CHAPTER 1

Dimensional Modeling Primer

In this first chapter we lay the groundwork for the case studies that follow. We'll begin by stepping back to consider data warehousing from a macro perspective. Some readers may be disappointed to learn that it is not all about tools and techniques—first and foremost, the data warehouse must consider the needs of the business. We'll drive stakes in the ground regarding the goals of the data warehouse while observing the uncanny similarities between the responsibilities of a data warehouse manager and those of a publisher. With this big-picture perspective, we'll explore the major components of the warehouse environment, including the role of normalized models. Finally, we'll close by establishing fundamental vocabulary for dimensional modeling. By the end of this chapter we hope that you'll have an appreciation for the need to be half DBA (database administrator) and half MBA (business analyst) as you tackle your data warehouse.

Chapter 1 discusses the following concepts:

- **Business-driven goals of a data warehouse**
- **Data warehouse publishing**
- **Major components of the overall data warehouse**
- **Importance of dimensional modeling for the data warehouse presentation area**
- **Fact and dimension table terminology**
- **Myths surrounding dimensional modeling**
- **Common data warehousing pitfalls to avoid**

Different Information Worlds

One of the most important assets of any organization is its information. This asset is almost always kept by an organization in two forms: the operational systems of record and the data warehouse. Crudely speaking, the operational systems are where the data is put in, and the data warehouse is where we get the data out.

The users of an operational system *turn* the wheels of the organization. They take orders, sign up new customers, and log complaints. Users of an operational system almost always deal with one record at a time. They repeatedly perform the same operational tasks over and over.

The users of a data warehouse, on the other hand, *watch* the wheels of the organization turn. They count the new orders and compare them with last week's orders and ask why the new customers signed up and what the customers complained about. Users of a data warehouse almost never deal with one row at a time. Rather, their questions often require that hundreds or thousands of rows be searched and compressed into an answer set. To further complicate matters, users of a data warehouse continuously change the kinds of questions they ask.

In the first edition of *The Data Warehouse Toolkit* (Wiley 1996), Ralph Kimball devoted an entire chapter to describe the dichotomy between the worlds of operational processing and data warehousing. At this time, it is widely recognized that the data warehouse has profoundly different needs, clients, structures, and rhythms than the operational systems of record. Unfortunately, we continue to encounter supposed data warehouses that are mere copies of the operational system of record stored on a separate hardware platform. While this may address the need to isolate the operational and warehouse environments for performance reasons, it does nothing to address the other inherent differences between these two types of systems. Business users are underwhelmed by the usability and performance provided by these pseudo data warehouses. These imposters do a disservice to data warehousing because they don't acknowledge that warehouse users have drastically different needs than operational system users.

Goals of a Data Warehouse

Before we delve into the details of modeling and implementation, it is helpful to focus on the fundamental goals of the data warehouse. The goals can be developed by walking through the halls of any organization and listening to business management. Inevitably, these recurring themes emerge:

- "We have mountains of data in this company, but we can't access it."
- "We need to slice and dice the data every which way."
- "You've got to make it easy for business people to get at the data directly."
- "Just show me what is important."
- "It drives me crazy to have two people present the same business metrics at a meeting, but with different numbers."
- "We want people to use information to support more fact-based decision making."

Based on our experience, these concerns are so universal that they drive the bedrock requirements for the data warehouse. Let's turn these business management quotations into data warehouse requirements.

The data warehouse must make an organization's information easily accessible. The contents of the data warehouse must be understandable. The data must be intuitive and obvious to the business user, not merely the developer. Understandability implies legibility; the contents of the data warehouse need to be labeled meaningfully. Business users want to separate and combine the data in the warehouse in endless combinations, a process commonly referred to as *slicing and dicing*. The tools that access the data warehouse must be simple and easy to use. They also must return query results to the user with minimal wait times.

The data warehouse must present the organization's information consistently. The data in the warehouse must be credible. Data must be carefully assembled from a variety of sources around the organization, cleansed, quality assured, and released only when it is fit for user consumption. Information from one business process should match with information from another. If two performance measures have the same name, then they must mean the same thing. Conversely, if two measures don't mean the same thing, then they should be labeled differently. Consistent information means high-quality information. It means that all the data is accounted for and complete. Consistency also implies that common definitions for the contents of the data warehouse are available for users.

The data warehouse must be adaptive and resilient to change. We simply can't avoid change. User needs, business conditions, data, and technology are all subject to the shifting sands of time. The data warehouse must be designed to handle this inevitable change. Changes to the data warehouse should be graceful, meaning that they don't invalidate existing data or applications. The existing data and applications should not be changed or disrupted when the business community asks new questions or new data is added to the warehouse. If descriptive data in the warehouse is modified, we must account for the changes appropriately.

The data warehouse must be a secure bastion that protects our information assets. An organization's informational crown jewels are stored in the data warehouse. At a minimum, the warehouse likely contains information about what we're selling to whom at what price—potentially harmful details in the hands of the wrong people. The data warehouse must effectively control access to the organization's confidential information.

The data warehouse must serve as the foundation for improved decision making. The data warehouse must have the right data in it to support decision making. There is only one true output from a data warehouse: the decisions that are made after the data warehouse has presented its evidence. These decisions deliver the business impact and value attributable to the warehouse. The original label that predates the data warehouse is still the best description of what we are designing: a decision support system.

The business community must accept the data warehouse if it is to be deemed successful. It doesn't matter that we've built an elegant solution using best-of-breed products and platforms. If the business community has not embraced the data warehouse and continued to use it actively six months after training, then we have failed the acceptance test. Unlike an operational system rewrite, where business users have no choice but to use the new system, data warehouse usage is sometimes optional. Business user acceptance has more to do with simplicity than anything else.

As this list illustrates, successful data warehousing demands much more than being a stellar DBA or technician. With a data warehousing initiative, we have one foot in our information technology (IT) comfort zone, while our other foot is on the unfamiliar turf of business users. We must straddle the two, modifying some of our tried-and-true skills to adapt to the unique demands of data warehousing. Clearly, we need to bring a bevy of skills to the party to behave like we're a hybrid DBA/MBA.

The Publishing Metaphor

With the goals of the data warehouse as a backdrop, let's compare our responsibilities as data warehouse managers with those of a publishing editor-in-chief. As the editor of a high-quality magazine, you would be given broad latitude to manage the magazine's content, style, and delivery. Anyone with this job title likely would tackle the following activities:

- Identify your readers demographically.
- Find out what the readers want in this kind of magazine.
- Identify the "best" readers who will renew their subscriptions and buy products from the magazine's advertisers.

