What we just saw was a brief review of the Mapping Editor. Now let's begin to use it to design our mapping of the staging table. In designing any mapping in OWB, there will be a source(s) that we pull from, a target(s) that we will load data into, and several operators in between depending on how much manipulation of data we need to do between source and target. The layout will begin with sources on the left and proceed to the final targets on the right of the canvas as we design it. Right now we know that we have to pull data from the acme_pos transactional database in SQL Server as our source and load it into the pos_trans_stage table that we just defined as our target. So let's begin by including these objects into our mapping.

We need to look at the source data and determine what tables we will need to pull the data from so that we know which of the objects to include in our mapping. We first looked at the source data for the POS transactional database back in topic 2. So if you need to refresh your memory about what that looked like, now would be a good time to go back and review that quickly before moving on. In the next section, we'll start by adding a source table.

Adding source tables

We know that the first piece of information we need for loading into our staging table is the sales data—the quantity and dollar amount of each sale, and the date of the sale. Looking at our acme_pos source database, we know that data is stored in the POS_Transactions table. Therefore, we'll start our mapping by including this table.

There are a couple of ways we can add a table to our mapping. One way is to use the Projects Navigator window and the other way is to use the Palette window. Which one we choose is really just a matter of preference.

We'll use the Projects Navigator window to find the table that we want to include in our mapping. To find an object in the Projects Navigator, we have to know what module it is located under.

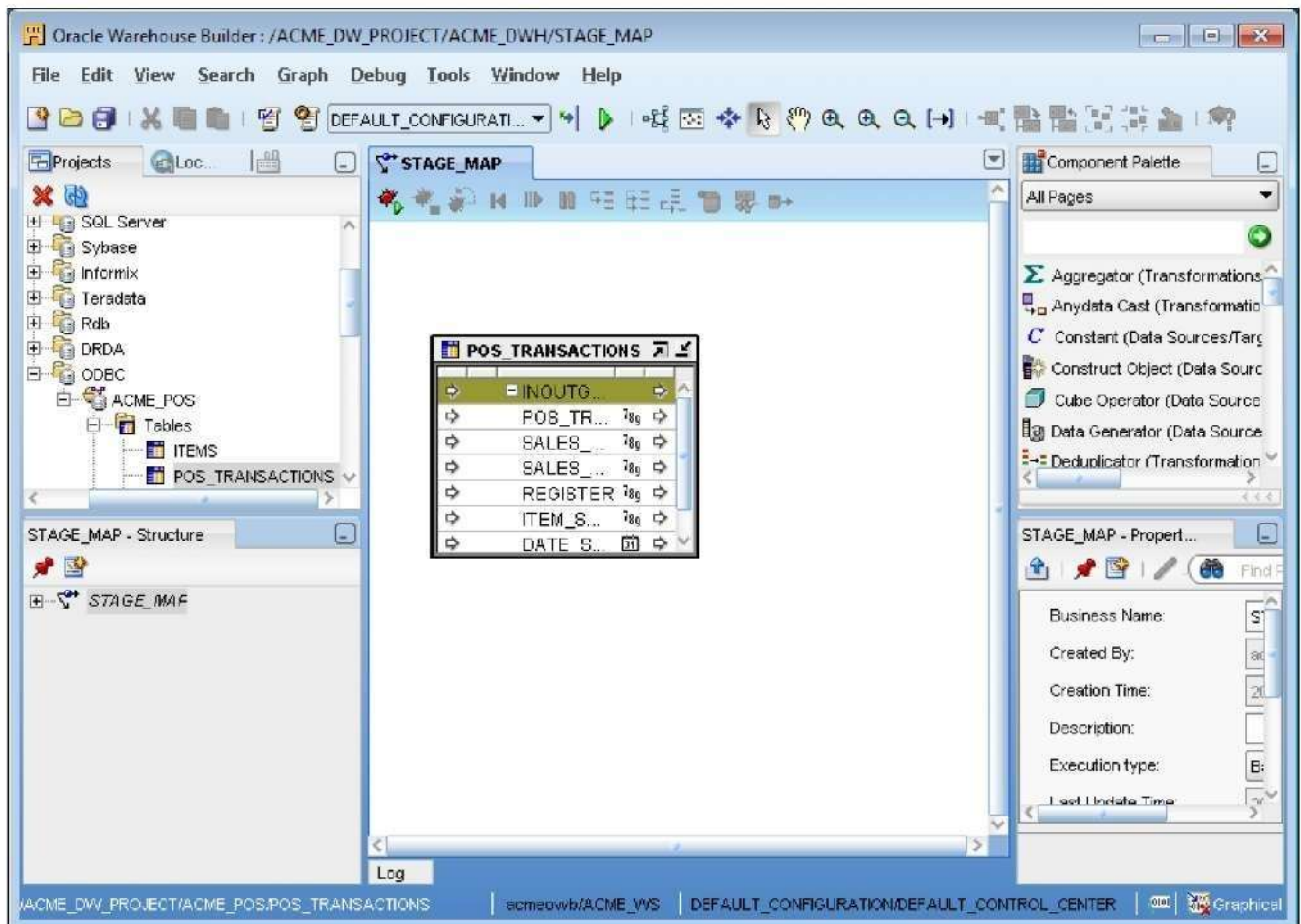
Collections

Our project is not that big so it's easy to find objects but it's easy for projects to become very large. One feature we're not covering in this introductory topic that can assist with that is Collections. That feature allows you to group objects from your project into arbitrary folders for ease in accessing them and for organizing them. For more information, consult the Oracle Warehouse Builder Concepts Guide, topic 3, at the following URL:

http://download.oracle.com/docs/cd/E11882_01/owb.112/e10581/uitour.htm#BABDCHCJ

In our case, we know the POS_TRANSACTIONS table is defined under the ACME_POS module. So let's navigate to the Databases | Non-Oracle | ODBC | ACME_POS node in the Projects Navigator tab to find the POS_TRANSACTIONS table entry.

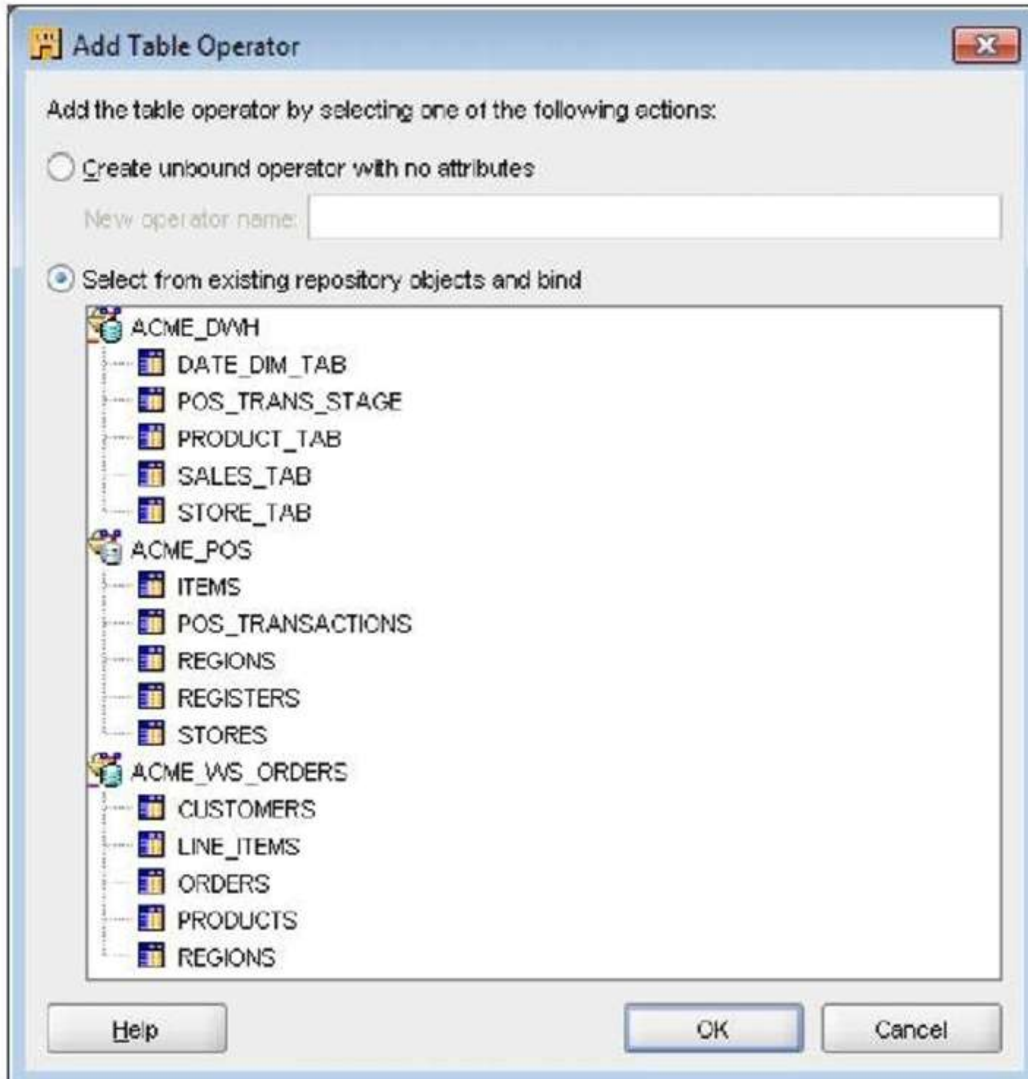
We can also find things very quickly with the Design Center search function by clicking on the Search main menu entry and selecting Find... or by selecting Ctrl-F from the keyboard. Just enter the name of the object we're searching for and click the Find button and it will take us right to it. That's very helpful if we can't remember which module we created the object under.
Click and hold the left mouse button on POS_TRANSACTIONS, drag it over to the Mapping window, and release the left mouse button to drop the table into our mapping. Our Mapping Editor window should now look similar to the following:



There are a couple of items to note about how the Mapping Editor window looks. The Properties Inspector window no longer shows the mapping information. It has changed to show the properties of the POS_TRANSACTION table as it is now highlighted in the Mapping canvas window.

