Statement and modify it as follows:

```
SELECT EmployeeEO.EMPLOYEE_ID,
       EmployeeEO.FIRST_NAME,
       EmployeeEO.LAST_NAME,
       EmployeeEO.FULL_NAME AS EMPLOYEE_NAME,
       EmployeeEO.EMAIL_ADDRESS AS EMPLOYEE_EMAIL,
       EmployeeEO1.EMPLOYEE_ID AS MANAGER_ID,
       EmployeeEO1.FULL_NAME AS MANAGER_NAME,
       EmployeeEO1.EMAIL_ADDRESS AS MANAGER_EMAIL,
       FwkLookupCode.MEANING AS POSITION_DISPLAY,
       decode(nvl(to_char(EmployeeEO.END_DATE), 'N'),
       'N', 'DeleteDisabled', 'DeleteEnabled') AS
DELETE_SWITCHER,
       decode(nvl(to_char(EmployeeEO.END_DATE), 'Y'),
       'Y', 'okind_status.gif', 'criticalind_status.gif') AS
       EMPLOYEE_STATUS
FROM   FWK_TBX_EMPLOYEES EmployeeEO,
       FWK_TBX_EMPLOYEES EmployeeEO1,
       FWK_TBX_LOOKUP_CODES_VL FwkLookupCode
WHERE  EmployeeEO.MANAGER_ID = EmployeeEO1.EMPLOYEE_ID (+)
AND EmployeeEO.POSITION_CODE = FwkLookupCode.LOOKUP_CODE
AND FwkLookupCode.LOOKUP_TYPE = 'FWK_TBX_POSITIONS'
```

Note: For clarity, modifications to the existing Query Statement are indicated in **BOLD** type. Additionally, it is easy to miss that a , (comma) has been added after the AS POSITION_DISPLAY line of the SELECT.

4. Click the **Attribute Mappings** category, and confirm that all of the attributes are mapped as expected. Verify that POSITION_DISPLAY, DELETE_SWITCHER and EMPLOYEE_STATUS are SQL View Attributes.
5. Click the **Apply** button, then the **OK** button.

Task 2: Build Your View-layer Components

Step 1: Create Your Switcher Region

Switchers can be used in tables to conditionally display alternate content in a column. In this task, you will configure a switcher to display an appropriate Delete image based on the employee's status. See Table Content Switchers in the OA Framework Developer's Guide for more information about this feature.

Create the switcher bean that will bind to the Delete Switcher view attribute you just created to determine which Delete icon to display.

1. Click the **EmployeePG** in the Applications Navigator pane, and then click the **ResultsRN** in the Structure pane.
2. Right-click **ResultsRN**, and click **New** > **switcher** from the context menu. Set the properties of **region1** as follows:

   ID = **DeleteSwitcherRN**

   Attribute Set = **`/oracle/apps/fnd/attributesets/Buttons/Delete`**

   View Instance = **AllEmployeesVO1**

   View Attribute = **DeleteSwitcher**

Step 2: Configure Your Switcher Cases

Switchers mimic a programmatic switch: you create a separate case for each option that might display. The switcher bean binds to the view object attribute that returns the name of the case to render. Each case that you add must have an `ID` value that matches one of your decode return value names.

Configure the first switcher case item under your DeleteSwitcherRN to show the disabled Delete image.

1. Click the **default <case>** switcher case in the Structure pane, right-click and click **New** > **Item** from the context menu. Set the **item1** properties as follows:

   ID = **DeleteDisabled**

   Item Style = **image**

   Image URI = **`deleteicon_disabled.gif`**

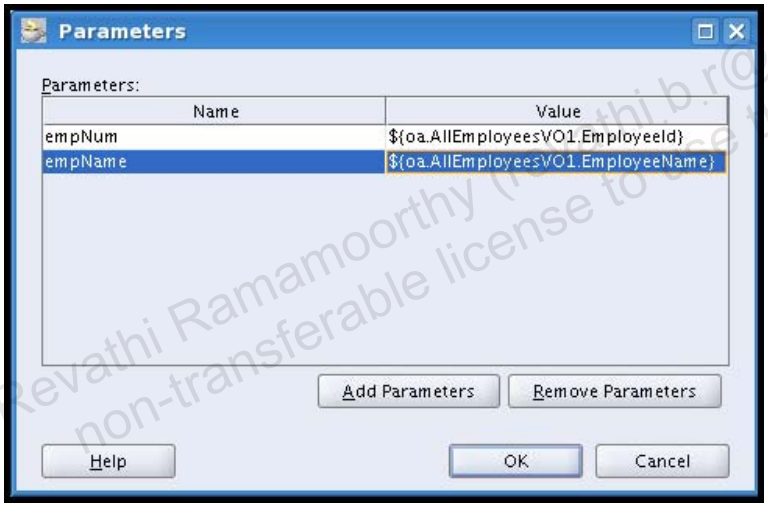
   Additional Text = **Delete employee action is disabled.**

   Height = **24**

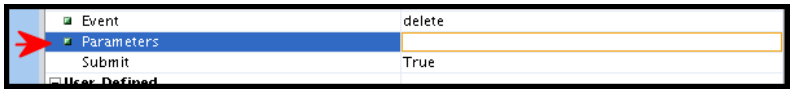
   Width = **24**

   Note: You must specify the `Height` and `Width` properties for any images that you add to a page, except for the branding image. To find the values to enter, look for your image in either the Oracle Browser Look and Feel UI Guideline Icon Repository or the Ancillary Graphic Repository. See the Adding Images to Your Pages document as part of the BLAF for additional information about icon creation and use procedures.
2. Right-click **DeleteSwitcherRN**, and click **New** > **case** from the context menu.

3. Right-click the **new <case>** switch case, and click **New** > **item** from the context menu. Set the **item1** properties as follows:
   ID = **DeleteEnabled**
   Item Style = **image**
   Image URI = **`deleteicon_enabled.gif`**
   Additional Text = **Delete employee action is enabled.**
   Height = **24**
   Width = **24**
   Action Type = **fireAction**
   Event = **delete**
4. Click the **(…)** button in the Parameter property, and do the following:
5. Click the **Add Parameters** button.
   Name = **empNum**
   Value = **${oa.AllEmployeesVO1.EmployeeId}**
6. Click the **Add Parameters** button.
   Name = **empName**
   Value = **${oa.AllEmployeesVO1.EmployeeName}**



7. Click the **OK** button
   Note: There is nothing that appears in the properties inspector to indicate there are parameters except for the green check-box that the default has changed.



8. You will use these parameters later to ensure that you delete only the employee that is selected for deletion. At runtime, OA Framework automatically creates and populates formParameter (hidden) items for the empName and empNum parameters you configure. These formParameter values are then added to the request.
9. When the user clicks the **Delete** icon, you submit the form (Post) so you can handle the event in the `processFormRequest()`. Normally, you would set a `Destination URI` property for an image. However, the selection results in an HTTP GET and not a POST. You want to perform a POST. To do this, you explicitly configure the image to perform a form submit when clicked.

Step 3: Configure Your Bound Value

A switcher is one approach for conditionally displaying alternatives. You can achieve the same result using a bound value, specifically `oracle.apps.fnd.framework.webui.OADataBoundValueViewObject` references an employee status attribute in the AllEmployeesVO view object.

In simple terms, a bound value lets you configure a data source for a component property. You have already configured bound values implicitly when you specified the `View Object Instance` and `View Object Attribute` property values for items in other labs. This task shows how to explicitly configure bound values for selected component properties.

Tip: Although this example shows how to use bound values to conditionally display alternate images in a table column, as a rule, you should use a switcher for this particular problem as they are fully declarative -- and therefore personalizable. Bound values, like any programmatic UI configuration, should be used only when necessary.

1. In the **EmployeePG**, right-click the **ResultsRN** in the Structure pane, and click **New** > **Item** from the context menu. Set **item1** properties as follows:

   ID = **EmpStatus**

   Item Style = **image**

   View Instance = **AllEmployeesVO1**

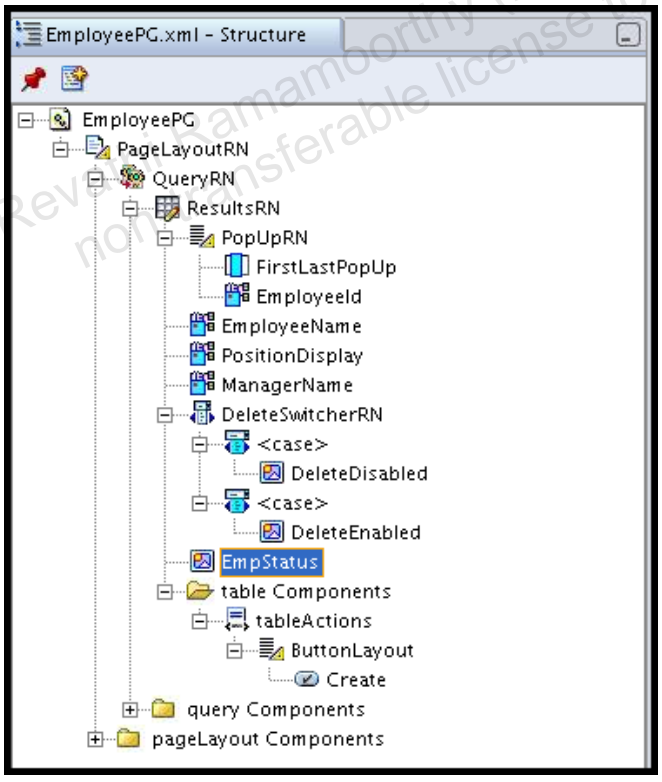
   View Attribute = **EmployeeStatus**

   Prompt = **Status**

   Additional Text = **Current Employment status**

   Height = **16**

   Width = **16**



Task 3: Write Your Model-layer Programmatic Elements

Step 1: Confirm Your EmployeeEO Method

1. Verify that you have the following `remove()` method in your **EmployeeEOImpl** class. If not, add it, compile, and save your change. This method implements the delete.

```
/*

*****************************************************************
 * Add entity delete logic here.

*****************************************************************
 */
public void remove() {
  super.remove();
} // end remove()
```

Step 2: Add a deleteEmployee( ) Method to Your EmployeesAM

1. Add a **deleteEmployee()** method to the **EmployeesAMImpl** class that takes an `employeeNumber` parameter and looks for a matching row in AllEmployeesVO view object.

   Note: Calling the `remove()` method on your AllEmployees`VORowImpl` class delegates to the `remove()` method in your `EmployeeEOImpl` class.

   Tip: This code illustrates how to iterate through the result set manually so you can see how this works, however, there are convenience methods in the `oracle.apps.fnd.framework.server.OAViewObjectImpl` class that let you quickly find one or more matching rows for given values. See the Oracle Application Framework Javadoc for additional information.

```
import oracle.jbo.domain.Number;
import oracle.jbo.RowSetIterator;
import oracle.apps.fnd.framework.OAViewObject;
...
/*

*****************************************************************
***
 * Deletes an employee.

*****************************************************************
***
 */
public void deleteEmployee(String employeeNumber) {
  // First, find the selected employee in your VO.
```

```
  // When you find it, call remove( ) on the row which in turn
  // calls remove on the associated EmployeeEOImpl object.
  int empToDelete = Integer.parseInt(employeeNumber);

  OAViewObject vo = (OAViewObject)getAllEmployeesVO1();
  AllEmployeesVORowImpl row = null;

  // This tells the number of rows that have been fetched in the
  // row set, and will not pull additional rows in like some of
the
  // other "get count" methods.
  int fetchedRowCount = vo.getFetchedRowCount();

  // Use a separate iterator, even though you could step through
the
  // rows without it, because you don't want to affect row
currency.
  RowSetIterator deleteIter =
vo.createRowSetIterator("deleteIter");
  if (fetchedRowCount > 0) {
    deleteIter.setRangeStart(0);
    deleteIter.setRangeSize(fetchedRowCount);
    for (int i = 0; i < fetchedRowCount; i++) {
      row =
(AllEmployeesVORowImpl)deleteIter.getRowAtRangeIndex(i);
      // For performance reasons, generate ViewRowImpls for all
      // VOs. When you need to obtain an attribute value,
      // use the named accessors instead of a generic String
      // lookup.

      Number primaryKey = row.getEmployeeId();
      if (primaryKey.compareTo(empToDelete) == 0) {
        // This performs the actual delete.
        row.remove();
        getTransaction().commit();
        break; // only one possible selected row in this case
      }
    }
  }
  // Always close the iterator when you're done.
  deleteIter.closeRowSetIterator();
} // end deleteEmployee
```

Task 4: Write Your Controller-layer Programmatic Elements

Step 1: Handle the UI Elements (GET) in

All image columns should be centered. Image columns are automatically centered, but switcher columns (even when displaying images) are not. Additionally, you need to bind the image name returned in the EmployeeStatus attribute with the OA Framework image directory.

