

We are going to design and build our very first ETL mapping in OWB, but where do we get started? We know we have to pull data from the acme_pos transactional database as we saw back in topic 2. The source data structure in that database is a normalized relational structure, and our target is a dimensional model of a cube and dimensions. This looks like quite a bit of transforming we'll need to do to get the data from our source into our target. We're going to break this down into much smaller chunks, so the process will be easier.

Instead of doing it all at once, we're going to bite off manageable chunks to work on a bit at a time. We will start with the initial extraction of data from the source database into our target database without having to worry about transforming it. Let's just get the complete set of data over to our target database, and then work on populating it into the final structure. This is the role a staging area plays in the process, and this is what we're going to focus on in this topic to get our feet wet with designing ETL in OWB. We're going to stage the data initially on the target database server in one location, where it will be available for loading.

The first step is to design what our staging area is going to look like. The staging area is the interim location for the data between the source system and the target database structure. The staging area will hold the data extracted directly from the acme_pos source database, which will determine how we structure our staging table. So let's begin designing it.

Designing the staging area contents

We designed our target structure in topic 3, so we know what data we need to load. We just need to design a staging area that will contain data. Let's summarize the data elements we're going to need to pull from our source database. We'll group them by the dimensional objects in our target that we designed in topic 4, and list the data elements we'll need for each. The dimensional objects in our target are as follows:

- Sales

The data elements in the Sales dimensional object are: ° Quantity ° Sales amount

- Date

The data element in the Date dimensional object is: ° Date of sale

- Product

The data elements in the Product dimensional object are: ° SKU ° Name ° List price ° Department

° Category ° Brand

- Store

The data elements in the Store dimensional object are: ° Name ° Number ° Address1 ° Address2 ° City ° State

° Zip postal code ° Country ° Region

We know the data elements we're going to need. Now let's put together a structure in our database that we'll use to stage the data prior to actually loading it into the target. Staging areas can be in the same database as the target, or in a different database, depending on various factors such as size and space issues, and availability of databases. For our purposes, we'll create a staging area as a single table in our target database schema for simplicity and will use the Warehouse Builder's Table Editor to manually create the table.

This is the same technique we used to create metadata for the source structures in the ACME_POS SQL Server database back in topic 2. We'll get to use it again as we build our staging table.

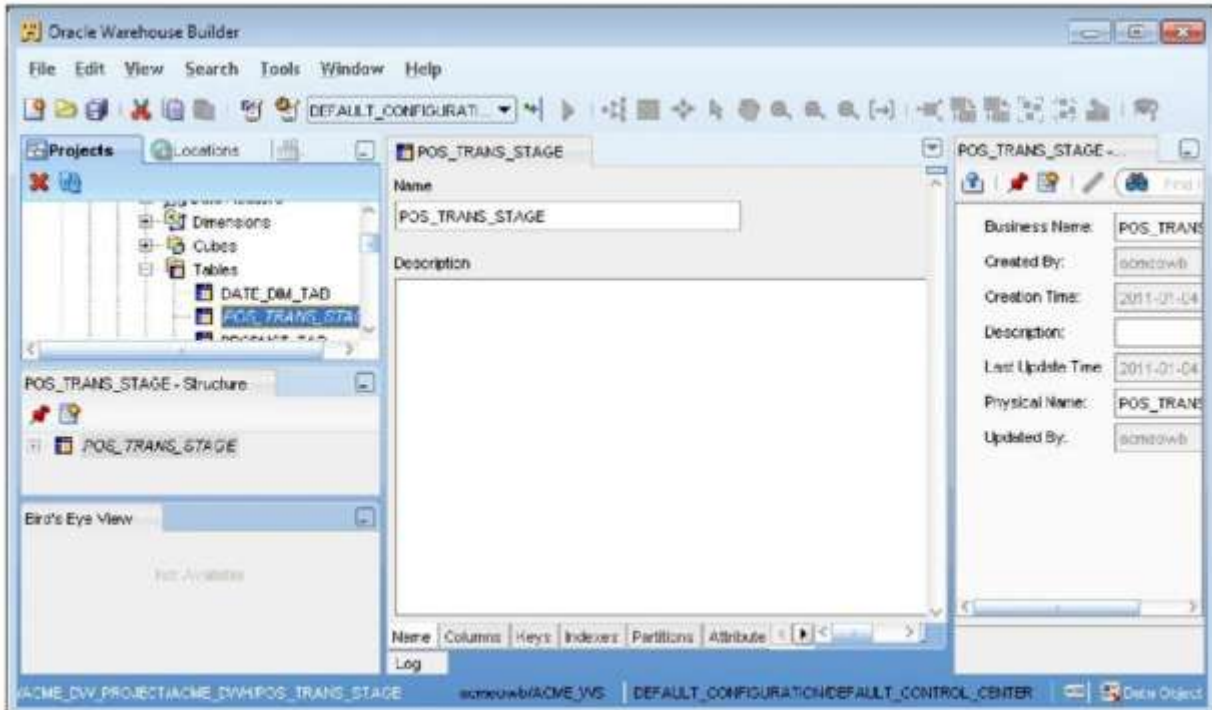
Building the staging area table with the Table Editor

