3

OF A THREE-PART SERIES

Beyond the Data Model: Designing the Data Warehouse

By Josh Jones and Eric Johnson

CA ERwin

TABLE OF CONTENTS

INTRODUCTION 3
DATA WAREHOUSE DESIGN 3
MODELING A DATA WAREHOUSE 3
- Data Warehouse Elements 4
- Star Schema 4
- Snowflake Schema 4
- Building the Model 4

EXTRACT, TRANSFORM, AND LOAD 7
- Extract 7
- Transform 7
- Load 7
- Metadata 8

SUMMARY 8

Without a doubt one of the most important aspects data storage and manipulation is the use of data for critical decision making. While companies have been searching their stored data for decades, it's only really in the last few years that advanced data mining and data warehousing techniques have become a focus for large businesses. Data warehousing is particularly valuable for large enterprises that have amassed a significant amount of historical data such as sales figures, orders, production output, etc. Now more than ever, it is critical to be able to build scalable, accurate data warehouse solutions that can help a business move forward successfully.

Database modeling goes beyond on-line transactional processing (OLTP) models for traditional relational databases and extends in the world of data warehousing. In addition to requiring that a schema be designed, data warehouses have several other components that are equally important. When working with data warehouses you also have to take into account metadata, and how the data is to be placed into the warehouse from one or more sources. This whitepaper will cover data warehouse design, working with metadata, building ETL processes, and offer some guideline for the overall design of your data warehouse solution.

DATA WAREHOUSE DESIGN

Put simply, a data warehouse is a repository of data. Typically, this is historical data about the business that has been conducted by a company. This may sound like a database, and at the most basic definition, it is. However, the generally accepted difference is that the term database actually refers to a relational database, and often more specifically an online transactional processing (OLTP) relational database. OLTP databases are designed to perform well under transactional loads, often supporting applications that are highly focused in nature, such as purchasing interfaces. Conversely, a data warehouse is for online analysis processing (OLAP), and built to meet the needs of users who are responsible for analyzing the data and making decisions based on it.

It is worth noting that there is not a strictly defined, unified standard to building data warehouses. There are at least two very well known methodologies behind designing a data warehouse; the "dimensional" model and the "normalized" model. Each of these methods works and each has its pros and cons. While we will be using the dimensional approach, it is worth noting the basics of each.

The normalized model is based on the normalization models defined by E.F. Codd. Specifically, the idea is to store the data in third normal form (3NF), grouping the data by topic, i.e. customers, orders, products. This allows the data warehouse to be updated with new types of data very easily, because you can add new topics without affecting the existing data. However, this method can be cumbersome for non-technical users to perform ad-hoc queries against, as they must have an understanding of how the data is related. Additionally, reporting style queries may not perform well because of the number of tables involved in each query.

In a nutshell, the dimensional model describes a data warehouse that has been built from the bottom up, gathering transactional data into collections of "facts" and "dimensions". The facts are generally, the numeric data (think dollars, inventory counts, etc.), and the dimensions are the bits of information that put the numbers, or facts, into context (think sales person names, regions, etc.). This method allows for very efficient queries from the standpoint of non-technical users, because the data is grouped in a logical way, and it performs quite well since the similar types of data are stored together. However, maintaining a dimensional data warehouse requires some effort, because adding new types of data may require that the entire warehouse be updated, or reprocessed, in order to incorporate the new types of data.

Finally, it is important to understand where the data for a data warehouse comes from. Obviously, a large portion of it (in some cases, all of it) will come from the operational databases being used by the enterprise. Just remember, in large enterprises, this may mean data could be gathered from databases on mainframes as well as distributed systems. However, there may be data stored in non-relational sources, such as file shares and even email servers. Because of this, the process of designing a data warehouse also includes the task of designing the processes that bring data into the data warehouse. The process of getting that data is often referred to as Extract, Transform, and Load (ETL) reflecting the three major operations that process raw data into the data warehouse.