1. To handle both of these conditions, add this logic to the end of the `processRequest()` method in the **EmpResultsCO** after the code you've previously added.

```
import oracle.cabo.ui.data.DictionaryData;
import oracle.cabo.ui.data.DataObjectList;
import oracle.apps.fnd.framework.webui.beans.table.OATableBean;
import oracle.cabo.ui.data.BoundValue;
import oracle.cabo.ui.data.bind.ConcatBoundValue;
import oracle.cabo.ui.data.bind.FixedBoundValue;
import oracle.apps.fnd.common.MessageToken;
import oracle.apps.fnd.framework.OAException;
import
oracle.apps.fnd.framework.webui.OADataBoundValueViewObject;
import oracle.apps.fnd.framework.webui.beans.OAImageBean;
...
// This controller is associated with the table.
OATableBean table = (OATableBean)webBean;

// You need to format the Switcher image column so the image
// is centered (this isn't done automatically for Switchers
// as it is for plain image columns). Start by getting the
// table's column formats.

// NOTE!!! You must call the prepareForRendering() method on
// the table *before* formatting columns. Furthermore, the
// call must be sequenced *after* the table is queried,
// and *after* you do any control bar manipulation.
table.prepareForRendering(pageContext);
DataObjectList columnFormats = table.getColumnFormats();
DictionaryData columnFormat = null;
int childIndex = pageContext.findChildIndex(table,
"DeleteSwitcherRN");
columnFormat =(DictionaryData)columnFormats.getItem(childIndex);
columnFormat.put(COLUMN_DATA_FORMAT_KEY, ICON_BUTTON_FORMAT);

// Implement the bound value for the Status Image
```

```
OAImageBean statusImageBean =
 (OAImageBean)table.findIndexedChildRecursive("EmpStatus");
if (statusImageBean == null) {
  MessageToken[] tokens = { new MessageToken("OBJECT_NAME",
"EmpStatus") };
  throw new OAException("AK", "FWK_TBX_OBJECT_NOT_FOUND", tokens);
}
// Define the OA Framework image directory
FixedBoundValue imageDirectory =
new FixedBoundValue(APPS_MEDIA_DIRECTORY);

// Define a binding between the image bean and the VO attribute
// that it will reference to get the .gif image value name.
// Note that the corresponding attribute values are obtained
using a
// decode() in the QueryVO view object.
OADataBoundValueViewObject statusBinding =
    new OADataBoundValueViewObject(statusImageBean,
"EmployeeStatus");

// Concatenate the image directory with the actual image name
// (as retrieved from the VO attribute decode() statement)
ConcatBoundValue statusCBV =
    new ConcatBoundValue(new BoundValue[] { imageDirectory,
                                            statusBinding });
// Tell the image bean where to get the image source attribute
statusImageBean.setAttributeValue(SOURCE_ATTR, statusCBV);

// For accessibility compliance, you specify the alternate text
// for an image.  Note you should never use static text as shown
// (always source translatable text from Message Dictionary when
// setting display text values programmatically), and ideally,
// the alternate text should in this case should clearly
indicate
// the status the image represents. Generally, we recommend that
// you use a Switcher as shown for the Delete column to easily
// show different images with associated alternate text, but we
// wanted to show how to use a bound value also in this lab.
statusImageBean.setAttributeValue(SHORT_DESC_ATTR,
                                  "Employee status indicator");
```

Step 2: Handle the Events (POST)

There are two events that you must handle:

- Event 1: The delete icon has been clicked, and a Warning dialog needs to be displayed.
- Event 2: The Warning dialog's OK button has been clicked, and the delete can be executed.

1. Add the following code to the end of your **`EmpResultsCO processFormRequest()`** method to handle these events.

   Note: See the Dialog Pages topic in the Oracle Application Framework Developer's Guide 12.1.2 or higher for additional information about this component.

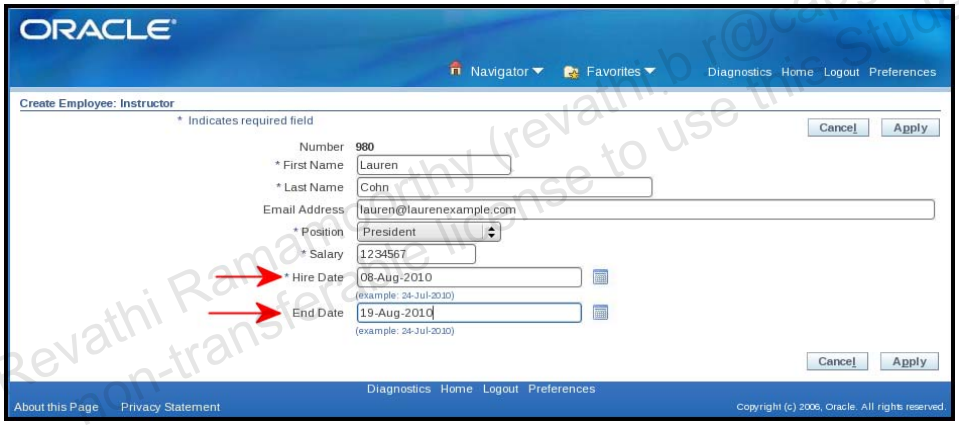
```
import oracle.apps.fnd.framework.webui.OADialogPage;
import java.io.Serializable;
...
if ("delete".equals(pageContext.getParameter(EVENT_PARAM))) {
  // The user has clicked a delete icon so display a Warning
  // dialog asking to confirm deleting the employee. Note you
  // configure the dialog so the "Yes" button submits to
  // this page so you handle the action in processFormRequest()

  String employeeNumber = pageContext.getParameter("empNum");
  String employeeName = pageContext.getParameter("empName");

  MessageToken[] tokens = { new MessageToken("EMP_NAME",
employeeName) };
  OAException mainMessage = new OAException("AK",
"FWK_TBX_T_EMP_DELETE_WARN", tokens);

  // Note even though you're going to make your Yes/No buttons
  // submit a form, you still need some non-null value in the
  // constructor's Yes/No URL parameters for the buttons to
render,
  // so just pass empty Strings for this.
  OADialogPage dialogPage = new
OADialogPage(OAException.WARNING, mainMessage, null, "", "");

  // Always use Message Dictionary for any Strings you want
  // to display.
  String yes = pageContext.getMessage("AK", "FWK_TBX_T_YES",
null);
  String no = pageContext.getMessage("AK", "FWK_TBX_T_NO",
null);

  // Set this value so the code that handles this button press
```

```
    // is descriptive.
    dialogPage.setOkButtonItemName("DeleteYesButton");

    // The following configures the Yes/No buttons to be submit
    // buttons, and handles the form submit in the originating
    // page (EmployeePG) so you can handle the "Yes" button
  selection
    // in this controller.
    dialogPage.setOkButtonToPost(true);
    dialogPage.setNoButtonToPost(true);
    dialogPage.setPostToCallingPage(true);

    // Set your Yes/No labels instead of the default OK/Cancel.
    dialogPage.setOkButtonLabel(yes);
    dialogPage.setNoButtonLabel(no);

    // You need to keep hold of the employeeNumber and
  employeeName.
    // The OADialogPage gives us a convenient means
    // of doing this. Note that the use of the Hashtable is
    // most appropriate for passing multiple parameters. See the
    // OADialogPage javadoc for an alternative when dealing with
    // a single parameter.
    java.util.Hashtable formParams = new java.util.Hashtable(1);
    formParams.put("empNum", employeeNumber);
    formParams.put("empName", employeeName);
    dialogPage.setFormParameters(formParams);

    pageContext.redirectToDialogPage(dialogPage);
  } else if (pageContext.getParameter("DeleteYesButton") != null)
  {

    // User has confirmed that they want to delete this
    // employee. Invoke a method on the AM to set the current
    // row in the VO and call remove() on this row.
    String employeeNumber = pageContext.getParameter("empNum");
    String employeeName = pageContext.getParameter("empName");
    Serializable[] parameters = { employeeNumber };
    OAApplicationModule am = pageContext.getApplicationModule(webBean);
    am.invokeMethod("deleteEmployee", parameters);

    // Now, redisplay the page with a confirmation message at
    // the top. Note that the deleteEmployee() method in the AM
```

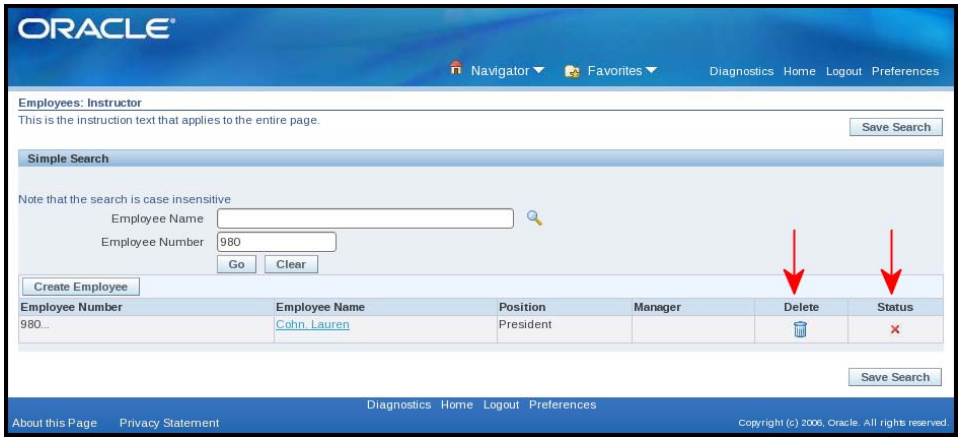
```
    // commits, and our code won't get this far if any exceptions
    // are thrown.

    MessageToken[] tokens = { new MessageToken("EMP_NAME",
employeeName) };
    OAException message = new OAException("AK",
"FWK_TBX_T_EMP_DELETE_CONFIRM", tokens,
OAException.CONFIRMATION, null);
    pageContext.putDialogMessage(message);
  }
```

Step 3: Test Your Work

1. Save your work.
2. Rebuild your **ClassProject**.
3. Run your **EmployeePG**.
4. Click the **Create Employee** button to create a new employee. Be sure to set the End Date value to any date >= today's date. You cannot test deleting an employee unless you create one with a non-null end date. Return to the Employee Query page.

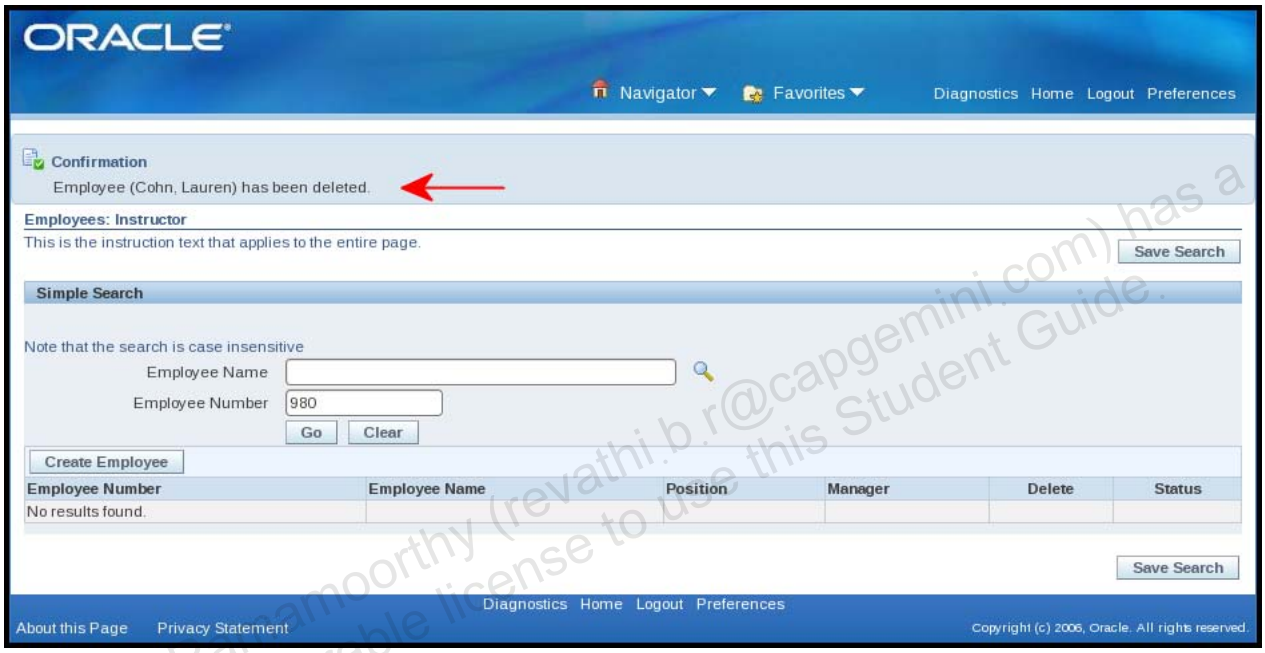


5. Query your new employee, and verify that the delete icon is enabled and the status image is shown correctly.

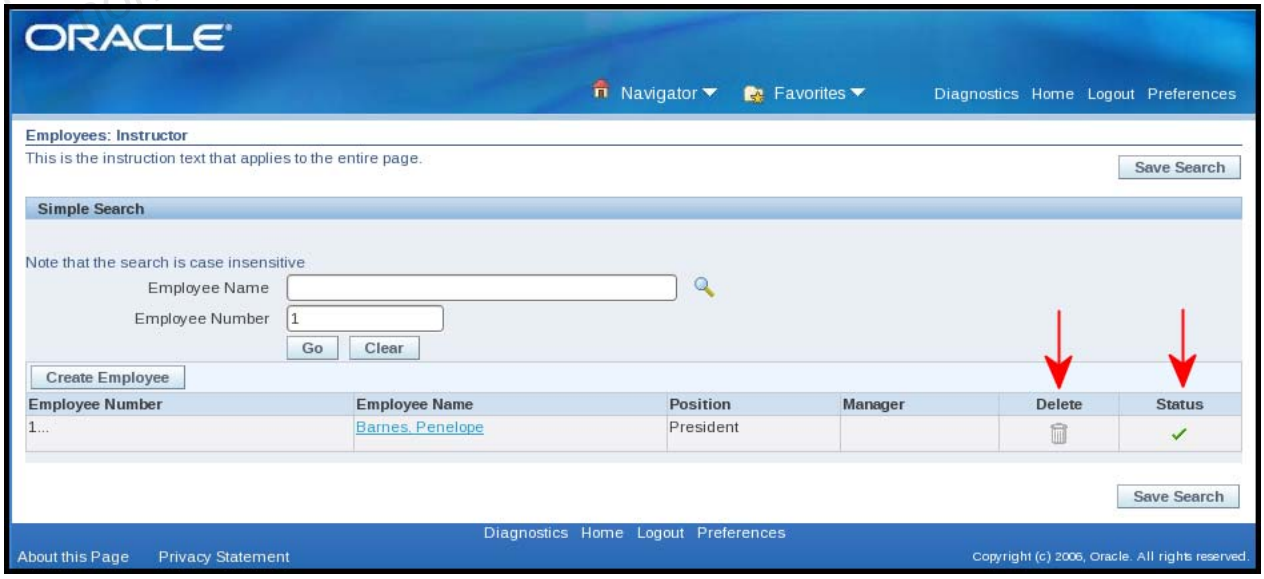


6. Click the **Delete** icon, and verify that the Warning dialog displays.

7. Click the **Yes** button in the Warning dialog to delete your employee. Verify that the confirmation displays.



8. To verify that the other case is working, query an existing Employee that doesn't have an end date. You will see that their Delete icon, the trash can, is grayed out, and their status has a green check icon.