To get started with building our staging area table, let's launch the OWB Design Center if it's not already running. Expand the acme_dw_project node and let's take a look at where we're going to create this new table. We've stated previously that all the objects we design are created under modules in the Warehouse Builder so we need to pick a module to contain the new staging table. As we've decided to create it in the same database as our target structure, we already have a module created for this purpose. We created this module back in topic 3 when we created our target user, acme_dwh, with a target module of the same name.

The steps to create the staging area table in our target database are:

1. Navigate to the Databases | Oracle | ACME_DWH module. We will create our staging table under the Tables node, so let's right-click on that node and select New Table from the pop-up menu. Notice that there is no wizard available here for creating a table and so we are using the Table Editor to do it.

2. Upon selecting New Table, we are presented with a popup asking us for the name of the new table and an optional description. Let's call it POS_TRANS_STAGE for Point-of-Sale transaction staging table. We'll just enter the name into the Name field, replacing the default TABLE_1 that it suggested for us. We'll click the OK button and the Table Editor screen will appear in our Design Center looking similar to the following:



This will look different depending on what windows are open. For example, the Bird's Eye View is visible, since the Mapping Editor was the last editor we were using and the Design Center will load windows from the last time it ran. We can just close any windows we don't need and resize any that we do.

3. The first tab is the Name tab where it displays the name we just gave it in the opening popup.

4. Let's click on the Columns tab next and enter the information that describes the columns of our new table. Earlier in this topic, we listed the key data elements that we will need for creating the columns. We didn't specify any properties of those data elements other than the name, so we'll need to figure that out.

One key point to keep in mind here is that we want to make sure the sizes and types of the fields will match the fields we want to pull the data from. If this is taken care of, we won't end up with any possible overflow errors generated by the database which could be caused by two character fields with different lengths for example.

Eventually, we know that we're going to have to use this new table that we're building as a source when we load our final target structure. This means we'll have to make sure our data sizes and types are compatible with our final structure also, and not just our sources. When we designed our target dimensions and cube, we made sure to specify correct sizes and types and so we shouldn't face any problem here. We can't change the source columns as they are fixed, which is another important consideration. The targets right now are only defined in metadata in the Warehouse Builder, so we can easily update them if needed.

The Warehouse Builder will actually tell us if we have a problem with the data types and field lengths when we use this table in a mapping either as a source or target table. It knows the size and type of the fields in the sources and targets because we imported or created tables to represent the sources, and it does a comparison internally. It will tell us if we're trying to map something too big for a field, or to a field of an incompatible data type. We don't want to have to wait until then to specify the correct size and type, so we'll create them accordingly now.