MODELING A DATA WAREHOUSE

Just as in relational data modeling, building a data model for a data warehouse requires a significant amount of work be done prior to the start of the project. This is because a data warehouse is a very structured, specific deliverable. You must account for what the initial data will be, and what data will be incorporated in the future. It is necessary to approach the design of a data warehouse much like an application design project; in fact, often times a data warehouse will be coupled with a front end reporting system, whether it is a web based application or a desktop client. As such, you'll follow standard design practices, such as gathering requirements, building documentation and diagrams, and generating a significant amount of data about the data, or metadata, that will help define the structure and scope of the data warehouse. These activities are outside the scope of this paper, because preparing for a data warehouse requires these tasks be com-

pleted prior to the beginning of the technical process described here.

DATA WAREHOUSE ELEMENTS

The basic pieces of a dimensional data warehouse model are tables and relationships, laid out in a schema. Specifically, we'll have fact tables and dimensional tables, and they will be related to one another in a manner that facilitates understanding of the relationships between those tables. This layout is called a schema, and there are two primary schema types used in a dimensional data warehouse: snowflake and star.

STAR SCHEMA

One of simplest schemas, star schemas are named such because they usually involve a single fact table, related to a single layer of dimension tables. Each dimension table has a simple primary key, as it is responsible for a single type of data. The fact table has an aggregate primary key, as it is the compounded collection of the primary keys from the dimension tables. Figure 1 shows a very simple star schema outlining sales data.

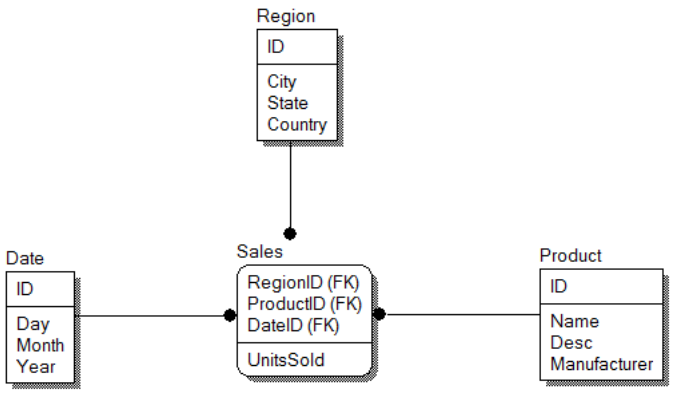


Figure 1. *Simple star schema.*

The simplicity of a star schema is that for each fact table, you will only ever have to join one level of dimension tables to get at the data. Unfortunately, the drawback of a star schema is that for intricate systems, it can become quite complex to read and query when it is necessary to retrieve a large amount of fact data.

SNOWFLAKE SCHEMA

The snowflake schema is very similar to the star schema, with the exception of the relationships between dimensions and facts. A snowflake schema still has fact tables at its center, with relationships to dimension tables. However, if there are dimensions with large numbers of attributes, it might be necessary to break the dimensions down into subdimension tables. The end result is a schema the visually looks like a snowflake, because of the layers of dimension tables. Figure 2 shows a slightly expanded view of the sales data in a data warehouse, employing a snowflake schema.