- Find potential new readers and make them aware of the magazine.
- Choose the magazine content most appealing to the target readers.
- Make layout and rendering decisions that maximize the readers' pleasure.
- Uphold high quality writing and editing standards, while adopting a consistent presentation style.
- Continuously monitor the accuracy of the articles and advertiser's claims.
- Develop a good network of writers and contributors as you gather new input to the magazine's content from a variety of sources.
- Attract advertising and run the magazine profitably.
- Publish the magazine on a regular basis.
- Maintain the readers' trust.
- Keep the business owners happy.

We also can identify items that should be nongoals for the magazine editor-in-chief. These would include such things as building the magazine around the technology of a particular printing press, putting management's energy into operational efficiencies exclusively, imposing a technical writing style that readers don't easily understand, or creating an intricate and crowded layout that is difficult to peruse and read.

By building the publishing business on a foundation of serving the readers effectively, your magazine is likely to be successful. Conversely, go through the list and imagine what happens if you omit any single item; ultimately, your magazine would have serious problems.

The point of this metaphor, of course, is to draw the parallel between being a conventional publisher and being a data warehouse manager. We are convinced that the correct job description for a data warehouse manager is *publisher of the right data.* Driven by the needs of the business, data warehouse managers are responsible for publishing data that has been collected from a variety of sources and edited for quality and consistency. Your main responsibility as a data warehouse manager is to serve your readers, otherwise known as business users. The publishing metaphor underscores the need to focus outward to your customers rather than merely focusing inward on products and processes. While you will use technology to deliver your data warehouse, the technology is at best a means to an end. As such, the technology and techniques you use to build your data warehouses should not appear directly in your top job responsibilities.

Let's recast the magazine publisher's responsibilities as data warehouse manager responsibilities:

- Understand your users by business area, job responsibilities, and computer tolerance.
- Determine the decisions the business users want to make with the help of the data warehouse.
- Identify the "best" users who make effective, high-impact decisions using the data warehouse.
- Find potential new users and make them aware of the data warehouse.
- Choose the most effective, actionable subset of the data to present in the data warehouse, drawn from the vast universe of possible data in your organization.
- Make the user interfaces and applications simple and template-driven, explicitly matching to the users' cognitive processing profiles.
- Make sure the data is accurate and can be trusted, labeling it consistently across the enterprise.
- Continuously monitor the accuracy of the data and the content of the delivered reports.
- Search for new data sources, and continuously adapt the data warehouse to changing data profiles, reporting requirements, and business priorities.
- Take a portion of the credit for the business decisions made using the data warehouse, and use these successes to justify your staffing, software, and hardware expenditures.
- Publish the data on a regular basis.
- Maintain the trust of business users.
- Keep your business users, executive sponsors, and boss happy.

If you do a good job with all these responsibilities, you will be a great data warehouse manager! Conversely, go down through the list and imagine what happens if you omit any single item. Ultimately, your data warehouse would have serious problems. We urge you to contrast this view of a data warehouse manager's job with your own job description. Chances are the preceding list is much more oriented toward user and business issues and may not even sound like a job in IT. In our opinion, this is what makes data warehousing interesting.

Components of a Data Warehouse

Now that we understand the goals of a data warehouse, let's investigate the components that make up a complete warehousing environment. It is helpful to understand the pieces carefully before we begin combining them to create a

data warehouse. Each warehouse component serves a specific function. We need to learn the strategic significance of each component and how to wield it effectively to win the data warehousing game. One of the biggest threats to data warehousing success is confusing the components' roles and functions.

As illustrated in Figure 1.1, there are four separate and distinct components to be considered as we explore the data warehouse environment—operational source systems, data staging area, data presentation area, and data access tools.

Operational Source Systems

These are the operational systems of record that capture the transactions of the business. The source systems should be thought of as outside the data warehouse because presumably we have little to no control over the content and format of the data in these operational legacy systems. The main priorities of the source systems are processing performance and availability. Queries against source systems are narrow, one-record-at-a-time queries that are part of the normal transaction flow and severely restricted in their demands on the operational system. We make the strong assumption that source systems are not queried in the broad and unexpected ways that data warehouses typically are queried. The source systems maintain little historical data, and if you have a good data warehouse, the source systems can be relieved of much of the responsibility for representing the past. Each source system is often a natural stovepipe application, where little investment has been made to sharing common data such as product, customer, geography, or calendar with other operational systems in the organization. It would be great if your source systems were being reengineered with a consistent view. Such an enterprise application integration (EAI) effort will make the data warehouse design task far easier.

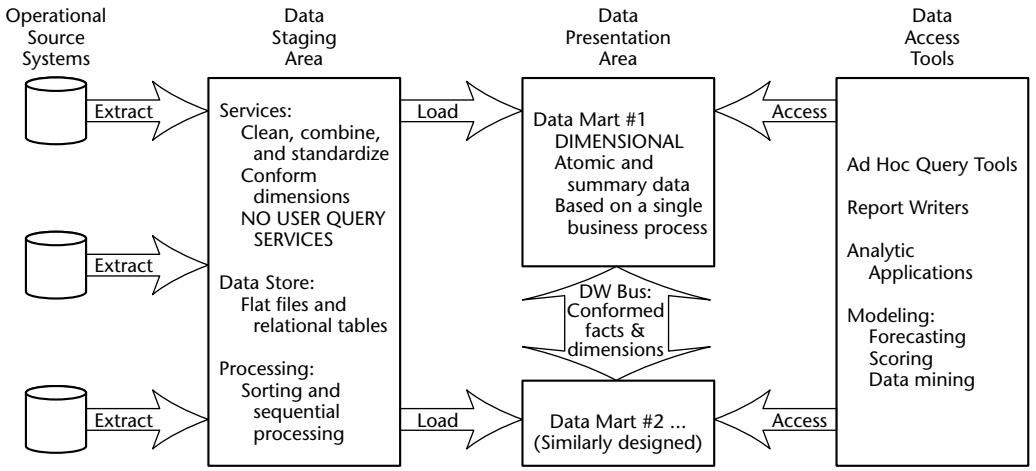


Figure 1.1 Basic elements of the data warehouse.

Data Staging Area

The data staging area of the data warehouse is both a storage area and a set of processes commonly referred to as *extract-transformation-load* (ETL). The data staging area is everything between the operational source systems and the data presentation area. It is somewhat analogous to the kitchen of a restaurant, where raw food products are transformed into a fine meal. In the data warehouse, raw operational data is transformed into a warehouse deliverable fit for user query and consumption. Similar to the restaurant's kitchen, the backroom data staging area is accessible only to skilled professionals. The data warehouse kitchen staff is busy preparing meals and simultaneously cannot be responding to customer inquiries. Customers aren't invited to eat in the kitchen. It certainly isn't safe for customers to wander into the kitchen. We wouldn't want our data warehouse customers to be injured by the dangerous equipment, hot surfaces, and sharp knifes they may encounter in the kitchen, so we prohibit them from accessing the staging area. Besides, things happen in the kitchen that customers just shouldn't be privy to.