The following will then be the column names, types, and sizes we'll use for our staging table based on what we found in the source tables in the POS transaction database:

```
SALE_QUANTITY NUMBER(0,0)
SALE_DOLLAR_AMOUNT NUMBER(10,2)
SALE_DATE DATE
PRODUCT_NAME VARCHAR2(50)
PRODUCT_SKU VARCHAR2(50)
PRODUCT_CATEGORY VARCHAR2(50)
PRODUCT_BRAND VARCHAR2(50)
PRODUCT_PRICE NUMBER(6,2)
PRODUCT_DEPARTMENT VARCHAR2(50)
STORE_NAME VARCHAR2(50)
STORE_NUMBER VARCHAR2(10)
STORE_ADDRESS1 VARCHAR2(60)
STORE_ADDRESS2 VARCHAR2(60)
STORE_CITY VARCHAR2(50)
STORE_STATE VARCHAR2(50)
STORE_ZIPPOSTALCODE VARCHAR2(50)
STORE_REGION VARCHAR2(50)
STORE_COUNTRY VARCHAR2(50)
```

There are a couple of things to note about these data elements. There are three groupings of data elements, which correspond to the three dimensional objects we created—our Sales cube and two dimensions, Product and Store.

We don't have to include the dimensional object names in the data element names, but it helps to organize the data elements for eventual load into the target objects. This way, we can readily see which elements go where when the time comes to map them into the target.

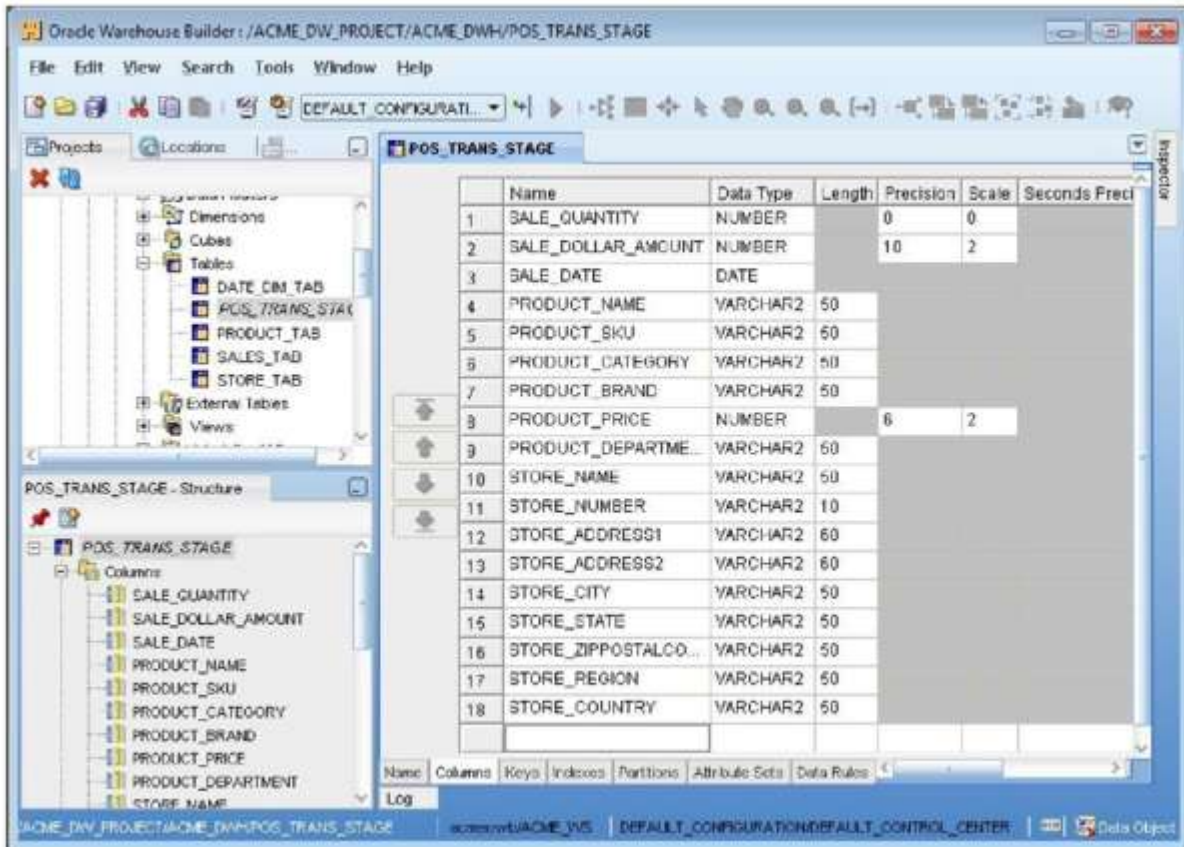
The second thing to note is that these data elements aren't all going to come from a single table in the source. For instance, the Store dimension has a store_region and store_country column, but this information is found in the regions table in the acme_pos source database. This means we are going to have to join this table with the stores table if we want to be able to extract these two columns.