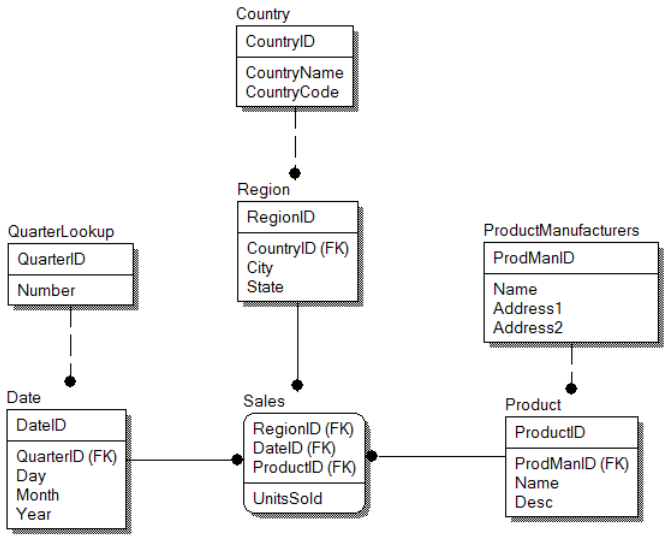


Figure 2. *Snowflake schema.*

While this is not an exhaustive look at a data model, it shows the basics of a snowflake schema. There is a central fact tables, Sales, and two layers of dimension tables, effectively normalizing the data. The advantage here is that logically, we can see how the data is related very easily. We can drill down from the sales, to the region, and lookup the country code. In a large system, a snowflake can also facilitate data mining queries that require complex comparisons across different dimensions. For example, if we need to see all sales from the 3rd quarter of 2007 for a specific country, filtered by product manufacturer, we're literally joining together seven tables. If this data had been stored in a star, we'd be joining four tables. However, from a performance standpoint, this schema may slow down larger queries that seek to simply retrieve data from a few dimensions. Just remember, because of the differences in performance and usability, it is vital to structure your schema according to your specific needs.

BUILDING THE MODEL

Let's now assume that we have a specific project, and have gathered the necessary requirements. The project will be fairly simple; we're building a data warehouse for a DVD retailer that wants to be able to figure out what products are being ordered and what their customer demographics are. This is a readily recognizable need; almost every business

is either selling a product or a service, and while the details vary, the basics stay the same. We'll need a model that diagrams our schema, with all of our facts and dimensions. We'll also need to discuss management of the data warehouse, and how we'll do the ETL for the warehouse.

Before we begin, remember that this is the modeling phase. So we will be using standard modeling terminology, i.e. entities, attributes, etc. Even though building a data model for a data warehouse is a much more straightforward process than a standard relational database (since there is almost always a one-to one relationship between entities and tables), we should still stick to the modeling lexicon until we actually begin building the physical database.

First, we know that our primary source is the operational database that supports the ordering system. And since there was a data model created for that database, we can actually use the data model as a source for our data warehouse, at least in the design phase. Note that if you are not lucky enough to be in this situation, it's possible to simply diagram the existing database as a jumping off point. Second, we'll be building this data warehouse inside of Microsoft SQL Server, using both the relational database engine as the data storage area for the data warehouse. First we'll construct the data model, and then build the necessary tables in a standard database to support that model.

Let's take a look at the source database's very simple data model, shown in Figure 3.

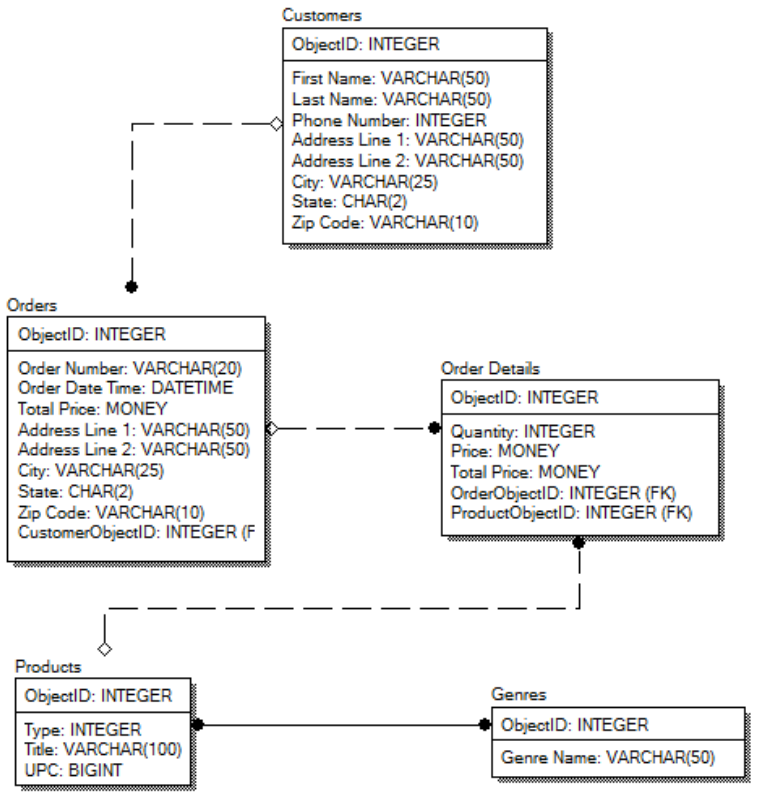


Figure 3. *Relational data model source.*

You can see that it is a standard, normalized data model that contains order, customer, and product data. What we need to do is convert this into a star schema, with fact tables around each type of data we are working with. In order to do that, we have to understand the data that is being stored, from a logical standpoint.

