Chapter 6: Introducing Dimensional Modeling

Overview

Beginning with this chapter, you embark on the data track of the Kimball Lifecycle. The majority of the effort required to build the data warehouse/ business intelligence (DW/BI) system is expended in this track of activities, so it's critical that you get started on the right foot with a properly designed data model that addresses the business requirements.

The authors of this book have spent most of their careers, going back to the early 1980s, designing and using databases that support the business's decision making process. From this collective experience, we have concluded that dimensional modeling is the most viable technique to deliver data for business intelligence because it addresses the twin non-negotiable goals of business understandability and fast query performance. Fortunately, many others in our industry have been similarly convinced over the years so we're no longer repeatedly debating this fundamental matter as we did in the 1990s.

This chapter begins with a brief discussion comparing dimensional models to normalized models. We then provide a primer of core dimensional modeling concepts, followed by a discussion of the enterprise data warehouse bus architecture for ensuring consistency and integrating your organization's dimensional models. We then delve deeper into the common patterns you'll likely encounter in your source data and the corresponding dimensional response. Finally, we describe misleading myths and misperceptions about dimensional modeling that unfortunately continue to circulate in the industry. Chapter 6 provides the conceptual foundation of core dimensional modeling techniques, which is subsequently leveraged in Chapter 7 with its discussion of the dimensional modeling design process and associated tasks.

This introduction to dimensional modeling is a must-read for the DW/BI team's data architects and dimensional modelers. Given the critical role played by the dimensional model, other team members such as the business analysts, ETL architects and developers, and BI application architects and developers should minimally read the first half of this chapter through the bus architecture discussion.

REFERENCE Obviously, we can't synthesize everything there is to know about dimensional modeling into a single chapter. We strongly suggest that the modeling team has access to *The Data Warehouse Toolkit, Second Edition* (Wiley Publishing, 2002) by Ralph Kimball and Margy Ross as a reference. That book includes more detailed guidance using examples from a variety of industries, such as retail, financial services, telecommunications, education, and healthcare/insurance, as well as diverse business functions, including inventory, procurement, customer relationship management, accounting, and human resources. It's everything you wanted to know about dimensional modeling, but were afraid to ask.

You can also search the articles and Design Tips available on our website at http://www.kimballgroup.com for additional dimensional modeling recommendations.

Making the Case for Dimensional Modeling

Before diving into specific guidance for designing dimensional models, we begin with a comparison to the normalized models typically encountered in the transaction source systems.

What Is Dimensional Modeling?

Dimensional modeling is a logical design technique for structuring data so that it's intuitive to business users and delivers fast query performance.

Dimensional modeling is widely accepted as the preferred approach for DW/BI presentation. Practitioners and pundits alike have recognized that the data presented to the business intelligence tools must be grounded in simplicity to stand any chance of success. Simplicity is a fundamental requirement because it ensures that users can easily understand databases, as well as allows software to efficiently navigate databases. In case after case, IT organizations, consultants, users, and vendors have gravitated to a simple dimensional structure to match the basic human need for simplicity.

Note It is probably accurate to say that the dimensional approach was not invented by any single person. It is an irresistible force in the design of databases that always surfaces when designers place understandability and performance as the highest goals.

Dimensional modeling divides the world into *measurements* and *context*. Measurements are captured by the organization's business processes and their supporting operational source systems. Measurements are usually numeric values; we refer to them as *facts*.

Facts are surrounded by largely textual context that is true at the moment the fact is recorded. This context is intuitively divided into independent logical clumps called *dimensions*. Dimensions describe the "who, what, when, where, why, and how" context of the measurement.

Each of the organization's business processes can be represented by a dimensional model that consists of a fact table containing the numeric measurements surrounded by a halo of dimension tables containing the textual context, as illustrated in Figure 6-1. This characteristic star-like structure is often called a *star join*, a term dating back to the earliest days of relational databases.

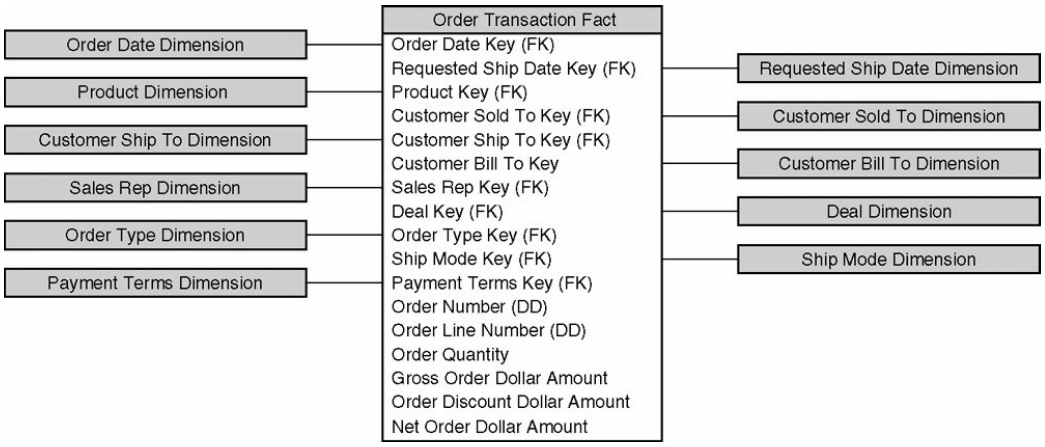


Figure 6-1: Dimensional model of the orders business process for a manufacturer

Dimensional models stored in a relational database platform are typically referred to as *star schemas*; dimensional models stored in multidimensional online analytical processing (OLAP) structures are called *cubes*.In Chapter 8, we recommend that OLAP cubes be populated from dimensional star schemas in almost all cases.

What about Normalized Modeling?

Third normal form (3NF) modeling is quite different from dimensional modeling. 3NF modeling is a design technique that seeks to eliminate data redundancies. Data is divided into many discrete entities, each of which becomes a table in the relational database. This normalization is immensely beneficial for transaction processing because it makes transaction loading and updating simple and fast.

Because normalized modeling captures the micro-relationships among data elements, even the simple orders business process results in dozens of tables that are linked together by a bewildering spider web of joins. We are all familiar with the oversized charts on the walls of database designers' cubicles. The normalized model for the enterprise has hundreds of logical entities. Enterprise resource planning (ERP) systems typically have thousands of entities. Each of these entities usually turns into a separate physical table when the database is implemented.

Note Dimensional and normalized models look strikingly different. Some designers react to this by saying that there must be less information in the dimensional model. It's the same information and the same data relationships; it's just structured differently.

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 database tables. Both 3NF and dimensional models can be represented in entity-relationship diagrams because both consist of joined relational tables; the key difference between 3NF and dimensional models is the degree of normalization. Because 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.

Note This already murky vocabulary is further muddied by people who use the term "relational modeling" to describe 3NF data models. Of course, dimensional modeling and 3NF modeling both result in physical tables substantiated in a relational database system. The resulting table structures differ, but all have the same relational database properties, regardless of the modeling technique.

It's worth noting that converting from normalized structures to dimensional models is relatively straightforward. The first step is to designate the many-to-many relationships in the normalized model containing numeric and additive non-key metrics as fact tables; fact tables are typically normalized to 3NF in a dimensional model because the related context is removed to dimension tables. The second step is to denormalize the remaining tables into flat dimension tables with single-part keys that connect directly to the fact table; dimension tables most often resemble second normal form tables with many low cardinality descriptors.

Note Interestingly, the dimensional modeling approach may predate normalized modeling. As best we can determine, in the late 1960s, a joint research project conducted by General Mills and Dartmouth University developed vocabulary consisting of "facts" and "dimensions." We believe this allowed Nielsen Marketing Research to carry these techniques forward with grocery and drug store audit data in the 1970s and later with scanner data in the 1980s. Several of this book's authors first became aware of these ideas from Nielsen in 1984.

Benefits of Dimensional Modeling

Understandability is one of the primary reasons that dimensional modeling is the widely accepted best practice for structuring the data presented to the business intelligence layer. Albert Einstein captured the design principles behind dimensional modeling when he said, "Make everything as simple as possible, but not simpler." The dimensional model is easier for a business user to understand than the typical source system normalized model because information is grouped into coherent business categories or dimensions that make sense to business people. The business categories help users navigate the model because entire categories can be disregarded if they aren't relevant to a particular analysis. But "as simple as possible" does not mean the model is simplistic. The dimensional model must reflect the complexities of the business. If you oversimplify, the model loses valuable information that's critical to understanding the business and its performance.

Query performance is the second dominant driver for dimensional modeling. Denormalizing dimension hierarchies and decode lookups can have a significant impact on query performance. Most relational database optimizers are tuned for star joins. The predictable framework of a dimensional model allows the database to make strong assumptions about the data that aid in performance. The database engine leverages the star join by first constraining the dimension tables and gathering the keys satisfying the query filters, and then querying the fact table with the Cartesian product of the relevant dimension keys. That's a far cry from trying to optimize the querying of hundreds of interrelated tables in a large normalized model. And even though the overall suite of dimensional models in an enterprise is complex, the query processing remains predictable; performance is managed by querying each fact table separately.

Beyond ease-of-use and query performance, there are a number of additional benefits associated with dimensional modeling. One subtle advantage is that each dimension is an equivalent entry point into the fact table. This symmetry allows the dimensional model to withstand unexpected changes in query patterns. There is no predictable pattern to the blizzard of queries coming from every possible angle during an ad hoc attack. The dimensional model's symmetrical structure allows it to effectively handle whatever is thrown at it. In technical terms, this means that query optimization for star join databases is simple, predictable, and controllable.

Best-of-breed ETL and BI tool vendors have incorporated dimensional intelligence into their products, such as pre-built ETL wizards to facilitate standard dimensional capabilities like the handling of slowly changing dimensions. Most BI tools prefer an underlying dimensional model to best showcase their capabilities.

Finally, dimensional models are gracefully extensible to accommodate unexpected new data. Existing tables can be changed in place either by simply adding new data rows in the table or executing an SQL `ALTER TABLE` command; data should not have to be reloaded. Graceful extensibility means that that no query or BI application needs to be reprogrammed to accommodate the change. And finally, graceful extensibility means that all old queries and applications continue to run without yielding different results.

Note Periodically, there's distracting noise in the industry about the feasibility of bypassing dimensional modeling and data warehouse databases to simply query and analyze operational data directly. Vendors promise magical middleware that hides the complexity of the source system so that business users can theoretically query the operational systems of record, eliminating the costly and time consuming extract, transformation, and load processing. Though middleware may be able to mask the underlying data structures, it doesn't address the inherent problems surrounding the performance conflicts of interest. You don't want open-ended queries contending for the same resources that are attempting to quickly capture the transactions of the business. You may find middleware solutions are only capable of relatively light on-the-fly data transformations, leaving your data integration requirements dangling. Also, middleware does nothing to address the realities of relatively short-term data retention in the operational source systems. Finally, it cannot support the tracking of dimension attribute changes over time.

Dimensional Modeling Primer

Having made the case for dimensional modeling, let us further explore the standard vocabulary of this modeling discipline. As we mentioned earlier, dimensional modeling distinguishes between measurements and context, or facts and dimensions in the parlance of a dimensional modeler.

Fact Tables

Fact tables store the performance measurements generated by the organization's business activities or events. The term *fact* refers to each performance measure. You typically don't know the value of a fact in advance because it's variable; the fact's valuation occurs at the time of the measurement event, such as when an order is received, a shipment is sent, or a service problem is logged. You can imagine standing on the loading dock to observe the shipments process; each movement of product onto an outbound truck generates performance measures or facts, such as the shipment quantity. Facts are typically continuously valued, meaning they can take on virtually any value within a broad range.

Note Fact tables naturally correspond to business process measurement events. Some pundits insist that fact tables should be designed around common business questions and reports, but we strongly disagree.

Nearly every fact is numeric. The most useful facts are both numeric and additive. Additivity is important because BI applications seldom retrieve a single fact table row; queries typically select hundreds or thousands of fact rows at a time, and the only useful thing to do with so many rows is to add them up. Most metrics are fully additive. However, numerical point-in-time measures of intensity, such as account balances or inventory levels, are semi-additive because they can't be summed across time periods. Some numeric measures are completely non-additive, such as a unit price or a temperature, because these facts can't be added across any dimension. Since most textual context is drawn from a discrete list of domain values, such context should be stored in dimension tables, rather than as text facts.

While much attention is focused on conformed dimensions, facts also conform if their definitions are identical. Conformed facts are allowed to have the same standardized name in separate tables. If facts do not conform, then the different interpretations much be given different names.

Note Not every numeric value belongs in the fact table. For example, the square footage of a facility is a static descriptor that should be stored in the facility dimension, not the fact table. A good rule of thumb is to ask whether the attribute is discretely valued and used for filtering or labeling, in which case it belongs in the dimension table. If the attribute takes on lots of values and is used in calculations, then it should be handled as a fact. This rule of thumb addresses 99 percent of all numeric values you will encounter. The remaining percent can be modeled either as facts, or dimension attributes, or as both simultaneously.

Fact tables are huge, with millions or billions of rows, but they're efficient. Because fact tables often store more than 90 percent of the data in a dimensional model, we're careful to minimize redundant data in a fact table. Also, we strive to store the measurements resulting from a business process in a single fact table that's shared across the organization. Because measurement data is the most voluminous data, we purposely avoid duplicating the detailed metrics in multiple fact tables around the enterprise, as we further describe with the enterprise data warehouse bus architecture.

Fact Table Keys

Fact tables are characterized by a multipart key made up of foreign keys from the intersecting dimension tables involved in a business process. The multipart key means that fact tables always express a many-to-many relationship.

Every foreign key in the fact table must match to a unique primary key in the corresponding dimension table. This implies that the fact table's foreign keys should never be null because this would violate referential integrity. Sometimes a dimension value is missing from a perfectly legitimate measurement event, such as a purchase made by someone without a frequent shopper loyalty card. In this case, the fact table row should reference a special key in the customer dimension representing "Not a Frequent Shopper" to establish a proper foreign key-primary key relationship.

The primary key of the fact table is typically a subset of the dimension foreign keys. Sometimes the primary key uniquely identifying a row in the fact table consists of degenerate dimensions, as we discuss later in this chapter.

Fact Table Granularity

The fact table's *grain* is the business definition of the measurement event that produces the fact row. Declaring the grain means saying exactly what a fact table row represents by filling in the blank in the following phrase: A fact row is created when____occurs. While the grain determines the fact table's primary key, granularity itself is always expressed in business terms, in advance of looking at the foreign keys that may be present.

All the rows in a fact table should have a single uniform grain. The grain is determined by the physical realities of the data's source; its declaration becomes clear when you visualize the measurement process. Once the grain is established, the design can proceed by determining the dimensions that decorate the measurements at that level of detail. This logical sequence of first declaring the grain and then assembling the dimensional context is immensely helpful, and is a key characteristic of the dimensional design process. The grain is your anchor: you must include enough dimensional context in your design to implement the grain, and you must exclude dimensional context that violates the grain.

Fact tables should contain the lowest, most detailed atomic grain captured by a business process. There is tremendous power, flexibility, and resiliency in granular atomic fact data. It allows queries to ask the most precise questions possible. Even if users don't care about the particulars of a single transaction or transaction line, their "question of the moment" requires summarizing these details in unpredictable ways. Atomic data withstand assaults from unpredictable ad hoc queries because the details can be grouped "any which way." Granular atomic fact tables are also more impervious to changes; they can be gracefully extended by adding newly sourced facts, newly sourced dimension attributes, and by adding entirely new dimensions (assuming the grain is not altered by the new dimension).

Note Architect Mies van der Rohe is credited with saying, "God is in the details." Delivering dimensional models populated with the most detailed data possible ensures maximum flexibility and extensibility. Delivering anything less in your dimensional models undermines the foundation necessary for robust business intelligence.

Dimension Tables

In contrast to the stark and sleek qualities of fact tables consisting of just keys and numeric measurements, dimension tables are anything but stark and sleek; they're filled with big and bulky descriptive fields. The attributes in a dimension table serve two critical purposes: query constraining/filtering and query result set labeling. In many ways, the power of the data warehouse is proportional to the quality and depth of the dimension attributes; robust dimensions translate into robust querying and analysis capabilities.

Note You can listen for dimensions or their attributes as the "by" words (by year, by product, by region) that are used by business people when they're requesting a query or report.

Dimension attributes are typically textual fields, or they are discrete numbers that behave like text. The beauty of the dimensional model is that all the attributes are equivalent candidates to become filters or labels. Dimension attributes should be:

- Verbose (labels consisting of full words)
- Descriptive
- Complete (no missing values)
- Discretely valued (take on only one value for each row in the dimension table)
- Quality assured (no misspellings, impossible values, obsolete or orphaned values, or cosmetically different versions of the same attribute).

Tip Null values in dimension attribute columns can be confusing to business users and cause rude behavior in an SQL query, so we generally recommend a descriptive string, such as "Unknown" or "Not applicable" instead.

It's not uncommon for a dimension table to have dozens of attributes, although that's not always the case. Codes and abbreviations can be stored as dimension attributes; however it's advisable to include a corresponding descriptive field. In most instances, designers who insist that the business users prefer codes over textual fields haven't spent much time with the users. Many business intelligence tools can perform decoding within their semantic metadata layer; however we encourage you to store the descriptive decodes in the database rather than relying on the BI tool because often more than one tool is used by an organization. Broader consistency is ensured when the description is maintained once by the ETL system and physically stored in the table.

Note Operational codes stored as dimension attributes often have embedded meaning. For example, the first two characters might indicate a line of business, while the next two designate a region, and so on. In this case, you should establish separate verbose dimension attributes for each embedded component so they're readily available for filtering and labeling.

Almost inevitably, dimension tables represent hierarchical relationships. It's not at all unusual to resolve several many-to-one hierarchies within a single dimension. This is a natural consequence of denormalization. For example, if products roll up into brands and brands roll into categories, we would represent this hierarchical relationship within a single product dimension table that included attributes for the corresponding brand and category descriptions. We understand that this results in redundant storage of the brand and category description on each product row. Though we could save space by storing the brand and category decodes in separate dimensions, doing so would hinder both understandability and query performance. We further explore this argument later in this chapter when we discuss snowflaking.

Dimension tables consist of highly correlated clumps of attributes grouped to represent the key objects of a business, such as its products, customers, employees, or facilities. You can think of them as the nouns of the data warehouse. There is some degree of designer judgment and intuition about the decision to create separate or combined dimensions. For example, in a retail schema, you could attempt to combine the product dimension with the store dimension and make a single monolithic product-store dimension. Assume you have 1,000 products and 100 stores. If there was no meaningful correlation between product and store, and every product was sold in every store, then your combined product-store dimension would be the Cartesian product of the two original dimensions with 100,000 product-stores. Although this new combined dimension would contain exactly the same information as separate store and product dimensions, you would undoubtedly reject this design because the larger combined dimension would be unwieldy, potentially present performance problems, deliver no user interface advantages, and likely not seem logical to the business users.

Most dimensional models end up with somewhere between 8 and 15 dimension tables. Fact tables that store transactional details end up being the most dimensional, and those that capture period snapshots invariably have a fewer number of dimensions. Some industries, such as health care, are notoriously complicated with numerous many-to-many relationships that can't be resolved within a dimension table. In these cases, the fact table may have upwards of twenty dimension foreign keys; however, this should be the exception at the high end of the spectrum rather than the rule.

Dimension Table Keys

Whereas fact tables have a multipart key, dimension rows are uniquely identified by a single key field. We strongly suggest that the dimension tables' primary keys are simple integers assigned in sequence starting with 1, meaning that when we create a key for a new dimension record, we simply add 1 to the last value we have used. These surrogate keys are meaningless; they merely serve as join fields between the fact and dimension tables.

In most cases, a 4-byte integer makes a great surrogate key because it can represent 2^{32} values, or more than two billion positive integers, starting with 1. Two billion is enough for just about any reasonable dimension, including customer.

There are several advantages of using surrogate dimension keys instead of merely referencing the operational natural keys:

- **Performance.** Surrogate integer keys are tight and efficient, especially compared to the sometimes bulky alphanumeric natural keys used by the operational transaction systems. Compact surrogate keys translate into better performance due to more efficient joins, smaller indices, and more fact rows per block. A single field surrogate key allows much more efficient join processing than a multi-field key containing original source system values.

