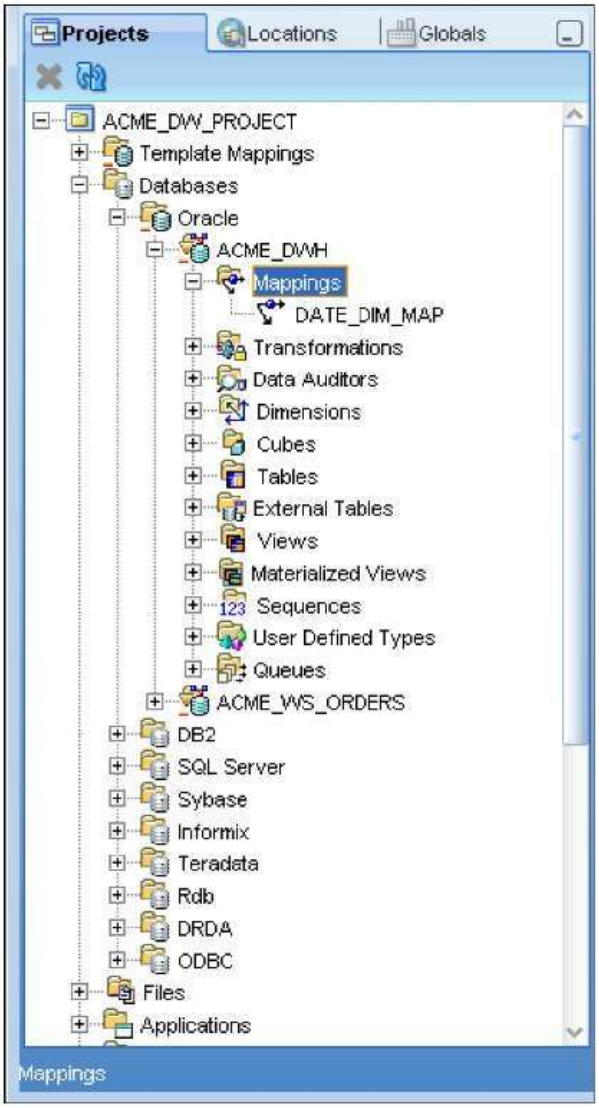
We are now going to look at the Warehouse Builder and its features for designing and building our ETL process. OWB handles this with what are called mappings. A mapping is composed of a series of operators that describe the sources, targets, and a series of operations that flow from source to target to load the data. It is all designed in a graphical manner using the Mapping Editor, which is available in the Design Center. Let's run the Design Center now and take a look at the Mapping Editor, its features, and some of the operators that are available to us. Launch the Design Center as we discussed in topic 2 in the Overview of Warehouse Builder Design Center section.

OWB Mappings

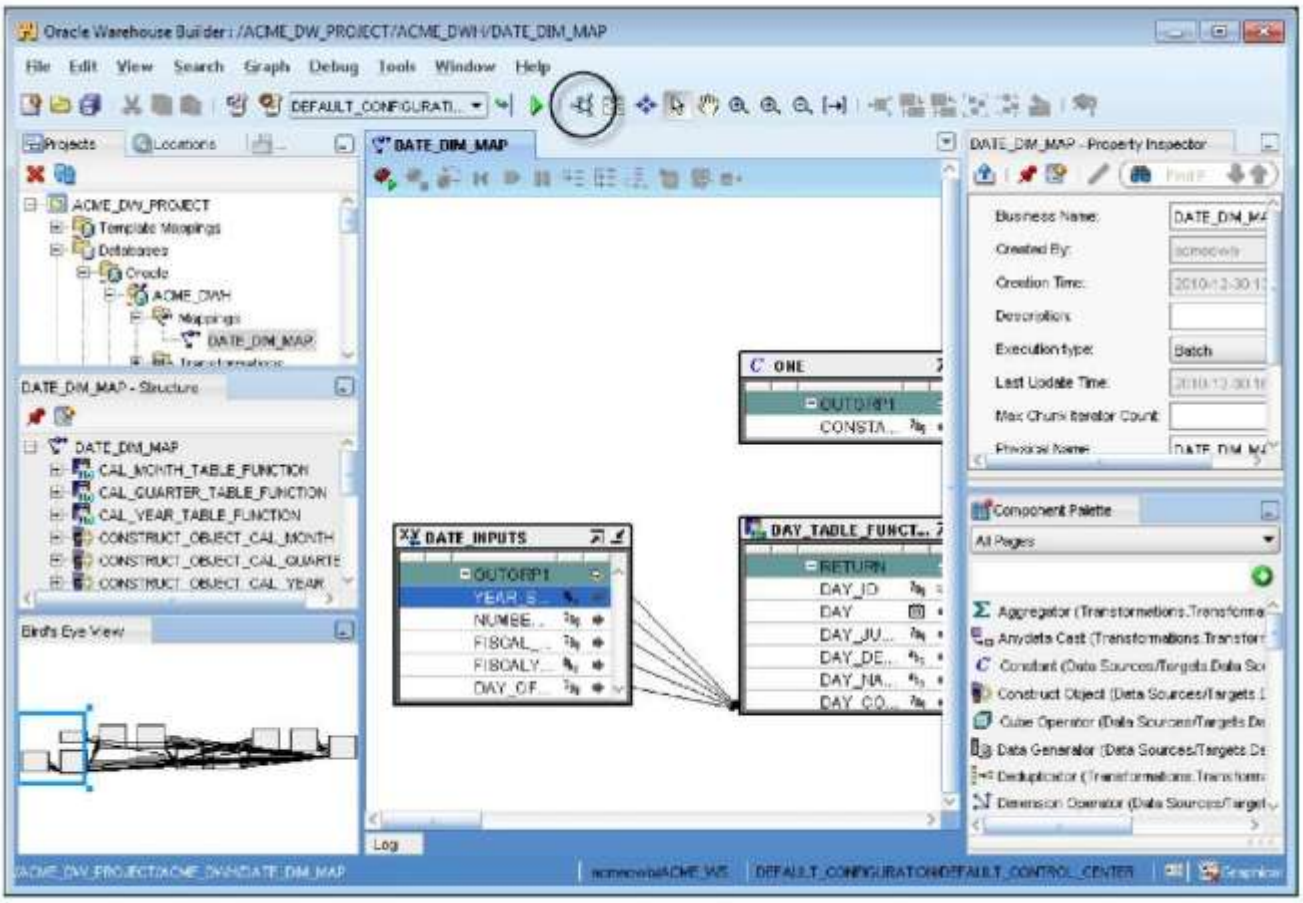
In the Design Center | Project Navigator window, expand the ACME_DW_ PROJECT project (if it is not already expanded) by clicking on the plus sign beside it. To access the Mapping Editor, we need a mapping to work on. So to begin with, we could create an empty mapping at this point.