Looking at the model, the first fact we can see is that there is an order taking place, and we already know that these are DVDs (our products). So we'll have a fact entity of FactOrders, with dimensions describing when the order was made, how much it was for, and what the ordered items where. Notice that in the relational model, orders have order details, and the order details contain the relationship to the actual product, since a single order can have more than one product. Additionally, orders are placed by customers, but we have a fairly simple entity storing that data. But note that in the analysis of this data, we generally will be describing the data either from the standpoint of a specific product ("How many copies of title X did we sell last month?") or from the customer standpoint ("What types of titles are popular in the US versus outside of the US?"). This means that we won't care as much about the distant relationship between the Orders entity and the Products entity. However, all of this is related to a single fact; orders are placed. Everything else is a matter of detailing that order.

Now we can begin adding objects to our data model. Figure 4 shows the fact entity, FactOrders. Remember that in a star schema, the fact entity's primary key consists of the foreign keys from the dimension entities. So at the early stage, we are only concerned with storing attributes specific to orders that are not dimensions that will qualify the orders. And since this is a flat look at every item ordered, we'll actually collapse data from the Order and Order Details entities contained in the relational database.

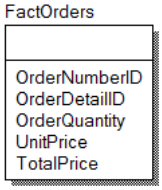


Figure 4. *The FactOrders entity, with no dimensions associated.*

This gives us the basis for the facts, which is each order. Note the "OrderNumberID" and "OrderDetailID" attributes are actually the primary keys from the Orders and OrderDetails entities. This is done to correlate every line item to its corresponding order. We then have the detail information we need; how many items were ordered, what the unit price is, and what the total for that line is. Now we can see the need for context around these numbers. When did the orders take place? Who ordered these items? What are the details around each item? The answers to these questions help us determine the dimensions.

We are now ready to add the products into the model. Since the products are part of the description of what an order is, they are a dimension of the orders. Figure 5 shows the addition of the DimProducts entity, and the ensuing relationship between the dimension and fact entities.

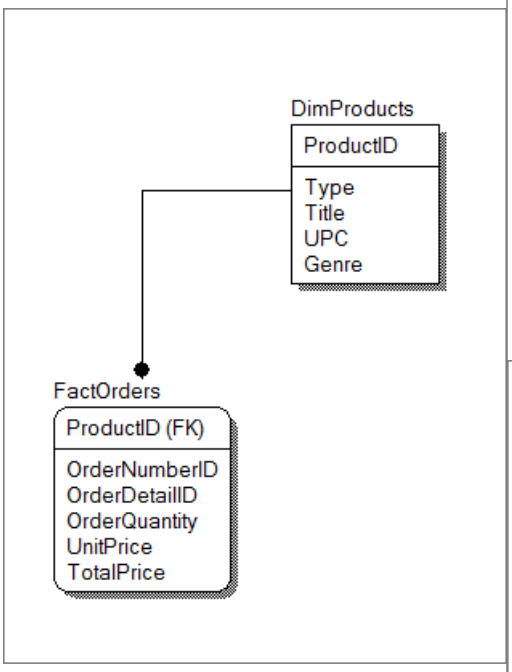


Figure 5. *The FactOrders entity and the DimProducts entity.*

Now let's go ahead and add the customer entity. Fortunately, in this situation, the customer entity is fairly simple, because the relational storage of the customer information is in a single entity. Figure 6 shows us the data model for our very simple data warehouse.

