Defining and Importing Source Data Structures

The Warehouse Builder software and Oracle database have been installed, and we're ready to begin building our data warehouse. The first thing we have to do is define what our sources of data will be. If we are going to build a useful data warehouse, we have to know what kinds of information our users are going to need out of the warehouse. To know that, we have to know the following:

• The format in which the data is currently stored and where it is stored.

• Whether there is a transactional database currently in use or not, which supports day-to-day operations and from which we'll be pulling the data.

A transactional database is different from a data warehouse database in that it is designed to support the day-to-day transactions that keep an organization running.

• Whether the database is an Oracle database or another vendor's database such as Microsoft SQL Server.

• Whether there are any flat files of information saved from database tables or other files that users keep, which might be a source of information.

A flat file is a file in text format that stores data in some kind of delimited format. The most common example of this kind of file is a CSV file, or a comma-separated file, that can be saved from a spreadsheet or extracted from a database table. It is called a flat file because it is in a text-only format and doesn't need to be interpreted by another program or application to read it.

The Warehouse Builder can help us with importing data from any (or all) of these formats into our data warehouse, and we're going to see how to do that in this topic by covering the following major topics:

• Analysis of the source systems for the data warehouse we'll be building

• The Point of Sale transactional database

• A website orders database

• An overview of the Warehouse Builder Design Center

• Importing and defining source database object metadata

• Creating projects and modules in OWB, including Oracle and SQL Server modules

• Creating a SQL Server database connection using an ODBC gateway

• Configuring Oracle Heterogeneous Services for the ODBC gateway

• Defining source metadata manually with table editor

• Importing source metadata from a file

Preliminary analysis

In any data warehouse project, we are going to need to do some up-front analysis to determine what data will need to be captured into our warehouse. The analysis will tell us where the data is located, and in what format, so that we can begin to define our source data structures in the Warehouse Builder. In our case, we will presume that we have interviewed the management at the ACME Toys and Gizmos company and they have indicated the following:
• The high-priority information that they would like to see from this data warehouse project is sales-related data for all their stores

• They don't have an idea about the comparative sales in the various stores, so they need some way to view all that data together to do an analysis that shows how well, or poorly, the stores are doing

• In the future, they would also like to be able to compare store sales with their website sales, but that will not be required for this first data warehouse we build

We are doing a very simple analysis of our data warehouse project because the focus of the topic is primarily on OWB. This topic is all about using the Warehouse Builder and that begins after the initial analysis, and so we will cover just enough information to lay the groundwork for what follows. For more coverage of the design and analysis phase from a very practical standpoint, a very good topic you should look up is "The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling",

Ralph Kimball and Margy Ross, John Wiley and Sons, Inc. This topic covers in detail the analysis and design considerations you should take into account when designing a data warehouse and uses practical examples from a number of industries.

ACME Toys and Gizmos source data

Talking to users, administrators, and database administrators in ACME has helped us discover that there is a transactional system in use (called a Point-of-Sale or POS system). This system supports the stores that ACME has located in various cities throughout the country, and in other countries in Europe and Asia.

This system maintains data in a Microsoft SQL Server database named ACME_POS, and tracks individual sales transactions that occur for all of ACME's toys and gizmos. This database contains tables that store information about each sale along with all associated sales information such as the item sold, its price, the store in which it is sold, the register that processes the sale, and the employee who made the sale. Right away, we recognize that this would be a good source of data to help satisfy the management objective of analyzing their sales data better.

We have also found that the IT department that runs the website for ACME Toys and Gizmos has its own database that supports the website order management process. It is implemented in Oracle and handles the processing of all orders taken for products through the website. It contains tables that store information about:

• The orders taken

• Information about the customer who placed the order

• Information about the individual products that were ordered

This is an example of the source data that might not be needed in the initial stages of a data warehouse project, but that could be requested by users at a later stage. Then we can expand the data warehouse implementation to include website sales data. We will also take a look at importing source metadata for it later in this topic.

Scripts have been provided on the Packt website at **http://www.packtpub.com/files/code/3449_Code.zip**. These scripts can be used to build the Oracle website orders database referred to in this topic. We also have a CSV file in this code bundle that will be required to import metadata. We have

a script to install the SQL Server database. Instructions to use these files can be found with the code bundle.
However, working on it beyond that is not in the scope of this topic.