Lab - Implementing an Update

Chapter 7

Overview

For this lab, you will be implementing the basic components of an Employee Update page. This lab assumes, and relies upon, you having completed all the previous labs. If you have not finished the previous labs, please do so before continuing with this lab.

After completing this lab, you will have:

- Implemented a simple single-page update.
- Implemented a multi-page update flow using a train UI object.
- Created and used a shared object.

Task 1: Initial Set-up

For the most part, a single page update and insert page are identical. Since you already created an insert page, the quickest way to implement a new corresponding update page with the same items and underlying view object is to simply copy the page definition and then make any necessary modifications.

Note: You are using the page copy technique shown here to avoid having to recreate a page that is virtually identical to one that you have already created. You could also implement pages that are similar as a single page with a dynamic user interface. The PPR tasks of the Lab: Implementing a Create introduced you to this technology. For more information about the Dynamic User Interface, the Oracle Application Framework Developer's Guide 12.1.2 or higher is a good place for additional information

Step 1: Copy Your Page

JDeveloper does not provide the ability to save as or copy/paste a page document, you need to use the file system for the following task.

1. Save all your work and close JDeveloper.
2. Using the GUI or command-line tool of your choice, copy the EmpCreatePG into EmpUpdatePG. The file should be in the `JDEV_USER_HOME/myprojects/<student#>/oracle/apps/ak/employee/webui` directory.

   For example, in Linux you would issue the following command:

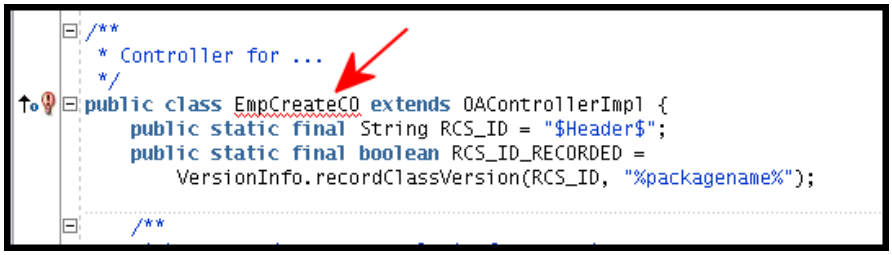
   ```
   cp $JDEV_USER_HOME/
   myprojects/<student#>/oracle/apps/ak/employee/webui/EmpCreatePG.xml  $JDEV_USER_HOME/
   myprojects/<student#>/oracle/apps/ak/employee/webui/EmpUpdatePG.xml
   ```

3. Start JDeveloper
4. Click the **<student#>.oracle.apps.ak.employee.webui** package in the Applications Navigator.
5. You should now see EmpUpdatePG listed in your Applications Navigator pane.

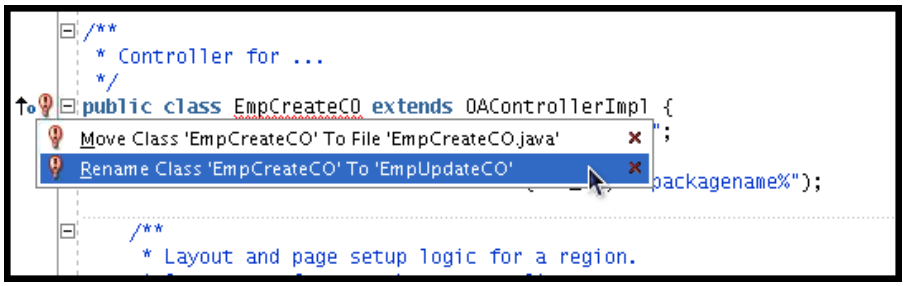
Step 2: Copy Your Controller

You need to change the controller associated with your new update page.

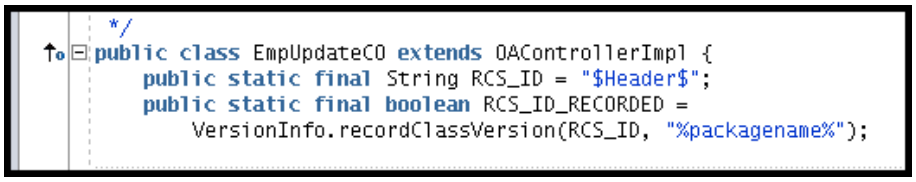
1. Open your **EmpCreateCO** controller.
2. Click **File** > **Save As** from JDeveloper's menu. Name your new controller **`EmpUpdateCO.java`**.
3. Once saved, there is now an error in the class declaration in JDeveloper with an alert icon. You have copied the file and saved it, but the declaration is to the wrong class.



4. Click the **Alert** icon.



5. Click **Rename Class 'EmpCreateCO' to 'EmpUpdateCO'** as your solution of choice. When you do, the class will be renamed.



Tip: If you forgot to change the class name to match the file name and you recompile, you will get a warning similar to the following:

Warning(28,8): class EmpCreateCO is public; should be declared in a file named CreateCO.java; or the class should be renamed to EmpUpdateCO to match the filename.

Step 3: Modify Your EmpUpdatePG to Use the EmpUpdateCO

You need to change the page to use the newly copied controller.

1. Click the **EmpUpdatePG** page's **PageLayoutRN** region in the Structure pane to open the property inspector.
2. Change the **PageLayoutRN's** properties as follows:

   Controller Class = **`<student#>.oracle.apps.ak.employee.webui.EmpUpdateCO`**

   Window Title = **<Your Name> Update**

   Title = **Update Employee: <Your Name>**

   Note: While you copied the EmpCreatePG page and the EmpCreateCO controller, and then renamed them, you are still using the same AM as the EmpCreate page as EmpUpdate is still part of the same transaction. Be careful not to change anything else at this step in the property inspector.

Step 4: Compile and Save Your Work

1. Save your work.
2. Rebuild **ClassProject** to ensure that you have no errors. Resolve any errors encountered.

Task 2: Modify the EmployeePG View-layer Components

In this task, you will add an update image column to your ResultsRN table.

Step 1: Add the Update Column to Your ResultsRN

1. Right-click the **ResultsRN** table of your EmployeePG in the Structure pane, and click **New** > **Item** from the context menu. Set **item1's** properties as follows:

   ID = **UpdateImage**

   Item Style = **image**

   Attribute Set = **`/oracle/apps/fnd/attributesets/Buttons/Update`**

   Image URI = **`updateicon_enabled.gif`**

   Additional Text = **Select to Update Employee.**

   Height = **24**

   Width = **24**

   Action Type = **fireAction**

   Event = **update**
2. Click the **(…)** button in the Parameter property, and do the following:
3. Click the **Add Parameters** button.

   Name = **empNum**

   Value = **${oa.AllEmployeesVO1.EmployeeId}**
4. Click the **Add Parameters** button.

   Name = **empName**

   Value = **${oa.AllEmployeesVO1.EmployeeName}**
5. Click the **OK** button

   Note: You will use these parameters later to ensure that you update only the employee that is selected for update. At runtime, the OA Framework automatically creates and populates formParameter (hidden) items for the empName and empNum parameters you configure. These formParameter values are then added to the request.

Task 3: Modify the Controller-layer Components

Step 1: Copy an AM Method from EmpDetailsAM

You must include the initiDetails() method from the EmpDetailsAMImpl, into the EmpCreateAMimpl for your EmpUpdatePG. Since the EmpUpdatePG uses the EmpCreateAM, and therefore the EmpCreateAMImpl, you must copy initDetails() into EmpCreateAMImpl.

1. Copy the following code from EmpDetailsAMImpl into **EmpCreateAMImpl**.

```
/**
 * @param employeeNumber
 */
public void initDetails(String employeeNumber) {
  EmployeeDetailsVOImpl vo = getEmployeeDetailsVO1();
  if (vo == null) {
    MessageToken[] errTokens =
    { new MessageToken("OBJECT_NAME", "EmployeeDetailsVO1") };
    throw new OAException ("AK", "FWK_TBX_OBJECT_NOT_FOUND",
errTokens);
  }
  vo.initQuery(employeeNumber);
} // end initDetails()
```

Step 2: Modify the EmpResultsCO Controller UI Elements (GET)

1. Add **processRequest()** logic to handle Back button navigation. Add the `else if` logic shown in **BOLD** to the `if` statement that you already added in the Create lab.

   Note: This code is added to the **processRequest()** method of the **EmpResultsCO**.java class after the `if` statement shown. The added code is shown in **BOLD** for clarity.

```
if
(TransactionUnitHelper.isTransactionUnitInProgress(pageContext,
"empCreateTxn", false)) {
  am.invokeMethod("rollbackEmployee");
  TransactionUnitHelper.endTransactionUnit(pageContext,
                                           "empCreateTxn");
} else if
(TransactionUnitHelper.isTransactionUnitInProgress(pageContext,
"empUpdateTxn", false)) {
  am.invokeMethod("rollbackEmployee");
  TransactionUnitHelper.endTransactionUnit(pageContext,
                                           "empUpdateTxn");
}
```

Step 3: Modify the EmpResultsCO Event Handling (POST)

1. Add **processFormRequest()** logic to determine if the update icon was clicked. Add this code to the end of the existing processFormRequest() method.

```
} else if
("update".equals(pageContext.getParameter(EVENT_PARAM))) {
  // The user has clicked an update icon so you want to navigate
  // to the Update page. Later, you will change the single-page
  // update to a multi-page flow.

pageContext.setForwardURL("OA.jsp?page=/<student#>/oracle/apps/a
k/employee/webui/EmpUpdatePG",
                          null,

OAWebBeanConstants.KEEP_MENU_CONTEXT,
                          null, null, false,

OAWebBeanConstants.ADD_BREAD_CRUMB_NO,
                          OAWebBeanConstants.IGNORE_MESSAGES);
}
```

Step 4: Modify the EmpUpdateCO Controller UI Elements (GET)

1. Change the logic in the **EmpUpdateCO** controller to properly support the update action and page flow. Replace the existing **processRequest()** method, which is configured for Create, with this **processRequest()** method code, which is configured for Update.

```
import java.io.Serializable;
...

public void processRequest(OAPageContext pageContext, OAWebBean
webBean) {
  super.processRequest(pageContext, webBean);
  OAApplicationModule am =
pageContext.getApplicationModule(webBean);

  // Put a transaction value indicating that the update
transaction
  // is now in progress.
  TransactionUnitHelper.startTransactionUnit(pageContext,
                                             "empUpdateTxn");

  String empNum = pageContext.getParameter("empNum");
  // You will use this at the end of the flow for a confirmation
  // message.
```

```
  String empName = pageContext.getParameter("empName");
  pageContext.putTransactionValue("empName", empName);
  Serializable[] params = { empNum };
  // For the update, since you're using the same VO as the
Details
  // page, you can use the same initialization logic.
  am.invokeMethod("initDetails", params);
} // end processRequest()
```

Step 5: Modify the EmpUpdateCO Controller Event Handling (POST)

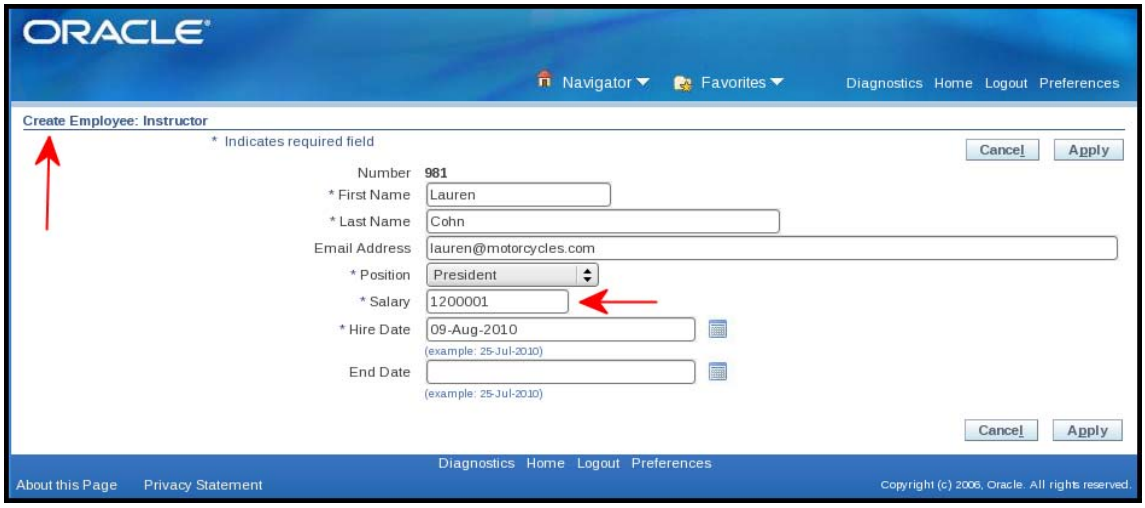
1. Change the logic in the **EmpUpdateCO** controller to properly support the update action and page flow. Replace the existing **processFormRequest()** method, which is configured for Create, with this **processFormRequest()** method code, which is configured for Update.