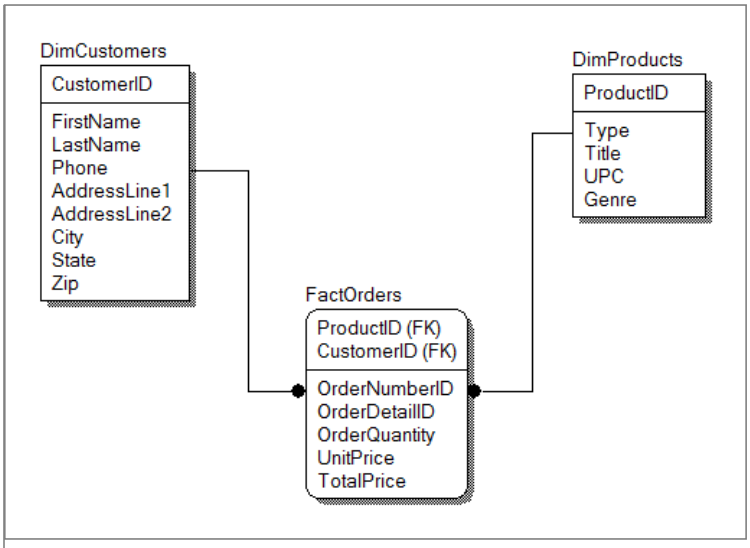


Figure 6. *A very simple star schema for the orders information.*

In this view, we can now see how a data warehouse flattens, or "denormalizes" data for querying. This model would not work well at all for a transactional system, but will help those queries that are trying to tie large numbers of records together, usually aggregating certain data points along the way. Again, this is a very simple look at how to construct a data warehouse schema based on a logical model or physical database. When you introduce this method into a project where there is significantly more data begin stored in the source database(s), the resulting will be more complex. Just remember that at the very low level, you'll have either a collection of star schemas, or a snowflake schema (which is a slightly normalized collection of star schemas). Figure 7 shows a slightly more expanded view of our little data warehouse, with some of the data pulled out into other dimensions for lookups.

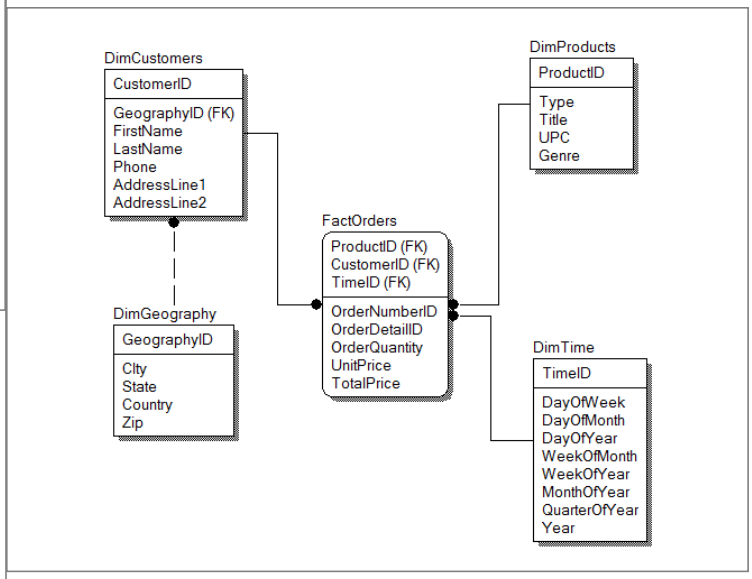


Figure 7. *A more fleshed out data warehouse star schema*

This schema allows us a little more detail. For example, we've added the time dimension, DimTime, which allows us to more granularly analyze when orders were placed. We've also added the DimGeography entity, which separates and stores more information about the location of our customers. Technically, we're now looking at a form of snowflake schema, since we're more than one level away from the fact entity. However, this allows us to reuse certain dimensions (both DimTime and DimGeography) as needed by other fact entities. If this schema is actually part of a larger system that includes data about vendors, shipping information, employees, etc., then we have reusable dimensions that may help standardize the data and keep it consistent. However, this will be at the cost of performance, and can start to cause confusion when visually analyzing this data. Remember, when building a data warehouse, keep in mind that non-technical users will likely have an ad-hoc tool allowing them a certain degree of freedom in exploring the data. Try to keep it from being too confusing, while still keeping the best structure possible.

Once the model has been built, provided you are using a tool such as CA's ERwin Data Modeler, you can simply generate the physical model based on this logical model. Then, you can proceed to create the schema necessary in your physical storage medium (such as Microsoft SQL Server), and begin loading data.