We now have the information we need to populate the Columns tab in the Data Object Editor window for our staging table. We'll enter the above column names and types into the list of columns to complete the definition of our staging table.

Just as we saw back in topic 4 when entering column information for the product dimension, the Warehouse Builder attempts to make intelligent guesses of data types based on the name. That is actually controlled by a file containing regular expressions for various naming options and the data types, sizes and precisions to use. We can view that file to see what assumptions it is making and could even add our own entries or edit existing ones if we wanted. The file is in the owb\bin\admin folder under our Oracle home folder and is named Oracle_ItemDefaults.properties. Here is an example from the file for matching any column name that has the word NAME in it:

```
# Name Pattern
NameMatch = .*NAME.*
datatype = VARCHAR2
length = 30
```

When completed, our column list should look like the following screenshot:



The Property Inspector has been minimized and the Bird's Eye View window closed to make more room for the main editor window in the above image. Feel free to position windows in any manner that is most useful to you.

6. We'll save our work using the Ctrl+S keys, or from the File | Save All main menu entry in the Design Center before continuing through the rest of the tabs. We didn't get to do this back in topic 2 when we first used the Table Editor.

The other tabs in Table Editor are: • Keys

The next tab after Columns is Keys where we can enter any one of the four different types of constraints on our new table. A constraint is a property that we can set to tell the database to enforce some kind of rule on the table that limits (or constrains) the values that can be stored in it. There are four types of constraints:

- **Check constraint:** A constraint on a particular column that indicates the acceptable values that can be stored in the column.

- **Foreign key:** A constraint on a column that indicates a record must exist in the referenced table for the value stored in this column. We talked about foreign keys back in topic 2 when we discussed

the acme_pos transactional source database. A foreign key is also considered a constraint because it limits the values that can be stored in the column that is designated as a foreign key column.

° **Primary key:** A constraint that indicates the column(s) that make up the unique information that identifies one and only one record in the table. It is similar to a unique key constraint in which values must be unique. The primary key differs from the unique key as other tables' foreign key columns use the primary key value (or values) to reference this table. The value stored in the foreign key of a table is the value of the primary key of the referenced table for the record being referenced.

° **Unique key:** A constraint that specifies the column(s) value combination(s) cannot be duplicated by any other row in the table.

Now that we've discussed each of these constraints, we're not going to use any for our staging table. In general, we want maximum flexibility in storage of all types of data that we pull from the source system. Setting too many constraints on the staging table can prevent data from being stored in the table if data violates a particular constraint.

In this case, our staging table is a standalone table, so we don't have to worry about whether the data relates to any other tables via a foreign key. We want all the data available to our mapping, which will handle any transformations needed to make the data fit into the target system. So, no constraints are needed on this source staging table. In the next topic, we'll have an opportunity to revisit this topic and create a primary key on a table.

• **Indexes**

The next tab provided in the Table Editor is the Indexes tab. We were introduced to indexes at the end of topic 4 when we discussed the details displayed for a cube in the Cube Editor on the Storage tab. An index can greatly facilitate rapid access to a particular record. It is generally useful for permanent tables that will be repeatedly accessed in a random manner by certain known columns of data in the table. It is not desirable to go through the effort of creating an index on a staging table, which will only be accessed for a short amount of time during a data load. Also, it is not really useful to create an index on a staging table that will be accessed sequentially to pull all the data rows at one time. An index is best used in situations where data is pulled randomly from large tables, but doesn't provide any benefit in speed if you have to pull every record of the table.

Indexes are automatically created for us by the database in certain situations to support constraints. A primary key will have an index backing it up, consisting of the primary key column(s). A unique key is implemented with a unique index on the columns specified for the key. So if we were looking at creating indexes on a regular table, we would already have some if we'd specified these constraints. This is just something to keep in mind when deciding what to index in a table.

