Data Modeling for Business Intelligence with Microsoft SQL Server: Modeling a Data Warehouse

By Josh Jones and Eric Johnson

CA ERwin

TABLE OF CONTENTS

INTRODUCTION

For decades, businesses of all shapes and sizes have been storing their historical data. Everything from cash register rolls stored in filing cabinets to web application transactions being stored in relational databases have been sitting quietly, often collecting dust, just waiting for someone to dispose of them. Fortunately, as we've transitioned from paper based records to computer based records, we've discovered that mining this historical data can yield a veritable gold mine of information that can help guide a business towards future success. Understanding everything from geographical product preferences to age based sales demographics to employee performance can help companies make smart decisions about what they sell, and how they sell it. Today, we can use modern technology such as Microsoft SQL Server to run both the transactional databases that support our mission critical applications as well as the analytical databases that can help drive our businesses forward.

In this series, "Data Modeling for Business Intelligence with Microsoft SQL Server", we'll look at how to use traditional data modeling techniques to build a data model for a data warehouse, as well as how to implement a data warehouses and their accompanying processing loads. We'll also look at CA's ERWin Data Modeler to expedite and improve upon traditional design methodology.

WHAT IS BUSINESS INTELLIGENCE?

Business Intelligence (BI), the term that represents gathering, analyzing, and presenting business data, has evolved to refer to the specific set of technologies and business processes that help companies get the most out of their data. A BI solution generally includes: the technologies that hold the source data; the processes to extract that data; calculations and manipulations to reveal trends within that data; and the reports and interactive tools to display the data to users throughout the company. Additionally, it's very important for the company/organization to figure out what data they are interested in seeing, otherwise there is an almost endless number of ways to combine, calculate, split, and trend any type of data.

Once a company has decided to pursue a comprehensive BI solution, there are a number of components that must be discovered and/or developed. Specifically, these are:

- **Data Sources**
  Where is all of the data stored? What types of systems are being used? How much data do we have available?
- **Data Destinations**
  How is the data going to be presented? What are we using to store the new data?
- **Tools**
  Are we building internal tools or purchasing something? What are we using to perform calculations and other trending processes?
- **Business Rules**
  What are our thresholds? What data is critical versus noncritical? How far back to we want to look at historical data? Who needs to access the output of the BI solution?

DATA MARTS VERSUS DATA WAREHOUSES

One of the primary decisions that must be made when planning a BI solution relates to how the historical data will be gathered and stored to be used for analysis. Not only what hardware must be used, but logically, what type of data structure will be housing the data. The two primary structures used for this are the *data mart* and the *data warehouse*. In short, these two components are basically the same in terms of structure, but differ in their usage. They are both databases, typically stored in a Relational Database Management System (RDBMS), that are custom built to hold analytical data. However, they are not relational databases, because they typically don't follow basic design rules such as normalization and referential integrity (though these concepts are by no means disposed of; they are just used in a different manner).

The difference between a data mart and a data warehouse is generally a question of scope. Data marts tend to store data related to a small subset of the business, such as a department or a specific product. Conversely, a data warehouse tends to support the entire business, or large sections of the business, depending on the size of the company. So, you could imagine that data marts store the lower level version of the data, while data warehouses will aggregate (or be fed by) the data from the data marts. In some environments, this process is reversed (data marts are built after the data warehouse has been loaded with data), but the concept is still the same. For the purposes of this whitepaper, we'll refer to these two items nearly interchangeably; just remember that the same information applies to both unless otherwise noted.

Within both the data mart and data warehouse is the "cleansed" form of the historical data. Typically, historical data is transactional data, such as sales transactions, order details, billable items (such as minutes) and usage details (i.e. number of phone calls per hour in a telephone system). For the purposes of analysis, this could be a formidable amount of data that would be difficult for end users, such as business analysts, to comb through and use to identify trends. Data warehouses are used to hold the calculated version of that data. Upon loading, the data will have been cleansed, meaning that duplicate records will be removed, unnecessary fields of data will be eliminated, and certain values, such as totals and averages, will be pre-calculated and stored along with the original data. This allows for faster, smarter queries against the data to enable end users to get their information faster.