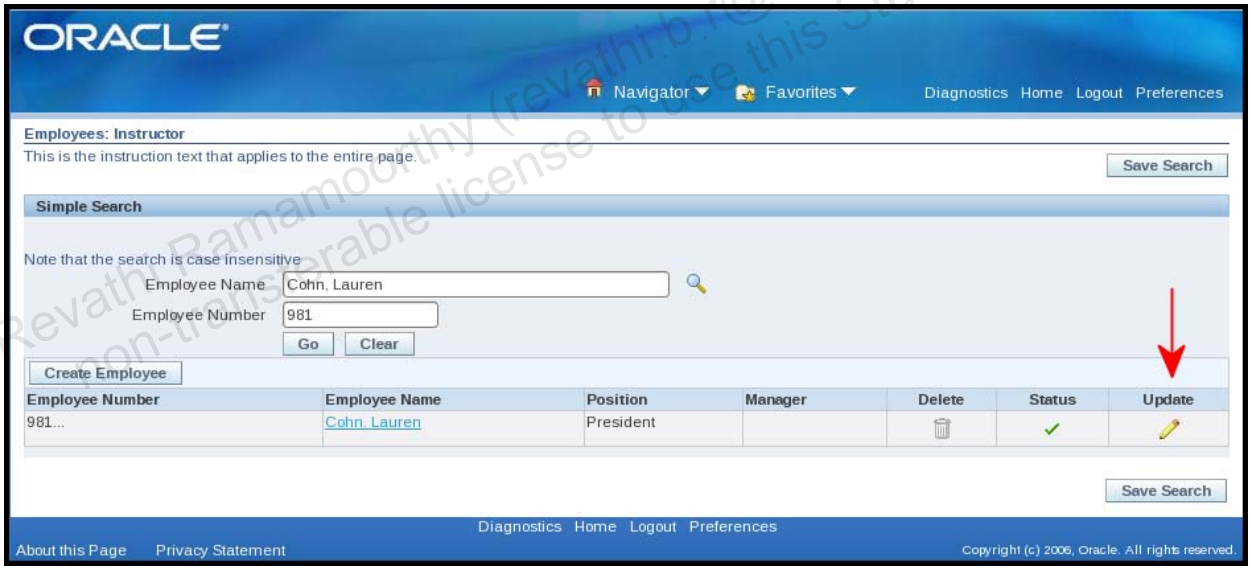
```
public void processFormRequest(OAPageContext pageContext,
                               OAWebBean webBean) {
  super.processFormRequest(pageContext, webBean);
  OAApplicationModule am =
pageContext.getApplicationModule(webBean);
  if (pageContext.getParameter("Apply") != null) {
    OAViewObject vo =
(OAViewObject)am.findViewObject("EmployeeDetailsVO1");
    String employeeName =
(String)vo.getCurrentRow().getAttribute("FirstName") + " " +
    (String)vo.getCurrentRow().getAttribute("LastName");
    Number employeeNumber =
(Number)vo.getCurrentRow().getAttribute("EmployeeId");
    String employeeNum =
String.valueOf(employeeNumber.intValue());
    MessageToken[] tokens =
      { new MessageToken("EMP_NAME", employeeName),
        new MessageToken("EMP_NUMBER", employeeNum) };
    OAException confirmMessage = new OAException("AK",
"FWK_TBX_T_EMP_UPDATE_CONFIRM", tokens,
OAException.CONFIRMATION, null);
    am.invokeMethod("apply");
    OADialogPage dialogPage = new
OADialogPage(OAException.CONFIRMATION, confirmMessage, null,
"OA.jsp?page=/<student#>/oracle/apps/ak/employee/webui/EmployeeP
G",
"OA.jsp?page=/<student#>/oracle/apps/ak/employee/webui/EmployeeP
G");
    dialogPage.setOkButtonItemName("CreateOKButton");
    dialogPage.setOkButtonLabel("OK");
```

```
      dialogPage.setNoButtonLabel("No");
      dialogPage.setPostToCallingPage(true);
      pageContext.redirectToDialogPage(dialogPage);
    }
    if (pageContext.getParameter("Cancel") != null) {
      am.invokeMethod("rollbackEmployee");
      TransactionUnitHelper.endTransactionUnit(pageContext,
                                               "empUpdateTxn");

pageContext.forwardImmediately("OA.jsp?page=/<student#>/oracle/a
pps/ak/employee/webui/EmployeePG",
                               null,

OAWebBeanConstants.KEEP_MENU_CONTEXT,
                               null, null, false,

OAWebBeanConstants.ADD_BREAD_CRUMB_NO);
    } else if
("positionChange".equals(pageContext.getParameter(OAWebBeanConst
ants.EVENT_PARAM))) {
      am.invokeMethod("handlePositionChangeEvent");
    }
}
```

Step 6: Test Your Work

1. Save your work.
2. Rebuild **ClassProject**.
3. Run your **EmployeePG**.
4. Click the **Create Employee** button. Create a new employee that you will use for testing your update functions.
5. After creating the new employee, return to the EmployeePG, and query your new employee. Confirm that the Update icon is present.
6. Click the **Update** icon.
7. Make changes to the base data of the new employee.
8. Click the **Apply** button.
9. Query your newly updated employee, and use the drill-down to details to confirm that your updates have been applied.





Task 4: Create Your View-layer Components for Multi-Page

Now that you have a single page update, you will convert this to a multi-page update flow including shared regions for the train locator bean and the navigation buttons.

For the purposes of implementing Back button logic, this exercise assumes that, after committing, it is not appropriate to return to any of the pages in the Update transaction using the browser Back button. Assume, for example, that all employee updates trigger a Workflow approval, so you cannot make changes after you commit.

Step 1: Copy Your Page

1. Save all your work and close JDeveloper.
2. Using the GUI or command-line tool of your choice, make three (3) copies of the EmpUpdatePG into UpdateEmpDetailsPG, UpdateEmpAssignPG, and UpdateEmpReviewPG. The file should be in the `JDEV_USER_HOME/myprojects/<student#>/oracle/apps/ak/employee/webui` directory.

   Note: The Linux command is **cp**. See Task 1, Step 1 in this lab for an additional example of the **cp** command. The format of the command is as follows:

   cp <source> <destination>
3. Start JDeveloper
4. You should now see all three (3) pages, UpdateEmpDetailsPG, UpdateEmpAssignPG, and UpdateEmpReviewPG, listed in your Applications Navigator pane.

Step 2: Copy Your Controller

You need to create three (3) controllers that will be associated with your three (3) new pages.

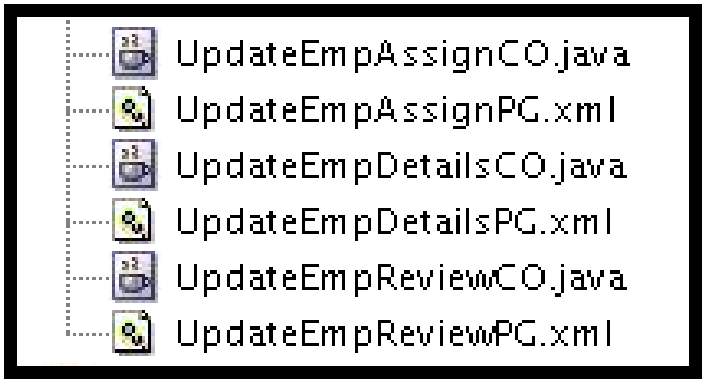
1. Start JDeveloper.
2. Open your **EmpUpdateCO** controller.
3. Click **File** > **Save As** from JDeveloper's menu. Name your new controller **UpdateEmpDetailsCO**.

   Note: The class declaration in JDeveloper has an alert icon.
4. Click the **Alert** icon.
5. Click **Rename Class 'EmpUpdateCO' to 'UpdateEmpDetailsCO'**. When you do, the class will be renamed.
6. Click your **EmpUpdateCO** controller again.
7. Click **File** > **Save As** from JDeveloper's menu. Name your new controller **UpdateEmpAssignCO**.

   Note: The class declaration in JDeveloper has an alert icon.
8. Click the **Alert** icon.
9. Click **Rename Class 'EmpUpdateCO' to 'UpdateEmpAssignCO'**. When you do, the class will be renamed.
10. Click your **EmpUpdateCO** controller again.
11. Click **File** > **Save As** from JDeveloper's menu. Name your new controller **UpdateEmpReviewCO**.

    Note: The class declaration in JDeveloper has an alert icon.
12. Click the **Alert** icon.

13. Click **Rename Class 'EmpUpdateCO' to 'UpdateEmpReviewCO'**. When you do, the class will be renamed.
14. Save your work.



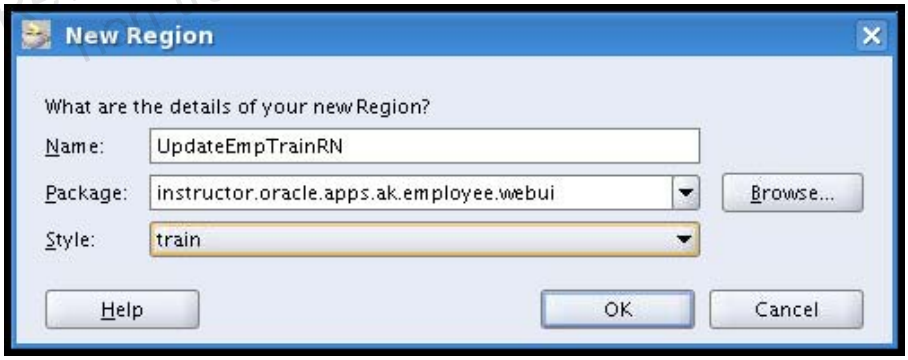
Step 3: Create Your Standalone (Shared) Train - UpdateEmpTrainRN

You are building the shared region (UpdateEmpTrain) that includes an OANavigationBarBean component, and has Cancel and Submit buttons on the appropriate pages (the Cancel button always renders, and the Submit button should render on the final page in the flow).

If you click the Submit button, a Confirmation page dialog will display. If you click the Back button, you will return to the last step in the flow, and if you click the Cancel button you will return to the Search and Results page where you started.

You do not have to implement any navigation logic; the OA Framework handles this for you using the links that you configure below as part of the OANavigationBarBean.

1. Right-click the **<student#>.oracle.apps.ak.employee.webui** package in the Applications Navigator, and click **New...** from the context menu. In the New Gallery, click **Web Tier** > **OA Component** > **Region**. Click the **OK** button.
2. Set the **Name** to **UpdateEmpTrainRN**.
3. Set the **Style** to **train**.
4. Click the **OK** button.



5. Click **UpdateEmpTrainRN** in the Applications Navigator pane. Click **UpdateEmpTrainRN** in the Structure pane. Set the **Allow Interaction** property to **True** in the property inspector.
6. Save your work.

Step 4: Add Your Train Nodes

1. Right-click the **UpdateEmp**TrainRN in the Structure pane, and click **New** > **link** from the context menu. Set **item1's** properties as follows:

   ID = **TrainStep1**

   Destination URI = **`OA.jsp?page=/<student#>/oracle/apps/ak/employee/webui/UpdateEmpDetailsPG`**

   Text = **Step 1**

2. Right-click the UpdateEmpTrainRN in the Structure pane, and click **New** > link from the context menu. Set **item1's** properties as follows:

   ID = **TrainStep2**

   Destination URI = **`OA.jsp?page=/<Student#>/oracle/apps/ak/employee/webui/UpdateEmpAssignPG`**

   Text = **Step 2**

3. Right-click the **UpdateEmp**TrainRN in the Structure pane, and click **New** > **link** from the context menu. Set **item1's** properties as follows:

   ID = **TrainStep3**

   Destination URI = **`OA.jsp?page=/<student#>/oracle/apps/ak/employee/webui/UpdateEmpReviewPG`**

   Text = **Step 3**

Step 5: Create Your Standalone (Shared) Train Footer

1. Right-click the **<student#>.oracle.apps.ak.employee.webui** package in the Applications Navigator, and click **New...** from the context menu. In the New Gallery, click **Web Tier** > **OA Component** > **Region**. Click the **OK** button.
2. Set the **Name** to **UpdateEmpTrainFooterRN**.
3. Set the **Style** to **pageButtonBar**.
4. Click the **OK** button.