The key architectural requirement for the data staging area is that it is off-limits to business users and does *not* provide query and presentation services.

Extraction is the first step in the process of getting data into the data warehouse environment. Extracting means reading and understanding the source data and copying the data needed for the data warehouse into the staging area for further manipulation.

Once the data is extracted to the staging area, there are numerous potential transformations, such as cleansing the data (correcting misspellings, resolving domain conflicts, dealing with missing elements, or parsing into standard formats), combining data from multiple sources, deduplicating data, and assigning warehouse keys. These transformations are all precursors to loading the data into the data warehouse presentation area.

Unfortunately, there is still considerable industry consternation about whether the data that supports or results from this process should be instantiated in physical normalized structures prior to loading into the presentation area for querying and reporting. These normalized structures sometimes are referred to in the industry as the *enterprise data warehouse*; however, we believe that this terminology is a misnomer because the warehouse is actually much more encompassing than this set of normalized tables. The enterprise's data warehouse more accurately refers to the conglomeration of an organization's data warehouse staging and presentation areas. Thus, throughout this book, when we refer to the enterprise data warehouse, we mean the union of all the diverse data warehouse components, not just the backroom staging area.

The data staging area is dominated by the simple activities of sorting and sequential processing. In many cases, the data staging area is not based on relational technology but instead may consist of a system of flat files. After you validate your data for conformance with the defined one-to-one and many-to-one business rules, it may be pointless to take the final step of building a full-blown third-normal-form physical database.

However, there are cases where the data arrives at the doorstep of the data staging area in a third-normal-form relational format. In these situations, the managers of the data staging area simply may be more comfortable performing the cleansing and transformation tasks using a set of normalized structures. A normalized database for data staging storage is acceptable. However, we continue to have some reservations about this approach. The creation of both normalized structures for staging and dimensional structures for presentation means that the data is extracted, transformed, and loaded twice—once into the normalized database and then again when we load the dimensional model. Obviously, this two-step process requires more time and resources for the development effort, more time for the periodic loading or updating of data, and more capacity to store the multiple copies of the data. At the bottom line, this typically translates into the need for larger development, ongoing support, and hardware platform budgets. Unfortunately, some data warehouse project teams have failed miserably because they focused all their energy and resources on constructing the normalized structures rather than allocating time to development of a presentation area that supports improved business decision making. While we believe that enterprise-wide data consistency is a fundamental goal of the data warehouse environment, there are equally effective and less costly approaches than physically creating a normalized set of tables in your staging area, if these structures don't already exist.

It is acceptable to create a normalized database to support the staging processes; however, this is not the end goal. The normalized structures must be off-limits to user queries because they defeat understandability and performance. As soon as a database supports query and presentation services, it must be considered part of the data warehouse presentation area. By default, normalized databases are excluded from the presentation area, which should be strictly dimensionally structured.

Regardless of whether we're working with a series of flat files or a normalized data structure in the staging area, the final step of the ETL process is the loading of data. Loading in the data warehouse environment usually takes the form of presenting the quality-assured dimensional tables to the bulk loading facilities of each data mart. The target data mart must then index the newly arrived data for query performance. When each data mart has been freshly loaded, indexed, supplied with appropriate aggregates, and further quality

assured, the user community is notified that the new data has been published. Publishing includes communicating the nature of any changes that have occurred in the underlying dimensions and new assumptions that have been introduced into the measured or calculated facts.

Data Presentation

The data presentation area is where data is organized, stored, and made available for direct querying by users, report writers, and other analytical applications. Since the backroom staging area is off-limits, the presentation area *is* the data warehouse as far as the business community is concerned. It is all the business community sees and touches via data access tools. The prerelease working title for the first edition of *The Data Warehouse Toolkit* originally was *Getting the Data Out*. This is what the presentation area with its dimensional models is all about.

We typically refer to the presentation area as a series of integrated data marts. A data mart is a wedge of the overall presentation area pie. In its most simplistic form, a data mart presents the data from a single business process. These business processes cross the boundaries of organizational functions.

We have several strong opinions about the presentation area. First of all, we insist that the data be presented, stored, and accessed in dimensional schemas. Fortunately, the industry has matured to the point where we're no longer debating this mandate. The industry has concluded that dimensional modeling is the most viable technique for delivering data to data warehouse users.

Dimensional modeling is a new name for an old technique for making databases simple and understandable. In case after case, beginning in the 1970s, IT organizations, consultants, end users, and vendors have gravitated to a simple dimensional structure to match the fundamental human need for simplicity. Imagine a chief executive officer (CEO) who describes his or her business as, "We sell products in various markets and measure our performance over time." As dimensional designers, we listen carefully to the CEO's emphasis on product, market, and time. Most people find it intuitive to think of this business as a cube of data, with the edges labeled product, market, and time. We can imagine slicing and dicing along each of these dimensions. Points inside the cube are where the measurements for that combination of product, market, and time are stored. The ability to visualize something as abstract as a set of data in a concrete and tangible way is the secret of understandability. If this perspective seems too simple, then good! A data model that starts by being simple has a chance of remaining simple at the end of the design. A model that starts by being complicated surely will be overly complicated at the end. Overly complicated models will run slowly and be rejected by business users.

Dimensional modeling is quite different from third-normal-form (3NF) modeling. 3NF modeling is a design technique that seeks to remove data redundancies. Data is divided into many discrete entities, each of which becomes a table in the relational database. A database of sales orders might start off with a record for each order line but turns into an amazingly complex spiderweb diagram as a 3NF model, perhaps consisting of hundreds or even thousands of normalized tables.

The industry sometimes refers to 3NF models as *ER models.* ER is an acronym for *entity relationship.* Entity-relationship diagrams (ER diagrams or ERDs) are drawings of boxes and lines to communicate the relationships between tables. Both 3NF and dimensional models can be represented in ERDs because both consist of joined relational tables; the key difference between 3NF and dimensional models is the degree of normalization. Since both model types can be presented as ERDs, we'll refrain from referring to 3NF models as ER models; instead, we'll call them *normalized models* to minimize confusion.