There is no wizard for creating mappings as there is for importing source metadata, or creating dimensions and cubes. There are too many options to lay out the mapping for a wizard, so it's difficult to make any kind of intelligent guesses as to how to design. However, there are some cases where we get a mapping for "free" such as the DATE_DIM_ MAP, which was created for us automatically by the Time Dimension wizard — but that is the exception rather than the rule.

Mappings are created in the Mappings node. We can find it under the module we created to hold our data warehouse design under the Databases | Oracle node in our project. Expand that module, which we called ACME_DWH, and then expand the Mappings node underneath it. For reference, your Project Navigator should look like this now:



The DATE_DIM_MAP we see under Mappings is the mapping that was created for us automatically by the Time Dimension wizard. Instead of creating a new mapping, which will have nothing in it yet, let's open this mapping and take a look at it for our initial exploration of the features in OWB for designing mappings. Let's double-click on the DATE_DIM_MAP mapping. It will launch the Mapping Editor and load the DATE_DIM_MAP into it. We are not going to modify it, but we will use the displayed Mapping Editor window to familiarize ourselves with its features. This is a very similar concept to the data object editors we looked at in the last topic but for mappings, not data objects. We're going to go into more detail in using the Mapping Editor simply because we need to use it to build our mappings; there is no wizard available to us. The Mapping Editor window looks like the following:



The Mapping Editor opens in the Design Center the same way as the data object editors we saw in the last topic. The primary difference is the mapping editor is graphical where the data editors are primarily textual, with one tab, the Physical Bindings, displaying objects graphically. A few other additional windows are noticeable also, the Component Palette, the Structure View, and the Bird's Eye View windows.

The Component Palette should open automatically when a mapping is edited but the Bird's Eye and Structure views do not, but they are good to have open for quickly navigating around the canvas and viewing the objects in the mapping. Go ahead and open them from the View menu by selecting Bird's Eye and Structure if they are not already open. The window for the above image was re-sized to better fit here, but we're going to maximize the window while we work with it for ease of use as we can view more information on the screen. It is a good idea to make your screen resolution as high as possible so more screen real estate is available for displaying. Play around with the layout of the windows to come up with an orientation that works best for you. If the Structure and Birds Eye view are not being used, just close them to make more room. As you use the tool more you'll figure out what works best for you.

Your view might look different or be perhaps all jumbled from previous edits. It's very easy to move objects around on the canvas (that big window in the middle). We can go on clicking on the

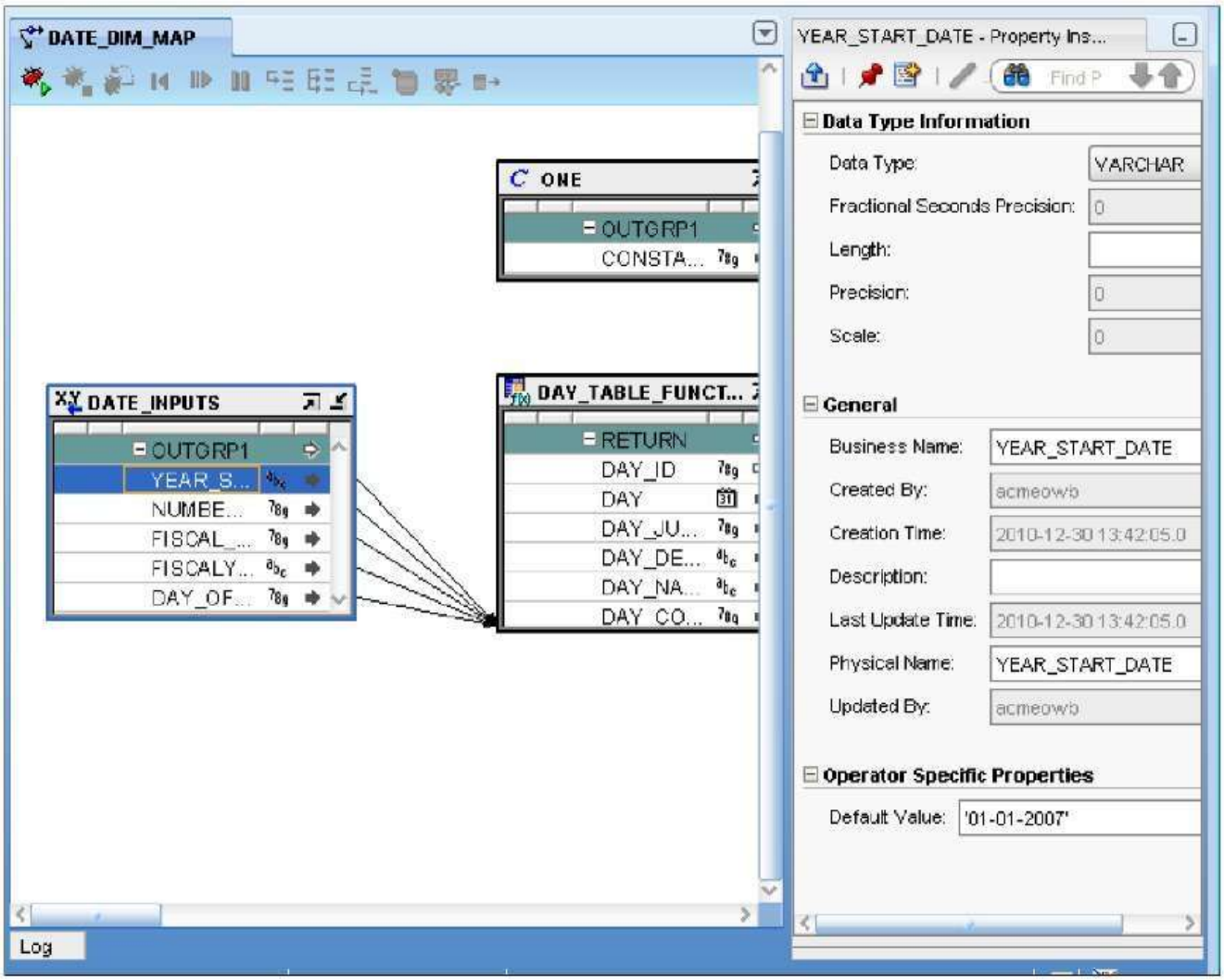
objects and dragging them into new locations to try to neaten it up but that's too much work. The Mapping Editor, as with all the graphical editors, provides a convenient Auto Layout option that will do all that for us. Click on the Auto Layout button in the toolbar to spread everything out. It is circled (at the top) for your reference in the previous image. The command is also available from the Graph menu entry under the Zoom and Layout sub-menu.
Let's briefly discuss some of the main features we can see in the Mapping Editor screen. The Oracle Warehouse Builder Data Modeling, ETL and Data Quality Guide, which is available at **http://download.oracle.com/docs/cd/E118 82_01/ owb.112/e10935/mappings.htm#BEIGDJAE**, covers each of these windows in greater detail in topic 5 on PL/SQL mappings along with other details about mappings. So we'll just touch upon them briefly, especially as we've seen some of them in the last topic:

• Mapping
The Mapping window is the main working area in the center of the above image where we will design the mapping. This window is also referred to as the canvas. This is the graphical display that will show the operators being used and the connections between the operators that indicate the data flow from source to target.

• Structure View
We haven't seen this window yet. It provides a hierarchical view of the objects in the editor, including operators and attributes of those operators for a mapping. It will display the structure for a data object also if opened when editing a data object. This window is similar to the old Explorer window from the data and mapping editor windows in the previous release.

• Property Inspector
The Property Inspector window displays the various properties that can be set for objects in our mapping. It is the same window we saw when looking at the data object editors in the last topic. When an object is selected in the canvas, its properties will display in this window. We can resize any of these windows by holding the mouse pointer over the edge of a window until it turns into a double arrow, and then clicking and dragging to resize the window so we can see the contents better. To investigate the properties window a little closer, let's select the DATE_INPUTS operator. We can scroll the Structure View window until we see the operator and then click on it, or we can scroll the main canvas until we see it and then click on the top portion of the frame to select it. It is the first object on the left and defines inputs into DATE_DIM_MAP. It is visible in the previous image. After clicking on it, all the properties applicable to it will be displayed in the property inspector window. The only properties displaying for the DATE_INPUTS operator are the standard properties associated with any operator- name, description, and some creation and update information. Let's click on one of the attributes to see a better example of more specific attributes. Click on

YEAR_START_ DATE within the DATE_INPUTS operator. It is the first attribute of the DATE_INPUTS operator and can be selected in the Structure View or on the canvas by clicking on it and is shown in the following image, which is a portion of the Design Center showing the Mapping Editor window and properties we're referring to. The windows have been resized to better display the information being referred to here:



Now we can see some more interesting properties. YEAR_START_DATE is an attribute of the DATE_INPUTS object and defines the starting date to use for the data that will be loaded by this mapping. The properties that can be set or displayed for it include the characteristics about this attribute such as what kind of data type it is, its size, and its default value. Recalling our running of the Time Dimension Wizard in the last topic, there was one option to specify the year range for the dimension and we chose 2007 as the start year and that is what formed the source for the default value we can see here. Do not change anything but just click on a few more objects or attributes to look around at the various properties.

• **Component Palette**

The Component Palette contains each of the objects that can be used in our mapping. We can click on the object we want to place in the mapping and drag it onto the canvas. This list will be customized based on the category selection in the dropdown at the top of the window to view either all the components or subsets of the components by category/purpose.