- **Buffer from operational key management practices.** In the operational systems, data is often retained for relatively short periods of time, such as 30 to 60 days. Due to their limited retention needs, operational keys may be reused or reassigned after a period of dormancy. Although this is not problematic for the transaction processing systems, it would wreak havoc in the data warehouse where data is retained for years. If you relied on the operational natural key in the warehouse, previously obsolete products or deceased customers might reappear as miraculously resurrected when their natural keys are reassigned in the future by the operational system.
- **Mapping to integrate disparate sources.** In some operational environments, the same entity is assigned different natural keys by different source systems. These disparate natural keys can be matched and then associated with a neutral surrogate via a mapping table in the ETL system to uniquely identify the linked natural keys.
- **Handle unknown or not applicable conditions.** As we mentioned earlier, the fact table's foreign keys should never be null. Using a surrogate key instead of the operational natural key allows us to easily assign a surrogate key value to a row representing these null conditions.
- **Track changes in dimension attribute values.** Although dimension tables are much more static than fact tables, they are still subject to change. As we discuss later in this chapter when we cover type 2 slowly changing dimensions, one of the most popular and compliant ways to capture dimension changes is to create a new dimension row representing the new attribute profile, and then assign the next available surrogate as its primary key.

Since we've focused on the benefits, it's only fair to acknowledge that there's also a cost to using surrogate keys. Assigning and managing dimension table surrogate keys and appropriately substituting them for the operational natural keys in fact rows puts a burden on the ETL system. Fortunately, the pipeline processing is virtually identical across dimension and fact tables, so it can be built once and then reused. Also, many ETL tool vendors deliver built-in capabilities to facilitate the surrogate key processing.

Conformed Dimensions

Standardized *conformed dimensions* are the goal for any well-architected data warehouse. Sometimes referred to as master dimensions or common reference dimensions, conformed dimensions are shared across the enterprise's data warehouse environment, joining to multiple fact tables representing various business processes.

Conformed dimensions come in two flavors. With the basic flavor, two conformed dimensions are absolutely identical, with the same keys, attribute names, attribute definitions, and domain values regardless of the fact table that they join to. For example, the product dimension referenced by the sales order fact table is identical to the product dimension joined to the inventory facts. This dimension table delivers the same content, interpretation, and presentation regardless of the business process involved.

Alternatively, dimensions conform when one dimension is a perfect subset of a more detailed, granular dimension table. In this case, the attributes that are common to both the detailed and shrunken subset dimension have the same attribute names, definitions, and domain values. For example, as illustrated in Figure 6-2, the sales fact table joins to a dimension table at the individual product level. But sales forecast facts are captured at the granularity of a brand rather than an individual product, so you need a shrunken brand dimension that conforms to the product dimension. Attributes that are common to both the detailed product dimension and shrunken brand dimension should be labeled, defined, and valued identically in both tables, even though the entire tables are not identical.

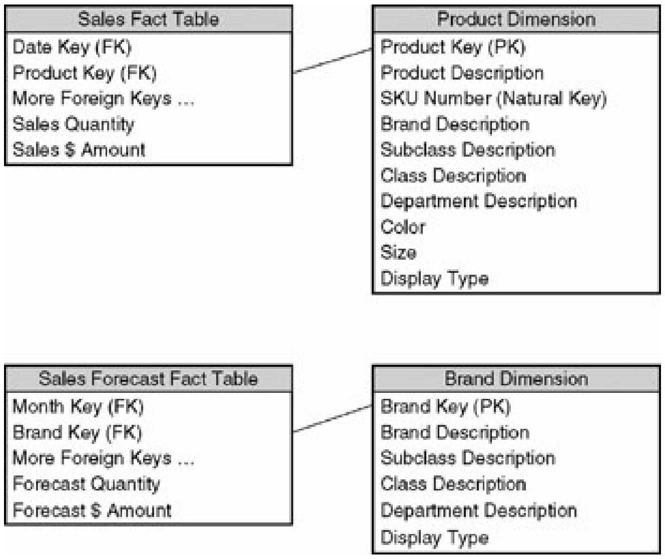


Figure 6-2: Conformed detailed and shrunken dimension tables corresponding to fact tables at different granularity

Note Shrunken conformed dimension tables are created to describe fact tables that either naturally capture measurements at a higher level of detail, or facts that have been aggregated to a less granular, rolled up level for performance reasons.

Conformed dimensions are enormously important to the enterprise nature of the DW/BI system because they deliver the following benefits:

- **Consistency.** Conformed dimensions ensure that every fact table is filtered consistently and the resulting query answer sets are labeled consistently.
- **Integration.** Strict adherence to conformed dimensions allows the DW/BI environment to operate as an integrated whole. Conformed dimensions allow queries to drill across fact tables representing different processes by issuing separate queries of each individual fact table, and then joining the result sets on common dimension attributes.
- **Reduced development time to market.** While there's an organizational investment required to define and manage conformed dimensions, once built, they can dramatically reduce the development time for a project because the common dimensions are available without recreating the wheel over and over again.

Recently, there's lots of industry interest in master data management. Conformed dimensions are the descriptive master data for the DW/BI environment. We elaborate on the role of data stewards for establishing these all-important conformed dimensions in Chapter 7; they play a critical role as organizational agreement on conformed dimensions faces more geo-political challenges than technical hurdles. And while conformed dimensions may be replicated either logically or physically throughout the enterprise, they should be built once by the ETL system as we further describe in Chapter 9.

Note Even in a highly diversified conglomerate business, there are typically a handful of core product or customer attributes that should be conformed across the enterprise, although it would be foolhardy to attempt to get folks in the disparate lines of business to agree on all the attributes describing their businesses. The procedures we describe for building and managing conformed dimensions give you the flexibility needed to reap the value of commonly defined attributes, without forcing the enterprise to give up "private" attributes.

Four-Step Dimensional Design Process

With this understanding of key vocabulary and core concepts, now it's important to describe our approach for bringing together the fact and dimension puzzle pieces. The logical design of a dimensional model is driven by four key decisions, as we further explore in Chapter 7.

Step 1 — Choose the Business Process

The first step in the design process is to determine the business process or measurement event to be modeled. As we briefly described in Chapter 3 and elaborate on in the next section, each row of the enterprise data warehouse bus matrix corresponds to a candidate business process identified while gathering the business requirements. This selection step likely occurred during the prioritization activity with senior business management.

Step 2 — Declare the Grain

Once the business process has been identified, the design team must declare the grain of the fact table. It is crucial to crisply define exactly what a fact table row is in the proposed business process dimensional model. Without agreement on the grain of the fact table, the design process cannot successfully move forward.

Note Be very precise when defining the fact table grain in business terms. Do not skip this step. The grain is the business answer to the question "What is a fact row, exactly?"

Generally, the fact table grain is chosen to be as atomic or finely grained as possible. Fact tables designed with the most granular data produce the most robust design. Atomic data is far better at responding to both unexpected new queries and unanticipated new data elements than higher levels of granularity.

Step 3 — Identify the Dimensions

Once the grain of the fact table is firmly established, the choice of dimensions is fairly straightforward. It is at this point you can start thinking about foreign keys. The grain itself will often determine a primary or minimal set of dimensions. From there, the design is embellished with additional dimensions that take on a unique value at the declared grain of the fact table.

Step 4 — Identify the Facts

The final step in the four-step design process is to carefully select the facts or metrics that are applicable to the business process. The facts may be physically captured by the measurement event or derived from these measurements. Each fact must be true to the grain of the fact table; do not mix facts from other time periods or other levels of detail that do not match the crisply declared grain.

As illustrated in Figure 6-3, before you drive stakes in the ground for each of these four decision points, it's important that you have a solid understanding of the needs of the business, along with an understanding of the realities of the available source data.

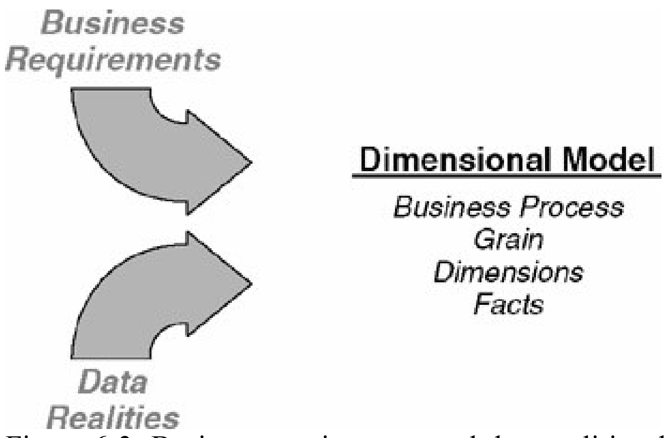


Figure 6-3: Business requirements and data realities drive the decisions made during the four-step process

Enterprise Data Warehouse Bus Architecture

Planning the construction of your overall DW/BI environment is a critical activity. Historically, the debate has focused on the alternatives of building the entire enterprise data warehouse from a centralized, planned perspective or building smaller, independent solutions for specific business units. Of course, neither of these approaches is effective. The galactic approach results in development efforts that take far too long to deliver business value, and thus lose business enthusiasm and ultimately falter. On the other hand, building independent departmental systems may lead to quick results, but will soon erode due to the proliferation of stovepipe, non-integrated, stand-alone solutions that fail to meet overall enterprise objectives.

Note Departmental, stand-alone solutions are often referred to as *data marts*. When we first started using the data mart terminology in the 1990s, we were describing process-centric detailed databases that represented a subset of the enterprise's overall data architecture — a far cry from a stand-alone point solution. However, the term was hijacked to refer to independent, non-architected, departmental databases. Given the disparity in definitions and lack of consistent understanding, we've migrated away from the data mart nomenclature.

Planning Crisis

The task of planning the enterprise DW/BI data architecture is daunting. The newly appointed program manager of the DW/BI effort in a large enterprise is faced with two huge and seemingly unrelated challenges. On the one hand, the manager is supposed to understand the content and location of the most complicated asset owned by the enterprise: the source data. Somehow the DW/BI manager is supposed to become an authority on exactly what is contained in all those legacy mainframe, ERP, web server, application server, and other business systems. Every data element in every system must be understood. The DW/BI manager must be able to retrieve any requested element of data and, if necessary, clean it up and correct it. On the other hand, the DW/BI manager is supposed to understand exactly what keeps management awake at night. The DW/BI system is expected to contain exactly the data needed to answer the burning questions du jour. Of course, the manager is free to drop in on senior management at any time to discuss current corporate priorities, as long as they can ensure that the DW/BI system is done soon.

To relieve this pressure, DW/BI managers take refuge in carving off a little piece of the required solution for a given department and bringing it to completion. Often the DW/BI manager is pleasantly surprised by the success of this effort. As all good managers will, the DW/BI manager replicates a successful process and continues with further point implementations.

Unfortunately, in many cases, building separate point solutions rather than a conformed enterprise data warehouse has become an excuse for ignoring any kind of design framework that might tie these independent systems together into a coherent whole. Isolated, independent stovepipe solutions that cannot usefully be tied together are the bane of DW/BI systems. They are much worse than a simple lost opportunity for analysis. Independent stovepipe systems are dead ends that continue to perpetuate incompatible views of the enterprise. Stovepipe solutions enshrine the reports that cannot be compared with one another. And stovepipe systems become legacy implementations in their own right, where, by their very existence, they block the development of an integrated enterprise data warehouse.

So if building the data warehouse all at once is too daunting and building it as isolated pieces defeats the overall goal, what is to be done?

Bus Architecture

The answer to this dilemma is to start with a quick and succinct effort that defines the overall enterprise DW/BI data architecture. In Chapter 3, we described an initial program-level requirements gathering process that results in the creation of the enterprise data warehouse bus matrix, shown in Figure 3.5. Each row of the matrix corresponds to a business process within the organization. Each column corresponds to a dimension of the business. The matrix cells are shaded to indicate which columns are related to each row.

The enterprise data warehouse bus matrix *is* the overall data architecture for the DW/BI system. The matrix delivers the big picture perspective, regardless of database or technology preferences, while also identifying reasonably manageable development efforts. Each business process implementation incrementally builds out the overall architecture. In this way, the DW/BI manager gets the best of both worlds. Multiple development teams can work on components of the matrix fairly independently and asynchronously, with confidence that they will fit together like the pieces of a puzzle. At some point, enough dimensional schemas are built to make good on the promise of an integrated enterprise data warehouse environment.

Developing the data architecture via the bus matrix is a rational approach to decomposing the daunting task of planning an enterprise data warehouse. It establishes an architectural framework that guides the overall design, but divides the problem into bite-sized implementation chunks. Build out your integrated, enterprise data warehouse iteratively, business process by business process, using a family of shared conformed dimensions to provide the required integration. These conformed dimensions have a uniform interpretation across the enterprise. Finally, you see the overall enterprise data warehouse for what it is: a collection of business processes bound together with a powerful architecture based on conformed dimensions.

Conformed dimensions are the "bus" of the enterprise data warehouse, as illustrated in Figure 6-4. The word *bus* is an old term from electrical power plants that is now used frequently in the computer industry. A bus is a common structure that everything connects to and derives power from. The bus in your computer is a standard interface that allows many different kinds of devices to connect. If all the devices conform to the standard interface definition imposed by the bus, then these devices can usefully coexist.

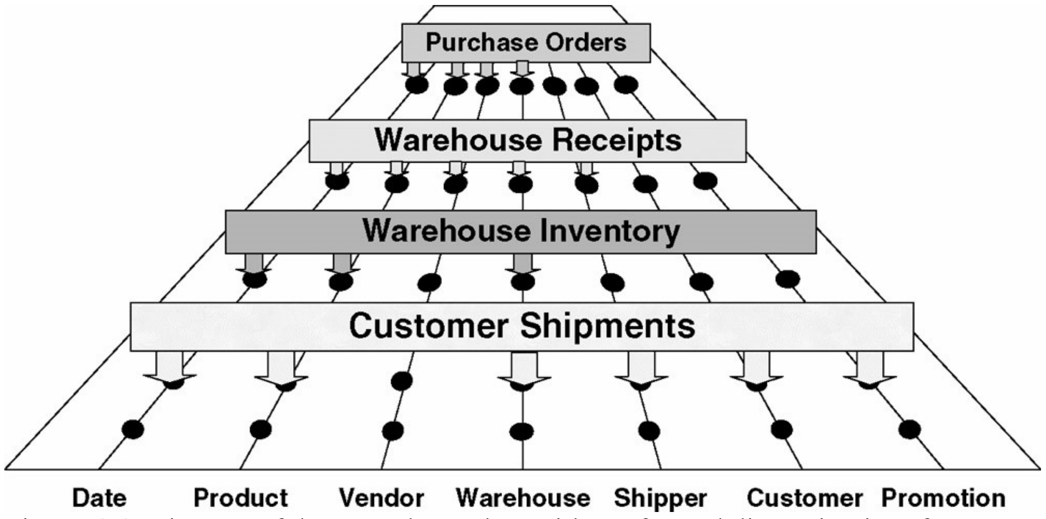


Figure 6-4: Diagram of data warehouse bus with conformed dimension interfaces

By defining a standard bus interface for the DW/BI environment, a new business process may be brought into the data warehouse where it can derive power from and usefully coexist with what is already there.

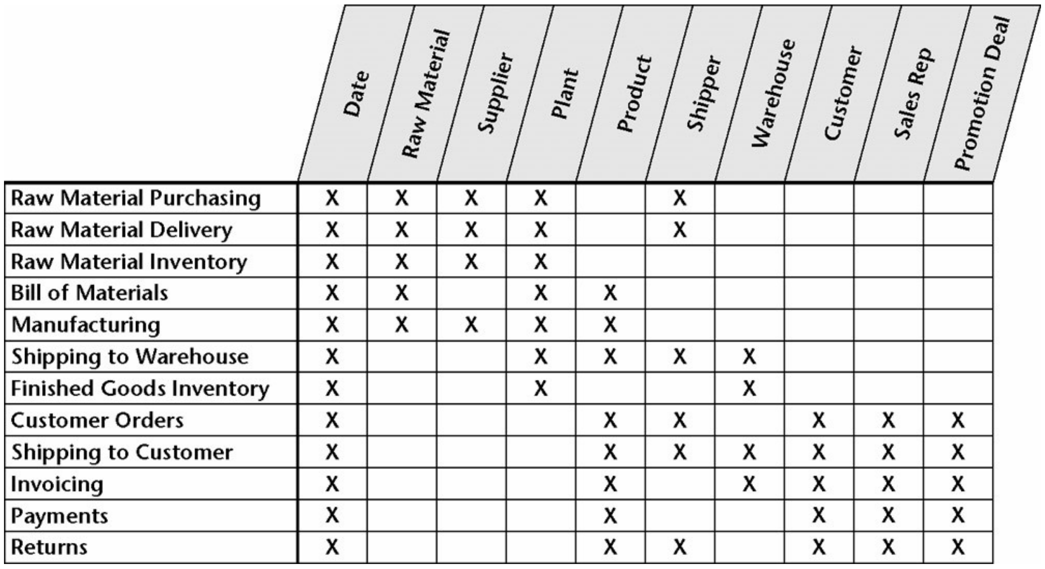
Value Chain Implications

Many businesses monitor a logical flow of internal activities through a series of business process steps. Each process spawns one or more fact tables because each step represents a unique measurement event.

In the manufacturing world, a product moves from the acquisition of raw materials through to finished goods and ultimately to customer delivery. Managing this flow from point-of-origin to point-of-consumption is often referred to as *supply chain management*, with a series of operational systems corresponding to the logical steps in the chain:

- Raw material purchasing
- Raw material delivery
- Raw material inventory
- Bill of materials
- Manufacturing
- Shipping to warehouse
- Finished goods inventory
- Customer orders
- Shipping to customers
- Invoicing
- Payments
- Returns

You can visualize this chain of events in a bus matrix as illustrated in Figure 6-5.



	Date	Raw Material	Supplier	Plant	Product	Shipper	Warehouse	Customer	Sales Rep	Promotion Deal
Raw Material Purchasing	X	X	X	X		X				
Raw Material Delivery	X	X	X	X		X				
Raw Material Inventory	X	X	X	X						
Bill of Materials	X	X		X	X					
Manufacturing	X	X	X	X	X					
Shipping to Warehouse	X			X	X	X	X			
Finished Goods Inventory	X			X			X			
Customer Orders	X				X	X		X	X	X
Shipping to Customer	X				X	X	X	X	X	X
Invoicing	X				X		X	X	X	X
Payments	X				X			X	X	X
Returns	X				X	X		X	X	X

Figure 6-5: Bus matrix for manufacturing supply chain

Though there is no physical product built or moved in the same sense as the manufacturing company, insurance companies have a similar value chain:

- Marketing
- Agent/broker sales
- Rating
- Underwriting
- Reinsuring
- Policy issuing
- Claims processing
- Claims investigation
- Claims payments
- Salvage
- Subrogation

These examples can be approached in a similar way. A fact table and its associated dimensions are defined for each step in the chain. The conformed dimensions are used as the bus framework for separately implementing each step in the chain. Dimensional models could be implemented in any sequence with the full confidence that the framework of conformed dimensions would always give guidance to the subsequent implementation efforts and guarantee that all the fact tables work together in the future.

Common Matrix Mishaps

When drafting a data warehouse bus matrix, people sometimes struggle with the level of detail expressed by each column or row. Matrix row mishaps commonly fall into the following two categories:

- **Departmental or overly encompassing rows.** The matrix rows shouldn't correspond to the boxes on your corporate organization chart, which represent functional groups, not business processes. Sure, some departments may be responsible for or acutely interested in a single business process, but the matrix rows shouldn't look like a list of direct reports to your CEO.
- **Report-centric or too narrowly defined rows.** At the opposite end of the spectrum, your bus matrix should not resemble a laundry list of requested reports. A single business process, such as shipping orders, often supports numerous analyses like customer ranking, sales rep performance, or product movement analysis. The matrix row should reference the business process, not the derivative reports or analytics.

When defining the matrix columns, architects naturally fall into the similar traps of defining columns that are either too broad or narrow:

- **Overly generalized columns.** A bus matrix "person" column might refer to a wide variety of people, from internal employees to external suppliers and customer contacts. Because there's virtually zero overlap between these populations, it adds confusion to lump them into a single generic dimension.
- **Separate columns for each level of a hierarchy.** The columns of the bus matrix should refer to dimensions at their most detailed level. Some business

process rows may require an aggregated version of the detailed dimension, like sales forecast metrics at the product group level.