CUBES

Once data has been loaded into a warehouse, there's another, far more analysis oriented structure that can be used; cubes. Analysis cubes, or data cubes as they are often called, are three (or more) dimensional arrays of values stored along specific dimensions (such as time). Cubes allow for much faster access into historical data, because of their multi-dimensional format. In Microsoft SQL Server Analysis Services, cubes are loaded from a given raw data source, and additional calculations and processing can be programmed to add new values and reveal trends in the data. This allows for the "precompilation" of data, which makes retrieval by end users even faster than a typical data warehouse stored in an RDBMS.

DATA MODELING A WAREHOUSE

When it comes to designing a data warehouse, there are quite a few traditional data modeling processes that are useful. When you design a data model, you will typically gather requirements, identify entities and attributes based on the data, normalize the model, determine relationships,

and document the model using a modeling tool such as ERWin. When you build a data warehouse, it's much the same. The difference comes in how you identify the data, and how you build entities and attributes. With a data warehouse, the data you are working with will be slightly different, because it will be a "rolled up" version of the data. You'll be lacking certain detail information. Additionally, you'll be thinking about your entities in terms of facts and dimensions (discussed below). And finally, unlike a traditional data model and database, your warehouse data model will be an almost exact representation of the physical data warehouse.

UNDERSTANDING THE DATA

In order to facilitate a discussion around data modeling for a warehouse, it will be helpful to have an example project to work with. For our purposes, let us suppose we are building a data model for a data warehouse that will support a simple retailing business (a very common business model). The data warehouse we are building will house all relevant historical data to support decisions around products that are carried, inventory numbers and employee workload. We'll need to be able to store and calculate sales history by product, by time-line, and by geography. We'll also need to be able to serve up data around warehouse activity (the physical warehouse, not the data warehouse) so we can determine staffing levels during the appropriate times of the year.

The very first thing you have to do, as with any development project, is begin to understand what data you are working with. With a data warehouse, there may be carry-over knowledge from the design and development of the operational data stores. However, more often than not, if you are a BI focused developer, you may be in the position of learning about the data (and therefore the business) before you can even begin to build anything. This means interviewing users, reviewing existing applications and documentation, and beginning to build a rough data dictionary for your own use.

One of the most important aspects of this information gathering phase is the discovery of all of the potential data sources for the warehouse. Many businesses have not only their primary operation database (usually a relational data-base), they may store process specific data in any number of formats; XML, flat file databases (such as Dbase), network shares, etc. Additionally, in the inception of the data ware-house, there may be archived historical data that must be loaded during the initial implementation. This data may be stored on tapes, and in any number of formats. Be prepared to discover quite a bit about the history of data storage in the business.

BUILDING THE MODEL

Once you have an understanding of the data that will be stored in the warehouse, it'll be time to actually build a data model. You'll identify your entities as two different types; facts and dimensions. Fact entities are entities that store the raw value information, such as sales totals, minutes, etc. Dimension tables store related data in terms of how you are measuring the facts. Examples of dimensions are time based data (days, months, years, etc.) and geographical locations (city, state, country, continent). When you are building queries to analyze data, you'll be retrieving fact data based on specific dimensions, so keep in mind that you need to look at your notes to see all of the possible dimensions, and create separate entities for them.

Once you have a list of the known facts and dimensions, we'll have fact entities and dimensional entities, and they will be related to one another in a manner that facilitates understanding of the relationships between those entities. 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 entity, related to a single layer of dimension entities. Each dimension has a simple primary key, as it is responsible for a single type of data. The fact entity has an aggregate primary key, as it is the compounded collection of the primary keys from the dimensions. Figure 1 shows a very simple star schema outlining sales data.

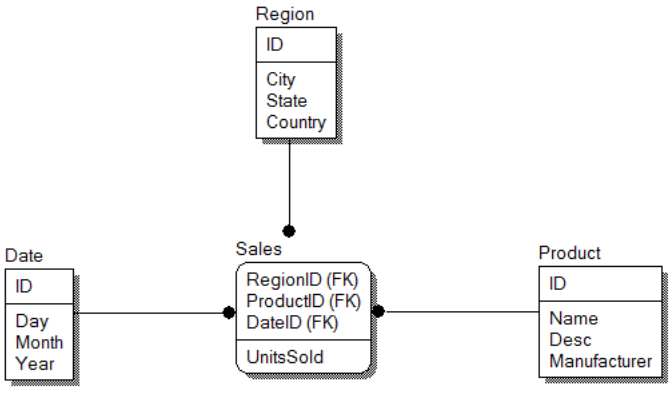


Figure 1. *Simple star schema.*

The simplicity of a star schema is that for each fact entity, you will only ever have to join one level of dimension entities to get at the data. Unfortunately, the drawback of a star

schema is that for complex 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 facts at its center, with relationships to dimensions. However, if there are dimensions with large numbers of attributes, it might be necessary to break the dimensions down into sub-dimension entities. The end result is a schema that visually looks like a snowflake, because of the layers of dimensions. Figure 2 shows a slightly expanded view of the sales data in a data warehouse, employing a snowflake schema.