Normalized modeling is immensely helpful to operational processing performance because an update or insert transaction only needs to touch the database in one place. Normalized models, however, are too complicated for data warehouse queries. Users can't understand, navigate, or remember normalized models that resemble the Los Angeles freeway system. Likewise, relational database management systems (RDBMSs) can't query a normalized model efficiently; the complexity overwhelms the database optimizers, resulting in disastrous performance. The use of normalized modeling in the data warehouse presentation area defeats the whole purpose of data warehousing, namely, intuitive and high-performance retrieval of data.

There is a common syndrome in many large IT shops. It is a kind of sickness that comes from overly complex data warehousing schemas. The symptoms might include:

- A $10 million hardware and software investment that is performing only a handful of queries per day
- An IT department that is forced into a kind of priesthood, writing all the data warehouse queries
- Seemingly simple queries that require several pages of single-spaced Structured Query Language (SQL) code
- A marketing department that is unhappy because it can't access the system directly (and still doesn't know whether the company is profitable in Schenectady)
- A restless chief information officer (CIO) who is determined to make some changes if things don't improve dramatically

Fortunately, dimensional modeling addresses the problem of overly complex schemas in the presentation area. A dimensional model contains the same information as a normalized model but packages the data in a format whose design goals are user understandability, query performance, and resilience to change.

Our second stake in the ground about presentation area data marts is that they must contain detailed, atomic data. Atomic data is required to withstand assaults from unpredictable ad hoc user queries. While the data marts also may contain performance-enhancing summary data, or aggregates, it is not sufficient to deliver these summaries without the underlying granular data in a dimensional form. In other words, it is completely unacceptable to store only summary data in dimensional models while the atomic data is locked up in normalized models. It is impractical to expect a user to drill down through dimensional data almost to the most granular level and then lose the benefits of a dimensional presentation at the final step. In Chapter 16 we will see that any user application can descend effortlessly to the bedrock granular data by using aggregate navigation, but only if all the data is available in the same, consistent dimensional form. While users of the data warehouse may look infrequently at a single line item on an order, they may be very interested in last week's orders for products of a given size (or flavor, package type, or manufacturer) for customers who first purchased within the last six months (or reside in a given state or have certain credit terms). We need the most finely grained data in our presentation area so that users can ask the most precise questions possible. Because users' requirements are unpredictable and constantly changing, we must provide access to the exquisite details so that they can be rolled up to address the questions of the moment.

All the data marts must be built using common dimensions and facts, which we refer to as *conformed*. This is the basis of the data warehouse bus architecture, which we'll elaborate on in Chapter 3. Adherence to the bus architecture is our third stake in the ground regarding the presentation area. Without shared, conformed dimensions and facts, a data mart is a standalone stovepipe application. Isolated stovepipe data marts that cannot be tied together are the bane of the data warehouse movement. They merely perpetuate incompatible views of the enterprise. If you have any hope of building a data warehouse that is robust and integrated, you must make a commitment to the bus architecture. In this book we will illustrate that when data marts have been designed with conformed dimensions and facts, they can be combined and used together. The data warehouse presentation area in a large enterprise data warehouse ultimately will consist of 20 or more very similar-looking data marts. The dimensional models in these data marts also will look quite similar. Each data mart may contain several fact tables, each with 5 to 15 dimension tables. If the design has been done correctly, many of these dimension tables will be shared from fact table to fact table.

Using the bus architecture is the secret to building distributed data warehouse systems. Let's be real—most of us don't have the budget, time, or political power to build a fully centralized data warehouse. When the bus architecture is used as a framework, we can allow the enterprise data warehouse to develop in a decentralized (and far more realistic) way.

Data in the queryable presentation area of the data warehouse must be dimensional, must be atomic, and must adhere to the data warehouse bus architecture.

If the presentation area is based on a relational database, then these dimensionally modeled tables are referred to as *star schemas.* If the presentation area is based on multidimensional database or online analytic processing (OLAP) technology, then the data is stored in *cubes.* While the technology originally wasn't referred to as OLAP, many of the early decision support system vendors built their systems around the cube concept, so today's OLAP vendors naturally are aligned with the dimensional approach to data warehousing. Dimensional modeling is applicable to both relational and multidimensional databases. Both have a common logical design with recognizable dimensions; however, the physical implementation differs. Fortunately, most of the recommendations in this book pertain, regardless of the database platform. While the capabilities of OLAP technology are improving continuously, at the time of this writing, most large data marts are still implemented on relational databases. In addition, most OLAP cubes are sourced from or drill into relational dimensional star schemas using a variation of aggregate navigation. For these reasons, most of the specific discussions surrounding the presentation area are couched in terms of a relational platform.

Contrary to the original religion of the data warehouse, modern data marts may well be updated, sometimes frequently. Incorrect data obviously should be corrected. Changes in labels, hierarchies, status, and corporate ownership often trigger necessary changes in the original data stored in the data marts that comprise the data warehouse, but in general, these are managed-load updates, not transactional updates.

Data Access Tools

The final major component of the data warehouse environment is the *data access tool(s)*. We use the term tool loosely to refer to the variety of capabilities that can be provided to business users to leverage the presentation area for analytic decision making. By definition, all data access tools query the data in the data warehouse's presentation area. Querying, obviously, is the whole point of using the data warehouse.

A data access tool can be as simple as an ad hoc query tool or as complex as a sophisticated data mining or modeling application. Ad hoc query tools, as powerful as they are, can be understood and used effectively only by a small percentage of the potential data warehouse business user population. The majority of the business user base likely will access the data via prebuilt parameter-driven analytic applications. Approximately 80 to 90 percent of the potential users will be served by these canned applications that are essentially finished templates that do not require users to construct relational queries directly. Some of the more sophisticated data access tools, like modeling or forecasting tools, actually may upload their results back into operational source systems or the staging/presentation areas of the data warehouse.

Additional Considerations

Before we leave the discussion of data warehouse components, there are several other concepts that warrant discussion.

Metadata

Metadata is all the information in the data warehouse environment that is not the actual data itself. Metadata is akin to an encyclopedia for the data warehouse. Data warehouse teams often spend an enormous amount of time talking about, worrying about, and feeling guilty about metadata. Since most developers have a natural aversion to the development and orderly filing of documentation, metadata often gets cut from the project plan despite everyone's acknowledgment that it is important.

Metadata comes in a variety of shapes and forms to support the disparate needs of the data warehouse's technical, administrative, and business user groups. We have operational source system metadata including source schemas and copybooks that facilitate the extraction process. Once data is in the staging area, we encounter staging metadata to guide the transformation and loading processes, including staging file and target table layouts, transformation and cleansing rules, conformed dimension and fact definitions, aggregation definitions, and ETL transmission schedules and run-log results. Even the custom programming code we write in the data staging area is metadata.