EXTRACT, TRANSFORM, AND LOAD

We have mentioned several times that data in a data warehouse comes from some other system or systems. How do you go about extracting the data from the other systems, transforming it to meet the data warehouse specifications, and ultimately load it into the data warehouse? This process has been named based in three steps: Extract, Transform, and Load (ETL). Any data warehouse needs an appropriate ETL solution in place to facilitate the harvesting and loading of data from all the disparate sources. Furthermore, as we have mentioned, the schema of a data warehouse is very different than that of an OLTP system, so you have to take steps to change the data as you bring it in. Whether your ETL tool is a custom application, or you use a third party ETL tool, it is an absolute requirement that one exist.

There are literally dozens of tools already in the marketplace to perform ETL functions; everything from database vendor specific integration tools like Microsoft's SQL Server Integration Services to stand alone ETL tools such as those from Informatica. There are several key areas that need to be addressed by an ETL tool. Specifically, functionality in terms of transforming the data, data source connectivity, and meta-data interoperability are all key to a good ETL tool. And remember, it's not necessarily a requirement that a tool be purchased; you may need to develop your own ETL process. Keep in mind that either way, you need to make sure that your data warehouse is loaded with cleansed, accurate data in a timely fashion. Let's take a closer look at each phase of the process.

EXTRACT

As we've already discussed, there are likely to be a diverse set of data sources for a data warehouse. While many times there are relational databases that house the bulk of an enterprise's information, there may also be information stored in flat files (such as XML files), or in legacy systems such as VSAM databases. Whatever the source, they key component to extraction is the parsing of data into a common format. This will help during the next phase, the Transform phase, because it'll be possible to iterate quickly through the data to check for problems, and execute cleansing processes.

TRANSFORM

During the Transform phase, rules and calculations can be performed on the extracted data to prepare it for insertion into the data warehouse. While transforming data from a relational database is usually (though not always) pretty straightforward, cleaning up data from other data sources can be fairly difficult because of storage differences (think datatypes). Additionally, there may be differing sources for data that must be combined during the load phase. For example, the source operational database may store the fundamental data for a billing system. However, the invoice details that were actually sent to clients may be stored in XML files for distribution to specific client systems. The data warehouse may need to contain both sets of information; the Transform phase will be responsible for combining the relevant data into a loading-friendly format, including comparing dates and customer numbers to ensure that the correct invoice details have been correlated to the billable information from the operational database. Some of the common tasks that occur during the transform phase are:

- Removing duplicate data
- Validating data, in terms of format (or any other business rules)
- Generating surrogate key values, i.e. for dimension tables
- Deriving calculated values
- Cleansing data to decode coded values for human readability (i.e. 1 = Male, 2 = Female, etc.)

LOAD

Once the data has been cleansed and combined into common formats and structures, it is ready to be loaded into the data warehouse. The key to this phase is timing. How often

is data loaded? When is the best time to load the warehouse? Often times, the load phase is very process intensive, so it may be necessary to schedule the load to happen when there is low usage of both the source and destination systems in order to avoid user or customer impact. Additionally, when using advanced analytical software to process data in the data warehouse, often times it is necessary to execute a "re-process" on that system once the data warehouse has been reloaded. In any case, this process must be carefully designed with performance and user impact in mind.

Finally, the key to any ETL process is its scalability. The process must be flexible enough to add new sources, and modify transformations as business rules and needs change over time. Be sure to account for growth in both volume and functionality when designing an ETL process for your data warehouse.

METADATA

Metadata is the data about the data. Or, in the case of a data warehouse, the data about the data, where it came from, how it got there, and what it means. Think in terms of a library; the books in the library hold countless amounts of data, but how do you find the right book? You guessed it; metadata. The information about the book such as title, author, publication date, shelf location, and a short description of the book would make up the books metadata. Without this metadata, you would not be able to find a specific book let alone choose from among thousands of book the one that has the information you need. In a data warehouse system it's no different. You need to generate metadata about every piece of data being stored in the warehouse. You need to know where the data comes from, what is used for, how it may be summarized, and where the data is inside the data warehouse.

Ideally, the metadata for a data warehouse contains all of the information about both the ETL and the actual data at rest. This means that you'll have data about how the data gets into the warehouse, as well as what the data in the warehouse is. Additionally, the metadata will hierarchical, so as to provide logical ordering of the metadata, much like a book index, so that it can be read and understood quickly. Regardless of storage format however, there are three basic types of metadata for a data warehouse[1]:

1. **RDBMS Metadata**—This is metadata about the relational database housing the data warehouse. This can be data about the table structure as well as the data in the tables.