If your Properties window does not show the POS_TRANSACTIONS properties, simply click on the POS_TRANSACTIONS operator in the Mapping window. Make sure you click on the title bar of the operator window because if you click inside the window, it will select one of the attributes or groups and display the properties for that instead of displaying the properties for the operator as a whole.
Another item to note is that now we have an object in our Mapping window instead of a blank canvas. These objects that make up a mapping are called operators. In this particular case, it is a Table operator that we have placed into the mapping to represent the POS_Transactions table.

Having just one operator in our mapping is not enough, so let's try including the remainder of the tables we'll need from our source database. We clearly need some product information to fill in. From our analysis in topic 2 of the acme_pos source database structures, we know that product information comes from the Items table. We'll include that table in our mapping now, but instead of using the Projects Navigator window as we did for the POS_TRANSACTIONS table, let's see how the Component Palette window works for including objects into our mapping.
The operators in the Component Palette are sorted alphabetically, so we'll scroll the window until we see the Table Operator. Click and drag the Table Operator from the Component Palette window onto the Mapping window. As soon as we drag it onto the Mapping window, we are presented with a pop up like the following screenshot:



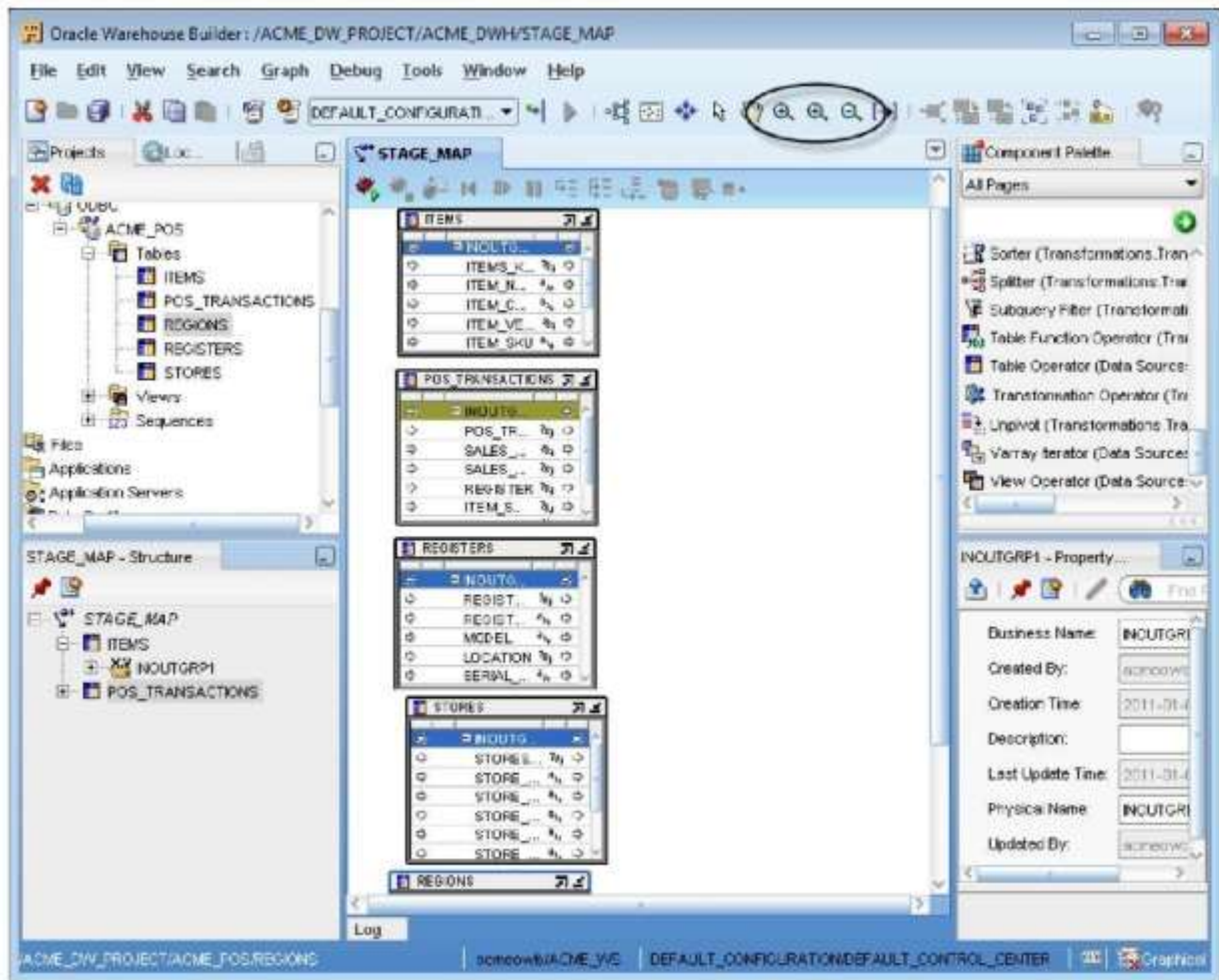
This pop up asks us which table we want to include as this table operator. We have a couple of options offered to us. We can create an unbound operator, which has no attributes; in other words, it is a blank table that we can define as we like, or we can specify an existing table from our project. An unbound operator is one which is not associated with (that is, bound to) an existing database object. The act of binding in OWB associates a generic operator with an actual defined object in the project. When we dragged our pos_transactions table from the Projects window, it did not ask us about this because we started with a specific named table. The operators in the Component Palette window are all generic and are not associated with any specific object of that type. With unbound operators, you can actually use the Mapping Editor to create a data object. We'll actually get to do this in the next topic when we have to create a lookup table.

We're going to stick to the objects already defined for our operators. So we're going to select the ITEMS table under the ACME_POS entry in the list of table names that the pop-up window presents to us. We will click on the OK button to include the ITEMS table operator in our

mapping. Notice how the Add Table Operator dialog box presents the information to us. It lists every possible table in our project and organizes them by module. acme_dwh is our main data warehouse module that we created for our target in order to build our data warehouse. We can see the tables that were created for our cube and dimensions along with the staging table we created. acme_ws_orders is the web site order's database that we imported for source data from the web site, and acme_pos is what we're working with right now to build a staging area.
Let's talk about organizing our tables in the Mapping window before we go any further. In general, it's always a good idea to place source operators on the left and the target operators on the right. So let's just make sure we keep the tables we're dropping into our Mapping window towards the left side of the window, one above the other. Click and hold on the header of the operator to drag it around.

We've seen how to include a table operator in our mapping using either of the two methods. Using either one now, we'll include the remainder of the source tables that we're going to need into our mapping—the registers, stores, and regions tables. It should be clear why we need the stores and the regions tables. These tables contain information about each store that we'll need, including the address, region, and country. But why do we need the registers table? We're not going store any information about the register used. From our analysis back in topic 2, we saw that the register information pointed to the store where the register was located in and the main pos_transactions table only had a foreign key column for the register—not the store or region. This information was kept in separate tables with a foreign key to the store, which is stored in the registers table. So, if we hope to be able to retrieve the store and region information out of the source database, we're going to need the registers table to get there.
Now go ahead and include those three tables into the mapping using either of the two methods we just discussed. When that is completed, make sure the tables are organized vertically on the left side of the mapping window in the following order: the items operator on top, then pos_transactions, then the registers operator next, then the stores operator, and then the regions operator at the bottom. We'll see soon why we are paying attention to the order in which we display the operators in the Mapping canvas.
You'll notice that while dragging objects around the Mapping window, it will grow in size automatically to hold the objects we are placing there. So we needn't be too concerned if our source tables are not all the way over to the left. When we place our target table, we'll put it to the right-hand side of the sources. Just make sure that the source tables are all together.

Your Mapping Editor window should now look similar to the following:



The five source tables may not all be visible at once as we saw in the previous screenshot. The Mapping view has been zoomed out, so mostly all are visible here. However, there is a trade-off in readability the more we zoom out. You can manipulate the zoom yourself to find a size that's comfortable for your viewing. The zoom buttons in the toolbar have been circled in the screenshot we just saw. Click on the magnifying glass with the plus sign to zoom in and the minus sign to zoom out. There are two magnifying glasses with a plus sign. The one on the left is an interactive zoom and the one on the right is the regular zoom in button.
Now that we have our source table all included and laid out in the Mapping window, we'll move on to discuss getting our target included in the mapping.

Adding a target table

Let's now turn our attention to the target for this particular mapping. As this is a staging-related mapping, we're going to be loading our staging table and so that will become our target. Let's find the POS_TRANS_STAGE table in the Projects window. We'll navigate to Databases | Oracle | ACME_DWH | Tables | POS_ TRANS_STAGE in Projects, and click and drag the

POS_TRANS_STAGE table to the righthand side of our source tables in the Mapping window. Let's leave some space between the source table and the target tables. We'll shortly see the reason behind this when we start connecting our source to the target.

Connecting source to target

The process of connecting the source to the target is the means of telling the Warehouse Builder which data fields from the source go in which data fields in the target. We might be tempted to just connect the data fields from the source tables directly to the corresponding fields in the target. For instance, we know that the items table has the item_name field, which needs to be stored in the target in the product_name column; so why can't we just connect the two directly?