• **Partitions**

So now that we have nixed the idea of creating indexes on our staging table, let's move on to the next tab in the Table Editor for our table, Partitions. Partition is an advanced topic that we won't be covering here but for any real-world data warehouse, we should definitely consider implementing partitions. A partition is a way of breaking down the data stored in a table into subsets that are stored separately. This can greatly speed up data access for retrieving random records, as the database will know the partition that contains the record being searched for based on the partitioning scheme used. It can directly home in on a particular partition to fetch the record by completely ignoring all the other partitions that it knows won't contain the record.

There are various methods the Oracle Database offers us for partitioning the data and they are covered in depth in the Oracle documentation. Oracle has published a document devoted just to Very Large Databases (VLDB) and partitioning, which can be found

at http://download.oracle.com/docs/cd/E11882_01/server.112/e1654i/toc.htm.

Not surprisingly, we're not going to partition our staging table for the same reasons we didn't index it. So let's move on with our discussion of the Editor tabs for a table.

• **Attribute Sets**

The next tab is the Attribute Sets tab. An Attribute Set is a way to group attributes of an object in an order that we can specify when we create an attribute set. It is useful for grouping subsets of an object's attributes (or columns) for a later use. For instance, with data profiling (analyzing data quality for possible correction), we can specify attribute sets as candidate lists of attributes to use for profiling. This is a more advanced feature and as we won't need it for our implementation, we will not create any attribute sets.

• **Data Rules**

The next tab is Data Rules. A data rule can be specified in the Warehouse Builder to enforce rules for data values or relationships between tables. It is used for ensuring that only high-quality data is loaded into the warehouse. There is a separate node—Data Rules—under our project in the Design Center that is strictly for specifying data rules. A data rule is created and stored under this node. This is a more advanced feature. We won't have time to cover it in this introductory topic, so we will not have any data rules to specify here.

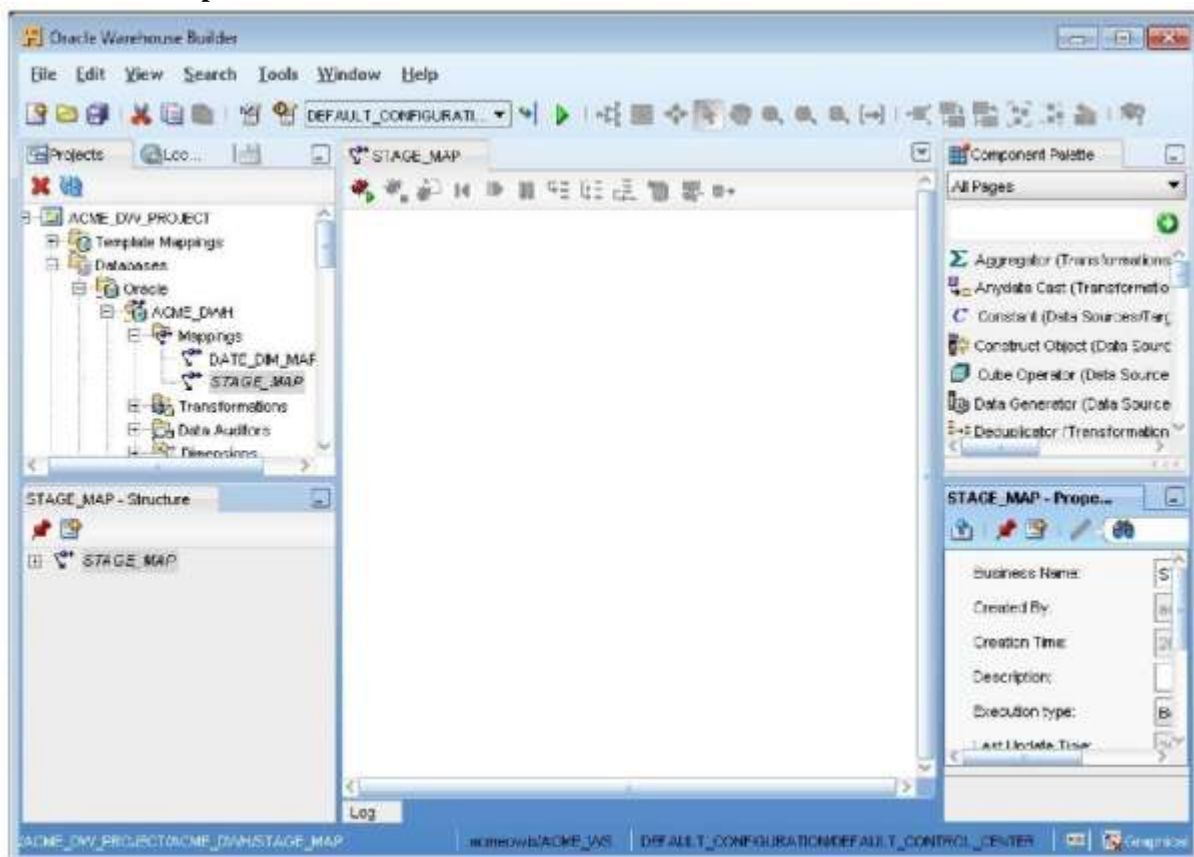
This completes our tour through the tabs in the Table Editor for our table. These are the tabs that will be available when creating, viewing, or editing any table object. At this point, we've gone through all the tabs and specified any characteristics or attributes of our table that we needed to specify. This table object is now ready to use for mapping. Our staging area is now complete so we can proceed to creating a mapping to load it. We can now close the Table Editor window before proceeding by selecting File | Close from the Design Center main menu or by clicking on the X in the window title tab.