Step 6: Add Your Navigation Buttons

In this step, you will add a Cancel button, an `OANavigationBarBean` (to render the Next/Back buttons and the "Step X of Y" poplist) and a Submit button. You will add code a bit later to conditionally display and handle the Submit button.

1. Right-click the **UpdateEmpTrainFooterRN** in the Structure pane, and click **New** > **Item** from the context menu. Set the **item's** properties as follows:

   ID = **Cancel**

   Item Style = **submitButton**

   Attribute Set = **`/oracle/apps/fnd/attributesets/Buttons/Cancel`**

   Disable Server-Side Validation = **True**

   Disable Client-Side Validation = **True**

2. Right-click the **UpdateEmpTrainFooterRN** in the Structure pane, and click **New** > **Region** from the context menu. Set the **region's** properties as follows:

   ID = **NavBar**

   Region Style = **navigationBar**

   First Step = **1**

   Last Step = **3**

3. Right-click the **UpdateEmpTrainFooterRN** in the Structure pane, and click **New** > **Item** from the context menu. Set the **item's** properties as follows:

   ID = **Submit**

   Item Style = **submitButton**

   Attribute Set = **`/oracle/apps/fnd/attributesets/Buttons/Submit`**

Step 7: Add Links to Your Navigation Bar

Since you are coupling your navigation bar with an interaction train, it should be interactive as well (meaning that a poplist displays after the first step that lets the user navigate directly to a step as they would by clicking a train node). To do this, you add a link for each page in the flow. Note that these links must have the same `Destination URI` as the corresponding train step nodes, and the `Warn About Changes` property is set to False since users should be able to navigate back and forth within the flow without being warned about a loss of data.

1. Click the **UpdateEmpTrainFooterRN**. Right-click the **NavBar** in the Structure pane, and click **New** > **link** from the context menu. Set the **item's** properties as follows:

   ID = **Step1Link**

   Destination URI = **`OA.jsp?page=/<student#>/oracle/apps/ak/employee/webui/UpdateEmpDetailsPG`**

   Warn About Changes = **False**

   Text = **Step 1 of 3: Details**

2. Right-click the **NavBar** in the Structure pane, and click **New** > **link** from the context menu. Set the **item's** properties as follows:

   ID = **Step2Link**

   Destination URI = **`OA.jsp?page=/<student#>/oracle/apps/ak/employee/webui/UpdateEmpAssignPG`**

   Warn About Changes = **False**

   Text = **Step 2 of 3: Assignment**

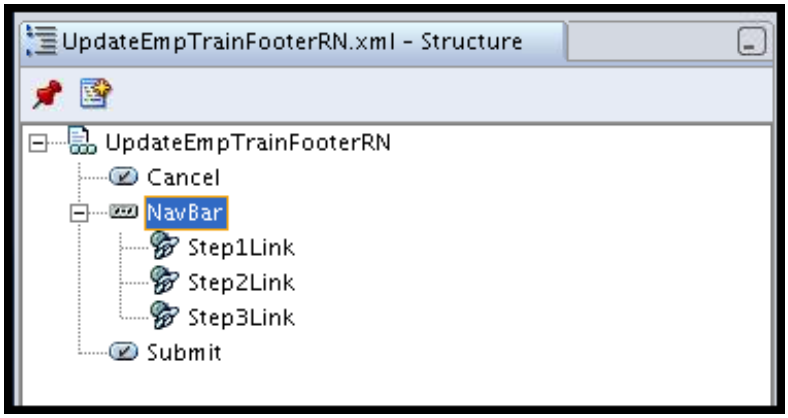
3. Right-click the **NavBar** in the Structure pane, and click **New** > **link** from the context menu. Set the **item's** properties as follows:

   ID = **Step3Link**

   Destination URI = **`OA.jsp?page=/<student#>/oracle/apps/ak/employee/webui/UpdateEmpReviewPG`**

   Warn About Changes = **False**

   Text = **Step 3 of 3: Review**



Step 8: Change UpdateEmpDetailsPG Properties

1. Click your **PageLayoutRN** in **UpdateEmpDetails** and set the following properties:

   Controller Class = **`<student#>.oracle.apps.ak.employee.webui.UpdateEmpDetailsCO`**

   Window Title = **<Your Name> Update Details**

   Title = **Update Employee: Details <Your Name>**

2. Click the **MainRN**, and remove the following regions and items.

   Note: To remove an item or region, right-click it in the Structure pane, and click Delete from the context menu.

   PositionCode

   ManagerName

   Salary

   StartDate

   EndDate

   ManagerIdLayout

   PageButtons (under the PageLayoutRN)

3. Right-click the **PageLayoutRN**, and click **New** > **location** from the context menu. Set the **region's** properties as follows:

   ID = **TrainRN**

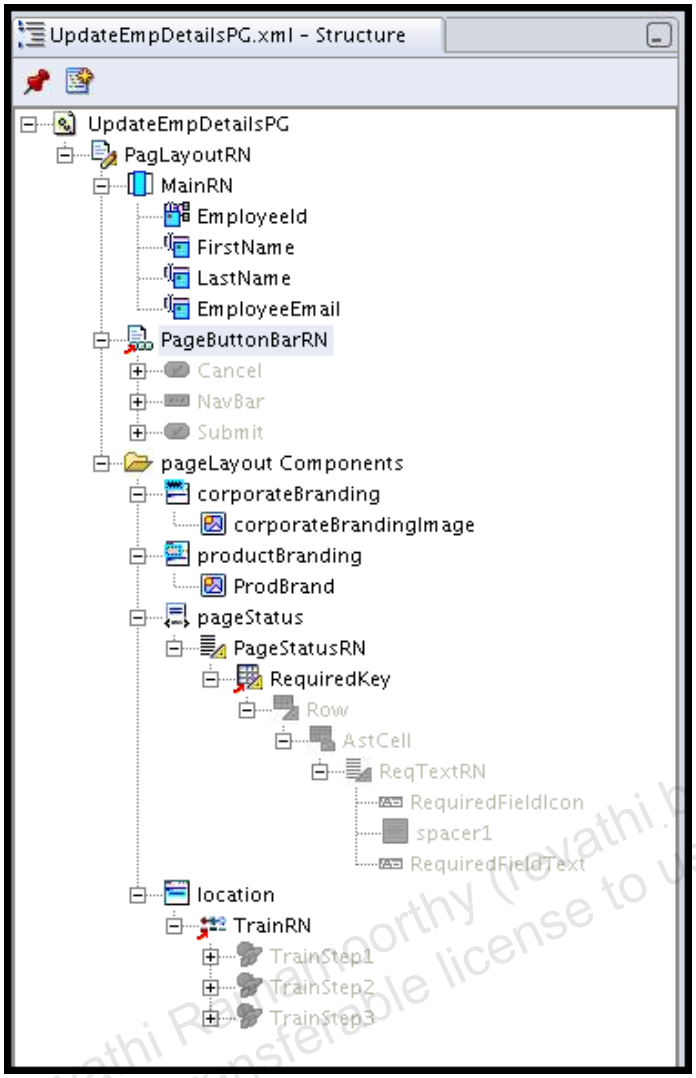
   Extends = **`/<student#>/oracle/apps/ak/employee/webui/UpdateEmpTrainRN`**

4. Right-click the **PageLayoutRN**, and click **New** > **Region** from the context menu. Set the **region's** properties as follows:

   ID = **PageButtonBarRN**

   Region Style = **pageButtonBar**

   Extends = **`/<student#>/oracle/apps/ak/employee/webui/UpdateEmpTrainFooterRN`**



Step 9: Change UpdateEmpAssignPG Properties

1. Click your **PageLayoutRN** in UpdateEmpAssignPG and set the following properties:
   Controller Class = **`<student#>.oracle.apps.ak.employee.webui.UpdateEmpAssignCO`**
   Window Title = **<Your Name> Update Assignment**
   Title = **Update Employee: Assignment <Your Name>**
2. Click the **MainRN**, and remove the following regions and items.
   EmployeeId
   FirstName
   LastName
   EmployeeEmail
   PageButtons

3. Right-click the **PageLayoutRN**, and click **New** > **location** from the context menu. Set the **region's** properties as follows:

   ID = **TrainRN**

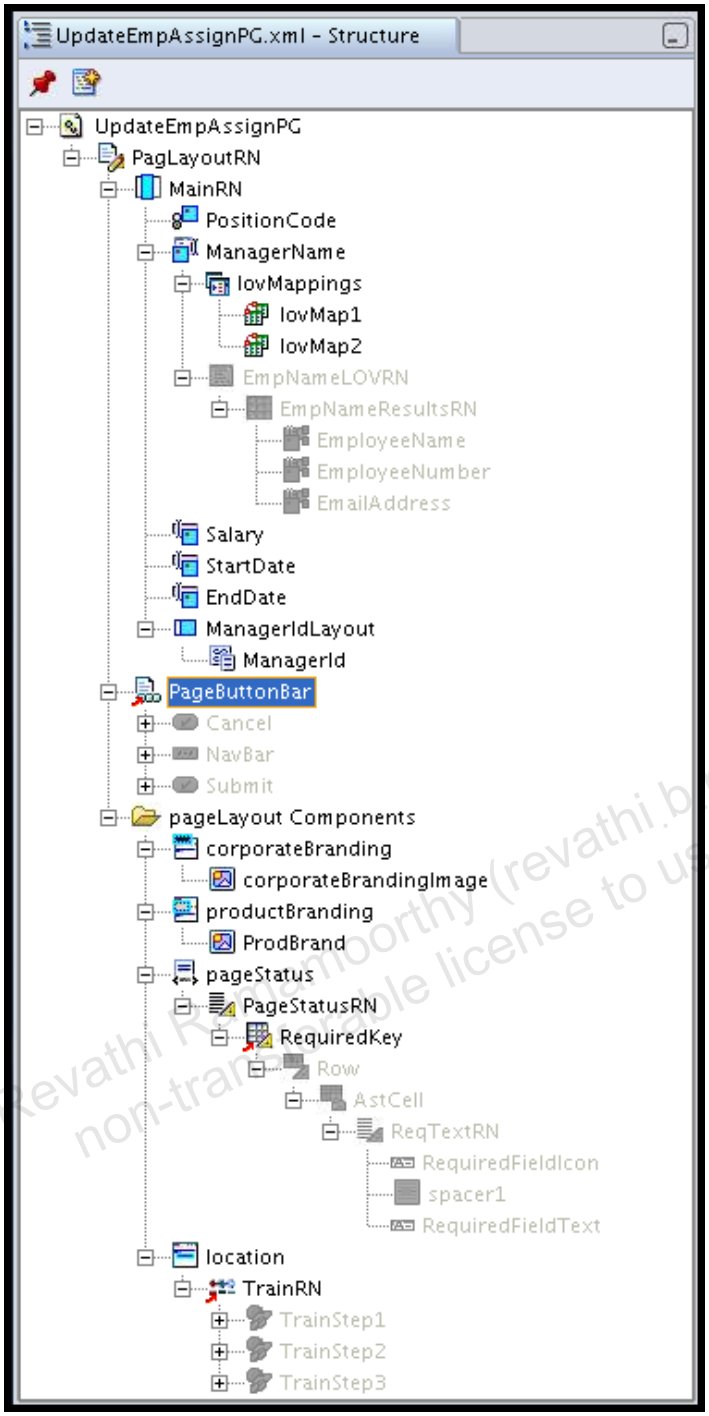
   Extends = **`/<student#>/oracle/apps/ak/employee/webui/UpdateEmpTrainRN`**

4. Right-click the **PageLayoutRN**, and click **New** > **Region** from the context menu. Set the **region's** properties as follows:

   ID = **PageButtonBarRN**

   Region Style = **pageButtonBar**

   Extends = **`/<student#>/oracle/apps/ak/employee/webui/UpdateTrainFooterRN`**



Step 10: Change UpdateEmpReviewPG Properties

1. Click your **PageLayoutRN** in UpdateEmpReviewPG and set the following properties:
   Controller Class = **`<student#>.oracle.apps.ak.employee.webui.UpdateEmpReviewCO`**
   Window Title = **<Your Name> Update Review**
   Title = **Update Employee: Review <Your Name>**
   Under the PageLayout remove the **PageButtons**.
2. Click the following items, and change their **Item Style** property to **messageStyledText**:

Note: You will see confirmation dialogs, accept them. This page is the Review page, so you are no longer concerned with inputting data. You are changing the items to display-only.

