Building a Database-Enabled Web Application from Scratch -- 2

# Purpose of Exercise

In the previous exercise, we started with a logical model developed on Oracle SQL Developer Data Modeler, create the corresponding database in APEX and then build a simple web application to operate on single tables. At the end of this exercise you will be able to:

* Create master-detail screens for situations that involve two tables -- like sales orders where we need to coordinate operations on the main table and the order details table).
* Create user-friendly lookup fields for entering foreign key values.
* Create a home page for your application.

# The Big Picture

* **Where we stand:** After completing the previous lab, your application should look like this (Figure 1):

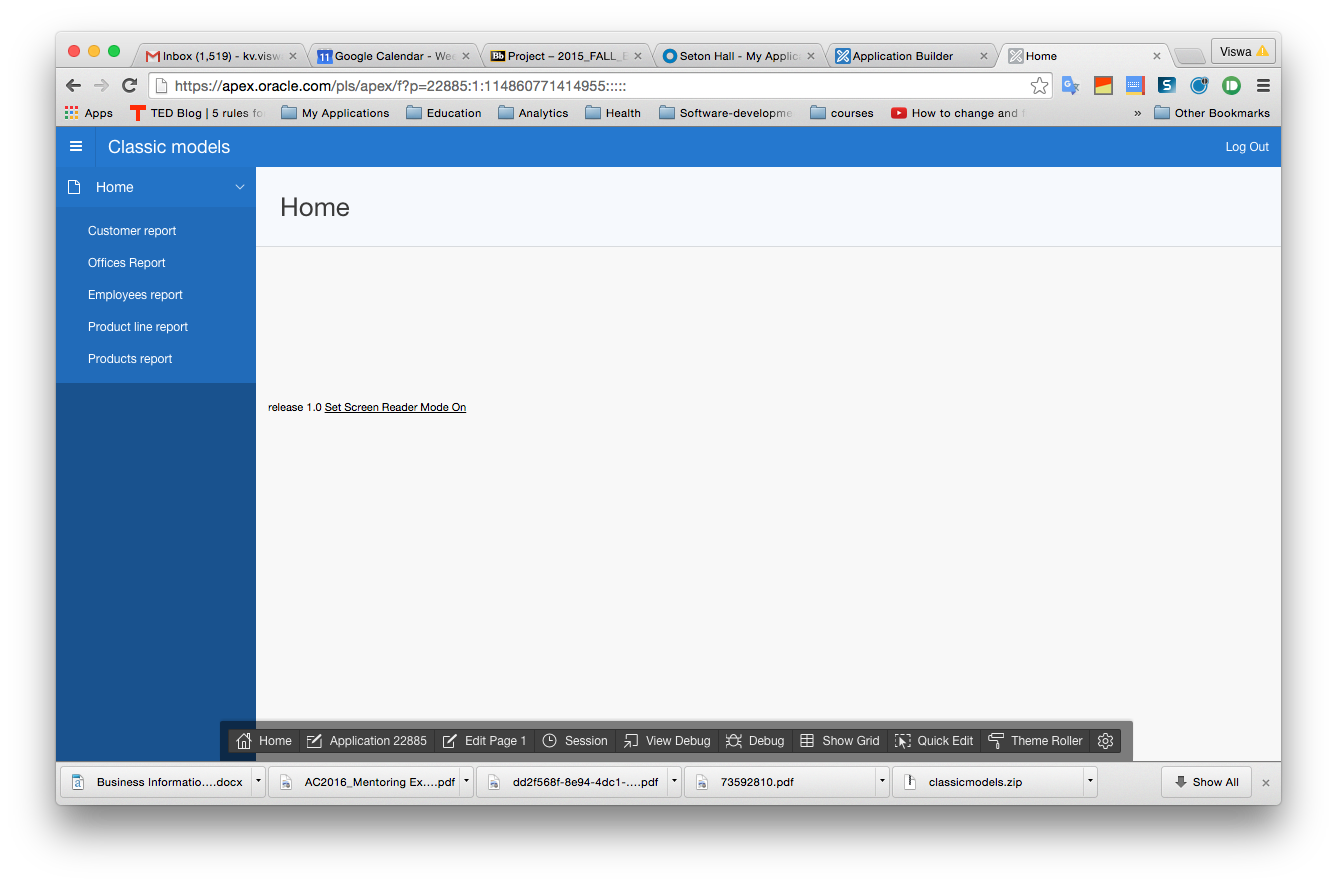


Figure : Our application at the start of this exercise

The application has tabs for *Home*, *Customer report*, *Employee report*, *Product report* and *Offices report*. (In retrospect, we could have called these *Employees, Products, Offices and Employees*).