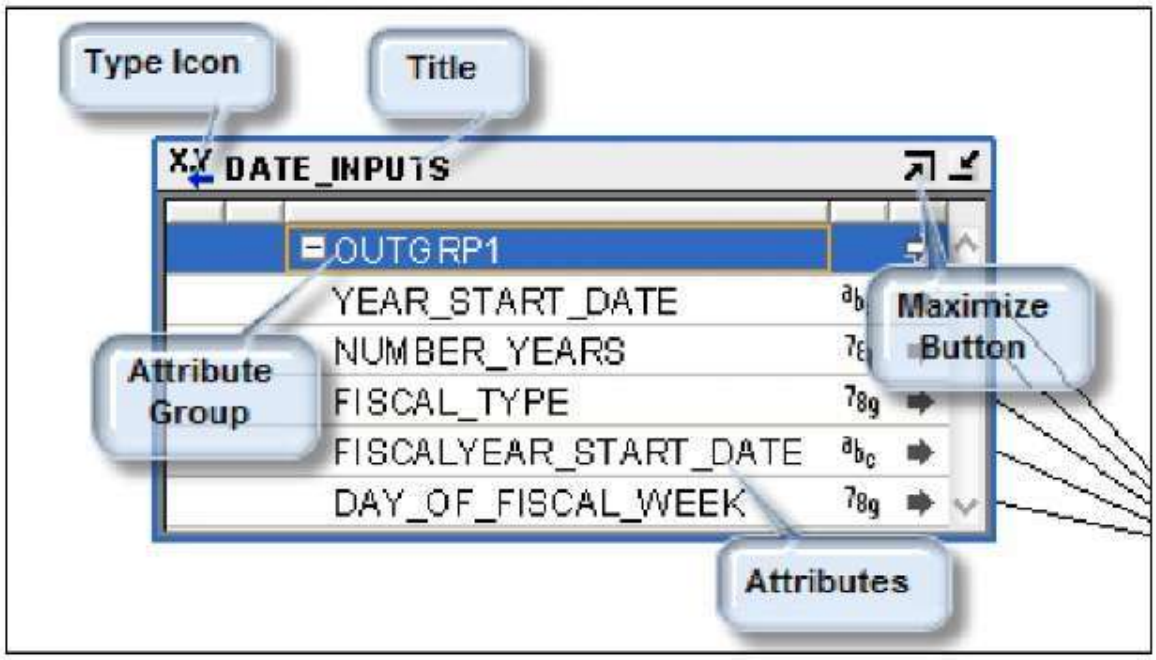
• Bird's Eye View

This window displays a miniature version of the entire canvas and allows us to scroll around the canvas without using the scroll bars. We can click and drag the blue-colored box around this window to view various portions of the main canvas. The main canvas will scroll as we move the blue box. Go ahead and give that a try. We will find that in most mappings, we'll quickly outgrow the available space to display everything at once and will have to scroll around to see everything. This can come in very handy for rapidly scrolling the window.

The canvas layout

Let's take a closer look at some of the general features of the operators we can see in our canvas, and then at some of the features specific to different operators. Operators on the canvas are represented by boxes with a title that indicates the name of the operator and an icon that indicates the type. In the canvas, we'll take a look at the operator that is on the far left of the canvas called DATE_INPUTS. This operator happens to be a Mapping Input Parameter operator.

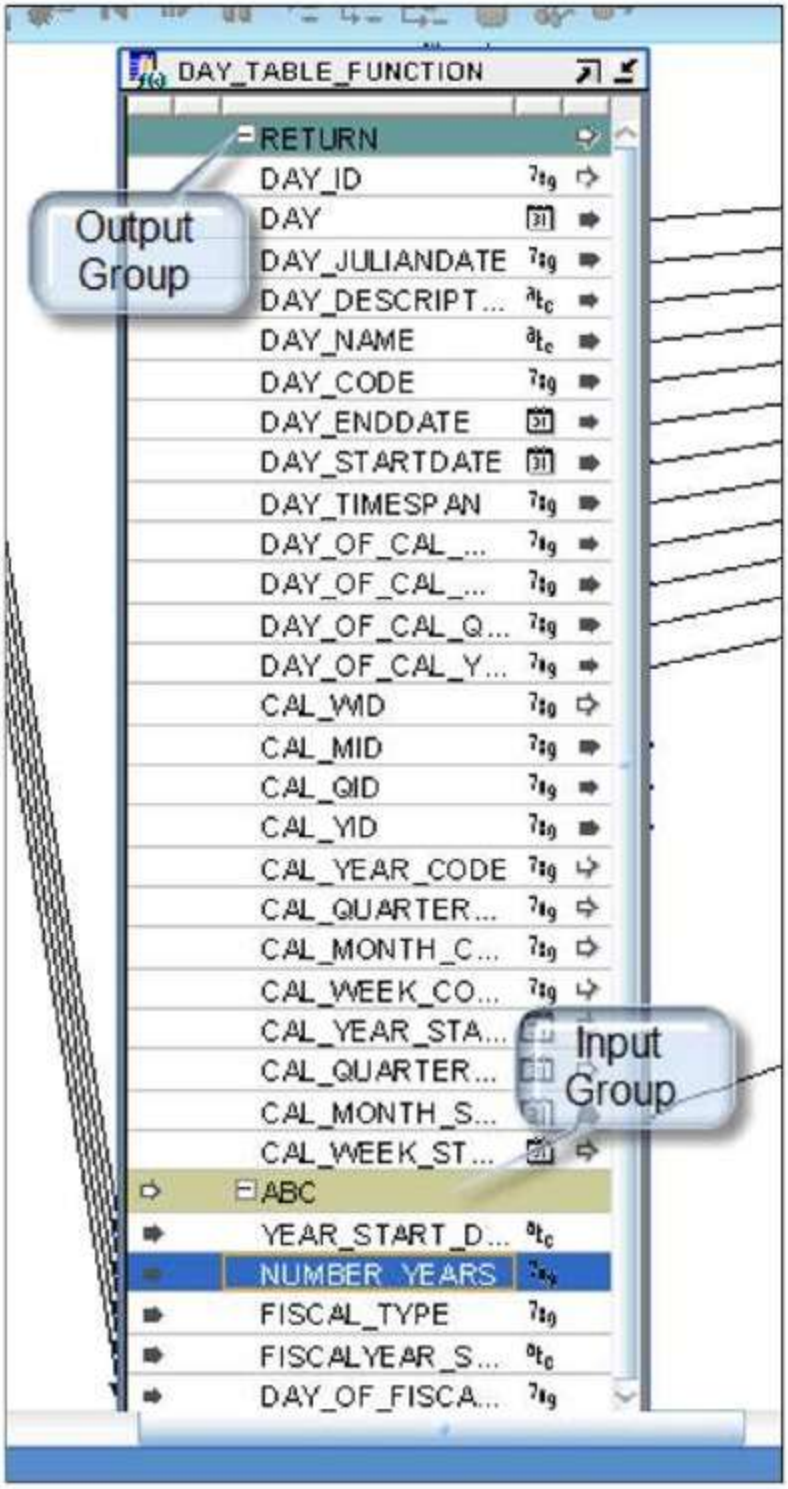
It is shown in the following screenshot with the key features highlighted with callouts:



The box can be resized by clicking the left mouse button and holding it over an edge of the operator, and then dragging it to a new size. To show the entire contents at once, you can click on the maximize button to instantly expand the box, as shown in the above screenshot.

