Chapter No.3
"Designing the Target Structure"
In this package, you will find:

A Biography of the author of the book
A preview chapter from the book, Chapter NO.3 "Designing the Target Structure"
A synopsis of the book’s content
Information on where to buy this book

About the Author

**Bob Griesemer** has over 27 years of software and database engineering/DBA experience in both government and industry, solving database problems, designing and loading data warehouses, developing code, leading teams of developers, and satisfying customers. He has been working in various roles involving database development and administration with the Oracle Database with every release since Version 6 of the database from 1993 to the present. He has also been performing various tasks, including data warehouse design and implementation, administration, backup and recovery, development of Perl code for web-based database access, writing Java code utilizing JDBC, migrating legacy databases to Oracle, and developing Developer/2000 Oracle Forms applications. He is currently an Oracle Database Administrator Certified Associate, and is employed by the Northrop Grumman Corporation, where he is currently a Senior Database Analyst on a large data warehouse project.

For More Information:

I'd like to thank David Allan of the Oracle Warehouse Builder development team at Oracle for agreeing to review the book and for putting up with my numerous questions and requests for clarification. His input was extremely beneficial in explaining new functionality of OWB. I'd like to acknowledge my co-worker Ed Cody, whose work on his book, *The Business Analyst's Guide to Oracle Hyperion Interactive Reporting 11*, with Packt Publishing, inspired me to get started on the second edition of my book. Lastly and most importantly, of course, to my family, wife Lynn and children Robby, Melanie, Hilary, Christina, Millie and Mikey, thanks for being the inspiration and motivation behind everything I do.
Oracle Warehouse Builder
11gR2: Getting Started

Competing in today's world requires a greater emphasis on strategy, long-range planning, and decision making, and this is why businesses are building data warehouses. Data warehouses are becoming more and more common as businesses have realized the need to mine the information that is stored in electronic form. Data warehouses provide valuable insight into the operation of a business and how best to improve it. Organizations need to monitor their processes, define policy, and at a more strategic level, define the visions and goals that will move the company forward in the future. If you are new to data warehousing in general, and to Extract, Transform, and Load (ETL) in particular, and need a way to get started, the Oracle Warehouse Builder is a great application to use to build your warehouse. The Oracle Warehouse Builder (OWB) is a tool provided by Oracle that can be used at every stage of the implementation of a data warehouse right from the initial design and creation of the table structure to ETL and data-quality auditing.

We will build a basic data warehouse using the latest release of Oracle Warehouse Builder, 11gR2. It has the ability to support all phases of the implementation of a data warehouse from designing the source and target information, the mappings to map data from source to target, the transformations needed on the data, and building the code to implementing the mappings to load the data. You are free to use any or all of the features in your own implementation.

What This Book Covers

This book is an introduction to the Oracle Warehouse Builder (OWB). This is an introductory, hands-on book so we will be including in this book the features available in Oracle Warehouse Builder 11gR2 that we will need to build our first data warehouse. The chapters are in chronological order to flow through the steps required to build a data warehouse with a couple of chapters at the end on special topics, including one devoted to a major new feature of OWB 11gR2, code templates. So if you are building your first data warehouse, it is a good idea to read through each chapter sequentially to gain maximum benefit from the book. Those who have already built a data warehouse and just need a refresher on some basics can skip around to whatever topic they need at that moment.

We'll use a fictional toy company, ACME Toys and Gizmos, to illustrate the concepts that will be presented throughout the book. This will provide some context to the information presented to help you apply the concepts to your own organization. We'll actually be constructing a simple data warehouse for the ACME Toys and Gizmos company. At the end of the book, we'll have all the code, scripts, and saved metadata that

For More Information:  
was used. So we can build a data warehouse for practice, or use it as a model for building another data warehouse.

Chapter 1, An Introduction to Oracle Warehouse Builder, starts off with a highlevel look at the architecture of OWB and the steps for installing it. It covers the schemas created in the database that are required by OWB, and touches upon some installation topics to provide some further clarification that is not necessarily found in the Oracle documentation. Most installation tasks can be found in the Oracle README files and installation documents, and so they won't be covered in depth in this book.

Chapter 2, Defining and Importing Source Data Structures, covers the initial task of building a data warehouse from scratch, that is, determining what the source of the data will be. OWB needs to know the details about what the source data structures look like and where they are located in order to properly pull data from them using OWB. This chapter also covers how to define the source data structures using the Data Object Editor and how to import source structure information. It talks about three common sources of data—if at files, Oracle Databases, and Microsoft SQL Server databases—while discussing how to configure Oracle and OWB to connect to these sources.