* **What we will accomplish:** 
  + **Entering foreign key values nicely:** Click on the *Customers report* tab and then click on the edit icon to the left of any row. In the resulting screen you see the field called *Sales Rep Employee Id* and the field has a number in it. We know that this number represents the *employee id* of the sales rep for this customer. However, just looking at the number does not tell us who the actual sales rep is. Unless our company has very few sales reps, knowing the sales rep id does not tell us much. Ideally we would like the form to display the name of the corresponding sales rep. Also, in a form where we create a new customer, we would like to be able to select a sales rep based on the rep’s name rather than be forced to enter the sales rep’s id. Although we want the screen to show the name, internally the table should still store the id, but we want that process to be automated. After we finish, we will have this (Figure 2). We will soon see how to accomplish this.

|  |  |
| --- | --- |
| **Customer form now – Sales rep employee id shown as number** | **Changed customer form -- Sales rep’s name shown instead of id** |

Figure 2: Adding a selection list to the customer form

* + Thus far, our application enables us to *maintain* data for objects for which the complete details (other than a foreign key value) reside in a single table – like *customers*, *employees* and so on. For things like sales orders, we need to look at two tables to get the whole picture. The basic information resides in one table and the item details reside in another. We will create a *master-detail* form to manage this. Like the earlier screens, the initial screen will show a report of sales orders. Editing a sales order however now shows both the order header and the order item details as shown in Figure 3. Note how the top portion shows the details from the *sales order* table (with the customer who placed the order as well as the various dates associated with the order) and the bottom portion shows all the lines items for the order. In the example shown, the order has four line items and hence we see four rows in the bottom. You will generally need master-detail forms for associative entity types.
  + The application’s home page currently has no content and looks pretty insipid. You saw in the sample application that you viewed before doing the exercise that the home page had some content. We will add this in as well.

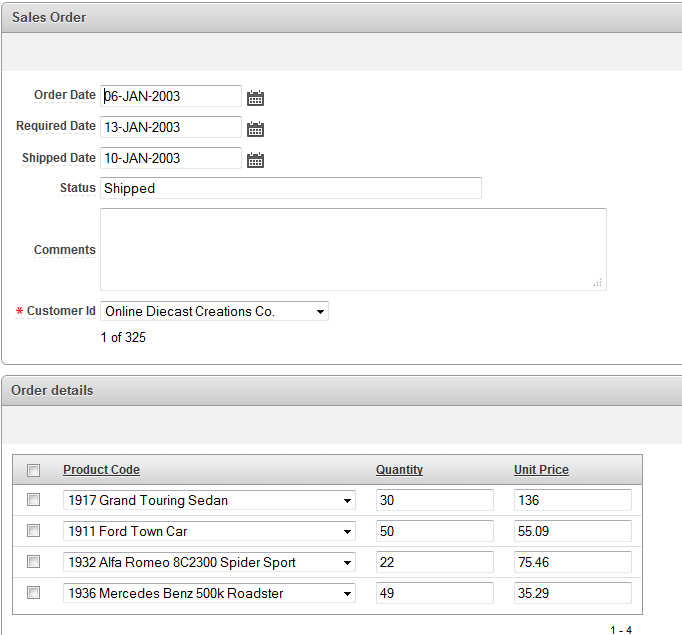
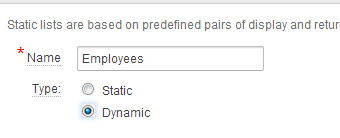