2. **Source Systems Metadata**—This is the data about where all of the content data comes from. Schemas, update frequency, and process information can all be stored with this type of data.

3. **Staging Metadata**—This is the ETL data. Definitions of transformations and destination objects, logs, etc.

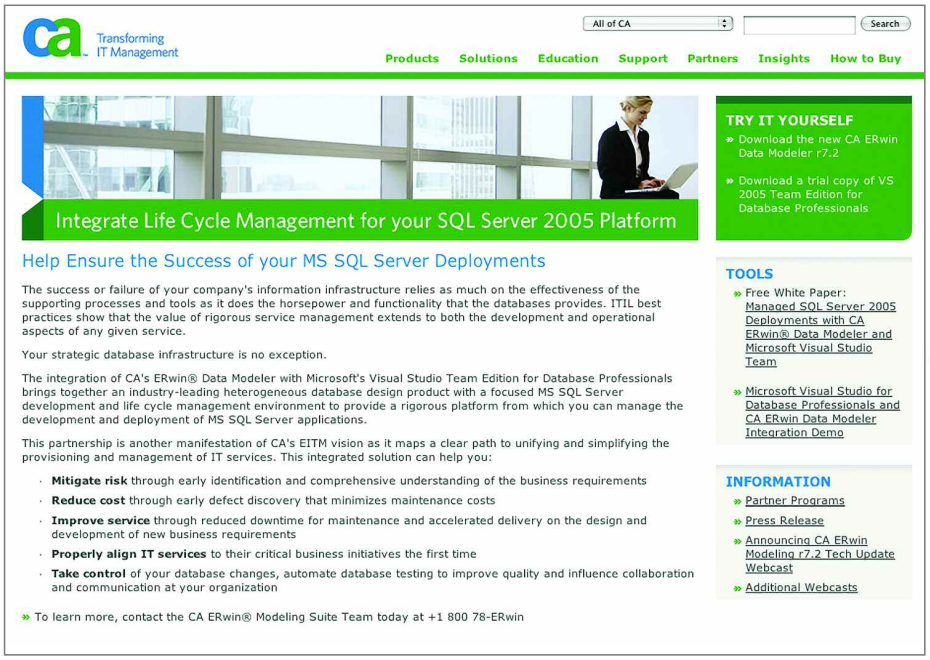
While the usage and definition of metadata in the data warehousing world is very fluid, these few items will make sure that you have handy reference material for both the process of loading your data warehouse as well as what's in the warehouse itself. The latter can be very valuable to business users that must explore the data in the warehouse, so be sure to update it frequently.

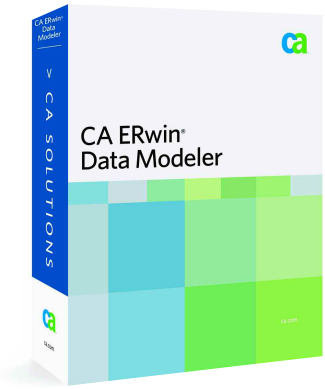
.

SUMMARY

A data warehouse is valuable tool for businesses to be able to make smart decisions for their future. Building a data warehouse, while similar to building a database, requires unique thought and preparation. Be sure to thoroughly analyze the existing data to look for the proper facts and dimensions, and decide early on what schema will be used. Once the model has been built, designing the ETL process for the data warehouse will require the consideration of third party tools, as it usually very complex and difficult to build from scratch. Make sure to document everything, and build metadata around your data warehouse for yourself as well as your users. CA's ERwin Data Modeler can help create very detailed metadata for your environment, as well as manage the entire data warehouse modeling process from beginning to end. And when Erwin is used to create both the relational database and data warehouse models, you will have unified model management process that will enable the expansion of your data models into the future easy.

[1] Ralph Kimball, The Data Warehouse Lifecycle Toolkit, Wiley, 1998, ISBN 0-471-25547-5





Visit: ca.com/erwin/msdn. *CA ERwin Data Modeler provides a way to create a logical structure of your data and map that structure to a physical data store.*

ca.com/erwin/msdn