A handy feature available for viewing operators is to minimize them to better see the layout and mapping line connections. You can use the Ctrl-A key combination to select all operators on the canvas then select Graph…Minimize from the main menu to minimize them all and then Graph…Zoom and Layout. Auto Layout to automatically arrange them in a logical orientation.
We can see some lines of information listed inside the box, which are the attributes of this operator. There are two major types of attributes—an input group and an output group. In this case, we can see one group named OUTGRP1 which tells us this operator has only an output group. This operator represents the input of information into the map at the beginning, so it is not meant to have any other operators mapped as input to it. Thus, it has no input attributes, but only output attributes.

Take a look at the operator on the right of the DATE_INPUTS operator called DAY_TABLE_FUNCTION. It has both input and output attributes as shown in the next screenshot, because this operator represents a PL/SQL function. A PL/SQL function takes the values supplied as input attributes in the Input group as parameters to the function and returns the value or values indicated in the Output group as a return value from the function.



Renaming attribute group names

The names INGRP1 and OUTGRP1 are generic names that OWB uses as default names for input and output groups when there is only one of each. We can change those names if we want to and that is

exactly what the Warehouse Builder has done with the DAY_TABLE_FUNCTION operator shown above. It has renamed the INGRP1 input group as ABC and OUTGRP1 output group as RETURN.

There are some operators that contain more than one input or output group. For instance, Joiner Operator is an operator to represent a join of two or more tables. This operator will have an input group defined for each table. In that case, each input group would have a different number incrementing from 1 (for example, INGRP2, INGRP3, and so on). It makes sense in this case to rename the input groups to match the tables being joined, but that is not a requirement.
An attribute group name is edited in the Details window for the group name. This window is accessible by right-clicking on the group name in the canvas and selecting Open Details... from the pop-up menu or by double clicking on the group name. It can also be edited by just clicking the group name and editing the name properties in the Property Inspector window. Using the Details window has the added benefit of displaying whether the group is an input group, an output group or both by including a column labeled Direction which we don't see in the Property Inspector. We'll be making use of that Details window in the next topic when we actually start building a mapping.

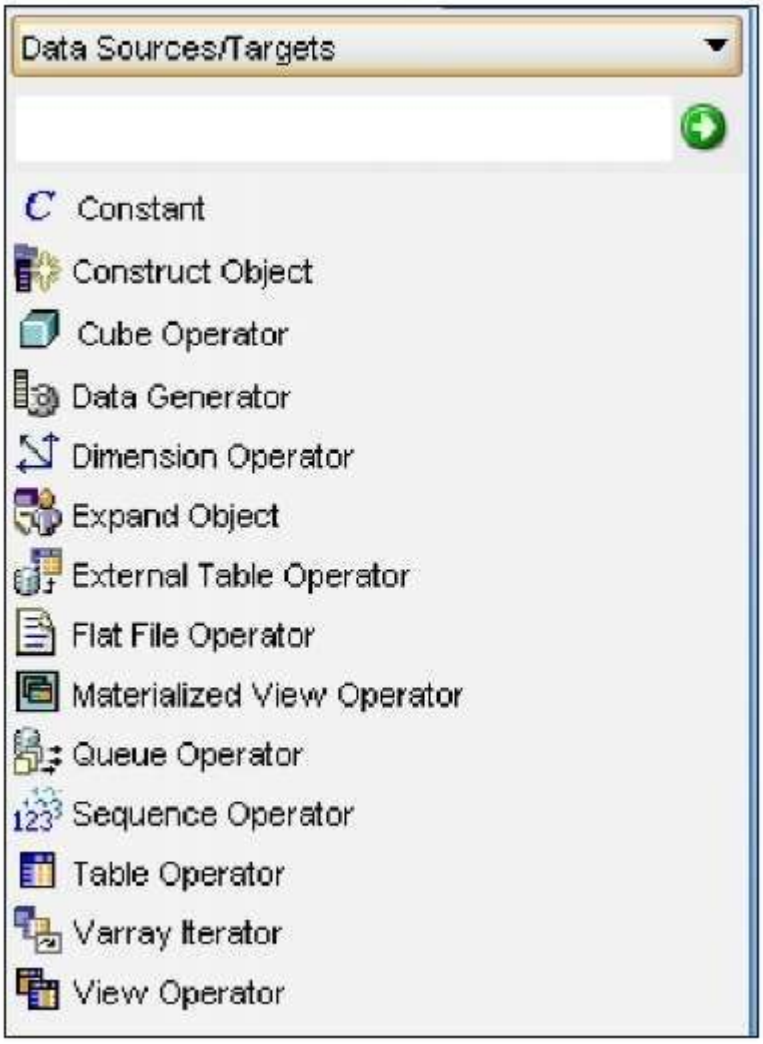
We can also note with day_table_function that the number of attributes is greater than what's displayed in the operator window depending on how big we've made the window. We can scroll down through the list of attributes, or resize the operator in the canvas to see more of them.

That was a brief introduction to the user interface for the Mapping Editor and the operators as displayed in the canvas. The Oracle Warehouse Builder Data Modeling, ETL and Data Quality Guide includes additional details about the various windows and their purpose, as well as the toolbars and menus that are available. We will now look at the various operators that OWB provides us with in a little more detail.