Figure 3: master detail form for sales orders

# Detailed Steps

1. The steps below will work fine for the *Classic Models* example. However, for your project, you should first create forms for the basic entities (non-associative entities) and then add one row of data to each one. You can do that by invoking the form and then using the create button to add one row of data. You should create the data in a sensible order. For example if you have a *Product* table that has a foreign key *product\_line\_id* then you will not be able to add a product until you have at least one product line. So you should first add a row for product line and then add the row for product. By the end of this step, you will have at least one row for every non-associative entity type. You can then follow the steps below for your project as well.
2. **Adding a selection list to the customer form** **–** We have already seen that the customer form has a field called *sales rep employee id* and instead of showing a bland number in this field we would like to show the name of the corresponding sales rep. We will accomplish this in two steps – first we will create an APEX object called a List of Values (LOV) that lists all employees and then we will attach this LOV to the *sales rep employee id* field of the form.
3. **Creating the employees LOV:**
   1. If you have not logged in to your APEX environment,do so now *using your administrator or developer id*. Then click on *Application Builder* and then click on the *Edit* button for the *Classic Models* application.
   2. We will create our LOVs as a *Shared Component* because we can create it once, and possibly use it in many forms throughout the application. For example, we can create an *employees* LOV and then use it to select an employee in the customer form. We can also use the very same LOV to select the manager of an employee on the employee form. Click now on *Shared components* to initiate the process of managing the shared components of our application.
   3. In the *"Other components"* section of the page, click on *Lists of Values*. You will see “No list of values found” – which is perfect since we have not yet created any LOVs for our application. Click on *Create*. On the next screen note that *From scratch* is already selected. Click *Next*.
   4. On the next screen enter *Employees* as the name for the LOV (like Employees\_LOV) and select *Dynamic* and then click *Next*. We selected *Dynamic* because if the values in the employee table change, we would like this list to show the current values. This is not a fixed or static list. It changes.



* 1. To appreciate what we will do next, recall now what we are trying to accomplish. We have a field called *sales rep employee id* on the customer form and this has an underlying numeric value representing the id of the corresponding employee. Since displaying the id seems pretty user-unfriendly, we would like APEX to go out and get the name of the corresponding employee and show it instead. Also, when creating a new customer, rather than having to remember the ids of all employees, we would like APEX to display a list of employee names and allow us to select a name and leave it to APEX to fill the id of the employee in the field.
  2. Therefore, while editing a customer, we need APEX to go out to the database and get a list of all the employees. Of course it needs to use an SQL for this. We will now create that SQL in APEX. In the *Source: List of values Query* field APEX shows us a sample of what the SQL should look like. This is only a sample and does not apply to our case. Enter what is shown in Figure 4 exactly as it is.