Metadata surrounding the warehouse DBMS accounts for such items as the system tables, partition settings, indexes, view definitions, and DBMS-level security privileges and grants. Finally, the data access tool metadata identifies business names and definitions for the presentation area's tables and columns as well as constraint filters, application template specifications, access and usage statistics, and other user documentation. And of course, if we haven't

included it already, don't forget all the security settings, beginning with source transactional data and extending all the way to the user's desktop.

The ultimate goal is to corral, catalog, integrate, and then leverage these disparate varieties of metadata, much like the resources of a library. Suddenly, the effort to build dimensional models appears to pale in comparison. However, just because the task looms large, we can't simply ignore the development of a metadata framework for the data warehouse. We need to develop an overall metadata plan while prioritizing short-term deliverables, including the purchase or construction of a repository for keeping track of all the metadata.

Operational Data Store

Some of you probably are wondering where the operational data store (ODS) fits in our warehouse components diagram. Since there's no single universal definition for the ODS, if and where it belongs depend on your situation. ODSs are frequently updated, somewhat integrated copies of operational data. The frequency of update and degree of integration of an ODS vary based on the specific requirements. In any case, the *O* is the operative letter in the ODS acronym.

Most commonly, an ODS is implemented to deliver operational reporting, especially when neither the legacy nor more modern on-line transaction processing (OLTP) systems provide adequate operational reports. These reports are characterized by a limited set of fixed queries that can be hard-wired in a reporting application. The reports address the organization's more tactical decision-making requirements. Performance-enhancing aggregations, significant historical time series, and extensive descriptive attribution are specifically excluded from the ODS. The ODS as a reporting instance may be a stepping-stone to feed operational data into the warehouse.

In other cases, ODSs are built to support real-time interactions, especially in customer relationship management (CRM) applications such as accessing your travel itinerary on a Web site or your service history when you call into customer support. The traditional data warehouse typically is not in a position to support the demand for near-real-time data or immediate response times. Similar to the operational reporting scenario, data inquiries to support these real-time interactions have a fixed structure. Interestingly, this type of ODS sometimes leverages information from the data warehouse, such as a customer service call center application that uses customer behavioral information from the data warehouse to precalculate propensity scores and store them in the ODS.

In either scenario, the ODS can be either a third physical system sitting between the operational systems and the data warehouse or a specially administered hot partition of the data warehouse itself. Every organization obviously needs

operational systems. Likewise, every organization would benefit from a data warehouse. The same cannot be said about a physically distinct ODS unless the other two systems cannot answer your immediate operational questions. Clearly, you shouldn't allocate resources to construct a third physical system unless your business needs cannot be supported by either the operational data-collection system or the data warehouse. For these reasons, we believe that the trend in data warehouse design is to deliver the ODS as a specially administered portion of the conventional data warehouse. We will further discuss hot-partition-style ODSs in Chapter 5.

Finally, before we leave this topic, some have defined the ODS to mean the place in the data warehouse where we store granular atomic data. We believe that this detailed data should be considered a natural part of the data warehouse's presentation area and not a separate entity. Beginning in Chapter 2, we will show how the lowest-level transactions in a business are the foundation for the presentation area of the data warehouse.

Dimensional Modeling Vocabulary

Throughout this book we will refer repeatedly to fact and dimension tables. Contrary to popular folklore, Ralph Kimball didn't invent this terminology. As best as we can determine, the terms *dimensions* and *facts* originated from a joint research project conducted by General Mills and Dartmouth University in the 1960s. In the 1970s, both AC Nielsen and IRI used these terms consistently to describe their syndicated data offerings, which could be described accurately today as dimensional data marts for retail sales data. Long before simplicity was a lifestyle trend, the early database syndicators gravitated to these concepts for simplifying the presentation of analytic information. They understood that a database wouldn't be used unless it was packaged simply.

It is probably accurate to say that a single person did not invent the dimensional approach. It is an irresistible force in the design of databases that will always result when the designer places understandability and performance as the highest goals.

Fact Table

A fact table is the primary table in a dimensional model where the numerical performance measurements of the business are stored, as illustrated in Figure 1.2. We strive to store the measurement data resulting from a business process in a single data mart. Since measurement data is overwhelmingly the largest part of any data mart, we avoid duplicating it in multiple places around the enterprise.

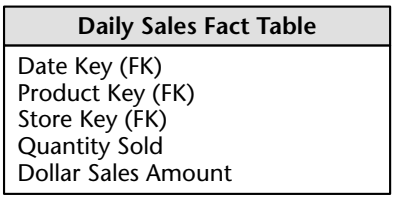


Figure 1.2 Sample fact table.

We use the term *fact* to represent a business measure. We can imagine standing in the marketplace watching products being sold and writing down the quantity sold and dollar sales amount each day for each product in each store. A measurement is taken at the intersection of all the dimensions (day, product, and store). This list of dimensions defines the *grain* of the fact table and tells us what the scope of the measurement is.

A row in a fact table corresponds to a measurement. A measurement is a row in a fact table. All the measurements in a fact table must be at the same grain.

The most useful facts are numeric and additive, such as dollar sales amount. Throughout this book we will use dollars as the standard currency to make the case study examples more tangible—please bear with the authors and substitute your own local currency if it doesn't happen to be dollars.

Additivity is crucial because data warehouse applications almost never retrieve a single fact table row. Rather, they bring back hundreds, thousands, or even millions of fact rows at a time, and the most useful thing to do with so many rows is to add them up. In Figure 1.2, no matter what slice of the database the user chooses, we can add up the quantities and dollars to a valid total. We will see later in this book that there are facts that are semiadditive and still others that are nonadditive. Semiadditive facts can be added only along some of the dimensions, and nonadditive facts simply can't be added at all. With nonadditive facts we are forced to use counts or averages if we wish to summarize the rows or are reduced to printing out the fact rows one at a time. This would be a dull exercise in a fact table with a billion rows.

The most useful facts in a fact table are numeric and additive.

We often describe facts as continuously valued mainly as a guide for the designer to help sort out what is a fact versus a dimension attribute. The dollar sales amount fact is continuously valued in this example because it can take on virtually any value within a broad range. As observers, we have to stand

out in the marketplace and wait for the measurement before we have any idea what the value will be.

It is theoretically possible for a measured fact to be textual; however, the condition arises rarely. In most cases, a textual measurement is a description of something and is drawn from a discrete list of values. The designer should make every effort to put textual measures into dimensions because they can be correlated more effectively with the other textual dimension attributes and will consume much less space. We do not store redundant textual information in fact tables. Unless the text is unique for every row in the fact table, it belongs in the dimension table. A true text fact is rare in a data warehouse because the unpredictable content of a text fact, like a free text comment, makes it nearly impossible to analyze.