Rather than creating separate matrix columns, such as product, product group, and line of business, for each level of the product hierarchy, we advocate a single column for product. As the cell is shaded to show participation with a business process row, you can denote the level of detail in the cell (if it's not at the most granular level). An even more extreme example of this mishap is to list each individual descriptive attribute as a separate column; this defeats the concept of dimensions and results in a completely unruly matrix.

Taking the Pledge

If the DW/BI team succeeds in establishing a set of master conformed dimensions for the enterprise, it is extremely important that the development teams actually use these dimensions. The commitment to use the conformed dimensions is more than a technical decision. It is a business policy decision that is critical to making the enterprise data warehouse function.

Note Because the creation of conformed dimensions is as much a political decision as it is a technical decision, the use of the conformed dimensions must be supported from the highest executive levels. This issue should become a sound bite for the enterprise CIO. Fortunately, the hype around master data management has elevated awareness of this issue and opportunity.

More on Dimensions

This section focuses on common patterns and challenges that a designer is likely to confront when modeling dimension tables.

Note Chapter 9 contains explicit advice for handling these common design concepts in the ETL system, such as the subsystem for managing slowly changing dimensions. Right now, we're focused on recognizing and modeling the patterns. We'll defer discussion about creating these tables and attributes until Chapter 9.

Date and Time

The date dimension occupies a special place in every data warehouse because virtually every fact table is a time series of observations; fact tables always have one or more date dimensions. Figure 6-6 illustrates a standard date dimension table at a daily grain that contains a number of useful attributes for navigating, filtering, and describing calendars. Only a handful of these attributes can be generated directly from an SQL date format field; holidays, workdays, fiscal periods, last day of month flags, and other groupings or filters must be embedded in the dimension. Remember that the primary reason for this detailed date table is to remove every last vestige of calendar knowledge from the BI applications. Calendar navigation should be driven through the date dimension table, not through hard coded application logic.

Date Dimension

- Date Key (PK)
- Date
- Full Date Description
- Day of Week
- Day Number in Calendar Month
- Day Number in Calendar Year
- Day Number in Fiscal Month
- Day Number in Fiscal Year
- Last Day in Week Indicator
- Last Day in Month Indicator
- Calendar Week Ending Date
- Calendar Week Number in Year
- Calendar Month
- Calendar Month Number in Year
- Calendar YYYY-MM
- Calendar Quarter
- Calendar Year-Quarter
- Calendar Year
- Fiscal Week
- Fiscal Week Number in Year
- Fiscal Month
- Fiscal Month Number in Year
- Fiscal Year-Month
- Fiscal Quarter
- Fiscal Year
- Holiday Indicator
- Weekday Indicator
- Workday Indicator
- SQL Date Stamp
- … and more

Figure 6-6: Example date dimension table

Surrogate Date Keys

Earlier in this chapter, we discussed the benefits of using surrogate keys for dimension tables. The mandate to use surrogate keys also applies to date dimensions. It is a mistake to use an SQL date format field as the join key between the date dimension table and fact table. First, a date format key is typically 8 bytes, so you are wasting 4 bytes in the fact table for every date key in every row. Second, a date format field can't easily accommodate special date dimension rows to identify an unknown date or future date. These special cases can be handled gracefully with a surrogate date key.

For most dimensions, the surrogate key should be totally meaningless. However, with the date dimension, we encourage you to assign surrogate sequence number keys in chronological order. Unlike most dimensions, there's no need to worry about new, totally unexpected dates.

Designers sometimes want to use an even more meaningful integer in the form of YYYYMMDD as the primary key of the date dimension table instead of a surrogate sequence number. If the motivation for using a YYYYMMDD key is to bypass the date dimension and filter directly on a recognizable field in the fact table, then this is a bad idea. If the reason behind the YYYYMMDD key is to establish and maintain date-based fact table partitions, as we discuss in Chapter 8 on performance tuning, then we're willing to bend the rules about intelligent keys, as long as you promise that all queries will continue to utilize the date dimension table.

Time of Day

Fine-scale tracking of individual transactions may introduce a meaningful date and timestamp into the data that is accurate to the minute or even the second. In this case, we do not create a combined date/time dimension with one row for every minute or second of every day. Because there are more than 31 million seconds in a year, this combined dimension would be enormous. Date analysis is usually quite distinct from the analysis of time periods within the day, so we separate these into two distinct dimensions in the fact table.

Time of day should be treated as a dimension only if there are meaningful textual descriptions for periods within the day, such as the lunch hour rush or third shift, or hierarchical groupings. This dimension table would have 1,440 rows if time of day was tracked to the minute or 86,400 rows if the grain was one row per second. If the business does not need textual descriptors for time slices within a day, then the time of day may be expressed as a simple non-additive fact or a date/timestamp.

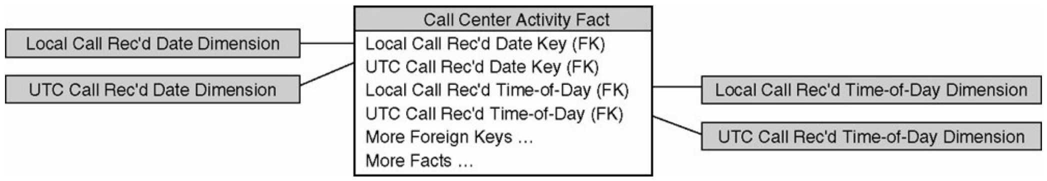
Date/Timestamps

In addition to the date dimension and potential time of day dimension, we sometimes store a full SQL date/timestamp in the fact table to support precise time interval calculations across fact rows.

Note If the business needs to calculate and analyze the lag between two specific date/timestamps, then these calculations should be performed in the ETL system and populated as facts for fast and simple direct user access. An example would be the elapsed time in insurance processing between the original claim date and first payment date, which is a key measure of claim processing efficiency.

Multiple Time Zones

For businesses spanning multiple time zones, the date and time of day may need to be expressed both in local time, as well as a global standard time such as Coordinated Universal Time (UTC). This is done with separate date and time dimensions for the local time zone and UTC, as shown in Figure 6-7. Merely providing time zone offsets in either a dimension or fact table isn't effective because of the number of possible time zones, the complexities of switching to and from daylight savings time, and the challenges of applying an offset that translates into the prior or following day on a UTC basis.


Local Call Rec'd Date Dimension
UTC Call Rec'd Date Dimension
Call Center Activity Fact
Local Call Rec'd Date Key (FK)
UTC Call Rec'd Date Key (FK)
Local Call Rec'd Time-of-Day (FK)
UTC Call Rec'd Time-of-Day (FK)
More Foreign Keys ...
More Facts ...
Local Call Rec'd Time-of-Day Dimension
UTC Call Rec'd Time-of-Day Dimension
Figure 6-7: Fact table with multiple time zones

Having two explicit time zone interpretations can be quite useful. For example, in analyzing call center behavior for multiple centers in different time zones, calling behavior can be aligned to real time by using the UTC interpretation and aligned on similar local times by using the local time dimensions. Of course, you may opt to standardize on a more familiar time zone, such as the time zone of the organization's headquarters, rather than relying on UTC.

Degenerate Dimensions

Many dimensional designs revolve around a transaction control document, such as an order, invoice, bill of lading, or airline ticket. Usually these control documents correspond to a transaction with one or more line items. Given this perspective, you can quickly visualize the contextual dimensions, but what do you do with the order number, invoice number, bill of lading number, or ticket number itself?

These numbers are the parent header keys used in the design of parent-child transaction systems. The answer is not to discard these header numbers; they should go directly into the fact table. The header ticket number looks like a dimension key (although we don't assign a surrogate key), but by the time you have triaged and decomposed the header record information into separate dimensions, like the transaction date and customer dimensions, the header number is sitting by itself without any remaining attributes. This is called a *degenerate dimension*.

Note Degenerate dimensions usually occur in transaction fact table designs where there is a natural parent-child data structure.

Degenerate dimensions are normal and useful. The degenerate dimension key should be the actual transaction number, sitting in the fact without joining to anything. There is no point in creating a corresponding dimension table because it would contain nothing but the transaction number anyway. The degenerate key is the glue that holds the line item rows together. It is required to answer broad questions, like what's the average number of line items on a transaction. The degenerate key can also be used as a link back to the operational systems for compliance, auditing, or researching data quality concerns. Finally, the degenerate dimension often serves as part of the fact table's primary key.

Slowly Changing Dimensions

By definition, dimension table attributes change relatively infrequently. Whereas the values in a fact table potentially change with every measurement event, dimension table attributes are more stable and static. However, many are still subject to change, albeit more slowly and unpredictably than the facts. The dimensional model needs to track time-variant dimension attributes as required by the business requirements. There are three fundamental techniques for handling slowly changing dimensions (SCDs), as well as a myriad of hybrid approaches.

Note Within a single dimension table, you may elect to utilize several different methods for tracking attribute changes, depending on the business needs.

Type 1: Overwrite the Dimension Attribute

With a type 1 slowly changing dimension response, when the attribute value changes, the old value is merely updated or overwritten with the most current value. The dimension attribute reflects the latest state, but any historical values are lost. Certainly, this technique is most appropriate for processing corrections. It is also used when the old value has no business significance, understanding that any historically accurate associations are lost with a type 1. In other words, type 1 does not preserve the attribute value that was effective at the time the historical fact row was created.

While the type 1 response appears simple to implement, there's an unexpected complication with type 1 attribute changes. If fact data have been previously *aggregated based on the type 1 attribute*, when the dimension value is overwritten, then any ad hoc summarization based on the new value will no longer tie to the pre-aggregated data. The aggregated data obviously needs to be updated to reflect the new attribute value so that the detail and aggregate data remain in sync.

Caution Don't immediately default to type 1 because of the perceived ease of implementation.

Type 2: Add a New Dimension Row

The type 2 slowly changing dimension response is the most popular and powerful technique for accurately tracking changes in attribute values.

With a type 2 change, a new row with a new surrogate primary key is inserted into the dimension table to reflect the new attribute values. Both the prior and new rows include the natural key (or durable identifier) as an attribute, along with a row effective date, row expiration date, and current row indicator, as illustrated in Figure 6-8.

PRODUCT KEY	Product Description	Product Code	Department	Effective Date	Expiration Date	Current Row Ind
12345	IntelliKidz 1.0	ABC999-Z	Education	2/15/2007	5/31/2007	Not current
25984	IntelliKidz 1.0	ABC999-Z	Strategy	6/1/2007	12/31/2007	Not current

34317	IntelliKidz 1.0	ABC999-Z	Critical Thinking	1/1/2008	12/31/9999	Current

Open table as spreadsheet

Figure 6-8: Sample rows from slowly changing dimension type 2

The type 2 response is used when a meaningful change to the dimension has taken place and it is appropriate to perfectly partition history by the changed attribute. Each surrogate key corresponds to a unique version of the dimension row that was true for a span of time. Thus each surrogate key is used in the corresponding fact rows during the time when the particular instance was valid. Obviously, type 2 is necessary if compliance is critical.

Note The type 2 approach requires the use of a surrogate key to uniquely identify the new profile. Some designers suggest that you uniquely identify changed dimension rows by concatenating an effective date to the natural key. We strongly discourage this practice. Joining this concatenated dimension key to the fact table would require a double barrel join which is highly inefficient, especially when the join on the effective date is a non-equal join.

Type 3: Add a New Dimension Attribute

The type 3 response to a slowly changing dimension attribute is used relatively infrequently. Type 3 is usually utilized when the change, like the redrawing of sales district boundaries or redefinition of product category boundaries, is a "soft" change. In other words, although the change has occurred, it is still logically possible to act as if the change had not occurred. For example, you may want to track sales performance with either the old or the new sales district definitions. In this case, you cannot partition history disjointly as in a type 2 response, but you simultaneously provide both the old attribute value and new value in the same dimension row. This allows users to choose between the two versions at will.

With a type 3 response, a new column is added to the dimension table. The old attribute value is pushed into a "prior" attribute column and the new attribute value is overwritten in the existing column. This causes all existing BI applications to switch seamlessly to the new attribute assignment, but allows a query or report to return to the old assignments by simply referencing the prior attribute column. With this technique, no new dimension rows are created and no new surrogate keys are created. It should be obvious that this technique works best for a limited number of such soft changes.

If a dimension attribute changes with a predictable rhythm, such as annually, and the business needs to summarize facts based on any historical value of the attribute, not just the historically accurate and current as we've primarily been discussing, you could have a series of type 3 attributes in the dimension. On every dimension row, there would be a "current" attribute that is overwritten, as well as attributes for each annual designation, such as the 2007 assignment, 2006 assignment, and so on. Generalized reporting would likely most often use the current assignment attribute in the dimension.

Mini-Dimensions: Add a New Dimension

The slowly changing dimension workhorse technique, type 2, is inappropriate for tracking attribute changes in large dimension tables with millions of rows. Though relational databases are capable of supporting dimension tables of this size, it is important to adopt a conservative design to keep these dimensions under control.

Unfortunately, huge customer dimensions are even more likely to change than medium-sized product dimensions, which seemingly puts you between a rock and a hard place. You must track the changing nature of the customer dimension, but don't dare use the type 2 technique due to the already imposing size of the table and the frequency of changes.

The solution to this dilemma is to break off the more frequently changing customer attributes into their own separate dimension table, as in Figure 6-9, called a *mini-dimension*. In this example, the new mini-dimension contains customer demographics. The mini-dimension's grain is one row per demographics profile, whereas the grain of the primary customer dimension is one row per customer.

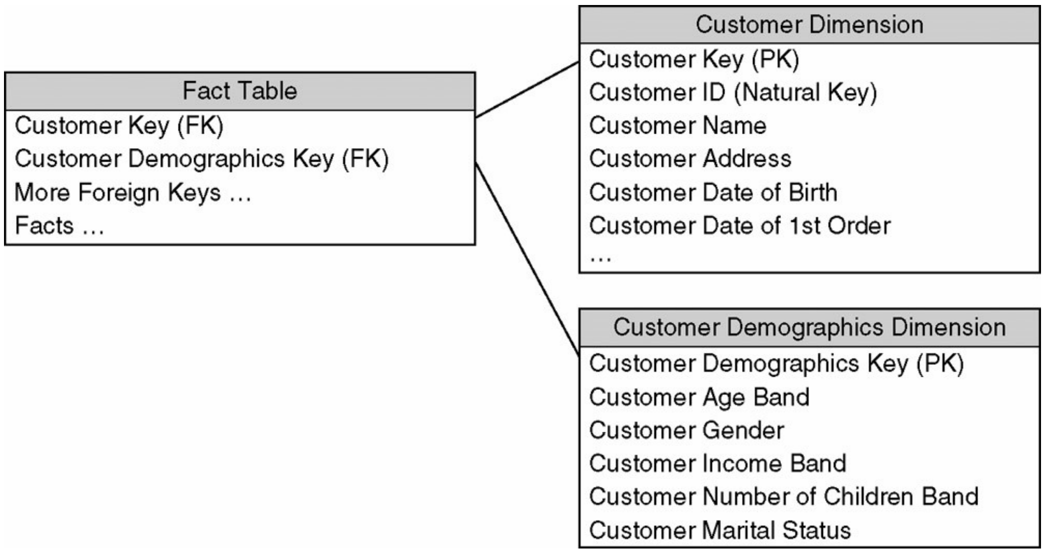


Figure 6-9: Fact table with a mini-dimension to track customer demographic changes

Although the mini-dimension technique allows you to track relatively frequent changes in a large dimension table, it requires a few compromises:

- **Banded values.** The demographic attributes have been clumped into banded ranges of discrete values. This places a limit on the specificity of the data, such as income, and makes it quite impractical to change to a different set of value bands at a later time.
- **Restricted growth.** The mini-dimension itself cannot grow too large; a gross general rule of thumb is that the mini-dimension should have fewer than a couple hundred thousand rows. Surprisingly, a workable solution to this challenge is to build a second mini-dimension. It's not uncommon for an organization tracking a very rich set of customer attributes to need separate

mini-dimensions for customer demographics, geographic attributes, credit bureau attributes, and market segmentation attributes and scores.

- **Separation from core dimension.** The mini-dimension has been physically separated from the more constant descriptors of the customer, making it harder to browse the data as if it were a single dimension; the demographics data is linked to the core customer data only through the fact table. If desired, you may opt to populate type 1 columns for the current values of the mini-dimension attributes within the customer dimension. This facilitates stand-alone demographic analysis of your customer base without traversing the fact table. It is important to clearly label the type 1 mini-dimension attributes in the core customer dimension as "current."

Finally, a sharp eyed reader may notice that this approach allows you to associate demographics with the customer only when a fact table row occurs, such as a sales event or insurance claim event. If no fact event happens, then no linkage is established. If there is a danger of missing a demographics association, a supplemental factless fact table, as we discuss later in this chapter, can be used to capture the relationship between the customer dimension and mini-dimension over time; a row is loaded when a relationship change occurs. Fortunately, in many industries, there is a natural monthly business process that results in a periodic snapshot fact table that effortlessly captures the relationship of all customers and their applicable mini-dimension values without the need for an additional factless fact table.

Hybrid Slowly Changing Dimension Techniques

These fundamental slowly changing techniques are adequate for most situations. However, sometimes you need hybrid variations that build on these basics to serve more sophisticated analytics. Before using these hybrids, remember the need to balance analytic power against ease of use; strive to maintain equilibrium between flexibility and complexity. These hybrid techniques are not for the faint of heart.

Business folks sometimes want to preserve the historically accurate dimension attribute associated with a fact, but also want the option to roll up historical facts based on current dimension characteristics. None of our fundamental slowly changing dimension techniques addresses this requirement entirely.

One solution is to capture type 2 attribute changes by adding a row to the primary dimension table, but in addition, there's a "current" attribute on each row that is overwritten (type 1) for the current *and* all previous type 2 rows. When a change occurs, the most current dimension row has the same value in the uniquely labeled current and historical ("as was") columns, as illustrated in Figure 6-10.

PRODUCT KEY	Product Description	Product Code	Historical Department	Current Department	Expiration Date	Effective Date	Current Row Ind
12345	IntelliKidz 1.0	ABC999-Z	Education	Critical Thinking	2/15/2007	5/31/2007	Not current
25984	IntelliKidz 1.0	ABC999-Z	Strategy	Critical Thinking	6/1/2007	12/31/2007	Not current

34317	IntelliKidz 1.0	ABC999-Z	Critical Thinking	Critical Thinking	1/1/2008	12/31/9999	Current

Open table as spreadsheet

Figure 6-10: Hybrid slowly changing dimension with historical and current attributes

Instead of having the historical and current attributes reside in the same physical table, the current attributes could sit in an outrigger table joined to the dimension natural key or durable identifier in the dimension table. The outrigger contains just one row of current data for each natural key in the dimension table; the attributes are overwritten whenever change occurs. Alternatively, this outrigger could be a view of the type 2 dimension table, restricted to just the current rows.

Finally, if you have a million-row dimension table with many attributes requiring historical and current tracking, you should consider including the dimension natural key or durable identifier as a fact table foreign key, in addition to the surrogate key for type 2 tracking. This technique essentially associates two dimension tables with the facts, but for good reason. The type 2 dimension has historically accurate attributes for filtering or grouping based on the effective values when the fact row was loaded. The dimension natural key joins to a table with just the current type 1 values. You use the attributes on this dimension table to summarize or filter facts based on the current profile, regardless of the values in effect when the fact row was loaded. Again, the column labels in this table should be prefaced with "current" to reduce the risk of user confusion.

Role-Playing Dimensions

When the same physical dimension table plays distinct logical roles in a dimensional model, we refer to it as *role-playing*. The most common example of role-playing involves the date dimension. Fact tables always have at least one date dimension foreign key, but sometimes there are multiple dates associated with each fact row, such as the transaction date of the event, as well as the effective date. Later in this chapter, we discuss accumulating snapshot fact tables, which often have a handful of dates on each row.

Each date foreign key in the fact table points to a specific date dimension table, as shown in Figure 6-11. You can't join these multiple foreign keys to the same table because SQL would interpret the two-way simultaneous join as requiring the dates to be the same, which isn't very likely.