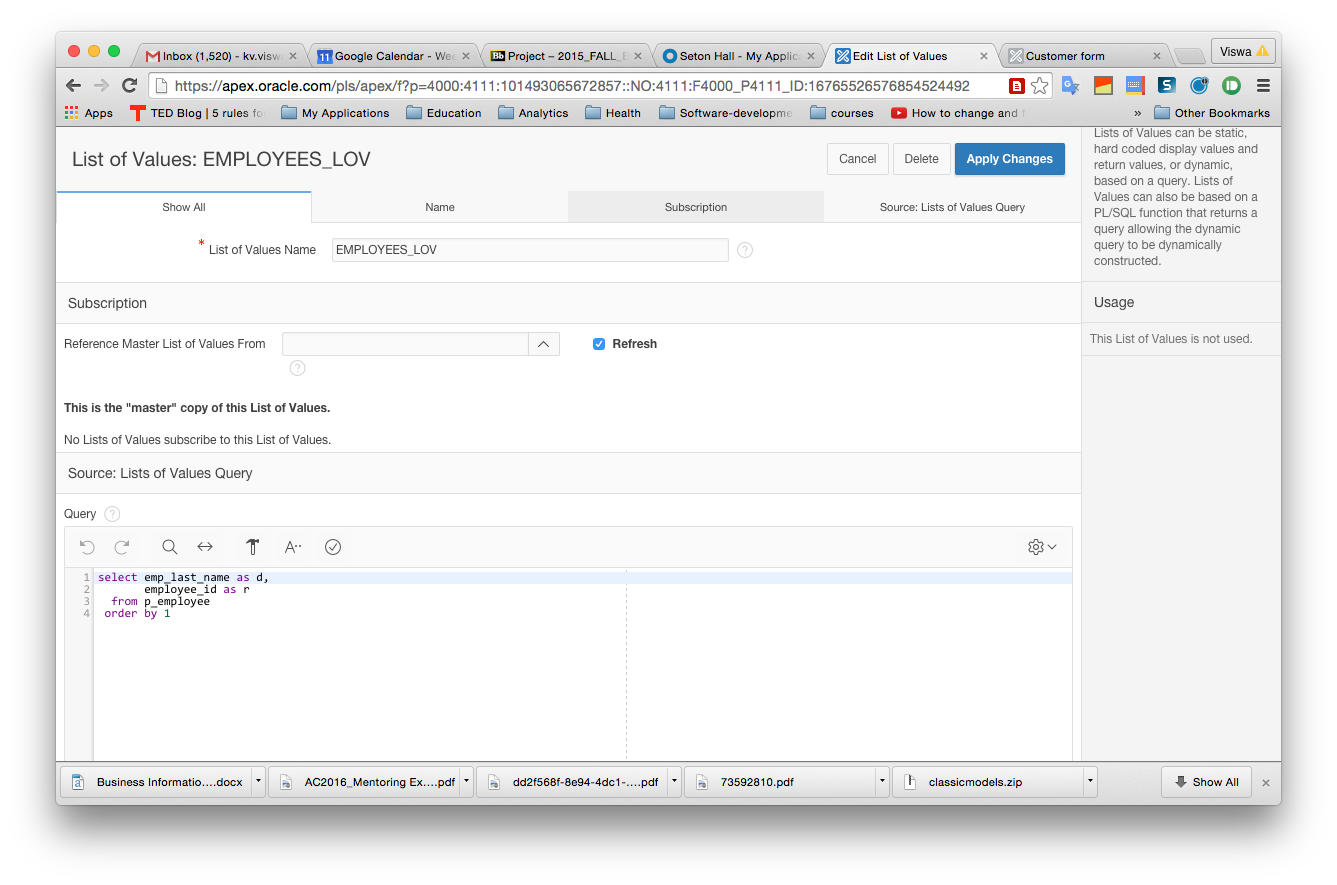


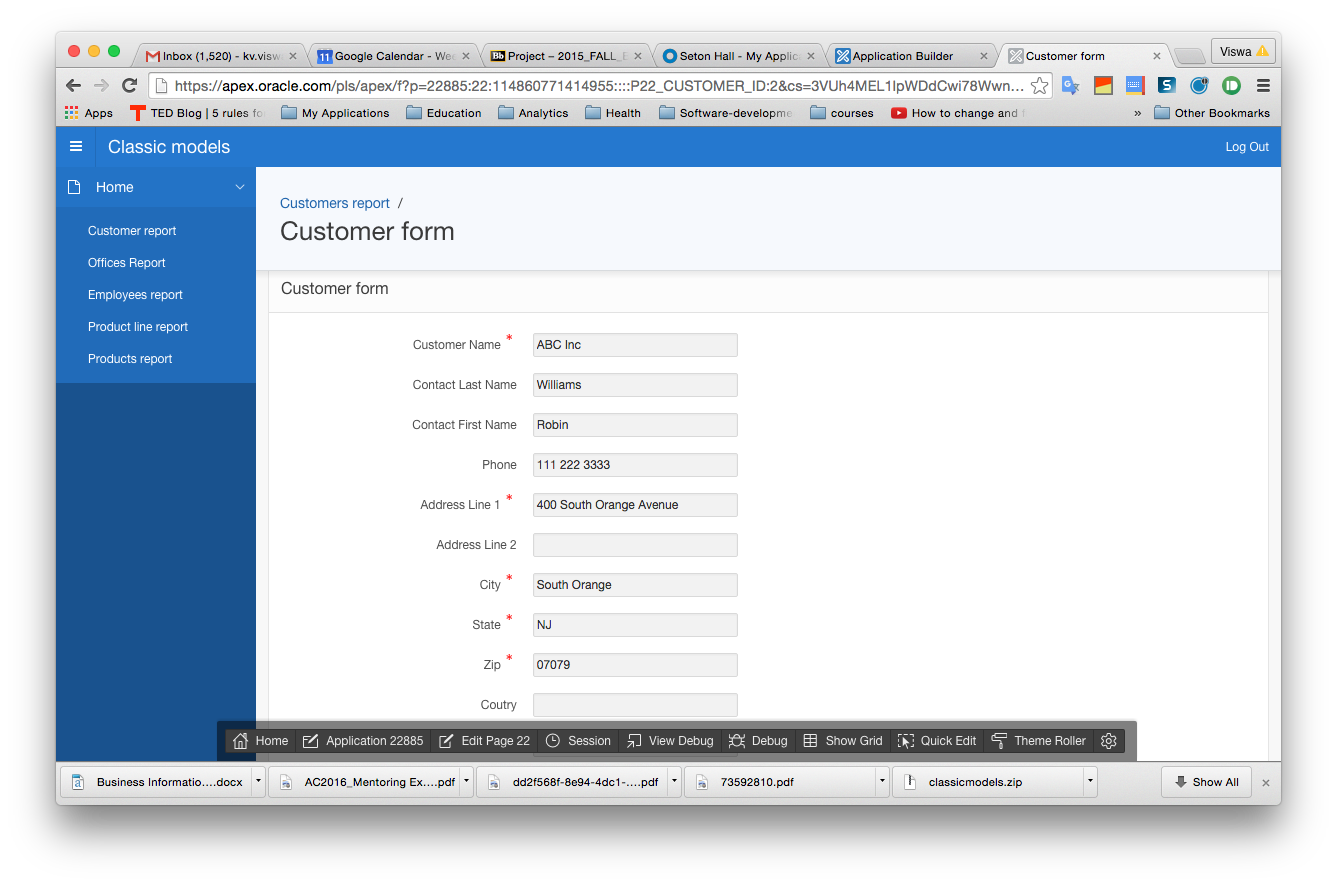
Figure 4: Entering the SQL for the LOV

The SQL query for a LOV always produces two columns as output. The first output column is what APEX displays on the screen and the second output column is what it actually stores in the underlying table. In the present case, the value we want to display is the employee’s name and what we want to store is the employee’s id. Note that we have simplified life by showing only the employee’s last name. Ideally we would like to show both the first and last names. This would mean that you will need to create a value that combines both of these. If you would like to do that, you can enter the following as the first line of your SQL instead of what I have shown above:  
  
SELECT EMP\_LAST\_NAME || ', ' || EMP\_FIRST\_NAME as d, EMPLOYEE\_ID as r

The above SQL has the effect of concatenating the last name a comma and the first name all as one single output column. Most of the time, you can manage with the simpler version that uses just one field.

Now click on *Create Lost of Values*. You will see the new LOV listed now.

1. **Attaching the LOV to the form field:** We can now attach the LOV to the *sales rep employee id* field on the customer form. To do this, we have to edit the definition of the form.
   1. One easy way to accomplish this is to run the application and get to the form and then edit it. To do that, click on the *Application xxxxx* link in the breadcrumb area.
   2. On the resulting screen, click *Run* to run your application. Click on the *Customers report* tab and then edit any individual customer – it does not matter which customer you choose. You now have something like Figure 5. Along the bottom of the screen, you see the developer menu, a way for developers to quickly make changes to some aspects of the current page of the running application. End users cannot see this menu. To make changes to some fields of the current page, one very quick way is to click on *Quick edit* on the developer menu. When you do that, the screen changes. You can then click on the text box for the "Sales Rep Employee ID" field to edit some of its properties.
   3. You get a new window that allows you to control lots of things. We will not look at the plethora of possibilities. The screen now looks like what is shown in Figure 6. In the area called "Identification" on the right hand side (highlighted in the figure), Note that the *Type* now shows as *Number field*. This is why the value is just showing up as a number – because the corresponding database field is a number. We want this to display as a list that will contain the LOV of employees. To achieve this, drop down the *Type* field and choose *Select list*. Then scroll down till you see "List of values" on the right. (See figure 7).