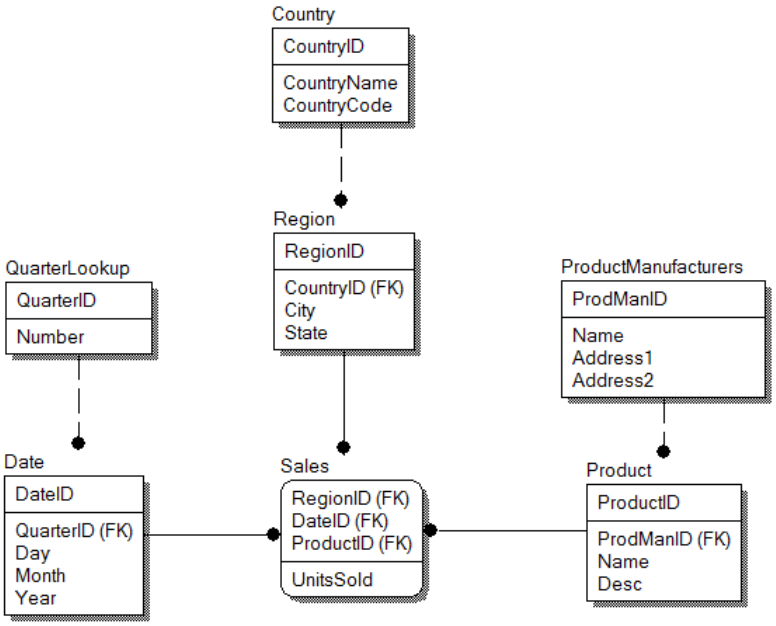


Figure 2. *Snowflake schema.*

While this is not an exhaustive look at a data model, it shows the basics of a snowflake schema. There is a central fact entity, Sales, and two layers of dimensions, 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 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.

Once you've settled on a schema (for most warehouses, it will be a snowflake; for data marts, often times star works well), you can start plugging in your specific facts and dimensions. At this point, it's imperative to constantly review the model as you build it looking for key design problems such as duplicate data and missing facts and/or dimensions. Go back to the specifications of the project. Are you able to answer all of the questions using your model? Don't get lost in the process of designing the model, which is easy to do when focusing on proper design.

BUILDING THE MODEL

Let's now go back to the original sample project. 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.

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 on page 6. 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.