Chapter 3, Designing the Target Structure, explains designing the data warehouse target. It covers some options for defining a data warehouse target structure using relational objects (star schemas and snowflake schemas) and dimensional objects (cubes and dimensions). Some of the pros and cons of the usage of these objects are also covered. It introduces the Warehouse Builder for design and starts with the creation of a target user and module.

Chapter 4, Creating the Target Structure in OWB, implements the design of the target using the Warehouse Builder. It has step-by-step explanations for creating cubes and dimensions using the wizards provided by OWB.

Chapter 5, Extract, Transform, and Load Basics, introduces the ETL process by explaining what it is and how to implement it in OWB. It discusses whether to use a staging table or not, and describes mappings and some of the main operators in OWB that can be used in mappings. It introduces the Warehouse Builder Mapping Editor, which is the interface for designing mappings.

Chapter 6, ETL: Putting it Together, is about creating a new mapping using the Mapping Editor. A staging table is created with the Data Object Editor, and a mapping is created to map data directly from the source tables into the staging table. This chapter explains how to add and edit operators, and how to connect them together. It also discusses operator properties and how to modify them.

Chapter 7, ETL: Transformations and Other Operators, expands on the concept of building a mapping by creating additional mappings to map data from the staging table into cube and dimensions. Additional operators are introduced for doing transformations of the data as it is loaded from source to target.

For More Information:
Chapter 8, Validating, Generating, Deploying, and Executing Objects, covers in great detail the validation of mappings, the generation of the code for mappings and objects, and deploying the code to the target database. This chapter introduces the Control Center Service, which is the interface with the target database for controlling this process, and explains how to start and stop it. The mappings are then executed to actually load data from source to target. It also introduces the Control Center Manager, which is the user interface for interacting with the Control Center Service for deploying and executing objects.

Chapter 9, Extra Features, covers some extra features provided in the Warehouse Builder that can be very useful for more advanced implementations as mappings get more numerous and complex. The metadata change-management features of OWB are discussed for controlling changes to mappings and objects. This includes the recycle bin, cutting.copying and pasting objects to make copies or backups, the snapshot feature, and the metadata loader facility for exporting metadata to a file. Keeping objects synchronized as changes are made is discussed, and so is the autobinding of tables to dimensional objects. Lastly, some additional online references are provided for further study and reference.

Chapter 10, Code Template Mappings, covers a major new feature of the 11gR2 release of OWB—code templates, which are the knowledge module functionality brought over into OWB from Oracle Data Integrator. It includes detailed descriptions of implementing a JDBC connection to an external database and the implementation of a code template mapping to access it. It includes discussion of the main code templates provided by default with OWB 11gR2 and describes everything you need to know to implement your first code template mapping.

For More Information:
We have our entire source structures defined in the Warehouse Builder. But before we can do anything with them, we need to design what our target data warehouse structure is going to look like. When we have that figured out, we can start mapping data from the source to the target. So, let's design our target structure. First, we're going to take a look at some design topics related to a data warehouse that are different from what we would use if we were designing a regular relational database. We'll then discuss what our design will look like, and after that we'll be ready to move right into creating that design using the Warehouse Builder in the next chapter.

The specific topics we'll discuss in this chapter include the following:

- Data warehouse design
  - Dimensional design
  - Cube and dimensions
  - Dimensional Model Implementation
  - Relational (star schema)
  - Multidimensional (OLAP)
  - Designing the ACME data warehouse
  - Identifying dimensions
  - Designing the cube

- Data Warehouse Design in OWB
  - Creating a target user and module
  - OWB design objects

For More Information:  
Data warehouse design
When it comes to the design of a data warehouse, there is basically one option that makes the most sense for how we will structure our database and that is the **dimensional** model. This is a way of looking at the data from a business perspective that makes the data simple, understandable, and easy to query for the business end user. It doesn't require a database administrator to be able to retrieve data from it.

When looking at the source databases in the last chapter, we saw a normalized method of modeling a database. A normalized model removes redundancies in data by storing information in discrete tables, and then referencing those tables when needed. This has an advantage for a transactional system because information needs to be entered at only one place in the database, without duplicating any information already entered. For example, in the ACME Toys and Gizmos transactional database, each time a transaction is recorded for the sale of an item at a register, a record needs to be added only to the transactions table. In the table, all details regarding the information to identify the register, the item information, and the employee who processed the transaction do not need to be entered because that information is already stored in separate tables. The main transaction record just needs to be entered with references to all that other information.

This works extremely well for a transactional type of system concerned with daily operational processing where the focus is on getting data into the system. However, it does not work well for a data warehouse whose focus is on getting data out of the system. Users do not want to navigate through the spider web of tables that compose a normalized database model to extract the information they need. Therefore, dimensional models were introduced to provide the end user with a flattened structure of easily queried tables that he or she can understand from a business perspective.