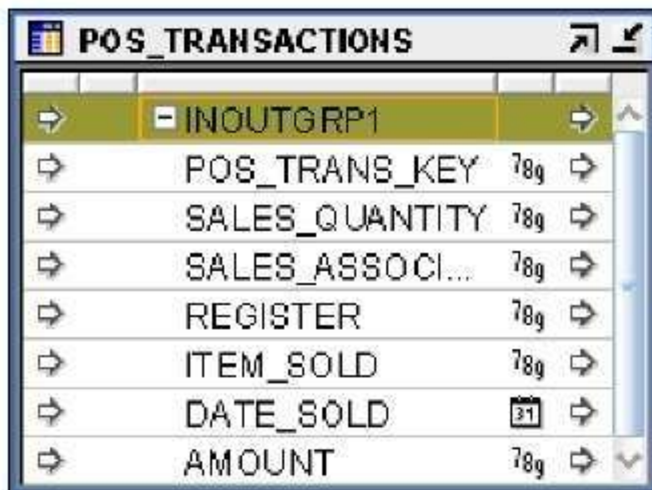
The reason we can't connect the two directly is because we have to keep in mind what that means in terms of mapping. If we just connect a line from a source table attribute to a target table attribute, we're telling the Warehouse Builder that there is a one-to-one mapping from source to target. This means we can read a record from the source table and store the column values directly in the target table with no need of further manipulation. The problem in our case is that we're including information in our target table from multiple source tables, and the Warehouse Builder is not going to let us connect multiple source tables to a single target table directly as it won't know how to combine the data.

In addition to the joining of the tables, connecting directly from source to target would also imply that the data was at the granularity we need. We discussed granularity (or the level at which the data is stored) in topic 3 and decided that our warehouse would store the data by product, store, and date. However, looking at our source data, we can see that the data is actually stored by register. This is actually a lower level of detail than we need, so we'll need to sum up the data for each register in a store to get the total for the store.
This means we will have to provide some kind of intervening operators to get the data combined from the five source tables into the one target table, summed by product, store, and date. But what operators are we going to use? Remember our discussion in topic 5 that introduced us to the various operators available in the Warehouse Builder. We particularly mentioned in the Transformations section that directly connecting source to target would only work for a one-to-one mapping between a source table and a target table. Then went on to discuss some of the operators that can be used for data flows, and one of them was a Joiner operator. If you re-read the explanation, you'll see that a joiner is exactly what we need in this case because we have to take multiple source tables and combine (or join) them into one record in the target. We also discussed an Aggregator operator that can be used to aggregate data. In this case, we need to sum data at a higher level before storing it. So this should be exactly the operator we need for that purpose.

Now that we've settled on the two data flow operators we need, let's place them into our mapping between the sources and the target. Now you can see why we left some space between the sources and the target. Scroll down through the Component Palette window until the Joiner operator is visible, drag this operator into the Mapping window, and drop it between the sources and target.

Joiner operator attribute groups

We were introduced to the concept of attribute groups in the last topic when we were looking at date_dim_map in the Mapping Editor. It's time to talk about attribute groups again, because we can see that the Joiner operator has three groups defined, but the attributes in our table operators are all in one group. The groups in operators we saw are generally input groups, output groups, or both.
In our table operators we can see that there is only one group as we mentioned, called INOUTGRP1. The following screenshot is an example of that using the POS_ TRANSACTIONS table operator:



There are actually two clues to identify the attributes that can be used for both input and output: one is the name of the group, which has both IN and OUT in it if the default names have not been changed, and the second is the little arrows that appear on each attribute line — one arrow on the left pointing in for input and one on the right pointing out for output.

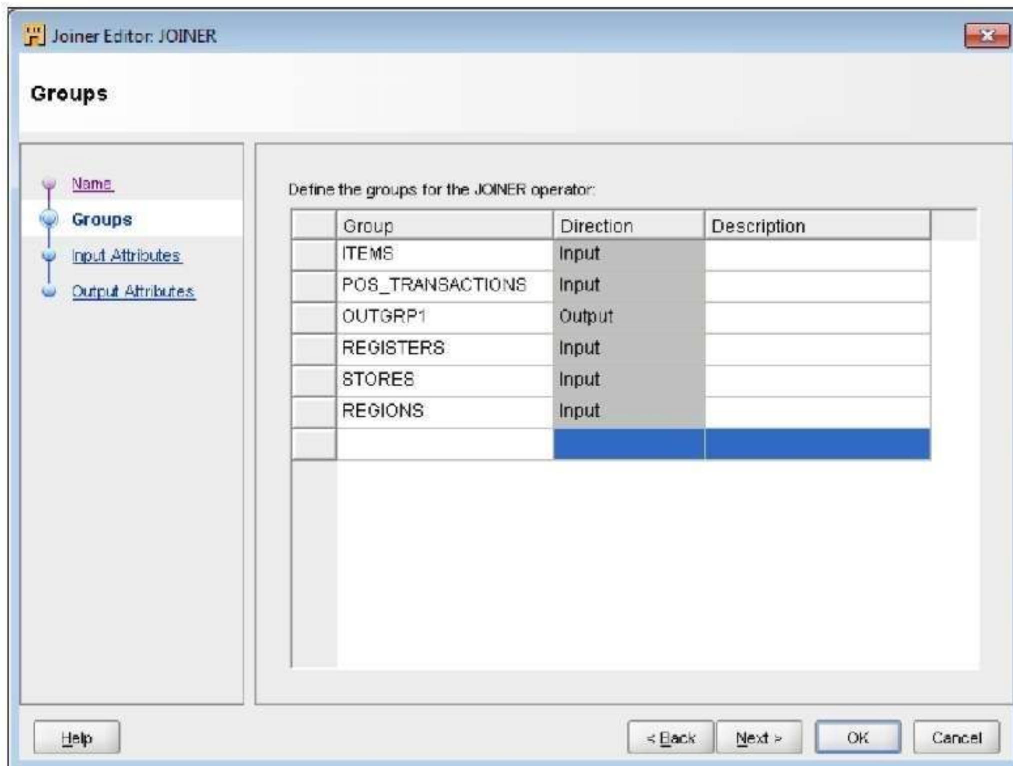
If an attribute is in an input group, then we can connect an attribute from another operator on the left to let the data flow from that attribute to this one. These connections always enter an operator from the left. Now we can see why it's a good idea to put our sources on the left and targets on the right. This is the direction the data flows through the operators.
If the attribute can be used for output in either an output group or an in/out group, then it means the data from the attribute can be used as input into another operator. Output from an operator always flows out from the right side of the operator.

If we look at the Joiner operator we just dropped into our mapping, we can see that there are no attributes defined in any of the three groups. Before we add attributes, let's talk briefly about the groups in a Joiner operator. By default, the operator is created with two input groups and one output group. Each input group corresponds to a separate table or other data operator, and the output group represents the combined (joined) output from the input tables.

We have five source tables to join together, but this Joiner operator has only two input groups. Have no fear; a Joiner can have more than two input groups. We have to edit this Joiner to add three more input groups. To edit it, right-click on the header of the box and select Open Details... to open the Joiner Editor or just double click on the header . This dialog box will allow us to edit the number of groups as well as change the group names if we want something different from INGRP1 and INGRP2.
The Joiner Editor can be used to edit not only the groups, but also the attributes that compose each group. So if we right-click inside the Joiner box on a group and select Open Details..., we will get the same dialog box with just the individual tab selected that corresponds to the group we clicked on.
With the Joiner Editor open, let's click on the Groups entry on the left. We'll add three new groups by typing their names in the empty box at the bottom of the Group column on the right. We could use ingrp3, ingrp4, and ingrp5 as the group names for our new groups but lets go ahead and use the actual table names. Enter registers then stores and then regions and hit the enter key after each. Notice how it automatically populated the group type as INPUT and doesn't allow us to change it. This means it knew we wanted input groups and not output groups. Did it read our minds? Well, not exactly. As it turns out, Joiners can have only one output group and no combined input/output groups. So the only kind of group left to add when we enter a group name is an input group.
While we're on the Groups tab, let's modify the names of our two default input groups to the other two table names. By just clicking on the name of the input group, we can type in a new name. So, let's rename each of those default input groups as follows:
• INGRP1 to ITEMS

• INGRP2 to POS_TRANSACTIONS

After entering three new names and renaming the two default names this is how it will look:



Now we'll click on the OK button to close out the Joiner Editor dialog box.

You may be wondering if there is a reason why we picked a particular sequence of input tables and paid attention to the order in which we displayed the table operators on the canvas. It's to match the sequence of tables that we set up in the Mapping window from top to bottom as displayed in the previous image. However, this is not a hard and fast requirement; they could be in any order. To ensure a good appearance, that is, keep lines from crisscrossing all over the mapping when we're done with it, it's a good idea to plan ahead and use the same order. We'll see this later.

Also, we don't have to worry about capitalization as the editor will automatically put everything we type into uppercase.