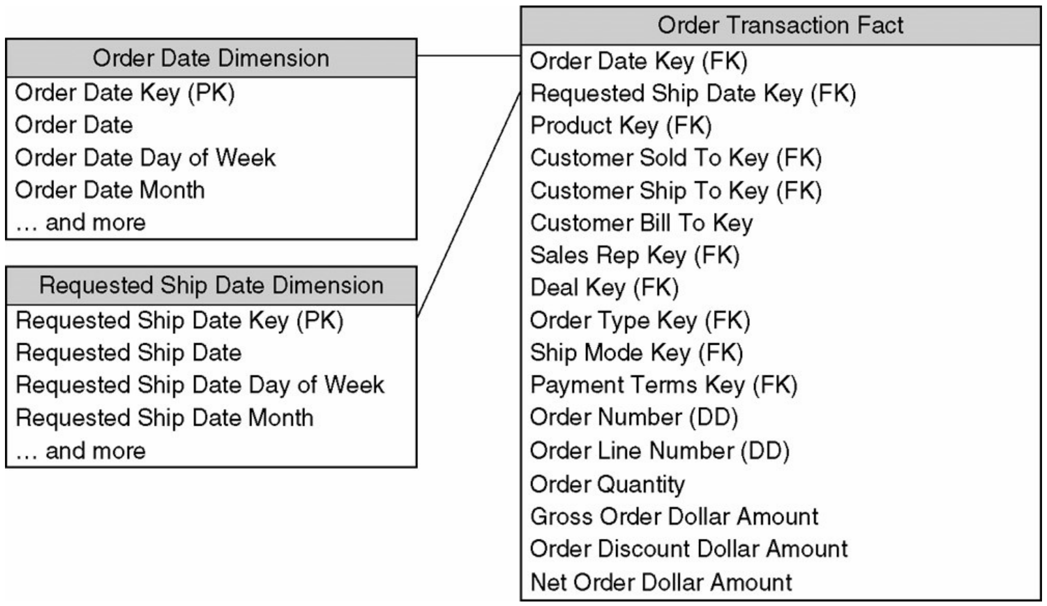


Figure 6-11: Role-playing date dimensions

Instead of a two-way join, you need to fool SQL into believing that there are independent date dimension tables with unique column labels. If you don't uniquely label the columns, users won't be able to tell the attributes apart when several are dragged into a report.

Even though you cannot literally use a single date dimension in Figure 6-11, you want to build and administer a single physical date table behind the scenes. The illusion of independent date dimension tables is created by using views or aliases, depending on the database platform. Now that you have uniquely described date dimensions, they can be used as if they were independent with completely unrelated constraints.

Depending on your industry, role playing can occur with a variety of dimensions, in addition to the date, such as:

- Customer (ship to, bill to, and sold to)
- Facility or port (origin, destination)
- Provider (referring, performing)
- Representative (seller, servicer)

Note A single dimension may play several simultaneous roles with a fact table. The underlying dimension may exist as a single physical table, but each of the roles must be presented in a separately labeled view.

Junk Dimensions

Especially when modeling a complex transactional business process, after identifying the obvious business dimensions, you are often left with a number of miscellaneous flags and text attributes that can't find an appropriate home in any of the existing dimension tables. This situation leaves the designer with a number of bad alternatives, all of which should be avoided, including:

- **Leave them in the fact table.** Leaving the flags and text attributes in the fact table is bad because it might cause the fact table row length to swell alarmingly. It would be a shame to create a nice tight design and then leave several uncompressed 20-byte text columns in the fact row.
- **Make them into separate dimensions.** Making each flag and attribute into its own separate dimension doesn't work either. This could cause the fact table to swell to an unusually large number of dimension foreign keys. As we mentioned earlier, we strive to have fewer than twenty foreign keys in a fact table for most industries and business processes.
- **Eliminate them.** While tempting, stripping out the flags and indicators could be a mistake if there is demonstrated business relevance to these fields. It is worthwhile, of course, to examine this question. If the flags and text attributes are incomprehensible, inconsistently populated, or only of operational significance, they should be left out.

The key to solving this problem is to study the flags and text attributes carefully and to place them into one or more *junk* dimensions, as illustrated in Figure 6-12, rather than adding more clutter to the fact table.

Open table as spreadsheet

Invoice Indicator Key	Payment Terms	Order Mode	Ship Mode
1	Net 10	Telephone	Freight
2	Net 10	Telephone	Air
3	Net 10	Fax	Freight
4	Net 10	Fax	Air
5	Net 10	Web	Freight
6	Net 10	Web	Air
7	Net 15	Telephone	Freight
8	Net 15	Telephone	Air
9	Net 15	Fax	Freight
10	Net 15	Fax	Air
11	Net 15	Web	Freight
12	Net 15	Web	Air
13	Net 30	Telephone	Freight
14	Net 30	Telephone	Air
15	Net 30	Fax	Freight
16	Net 30	Fax	Air
17	Net 30	Web	Freight
18	Net 30	Web	Air
19	Net 45	Telephone	Freight
20	Net 45	Telephone	Air
21	Net 45	Fax	Freight

Invoice Indicator Key	Payment Terms	Order Mode	Ship Mode
22	Net 45	Fax	Air
23	Net 45	Web	Freight
24	Net 45	Web	Air

Figure 6-12: Sample rows from a junk dimension

Note *Junk dimension* is a highly technical term; when describing the dimension to business users, you probably want to use different vocabulary, such as the "invoice indicator" dimension.

A simple example of a junk dimension would be taking ten yes/no indicators out of the fact table and putting them into a single dimension. It is okay if there seems to be no correlation among these ten indicators. You simply pack all ten indicators into one dimension with ten attributes; while you're at it, convert the yes/no indicator values to more meaningful, descriptive labels. At the worst, you have 2^{10} = 1,024 rows in this junk dimension. It probably isn't very interesting to browse among these indicators within the dimension because every indicator occurs with every other indicator. But the junk dimension is a useful holding place for the indicators to support filtering or reporting. And the ten columns in the fact table have been replaced with one small surrogate key.

A subtle issue in the design is whether you create all the combinations in the junk dimension in advance or whether you create junk dimension rows for the combinations of flags as you encounter them in the data. The answer depends on how many possible combinations you expect and what the maximum number could be. Generally, when the number of theoretical combinations is very high and you don't think you will encounter them all, you should build a junk dimension row at extract time whenever you encounter a new combination of fields.

Note Data profiling can play a role in helping identify appropriate junk dimension opportunities and strategies. Many times what look like independent junk attributes in the fact table turn out to be correlated. At a recent healthcare client, a set of claim transaction attributes theoretically had more than one trillion possible combinations, but when the data was profiled, it was determined that fewer than 80,000 unique combinations were ever observed.

As we mentioned when discussing the mini-dimension, sometimes you may create multiple junk dimensions for a single fact table, depending on the correlations between the miscellaneous flags and indicators.

A final kind of messy attribute that fits well into the junk dimension approach is the open-ended comments field that is often attached to a fact row. If you decide that the contents of the comments field warrant inclusion in the data warehouse, you should proceed by pulling them out of the fact row and into a junk dimension. Often, many of these freeform text comments fields are blank, so you will need a special surrogate key that points to the "no comment" row in the dimension; most of your fact table rows will use this key.

Note Text facts are a dimensional modeling oxymoron. Fact tables should consist only of keys, numerical measurements, and degenerate dimensions.

Snowflaking and Outriggers

Dimensions are *snowflaked* when the redundant attributes and decodes are removed to separate tables and linked back into the original table with artificial keys. In other words, a snowflaked dimension is normalized. An example snowflaked product dimension table is shown in Figure 6-13.

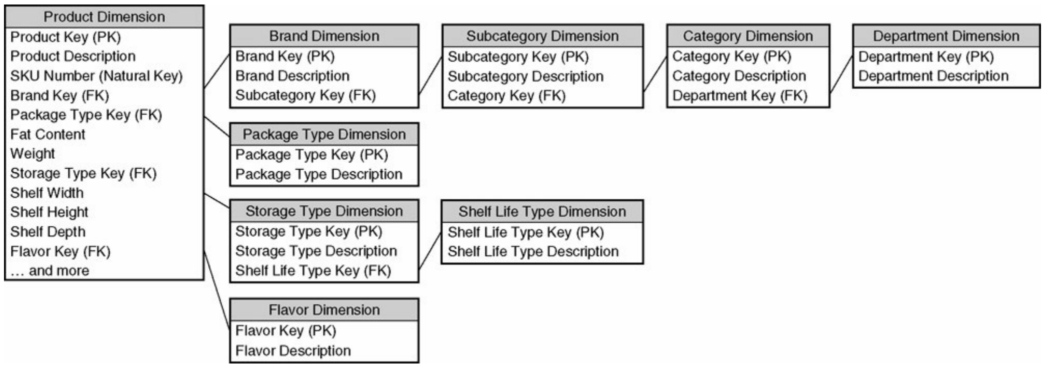


Figure 6-13: Snowflaked dimension tables

Generally, snowflaking is not recommended for your dimensional models because it almost always makes the user presentation more complex and less legible. Database designers who are comfortable with third normal form design disciplines often like this intricacy, but users are typically intimidated by it. Snowflaking also makes most forms of browsing among the dimension attributes slower. Remember that most browses involve selecting the distinct values of a dimension attribute subject to one or more other attributes in the dimension being constrained. Obviously a snowflaked category table will perform extremely well if the user is asking only for the distinct values of category with no ancillary constraints, but if the user is constraining on package type at the same time, then the query needs to join the snowflaked category table back through to the product dimension table and possibly out through another snowflake link to the package type table where the constraint has been placed.

Note Any argument that snowflaking of dimensional models assists in their maintenance is specious. Remember, the maintenance work happens in the ETL system before the data is loaded into the dimensional schema in the presentation server.

Some designers snowflake dimensions because they want to save disk space. Surely they can save space by eliminating all those nasty repeating text columns. Granted, if the artificial key required for each snowflaked attribute is shorter than the text string it replaces, some space will be saved. Suppose in a 250,000 row product table, we replaced every 15-byte category name with a 2-byte category key. We have thereby saved approximately 13 bytes on 250,000 rows or 3.25 MB. Our overall product table may well be larger than 100 MB, and the associated sales fact table is potentially hundreds of gigabytes. We have just saved 3 MB on a base of over 100 GB, or 0.003 percent. This negligible savings has been paid for with extra administration in creating and populating the artificial keys, as well as an almost certain performance penalty for cross-attribute browsing.

Despite this general recommendation against snowflaking, there are some situations where we agree it might be advantageous to build an "outrigger" dimension that looks like the beginning of a snowflake, as illustrated in Figure 6-14. In this example, the outrigger is a set of demographic attributes that exist at the county level; all customers in a county share an identical set of attributes. Furthermore, the county demographic data consists of a label, population count, and population percentage for 50 different demographic groupings, so there are 150 attributes. In this case, it makes sense to isolate this demographic data in an outrigger. The demographic data is available at a significantly different grain than the primary dimension and is administered on a different frequency. Also, you might really save space in this case if the underlying dimension is large.

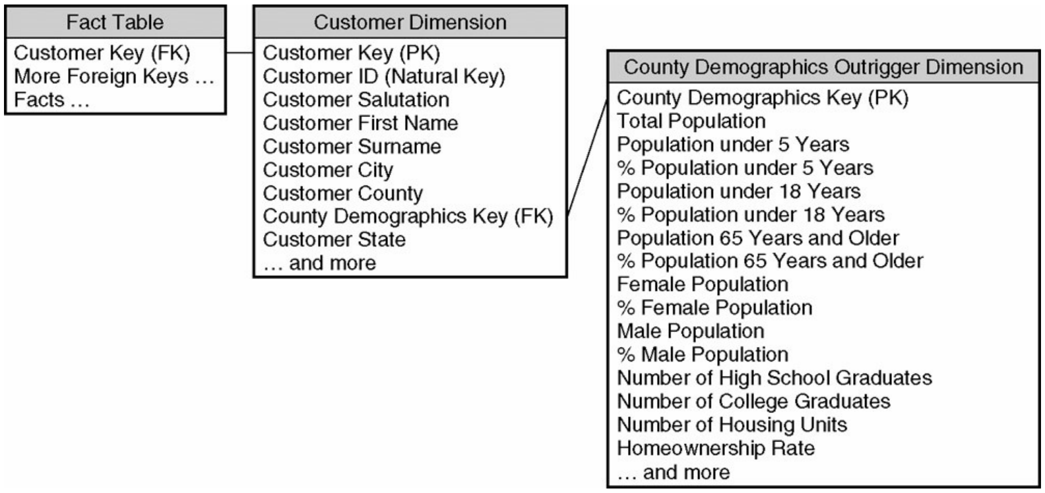


Figure 6-14: Example of a permissible snowflaked outrigger

A more commonly encountered outrigger is a date dimension snowflaked off a primary dimension, such as a customer's date of birth or the product's introduction date. In this scenario, the outrigger date attributes should be descriptively and uniquely labeled to distinguish them from the other dates associated with the business process. It only makes sense to outrigger a primary dimension table's date attribute if the business needs to filter and group the date by non-calendar date attributes, such as the fiscal period or business day indicator. Otherwise, you could just treat the date attribute as a standard date format field. If you do use a date outrigger, be careful that the outrigger dates all fall within the date range stored in the standard date dimension table.

Note Though outriggers are a permissible form of snowflaking, your dimensional model should not be littered with outriggers given the potentially negative impact on both legibility and query performance. Outriggers should be the exception rather than the rule.

Handling Hierarchies

Hierarchies are a fact of life in a dimensional model. In this section, we review the common fixed hierarchical relationships, as well as variable depth hierarchies.

Fixed Hierarchies

As a result of dimension table denormalization, simple many-to-one hierarchies within a dimension table are often flattened and presented as a series of attributes in the dimension. In fact, it's not unusual to have more than one well-defined hierarchy in a given dimension. For example, the marketing and finance departments may have incompatible and different views of the product hierarchy. Despite your efforts to define common, conformed dimensions for the enterprise, you may need to retain all of the marketing-defined attributes and the finance-defined attributes in the detailed master product table.

Hierarchies are often associated with drill paths in the data. Drilling down is the oldest and most venerable maneuver performed by users in a DW/BI system. Drilling down means nothing more than delving into more detail, typically by adding another dimension attribute to an existing SQL request. Drilling can occur within a predetermined hierarchy; however, you can also drill down to more detail by using any unrelated attribute that is not part of a hierarchy.

Variable Depth Hierarchies via Bridge Tables

Unfortunately, not all data hierarchies are simple and fixed. The dimensional modeler's challenge is to balance the tradeoff between ease of use and flexibility when representing more complex hierarchies, such as variable depth or ragged hierarchies, because they require a more complex solution than simple denormalization.

Representing a variable depth organization structure or parts hierarchy is a difficult task in a relational environment. Let's assume that you want to report the revenues of a set of commercial customers who have subsidiary relationships with each other. Your organization sells products or services to any of these commercial customers; you may want to look at them individually or as families arranged in a hierarchical structure.

The traditional computer science response to this hierarchy would be to put a recursive pointer in each customer dimension row containing the customer key of its parent. Although this is a compact and effective way to represent an arbitrary hierarchy, this kind of recursive structure cannot be used effectively with standard SQL when you want to both navigate and summarize fact table measurements based on the recursive relationship.

As we further explain in *The Data Warehouse Toolkit, Second Edition*, you can navigate a variable depth hierarchy through the use of a bridge table. Illustrated in Figure 6-15, the bridges its between the fact table and customer dimension when you want to navigate the hierarchy. If the bridge table is omitted, you can filter and group facts based on the individual customer involved in the transaction, but there's no ability to roll up or drill down through the customer's organization structure.

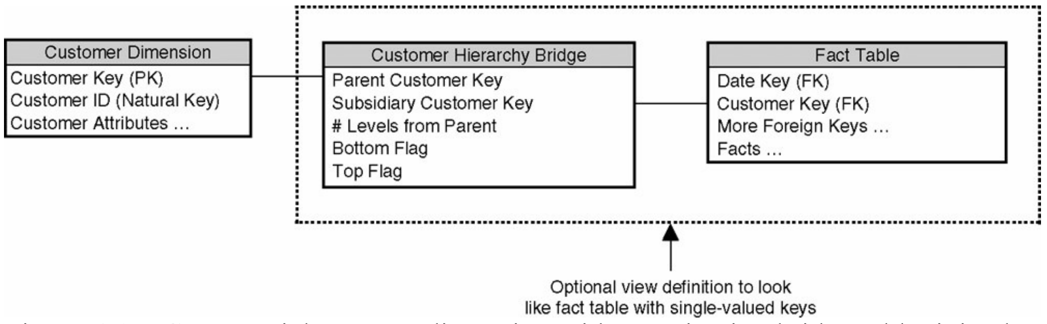


Figure 6-15: Commercial cutomer dimension with a navigation bridge table, joined to descend the tree

The organization bridge table contains one row for each pathway from a customer entity to each subsidiary beneath it. There is also a row for the zero-length pathway from a customer to itself. Each pathway row contains the customer key of the rollup parent entity, customer key of the subsidiary entity, number of levels between the parent and the subsidiary, a flag that identifies a subsidiary as being at the bottom-most possible level, and a flag that indicates whether there are any more nodes above the parent.

When you want to descend the organization hierarchy, you join the tables as shown in Figure 6-15. You can now group or constrain on any attribute within the customer table, and retrieve facts for all the subsidiaries at or below all the customers implied by the constraint. You can use the number of levels field in the organization bridge table to control the depth of the analysis. A value of 1 would give all the direct subsidiaries of a customer. A value of "greater than zero" would give all subsidiary customers but not the original parent.

Alternatively, when you want to ascend the organization hierarchy, the joins would be altered. The parent customer key would now be joined to the fact table, and the subsidiary customer key would be joined to the customer dimension table.

Although the bridge table allows SQL to navigate and aggregate facts in an unpredictably deep hierarchy, there are downsides to this approach because our twin goals of usability and query performance are both negatively compromised. The bridge table can quickly become huge and unwieldy with the explosion of a complicated organization or bill-of-materials rollup structure. Query performance suffers when the bridge is used. And though the computer scientist may be able to easily navigate the bridge, most users will come to a crashing halt when asked to cross the bridge. Fortunately, some OLAP cube products have built in awareness of recursive parent-child hierarchies to help ease the burden.

Many-Valued Dimensions with Bridge Tables

So far in our designs, there has been a straightforward many-to-one relationship between the fact table and associated dimension tables; many rows in the fact table join to a single row in the dimension table. However, there are some situations in which each fact measurement can be associated with multiple dimension values.

Health care offers a vivid example of a many-valued dimension. Assume we're modeling the hospital billing business process with a row in the fact table for each line item on a bill. The problem for the designer is what to do with the diagnosis dimension. Real patients often have more than one diagnosis. Really sick patients may have ten or more diagnoses. To handle this open-endedness, you cannot add multiple diagnosis dimension foreign keys to the fact table. Not only do you not know what the extreme upper bound is for the potential number of diagnoses, but the resulting schema would be awkward to query. OR joins across dimensions are anathema to relational systems and SQL's `GROUP BY` logic fails. Likewise, you can't change the grain of the fact table to be one row per diagnosis associated with a hospital bill line item because the billing metrics don't make sense at this granularity.

To resolve the design problem and provide for an arbitrary number of diagnoses, a bridge table is inserted between the fact table and the diagnosis dimension, as in Figure 6-16. Each row in the diagnosis group table contains the diagnosis group key, individual diagnosis key, and weighting factor.

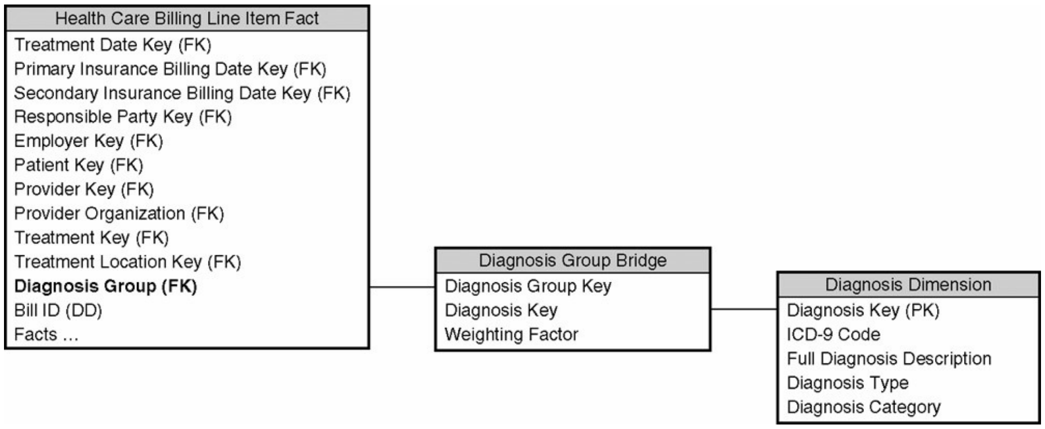


Figure 6-16: Solving the many-valued diagnoses problem with a bridge table

A patient with three diagnoses would have three rows in their respective diagnosis group. The weighting factor is a fractional value that sums to one for each diagnosis group; the determination of the weighting factor is a business issue. The individual diagnosis key joins to the diagnosis table, which contains a full description of each diagnosis.