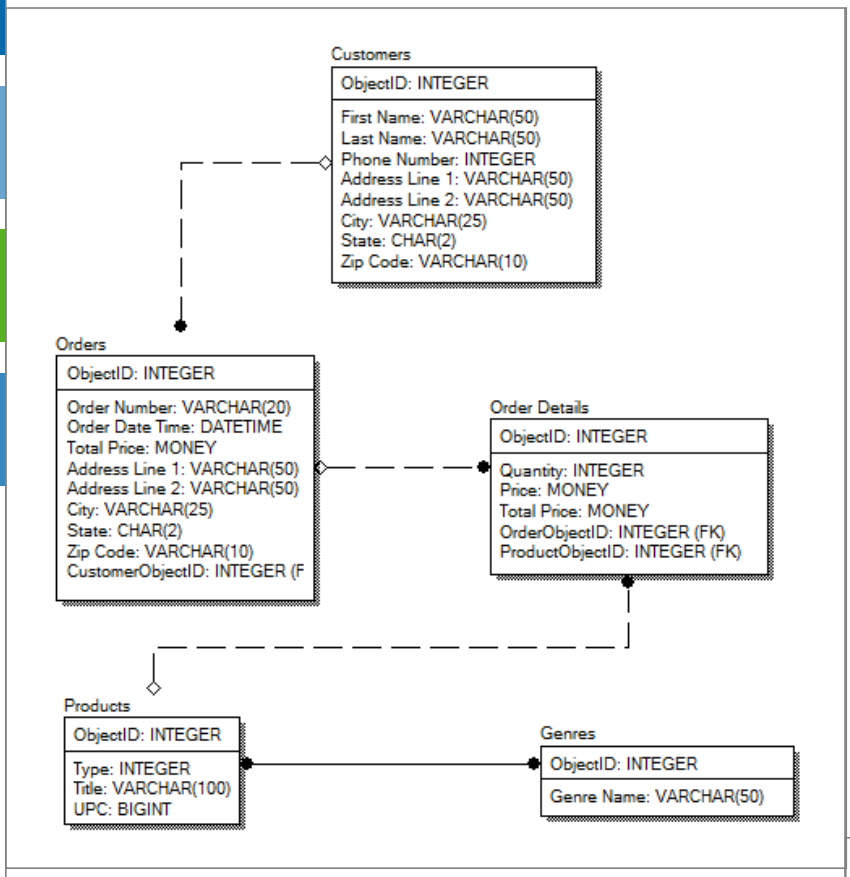


Figure 3. *Relational data model source.*

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.

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. So, Figure 5 on page 7 shows the addition of the DimProducts entity, and the ensuing relationship between the dimension and fact entities.

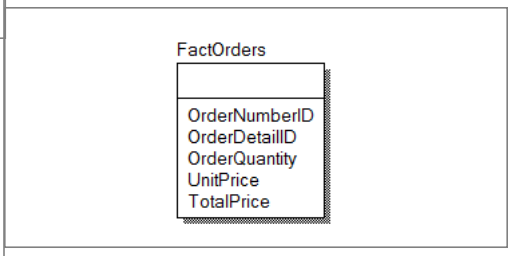


Figure 4. *The FactOrders entity, with no dimensions associated.*

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 on page 7 shows us the data model for our very simple data warehouse.

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 ware-

house 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.

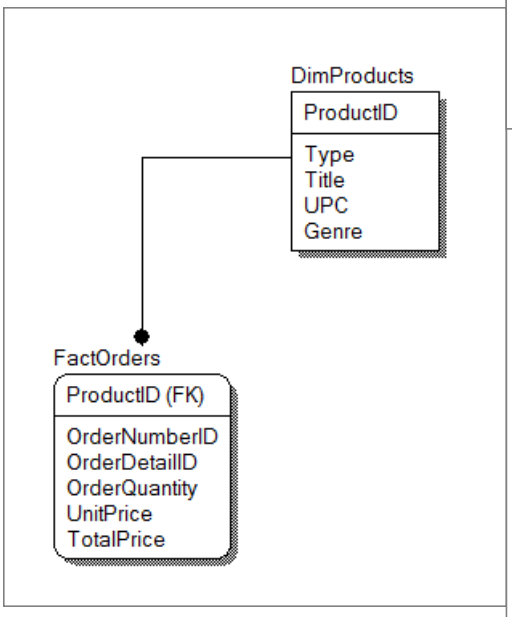


Figure 5. *The FactOrders entity and the DimProducts entity.*

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,

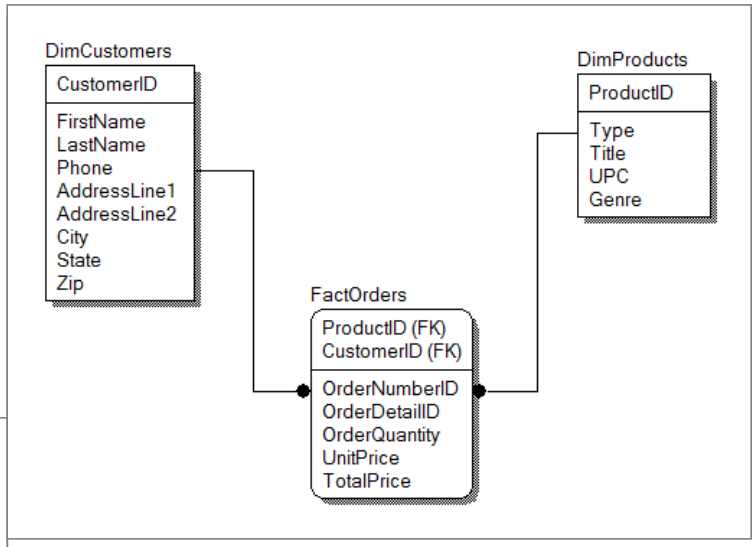


Figure 6. *A very simple star schema for the orders information.*

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.