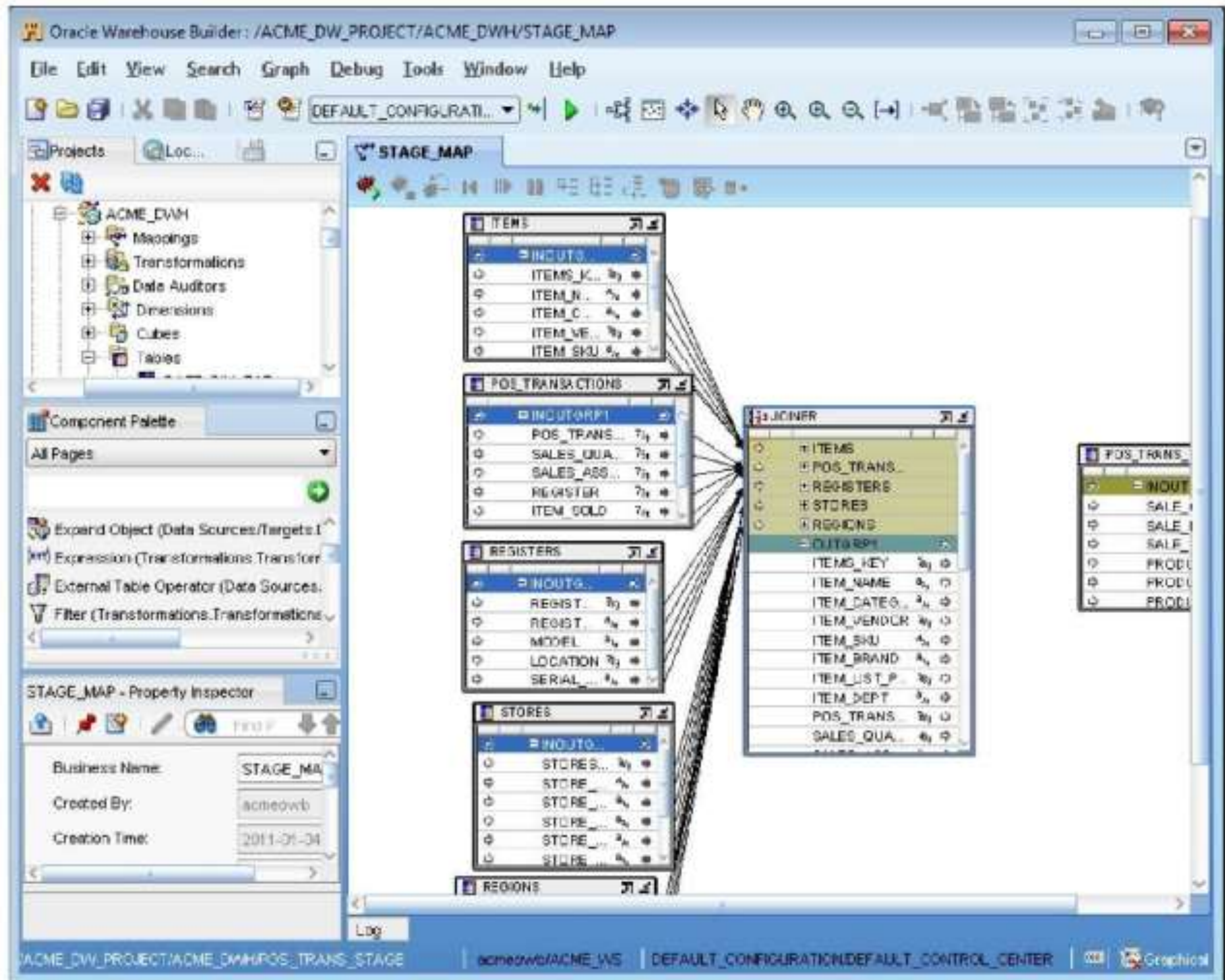
Connecting operators to the Joiner

Now that we have our Joiner groups defined, it's time to start making some connections between operators. The act of connecting operators is a matter of clicking and dragging a line from an output attribute of one operator to an input attribute of another operator, or from one output group in one operator to an input group in another operator. If we connect two attribute groups together, we're telling the Mapping Editor to go ahead and connect every attribute in the group. If we have several attributes, this is a convenient way to connect them. So, click and drag INOUTGRP1 of the ITEMS table operator onto the ITEMS group of the JOINER. Immediately, it will add all the attributes from the ITEMS table to the ITEMS group in the JOINER and connect each one with a line.

Alternatively, we could have clicked and dragged a line from each attribute in the ITEMS table and dropped it on the ITEMS group in the JOINER. But it was quicker and more efficient to drag the entire group even though we won't be using every attribute.
Leaving attributes in the JOINER that we're not going to use will not affect the final result of our mapping. We're just going to drag the attributes we're going to need over to the target table in a moment. OWB will ignore the attributes we don't use when it builds its underlying code.

Notice that if we now scroll down our JOINER operator to where the OUTGRP1 is visible, we can see that it automatically added attributes to the output group corresponding to each of the input group attributes.

To better view the attributes in the JOINER operator, we can make the box bigger by clicking and dragging the border of the box to a bigger size, either vertically or horizontally. Also, if we don't want to see all the attributes, we can click on the minus sign on a group to collapse it so that the attributes are not visible. Clicking on the plus sign then restores the attributes.
Now let's repeat the same procedure with the POS_TRANSACTIONS, REGISTERS, STORES, and REGIONS tables. That is, let's drag the INOUTGRP1 group to the corresponding group in the JOINER for each table to connect them. Feel free to click on the minus sign on each group to collapse it to get a better view of the next group. If the ordering of the tables has been maintained between the JOINER groups and the table operators, and all the input groups of the JOINER have been collapsed (which is why the attributes are not visible), the mapping should look similar to the following screenshot:



The Component Palette and Property Inspector windows have been moved over to the left side and the Structure window closed to make more room for the main canvas area of the Mapping Editor. Your overall window will probably look different but the important part is that your main canvas should look similar to how the objects are laid out above.

Defining operator properties for the JOINER

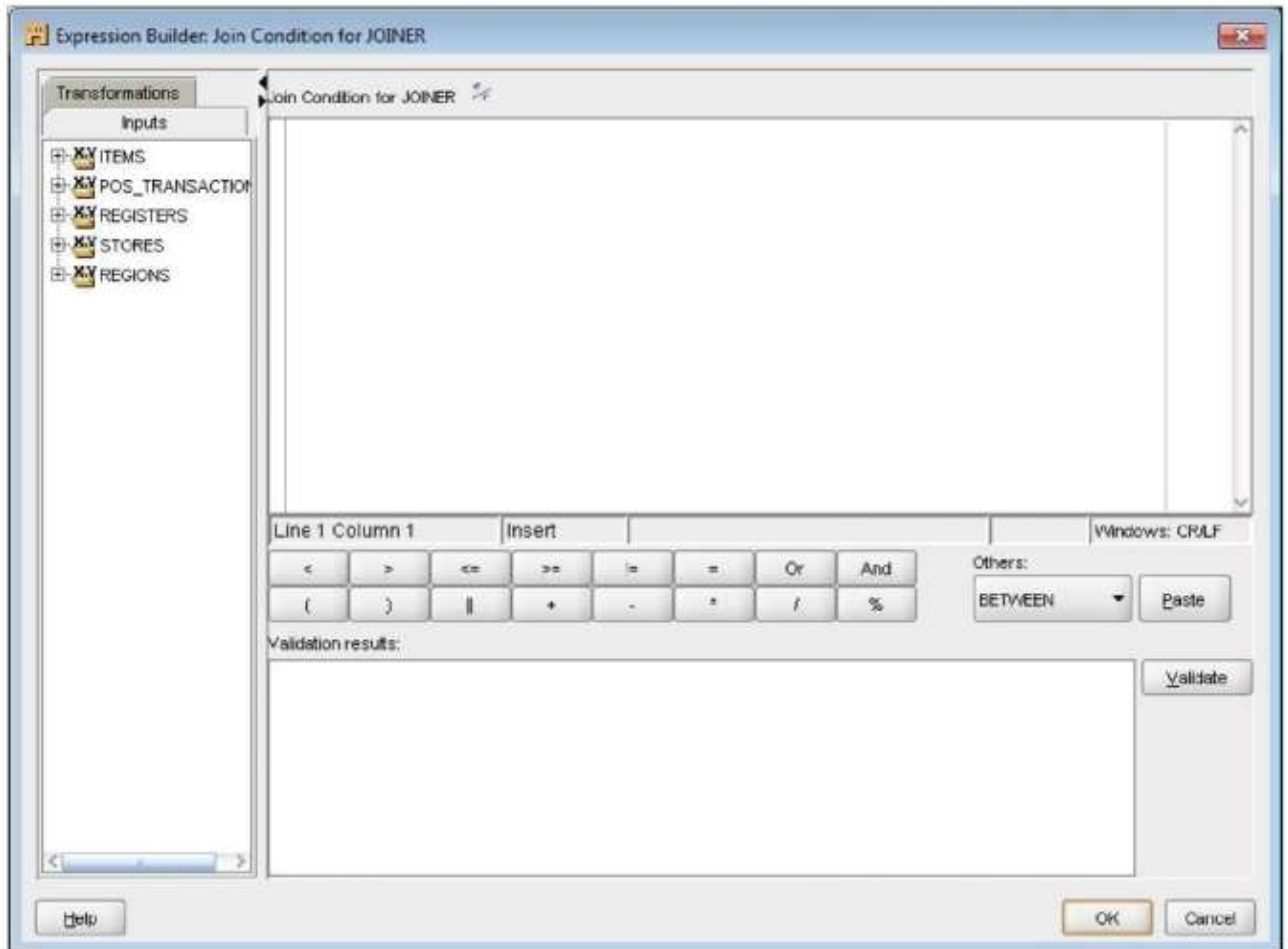
The next step in the process is to specify how we want the tables joined. We need to identify the attributes to use in the join condition. We will do that by modifying a property of the JOINER. This will be our first experience of working with the Properties Inspector window in the Mapping Editor. If the JOINER operator is not already selected, click once on the header of the box to select it and the Properties window will immediately change to display the properties of the selected object; in this case it's JOINER. We can see a property mentioned there, Join Condition. If it is not immediately visible, the properties can be scrolled down until it is.

If you click inside the JOINER operator on a group or attribute, the Properties window will display the properties for that group or attribute and not the JOINER operator as a whole. We want the JOINER properties, so make sure the JOINER itself is highlighted by clicking on the header of the JOINER window.

Click on the blank box to the right of the Join Condition label. The Properties window will now look like the following screenshot, which is ready for our input of the join condition:

Now we can type the join condition directly in the white box. This can get a bit tedious, especially as we'd need to know the correct syntax for specifying attributes. The Warehouse Builder refers to attributes by group as well as attribute name. So, to help us out with this, OWB provides the Expression Builder. This is a dialog box we can invoke to interactively build our Joiner condition.