FirstName
LastName
EmployeeEmail
PositionCode
ManagerName
Salary
StartDate
EndDate

3. Right-click the **PageLayoutRN**, and click **New** > **location** from the context menu. Set the **region's** properties as follows:

   ID = **TrainRN**

   Extends = **`/<student#>/oracle/apps/ak/employee/webui/UpdateEmpTrainRN`**

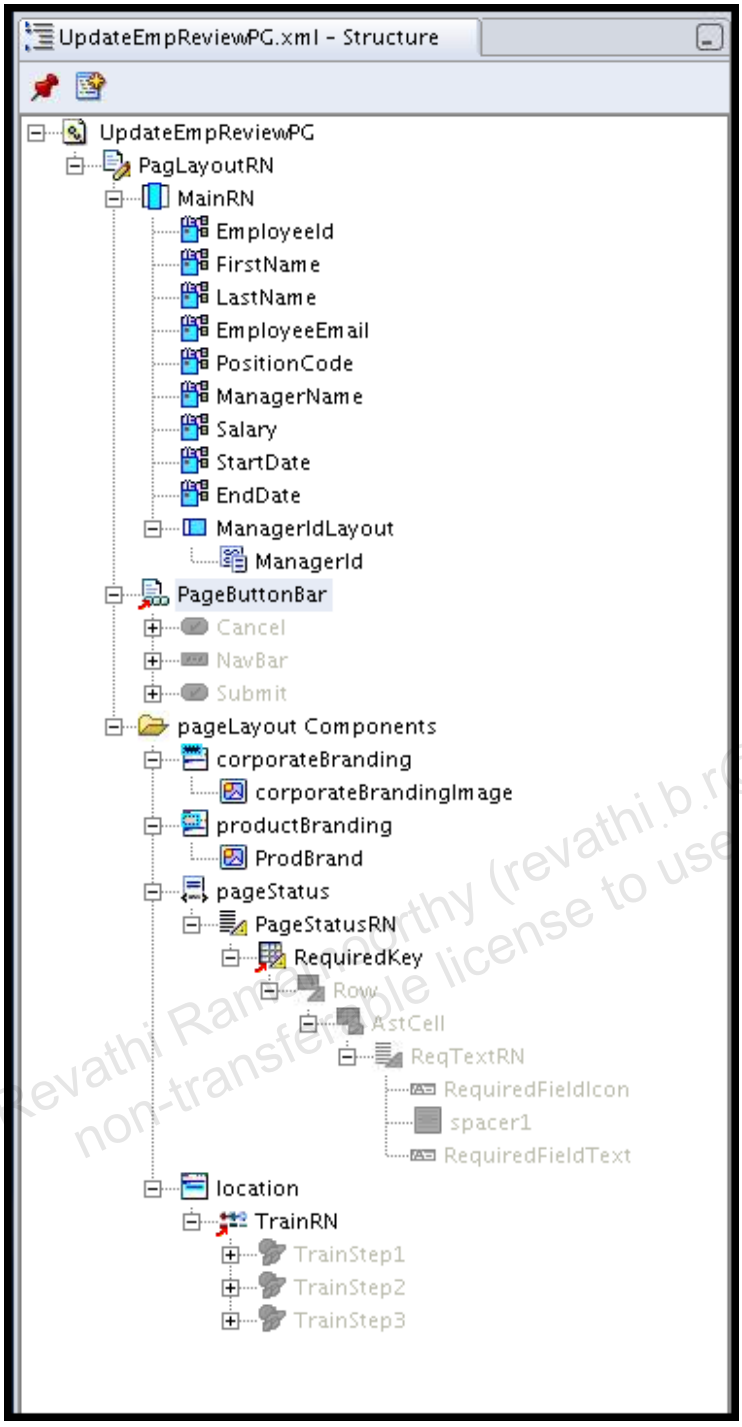
4. Right-click the **PageLayoutRN**, and click **New** > **Region** from the context menu. Set the **region's** properties as follows:

   ID = **PageButtonBarRN**

   Region Style = **pageButtonBar**

   Extends = **`/<student#>/oracle/apps/ak/employee/webui/UpdateEmpTrainFooterRN`**

5. Save your work.



Task 5: Create Your Controller-layer Components

Step 1: Modify Your UpdateEmpDetailsCO Controller

1. Open your **UpdateEmpDetailsCO** controller and replace the existing processRequest() method with the following new **processRequest()** method.

```
public void processRequest(OAPageContext pageContext, OAWebBean
webBean) {
  super.processRequest(pageContext, webBean);
  if (!pageContext.isBackNavigationFired(false) &&
     (!"goto".equals(pageContext.getParameter(EVENT_PARAM)))) {
    TransactionUnitHelper.startTransactionUnit(pageContext,
"empUpdateTxn");
    String empName = pageContext.getParameter("empName");
    pageContext.putTransactionValue("empName", empName);
    String empNum = pageContext.getParameter("empNum");
    Serializable[] params = { empNum };
    OAApplicationModule am =
pageContext.getApplicationModule(webBean);
    am.invokeMethod("initDetails", params);
  } else {
    if
(!TransactionUnitHelper.isTransactionUnitInProgress(pageContext,
"empUpdateTxn", true)) {
      OADialogPage dialogPage = new
OADialogPage(NAVIGATION_ERROR);
      pageContext.redirectToDialogPage(dialogPage);
    }
  }
} // end processRequest()
```

2. Remove all the code from the existing processFormRequest() method until your **processFormRequest()** method looks as follows:

```
public void processFormRequest(OAPageContext pageContext,
                               OAWebBean webBean) {
  super.processFormRequest(pageContext, webBean);
}
```

Step 2: Modify Your UpdateEmpAssignCO Controller

1. Open your **UpdateEmpAssignCO** controller and replace the existing processRequest() method with the following new **processRequest()** method to ensure that the user cannot return to the page after committing and then attempt to make further changes.

```
public void processRequest(OAPageContext pageContext, OAWebBean
webBean) {
  //Always call this first
  super.processRequest(pageContext, webBean);
  if
(!TransactionUnitHelper.isTransactionUnitInProgress(pageContext,
"empUpdateTxn",true)) {
    OADialogPage dialogPage = new
OADialogPage(NAVIGATION_ERROR);
    pageContext.redirectToDialogPage(dialogPage);
  }

  //Keep the functionality of the PPR for position code
validation
  OAApplicationModule am =
pageContext.getApplicationModule(webBean);
  am.invokeMethod("init");

} //  end processRequest
```

```
public void processRequest(OAPageContext pageContext, OAWebBean webBean) {
    //Always call this first
    super.processRequest(pageContext, webBean);
    if (!TransactionUnitHelper.isTransactionUnitInProgress(pageContext,
                                                           "empUpdateTxn",
                                                           true)) {
        OADialogPage dialogPage = new OADialogPage(NAVIGATION_ERROR);
        pageContext.redirectToDialogPage(dialogPage);
    }

    //Keep the functionality of the PPR for the position code validation
    OAApplicationModule am = pageContext.getApplicationModule(webBean);
    am.invokeMethod("init");

} //  end processRequest
```

2. Remove all the code from the existing processFormRequest() method until your **processFormRequest()** method looks as follows:

```
public void processFormRequest(OAPageContext pageContext,
```

```
                                        OAWebBean webBean) {
    super.processFormRequest(pageContext, webBean);

    //Handle the PPR for position change
    OAApplicationModule am =
  pageContext.getApplicationModule(webBean);
    if
  ("positionChange".equals(pageContext.getParameter(OAWebBeanConst
  ants.EVENT_PARAM))) {
      // The PositionCode PPR change event has fired.
      am.invokeMethod("handlePositionChangeEvent");
    }
  }
```

```
public void processFormRequest(OAPageContext pageContext,
                               OAWebBean webBean) {
    super.processFormRequest(pageContext, webBean);

    //Handle the PPR for position change
    OAApplicationModule am = pageContext.getApplicationModule(webBean);
    if ("positionChange".equals(pageContext.getParameter(OAWebBeanConstants.EVENT_PARAM))) {
        // The PositionCode PPR change event has fired.
        am.invokeMethod("handlePositionChangeEvent");
    }


} // end processFormRequest
```

Step 3: Modify Your UpdateEmpReviewCO Controller

1. Open your **UpdateEmpReviewCO** controller and replace the existing processRequest() method with the following new **processRequest()** method to ensure that the user cannot return to the page after committing and then attempt to make further changes.

```
public void processRequest(OAPageContext pageContext,
                           OAWebBean webBean) {
  super.processRequest(pageContext, webBean);
  if (!TransactionUnitHelper.isTransactionUnitInProgress(pageContext,

"empUpdateTxn",
                                                         true)) {
    OADialogPage dialogPage = new
OADialogPage(NAVIGATION_ERROR);
    pageContext.redirectToDialogPage(dialogPage);
  }
} // end processRequest()
```

2. Remove all the code from the existing processFormRequest() method until your **`processFormRequest()`** method looks as follows:

```
public void processFormRequest(OAPageContext pageContext,
                               OAWebBean webBean) {
  super.processFormRequest(pageContext, webBean);
}
```

Step 4: Create the controller for UpdateEmpTrainFooterRN

1. Right-click the **UpdateEmpTr**ainFooterRN in the Structure panel, and click **Set New Controller ...** from the context menu.
2. Set the **Package Name** to **`<student#>.oracle.apps.ak.employee.webui`**.
   Note: The default package name lists webui twice. You need to correct this.
3. Set the **Name** to **UpdateEmpTrainFooterCO**.
4. Click the **OK** button.

Step 5: Handle Your Footer Region's UI Elements (GET)

Now add code to control whether the submit button is displayed (based on the current page), and indicate to the `OANavigationBarBean` what step to render.

1. Add the following code to the processRequest() method.

```
import oracle.apps.fnd.framework.webui.beans.form.OASubmitButtonBean;
import oracle.apps.fnd.framework.webui.beans.nav.OATrainBean;
import oracle.apps.fnd.framework.OAApplicationModule;
import oracle.apps.fnd.framework.webui.OAWebBeanConstants;
...

public void processRequest(OAPageContext pageContext,
                           OAWebBean webBean) {
  super.processRequest(pageContext, webBean);
  // Figure out whether the Submit button should be rendered or
  // not; this should appear only on the final page (Step 3).
  // The OATrainBean is a named component of the page layout, so
  // you have a special way of getting a handle to it (you can't
  // "find" it like you do for normal, indexed children that
would
  // be below the current region in the hierarchy.
  OATrainBean trainBean =
(OATrainBean)pageContext.getPageLayoutBean().getLocation();
  // You must call the following before getting the target page
  // index.
  trainBean.prepareForRendering(pageContext);
  int step = trainBean.getClickedTrainStepRenderedIndex();
```

```
  if (step + 1 != trainBean.getNumberOfRenderedTrainSteps()) {
      OASubmitButtonBean submitButton =
(OASubmitButtonBean)webBean.findIndexedChildRecursive("Submit");
    submitButton.setRendered(false);
  }
} // end processRequest()
```

Step 6: Handle Your Footer Region's Events (POST)

Add the following `processFormRequest()` method to your UpdateEmpTrainFooterCO to display a Confirmation message and commit the transaction when the user clicks the Submit button. It also clears the middle tier cache when the Cancel button is clicked.

Note that the OA Framework handles the `OANavigationBarBean` navigation (it can do this when it is interactive since it has the links), so you do not need to worry about explicitly navigating from one page to the next.

1. Add the following code to the **processFormRequest()** method.

```
import oracle.apps.fnd.common.MessageToken;
import oracle.apps.fnd.framework.OAException;
import oracle.apps.fnd.framework.webui.OADialogPage;
...

public void processFormRequest(OAPageContext pageContext,
                               OAWebBean webBean) {
  super.processFormRequest(pageContext, webBean);
  OAApplicationModule am = pageContext.getApplicationModule(webBean);

  // This button should only be displayed on the final page...
  if (pageContext.getParameter("Submit") != null) {
    am.invokeMethod("apply");

    String employeeName =
(String)pageContext.getTransactionValue("empName");

    // Assuming the commit succeeds, you'll display a Confirmation
    // dialog that takes the user back to the Query.

    MessageToken[] tokens = { new MessageToken("EMP_NAME",
employeeName) };

    OAException confirmMessage = new OAException("AK",
"FWK_TBX_T_EMP_UPDATE_CONFIRM", tokens);

    OADialogPage dialogPage = new
OADialogPage(OAException.CONFIRMATION, confirmMessage, null,
"OA.jsp?page=/<student#>/oracle/apps/ak/employee/webui/EmployeePG",
null);
```

```
    // Note that we release the root "UI" application module
    // so we can correctly handle any subsequent "Back" button
    // navigation and attempts to resubmit the transaction.

    pageContext.releaseRootApplicationModule();
    pageContext.redirectToDialogPage(dialogPage);

  } else if (pageContext.getParameter("Cancel") != null) {
    // Cancel button handling is required for Back button support.
    am.invokeMethod("rollbackEmployee");

    // Remove the "in transaction" indicator
    pageContext.removeTransactionValue("empInUpdateTxn");

    // retain AM

pageContext.forwardImmediately("OA.jsp?page=/<student#>/oracle/apps/ak
/employee/webui/EmployeePG", null,
OAWebBeanConstants.KEEP_MENU_CONTEXT, null, null, true,
OAWebBeanConstants.ADD_BREAD_CRUMB_NO);
  }
} // end processFormRequest()
```


Step 7: Modify Your EmpResultsCO to Use Multi-Page Update

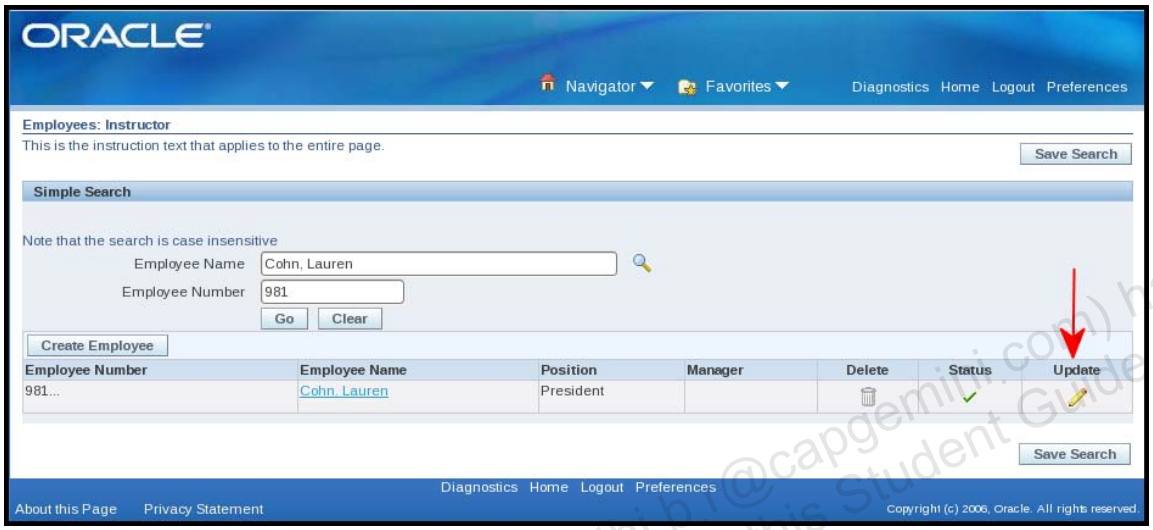
Currently, your EmployeePG's EmpResultCO forwards to the EmpUpdatePG when the update icon is clicked. You need to change this to forward to the UpdateEmpDetailsPG instead.