Developer menu

Figure 5: Customer form showing the developer menu

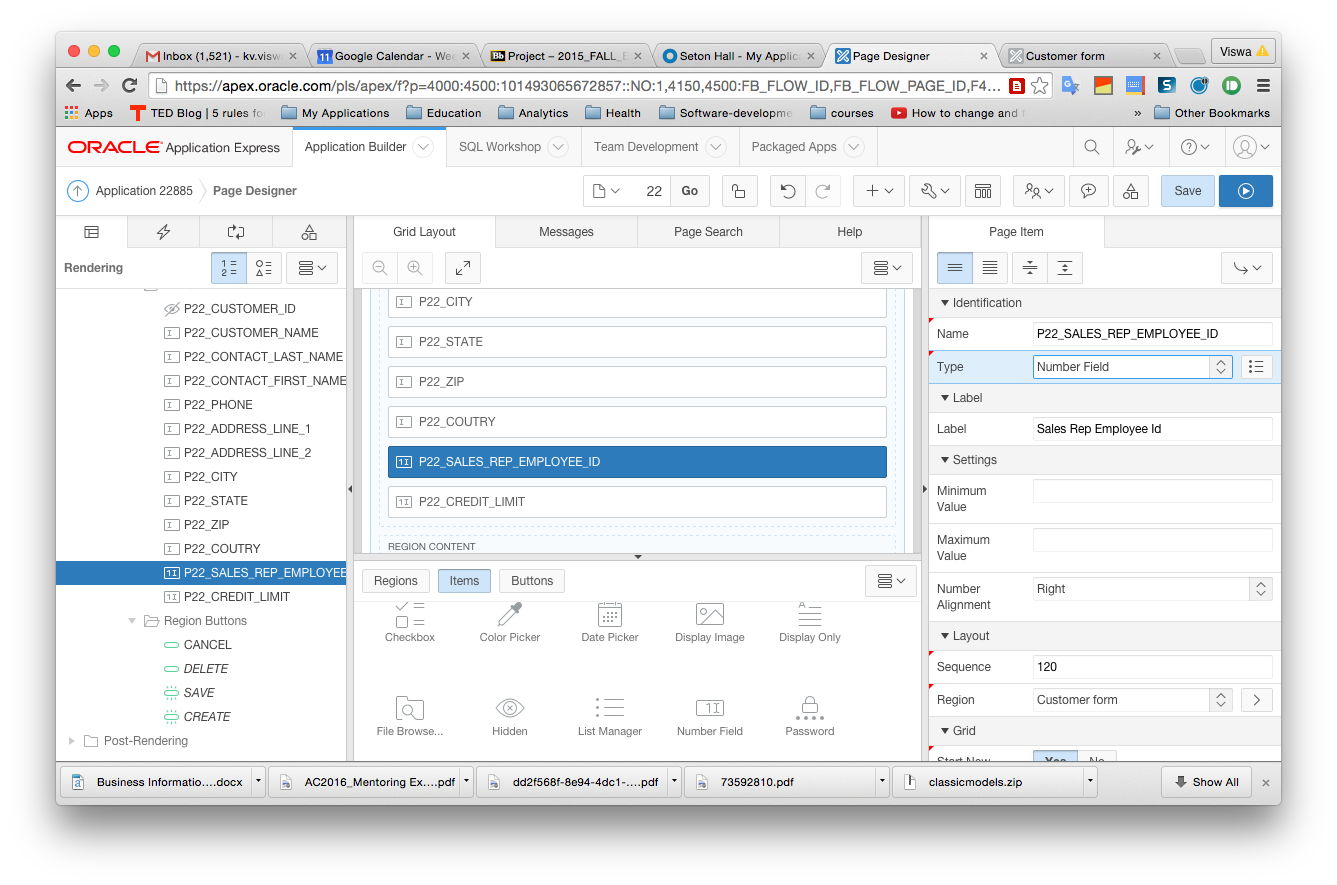


Figure 6: Converting the Sales rep employee id field to a Select list

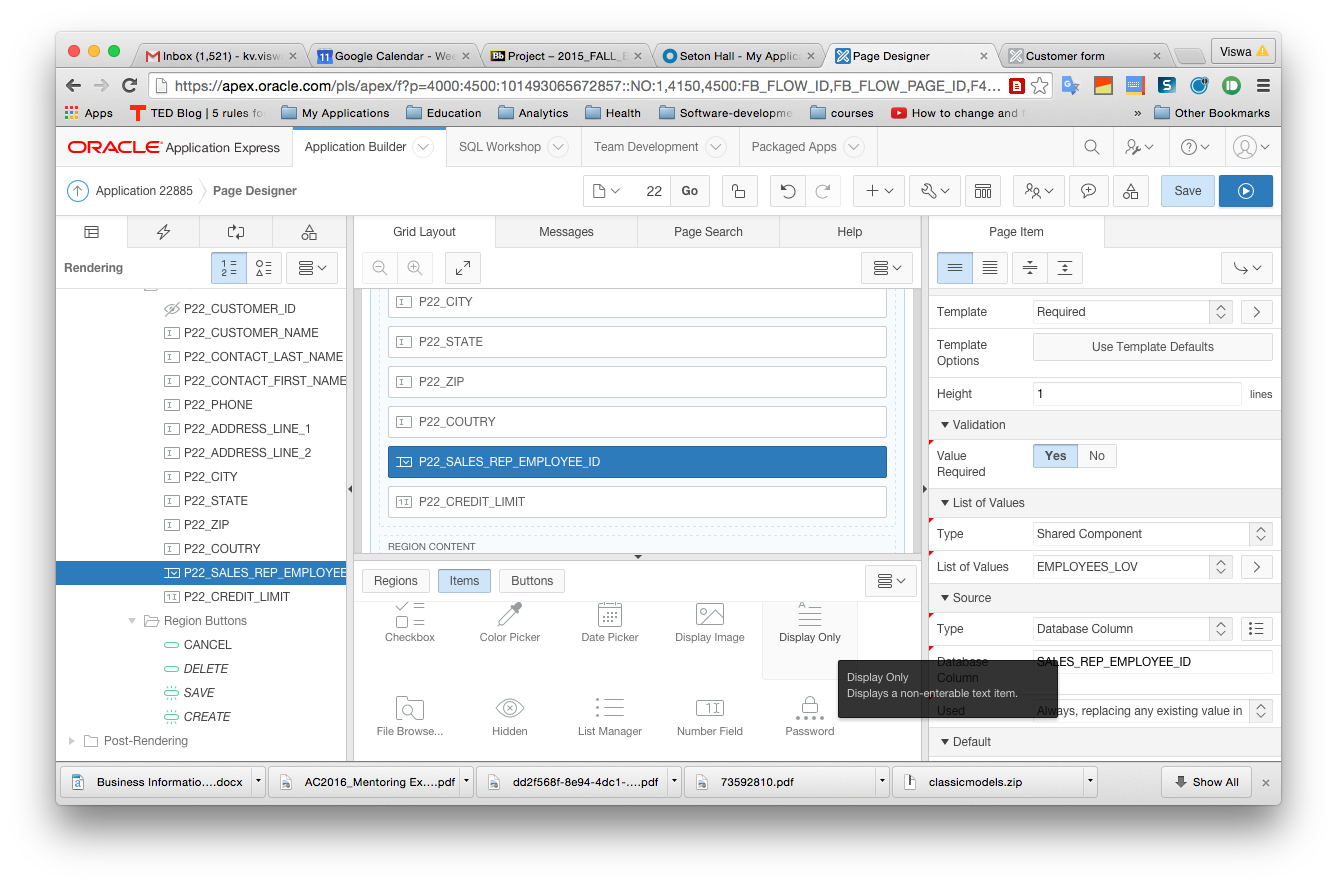


Figure 7: The List of values section

* 1. Under "Type" select "Shared components"
  2. Now we need to let APEX know which LOV to use as the list with this field. Click on *List of values* and select *EMPLOYEES\_LOV* (the one we created earlier). Now click on *Save* (on top right) to complete the process of changing the field from an integer field to one that will display a drop-down list.
  3. You are now ready to see the impact of all this work. Click the arrow butto next to "Save". This will run the page that we just edited. Note that the field "Sales Rep Employee\_id" now shows a drop down list instead of being a number. If we make a selection from the list, it will automatically insert the corresponding id into the table.
  4. Before we leave this, I want to ensure that you get some practice doing this. In your application, go to the *Employees report* tab and edit any employee. Note that the *Office id* field displays as a number. Why not display this as the city where the office is located? This field could also be a select list populated by a LOV. Before creating the LOV, make sure to enter a few records into the P\_Office table if you have not already done so. I list below the steps in outline for you to create and attach the LOV:
     1. Click on *Application xxxxx* in the developer menu at the bottom.
     2. Click on *Shared components*
     3. Click on *Lists of values*
     4. Click on *Create*
     5. Let *From scratch* be as is and click on *Next*
     6. Enter *Offices\_LOV* for the LOV name and select *Dynamic*. Click on *Next.*
     7. For the SQL, instead of entering it, you can make APEX do the hard work. Click on *Create dynamic list of values* right below the box with the sample SQL code. Select P\_*Office* as the table. Leave the *Table/View Owner* field alone and click *Next*.
     8. Select *CITY* as the *Display column* and *Office id* as the *Return value*. Click *Next* and then *Finish.* This will transfer the generated SQL into the appropriate place. Click *Create List of Values*. Now we have the LOV.
     9. Run the application and go to the form to edit a specific employee.
     10. Click the option at the bottom to show the edit links.
     11. Click the edit link for *Office id*.
     12. Complete the remaining steps by following the instructions given for the earlier example to change the field to a selection list and to attach the LOV to it.. (I do not want you to mechanically complete these steps and therefore am challenging you a little to apply the earlier instructions to a new case.)