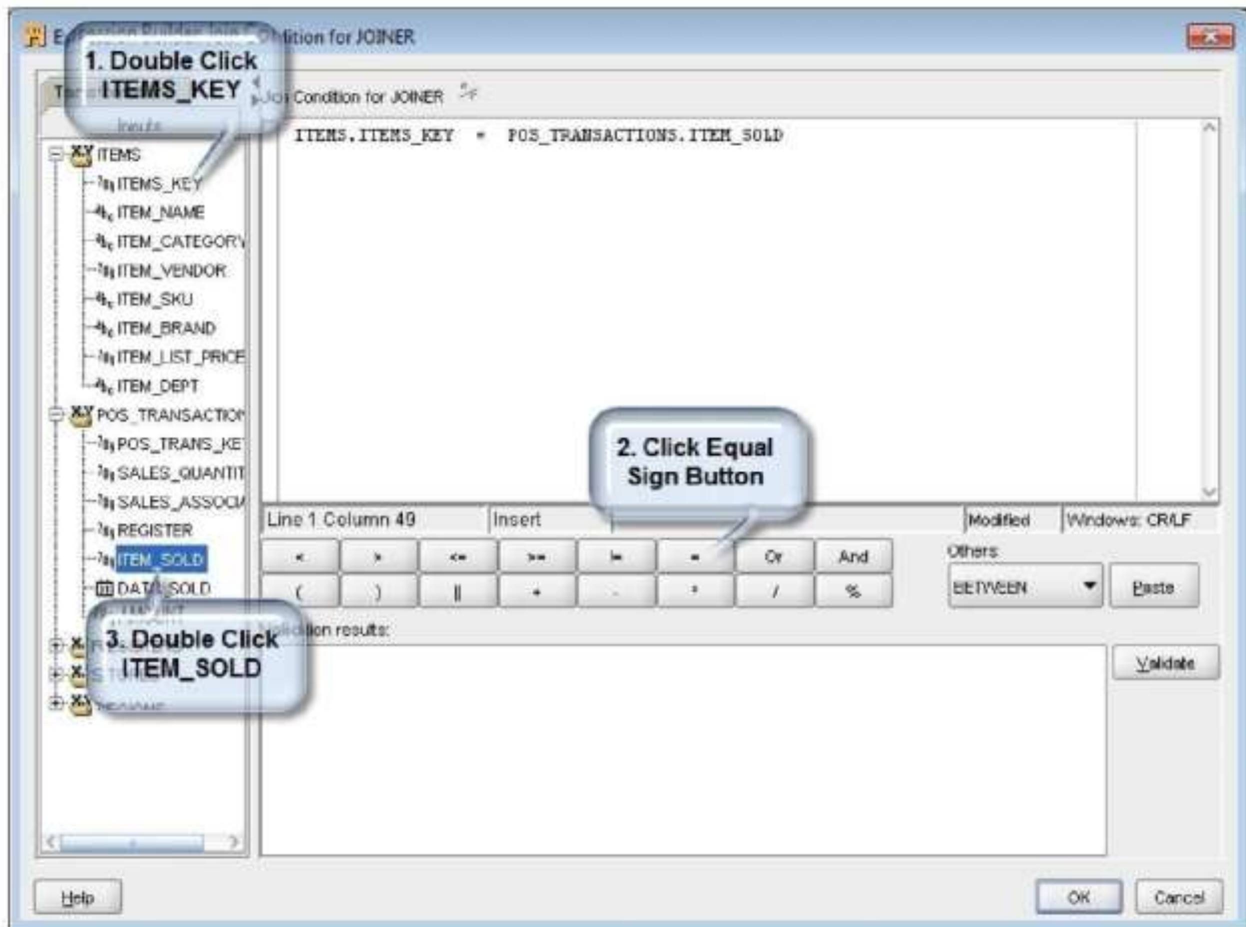
We can invoke Expression Builder by clicking on the button with the three dots (...) to the right of the blank white box. It looks like the following screenshot before anything is filled in:



Notice on the left the list of input groups in the JOINER. From this list we're going to select the attributes we need and will include them in our expression in the correct format. We just need to make sure we specify the correct attributes and the correct join relation, which will be the equal to (=) symbol in our case.
We do need to know a little about SQL join syntax at this point. The Expression Builder provides us with the list of attributes and the relational operator buttons, which will insert the indicated relations. However, we need to insert them in the right order. Fortunately, the syntax is not very complicated. We just need to specify which column from one table equals which column from the table being joined to. Then include an equality for each table with each of the equalities separated by AND.

We've seen the items table from our analysis in topic 2 of the source data structures in the acme_pos database. So we know that the pos_transactions table contains a foreign key field pointing to the record in the items table for the particular item for that transaction. This gives us clues about which columns will be needed in the first join equality — the item_sold attribute from

pos_transactions and the items_key attribute from the items operator. So, we'll expand the ITEMS group on the left and double-click the ITEMS_KEY attribute to add it to the expression. As we want every record included where the ITEM_SOLD equals the ITEMS_KEY, we will include an equal sign next by clicking on the button with the = sign on it. We'll finish this first relation by expanding the POS_TRANSACTIONS group and double-clicking on the ITEM_SOLD attribute to include it. Our expression now looks like the following with the steps highlighted with callouts:



The attributes include the group name first. This is the syntax it uses to identify the specific attribute. It's common for the same attribute name to appear in more than one group, and this syntax will make it explicit which attribute is being referred to.

We're not done yet, because so far we've only accounted for two of the four tables in our join condition. We also need to include registers, stores, and regions tables. The register attribute of the pos_transactions group contains the foreign key to the registers_key attribute of the registers table, so let's add that one. But before we add it, we need an and. So let's click on the And button (which is near the button labeled with an equal to sign) to enter it into our mapping, and then press the Enter key to advance to the next line. We move to a new line to prevent our expression from extending past

the viewable window. The expression could extend past it and still work. But for ease of viewing, we'll enter it vertically instead of horizontally so that we don't have to scroll.

Now we'll enter the following:

- The REGISTER attribute by double-clicking on it in the POS_ TRANSACTIONS group

- The equal to sign by clicking on the corresponding button

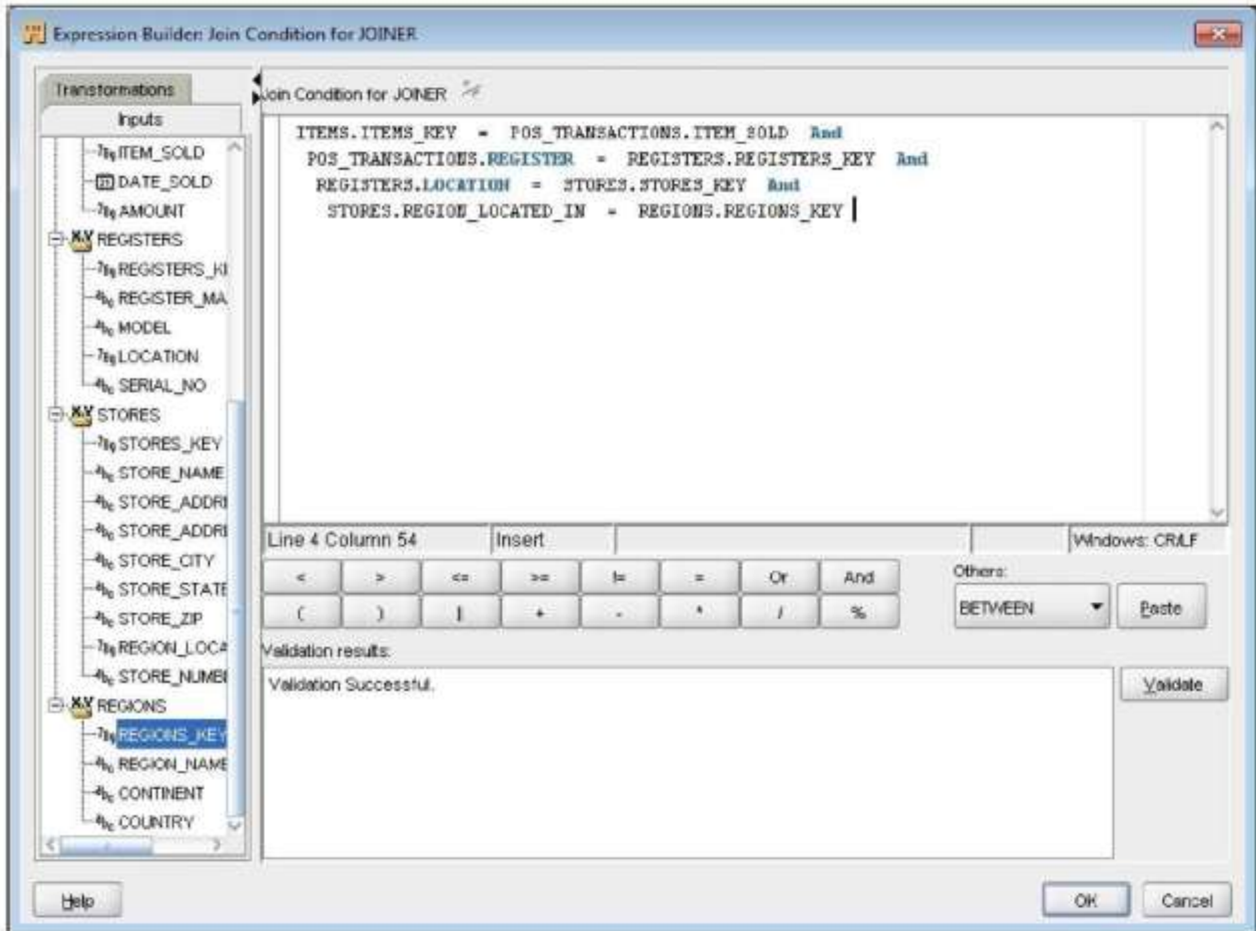
- The REGISTERS_KEY attribute by double-clicking on it under the REGISTERS group

- This expression is followed by another and by clicking on the And button

- Press the Enter key

The equal to sign and the And can be typed in manually if you prefer, rather than having to click on the buttons each time. Some people find that quicker to enter, and will just double-click the attributes needed and manually type the other operators that are needed.

Continue in the like manner with the location in the registers group equal to stores_key in the stores group, and region_located_in in the stores group equal to the regions_key in the regions group. Do not include And after this last part of the expression as it is the end. When completed, the expression should look like the one in the following screenshot:



We can click on the Validate button now to make sure the expression we just entered is a valid expression, meaning that we used the correct SQL syntax. We should get the Validation Successful message.