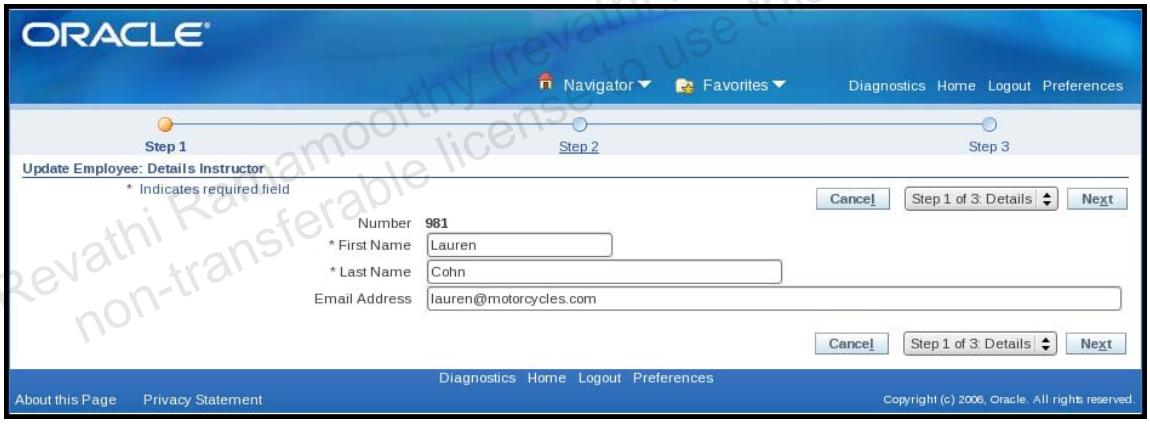
1. Change the **processFormRequest()** method code to read as follows:
   Note: You only need to change one (1) line (snippet) of code, at the end of the processFormRequest() method in the section where you test for update.

```
...
else if ("update".equals(pageContext.getParameter(EVENT_PARAM)))
{
  // The user has clicked an update icon so you want to navigate
  // to the Update page. Later, you will change the single-page
  // update to a multi-page flow.

pageContext.setForwardURL("OA.jsp?page=/<student#>/oracle/apps/a
k/employee/webui/UpdateEmpDetailsPG", null,
OAWebBeanConstants.KEEP_MENU_CONTEXT, null, null, false,
OAWebBeanConstants.ADD_BREAD_CRUMB_NO,
OAWebBeanConstants.IGNORE_MESSAGES);
}
...
```

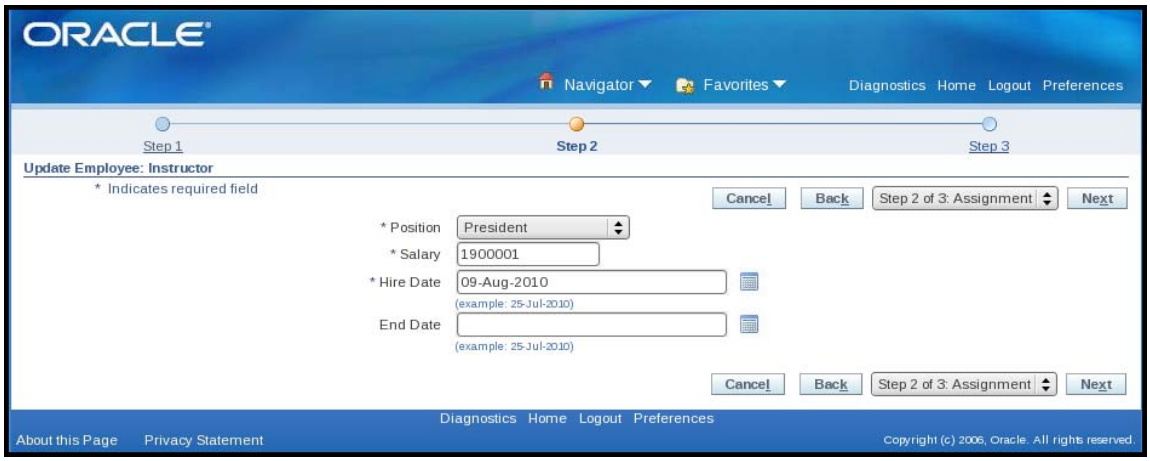
Step 8: Test Your Work

1. Save your work.
2. Rebuild **ClassProject**.
3. Run the **EmployeePG**.
4. Either query an existing employee record for update, or create a new employee and query that new employee record. Click the update icon.
5. Walk through the multipage update flow.

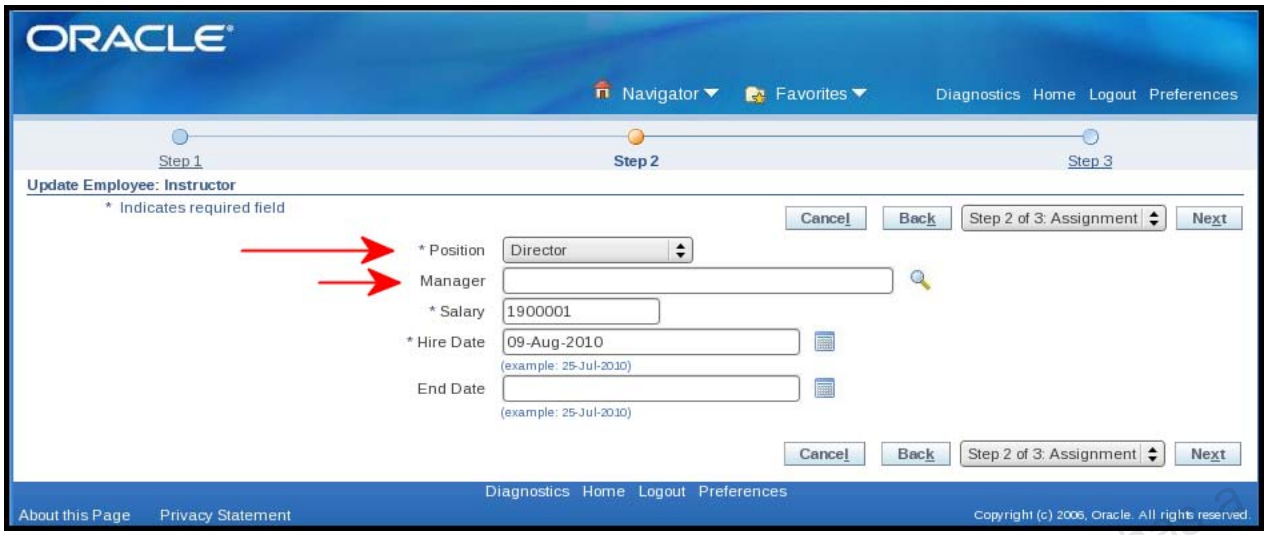


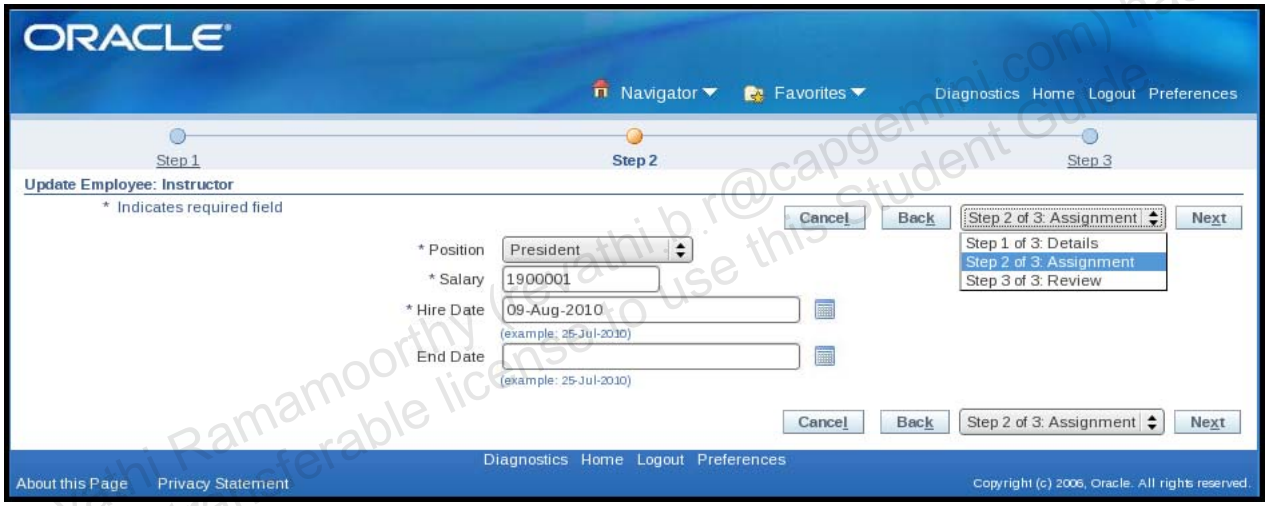


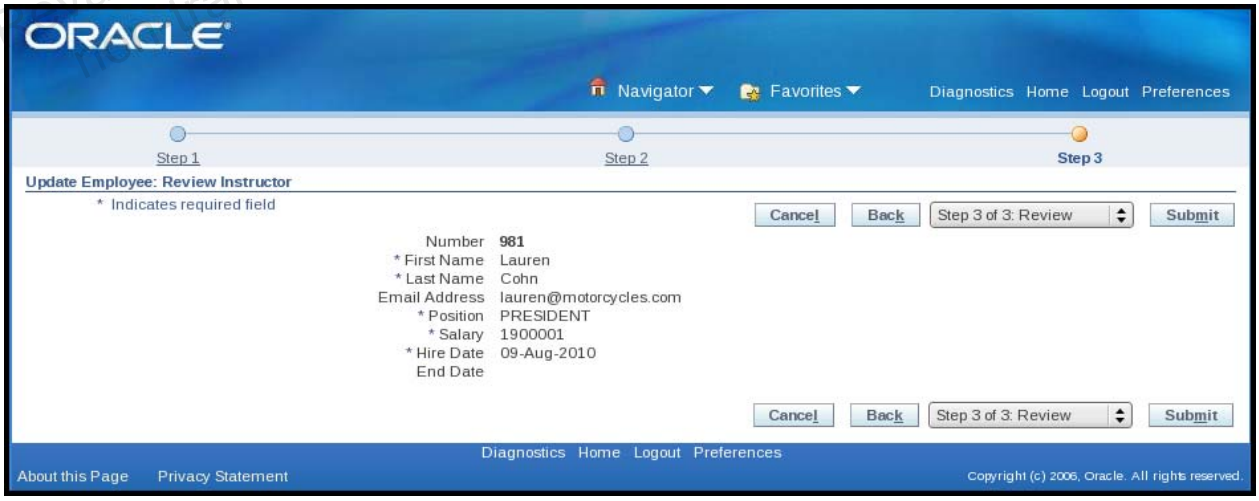
Note: In the below image that the President position code still works.

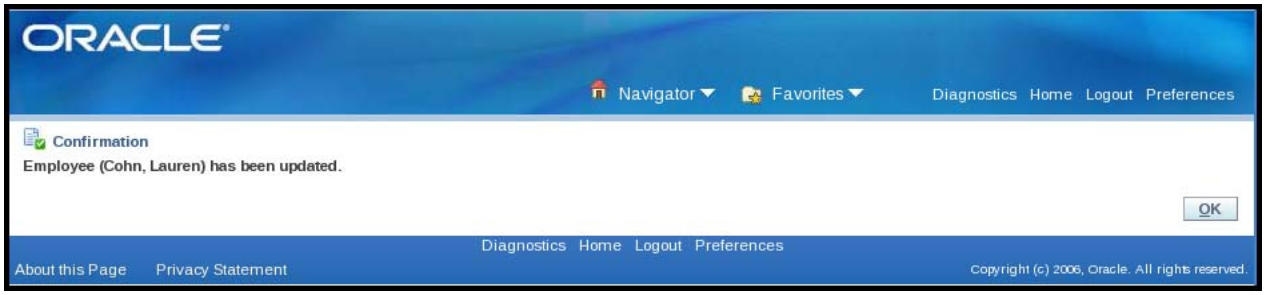


Note: The PPR event still fires.