The POS transactional source database

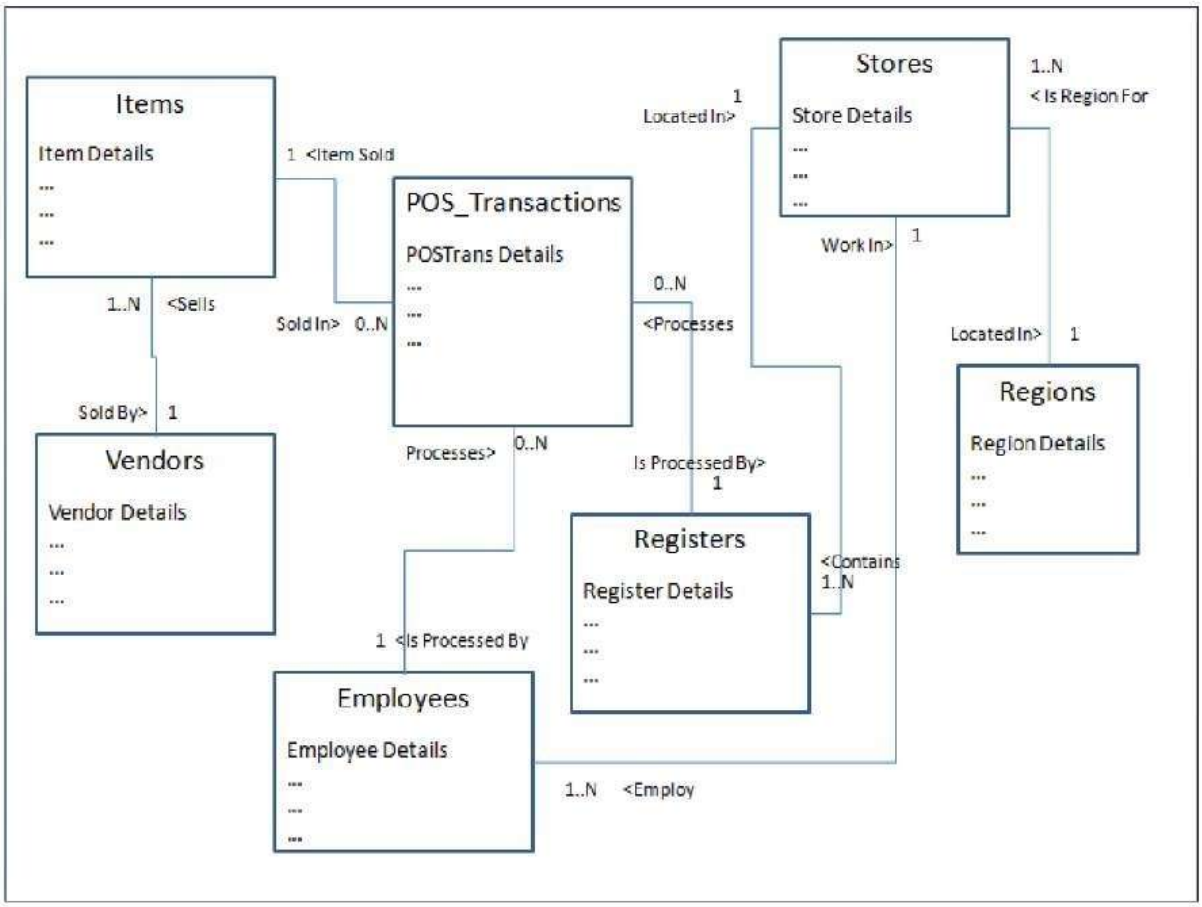
The DBA (Database Administrator—the person responsible for the maintenance and administration of the database) is in charge of the POS transactional database. The DBA has provided an Entity-Relationship (ER) diagram of the database to help us understand the database and the relationships between the various tables. The diagram is in the UML (Universal Modeling Language) notation. The following image depicts a simplified version of the diagram containing the main tables of interest and the relationships between them, including the cardinalities. The cardinality indicates how the records in one table relate to records in the other. The cardinality can be expressed as many-to-many, one-to-many, many-to-one, or one-to-one, and is indicated in the diagram with counts composed of the following:

- 0..N—zero or more

- 1..N—one or more

- 1 — one only

The details about the columns in each table will be covered when we define the metadata for them. If you are familiar with ER diagrams, the process of implementing a database based on the diagram, and the concept of normalization, you can skip the following section and move on to the The website order management database section.



The main table in the ACME_POS database is the POS_Transactions table. It holds information about each transaction that takes place in a store, including the cash register that processed the transaction, the employee who worked the register, the item sold, the quantity sold, and the date. However, not all of that information is stored directly in the POS_Transactions table; only the date and quantity are stored directly. If all the details about the item were included in every record in the POS_ Transactions table, there would be a large amount of duplicated information. After all, the store is not going to sell an item only one time because if it did, it wouldn't be in business for long. There will be potentially hundreds to thousands of sales of the same item each day, depending on how busy a particular store is. With each of those sales, a row gets placed in the POS Transactions table.