In our sample fact table (see Figure 1.2), if there is no sales activity on a given day in a given store for a given product, we leave the row out of the table. It is very important that we do not try to fill the fact table with zeros representing nothing happening because these zeros would overwhelm most of our fact tables. By only including true activity, fact tables tend to be quite sparse. Despite their sparsity, fact tables usually make up 90 percent or more of the total space consumed by a dimensional database. Fact tables tend to be deep in terms of the number of rows but narrow in terms of the number of columns. Given their size, we are judicious about fact table space utilization.

As we develop the examples in this book, we will see that all fact table grains fall into one of three categories: transaction, periodic snapshot, and accumulating snapshot. Transaction grain fact tables are among the most common. We will introduce transaction fact tables in Chapter 2, periodic snapshots in Chapter 3, and accumulating snapshots in Chapter 5.

All fact tables have two or more foreign keys, as designated by the FK notation in Figure 1.2, that connect to the dimension tables' primary keys. For example, the product key in the fact table always will match a specific product key in the product dimension table. When all the keys in the fact table match their respective primary keys correctly in the corresponding dimension tables, we say that the tables satisfy *referential integrity.* We access the fact table via the dimension tables joined to it.

The fact table itself generally has its own primary key made up of a subset of the foreign keys. This key is often called a *composite* or *concatenated key.* Every fact table in a dimensional model has a composite key, and conversely, every table that has a composite key is a fact table. Another way to say this is that in a dimensional model, every table that expresses a many-to-many relationship must be a fact table. All other tables are dimension tables.

Fact tables express the many-to-many relationships between dimensions in dimensional models.

Only a subset of the components in the fact table composite key typically is needed to guarantee row uniqueness. There are usually about a half dozen dimensions that have robust many-to-many relationships with each other and uniquely identify each row. Sometimes there are as few as two dimensions, such as the invoice number and the product key. Once this subset has been identified, the rest of the dimensions take on a single value in the context of the fact table row's primary key. In other words, they go along for the ride. In most cases, there is no advantage to introducing a unique ROWID key to serve as the primary key in the fact table. Doing so makes your fact table larger, while any index on this artificial ROWID primary key would be worthless. However, such a key may be required to placate the database management system, especially if you can legitimately, from a business perspective, load multiple identical rows into the fact table.

Dimension Tables

Dimension tables are integral companions to a fact table. The dimension tables contain the textual descriptors of the business, as illustrated in Figure 1.3. In a well-designed dimensional model, dimension tables have many columns or attributes. These attributes describe the rows in the dimension table. We strive to include as many meaningful textlike descriptions as possible. It is not uncommon for a dimension table to have 50 to 100 attributes. Dimension tables tend to be relatively shallow in terms of the number of rows (often far fewer than 1 million rows) but are wide with many large columns. Each dimension is defined by its single primary key, designated by the PK notation in Figure 1.3, which serves as the basis for referential integrity with any given fact table to which it is joined.

Dimension attributes serve as the primary source of query constraints, groupings, and report labels. In a query or report request, attributes are identified as the *by* words. For example, when a user states that he or she wants to see dollar sales by week by brand, week and brand must be available as dimension attributes.

Dimension table attributes play a vital role in the data warehouse. Since they are the source of virtually all interesting constraints and report labels, they are key to making the data warehouse usable and understandable. In many ways, the data warehouse is only as good as the dimension attributes. The power of the data warehouse is directly proportional to the quality and depth of the

dimension attributes. The more time spent providing attributes with verbose business terminology, the better the data warehouse is. The more time spent populating the values in an attribute column, the better the data warehouse is. The more time spent ensuring the quality of the values in an attribute column, the better the data warehouse is.

Dimension tables are the entry points into the fact table. Robust dimension attributes deliver robust analytic slicing and dicing capabilities. The dimensions implement the user interface to the data warehouse.

The best attributes are textual and discrete. Attributes should consist of real words rather than cryptic abbreviations. Typical attributes for a product dimension would include a short description (10 to 15 characters), a long description (30 to 50 characters), a brand name, a category name, packaging type, size, and numerous other product characteristics. Although the size is probably numeric, it is still a dimension attribute because it behaves more like a textual description than like a numeric measurement. Size is a discrete and constant descriptor of a specific product.

Sometimes when we are designing a database it is unclear whether a numeric data field extracted from a production data source is a fact or dimension attribute. We often can make the decision by asking whether the field is a measurement that takes on lots of values and participates in calculations (making it a fact) or is a discretely valued description that is more or less constant and participates in constraints (making it a dimensional attribute). For example, the standard cost for a product seems like a constant attribute of the product but may be changed so often that eventually we decide that it is more like a measured fact. Occasionally, we can't be certain of the classification. In such cases, it may be possible to model the data field either way, as a matter of designer's prerogative.

Product Dimension Table
Product Key (PK)
Product Description
SKU Number (Natural Key)
Brand Description
Category Description
Department Description
Package Type Description
Package Size
Fat Content Description
Diet Type Description
Weight
Weight Units of Measure
Storage Type
Shelf Life Type
Shelf Width
Shelf Height
Shelf Depth
... and many more

Figure 1.3 Sample dimension table.

We strive to minimize the use of codes in our dimension tables by replacing them with more verbose textual attributes. We understand that you may have already trained the users to make sense of operational codes, but going forward, we'd like to minimize their reliance on miniature notes attached to their computer monitor for code translations. We want to have standard decodes for the operational codes available as dimension attributes so that the labeling on data warehouse queries and reports is consistent. We don't want to encourage decodes buried in our reporting applications, where inconsistency is inevitable. Sometimes operational codes or identifiers have legitimate business significance to users or are required to communicate back to the operational world. In these cases, the codes should appear as explicit dimension attributes, in addition to the corresponding user-friendly textual descriptors. We have identified operational, natural keys in the dimension figures, as appropriate, throughout this book.

Operational codes often have intelligence embedded in them. For example, the first two digits may identify the line of business, whereas the next two digits may identify the global region. Rather than forcing users to interrogate or filter on the operational code, we pull out the embedded meanings and present them to users as separate dimension attributes that can be filtered, grouped, or reported on easily.

Dimension tables often represent hierarchical relationships in the business. In our sample product dimension table, products roll up into brands and then into categories. For each row in the product dimension, we store the brand and category description associated with each product. We realize that the hierarchical descriptive information is stored redundantly, but we do so in the spirit of ease of use and query performance. We resist our natural urge to store only the brand code in the product dimension and create a separate brand lookup table. This would be called a *snowflake.* Dimension tables typically are highly denormalized. They are usually quite small (less than 10 percent of the total data storage requirements). Since dimension tables typically are geometrically smaller than fact tables, improving storage efficiency by normalizing or snowflaking has virtually no impact on the overall database size. We almost always trade off dimension table space for simplicity and accessibility.