**Dimensional design**
A dimensional model takes the business rules of our organization and represents them in the database in a more understandable way. A business manager looking at sales data is naturally going to think more along the lines of "How many gizmos did I sell last month in all stores in the south and how does that compare to how many I sold in the same month last year?" Managers just want to know what the result is, and don't want to worry about how many tables need to be joined in a complex query to get that result. In the last chapter, we saw how many tables would have to be joined together in such a query just to be able to answer a question like the one above. A dimensional model removes the complexity and represents the data in a way that end users can relate to it more easily from a business perspective.

For More Information:
Users can intuitively think of the data for the above question as a cube, and the edges (or dimensions) of the cube labeled as stores, products, and time frame. So let's take a look at this concept of a cube with dimensions, and how we can use that to represent our data.

**Cube and dimensions**

The *dimensions* become the business characteristics about the sales, for example:

- A time dimension — users can look back in time and perform time series analysis, such as how a quarter compares to the same quarter last year
- A store dimension — information can be retrieved by store and location
- A product dimension — various products for sale can be broken out

Think of the dimensions as the edges of a cube, and the intersection of the dimensions as the measure we are interested in for that particular combination of time, store, and product. A picture is worth a thousand words, so let's look at what we're talking about in the following image:
Notice what this cube looks like. How about a Rubik's Cube? We're doing a data warehouse for a toy store company, so we ought to know what a Rubik's cube is! If you have one, maybe you should go get it now because that will exactly model what we're talking about. Think of the width of the cube, or a row going across, as the product dimension. Every piece of information or measure in the same row refers to the same product, so there are as many rows in the cube as there are products. Think of the height of the cube, or a column going up and down, as the store dimension. Every piece of information in a column represents one single store, so there are as many columns as there are stores. Finally, think of the depth of the cube as the time dimension, so any piece of information in the rows and columns at the same depth represent the same point in time. The intersection of each of these three dimensions locates a single individual cube in the big cube, and that represents the measure amount we're interested in. In this case, it's dollar sales for a single product in a single store at a single point in time.

But one might wonder if we are restricted to just three dimensions with this model. After all, a cube has only three dimensions — length, width, and depth. Well, the answer is no. We can have many more dimensions than just three. In our ACME example, we might want to know the sales each employee has accomplished for the day. This would mean we would need a fourth dimension for employees. But what about our visualization above using a cube? How is this fourth dimension going to be modeled? And no, the answer is not that we're entering the Twilight Zone here with that "dimension not only of sight and sound but of mind..." We can think of additional dimensions as being cubes within a cube. If we think of an individual intersection of the three dimensions of the cube as being another cube, we can see that we've just opened up another three dimensions to use — the three for that inner cube. The Rubik's Cube example used above is good because it is literally a cube of cubes and illustrates exactly what we're talking about.

We do not need to model additional cubes. The concept of cubes within cubes was just to provide a way to visualize further dimensions. We just model our main cube, add as many dimensions as we need to describe the measures, and leave it for the implementation to handle.

This is a very intuitive way for users to look at the design of the data warehouse. When it's implemented in a database, it becomes easy for users to query the information from it.
Implementation of a dimensional model in a database

We have seen how a dimensional model is preferred over a normalized model for designing a data warehouse. Now before we finalize our model for the ACME Toys and Gizmos data warehouse, let's look at the implementation of the model to see how it gets physically represented in the database. There are two options: a relational implementation and a multidimensional implementation. The relational implementation, which is the most common for a data warehouse structure, is implemented in the database with tables and foreign keys. The multidimensional implementation requires a special feature in a database that allows defining cubes directly as objects in the database. Let's discuss a few more details of these two implementations. But we will look at the relational implementation in greater detail as that is the one we're going to use throughout the remainder of the book for our data warehouse project.

Relational implementation (star schema)

Back in Chapter 2, we saw how ACME's POS Transactional database and Order Entry databases were structured when we did our initial analysis. The diagrams presented showed all the tables interconnected, and we discussed the use of foreign keys in a table to refer to a row in another table. That is fundamentally a relational database. The term relational is used because the tables in it relate to each other in some way. We can't have a POS transaction without the corresponding register it was processed on, so those two relate to each other when represented in the database as tables.

For a relational data warehouse design, the relational characteristics are retained between tables. But a design principle is followed to keep the number of levels of foreign key relationships to a minimum. It's much faster and easier to understand if we don't have to include multiple levels of referenced tables. For this reason, a data warehouse dimensional design that is represented relationally in the database will have one main table to hold the primary facts, or measures we want to store, such as count of items sold or dollar amount of sales. It will also hold descriptive information about those measures that places them in context, contained in tables that are accessed by the main table using foreign keys. The important principle here is that these tables that are referenced by the main table contain all the information they need and do not need to go down any more levels to further reference any other tables.