With this structure, the user or BI application can perform two types of queries. First, the user can ask for a correctly weighted summary of all billed charges grouped by diagnosis. The join through the bridge table picks up the weighting factor for each diagnosis and multiplies it by the fact amount to allocate the facts according to the contribution of each individual diagnosis in the group. The use of the weighting factor guarantees that the bottom line total is accurate.

Alternatively, the user can request an impact report that performs the same join across the tables, but does not use the weighting factor. In this case, an impact report deliberately double counts the amounts, but correctly determines the impact each diagnosis has in terms of the total fact amounts associated with that diagnosis.

A fact table representing account balances in a bank has an analogous problem. Each account may have one or more individual customers as account holders. To join an individual customer dimension table to an account balances fact table, you need an account-to-customer bridge table that has the same structure as the diagnosis group table in the previous example, as illustrated in Figure 6-17.

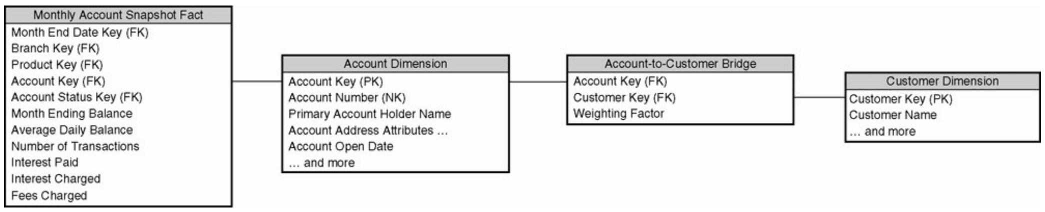


Figure 6-17: Solving the multiple customers per account problem with a bridge table

Analysis surrounding commercial customers' industry classification codes also faces the many-valued problem. Each commercial customer may be represented by one or more industry classification codes. Again, by building an industry classification group bridge table with the same structure as the last two bridge tables, you can sum up the facts by any attribute found within the individual industry classification dimension table, either correctly weighted or as an impact report. You may want to add an attribute on the customer dimension to designate their primary industry classification to simplify some analysis by avoiding the bridge.

Before we leave the topic of bridge tables, we want to reiterate that using bridge tables to resolve many-to-many relationships is the path of last resort due to the implications on usability and performance. Resolving the many-to-many issue directly in the fact table makes for a much simpler solution, however sometimes this just isn't feasible given the natural granularity of the measurement event. In these cases, the bridge comes to the rescue.

More on Facts

Now that we've explored dimension tables in more detail, we turn our attention to fact tables.

Three Fundamental Grains

Regardless of the industry and its associated business processes, all measurement events can be characterized as either transactions, periodic snapshots, or accumulating snapshots. We describe the characteristics of these three fundamental fact table grains in the following sections. The DW/BI environment often pairs complementary fact tables that deliver data at different base granularity for a more complete picture of the business. Though the fact tables share conformed dimensions, the rhythm and associated administration varies by fundamental grain.

Transaction Fact Tables

The most basic and common fact tables are transactional. The grain is specified to be one row per transaction, or one row per line on a transaction. The transaction grain is a point in space and time; the measurements at a transaction grain must be true at that moment.

Transaction data at its lowest level typically has the greatest number of dimensions associated with it. Whenever a transaction event occurs, extensive context about the transaction is also captured, as illustrated back in Figure 6-1.

Rows are inserted into a transactional fact table only when activity occurs. Once a transaction fact row is posted, it is typically not revisited.

As we've described earlier in this chapter, detailed transaction schema are very powerful. The minute measurements allow you to monitor the most granular activities of the business. However, you sometimes need to complement this gory detail with snapshots to more easily visualize behavior or performance trends.

Periodic Snapshot Fact Tables

The second most common type of fact table is the periodic snapshot. With periodic snapshots, you take a set of pictures, just as the name implies, capturing the performance of a business process spanning a well defined periodic interval of time and load these pictures into a fact table. At a predetermined interval, such as daily, weekly, or monthly, more pictures are taken at the same level of detail and stacked consecutively in the fact table. Note that this gives the designer a lot of latitude: Any fact that describes activity over the time span is fair game.

Some business processes are naturally represented as periodic snapshots. For example, the value of a bank account is naturally taken as a snapshot on a daily or monthly basis. You can envision inventory balances snapshots on a daily or weekly timeframe. The most familiar financial reports are periodic snapshots. Figure 6-18 illustrates a periodic monthly snapshot of automobile loans.

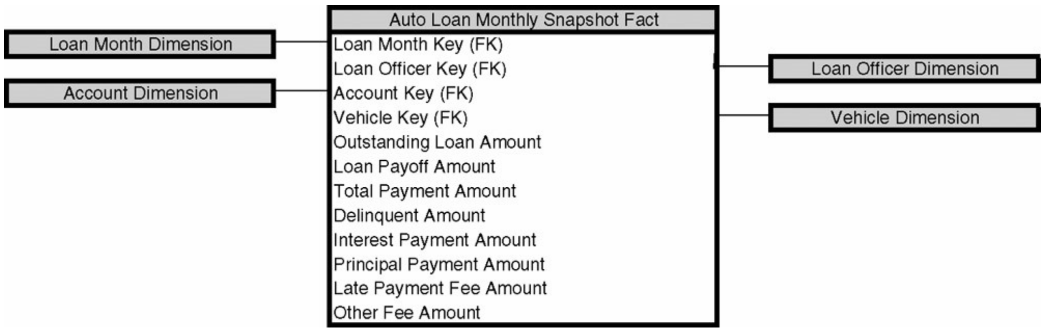


Figure 6-18: Periodic snapshot grained fact table for monthly loans

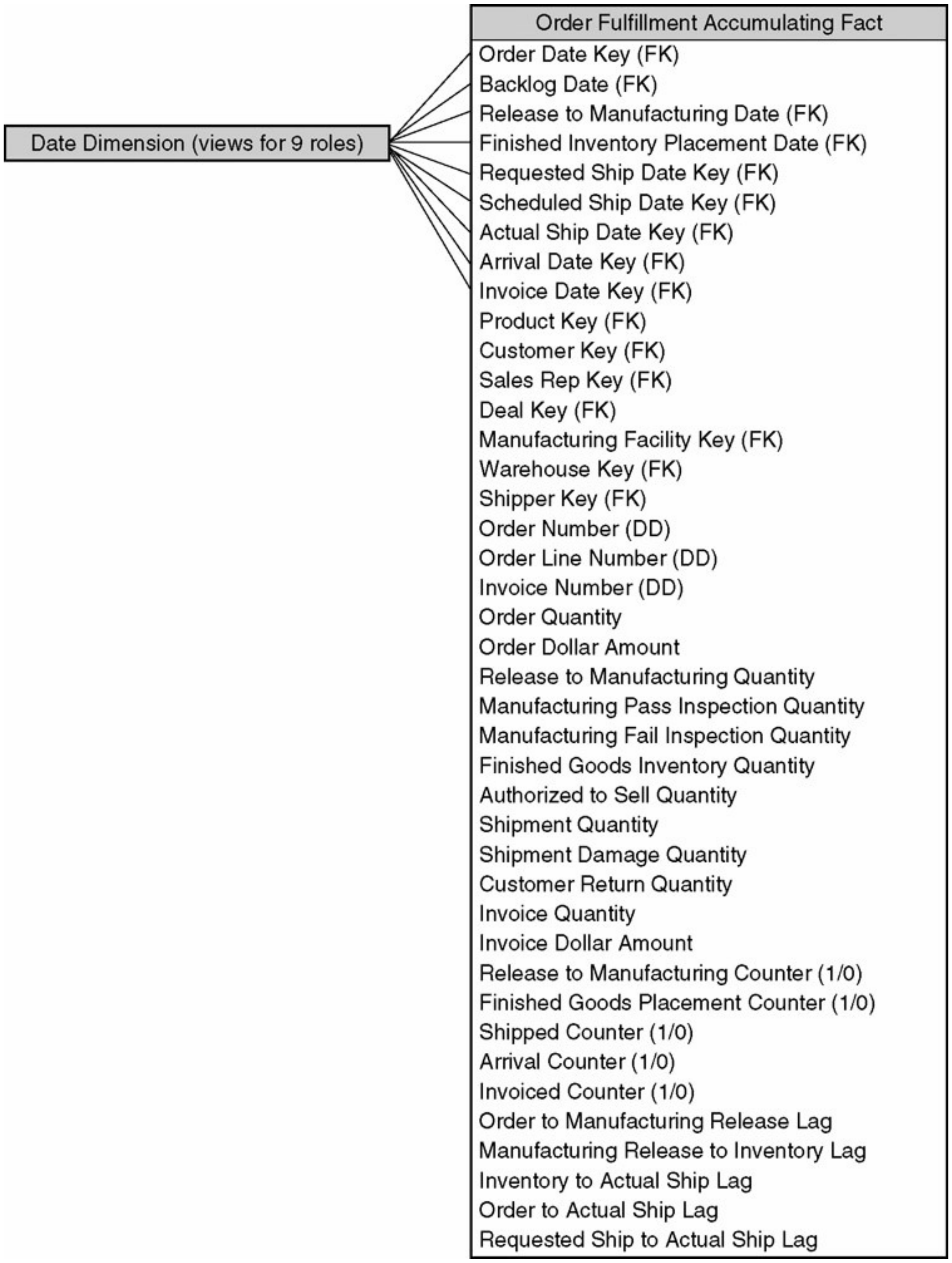
It should go without saying that the periodic snapshot complements the detailed transaction facts, but is not a replacement. You still need the granular details to ask highly precise questions about performance.

Like the transactional fact tables, periodic snapshot rows are typically not revisited once they've been loaded into the fact table. Periodic snapshots often share many of the same conformed dimensions with the transaction fact tables, however there are typically far fewer dimensions.

Accumulating Snapshot Fact Tables

The third type of fact table is the least frequently encountered, however it serves a very powerful function in certain applications.

Unlike periodic snapshots, which house regularly captured measurements, accumulating snapshots are used to represent activity over an indeterminate time span for processes that have a well defined beginning and end. For example, in Figure 6-19, a row is loaded into the accumulating snapshot fact table for each line on an order when it is originally received. As the order moves through the fulfillment process, the fact table rows are to reflect the latest status. Ultimately, the pipeline activity on an order line item is completed when the invoice is generated.


Date Dimension (views for 9 roles)
Order Fulfillment Accumulating Fact
Order Date Key (FK)
Backlog Date (FK)
Release to Manufacturing Date (FK)
Finished Inventory Placement Date (FK)
Requested Ship Date Key (FK)
Scheduled Ship Date Key (FK)
Actual Ship Date Key (FK)
Arrival Date Key (FK)
Invoice Date Key (FK)
Product Key (FK)
Customer Key (FK)
Sales Rep Key (FK)
Deal Key (FK)
Manufacturing Facility Key (FK)
Warehouse Key (FK)
Shipper Key (FK)
Order Number (DD)
Order Line Number (DD)
Invoice Number (DD)
Order Quantity
Order Dollar Amount
Release to Manufacturing Quantity
Manufacturing Pass Inspection Quantity
Manufacturing Fail Inspection Quantity
Finished Goods Inventory Quantity
Authorized to Sell Quantity
Shipment Quantity
Shipment Damage Quantity
Customer Return Quantity
Invoice Quantity
Invoice Dollar Amount
Release to Manufacturing Counter (1/0)
Finished Goods Placement Counter (1/0)
Shipped Counter (1/0)
Arrival Counter (1/0)
Invoiced Counter (1/0)
Order to Manufacturing Release Lag
Manufacturing Release to Inventory Lag
Inventory to Actual Ship Lag
Order to Actual Ship Lag
Requested Ship to Actual Ship Lag
Figure 6-19: Accumulating snapshot fact table for order fulfillment

Accumulating snapshot fact tables exhibit several unique characteristics compared to the other two types of fact tables. For starters, accumulating snapshots contain multiple date foreign keys, each corresponding to a major milestone in the workflow. While transactional schemas sometimes have several dates associated with each row, the accumulating fact table is distinctive because many of the date values are unknown when the row is first loaded. As we discussed earlier, you need a special row in the date dimension table to represent unknown future dates. This surrogate date key will be referenced extensively when the accumulating snapshot fact row is first loaded, and then is systematically overwritten as the actual date of the milestone occurrence is captured.

In addition to the typical quantities and amounts, accumulating snapshots often contain two additional types of facts. The first is the milestone counter; there is often a lengthy list of counters corresponding to the key milestones in the pipeline, which take on a 1 or 0 value. These counters are used to quickly determine how many fact rows have progressed to a given point in the workflow. Second, accumulating snapshots often have lag facts representing the length of time between milestones, such as the elapsed time between the first and second milestone, or the first milestone and the final event in the pipeline.

Of course, the primary differentiator of accumulating snapshots is that the fact row is updated. Accumulating snapshots are almost always complemented by one or more transaction fact tables. The accumulation process presents the key transactional activity side-by-side in a single fact table row. Although this delivers query performance and analytic power, it's not an adequate replacement for the underlying transaction fact tables because dimensional details are inevitably lost during the accumulation.

Note Transactions and snapshots are the yin and yang of a data warehouse. Transactions give us the fullest possible view of detailed behavior, whereas snapshots allow us to easily and quickly monitor overall performance. We often need them both because there is no simple way to combine these two contrasting perspectives.

Facts of Differing Granularity and Allocation

Sometimes a single business process produces measurements at different levels of detail. For example, metrics are captured for each line on an order, however a single freight charge might be levied for the entire order.

All the measurements in a single fact table must be valid for the table's stated granularity. In other words, you can't have facts with differing granularity in the same fact table; all the measurements must exist at the same level of detail.

So what's a dimensional modeler to do when confronted with measurements that naturally exist at different grains? The first response should be to try to force all of the facts down to the lowest level. This procedure is broadly referred to as *allocating* the higher level facts to a lower granularity to align with the other captured measurements, as illustrated in Figure 6-20.

Order Line Transaction Fact

- Order Date Key (FK)
- Requested Ship Date Key (FK)
- Product Key (FK)
- Customer Sold To Key (FK)

- Customer Ship To Key (FK)
- Customer Bill To Key
- Sales Rep Key (FK)
- Deal Key (FK)
- Order Indicator Key (FK)
- Order Number (DD)
- Order Line Number (DD)
- Order Line Quantity
- Gross Order Line Amount
- Order Discount Line Amount
- Net Order Line Amount
- Allocated Freight Amount by Weight
- Allocated Freight Amount by Value

Figure 6-20: Header grained facts allocated to the line fact table

Before allocation can occur, the data stewards need to define the allocation algorithm. Perhaps the header level freight charge is allocated to the line level based on weight, or value, or is merely spread evenly across the order lines. Whatever the rule, it must be defined by the business, possibly entailing some degree of compromise or controversy, which is all the more reason that you'll want the stewards involved. If the business can't agree on a single allocation scheme, you may end up with two uniquely labeled facts: allocated freight charge based on weight for the logistics department and allocated freight charge based on value for finance.

The beauty of allocation is that all the measurements generated by a business process can now live in a single fact table, with every dimension attribute as an equal candidate for filtering or grouping any measurement. Although this is optimal, sometimes it's not achievable. The business may not be able to reach consensus on an allocation rule. We find that discrete "box oriented" businesses, like consumer goods manufacturing, often can reach agreement on allocation rules. Conversely, infrastructure and service oriented businesses, like banks and telecommunications companies, typically have a more difficult time agreeing on allocation rules.

It may be critical that the header fact precisely matches the operational value, which may prohibit allocation and its inevitable rounding errors. In this situation, the modeler has no choice but to present a separate fact table with the granularity of one row per transaction header. This fact table, illustrated in Figure 6-21, would share most of the same dimensions as the lower grained line item fact table, with the notable exception of product. In addition to storing facts that are naturally captured at the higher level of detail, this fact table could also store aggregated totals from the order line.

Order Header Transaction Fact

- Order Date Key (FK)

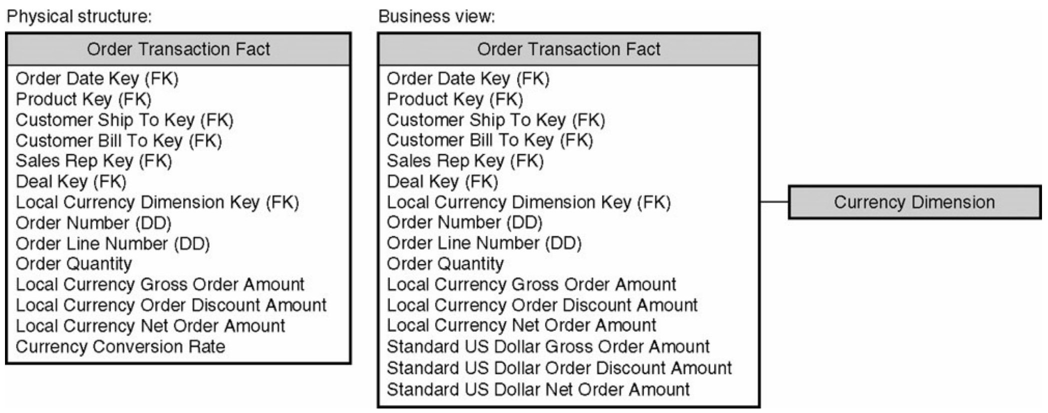
- Requested Ship Date Key (FK)
- Customer Sold To Key (FK)
- Customer Ship To Key (FK)
- Customer Bill To Key
- Sales Rep Key (FK)
- Deal Key (FK)
- Order Indicator Key (FK)
- Order Number (DD)
- Gross Order Header Amount
- Order Discount Header Amount
- Net Order Header Amount
- Order Header Freight Amount

Figure 6-21: Header grain fact table with aggregated line facts

Multiple Currencies and Units of Measure

Sometimes fact measurements need to be expressed in multiple currencies or units of measure. In both cases, packaging the measurements and conversion factors in the same fact table row provides the safest guarantee that the factors will be used correctly.

In the case of currency conversion, the most common requirement is to provide the measurements in the local currency in which they were captured, as well as converted to the standardized international currency for the organization. As shown in Figure 6-22, you could store one set of measurements, along with the appropriate conversion rate fact (corresponding to the transaction date/timestamp rate, or the end of business day close rate, or whatever rate is deemed appropriate per the business rules). You could also supplement the fact table with a dimension foreign key that designates the currency associated with the local currency facts.


Physical structure:
Order Transaction Fact
Order Date Key (FK)
Product Key (FK)
Customer Ship To Key (FK)
Customer Bill To Key (FK)
Sales Rep Key (FK)
Deal Key (FK)
Local Currency Dimension Key (FK)
Order Number (DD)
Order Line Number (DD)
Order Quantity
Local Currency Gross Order Amount
Local Currency Order Discount Amount
Local Currency Net Order Amount
Currency Conversion Rate
Business view:
Order Transaction Fact
Order Date Key (FK)
Product Key (FK)
Customer Ship To Key (FK)
Customer Bill To Key (FK)
Sales Rep Key (FK)
Deal Key (FK)
Local Currency Dimension Key (FK)
Order Number (DD)
Order Line Number (DD)
Order Quantity
Local Currency Gross Order Amount
Local Currency Order Discount Amount
Local Currency Net Order Amount
Standard US Dollar Gross Order Amount
Standard US Dollar Order Discount Amount
Standard US Dollar Net Order Amount
Currency Dimension
Figure 6-22: Fact table supporting multiple currencies

You can now deliver this fact table to the users and BI tools through a view that extrapolates using the conversion rate to display both sets of facts. Of course, the local currency facts are semi-additive because they can only be summed correctly when limited to a single currency; the query results would certainly be erroneous if you tried to add up facts captured in Thai bhat, Japanese yen, and euros.

Similarly, in a value chain involving several business processes monitoring the flow of products through a system or multiple measures of inventory at different points, sometimes a conflict arises in presenting the amounts. Different parties along the chain may wish to see the numbers expressed in different units of measure. For instance, manufacturing managers may wish to see the entire product flow in terms of carloads or pallets, whereas store managers may wish to see the amounts in shipping cases, retail cases, scan units (sales packs), or consumer units. Likewise, the same inventory quantity of a product may have several possible economic valuations. You may wish to express the valuation in terms of list price, original selling price, or final selling price. Finally, this situation may be exacerbated by having multiple fundamental quantity facts in each fact row.