We can see from the diagram that a separate table was created to hold item information and a link made from the main POS_Transactions table to the Items table. That link is created via a foreign key stored in the POS Transactions table for the Items table. Instead of storing all the information about the item in the POS_Transactions table, a single column called the foreign key is placed there. This foreign key has a value corresponding to a value in the primary key column of the Items table. A primary key is a value that uniquely identifies a row in the table and, therefore, is not duplicated. We

can then look up in the Items table for the information about the item for sale by using the value in the foreign key column for the item.
This concept of storing an item's information in a separate table results in a much greater accuracy of data, as we don't have to duplicate the item information. It is only entered once in the Item table. If it has to be updated, there is only one record in the Item table to update, and not thousands of records in the POS_Transactions table. This is known as database normalization. A transactional database is usually normalized due to this need for data accuracy.
The attributes of the POS_Transactions table are the individual pieces of information stored in it. Each of the attributes corresponds to one of the lines originating from the POS_Transactions table in the diagram, all except the quantity attribute. We can see that information about the employee who worked the register for the sale is stored in a separate table, the Employees table. This is similar to how the items sold are handled and stored in the Items table, as well as how the information about the register on which the sale was processed is stored in the Registers table.

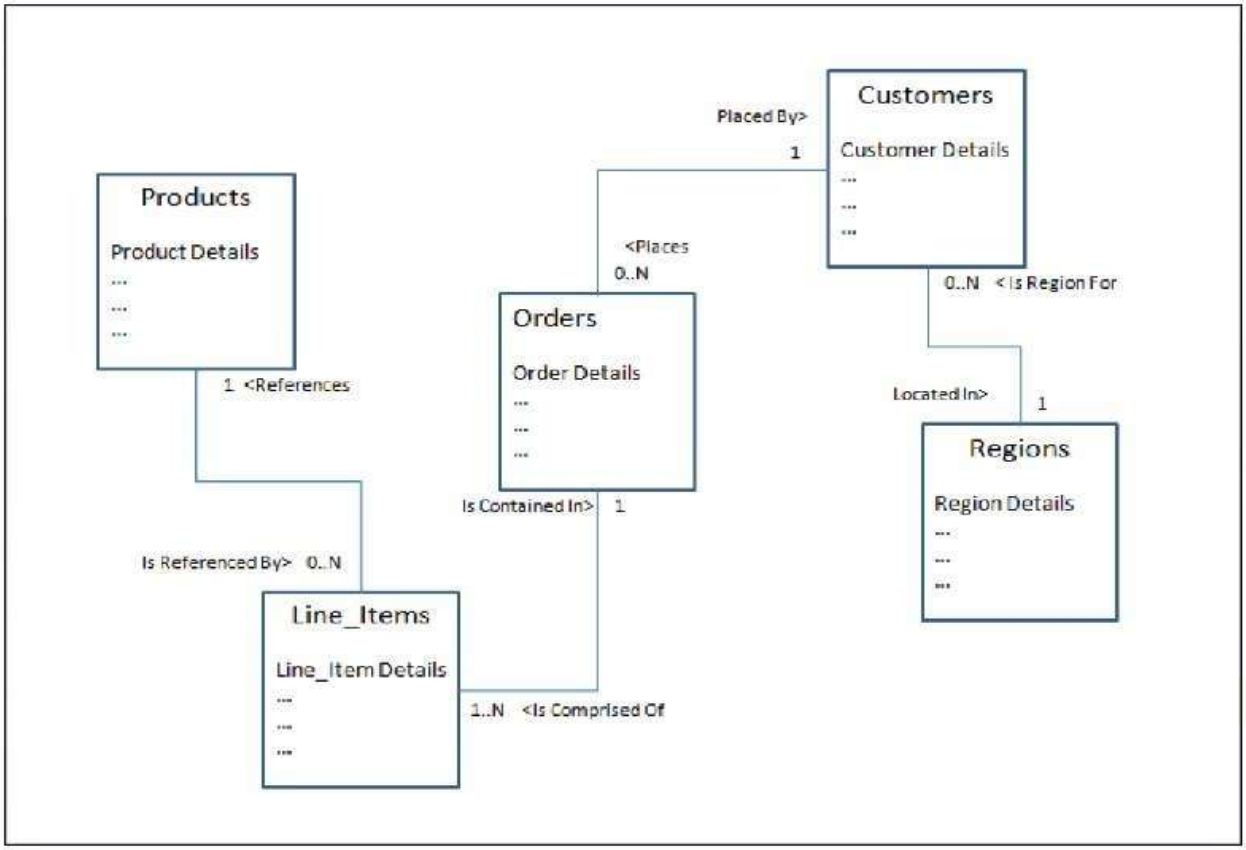
In addition to these tables, we can also notice that a few other tables in the diagram are linked in various ways to these tables. They provide us with even more information about the attributes of a transaction and, therefore, about the transaction itself. We can see a table hanging off the Items table called Vendors. This table stores the information about each vendor who supplies toys and gizmos to the ACME company. A table called Stores is linked to the Registers table. This table tells us information about the store in which the register is located and, therefore, about the store that made the sale. Linked to the Stores table is the Regions table, which provides a location breakdown by region for the stores. ACME Toys and Gizmos is a worldwide operation and likes to track sales by breaking the world up into regions such as Europe and Asia, and for the US it's Northeast, Southwest, and so on.
At this point we can begin to understand why the management found it so difficult to compare sales data from all their stores and website, and why they would like to implement a data warehouse for their data explorations. Let's look at what kind of SQL query would be required to determine the number of flying discs sold today in Europe that were supplied by a particular vendor. Let's just consider the number of tables involved. We need the POS_Transactions, Items, Vendors, Registers, Stores, and Regions tables. The only table we don't need is the Employees table. But, why do we need the Registers and Stores tables when they wanted to know only the amount sold in the region? Well, the answer is that it's very tough for management to get the data they need. You see there is no direct connection from the POS_Transactions table to the Regions table.
The only way to get the region for a given transaction is to look up the register that processed the transaction, and then look up the store in which the register is located and that store record will give you the region. All of this is done with one massive join SQL query to join all these tables together. A join query is one that pulls data from more than one table at a time. As the database has a normalized structure, we have to include those two additional tables in our join, which we really don't want, just to get to the information we want. If we're talking about millions of transactions,