For More Information:
Designing the Target Structure

The ER diagram of such an implementation would be shaped somewhat like a star, and thus the term **star schema** is used to refer to this kind of an implementation. The main table in the middle is referred to as the **fact table** because it holds the facts, or measures that we are interested in about our organization. This represents the cube that we discussed earlier. The tables surrounding the fact table are known as dimension tables. These are the dimensions of our cube. These tables contain descriptive information, which places the facts in a context that makes them understandable. We can't have a dollar amount of sales that means much to us unless we know what item it was for, or what store made the sale, or any of a number of other pieces of descriptive information that we might want to know about it.

It is the job of data warehouse design to determine what pieces of information need to be included. We'll then design dimension tables to hold the information. Using the dimensions we referred to above in our cube discussion as our dimension tables, we have the following diagram that illustrates a star schema:

Of course our star only has three points, but with a much larger data warehouse of many more dimensions, it would be even more star-like. Keep in mind the principle that we want to follow here of not using any more than one level of foreign key referencing. As a result, we are going to end up with a **de-normalized** database structure. We discussed normalization back in Chapter 2, which involved the use of foreign key references to information in other tables to lessen the duplication and improve data accuracy. For a data warehouse, however, the query time and simplicity is of paramount importance over the duplication of data. As for the data accuracy, it's a read-only database so we can take care of that up front when we load the data. For these reasons, we will want to include all the information we need right in the dimension tables, rather than create further levels of foreign key references. This is the opposite of normalization, and thus the term de-normalized is used.

For More Information:
Let's look at an example of this for ACME Toys and Gizmos to get a better idea of what we're talking about with this concept of de-normalization. Every product in our stores is associated with a department. If we have a dimension for product information, one of the pieces of information about the product would be the department it is in. In a normalized database, we would consider creating a department table to store department descriptions with one row for each department, and would use a short key code to refer to the department record in the product table.

However, in our data warehouse, we would include that department information, description and all, right in the product dimension. This will result in the same information being duplicated for each product in the department. What that buys us is a simpler structure that is easier to query and more efficient for retrieving information from, which is key to data warehouse usability. The extra space we consume in repeating the information is more than paid for in the improvement in speed and ease of querying the information. That will result in a greater acceptance of the data warehouse by the user community who now find it more intuitive and easier to retrieve their data.

In general, we will want to de-normalize our data warehouse implementation in all cases, but there is the possibility that we might want to include another level—basically a dimension table referenced by another dimension table. In most cases, we will not need nor want to do this and instances should be kept to an absolute minimum; but there are some cases where it might make sense.

This is a variation of the star schema referred to as a snowflake schema because with this type of implementation, dimension tables are partially normalized to pull common data out into secondary dimension tables. The resulting schema diagram looks somewhat like a snowflake. The secondary dimension tables are the tips of the snowflake hanging off the main dimension tables in a star schema.

In reality, we'd want at the most only one or two of the secondary dimension tables; but it serves to illustrate the point. A snowflake dimension table is really not recommended in most cases because of ease-of-use and performance considerations, but can be used in very limited circumstances. The Kimball book on Dimensional Modeling was referred to at the beginning of Chapter 2. This book discusses some limited circumstances where it might be acceptable to implement a snowflake design, but it is highly discouraged for most cases.

Let's now talk a little bit about the multidimensional implementation of a dimensional model in the database, and then we'll design our cube and dimensions specifically for the ACME Toys and Gizmos Company data warehouse.
Designing the Target Structure

Multidimensional implementation (OLAP)

A multidimensional implementation or OLAP (online analytic or analytical processing) requires a database with special features that allow it to store cubes as actual objects in the database, and not just tables that are used to represent a cube and dimensions. It also provides advanced calculation and analytic content built into the database to facilitate advanced analytic querying. Oracle’s Essbase product is one such database and was originally developed by Hyperion. Oracle recently acquired Hyperion, and is now promoting Essbase as a tool for custom analytics and enterprise performance management applications. The Oracle Database Enterprise Edition has an additional feature that can be licensed called OLAP that embeds a full-featured OLAP server directly in an Oracle database. This is an option organizations can leverage to make use of their existing database.

These kinds of analytic databases are well suited to providing the end user with increased capability to perform highly optimized analytical queries of information. Therefore, they are quite frequently utilized to build a highly specialized data mart, or a subset of the data warehouse, for a particular user community. The data mart then draws its data to load from the main data warehouse, which would be a relational dimensional star schema. A data warehouse implementation may contain any number of these smaller subset data marts.