Bringing Together Facts and Dimensions

Now that we understand fact and dimension tables, let's bring the two building blocks together in a dimensional model. As illustrated in Figure 1.4, the fact table consisting of numeric measurements is joined to a set of dimension tables filled with descriptive attributes. This characteristic starlike structure is often called a *star join schema.* This term dates back to the earliest days of relational databases.

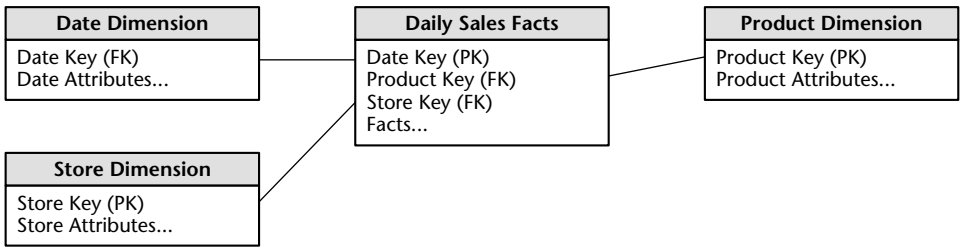


Figure 1.4 Fact and dimension tables in a dimensional model.

The first thing we notice about the resulting dimensional schema is its simplicity and symmetry. Obviously, business users benefit from the simplicity because the data is easier to understand and navigate. The charm of the design in Figure 1.4 is that it is highly recognizable to business users. We have observed literally hundreds of instances where users agree immediately that the dimensional model *is* their business. Furthermore, the reduced number of tables and use of meaningful business descriptors make it less likely that mistakes will occur.

The simplicity of a dimensional model also has performance benefits. Database optimizers will process these simple schemas more efficiently with fewer joins. A database engine can make very strong assumptions about first constraining the heavily indexed dimension tables, and then attacking the fact table all at once with the Cartesian product of the dimension table keys satisfying the user's constraints. Amazingly, using this approach it is possible to evaluate arbitrary *n*-way joins to a fact table in a single pass through the fact table's index.

Finally, dimensional models are gracefully extensible to accommodate change. The predictable framework of a dimensional model withstands unexpected changes in user behavior. Every dimension is equivalent; all dimensions are symmetrically equal entry points into the fact table. The logical model has no built-in bias regarding expected query patterns. There are no preferences for the business questions we'll ask this month versus the questions we'll ask next month. We certainly don't want to adjust our schemas if business users come up with new ways to analyze the business.

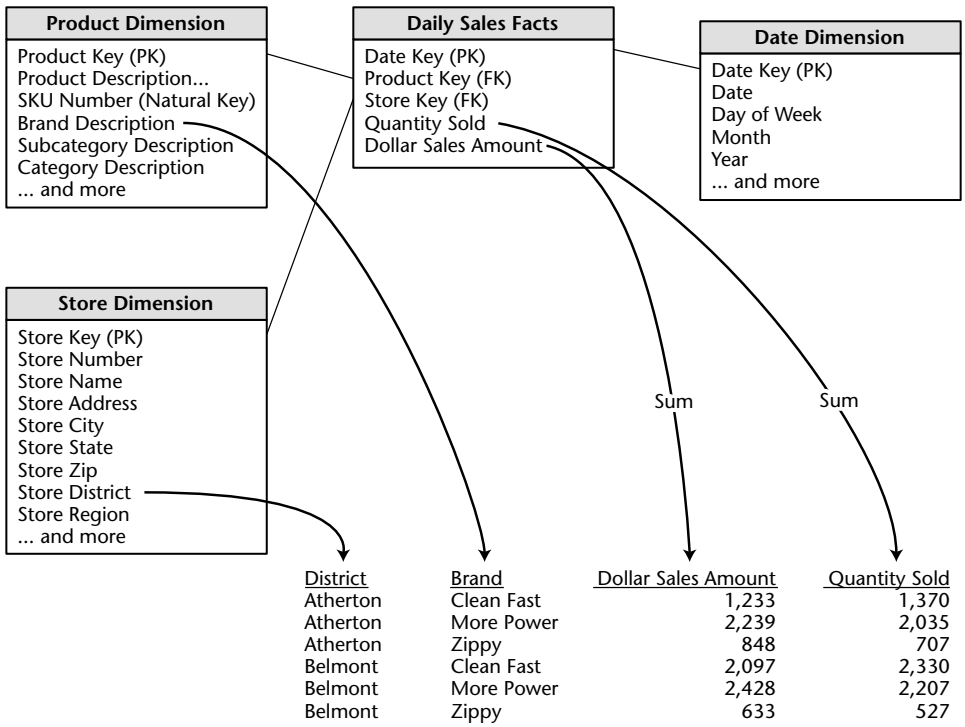
We will see repeatedly in this book that the most granular or atomic data has the most dimensionality. Atomic data that has not been aggregated is the

most expressive data; this atomic data should be the foundation for every fact table design in order to withstand business users' ad hoc attacks where they pose unexpected queries. With dimensional models, we can add completely new dimensions to the schema as long as a single value of that dimension is defined for each existing fact row. Likewise, we can add new, unanticipated facts to the fact table, assuming that the level of detail is consistent with the existing fact table. We can supplement preexisting dimension tables with new, unanticipated attributes. We also can break existing dimension rows down to a lower level of granularity from a certain point in time forward. In each case, existing tables can be changed in place either simply by adding new data rows in the table or by executing an SQL ALTER TABLE command. Data would not have to be reloaded. All existing data access applications continue to run without yielding different results. We'll examine this graceful extensibility of our dimensional models more fully in Chapter 2.

Another way to think about the complementary nature of fact and dimension tables is to see them translated into a report. As illustrated in Figure 1.5, dimension attributes supply the report labeling, whereas the fact tables supply the report's numeric values.

Finally, as we've already stressed, we insist that the data in the presentation area be dimensionally structured. However, there is a natural relationship between dimensional and normalized models. The key to understanding the relationship is that a single normalized ER diagram often breaks down into multiple dimensional schemas. A large normalized model for an organization may have sales calls, orders, shipment invoices, customer payments, and product returns all on the same diagram. In a way, the normalized ER diagram does itself a disservice by representing, on a single drawing, multiple business processes that never coexist in a single data set at a single point in time. No wonder the normalized model seems complex.