Consider an example where you have 13 fundamental quantity facts, 5 unit-of-measure interpretations, and 4 valuation schemes. It would be a mistake to present only the 13 quantity facts in the fact table and then leave it up to the user or BI application developer to seek the correct conversion factors in remote dimension tables. It would be equally bad to present all the combinations of facts expressed in the different units of measure in the fact table because this would require 13 × 5 quantity facts plus 13 × 4 valuation facts, or 117 facts in each fact table row! The correct compromise is to build an underlying physical row with 13 quantity facts, 4 unit-of-measure conversion factors, and 4 valuation factors. You need only 4 unit-of-conversion factors rather than 5 because the base facts are already expressed in one of the units of measure. Your physical design now has 21 facts, as shown in Figure 6-23.

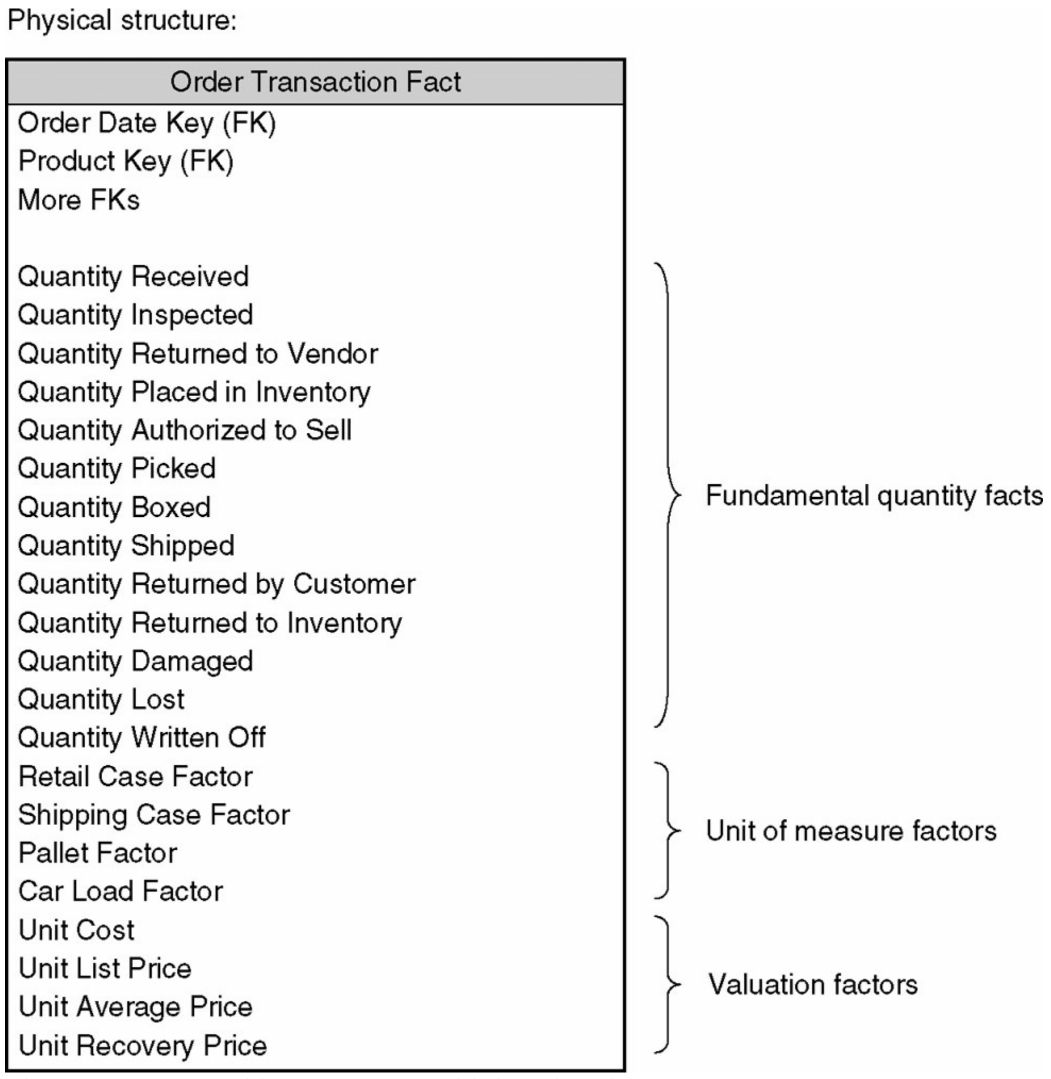


Figure 6-23: Fact table to support multiple units of measure

Sometimes designers are tempted to store the unit-of-measure conversion factors in the product dimension, however packaging these factors in the fact table reduces the pressure on the product dimension table to issue new product rows to reflect minor changes in these factors. These items, especially if they routinely evolve over time, behave much more like facts than dimension attributes.

You now deliver this fact table to the users through one or more views. The most comprehensive view could actually show all 117 combinations of units of measure and valuations, but obviously you could simplify the presentation for any specific group by only displaying the units of measure and valuation factors that the group wants to see.

Factless Fact Tables

Sometimes organizations have business processes where an event occurs, but no quantifiable measurements are created or generated. These events may be very important to the business because they represent the existence of a relationship between several dimensions, despite the fact that no counts or amounts are generated when the many-to-many collision of dimensions occurs. These factless events result in *factless fact tables*.

You can imagine a factless fact table for capturing student attendance events. You load a row into the fact in Figure 6-24 whenever a student shows up for a class. The grain is one row per class attended by a student each day.

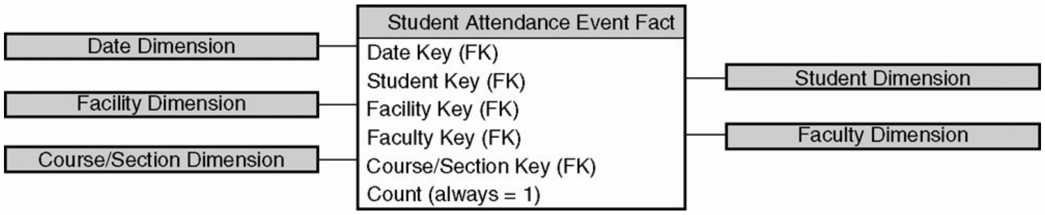


Figure 6-24: Factless fact table for student attendance events

We typically include a dummy counter fact whose value is always 1. You don't even need to physically store this fact; it merely facilitates counting to ask questions such as "how many students attended a class taught by this professor today?"

If the grain of the fact table was one row for each student enrollment (not attendance) in a class each day, then the attendance counter could take on a 1 value if the student attended class or zero otherwise. In this scenario, the fact table is no longer factless because the 1 or 0 valuation is a measurement.

Note Factless fact tables are the preferred method for recording events in a data warehouse where there is no natural numeric measurement associated with the event.

Consolidated Fact Tables

Thus far, this chapter has focused on dimensional models that are recognizable images of the operational source system supporting a single business process. In other words, if you have an orders system, you have a corresponding orders dimensional model. We recommend starting with single source dimensional models because they minimize the risk of committing to an impossible, overly ambitious implementation. Most of the risk comes from biting off too much ETL effort.

After several single source dimensional models have been implemented, it is reasonable to potentially combine them into a consolidated dimensional model. Consolidated fact tables merge measurements from multiple processes into a single fact table. Because the measurements physically sit side-by-side in a fact row, they need to be rolled up to a single common level of granularity. As shown in Figure 6-25, the consolidated actual versus forecast schema brings the measurements together for simplified variance analysis. Of course, with conformed dimensions, you could issue separate queries to a forecast fact table and a detailed fact table of actuals, and then combine the results on common dimension attributes; however doing this consolidation once in the ETL system rather than at query time will result in faster query response and a less complicated presentation.

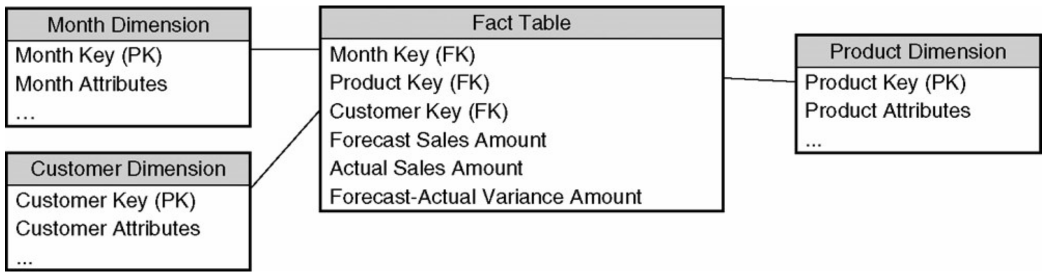


Figure 6-25: Consolidated fact table combining actual and forecast data

Just to reiterate, we suggest consolidated fact tables be built as a complement to existing detailed schema. Because you will invariably aggregate details to reach a common granularity across several processes, it's not appropriate to begin with the consolidated fact table.

Fables and Falsehoods About Dimensional Modeling

Despite the widespread industry acceptance of dimensional modeling, fables and fictitious claims continue to circulate through our industry. These false assertions are a distraction, especially if you're trying to align a team around common best practices. We'll describe the root misunderstandings that perpetuate the misleading myths, followed by specific fables and then the accurate factual response. Hopefully, you'll understand why these fables are as unfounded as fairy tales.

Fables Caused by Focusing on Departmental Reports

The first set of fables result from misunderstandings caused by focusing on departmental reports and analyses, rather than centering the design on measurement processes. Step 1 of the four-step process is to identify the business process; nowhere do we recommend specifying the top ten report layouts or queries.

Requirements analyses that focus exclusively on reports or query templates are vulnerable to modeling data to produce a specific report, rather than capturing the key metrics and related dimensions. Obviously, it's important to consider business usage when designing dimensional models. The dimension attributes must support the BI tool's filtering and labeling requirements. Robust dimension attributes translate into nearly endless slicing-and-dicing combinations. However, don't blindly focus on a top ten list in isolation because priorities and "hot" reports will inevitably evolve over time.

Note A caricature-like example of this "report-centric" approach is an organization that built more than one hundred dimensional models, using a combination of star schemas and OLAP cubes, to support order management analysis. They were practically creating a new dimensional model for each slightly different report request. Talk about an unsustainable approach!

Instead of concentrating on specific reports' requirements or departmental needs in a vacuum, we suggest focusing the dimensional design on the physical performance measurement process. In doing so, you can put the following fables to rest.

Fable: Dimensional models are built to address a specific business report or application. When the business needs a new report, another dimensional model is built.

Fact: Dimensional models should be built around physical measurement processes or events. A fact table row is created when a measurement occurs. The associated dimension attributes reflect contextual characteristics and hierarchies. If the business identifies a new report based on the same measurement process, there's absolutely no need to build a new schema. Measurement processes are relatively stable in most organizations; the analytic processes performed against these key metrics are much more fluid.

Fable: Dimensional models are departmental solutions. When a different department needs access to the data, a new dimensional model is built and labeled with the department's vocabulary, necessitating multiple extracts from the same source data repeatedly.

Fact: Dimensional models should not be departmentally bound. A fact table representing a fundamental measurement process need have only one physical instance that's shared across business functions or departments. There is no reason to create multiple extracts from the same source. Metrics resulting from the invoicing process, for example, are made available in a single dimensional model for access across the organization; there's no reason to replicate invoice performance metrics in separate departmental solutions for finance, marketing, and sales. Even if these departmental solutions were sourced from the same repository, they likely use similar, but slightly different naming conventions, definitions, and business rules, defeating the promise of a single version of the truth. The departmental approach is highly vulnerable to inconsistent, non-integrated point solutions.

Fable: You can't incorporate new data sources without rebuilding the original star schema or creating separate fact tables.

Fact: If the new data source is another capture system for an existing measurement process in the presentation server, the new data can be gracefully combined with the original data without altering any existing reporting applications, presuming the granularity is the same. If the new data source is at a different grain representing a new measurement process, a new fact table must be created. This has nothing to do with dimensional modeling; any data representation would create a new entity when a new table with different keys is introduced.

Fable: With dimensional modeling, the fact table is forced to a single grain, which is inflexible.

Fact: Having the discipline to create fact tables with a single level of detail assures that measurements aren't inappropriately double counted. A table with mixed-grain facts can only be queried by a custom application knowledgeable about the varying levels of detail, effectively ruling out ad hoc exploration. If measurements naturally exist at different grains, then the most foolproof design establishes a fact table for each level of detail. Far from being inflexible, this approach protects existing applications from breaking or recoding as changes occur.

Fables Caused by Premature Summarization

The next set of fables result from confusion about the appropriate level of detail in a dimensional model. Some people claim that dimensional models are intended for managerial, strategic analysis and therefore should be populated with summarized data, not operational details. We strongly disagree. Dimensional models should be populated with atomic data so business users can ask very precise questions. Even if users don't care about the details of a single transaction, their questions involve summarizing the details in unpredictable ways. Database administrators may pre-aggregate some information to avoid on-the-fly summarization as we describe in Chapter 8. However, these aggregates are performance-tuning complements to the atomic level, not replacements. If you create dimensional models with atomic details, the following fables are non-issues.

Fable: Dimensional models presuppose the business question and are therefore inflexible. When the requirements change, the model must be modified.

Fact: When you pre-summarize information, you've presupposed the business question. However, dimensional models with atomic data are independent of the business question because users can roll up or drill down ad infinitum. They are able to answer new, previously unspecified questions without database changes.

Fable: Dimensional models are only appropriate when there is a predictable pattern of usage. Dimensional models aren't appropriate for exploratory queries.

Fact: Both normalized and dimensional models contain the same information and data relationships; both are capable of answering exactly the same questions, albeit with varying difficulty. Dimensional models naturally represent the "physics" of a measurement event; fact tables contain the measurements and the dimension tables contain the context. A single dimensional model based on the most atomic data is capable of answering all possible questions against that data.

Fable: Dimensional models are not scalable. If detailed data is stored in a dimensional model, performance will be degraded.

Fact: Dimensional star schemas are extremely scalable. It isn't unusual for modern fact tables to have billions of rows corresponding to the billions of measurement transactions captured. Dimension tables with millions of rows are common. Dimensional models should contain as much history as required to address the business requirements. There's nothing about dimensional modeling that prohibits the storage of substantial history.

Fable: Dimensional models are not extensible and are unable to address future needs of the data warehouse.

Fact: Dimensional models that express data at the lowest level of detail deliver maximum flexibility and extensibility. Users can summarize atomic data any which way. Likewise, atomic data can be extended with additional attributes, measures or dimensions without disrupting existing reports and queries.

Fable: A dimensional model cannot support complex data. It eliminates many-tomany relationships between entities, allowing only many-to-one relationships. A dimensional model can be created from a normalized model; however, a normalized model could not be created from a dimensional model.

Fact: The logical content of dimensional models and normalized models are identical. Every data relationship expressed in one model can be accurately expressed in the other model. Dimension tables express many-to-one relationships, whereas fact tables represent many-to-many relationships.

Fables Caused by Overvaluing Normalization

Some people believe normalization solves the data integration challenge. Normalizing data contributes nothing to integration, except forcing data analysts to confront the inconsistencies across data sources.

Data integration is a process apart from any specific modeling approach. It requires identifying incompatible labels and measures used by the organization, then reaching consensus to establish and administer common labels and measures enterprise wide. In dimensional modeling, these labels and measures reside in conformed dimensions and conformed facts, respectively. As represented in the bus architecture, conformed dimensions are the integration "glue" across measurement business processes. Conformed dimensions are typically built and maintained as centralized persistent master data in the ETL system, then reused across dimensional models to enable data integration and ensure semantic consistency.

Fable: Dimensional modeling concepts like conformed dimensions put an undue burden on the ETL effort.

Fact: Data integration depends on standardized labels, values, and definitions. It's hard work both to reach organizational consensus and then implement the corresponding ETL system rules, but you can't dodge the effort, regardless of whether you're dealing with a normalized or dimensional model.

Fable: Dimensional modeling isn't appropriate when there are more than two unique source systems due to the complexities of integrating data from multiple sources.

Fact: The challenges of data integration have nothing to do with the modeling approach. Paradoxically, dimensional modeling and the bus architecture reveal the labels and measures of a business so clearly that an organization has no choice but to address the integration problem directly.

Fable: Changes to dimension attributes are only an issue for dimensional models.

Fact: Every data warehouse must deal with time variance. When the characteristic of an entity like a customer or product changes, you need a systematic approach for recording the change. Dimensional modeling uses standard techniques known as slowly changing dimensions. When normalized models step up to the issue of time variance, they typically add timestamps to the entities. These timestamps serve to capture every entity change, just like a slowly changing dimension type 2, but without using a surrogate key for each new row the query interface must issue a double-barreled join that constrains both the natural key and timestamp between every pair of joined tables, putting an unnecessary, unfriendly burden on every BI reporting application or query.

Fable: Multiple data marts can't be integrated. They're built bottoms up, catering to the needs of an individual department, not the needs of an enterprise. Chaos is the inevitable outcome.

Fact: It is definitely a struggle to integrate dimensional models built as departmental, stand-alone solutions that haven't been architected with conformed dimensions. That's precisely why we advise against this approach! Chaos won't result if you use the enterprise bus architecture's framework of conformed dimensions, then tackle incremental development based on business measurement processes. Organizational and cultural obstacles are inevitable as consistent definitions, business rules, and practices are adopted across the enterprise.

Conclusion

In this chapter we defined what dimensional modeling is and why it works. We discussed the need for standardized conformed dimensions to tie together all the dimensional models represented in the enterprise data warehouse bus matrix. Finally, we reviewed fundamental dimensional modeling design techniques. With this "what and why" grounding, you're now ready for Chapter 7, where we discuss how the dimensional modeling process and its key activities unfold.

Chapter 7: Designing the Dimensional Model

In Chapter 6 we discussed the dimensional modeling approach and outlined common dimensional design patterns. Now it's time to turn our attention to a process for building dimensional models.

We begin with a practical discussion of initial preparation activities that involve organizing the modeling team, reviewing the business requirements, determining naming conventions, setting up the modeling and data research environment, and scheduling facilities. The modeling process itself actually starts during the business requirements activity when the preliminary data warehouse bus matrix is developed. Based on the data architecture established by the bus matrix, the design team engages in initial design sessions that result in a high level model diagram. This is followed by an iterative detailed model development phase, then review and validation.

This chapter should be read by all team members, particularly the data modeler, who leads the effort. The project manager needs to understand the process to ensure that this critical piece of the project is on track. The business analyst must be comfortable with the dimensional modeling activities to help the data modeler translate the business requirements into the database design. Other team members need a basic familiarity with this topic to support the development of the dimensional model and to be prepared to carry the model forward into development of both the ETL system and business intelligence applications.

Modeling Process Overview

Creating a dimensional model is a highly iterative and dynamic process. After a few preparation steps, the design effort begins with an initial graphical model derived from the bus matrix. It identifies the scope of the design and clarifies the grain of the proposed fact tables. The initial design sessions should also identify the major dimensions applicable to each fact table, a proposed list of dimension attributes and fact metrics, and any issues requiring additional investigation.

After completing the high level model, the design team begins to assemble the model table by table and drills down into the definitions, sources, relationships, data quality concerns, and required transformations. The last phase of the modeling process involves reviewing and validating the model with interested parties, especially business representatives. The primary goals of this activity are to create a model that meets the business requirements, verify that data is available to populate the model, and provide the ETL team with a solid starting target.

Dimensional models unfold through a series of design sessions with each pass resulting in a more detailed and robust design that's been repeatedly tested against your understanding of the business needs. The process is complete when the model clearly meets the business requirements. A typical design usually requires three to four weeks for a single business process dimensional model, but the time required will vary depending on the complexity of the business process, opportunity to leverage preexisting conformed dimensions, availability of knowledgeable participants, existence of well documented detailed business requirements, and the difficulty of reaching organizational consensus.

Figure 7-1 shows the dimensional modeling process flow; the modeling process runs down the left side with the deliverables from each task on the right. The key inputs to the dimensional modeling process are the high level bus matrix and the detailed business requirements gathered during the requirements phase. The key deliverables of the modeling process are the detailed bus matrix, high level dimensional model, and detailed dimension and fact table designs.