Depending on what release number of the database you are running, you may get the following error message instead of the success message: An error occured during expression validation. Bad expression return type. This is a known bug, ID 7417869, which is reported to have occurred in Release 10.2.0.4 of the database and is fixed in 10.2.0.5. It also reports that the mapping will deploy and execute successfully even if this validation bug occurs. The bug report also says the error occurred in Release 11.1.0.7 of the database. However, it works in v11.2.0.1, which is the most recent publicly available version for download that we are using for this topic. The screenshot we just saw was taken from 11.2.0.1 and we can see the Validation Successful message. So if you are using this version, you should not see the error.
Click on the OK button and we are done specifying the join condition for the JOINER operator. We can now see that it filled in the join condition text into the Join Condition entry in the Properties

window. This completes the Joiner, but we still have to aggregate the data so that it's at the level we need for loading into the data warehouse. For aggregating data, we'll now include an Aggregator operator.

Adding an Aggregator operator

An Aggregator operator is used to apply an aggregate function to the data. Aggregation functions are documented in the Oracle Database SQL Language Reference
at **http://download.oracle.com/docs/cd/E11882_01/server.112/ e17118/functions0 03.htm#i8 92 03.** Topic 5 of that online document has a section devoted to aggregate functions. In our case, we will need to add up the sales quantities and dollar amounts for every product, and the store and date combination to have data at the right level to load into the data warehouse. For this aggregation of data, we can use a sum() function.
The Aggregator operator requires that we specify a few things for it to function correctly. As with any operator, there are attribute groups to set and an Aggregator operator has one input and one output group. For the input group, we will drag the output attributes of the Joiner operator. We have to specify a group by clause that the Aggregator operator is going to use to group the data, and it will create an output attribute for every attribute we use in the group by clause. We have to manually add output attributes for any of the values that are going to be summed up, and then specify the sum() function to use for them.

We'll follow a similar process to add the Aggregator operator as we did for the Joiner; drag an AGGREGATOR operator onto the canvas to the right of the JOINER operator, connect output attributes from the JOINER operator to the input of the AGGREGATOR operator, define properties for the AGGREGATOR operator, and then connect the output of the AGGREGATOR operator to the POS_TRANS_ STAGE table operator. Here we'll outline the steps to follow without going into as much detail as we did for the Joiner operator:

1. Drag an aggregator operator from the Component Palette window to the canvas and drop it to the right of the joiner operator between that operator and the pos_trans_stage target operator. You may have to move the pos_ trans_stage target operator further to the right to make enough room.
2. Connect the output attributes from the joiner operator as input to the aggregator operator by dragging the OUTGRP1 output group and dropping it on the INGRP1 input group of the aggregator operator. This will map every output attribute at once, so we don't have to do each one individually.
We have one issue we need to address with the input attributes for the Aggregator and that is related to the DATE_SOLD attribute from the Joiner operator. An attribute of the DATE type

includes both date and time of day. We are going to sum up the data by date. But if we include the time of day, we'll get multiple dates occurring on the same day that are treated as distinct because the time is different. We want the sales for every product in a store for a single date to sum together regardless of the time of day the sale occurred. We need to strip out the time from the DATE_SOLD attribute, so we just have the date as input into the Aggregator operator. For that task, we need a Transformation Operator to apply the TRUNC() function to the value first. We'll discuss Transformation Operators in greater depth in the next topic, but let's use one now to take care of the date.

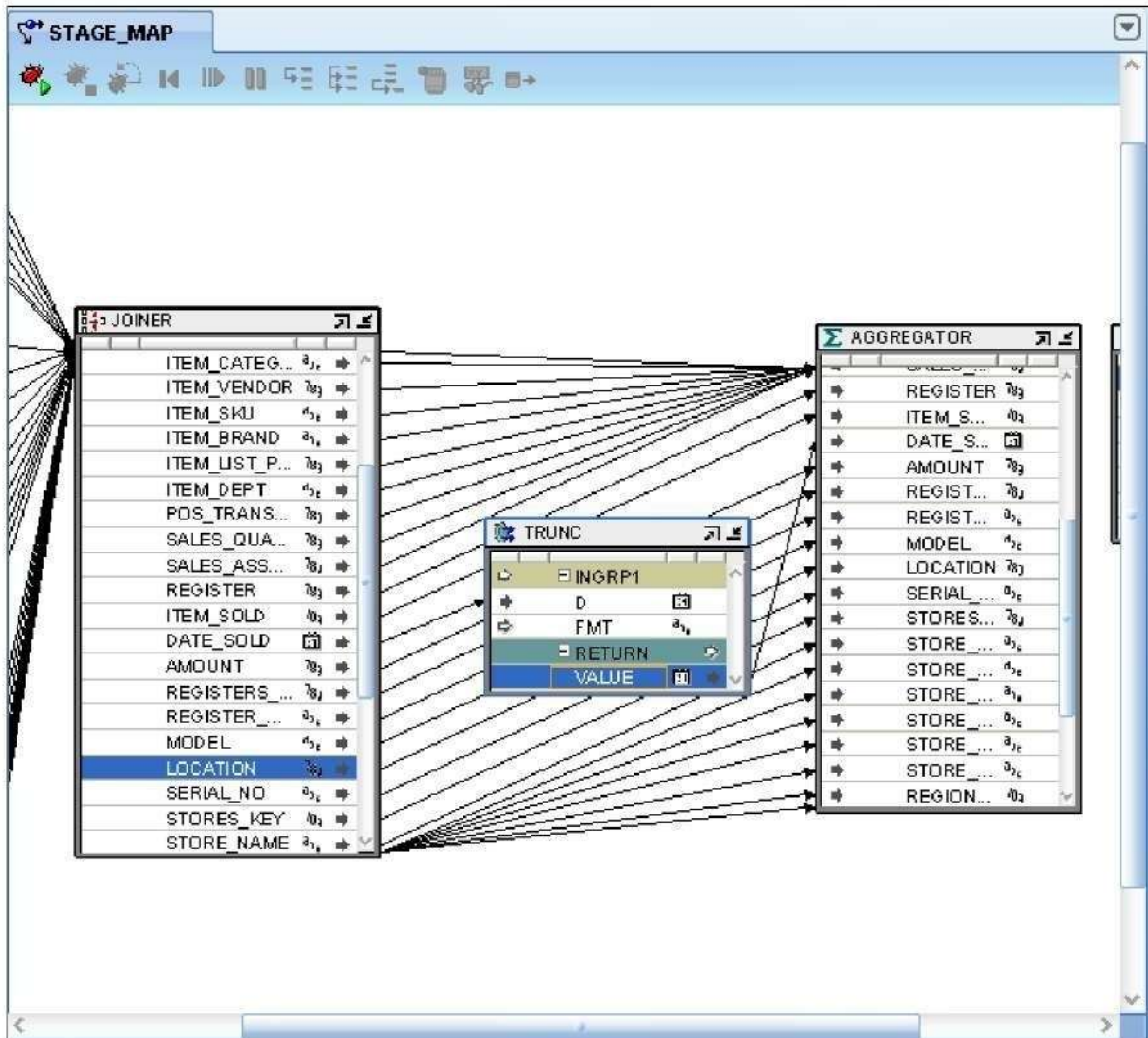
3. We need to remove the line that got dragged to the input of the Aggregator operator for the date_sold attribute by clicking on the line and pressing the Delete key, or right-clicking and selecting Delete from the pop-up menu. Make sure the correct line is selected. Attribute groups can be expanded to spread the lines apart better so that it's easier to click.

4. Drag a Transformation Operator from the Component Palette window and drop it on the canvas between the Joiner operator and the Aggregator operator near the DATE_SOLD attribute. In the resulting pop up that appears, we'll scroll down the window until the Date() functions appear and then select the trunc() function. It will look like the following:

TRUNC(IN DATE, IN VARCHAR2) return DATE Click on that line and then click on the OK button to select it. It will drop a TRUNC Transformation Operator on the canvas.

5. Connect the DATE_SOLD attribute in the OUTGRP1 group of the Joiner to the D attribute of the INGRP1 of the TRUNC transformation operator. Then connect the VALUE attribute of the RETURN output group of the TRUNC operator to the DATE_SOLD attribute of the INGRP1 group of the Aggregator operator. We're not going to worry about mapping anything into the FMT input attribute. That is for the optional format parameter for the TRUNC() SQL function which if not provided defaults to truncate the date to the nearest day which is what we need. The above referenced SQL Language Reference explains all about that.

The canvas should now look similar to the following screenshot:



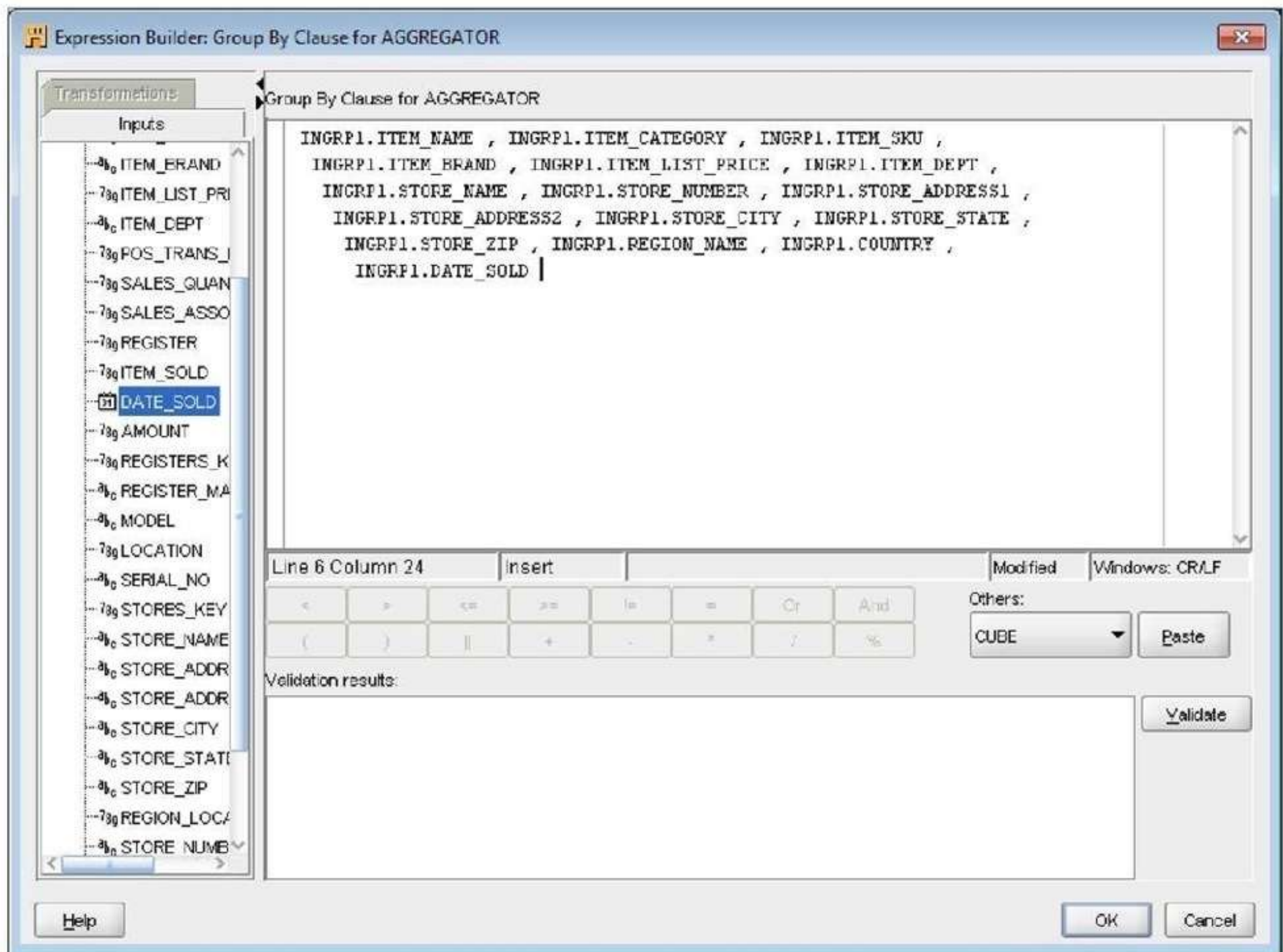
6. We have our input set for the Aggregator operator and now we need to address the output. Let's select the Aggregator operator by clicking on the title bar of the window where it says AGGREGATOR. The Properties window of the Mapping Editor will display the properties for the aggregator. If it doesn't, then make sure the title bar of the window was selected for the operator and not somewhere inside the operator.

7. The very first attribute listed is Group By Clause. We'll click on the ellipsis (...) on its right to open the Expression Builder for the Group By Clause. This is similar to how we launched it earlier to edit the join condition for the Joiner operator.

8. Enter the following attributes separated by commas by double-clicking each in the INGRP1 entry in the left window:

```
INGRP1.ITEM_NAME , INGRP1.ITEM_CATEGORY , INGRP1.ITEM_SKU ,
INGRP1.ITEM_BRAND , INGRP1.ITEM_LIST_PRICE , INGRP1.ITEM_DEPT
, INGRP1.STORE_NAME , INGRP1.STORE_NUMBER , INGRP1.STORE_
ADDRESS1 , INGRP1.STORE_ADDRESS2 , INGRP1.STORE_CITY , INGRP1.
STORE_STATE , INGRP1.STORE_ZIP, INGRP1.REGION_NAME , INGRP1.
COUNTRY , INGRP1.DATE_SOLD
```

When completed, it should look similar to the following screenshot:

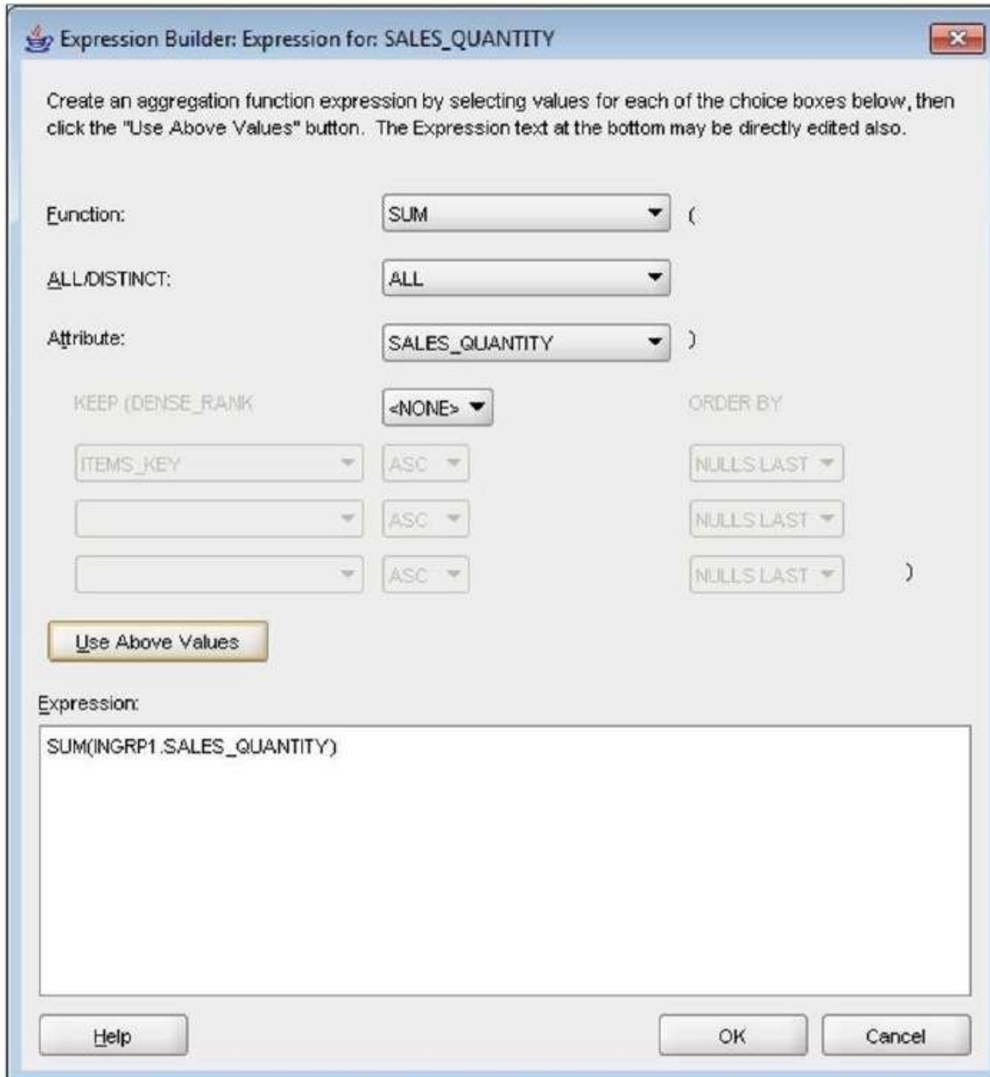


9. We'll click on the OK button to close the Expression Builder dialog box and looking at the AGGREGATOR now, we can see that it added an output attribute for each of these attributes in our group by clause. This list of attributes has every attribute needed for the POS_TRANS_STAGE operator except for the two number measures, sale_quantity and sale_dollar_ amount. So let's add them manually.

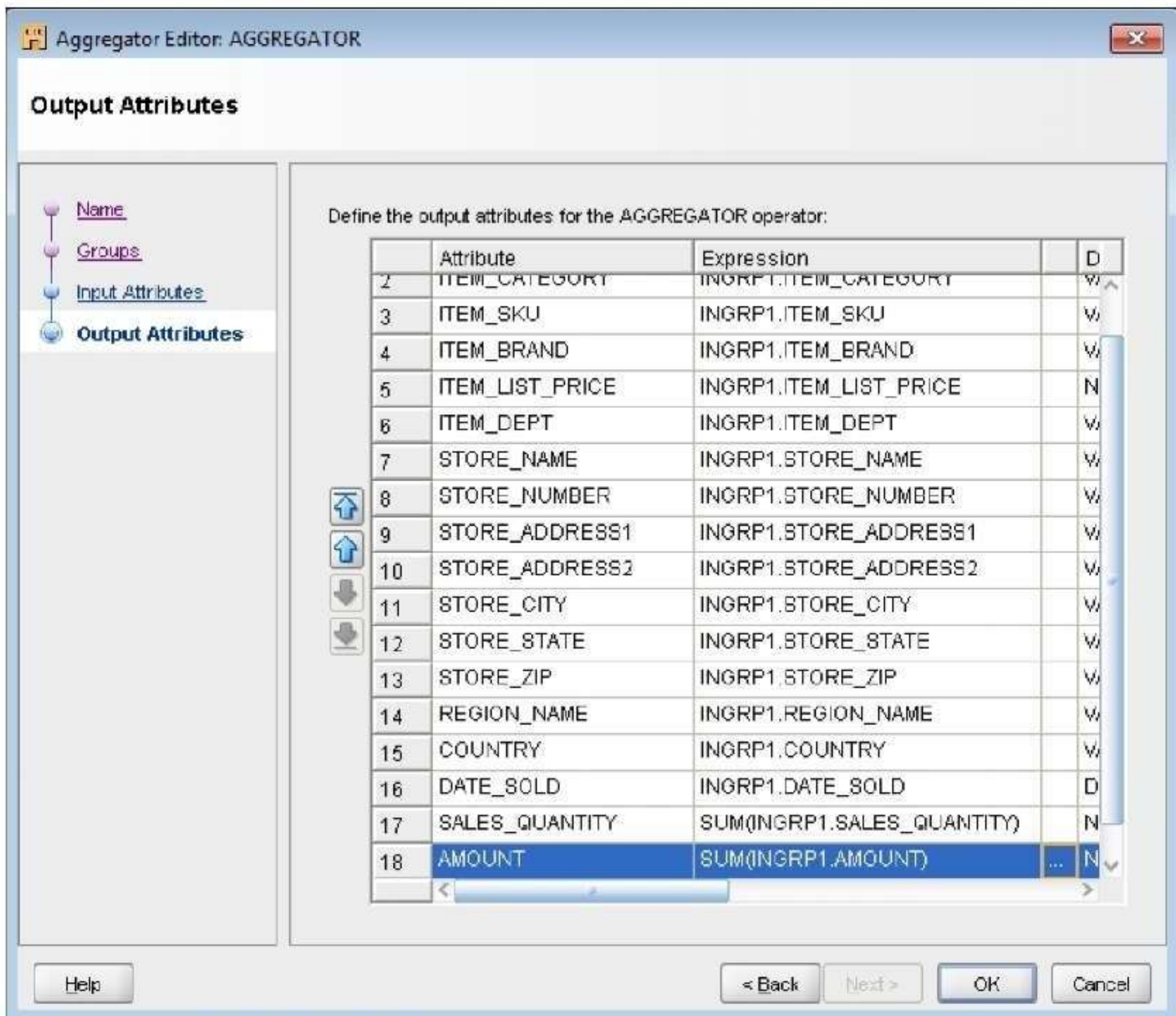
10. We'll right-click on the OUTGRP1 attribute group of the AGGREGATOR operator and select Open Details... from the pop up. We used this editor earlier for the Joiner to edit the groups, and now we're going to use it for the Aggregator to edit the attributes in a group.