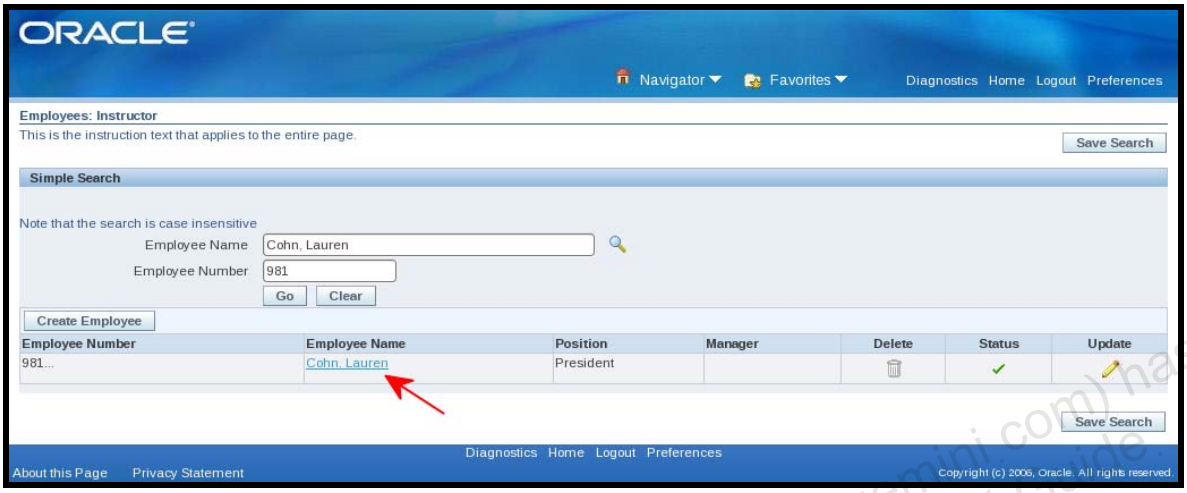


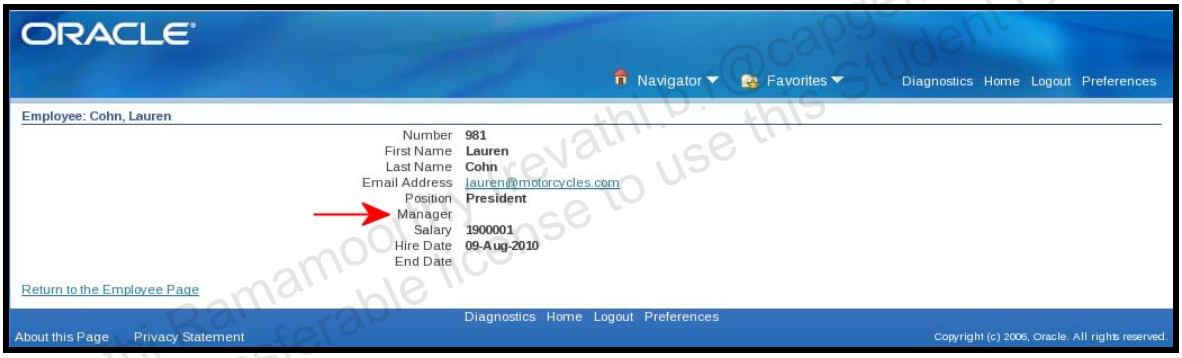


Task 6 – The Final Challenge

Knowing how to fix an issue is part of becoming an OA Framework Developer.

In the Multi-Step train, on Step 2, the PositionPVO works. Why then, if you query the Employee from the results page, and then to the details page, do you have the details page that still renders the Manager Field, even if President is present?





The EmpDetails page above was created early in the labs and existed prior to the PPR labs. The EmpDetails page has an AM that is different from the EmpCreate Page. Since the EmpDetails page displays information only, you will use a similar technique to the PPR lab and use a PVO. You will re-use some of the code you are already familiar with.

Hints:

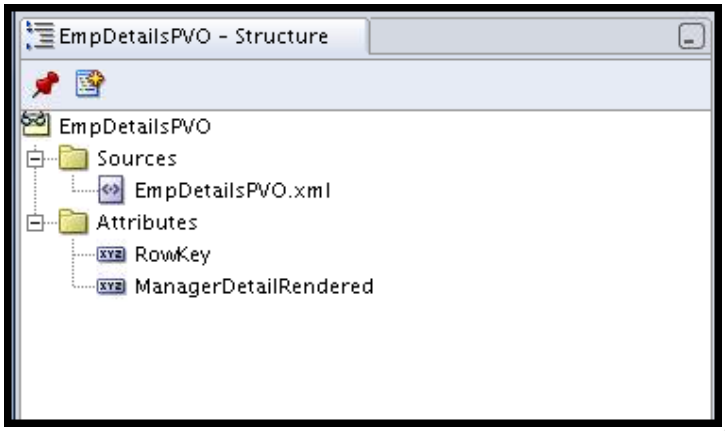
1. What is the Page that shows the details?
2. What is the AM associated with the page?
3. How will you bind the variables?

Step 1: Adding PPR to the EmpDetailsPG

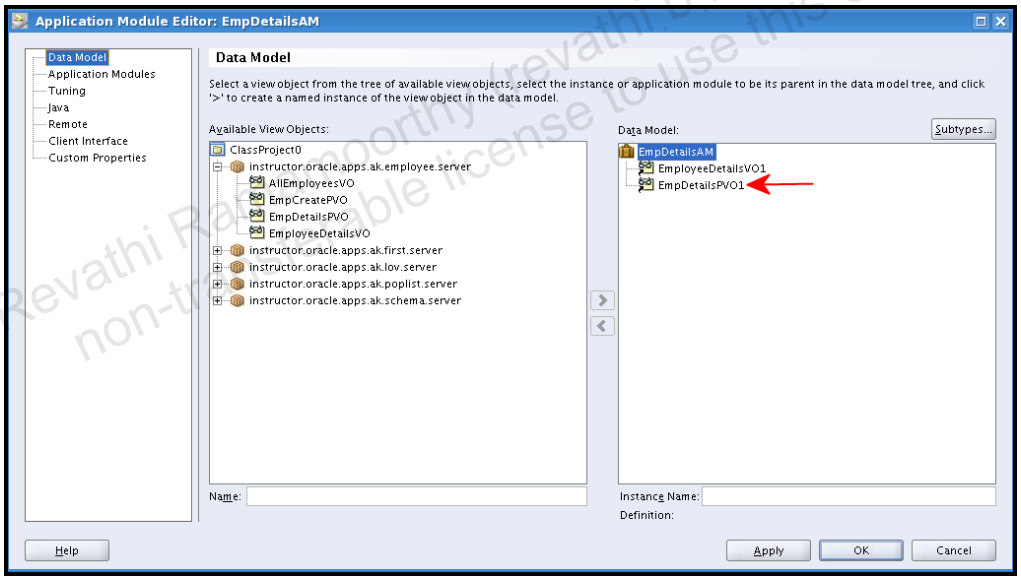
1. Create a new PVO called **EmpDetailsPVO** in the **`<student#>.oracle.apps.ak.employee.server package`**.
2. Select the **Rows Populated Programmatically** option.
3. Create a new attribute called **RowKey** with the following properties:
   Type = **Number**
   Select the **Updatable - Always** option
   Check the **Key Attribute** checkbox

Uncheck the **Queryable** checkbox

4. Create a new attribute called **ManagerDetailRendered** with the following properties:
   Type = Boolean
   Select the **Updatable - Always** option
   Uncheck the **Key Attribute** and **Queryable** checkboxes, if checked
5. Do not generate any Java files. Finish the VO.
6. Edit the **EmpDetailsPVO** and check the **Including All Transient Values** in the **Tuning** section



7. Add the **EmpDetailsPVO** to the **EmpDetailsAM**



8. Modify the **ManagerName** in the EmpDetailsPG to bind to the **EmpDetailsPVO** attribute
9. Set the **Rendered** property on the ManagerName to **${oa.EmpDetailsPVO1.ManagerDetailRendered}**
10. Add these import statements to the **EmpDetailsAMImpl** to conditionally render the ManagerName

```
import oracle.jbo.RowSetIterator;
import oracle.jbo.domain.Number;
import oracle.apps.fnd.framework.OAViewObject;
```

```
import oracle.apps.fnd.framework.OARow;
```

11. Add the following code in the **initDetails()** method immediately after the **vo.initQuery(EmployeeNumber)** in **EmpDetailsAMImpl**.

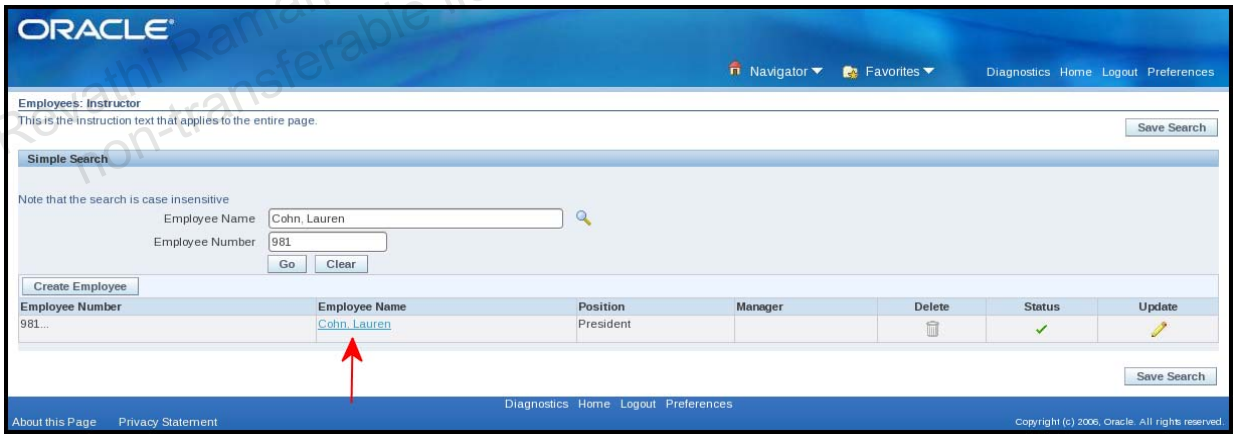
```
//Manager render start
OAViewObject appPropsVO =
(OAViewObject)findViewObject("EmpDetailsPVO1");
if (appPropsVO != null) {
  // Do not reinitialize the VO unless needed. Note that
  // this method call does not try to query the database for
  // VOs with no SELECT statement and only transient attributes.
  if (appPropsVO.getFetchedRowCount() == 0) {
    // Setting the match fetch size to 0 for an in-memory VO
    // prevents it from trying to query rows.  Calling
    // executeQuery() ensures that rows aren't lost after
    // a commit in the transaction (BC4J known issue
    // workaround).
    appPropsVO.setMaxFetchSize(0);
    appPropsVO.executeQuery();
    // You must create and insert a row in the VO before you
    // can start setting properties.
    appPropsVO.insertRow(appPropsVO.createRow());
    // Set the primary key value for this single-rwo VO.
    OARow row = (OARow)appPropsVO.first();
    row.setAttribute("RowKey", new Number(1));
  }
}
// Initialize the application properties VO (and the UI) based
// on the default employee position value set on the underlying
// object.
handlePositionDisplay();
//end
```

12. Add the **handlePositionDisplay()** method to the **EmpDetailsAMimpl**

```
public void handlePositionDisplay() {
  // Get the special, single-row application properties and make
  // the first (only) row current.
  OAViewObject vo =
(OAViewObject)findViewObject("EmpDetailsPVO1");
  OARow row = (OARow)vo.first();
```

```
  // Get the value of the view object attribute with the
position
  // code.
  EmployeeDetailsVOImpl detailsVO = getEmployeeDetailsVO1();
  RowSetIterator renderIter =
  detailsVO.createRowSetIterator("renderIter");
  int fetchedRowCount = detailsVO.getFetchedRowCount();
  renderIter.setRangeStart(0);
  renderIter.setRangeSize(fetchedRowCount);
  EmployeeDetailsVORowImpl detailRow =
(EmployeeDetailsVORowImpl)renderIter.getRowAtRangeIndex(0);
  String position =
(String)detailRow.getAttribute("PositionDisplay");
  if ((position == null) || ("President".equals(position))) {
    row.setAttribute("ManagerDetailRendered", Boolean.FALSE);
  } else {
    row.setAttribute("ManagerDetailRendered", Boolean.TRUE);
  }
} // end handlePositionDisplay()
```

13. Save and recompile your **ClassProject**.
14. Test your work.
15. Query or create an Employee.
16. Go that the employee you are testing and use the details page.



17. Now, the Details page behaves like the rest of the Employee Application you have created.

18. Congratulations! You have completed all the labs! Well done!