Open table as spreadsheet

PARTICIPANT	**PURPOSE/ROLE IN MODELING PROCESS**
Data modeler	Data modeler Primary design responsibility, facilitator
Power user	Business requirements, source expert, business definitions
Business analyst	Business analysis and source expert, business definitions
Data steward	Drive agreement on enterprise names, definitions, and rules
Source system developers	Source experts, business rules
DBA	Design guidance, early learning
ETL architect and developer	Early learning
BI architect and developer	BI application requirements, early learning
Business driver or governance steering committee	Naming and business definition issue resolution, model validation

Figure 7-1: Dimensional modeling process flow diagram

Although the graphic portrays a linear progression, you should understand that the process is quite iterative. You will make many passes through the dimensional model starting at a high level and drilling into each table and each column, filling in blanks, adding more detail, and changing the design based on new information.

Note If your organization is new to dimensional modeling, consider bringing in expert assistance. An experienced dimensional modeler with strong facilitation skills can help you avoid false starts, dead end paths, and endless days spinning on key decision points. This can save weeks of effort and result in a stronger design with crisp deliverables. Insist that the consultant facilitate the modeling process with your team rather than disappearing for a few weeks and returning with a completed design. This will ensure that your entire team has participated in the design process and understands the design tradeoffs. It also leverages the learning opportunity so the team can carry the model forward.

Get Organized

Before beginning to model, it is important to take some time to prepare. This will position the design team for success and assure an effective and efficient design process. Preparation for the dimensional modeling process includes identifying the roles and participants required, reviewing the business requirements documentation, setting up the modeling environment, determining naming conventions, and obtaining appropriate facilities and supplies.

Identify Design Participants

The best dimensional models result from a collaborative team effort. No single individual is likely to have the detailed knowledge of the business requirements and the idiosyncrasies of the source systems to effectively create the model themselves. Table 7-1 identifies the roles involved in the modeling process. Usually a core modeling team of two or three people does most of the detailed work with help from the extended team. The core modeling team is led by a data modeler with strong dimensional modeling skills and excellent facilitation skills; experience with the source systems is a plus. The core team should also include a business analyst who brings a solid understanding of the business requirements and the types of analysis to be supported, along with an appreciation for making data more useful and accessible. Ideally, the core team will include at least one representative from the ETL team with extensive source systems development experience and an interest in learning. The data modeler has overall responsibility for creating the dimensional model.

We recommend including one or more analytic business users, often called *power users*, as part of the core modeling team. These power users add significant insight, helping speed the design process and improving the richness and completeness of the end result. These are the users who have figured out how to get data out of the source systems and turn it into information. They typically know how to build their own queries, sometimes even creating their own private databases. They are particularly valuable to the modeling process because they understand the source systems from a business point of view, and they've already identified and created the business rules needed to convert the data from its form in the source system to something that can be used to support the decision making process.

Table 7-1: Major Participants in the Dimensional Modeling Process
Open table as spreadsheet

PARTICIPANT	PURPOSE/ROLE IN MODELING PROCESS
Data modeler	Primary design responsibility, facilitator
Power user	Business requirements, source expert, business definitions
Business analyst	Business analysis and source expert, business definitions
Data steward	Drive agreement on enterprise names, definitions, and rules

Table 7-1: Major Participants in the Dimensional Modeling Process
Open table as spreadsheet

PARTICIPANT	PURPOSE/ROLE IN MODELING PROCESS
Source system developers	Source experts, business rules
DBA	Design guidance, early learning
ETL architect and developer	Early learning
BI architect and developer	BI application requirements, early learning
Business driver or governance steering committee	Naming and business definition issue resolution, model validation

The core modeling team works closely with source system developers to understand the contents, meaning, business rules, timing, and other intricacies of the source systems that will populate the dimensional model. If you're lucky, the people who actually built or originally designed the source systems are still around. For any given dimensional model, there are usually several source system people you need to pull into the modeling process. There might be a DBA, a developer, and someone who works with the data input process. Each of these folks does things to the data that the other two don't know about. Get to know these folks; develop good relationships with them. Help them understand how the DW/BI system will make their lives better. You will need their full cooperation.

The DBA implementing the physical databases and the ETL architect and developer should also be included in the modeling process. Being actively engaged in the design process will help these individuals better understand the business reasons behind the model and facilitate their buy-in to the final design. Often the DBA comes from a transaction system background and may not understand the rationale for dimensional modeling. The DBA may naturally want to model the data using more familiar design rules and apply third normal form design concepts to normalize the dimensions, physically defeating your dimensional design. ETL designers often have a similar tendency. Without a solid appreciation of the business requirements and justification for the dimensional design, the ETL designer will want to streamline the ETL process by shifting responsibility for calculations to the BI tool, skipping a description lookup step, or using other time saving shortcuts. Though these changes may save ETL development time, the tradeoff may be an increase in effort or decrease in query performance for hundreds of business users compensating and doing the same work countless times in a less efficient manner.

Note Expect to repeatedly assign extra computation or data conditioning steps into the ETL process as opposed to burdening the BI application.

Before jumping into the modeling process, take time to consider the ongoing management and stewardship implications of the DW/BI environment. If your organization has an active *data stewardship* initiative, it is time to tap into that function. If there is no stewardship program, it's time to initiate the process. An enterprise DW/BI effort committed to dimensional modeling as an implementation approach must also be committed to a conformed dimension strategy to assure consistency across multiple business processes. An active data stewardship program can help an organization achieve its conformed dimension strategy. Agreeing on conformed dimensions in a large enterprise can be a difficult challenge. The difficulty is usually less a technical challenge and more an organizational communication challenge. Various groups across the enterprise are often committed to their own proprietary business rules and definitions. Data stewards must work closely with all of the interested groups to develop common business rules and definitions, and then cajole the various parts of the organization into embracing the common rules and definitions to develop enterprise consensus. Over the years, some have criticized the concept of conformed dimensions as being "too hard." Yes, it's difficult to get people in different corners of the business to agree on common attribute names, definitions, and values, but that's the crux of unified, integrated data. If everyone demands their own labels and business rules, then there's no chance of delivering the single version of the truth promised by DW/BI systems. Remember, however, that you have a big safety valve that takes pressure off of the enterprise data conforming effort. The separate business units do not have to give up their private attributes and labels, rather they only need to reach agreement with the other business units on an enterprise set of conformed attributes and facts that will be the basis for data integration. If absolutely necessary, this enterprise set can be completely new.

In addition to the roles already discussed, in Chapter 9 we describe two specific ETL team responsibilities required for the ongoing management of the data warehouse environment: the dimension manager and fact provider. The individuals who own these responsibilities are not critical to creating the business process dimensional model. However, they would benefit greatly by participating in the design process. The *dimension manager* is responsible for defining, building, and publishing one or more conformed dimensions. There can be more than one dimension manager in larger organizations; however, a given dimension is the responsibility of a single dimension manager. Dimension managers obviously work closely with the data stewards. Meanwhile, the *fact provider* is responsible for designing, building, and publishing a specific fact table after receiving the dimension handoff from the dimension managers.

Although involving more people in the design process increases the risk of slowing down the process, the improved richness and completeness of the design justifies the additional overhead.

Revisit the Requirements

Before the modeling process can begin, the modeling team must familiarize itself with the business requirements. The first step is to carefully review the detailed requirements documentation. The modeling team must appreciate the challenges and opportunities the business users face as they seek to leverage data and perform analysis in response to business problems. It's the modeling team's responsibility to translate the business requirements into a flexible dimensional model capable of supporting a broad range of analysis, not just specific reports. This can be challenging. Hopefully you've been able to recruit one or two power business users as core modeling team members to help provide this expertise, in addition to the business analyst.

In Chapter 3, we described the detailed project level business requirements document that focuses on the high priority business process and related analyses. It describes the questions and problems management and business analysts are most interested in resolving. The requirements document includes a list of proposed data elements, sample questions, and desired reports that would help answer the questions. All of these are critical inputs to defining the dimensional model. Ultimately, our final dimensional model must be able to easily support these requirements.

The temptation is to move directly into design, skipping the business requirements review. Often the data modeler is an employee of significant longevity with the organization. He may feel that he clearly understands the business and its needs. Please, resist the temptation and do not to skip the business requirements review. It is all too comfortable to develop a source-driven rather than a business-driven data model. An experienced dimensional modeler can easily build a reasonable dimensional model based on the source system data structures. However, this model will inevitably fall short of meeting the business needs in many small but substantive ways. These little shortcomings add up to a weak dimensional model. What's missed is an understanding of how the business derives value from the existing data. Often, the business crafts new categorizations of data based on a combination of attributes and statuses. Uncovering and including these hidden business rules in your dimensional models is the secret sauce for delivering an environment that exceeds, rather than merely meets, business users' expectations.

Note You must understand the business requirements in detail before you dive into the task of designing the dimensional model.

Use Modeling Tools

Before jumping into the modeling process, it's helpful to have a few tools in place. Using a spreadsheet as the initial modeling tool is most effective because it allows you to quickly and easily make changes as you iterate through the modeling process. Initially, use the simple spreadsheet model shown in Figure 7-2, which is merely a subset of the more detailed dimensional design worksheet shown in Figure 7-6. As we discuss later, Figure 7-6 captures the key elements of the logical model, plus many of the physical attributes. It also gives you a place to begin capturing ETL information, such as source system table and column(s) and a brief description of the extract and transformation rules. Finally, it includes an initial set of business metadata in the form of names, descriptions, sample values, and comments.

Promotion

Open table as spreadsheet

Attribute Name	Description	Alternate Names	Sample Values
Special Offer ID	Source system key		
Special Offer	Name/description of the	Promotion name,	Volume Discount 11

Attribute Name	**Description**	**Alternate Names**	**Sample Values**
Name	Special Offer	Special offer description	to 14; Fall Discount 2006
Discount Percent	Percent item is discounted		
Special Offer Type	Description of the type of promotion, special offer or discount.	Promotion Type	Volume Discount; Discontinued Product
Special Offer Category	Channel to which the Promotion applies	Promotion Category	Reseller; Customer
Start Date	First day the promotion is available		6/15/2008
End Date	Last day the promotion is available		12/31/2008
Minimum Quantity	Minimum quantity required to qualify for the promotion		0
Maximum Quantity	Maximum quantity allowed under the promotion		NULL

Open table as spreadsheet

Attribute Name	**Description**	**Alternate Names**	**Sample Values**
SalesReasonID	Sales reason ID from source system		
Sales Reason	Reason the customer bought the product, as reported by the customer (Internet only)		Demo Event; On Promotion; Price; Review; Sponsorship
Sales Reason Type	Grouping for Sales Reason		Marketing; Promotion; Other
Channel	Channel through which the item was sold		Customer; Reseller; Field Sales

Figure 7-2: Example attribute and metrics lis

Be sure to extend the attributes and metrics list to include other columns that may be useful in your environment, like physical name (if it will be different from the logical), a short description that might easily fit into a tooltip, or a long description that fully captures the definition of the column or table.

Once you reach the later stages of the modeling process and the model begins to firm up, you should convert to whatever modeling tool is used in your organization. Most of the popular modeling tools allow you to lay out the logical model and capture physical and logical names, descriptions, and relationships. Many tools are dimensionally aware with functions to support the creation of a dimensional model.

For example, some distinguish between a fact and dimension table, handle role-playing dimensions, and create multiple business process "layers" that correspond to the rows in the bus matrix. They also capture information in their metadata stores and generally have a way to export or exchange that metadata with other systems or a centralized metadata catalog. Once the detailed design is complete, the modeling tools can help the DBA forward engineer the model into the database, including creating the tables, indexes, partitions, views, and other physical elements of the database.

Establish Naming Conventions

The issue of naming conventions will inevitably arise during the creation of the dimensional data model. It's important that the labels you apply to the data are descriptive and consistent in portraying a solid business orientation. Remember, table and column names are key elements of the DW/BI system's user interface, both when users navigate the data model and in the BI applications. Think about what each column name communicates in a report. A column name like "Description" may be perfectly clear in the context of a data model, but communicates nothing in the context of a report.

Part of the process of designing a dimensional model is agreeing on common definitions and common labels. Naming is complex because different business groups have different meanings for the same name, like *revenue*, and different names with the same meaning, like *sales*. The challenge arises because most people are reluctant to change and unwilling to give up the familiar and learn a new way. Spending time on naming conventions is one of those tiresome tasks that seem to have little payback, but is definitely worth it in the long run.

Fortunately, establishing your naming conventions doesn't have to be time consuming. Most large organizations have a function such as data management or data administration that owns responsibility for naming conventions. Leverage their work and, if possible, use the existing naming conventions that your organization has already established. Find the documentation and implement any changes necessary to make them work for the DW/BI system; sometimes existing naming conventions need to be extended to support more user-friendly table and column names.

If your organization doesn't have a set of naming conventions you can adapt to the DW/BI model, you will need to establish them as you develop the dimensional model. To begin, you need to develop an initial strategy for your naming conventions. Even if your organization hasn't established formal naming standards, it may be helpful to study the table and column names in existing systems.

A common approach is to use a column name standard with three parts:

PrimeWord_ZeroOrMoreQualifiers_ClassWord

The prime word is a categorization word that often corresponds to the entity the column is from; in some cases, qualifiers may not be necessary. For example, the field in the sales fact table that represents the amount sold might be sales dollar amount.

Next, the core modeling team, in conjunction with the data stewards, should draft an initial set of names and the rationale. During the early design review sessions, set aside time to ensure the names make sense. In addition, spend time with key stakeholders such as the core business users and any senior managers who might have an opinion. If their preferred name for a column is different from your suggested name, ask them to explain what the term means to them. Look for missing qualifiers and class words to clarify the meaning. For example, an analyst might be interested in sales performance, but it turns out that the sales number is really sales_commissionable_amount, which is different from sales_gross_amount or sales_net_amount. The resulting name set should be used to update the data model. Keep track of the alternative names for each field and the reasons people offered for their preferred choices. This will be helpful in explaining any derivations or deviations in the final name set.

During the final model review sessions with key stakeholders, seek final agreement on the naming conventions. Generally, there will have been enough model review iterations and naming discussions so that most issues have already been resolved and the remaining issues are reasonably well understood. The goal is to reach consensus. Often this means someone has to accept the will of the majority and let go of their favorite name for a given column. It is surprising how emotional this can be; people feel pretty strongly about getting the names "right."

Once you have reached agreement on the final name set, document it carefully and work it into the final data model.

Provide for Source Data Research and Profiling

An important preparation task is identifying the resources available to the core modeling team during the design process to improve their understanding of the source data. There are several useful sources for detailed information about an organization's data, including the source system itself, data experts, existing query and reporting systems, and data profiling tools. Be sure to identify these resources prior to beginning the design effort.

Throughout the modeling process, the modeler needs to develop an ever increasing understanding of the source data to learn about its structure, content, relationships, and derivation rules. You need to verify that the data exists, that it is in a usable state, or at least its flaws can be managed, and that you understand what it will take to convert it into the dimensional model.

Data profiling uses query and reporting tools to explore the content and relationships in the system under investigation. Data profiling can be as simple as writing some SQL statements or as sophisticated as a detailed data profiling study. There are tools to make the data profiling task much easier and probably more complete. The major ETL tool vendors include data profiling capabilities in their products; in addition, there are stand-alone profiling tool vendors that provide complex data analysis capabilities well beyond the realm of simple queries.

Obtain Facilities and Supplies

Often overlooked, yet important preparation tasks are the scheduling of appropriate facilities and obtaining a stash of required supplies. Over a several week period, the design team will be scheduling formal design sessions several times each day surrounded by impromptu meetings and discussions. It is best to set aside a dedicated conference room for the duration of the design effort — no easy task in most organizations where meeting room facilities are always in short supply. The meeting room should have enough space to comfortably hold the modeling participants, as well as a couple of flip chart stands. While you're dreaming, big floor-to-ceiling whiteboards on all four walls would be nice, too!

In addition to a meeting facility, the team will need basic supplies, such as flip chart paper. The self-stick variety is very helpful; otherwise a roll of masking tape will suffice for wallpapering the walls with your paper designs. Also, it would be nice if you could bust the budget for a box of new markers. (It's amazing that old, dried up markers are never thrown away; they just find their way to any conference room used by the dimensional designers.) A laptop projector is often useful during the design sessions and is required for the design reviews.

Recall the Four-Step Modeling Process

As briefly described in Chapter 6, we use a four-step process for designing dimensional data models. The process does not occur during a single design session; rather it is the overall strategy that provides context to a series of design sessions over several weeks. Before we discuss the specific design activities, it's important to understand the high level approach. The bus matrix drafted during the requirements gathering identifies the enterprise's business processes and their associated dimensions; this is critical input to the four-step process.

Step 1 — Choose the Business Process

The first step is to choose the business process to be modeled. This choice should already have been made by the business during the prioritization activity discussed in Chapter 3. This business decision normally occurs after the high level business requirements have been gathered. Selecting the initial business process is often no more difficult than picking a row on the bus matrix.

Note A good test of whether you have accurately identified a business process is to move to Step 2 and attempt to declare the grain of the fact table for the most detailed data captured by a business process. If you can succinctly state the grain of the fact table for the proposed business process, you are in good shape. However, if you have trouble determining the grain of the fact table or if there is any confusion regarding the level of detail of a single row in a fact table, you are likely mixing more than a single business process. You need to step back and look carefully for additional business processes that may be involved.

Business processes are the fundamental building block of the dimensional data warehouse. We suggest you build your data warehouse iteratively, one business process at a time. Each business process will result in at least one fact table and sometimes a set of related fact tables. Ideally, the core data captured by the initial business process you choose will come from a single source system or module.

Step 2 — Declare the Grain

The second step in creating the high level dimensional model is to declare the grain, or the level of detail in the fact table for the selected business process. Declaring the grain means saying *exactly* what a fact table measurement represents. Example declarations of the grain include:

- An individual line item on a retail sales ticket as measured by a scanner device
- An individual transaction against an insurance policy
- A line item on a bill received from a doctor
- An airline boarding pass
- An inventory measurement taken every week for every product in every store.

Notice that these grain declarations are expressed in business terms. Perhaps you were expecting the grain to be a traditional declaration of the fact table's primary key. Although the grain ultimately is equivalent to the primary key, it's a mistake to list a set of dimensions and then assume that this list is the grain declaration. In a properly executed dimensional design, the grain is first anchored to a clear business measure event. Then, the dimensions that implement that grain become obvious. Every design decision made in Steps 3 and 4 depends on correctly visualizing the grain.

Caution The most common design error is not declaring the grain of the fact table at the beginning of the design process. If the grain isn't clearly defined, the whole design rests on quicksand. Discussions about candidate dimensions go around in circles and rogue facts sneak into the design.

Choosing the grain is a combination of what is needed to meet the business requirements and what is possible based on the data collected by the source system. Every fact table design must be rooted in the realities of available physical data sources. In some situations, the business users want data at a lower level of detail than that captured by the source system. In other cases, tremendously more data is available than is required by the business requirements. We strongly recommend starting with the goal of designing the fact tables at the lowest level of detail available, that is, at the most atomic level possible. Building fact tables at the most detailed level possible is one of the cornerstones of dimensional modeling. Atomic data is the most expressive, most dimensional, and most flexible. It is always possible to sum the atomic data to higher levels of summary, but it is impossible to drill down from the summary level if the detailed data is not available.

Note While we stress the benefits of atomic data, we are also realists. Atomic data for some processes in some industries represents extreme data volumes; we understand that a less detailed approach may be warranted in isolated cases.

Despite the design team having carefully considered and declared the grain of the fact table, it is reasonable for the design team to change their minds about the grain at some point in the design process. For instance, if the grain was declared to be the daily inventory level of individual stock items in a distribution center, the design team might decide to instead track lot numbers that roll up to stock items because they discovered that the legacy systems were correctly tracking lot numbers. Adding the extra detail of lot numbers might have a significant effect on the choice of dimensions and facts in Steps 3 and 4.

Note At any point during the design process, the design team must be able to clearly articulate the grain of the target fact table.

Step 3 — Identify the Dimensions

The third step is to determine the dimensions applicable to the fact table at the stated level of granularity. Most of the major dimensions will fall out naturally once you've determined the grain. One of the powerful effects of a clear fact table grain declaration is that you can very precisely visualize the dimensionality of the fact table. All of the dimensions in the bus matrix should be tested against the grain to see if they fit. Any dimension that takes on a single value in the presence of the grain is a viable candidate.

Sometimes the fact table grain is stated in terms of primary dimensions. "Daily inventory levels of individual stock items in a distribution center" clearly specifies the date, stock item, and the distribution center dimensions. Other dimensions can quickly be tested against the grain to see if they make sense.