BUILDING THE DATA WAREHOUSE

Once you have a fully functional model, it'll be time to implement a physical database. This is similar to deploying any other database. And since we're working with SQL Server and ERWin, you'll be able to generate and deploy a physical schema based directly on your model. Again, a data warehouse database will look nearly (if not actually) identical to your model, so this may be the simplest phase, depending on

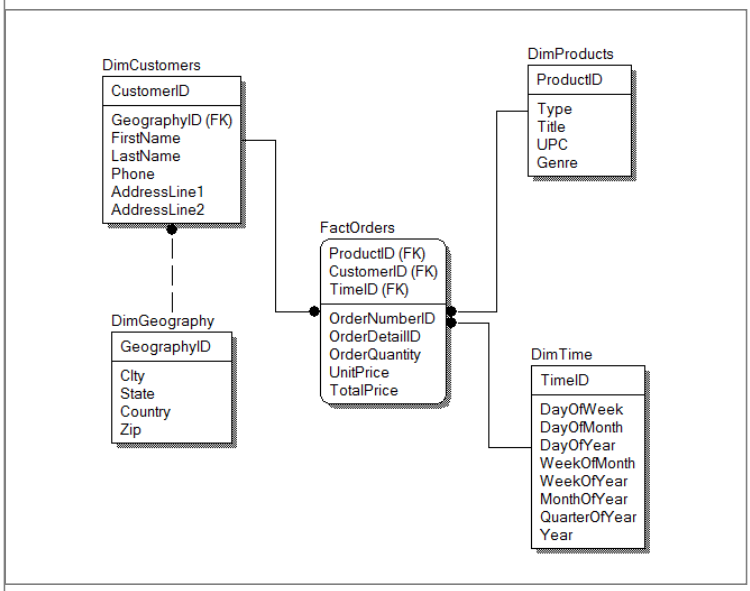


Figure 7. *A more fleshed out data warehouse star schema.*

the level of complexity inside the actual warehouse. Also, remember that this doesn't include the ETL (extraction, transform, and load) process, which will be significantly more complicated (and will be dealt with in the next paper in this series).

If you've used a data modeling tool, such as ERWin, then most likely this will be the easiest stage of the process. Most modeling tools support the generation of actual database schema, custom fitted for your RDBMS of choice. Usually, these are scripts, though in many cases the tool can connect directly to the RDBMS to create the database for you. Either way, you'll be able to take the data model itself and immediately create the database. While in relational design, the model may not always exactly mirror the physical database, with a data warehouse, it's likely to be very close. You will probably only need to create those objects in the physical database that can enhance performance, such as indexes and statistics.

USING SSAS TO ANALYZE THE DATA

Finally, once you have a warehouse built, you can begin building a cube that will assist in analyzing and presenting your data. The cube will take the fact and dimension data as inputs, and you can program any number of calculations and processing to add to and divide up your data as needed.

SQL Server Analysis Services (SSAS) provides quite a few pre-built aggregates that will help you quickly deploy at least the basic information you need. There is also the added benefit of moving the query processing of the data to a platform designed for delivering a high volume of calculated data to an end-user. While SQL Server's relational engine is a high-performance one, a relational database is inherently slow when trying to query large chunks of aggregated data. Analysis Services gives you a common platform to use with all types of end-user querying and reporting solutions such as Microsoft Office Excel, Microsoft SQL Server Reporting Services, Business Objects' Crystal Reports, and Proclarity. You can even develop custom querying applications using very standard interfaces, and the Analysis Services query language known as Multidimensional Expressions (MDX).

One important note: Having a physical data warehouse in a relational engine is not a prerequisite for using Analysis Services. It can BE your data warehouse, using your operational data stores as it's data source. However, this is generally a bad practice, particularly in large environments. This is because you have to refresh the data in the warehouse regularly, which requires a large amount of resources on the data source (since it is essentially querying the data). If the data source is also your operational database, you'll impact day-to-day application usage every time you refresh the warehouse. If nothing else, you should at least use a staging database, where you move the data from the operational data store via an ETL process (which will be discussed later in the series) out of the operational database at a reasonable rate. You can then point Analysis Services at the staging database, and import the data from there without affecting application usage. Conversely, you do not need Analysis Services to have a data warehouse. The type of warehouse we have been discussing, using an RDBMS to host a custom designed database, can be used for any data warehousing project. However, you will not have the ease of use that Analysis Services provides, particularly around performing calculations on the warehouse data. Analysis Services uses the Visual Studio development environment, which can make design and maintenance much easier for developers. It's also easier to develop entire BI solutions in Visual Studio, where the data warehouse is only one piece of the solution. As always, each environment and project will be different, and your mileage may vary.