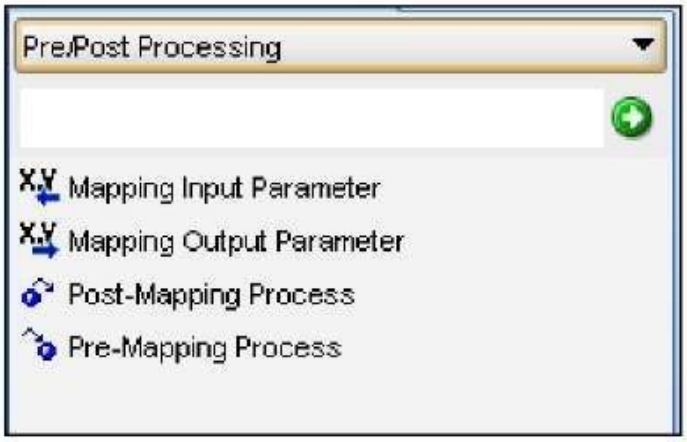
OWB operators

We'll discuss here the various operators using the same category breakdown that the Oracle Warehouse Builder Data Modeling, ETL, and Data Quality Guide **(http://download.oracle.com/docs/cd/E118 8 2_01/owb.112/e10935/transformdata_ intro.htm#i1127634)** uses in its section on the types of operators in topic 4 — Source and Target Operators, Transformations (Data Flow Operators), and Pre/Post Processing Operators. All of the operators are available to us from the Component Palette window in the Design Center when editing a mapping, so we can refer to it as we discuss each operator. The Component Palette has a dropdown that contains the same categories of operators we mentioned above. There are a couple of other categories that are for more advanced topics than we'll cover here (Pluggable Mapping and Real-Time Data Warehousing). They are described in the above

referenced documentation. The following screenshots display the complete list of operators in each of the three categories:


Data Sources/Targets
Constant
Construct Object
Cube Operator
Data Generator
Dimension Operator
Expand Object
External Table Operator
Flat File Operator
Materialized View Operator
Queue Operator
Sequence Operator
Table Operator
Varray Iterator
View Operator


Transformations
Aggregator
Anydata Cast
Deduplicator
Expression
Filter
Joiner
Lookup
Match Merge
Name and Address
Pivot
Set Operation
Sorter
Splitter
Subquery Filter
Table Function Operator
Transformation Operator
Unpivot


Pre/Post Processing
Mapping Input Parameter
Mapping Output Parameter
Post-Mapping Process
Pre-Mapping Process

Earlier in the ETL section of this topic, we discussed the various means at our disposal for performing ETL operations manually with applications such as SQL1Loader. We have also mentioned that OWB allows us to define the process graphically, and then generates the code for us as well. The bottom line is that OWB makes use of the existing facilities within the database and the utilities supplied with the database to accomplish the data load and transformations.
The operators we use will determine the kind of code that gets generated by OWB. Keep this in mind as we study these operators because it will help us understand some of the explanations that are supplied for the operators in the documentation.

Most of the operators will result in a PL/SQL mapping. So the explanations are in terms of the SQL or PL/SQL code element that is created for an operator.
As we can see, we have quite an extensive list of operators available to us and we won't have room here to talk about all of them. We won't need them all, and that is usually the case when designing mappings in OWB. We will be focusing on the main ones that we will need for our application and discuss some of the more common ones along with the ones we can see in the DATE_DIM_MAP. The operators found in the date_dim_map will be pointed out, so you can see an example by looking at that map. We'll talk about some of the operators in more detail as we actually begin to use them in the next topic. As much as possible, we will try to discuss the operators from a functionality standpoint without getting too bogged down by the actual code that is generated.
For the adventurous out there, you can take a look at the code that the Warehouse Builder will create for the mapping in the database. But unless you are an SQL coding wizard, you will become quickly overwhelmed. There is OWB-generated code I've had to deal with in a previous position that contains SQL insert statements that are over 650-lines long for a single statement. This is definitely not for the fainthearted. Have no fear, however; we don't have to dig into the code if we don't want to. This is the beauty of what a graphical interface does for us. For those who do love to delve into the code, there are ways to view it but that is definitely a more advanced topic.

Source and target operators

The Warehouse Builder provides operators that we will use to represent the sources of our data and the targets into which we will load data. We know we're going to be pulling data from non-Oracle database tables, and loading it into dimensions and cubes in our Oracle Database. We also saw in topic 2 how to import metadata from a flat file source. So there is another source type that we'll need to handle, a flat file. With that in mind, here are some of the operators we're going to potentially need:
• **Dimension Operator:** An operator that represents previously defined dimensions. As with our cube, our dimensions were defined in topic 4 and this operator will be used in our mapping to represent them. We can see an example of Dimension Operator in date_dim_map. This mapping is designed to load our date_dim dimension, and so an operator of the same name was created in it at

the end on the far right of the canvas. This operator includes logic for surrogate key generation also as well as support for slowly changing dimensions.

• **External Table Operator:** This operator represents external tables, which we have seen in topic 2. They can be used to access data stored in flat files as if they were tables. We will look at using an external table to access the flat file that we imported back in topic 2.

• **Table Operator:** This operator represents a table in the database. We will need to store data in tables in our Oracle Database at some point in the loading of data.

Those are the main operators we're going to need. There are a number of other operators that are defined for use as sources and targets in our mappings that can be very useful. The following are some of the more common operators:

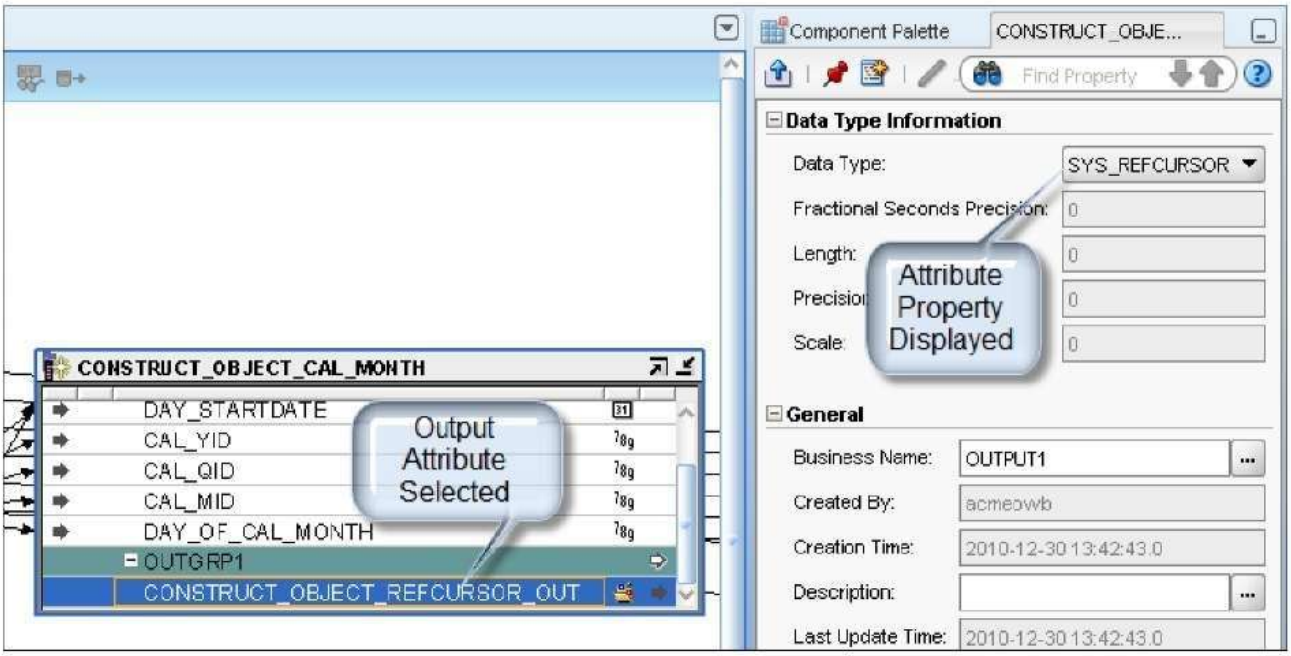
• Constant: Represents a constant value that is needed. It can be used to load a default value for a field that doesn't have any input from another source, for instance. The date_dim_map mapping contains a couple of constant values to represent hardcoded numbers. One is named one for the number 1, and one is named ZERO for a 0.

• **View Operator:** Represents a database view. Source data is frequently retrieved via a view in the source database that can pull data from multiple sources into a single, easily accessible view.
In the latest release of OWB we can now have inline views using this View Operator which allow us to define SQL right in the mapping and not have a physical view in the database. We won't be covering that in this topic but you can go to the OWB blog for more detailed information:

http://blogs.oracle.com/warehousebuilder/2010/08/owb_11gr2_mappings_and_inline_sql.html

• **Sequence Operator:** Can be used to represent a database sequence, which is an automatic generator of sequential unique numbers and is most often used for populating a primary key field.

• **Construct Object:** This operator can be used to actually construct an Oracle object type in our mapping. There are examples of this in date_dim_map, which builds the date_dim dimension. An object in this context refers to a PL/SQL object. We can see three Construct Object operators in date_dim_ MAP—for a calendar month (construct_object_cal_month), a calendar quarter (construct_object_cal_quarter), and a calendar year object (construct_object_cal_year). If we click on the attribute in the OUTGRPI output group of one of those construct operators, we can see in the Property Inspector window that it is of type SYS_REFCURSOR. An example is shown in the next screenshot with the CONSTRUCT_OBJECT_REFCURSOR_OUT attribute selected in the CONSTRUCT_OBJECT_CAL_MONTH object:



A SYS_REFCURSOR is a PL/SQL type that represents a cursor in PL/SQL. A cursor is used to point to the row of the result of the query that is defined for that cursor. This is a rather advanced topic to be covered in this topic, but is mentioned here as DATE_DIM_MAP contains some of this type.

These all represent sources and targets of data for our mappings. When we drag and drop one of these operators onto our canvas, it represents an actual database object. When created, every one of these will need to be bound to its underlying database object as we are using the relational storage option. For tables, attributes of the table operator will correspond to columns in the table, likewise for views and external tables. The same principle holds true for the cube and dimension operators. Attributes of the operator correspond to the attributes of the dimension or cube. If the dimension or cube is implemented relationally, they will correspond to the columns of the underlying table that is created.

A constant is implemented in the database as PL/SQL code using the PL/SQL syntax for representing a constant value. The value will be the value we would set on a constant's attribute. For sequences, constants are implemented as a database sequence object. The code is generated to invoke constants to retrieve the currval or nextval value from the underlying sequence as needed, depending on how we used it in our mapping. The currval variable will return the current value of the sequence, and the nextval variable will return the next value.

Transformations (data flow operators)

Sources and targets are good and we could end right there by connecting our sources directly to our targets. This would result in a complete mapping that would load our target from our source, but this

would mean there needs to be a one-to-one correspondence between our source and target. If that were the case, why bother creating a data warehouse target in the first place if it's only going to look exactly like the source? Just query the source data.

The true power of a data warehouse lies in the restructuring of the source data into a format that greatly facilitates the querying of large amounts of data over different time periods. For this, we need to transform the source data into a new structure. That is the purpose of the transformation (or data flow) operators. They are dragged and dropped into our mapping between our sources and targets. Then they are connected to those sources and targets to indicate the flow of data and the transformations that will occur on that data as it is being pulled from the source and loaded into the target structure. Some of the common data flow operators we'll see are as follows:

• **Aggregator:** There are times when source data is at a finer level of detail than we need. So we need to sum the data up to a higher level, or apply some other aggregation type function such as an average function. This is the purpose of the Aggregator operator. This is implemented behind the scenes using an SQL group by clause with an aggregation SQL function applied to the amount(s) we want to aggregate.
• **Deduplicator:** Sometimes our data records will contain duplicate combinations that we want to weed out so we're loading only unique combinations of data. The Deduplicator operator will do this for us. It's implemented behind the scenes with the distinct SQL function, which returns combinations of data elements that are unique.
• **Expression:** This represents an SQL expression that can be applied to the output to produce the desired result. Any valid SQL code for an expression can be used, and we can reference input attributes to include them as well as functions.
It's possible to write expressions in an Expression operator for which separate operators are predefined such as functions. We will generally get better performance out of our mappings if we use the prebuilt operators whenever possible rather than implement code in expressions. So, if there is an operator available, we'll use it and use an expression only if we have to.
• **Filter:** This will limit the rows from an output set to criteria that we specify. It is generally implemented in a where clause in SQL to restrict the rows that are returned. We can connect a filter to a source object, specify the filter criteria, and get only those records that we want in the output.
• **Joiner**: This operator will implement an SQL join on two or more input sets of data. A join takes records from one source and combines them with the records from another source using some combination of values that are common between the two. We will specify these common records as an attribute of the join. This is a convenient way to combine data from multiple input sources into one.
• **Lookup:** A Lookup operator (previously known as a Key Lookup) looks up data in a table based on some input criteria (the key) to return some information required by our mapping. It is similar to a