11. We’ll click on the Output Attributes entry on the left, and then enter a new attribute in the blank line at the end called SALES_QUANTITY and leave the type NUMERIC with 0 for precision and scale. We’ll enter AMOUNT next and make it type NUMERIC with precision 10 and scale 2. Now we need to apply the SUM() function to these two new attributes. The Aggregator Editor has a column for the expression to be associated with each attribute and we can see that expressions have already been filled in for the other attributes we indicated as the "group by" attributes and now we need to provide the expressions to sum up the two new attributes we just entered. So we’ll click on the expression for SALES_QUANTITY and then click on the ellipsis beside the Expression to launch the Expression editor for this attribute.

12. We’ll immediately notice something different. The Expression editor for output attributes of an Aggregator is custom built to apply aggregation functions. We’ll select SUM from the Function drop-down menu, ALL from the ALL/DISTINCT drop-down menu, and SALES_QUANTITY from the Attribute drop-down menu. We’ll then click on the Use Above Values button and the expression will fill in showing the SUM function applied to the SALES_QUANTITY attribute. This is shown in the next screenshot of the Expression editor:



13. We'll click on the OK button to save the expression and close the dialog box. Then we'll do the same thing for the amount output attribute of the Aggregator, but will select AMOUNT for the Attribute drop-down menu. After making these changes, this is how the Aggregator Editor will look:



We'll click on the OK button to close the AGGREGATOR Editor dialog box. We're almost done now. We've included the following:

- The source tables we need to pull the data from

- The target table we're going to store the data in

- A Joiner operator to join together the source tables

- An Aggregator to sum up the data

We have also connected the source tables as input to the Joiner operator and the Joiner as input to the Aggregator operator. The only thing left is to connect the output attributes of the Aggregator operator to the target input attributes. Before doing that, let's make the target table

operator box big enough to display all its attributes at once without having to scroll. Just click and hold on the bottom edge of the POS_TRANS_STAGE window and drag the window down until all the attributes are visible. Do the same to the Aggregator operator window, but make sure only the output group is expanded. The input group should be collapsed because we're finished working with it for now.
Make the following attribute connections between the Aggregator and the POS_ TRANS_STAGE table by clicking and dragging a line between attributes. We'll do individual attributes this time, not the whole group.

• SALES_QUANTITY to SALE_QUANTITY

• AMOUNT to SALE_DOLLAR_AMOUNT

• DATE_SOLD to SALE_DATE

• ITEM_NAME to PRODUCT_NAME

• ITEM_SKU to PRODUCT_SKU

• ITEM_CATEGORY to PRODUCT_CATEGORY

• ITEM_BRAND to PRODUCT_BRAND

• ITEM_LIST_PRICE to PRODUCT_PRICE

• ITEM_DEPT to PRODUCT_DEPARTMENT

• STORE_NAME to STORE_NAME

• STORE_NUMBER to STORE_NUMBER

• STORE_ADDRESS1 to STORE_ADDRESS1

• STORE_ADDRESS2 to STORE_ADDRESS2

• STORE_CITY to STORE_CITY

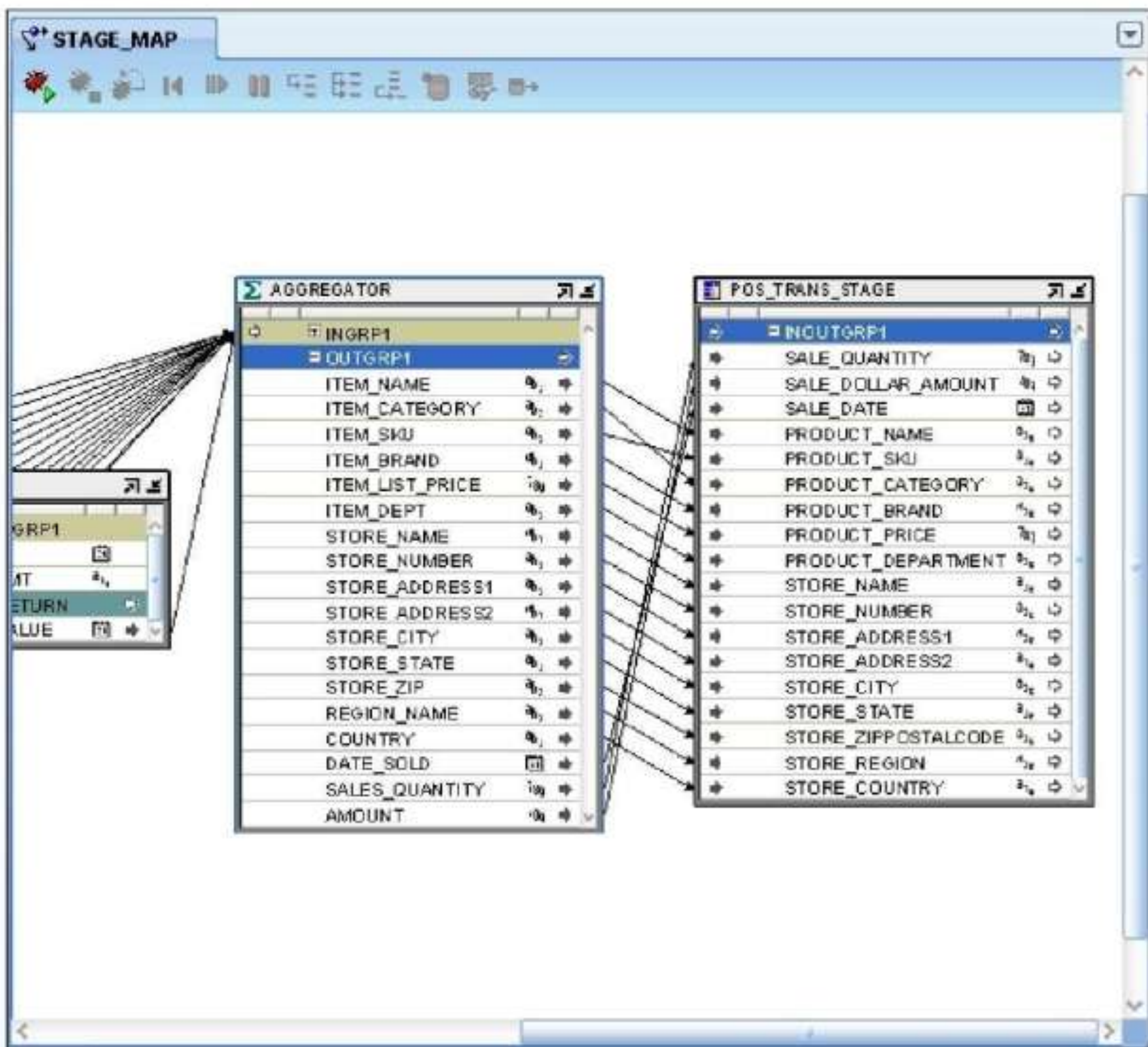
• STORE_STATE to STORE_STATE

• STORE_ZIP to STORE_ZIPPOSTALCODE

• REGION_NAME to STORE_REGION

• COUNTRY to STORE_COUNTRY

If we focus on just the Aggregator and the POS_TRANS_STAGE operators our mapping should now look like the following after making all those connections:



Notice that it's not always possible to avoid overlapping lines altogether, but we just want to avoid them as much as we can for readability. In this case, the overlapping lines are a result of

different ordering of the attributes in the AGGREGATOR operator and the POS_TRANS_STAGE table. Changing this would involve recreating the table to reorder columns, and that is just not worth the effort. It will often be the case that sources and targets don't line up like that, but when we add intervening operators over which we have a more direct control, that is where we can focus our efforts on being neat and orderly. Also, your ordering of attributes in the OUTGRP1 group of the AGGREGATOR operator may be different depending on the order in which you mapped the attributes from the Joiner. The order is not important as long as all the required attributes are present.
This completes the staging mapping. We've seen how to create a complete mapping in the Warehouse Builder. We'll make sure to save at this point so that we don't lose anything before we move on.

Summary

We saw how to create a mapping in the Warehouse Builder, including how to design a staging area table, build it with the Table Editor, design a mapping to populate it, and then create the mapping using source and target operators, and the intervening Joiner and Aggregator operators. We also got to use a Transformation Operator.

Now that we have completed our staging table mapping, we need to design mappings to get our dimensions and cube populated from the data in the staging table. We'll do that in the next topic while taking a look at transformations, which are a key feature of the Warehouse Builder for loading and cleaning up data.