We’ll be designing dimensionally and implementing relationally, so let’s now design our actual dimensions that we’ll need for our ACME Toys and Gizmos data warehouse, and talk about some issues with the fact data (or cube) that we’ll need. This will make the concepts we just discussed more concrete, and will form the basis for the work we do in the rest of the book as we implement this design. We’ll then close out this chapter with a discussion on designing in the Warehouse Builder, where we’ll see how it can support either of these implementations.

We have seen the word dimension used in describing both a relational implementation and a multidimensional implementation. It is even in the name of the second implementation method we discussed, so why does the relational method use it also? In the relational case, the word is used more as an adjective to describe the type of table taken from the name of the model being implemented; whereas in the multidimensional model it’s more a noun, referring to the dimension itself that actually gets created in the database. In both cases, the type of information conveyed is the same—descriptive information about the facts or measures—so its use in both cases is really not contradictory. There is a strong correlation between the fact table of the relational model and the cube of the dimensional model, and between the dimension tables of the relational model and the dimensions of the dimensional model.

For More Information:
Designing the ACME data warehouse

We have chosen to use a dimensional model for our data warehouse, so we'll define a cube with dimensions to represent our information. Let's lay out a basic structure of information we want each to contain. We'll begin with the dimensions, since they are going to provide the context for the measure(s) we will want to store in our cube.

Identifying the dimensions

To know what dimensions to design for, we need to know what business process we're going to be supporting with our data warehouse. Is management concerned with daily inventory? How about daily sales volume? This information will guide us in selecting the correct parts of the business to model with our dimensions.

We are going to support the sales managers in managing the daily sales of the ACME Toys and Gizmos Company, and they have already given us an example of the kind of question they want answered from their data warehouse, as we saw earlier. We used that to illustrate the cube concept and to show a star schema representation of it, so the information shows us the dimensions we need. Since management is concerned with daily sales, we need some kind of date/time dimension that will provide us the context for the sales data indicating what day the sale transaction took place.

We can pretty much be guaranteed that we will need a time/date type dimension for any data warehouse we design, since one of the main features of data warehouses is to provide time-series type analytical query capabilities (as we talked about earlier).

Are we going to need both the time and the date in this dimension, or will just the date be sufficient? We can get an answer to this question by also looking back at our business process, which showed that management is concerned with daily sales volume. Also, the implementation of the time dimension in OWB does not include the time of day since it would have to include 24 hours of time values for each day represented in the dimension due to the way it implements the dimension. In the future if time is needed, there are options for creating a separate dimension just for modeling time of day values. For our initial design, we'll call our time related dimension a Date dimension just for added clarity.

For More Information:
Another dimension we have included is to model the product information. Each sale transaction is for a particular product, and management has indicated they are concerned about seeing how well each product is selling. So we will include a dimension that we shall call Product. At a minimum we need the product name, a description of the product, and the cost of the product as attributes of our product dimension—so we'll include those in our logical model.

So far we have a Date dimension to represent our time series and a Product dimension to represent the items that are sold. We could stop there. Management would then be able to query for sales data for each day for each product sold by ACME Toys and Gizmos, but they wouldn't be able to tell where the sale took place. Another key piece of information the management would like to be able to retrieve is how well the stores are doing compared to each other for daily sales. Unless we include some kind of a location dimension, they will not be able to tell that. That is why we have included a third dimension called Store. It is used to maintain the information about the store that processed the sales transaction. For attributes of the store dimension, we can include the store name and address at a minimum to identify each store.

These dimensions should be enough to satisfy the management's need for querying information for this particular business process—the daily sales. We could certainly include a large number of other dimensions, but we'll stop here to keep this simple for our first data warehouse. We can now consider designing the cube and what information to include in it.

**Designing the cube**

In the case of the ACME Toys and Gizmos Company, we have seen that the main measure the management is concerned about is daily sales. There are other numbers we could consider such as inventory numbers: How much of each item is on hand? However, the inventory is not directly related to daily sales and wouldn't make sense here. We can model an inventory system in a data warehouse that would be separate from the sales portion. But for our purposes, we’re going to model the sales. Therefore, our main measure is going to be the dollar amount of sales for each item.