Table Operator that was discussed previously for sources and targets. However, a Lookup operator is geared toward returning a subset of rows from a table based on the key criteria we specify, rather than representing all the rows of a table, which the Table Operator does. It can look up data from a table, view, cube, or dimension.
The Lookup operator has been greatly enhanced in this new release of the Warehouse Builder. There is a blog posting that refers to the changes for additional reading at the following URL:

http://blogs.oracle.com/warehousebuilder/2010/09/owb_11gr2_lookup_operator.html

• **Pivot:** This operator can be useful if we have source records that contain multiple columns of data that is spread across columns instead of rows. For instance, we might have source records of sales data for the year that contain a column for each month of the year. But we need to save that information by month, and not by year. The Pivot operator will create separate rows of output for each of those columns of input.

1 Cube Operator: An operator that represents a cube that we have previously defined. We defined our cube back in topic 4 and this operator will be used to represent that cube in our mapping. It encapsulates logic like surrogate key lookup and early arriving facts (orphan management).

• **Set Operation:** This operator will allow us to perform an SQL set operation on our data such as a union (returning all rows from each of two sources, either ignoring the duplicates or including the duplicates) or intersect (which will return common rows from two sources).

• **Splitter:** This operator is the opposite of the Joiner operator. It will allow us to split an input stream of data rows into two separate targets based on the criteria we specify. It can be useful for shunting rows of data off to a side error table to flag them while copying the good rows into the main target.

• **Transformation Operator:** All these operators are transformation operators but there is one operator type specifically named "Transformation". This operator can be used to invoke a PL/SQL function or procedure with some of our source data as input to provide a transformation of data. For instance, the SQL trim() function can be represented by Transformation Operator to take a column value as input, and provide the value as output after having any whitespace trimmed from the value. This is just one example of a function that can be implemented with the Transformation Operator. There are numerous others available to us.

A Transformation Operator is an example of an operator that could be implemented in an Expression operator by simply invoking the trim() SQL function directly on an input value. But as we can implement a trim() directly using its own operator, we should do so for efficiency and consistency.

• **Table Function Operator:** A Table Function Operator can be seen in the date_dim_map map. There are three Table Function operators defined:
CAL_MONTH_TABLE_FUNCTION, CAL_QUARTER_TABLE_FUNCTION, and CAL_year_table_function. This kind of operator represents a Table Function, which is defined in PL/SQL

and is a function that can be queried like a table to return rows of information. The Table Function Operators are more advanced than we will be covering in this topic, but are mentioned here as date_dim_map includes them.

These are just some of the operators available to us for performing transformations on our data as it flows from source to target. Others are described in the Oracle Warehouse Builder Data Modeling, ETL, and Data Quality Guide as mentioned previously**(http://download.oracle.com/docs/cd/E11882_01/owb.112/ e10935/toc.htm).**

Other operators

There is a small group of operators that allow us to perform operations before the mapping process begins, or after the mapping process ends. These are the pre- and post-processing operators and mapping input and output operators. We can perform functions or procedures before or after a mapping runs, and can also accept input or provide output from a mapping process.

• **Mapping Input Parameter:** This operator allows us to pass a parameter(s) into a mapping process. It is very useful to make a mapping more generic by accepting a constant value as input that might change, rather than hardcoding it into the mapping. date_dim_map uses a Mapping Input Parameter operator as its very first operator on the left, which we discussed earlier when talking about Mapping Properties.
• **Mapping Output Parameter:** As the name suggests, this is similar to the Mapping Input Parameter operator, but provides a value as output from our mapping.
• **Post-Mapping Process**: Allows us to invoke a function or procedure after the mapping completes its processing. There may be some cleanup we want to do automatically such as deleting all the records from a table we're done with — perhaps a staging table that was used during the mapping process.
• **Pre-Mapping Process:** It's not too hard to figure out what this operator does. It allows us to invoke a function or procedure before the mapping process begins. Maybe our mapping needs to do a key lookup of a data value that is going to be stored in every row of output. But we don't want to invoke a Lookup operator for every record of input. So we could use a Pre-Mapping Process operator instead to invoke the function once at the beginning, which will make the returned value available for every row that is processed without having to re-invoke the procedure.
That concludes our discussion of some of the main operators we're going to encounter. If you are looking at the Oracle Warehouse Builder Data Modeling, ETL, and Data Quality Guide section that discusses these categories of operators, you no doubt noticed the other two categories we mentioned previously. They are also visible in the Component Palette drop down. Pluggable Mappings (of which there are three operators) are a more advanced feature that allows us to create a grouping of operators that can function as a single operator and be reused in other mappings and Real-Time

Data Warehousing mappings (of which there are only two operators) that are for creating real-time and batch mappings.

Summary

This topic has given us an overview of the Extract, Transform, and Load (ETL) process as well as the Warehouse Builder's support for designing our ETL process. We discussed the process of mapping and a little of what that involves in OWB. We took a look at the OWB Mapping Editor in the Design Center to get a feel for the windows available to us, and also looked at a list of some of the operators OWB provides for us to use in our mappings.
We're laying the groundwork here for the real fun that comes in the next topic where we get to put this knowledge to use in designing a mapping. In the next topic, we will also get to use some of these operators.