If you already have an existing normalized ER diagram, the first step in converting it into a set of dimensional models is to separate the ER diagram into its discrete business processes and then model each one separately. The second step is to select those many-to-many relationships in the ER diagrams that contain numeric and additive nonkey facts and designate them as fact tables. The final step is to denormalize all the remaining tables into flat tables with single-part keys that join directly to the fact tables. These tables become the dimension tables.



District	Brand	Dollar Sales Amount	Quantity Sold
Atherton	Clean Fast	1,233	1,370
Atherton	More Power	2,239	2,035
Atherton	Zippy	848	707
Belmont	Clean Fast	2,097	2,330
Belmont	More Power	2,428	2,207
Belmont	Zippy	633	527

Figure 1.5 Dragging and dropping dimensional attributes and facts into a simple report.

Dimensional Modeling Myths

Despite the general acceptance of dimensional modeling, some misperceptions continue to be disseminated in the industry. We refer to these misconceptions as *dimensional modeling myths*.

Myth 1. *Dimensional models and data marts are for summary data only*. This first myth is the root cause of many ill-designed dimensional models. Because we can't possibly predict all the questions asked by business users, we need to provide them with queryable access to the most detailed data so that they can roll it up based on the business question at hand. Data at the lowest level of detail is practically impervious to surprises or changes. Our data marts also will include commonly requested summarized data in dimensional schemas. This summary data should complement the granular detail solely to provide improved performance for common queries, but not attempt to serve as a replacement for the details.

A related corollary to this first myth is that only a limited amount of historical data should be stored in dimensional structures. There is nothing

about a dimensional model that prohibits the storage of substantial history. The amount of history available in data marts must be driven by the business's requirements.

Myth 2. *Dimensional models and data marts are departmental, not enterprise, solutions*. Rather than drawing boundaries based on organizational departments, we maintain that data marts should be organized around business processes, such as orders, invoices, and service calls. Multiple business functions often want to analyze the same metrics resulting from a single business process. We strive to avoid duplicating the core measurements in multiple databases around the organization.

Supporters of the normalized data warehouse approach sometimes draw spiderweb diagrams with multiple extracts from the same source feeding into multiple data marts. The illustration supposedly depicts the perils of proceeding without a normalized data warehouse to feed the data marts. These supporters caution about increased costs and potential inconsistencies as changes in the source system of record would need to be rippled to each mart's ETL process.

This argument falls apart because no one advocates multiple extracts from the same source. The spiderweb diagrams fail to appreciate that the data marts are process-centric, not department-centric, and that the data is extracted once from the operational source and presented in a single place. Clearly, the operational system support folks would frown on the multiple-extract approach. So do we.

Myth 3. *Dimensional models and data marts are not scalable*. Modern fact tables have many billions of rows in them. The dimensional models within our data marts are extremely scalable. Relational DBMS vendors have embraced data warehousing and incorporated numerous capabilities into their products to optimize the scalability and performance of dimensional models.

A corollary to myth 3 is that dimensional models are only appropriate for retail or sales data. This notion is rooted in the historical origins of dimensional modeling but not in its current-day reality. Dimensional modeling has been applied to virtually every industry, including banking, insurance, brokerage, telephone, newspaper, oil and gas, government, manufacturing, travel, gaming, health care, education, and many more. In this book we use the retail industry to illustrate several early concepts mainly because it is an industry to which we have all been exposed; however, these concepts are extremely transferable to other businesses.

Myth 4. *Dimensional models and data marts are only appropriate when there is a predictable usage pattern*. A related corollary is that dimensional models aren't responsive to changing business needs. On the contrary, because of

their symmetry, the dimensional structures in our data marts are extremely flexible and adaptive to change. The secret to query flexibility is building the fact tables at the most granular level. In our opinion, the source of myth 4 is the designer struggling with fact tables that have been prematurely aggregated based on the designer's unfortunate belief in myth 1 regarding summary data. Dimensional models that only deliver summary data are bound to be problematic. Users run into analytic brick walls when they try to drill down into details not available in the summary tables. Developers also run into brick walls because they can't easily accommodate new dimensions, attributes, or facts with these prematurely summarized tables. The correct starting point for your dimensional models is to express data at the lowest detail possible for maximum flexibility and extensibility.

Myth 5. *Dimensional models and data marts can't be integrated and therefore lead to stovepipe solutions*. Dimensional models and data marts most certainly can be integrated if they conform to the data warehouse bus architecture. Presentation area databases that don't adhere to the data warehouse bus architecture will lead to standalone solutions. You can't hold dimensional modeling responsible for the failure of some organizations to embrace one of its fundamental tenets.

Common Pitfalls to Avoid

While we can provide positive recommendations about dimensional data warehousing, some readers better relate to a listing of common pitfalls or traps into which others have already stepped. Borrowing from a popular late-night television show, here is our favorite top 10 list of common errors to avoid while building your data warehouse. These are all quite lethal errors—one alone may be sufficient to bring down your data warehouse initiative. We'll further elaborate on these in Chapter 16; however, we wanted to plant the seeds early on while we have your complete attention.

Pitfall 10. Become overly enamored with technology and data rather than focusing on the business's requirements and goals.

Pitfall 9. Fail to embrace or recruit an influential, accessible, and reasonable management visionary as the business sponsor of the data warehouse.

Pitfall 8. Tackle a galactic multiyear project rather than pursuing more manageable, while still compelling, iterative development efforts.

Pitfall 7. Allocate energy to construct a normalized data structure, yet run out of budget before building a viable presentation area based on dimensional models.

Pitfall 6. Pay more attention to backroom operational performance and ease of development than to front-room query performance and ease of use.

Pitfall 5. Make the supposedly queryable data in the presentation area overly complex. Database designers who prefer a more complex presentation should spend a year supporting business users; they'd develop a much better appreciation for the need to seek simpler solutions.

Pitfall 4. Populate dimensional models on a standalone basis without regard to a data architecture that ties them together using shared, conformed dimensions.

Pitfall 3. Load only summarized data into the presentation area's dimensional structures.

Pitfall 2. Presume that the business, its requirements and analytics, and the underlying data and the supporting technology are static.

Pitfall 1. Neglect to acknowledge that data warehouse success is tied directly to user acceptance. If the users haven't accepted the data warehouse as a foundation for improved decision making, then your efforts have been exercises in futility.

Summary

In this chapter we discussed the overriding goals for the data warehouse and the differences between data warehouses and operational source systems. We explored the major components of the data warehouse and discussed the permissible role of normalized ER models in the staging area, but not as the end goal. We then focused our attention on dimensional modeling for the presentation area and established preliminary vocabulary regarding facts and dimensions. Stay tuned as we put these concepts into action in our first case study in the next chapter.