which is not at all an unreasonable situation for any large sales operation, we would end up with highly inefficient queries that take a long time to run and make management very unhappy with the database.

So, we're going to solve this problem with our data warehouse, which will have a much better organization of tables for querying as we'll see in the next topic.

The website order management database

The DBA in charge of the Oracle database for the website order management system has provided us with its ER diagram for our information. As with the POS transaction database, an ER diagram is provided here in a shortened version to give us an idea of the tables involved and their relationships with each other. Later in this topic, we will have examples dealing with the POS transactional database as it contains the sales data for the stores. This database is presented here because of the possible future requirements to include this data. We'll use it to provide an example of importing from a database, and as an example of some minor issues that can be encountered when trying to analyze multiple sources of data.



This database holds the sales information for the website, and we have to understand it before importing the database. We can see that it has some tables that are similar to the tables in the

previous ER diagram that we just saw. The Orders table is the main table in this database instead of the POS_Transactions table. It holds information about customers who placed an order and a list of ordered items. The customer information includes the region in which the customer is located, and this is identical to the information in the Regions table in the POS Transactions database. The customer information is stored in a Customers table, which is linked to the Orders table, and we can see that the Regions table is linked to the Orders table through the Customers table. The list of ordered items is stored in the Line_Items table, which also has product information that identifies which product was ordered. The product information is stored in the Products table (which is similar to the Items table in the POS database). We can see that it has a link to the Line_Items table in our diagram.

Now we may get confused because this database has a Line_Items table and the other database has an Items table. But we're told that the Products table actually corresponds to the Items table, and not the Line_Items table. While this company is entirely fictional, this kind of issue of multiple departments, each with their own database and convention for naming tables and columns, is all too common in the real world for data warehouse projects. It's up to us to make sense out of it all and pull all that data into a single data warehouse where it can be queried at once. And this is what makes our job so interesting.

Let's look at one more issue with this order management database before we move on. This issue is the relationship between the Orders table and the Line_Items table. Each order is composed of a variable number of line items of ordered products. One person may place an order on the website for a doll and a fire truck, whereas another may order a game, a deck of cards, and a baseball bat. We've seen that this relationship between tables is accomplished in the database by storing a foreign key to the other table to indicate the relationship, but there could be any number of line items in an order. This would mean you will need any number of line item foreign keys stored in the Orders table, but that is not possible. The reason is that the foreign key in this situation is going the other way. The Line_Items table stores a foreign key to the order of which it is a part of, as a line item can be associated with only one order.

This was a brief overview of the source data structures we're going to be working with and also a very brief introduction to some database design issues. Without further ado, let's turn our attention to the Warehouse Builder, which is the real subject of this topic.