A very important topic to consider at this point is what will be the grain of the measure—the sales data—that we're going to store in our cube? The grain (or granularity) is the level that the sales number refers to. Since we're using sales as the measure, we'll store a sales number; and from our dimensions, we can see that it will be for a given date in a given store and for a given product. Will that number be the total of all the sales for that product for that day? Yes, so it satisfies our design criteria of providing daily sales volume for each product. That is the smallest and lowest level of sales data we want to store. This is what we mean by the grain or granularity of the data.
Levels/hierarchies

A dimensional model is naturally able to handle this concept of the different levels of data by being able to model a hierarchy within a dimension. The time/date dimension is an easy example of using various levels. Add up the daily totals to get the totals for the month, and add up 12 monthly totals to get the yearly sales. The time/date dimension just needs to store a value to indicate the day, month, and year to be able to provide a view of the data at each of those levels. Combining various levels together then defines a hierarchy. By storing data at the lowest level, we make available the data for summing at higher levels. Likewise, from a higher level, the data is then available to drill down to view at a lower level. If we were to arbitrarily decide to store the data at a higher level, we would lose that flexibility. We’ll discuss this further in the next chapter when we build our time dimension in the Warehouse Builder.

In this case, we have a source system—the POS Transactional system—that maintains the dollar amount of sales for each line item in each sales transaction that takes place. This can provide us the level of detail we will want to capture and maintain in our cube, since we can definitely capture sales for each product at each store for each day. We have found out that the POS Transactional system also maintains the count of the number of a particular item sold in the transaction. This is an additional measure we will consider storing in our cube also, since we can see that it is at the same grain as the total sales. The count of items would still pertain to that single transaction just like the sales amount, and can be captured for each product, store, and even date.

The only other pieces of information our cube is going to contain are pointers to the dimensions. In the relational model, the fact table would contain columns for the dollar amount, the quantity, the unit cost, and then foreign keys for each of the dimension tables.
There are times when it's valid in dimensional design to include more descriptive information right in the cube, rather than create a dimension for it. There may be some particularly descriptive piece of information that stands all by itself, which is not associated with anything else or whose additional descriptive information has already been included in other dimensions. In that case, it wouldn't make sense to create a whole dimension just for it; so it is included directly in the fact table or cube. This is referred to as a degenerate dimension. It is explained in more detail in the Kimball book on dimensional modeling we talked about earlier. There are many other aspects to dimensional design that we don't have the space to cover here, but are covered in the Kimball book in more detail. It would be a good idea for you to read this book or a similar one to get a better understanding of the detailed dimensional modeling concepts such as this.

Our design is drawn out in a star schema configuration showing the cube, which is surrounded by the dimensions with the individual items of information (attributes) we'll want to store for each. It looks like the following:

![Star Schema Diagram]

OK, we now have a design for our data warehouse. It's time to see how OWB can support us in entering that design and generating its physical implementation in the database.

For More Information:
Data warehouse design in OWB

The Warehouse Builder contains a number of objects, which we can use in designing our data warehouse, that are either relational or dimensional. OWB currently supports designing a target schema only in an Oracle database, and so we will find the objects all under the Oracle node in the Projects tab. Let's launch Design Center now and have a look at it. But before we can see any objects, we have to have an Oracle module defined to contain the objects. If you've been following along and working through the examples in this book, so far you should have one module already defined for the ACME website orders database—ACME_WS_ORDERS. We created this in the last chapter when we imported our metadata from that source. If that is the case, our Projects tab window will look similar to the following:

Creating a target user and module

We need a different module to create our target objects in. So before going any further, let's create a new module in the Projects tab for our target to hold our data warehouse design objects. However, before we can do that, we should have a target schema defined in the database that will hold our target objects when we deploy them.
So far we have discussed many different components such as the repository, workspaces, the design center, and so on. So, it can be confusing to know exactly where our main data warehouse is going to be located. The target schema is going to be the main location for the data warehouse. When we talk about our "data warehouse" after we have it all constructed and implemented, the target schema is what we will be referring to. Amid all these different components we discussed that compose the Warehouse Builder, the target is where the actual data warehouse will be built. Our design will be implemented there, and the code will be deployed to that schema by OWB to load the target structure with data from the sources.

Every target module must be mapped to a target user schema. Back in Chapter 1, when we ran the Repository Assistant to create the repository and workspace, we created the acmeowb user as the repository owner and mentioned that this user can be a deployment target for our data warehouse. However, it does not have to be the target user. It’s a good idea to create a separate user schema to become the target so that user roles in our database can be kept separate. Using the OWB repository owner schema would mean our target data warehouse would have to be on the same database server as our repository. In large installations, that will most likely not be the case. So for maximum flexibility, we’re going to create a separate user schema. In our case, that user will be created in the same database as the repository; but it can be moved to another database easily if we expand and add more servers.

Creating a target user

There are a couple of ways we can go about creating our target user—create the user directly in the database and then add to OWB, or use OWB to physically create the user. If we have to create a new user, and if it's on the same database as our repository and workspaces, it's a good idea to use OWB to create the user, especially if we are not that familiar with the SQL command to create a user. However, if our target schema were to be in another database on another server, we would have to create the user there. It’s a simple matter of adding that user to OWB as a target, which we’ll see in a moment. Let’s begin in the Design Center under the Globals tab. We talked about that Globals tab back in our introduction to the Design Center in Chapter 2. There we said it was for various objects that pertained to the workspace as a whole.