Now that we have our staging table defined, we are now ready to actually begin designing our mapping. We'll cover creating a mapping, adding/editing operators, and connecting operators together, but first lets do a quick review of the Mapping Editor.

Review of the Mapping Editor (Oracle Warehouse Builder 11gR2)

We were introduced to the Mapping Editor in the last topic and discussed its features, so we'll just briefly review it here before using it to create a mapping. We will create mappings that are in the Design Center under an Oracle Database module. In our case, we have created an Oracle Database module called acme_dwh for our target database. So this is where we will create our mappings. In Design Center, navigate to the ACME_DW_PROJECT | Databases | Oracle | ACME_DWH | Mappings node if it is not already showing. Right-click on it and select New Mapping from the resulting pop-up. We will be presented with a dialog box to specify a name and, optionally, a description of our mapping. We'll name this mapping STAGE_MAP to reflect what it is being used for, and click on the OK button.

This will open the Mapping Editor window for us to begin designing our mapping. An example of what we will see next is presented here:



Unlike our first look at the Mapping Editor in the last topic where we looked at the existing DATE_DIM_MAP, all new mappings start out with a blank slate upon which we can begin to design our mapping.

By way of comparison with the data object editors, the Mapping Editor has a blank area named Mapping which is the canvas. The Table Editor did not have a graphical canvas and the cube and dimension editors have a graphical depiction only on the Physical Bindings tab but were otherwise primarily text based. The mapping canvas performs the function of viewing and laying out objects. The other windows we can see above are the same ones we saw when using data object editors in the last topic but now pertain to the Mapping Editor.

Although the data object editors are not graphical, there is a graphical way to view the objects provided by what is called the Graphical Navigator which can be opened from the View main menu. It provides a blank canvas onto which we can drag and drop any data object to get a graphical depiction of the object and any objects related to it. Dropping a cube for instance, will display the cube with any dimensions that it references. This is the functionality that was in the Data Object Editor in the previous release of the Warehouse Builder on the dimensional and relational tabs that used to be in the editor.

The Mapping Editor uses the Structure window similar to the data object editors.

This window performs the same function for viewing operators/objects that we've already defined in our mapping. In the previous release this information was in the old Explorer window on the Selected Objects tab.

The Properties Inspector window is for viewing and/or editing properties of the selected element in the Mapping window. Right now we haven't yet defined anything in our mapping, so it's displaying our overall mapping properties.

The Component Palette displays all the operators that can be dragged and dropped onto the Mapping window to design our mapping. The objects in this case are specific to mappings, so we'll see all the operators that are available to us.