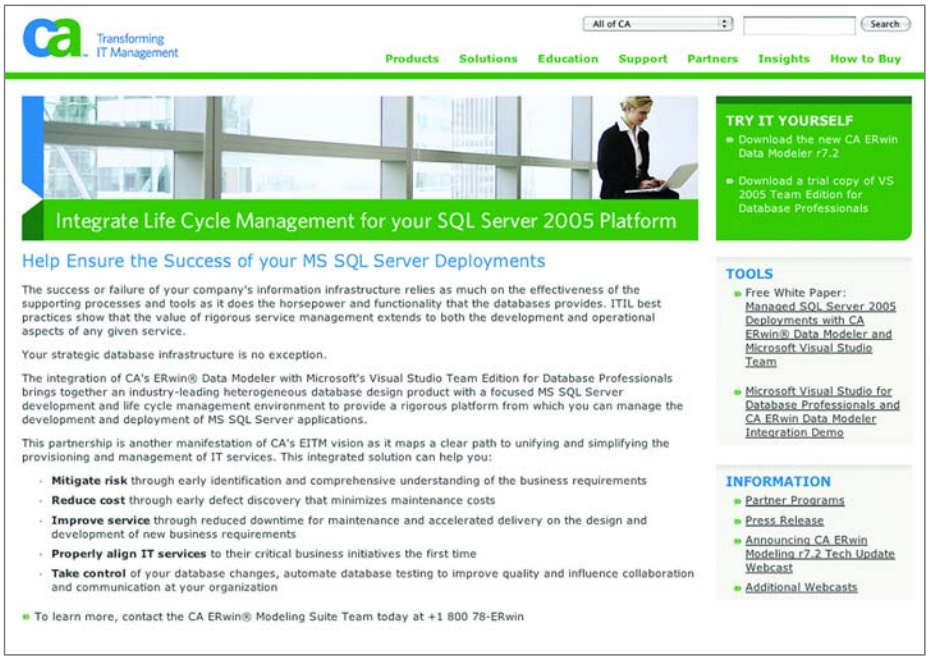
Once you have a data source (either a full blown data warehouse or at least a staging database), you can use Analysis Services' wizards to create a cube. It will allow you to create facts and dimensions in the cube that are identical to your warehouse, and additional facts and/or dimensions as needed. Then you can begin to use the enhancements available in Analysis Services to perform manipulations of the data. Unfortunately, Analysis Services is a huge product, and describing all of its functionality is outside of our scope. Just know that it can considerably enhance any data warehouse, while simultaneously easing the life of the developer.

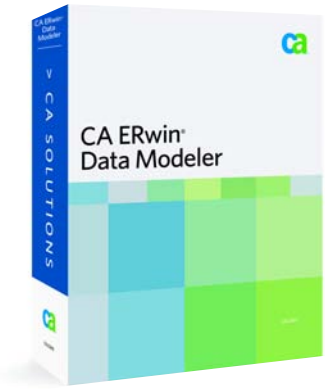
CONCLUSION

Just like any other design process, building a data warehouse requires a great deal of preparation and legwork to make sure that the development and deployment will be as smooth as possible. And using advanced tools such as Microsoft SQL Server Analysis Services and CA's ERWin Data Modeler will help ensure that you not only create a powerful solution, but that it is fully documented and scalable for the future.

ABOUT CONSORTIO SERVICES

Josh Jones and Eric Johnson are the Operating Systems and Database Systems Consultants, respectively, for Consortio Services, as well as co-hosts for the popular technology podcast "CS Techcast" (www.cstechcast.com). They offer consulting and training services in many areas of Information Technology. Check our www.consortioservices.com for more information. Eric and Josh have written a combined three books including the forthcoming "A Developers Guide to Data Modeling for SQL Server".





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