One of those object types is a Users object that exists under the Security node as shown here:
Right-click on the **Users** node and select **New User...** to launch the **Create User** dialog box as shown here:
Designing the Target Structure

With this wizard, we are creating a workspace user. We create a workspace user by selecting a database user that already exists or create a new one in the database. We'll just click the Next button to move on to step 1 as shown next:

If we already had a target user created in the database, this is where we would select it. We're going to click on the Create DB User... button to create a new database user.

We need to enter the system username and password as we need a user with DBA privileges in the database to be able to create a database user. We then enter a username and password for our new user ACME_DWH, for the ACME data warehouse. We can also specify the default and temporary tablespace for our new user, which we'll leave at the defaults. The dialog will appear like the following when completely filled in:

For More Information:
The new user will be created when you click on the OK button, and will appear in the right hand window of the Create User dialog already selected for us. Click on the Next button and we’ll be presented with the second step of the user creation process, whether to create a location using the user credentials or not as shown in the following image:
Designing the Target Structure

We discussed locations in the last chapter and saw how they were required for the Warehouse Builder to know where to connect to for the various tables and other database objects defined in modules we've defined in our project. Since we're going to use this new user we've just created as an eventual target for creating our data warehouse in then we will need to leave this checkbox checked so it creates a location based on this user. We could be just creating another authorized database user for accessing the workspace but not intending to use it as a target for any object creation in which case we wouldn't need a location defined for it. We'll leave the check box checked and click the Next button to proceed.

The final screen is just the Summary screen indicating the user to be created and whether a location will be created or not. We'll just click the Finish button and the user will be registered with the workspace, and we'll see the new username if we expand the Users node under Security in the Globals tab. Since we had indicated that we wanted a location created also, a location for the user will be evident on the Locations tab under the Locations...Databases...Oracle node. We can continue with creating our target module now that we have a user defined in the database to map to.

Notice that we could indicate whether we wanted a location created or not but had no way to specify the database location information. This is because it creates the user on the local database we were connected to when we logged into the Design Center, which is the location of our repository and workspaces. Due to this, this method can only be used to create the user if it is on the local database. In the next section where we create our target module, we'll get to specify the location and that dialog box will allow us to specify a remote database if needed.

Create a target module

We'll follow the same steps as we did in the last chapter where we created the ACME_WS_ORDERS module. Right-click on the Oracle object under Databases and select New Oracle Module... from the pop-up menu to launch the Create Module Wizard and step through the process. We'll name this module ACME_DWH for ACME Data Warehouse.

For More Information:
The next step is for creating or selecting a location to use. Since we just created the user to use as the target user and had the Warehouse Builder create the location automatically for us there is a location available now on the local server we can use. We'll just click the drop down and select the location labeled ACME_DWH_LOCATION. If we're creating our own test system, the source location may very well be the same as our target. But in real-world situations, it will likely be in a different database on a different server. If we had created a target user schema on a different database, this is the point at which we would be able to enter the connection information for that user in order to associate our target module with that user and make it a target. We would just create a new location by clicking the Edit button on the default ACME_DWH_LOCATION to specify the connection details for that other database.

We're not going to create a new location but will be selecting an existing one and for reference, the Step 2 screen should look like the following for selecting the location of the target module:
Designing the Target Structure

The **User Name** is the user we just created for this very purpose in the previous section. There is no password set for that user in the location yet but it will prompt us for that the first time we attempt to use it. The **Host** setting of **Win7VM** will be whatever the name is of the computer its running on so will vary. The Warehouse Builder uses the actual local computer name when creating the location for us rather than **localhost** but either will do.

If we had specified a user on a remote database the location information (**Host**, **Port**, and **Service Name**) would specify a user in another database if needed. If our user were not in this database, we would have just entered his or her appropriate host and port for the location and the service name of that remote database.

Now that we have our target database schema and a target module defined, which is associated with a location pointing to that target schema, we will now have two Oracle modules under our Oracle object in the Projects tab. We can continue our discussion of the design objects available to us in the Warehouse Builder for designing our database. First, let's make sure we save our work so far by using the **Ctrl+S** key combination or by selecting **Design | Save All** from the main menu.