1. **Creating a Master-Detail form:**  Generally we will need master-detail functionality for associative entity types. We will now create a master-detail form set for Sales Orders. P\_*Sales order* is the master table and p\_*Order details* is the detail table. It will be a good idea for you to go to the start of the document and look at the big picture to get an idea of what we are planning to do.
   1. Click on *Application xxxxx* in the developer menu at the bottom.
   2. Click on *Create page.*
   3. Click on *Form*.
   4. On the next page, select *Master-detail form*.
   5. Now we will be asked to first give the master table first (see the top where the current step is highlighted.). Select P\_*Sales Order* as your table. Immediately, APEX brings in all the columns of the table and also selects them all by moving them to the right. This is good for us. Click *Next.*
   6. Now we are being asked about the detail table. Select P\_*Order details*. Once again APEX does the smart things. Note that it showed P\_Order\_details as the only option. That is because that is the only table with which P\_Sales\_order has a 1:n relationship. Click *Next*.
   7. On the next screen select the radio button for "*Select primary key column"* for both the master and detail tables – you might need to scroll down to see the option for the detail table. Click *Next.*
   8. On the next screen, click *Next*. We do this because we have already created the primary key field as *Auto-increment* fields for the master table P\_*Sales order* and thus have all we need. It will look like nothing happened after you clicked. Actually you are being asked the same question again for the detail table – take a look at the table name. Click *Next* again*.*
   9. On the next screen, accept the defaults and click *Next.*
   10. On the next screen, select *edit detail as tabular form on same page* and click *Next*.
   11. On the next screen, select the option for Breadcrumb (you may need to scroll down); in *Entry name (Master report)* enter *Sales order;* in *Entry name (Master Detail page)* enter *Order* detail; click on *Home* for *Parent entry.* Click *Next*.
   12. On the next page, select *"Create a new navigation menu entry"* and enter Sales orders for *New navigation menu entry*. Select *Home* as the parent. Then click *Next.*  Then click *Create*.
   13. Test out the form now by running the application. You can do this in one of many ways. One way would be to select the *Run page* option that APEX displays as soon as it finishes creating the pages. Yet another is to click on *Application xxxxx* from the developer menu at the bottom and selecting *Run*. Test out the master-detail form. First invoke the "Sales orders" tab. Enter some of the details of the sales order and press create. This saves the order and shows it in the report. Edit the order and scroll down. Then press "Add row". Now you can enter a product id, quantity and unit price. However, as before we should be using an LOV for the product id so that the user can select products by name. Furthermore, we should also change the customer\_id field on the sales order to be an LOV as well.
   14. By now you already know that we need an LOV for this. Go ahead and create an LOV of products. The display field should be the product name and the return value should be the product code. Name this LOV as *Products*. Go ahead, you can do it. Feel free to look at the earlier steps and try out things. Don’t be too worried. If you make a mistake, you can always create a new one.
   15. Attaching this LOV to the field is a little different from what we did before because we are now attaching it to a report and not to a form. Run the application and get to the master detail page which displays the details for a sales order. You will note that clicking on *Quick edit* does not allow us to edit the product fields, but only the headings. We will get at this differently. Click on the option in the developer menu that says *Edit page xx*.This shows us a complex page indeed. There is a lot going on here; we see the structure of the entire report, and that is a lot more than we are ready for now. We will look at just what we need and glide smoothly past everything else.
   16. Right now, all we want is to get to the *Product\_code* field and attach our products LOV to it. In the leftmost column, click the small triangle sign near *Columns* under P\_*ORDRER\_DETAILS detail* (Figure 11). Then right click on *PRODUCT\_ID*. Now you can go to the right most area of the screen and do what we did earlier – make it a "Select list" and then attach the LOV. Thistime you will not have to scroll as much to get to the LOV part. Now when you run the page, you can see that the product name is showing up instead of the id.
2. We now have to pretty up the home page and we will be all done. We will not be very ambitious here. Rather than leave the page blank and dreary, we will just add some text. When we wanted to work on data from tables, we created forms and APEX took care of the rest. Now we want to just add some static text. We will do that in two steps. First add a new region to the page and then stuff some text into the region with some minor formatting.
   1. **Creating a new region on the home page:** Run the application once again. Go to the home page. Select the option *Edit page xx* from the developer menu at the bottom.
   2. Select *Regions*
   3. Right click and select *Create region*. This creates a new item called </> New. Selet it. On the extreme right area of the screen change the value of *Title*  to "Classic Models." Under *Text*, copy and paste the following HTML code and press "Save"  
      <br/><br/><br/><br/><br/><br/><center><b><font face="verdana" size=15 color="blue">Classic Models Inc.</font></B></center>

Now when you run the page, you will see a better home page.