**OWB design objects**

Looking at our **Projects tab** window with our target Oracle module expanded, we can see a number of objects that are available to us as shown here:

For More Information:  
There are objects that are relational such as **Tables**, **Views**, **Materialized Views**, and **Sequences**. Also, there are dimensional objects such as **Cubes** and **Dimensions**. We just discussed relational objects versus dimensional objects. We have decided to model our database dimensionally and this will dictate the objects we create. From the standpoint of providing the best model of our business rules and representing what users want to see, the dimensional method is the way to go as we already discussed. Most data warehouse implementations we encounter will use a dimensional design. It just makes more sense for matching the business rules the users are familiar with and providing the types of information the user community will want to extract from the database.

We are thinking dimensionally in our design, but what about the underlying physical implementation? We discussed the difference between the relational and multidimensional physical implementation of a database, and now it’s time to see how we will handle that here. The Warehouse Builder can help us tremendously with that because it has the ability to design the objects logically using cubes and dimensions in a dimensional design. It also has the ability to implement them physically in the underlying database as either a relational structure or a dimensional structure simply by checking a box.

In general, which option should be chosen? The relational implementation is best suited to large amounts of data that tend to change more frequently. For this reason, the relational implementation is usually chosen for the main data warehouse schema by most implementers of a data warehouse. It is much better suited to handling the large volumes of data that are imported frequently into the data warehouse. The multidimensional implementation is better suited to applications where heavy analytic processing is required, and so is a good candidate for the data marts that will be presented to users.

To be able to implement the design physically as a dimensional implementation with cubes and dimensions, we need a database that is designed specifically to support **OLAP** as we discussed previously. If that is not available, then the decision is made for us. In our case, when we installed the Oracle database in Chapter 1, we installed the Enterprise Edition with default options, and that includes the OLAP feature in the database, so we have a choice to make. Since we're installing our main data warehouse target schema, we'll choose the relational implementation.

---

**For More Information:**

Designing the Target Structure

For a relational implementation, the Warehouse Builder actually provides us two options for implementing the database: a pure relational option and the relational OLAP option. If we were to have the OLAP feature installed in our database, we could choose to still have the cubes and dimensions implemented physically in a relational format. We could have it store metadata in the database in the OLAP catalog, and so multidimensional features such as aggregations would be available to us. We could take advantage of the relational implementation of the database for handling large volumes of data, and still implement a query or reporting tool such as Oracle Discoverer or Oracle Business Intelligence Enterprise or Standard Edition (OBIEE) to access the data that made use of the OLAP features. The pure relational option just depends on whether we choose to deploy only the data objects and not the OLAP metadata. In reality, most people choose either the pure relational or the multidimensional. If they want both, they implement separate data marts. In fact, the default when creating dimensional objects and selecting relational for the implementation is to only deploy data objects. This case would allow us to use the dimensional objects to load the data warehouse without needing to deploy OLAP catalog objects representing them. Tools like OBIEE or Discoverer can still derive Business Intelligence objects for dimensional oriented models in those tools using just these relational dimensional objects in the database.

Just to be clear, does all this mean that if we haven't paid for the OLAP feature for our database, we can only design our data warehouse using the relational objects; and therefore must our decision to design dimensionally change? The answer to that would be an emphatic no, since we just mentioned how OWB will let us design dimensional objects, cubes and dimensions, and then implement them physically in the database as relational objects. The benefit is that the same dimensional design can be implemented at a later time in an OLAP database just by changing a single setting. There are features of the Warehouse Builder for handling dimensional features automatically for us, such as levels, surrogate keys, and slowly changing dimensions (all of which we'll talk about later) that designing dimensionally provides us. We would have to implement these manually if we designed our own tables. Most people who use the Warehouse Builder will use it in that way, so we'll definitely want to make use of that feature to maximize the usefulness of the tools to us. This provides us with flexibility and it is the way we are going to proceed with our design. We'll design dimensionally using a cube and dimensions, and then can implement it either relationally or dimensionally when we're ready.

For More Information:
Summary

We have now gone through the process of designing the target structure for our data warehouse. We began with a very high-level overview of data warehouse design topics, then talked about dimensional design and the relational versus multidimensional implementation, and then we discussed the differences between them. As was mentioned earlier, there are other books that are devoted solely to this topic and it would be good to read one or more of them to learn more about design than we've been able to cover here. Our design for ACME Toys and Gizmos is very rudimentary, just to give us an introduction to designing in OWB. You'll want to read in more detail about design when you tackle a real-world design because you may run into other issues we didn't have time or space to cover here.

We're going to actually implement the design in OWB in the next chapter.
Where to buy this book

You can buy Oracle Warehouse Builder 11gR2: Getting Started from the Packt Publishing website: http://www.packtpub.com/oracle-warehouse-builder-11g-r2/book

Free shipping to the US, UK, Europe and selected Asian countries. For more information, please read our shipping policy.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.