The grain declaration lets you think creatively about adding dimensions to a fact table that may not obviously exist in the source data. In retail sales data, marketplace causal factors like promotions and competitive effects may be very important to understanding the data, but this information may not exist in a literal sense in the source system. The grain definition "an individual line item on a retail sales ticket as measured by a scanner device" tells you that you can indeed add a causal "store condition" dimension to the fact table as long as the store condition descriptions vary appropriately by time, product, and store location. Similarly, a weather dimension can be added to the fact table design using the same logic.

Note The primary key of the fact table is almost never declared as the complete set of dimension foreign keys.

It is during this step that real world complexities begin to surface in the modeling process. What originally seemed to be a single dimension may end up being two or three dimensions. Often the design team has strong pre-conceived notions about how the data needs to be modeled based on how it's represented in the transaction system. These notions become difficult to change because the source system and all its complexities comprise the only historical window people have had on the data. In banking, for example, the account is a central organizing concept. In the source system, the account is actually a high level entity that contains several other business objects or dimensions. Once you know the account, you know the product, like checking or savings. Once you know the account, you know the customer because the account carries its own customer name and address. Once you know the account, you also know the branch because the account number includes a branch indicator. Yet in the final design, product, customer, and branch are typically their own dimensions.

Teasing out these hidden dimensions can be a challenge because they force people to rethink their understanding of the data and its purpose.

Choosing the dimensions may also cause you to rethink the grain declaration. If a proposed dimension doesn't match the grain of the fact table, either the dimension must be left out, the grain of the fact table changed, or a multi-valued design solution needs to be considered. Identification of a useful dimension that participates in the target business process may force the grain down to a lower level of detail. For example, a typical call detail record for a phone company usually includes the originating phone number and the called phone number. A cellular phone company may want to include the cell tower IDs connected to the originating and receiving phones. Because this could change many times during the phone call, what was one call record now becomes several. Adding the tower IDs forces the grain down to a lower level of detail and adds many more rows to the fact table.

Scrutinize the dimensions to make sure they make sense. Consider the impacts on usability and performance of splitting a large dimension into several dimensions or combining dimensions. Look for hidden foreign key relationships and normalized one-to-many relationship hierarchies. Ask hard questions about the assumptions behind each dimension—is it this way because it represents the true nature of the business process, or because it reflects the specific idiosyncrasies of the source system?

Step 4 — Identify the Facts

The final step in the modeling process is to identify the facts or measures from the business process. Declaring the fact table grain establishes the foundation for determining the appropriate dimensions. But declaring the grain also crystallizes the discussion about the measured numeric facts. Simply put, the facts must be true to the grain. Resist the urge to add facts that do not match the fact table grain because they usually introduce complexities and errors in the BI applications.

For many transaction-oriented business processes, there are only a few fundamental facts, like quantity and amount. There are many combinations and calculations derived from these fundamental facts that are used to monitor the business. These calculated measures are important, but are normally not part of the atomic fact table itself. At this early point in the design, you're working to identify the fundamental facts. However, keep track of all the computed facts you come up with as you'll need them soon.

Design the Dimensional Model

As with any data modeling effort, the development of the dimensional model is an iterative process. You will work back and forth between business user requirements and selected source file details to further refine the model. Always be willing to change the model as you learn more. You may identify another dimension or discover that two dimensions are really just one. Adjustments to the grain can also occur.

The process of building dimensional models typically moves through three phases, as illustrated in Figure 7-1. The first is a high level design session that defines the scope of the business process dimensional model. The second phase is detailed model development that involves filling in the attributes and metrics table by table and resolving any issues or uncertainties. The third phase is a series of model reviews, redesign, and validation steps, resulting in the final design documentation.

The entire process typically takes three to four weeks for a single business process, such as sales orders, or a couple of tightly related business processes such as health care claims, which include professional, facility, and drug claims in a set of distinct but closely aligned fact tables. More complex designs involving unrelated fact tables and numerous dimension tables will take longer, which is one reason we don't recommend them as a starting point.

We suggest the core design team schedule a series of design sessions during the model development period. Rather than trying to schedule full day design sessions, it's more beneficial to schedule a morning and afternoon session each day of two to three hour duration for three or four days each week. This approach recognizes that the design team members have other responsibilities and allows them to try to keep up in the hours before, after, and between design sessions. The unscheduled time also provides ample opportunities for the design team members to perform source data research, as well as allows time for the data modeler to iterate through the design documentation, keeping it fresh for each session.

Build the High Level Dimensional Model

The initial task in the design session is to create a high level dimensional model diagram for the target business process. Creating the first draft is straightforward because you start with the bus matrix to kickstart the design. An experienced data warehouse designer could develop the initial high level dimensional model and then present it to the design team for review. Though this is certainly possible, we strongly recommend against it because this approach does not allow the entire design team to observe and participate in the process nor will they learn how to develop a model themselves. Likewise, they will not appreciate the thought process behind the design and inevitable tradeoffs that were made. It is important to keep in mind the dual objective of most DW/BI project tasks: Get the work done and teach the team to become self-sufficient. You'll meet both goals if you work as a team to develop the model.

Conduct the Initial Design Session

The initial design session(s) should bring together the core modeling team, along with any other identified extended team members or interested participants from the source system and ETL team. The focus of this initial session is two-fold. The first is to bring the design team together for the first time to establish a common understanding of the project goals and design process and to establish an esprit de corps. To this end, it may be appropriate to spend much of the initial design meeting discussing the project objectives, scope, and roles and responsibilities. In most situations, a short introduction to dimensional modeling concepts is also appropriate before jumping into the design process itself. Secondly, you want to create the initial high level design. This session should be facilitated by the lead data modeler. It can take two design sessions or more to work through the initial model, so set expectations accordingly.

Your immediate goal is to create the high level dimensional model diagram which is a graphical representation of the dimension and fact tables for the business process. After this initial pass, you will make a quick second pass of the high level design to create an initial list of attributes for each dimension table and proposed metrics for each fact table.

The most effective method to develop the initial model is using flip charts. The lead modeler, acting as the facilitator, should work with the team to walk though the model capturing it on the flip charts. As you work through the model, you will wallpaper the conference room. Keep in mind that much of the initial modeling includes a great deal of brainstorming. Try to capture the ideas, but don't get bogged down. Until the high level graphical model is sketched out, it is difficult to access the design impact of what may seem like a simple request. As you work through the process, it is likely that the team will surface many issues and challenges that may derail the process. The facilitator needs to quickly identify these issues and park them on the issues list. At this point you are trying to very quickly lay out the high level design. There will be plenty of opportunities to talk through the details and issues later.

Document the High Level Model Diagram

The first draft of the design should be graphically summarized in a deliverable called the high level model diagram, shown in Figure 7-3. We often call this graphical representation the *bubble chart* for obvious reasons. This graphical model becomes the initial communications document for the design team. The high level model diagram is a data model at the entity level.

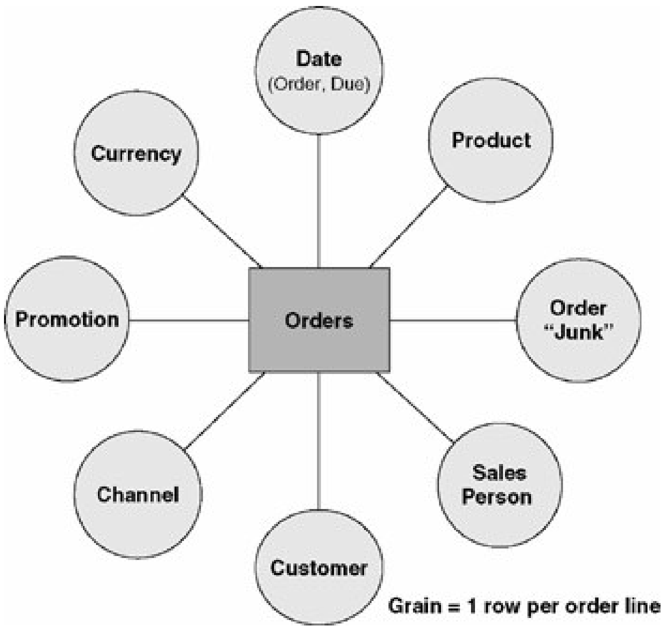


Figure 7-3: Example high level model diagram

The high level model shows the specific fact and dimension tables applicable to a given business process. It should clearly label each component, state the grain of the fact table, and show all the dimensions connected to the fact table. This graphical representation serves several purposes:

- It is a handy introduction to the design when communicating to interested stakeholders about the project, its scope, and contents.
- It facilitates model discussions between the design team and business partners. It is the jumping off point for communicating the detailed design and for supporting design reviews to ensure the business requirements can be met.
- It is a useful training tool, providing an introduction to the data model that is understandable by even the most casual business users.

To aid in understanding, it is helpful to retain consistency in the high level model diagrams. Each business process's fact table should be documented on a separate page. Think about the order of the dimensions around the fact table. We recommend that you put the date dimension at the top and work your way around the fact table in order of importance of the dimension. Once you have selected an order for the dimensions, make sure all diagrams are arranged in the same sequence. As you work with the model, people will remember the location of the dimension visually. If you change the location of a dimension from one high level model diagram to the next, they will get confused (and you probably will, too).

Each business process will likely have different dimensionality. We prefer to eliminate a dimension completely when it does not apply to a specific fact table, while leaving the location of the non-applicable dimensions blank for visual continuity.

Identify the Attributes and Metrics

After the high level dimensional model is completed, it's time to focus the design team on the initial set of dimension attributes and fact metrics. Again, keeping the four-step process in mind, the facilitator begins with the dimension tables, walking the design team through each one to identify all the relevant attributes required by the business. Following the initial pass for each dimension table, you then turn to the fact tables to identify the initial set of metrics. The deliverable of this process is called the *initial attribute and metric list*. During this process, utilize a spreadsheet as shown in Figure 7-2, creating one worksheet per table in the model and filling in its attributes or metrics.

At this point, the goal is to brainstorm. Likely, the team will identify a large number of attributes coming from a wide range of sources. That's great; just be sure not to let the team get caught up on sourcing, naming, or derivation concerns at this time. Write down the ideas and make a note of any concerns and controversies on the *issues list*. This document is helpful for remembering all the little details about the model's challenges and how you decided to resolve them.

The deliverables from these initial design sessions are the high level model diagram, the attributes and metrics list by table, and the issues list. At this point, you have identified all the dimension and fact tables, along with a proposed list of attributes and metrics. From here, it's time to jump into the detailed design process.

Develop the Detailed Dimensional Model

After completing the high level design, it's time to focus on the details. From this point forward, the core team should meet on a very regular basis to define a detailed dimensional model, table by table, column by column. It's most effective if you start with the dimension tables, and then work on the fact tables. Focus these sessions on one or two tables at a time — too many and the meeting gets bogged down. We suggest starting the detailed design process with a couple of easy, straightforward dimensions; the date dimension is always a favorite starting point. This will enable the modeling team to achieve early success, develop an understanding of the modeling process, and learn to work together as a team.

The detailed dimensional modeling process is about enriching the high level model with missing information, resolving design issues, and constantly testing the business requirements against the model to confirm completeness. The design team needs to work through each table to identify the interesting and useful attributes for each dimension and appropriate metrics for each fact table. You also need to determine the source locations, definitions, and preliminary business rules that specify how these attributes and metrics are populated.

Identify the Data Sources

An important component of the modeling process is determining the most appropriate data sources to populate the target design. Detailed data analysis is a significant part of the design effort. Several levels of analysis must occur. First, you must understand the data required to support the business requirements to help select the candidate data sources. Second, you must gain an in-depth understanding of each candidate data source, typically through a data profiling process. Part of the challenge of developing the dimensional model is aligning the business requirements with the unfortunate realities of the data available in the enterprise.

From the business requirements, you can identify both formal and informal data sources. The formal sources are supported by IT and will have some level of rigor associated with the maintenance and integrity of the data. You can expect to apply a more thorough data analysis against these formal sources. Informal data comes directly from the business and is often important to the decision making process. Although the informal data is not stored in the corporate data vaults, business users will want and need to integrate the informal data with the formal data sources.

Informal data sources are often included in the data warehouse through the development of a new IT process to capture this data. This may involve simply importing data from a shadow database or developing a user entry system for this data. In both cases, the input data needs to be treated as another data source. This means the data must flow through the rigors of the ETL process to be cleaned and manipulated as you would any other data source.

Note Do not simply allow users to input data directly into the data warehouse.

Understand Candidate Data Sources

After you have created the high level design, but before you can progress very far in creating detailed table designs, you need to make decisions about the data source for each table. The first step in this process is to understand all of the sources that are candidates for populating your models. Some candidate data sources may be identified in the business requirements document. Others may be implied based on the business processes and dimensions identified in the bus matrix. Additional candidate sources can be identified by the IT veterans who are familiar with the data lurking in the various nooks and crannies of the organization. They also know about the history and challenges of these sources. Although it may seem very basic to pull together a list of candidate sources with descriptive information about each, this consolidated list does not exist in a single place for most organizations today. Figure 7-4 shows a sample data source definition.

Open table as spreadsheet

Source	**Business Owner**	**IS Owner**	**Platform**	**Location**	**Description**
Gemini	Tom Owens	Alison Jones	Unix	HQ Chicago	Distribution center inventory
Billings	Craig Bennet	Steve Dill	MVS	MF Dallas	Customer Billings
Plants	Sylvia York	Bob Mitchell	Unix	6 Plants across country	Plant shipments
Sales Forecast	Sandra Phillips	None	Windows	HQ Sales Dept	Spreadsheet-based consolidated sales forecast
Competitor Sales	Sandra Phillips	None	Windows	HQ Sales Dept	Access database containing data from external supplier

Figure 7-4: Sample data source definitions

Profile and Select the Data Sources

From this list of candidate data sources, the design team needs to evaluate each and determine the best source. Gather and carefully review whatever documentation is available for the source systems, such as data models, file definitions, record layouts, written documentation, and source system programs. More advanced source systems may have their own metadata repositories with all this information already integrated for your convenience.

Perusing the source system data itself usually provides a quick jolt of reality. First, what you see typically does not match the documentation you carefully gathered and reviewed. Second, it is usually more difficult to unravel than you would hope. The older the system, the more time it's had to evolve. It may be advantageous to meet with the source system owners and core business users that have worked with data from the candidate data sources.

Data profiling is the systematic analysis of the source content, from counting the bytes and checking cardinalities to the most thoughtful diagnosis of whether the data can meet the goals of the DW/BI system. Detailed data profiling should be undertaken for every data source that is to be included in this iteration of the data warehouse. Results of the data profiling activities can help in the selection of the best source systems.

Profiling analysis is divided into a series of tests, starting with individual fields and ending with whole suites of tables comprising extended databases. Individual fields are checked to see that their contents agree with their basic data definitions and domain declarations using column screens described in Chapter 9. It is especially valuable to see how many rows have null values or contents that violate the domain definition. For example, if the domain definition is "telephone number," then alphanumeric entries clearly represent a problem. Moving beyond single fields, data profiling then uses structure screens to investigate the relationships between fields in the same table. Fields implementing a key to the data table can be displayed, together with higher level many-to-one relationships that implement hierarchies. Checking for unique key values is especially helpful because the violations (duplicate instances of the key field) are either serious errors, or reflect a business rule that has not been enforced in the source data.

Relationships between tables are also checked in the data profiling step, including assumed foreign key to primary key relationships and the presence of parents without children. Finally, data profiling can check complex business rules unique to the enterprise, such as verifying that all the preconditions have been met for approval of a major funding initiative. These last tests are called *business rule screens*.

The primary outcomes of the data profiling process include the following:

- Basic "go/no go" decision on the candidate data source under consideration. Data profiling may reveal that the data from the source system simply does not contain the information to support the business requirements. In some cases, this knowledge can derail the entire project.
- Source system data quality issues that must be corrected before the project can proceed. These corrections create a huge external dependency that must be well managed for the DW/BI system to succeed.
- Data quality issues that can be corrected in the ETL processing flow after the data has been extracted from the source system. These issues should be captured so the ETL development team can design the transformation logic and exception handling mechanisms required. But don't get your hopes up. Relatively few data quality errors can be decisively corrected by the ETL system.
- Unanticipated business rules, hierarchical structures, and foreign keyprimary key relationships. Understanding the data at a detailed level flushes out issues that may need to be incorporated into the dimensional data model.

From this analysis, a decision needs to be made regarding the best source system to populate the dimensional design. The selection of the primary data source is often driven by the business. Suggested criteria to consider include:

- **Data accessibility.** If two possible feeds exist for the data, one stored in binary files maintained by a set of programs written before the youngest project team member was born and the other from a system that already reads the binary files and provides additional processing, the decision is obvious. Use the second option for your source data.

- **Longevity of the feed.** Often data for the warehouse could be intercepted at many places throughout a processing stream. If the data coming out of step 5 is the cleanest, but that step is scheduled to be phased out in the next six months, then consider extracting from step 4. Understand the additional processing that is done in step 5 that may clean the data because you may need to include this in the ETL system.
- **Data accuracy.** As data is passed from system to system, modifications are made. Sometimes data elements from other systems are added; sometimes existing elements are processed to create new elements and other elements are dropped. Each system performs its own function well. However, it may become difficult or impossible to recognize the original data. In some cases, the data no longer represents what the business wants for analysis. If you provide the data from these downstream systems, the users may question the accuracy. In this case, it may be better to go back to the original source to ensure accuracy. However, keep in mind that pulling data from downstream systems is not always bad.

Note You will want to document any sources that were considered and not selected. This documentation will be helpful months later when questions may arise about the most appropriate source of the data.

During this period, the core design team is pestering the source system experts with questions and utilizing data profiling capabilities to understand the source data and determine its fitness for inclusion in the model. Your goal is to make sure the data exists to support the dimensional model and identify all of the business rules and relationships that will impact the model.

Establish Conformed Dimensions

During these detailed design sessions, the key conformed dimensions are being defined. Because the DW/BI system is an enterprise resource, these definitions must be acceptable across the enterprise. The data stewards and business analysts are key resources that should reach out to achieve organizational consensus on table and attribute naming, descriptions, and definitions. The design team can take the lead in driving the process, but it is ultimately a business task to agree on standard business definitions and names for the design. This will take some time, but it is an investment that will provide huge returns in terms of users' understanding and willingness to accept the dimensional model. Don't be surprised if folks in the front office, like the business sponsor/driver or governance steering committee, must get involved to resolve conformed dimension definition and naming issues.

Some dimension attribute information may come with the transactional event data, but it is usually minimal and often only in the form of codes. The additional attributes that the users want and need are often fed from other systems or from a master data management (MDM) system. For some dimension tables, there can be multiple sources, especially for the customer dimension. There are often separate non-integrated tables that are used across an organization; the sales, marketing, and finance departments may each have their own customer master data. There are two difficult issues when this happens. First, the customers who are included in these sources may differ and the attributes about each customer may differ. To an extent, this is good because it provides richer information. However, the common information may not match. Assuming you have adequate time and resources, you should work to combine and integrate these sources into a single conformed dimension with clear business rules for populating it with common conformed values.

Identify Base Facts and Derived Facts

The data profiling effort will identify the counts and amounts generated by the measurement event's source system. However, fact tables are not limited to just these base facts. There may be many more metrics that the business wants to analyze that are derived from the base facts, such as year-to-date sales, percentage difference in claims paid versus prior year, and gross profit. You need to document the derived facts, as well as the base facts.

There are two kinds of derived facts. Derived facts that are additive and can be calculated entirely from the other facts in the same fact table row can be shown in a user view as if they existed in the real data. The user will never know the difference.

The second kind of derived fact is a non-additive calculation, such as a ratio or cumulative fact that is typically expressed at a different level of detail than the base facts themselves. A cumulative fact might be a year-to-date or month-to-date fact. In any case, these kinds of derived facts cannot be presented in a simple view at the DBMS level because they violate the grain of the fact table. They need to be calculated at query time by the BI tool. Most BI tools today offer a library capability to develop and maintain derived facts. This is where the official formulas and calculations are defined and then used by everyone.

Defining the full range of metrics can be a challenging political process. You may be defining facts that will become metrics in organizational incentive plan calculations, so managers tend to have strong opinions about their definitions. Also, the broader the audience, the more opinions you need to consider, and the greater the potential divergence among them. Engage the data stewards to help in this process.

Though the derived facts do not need to be completed prior to moving forward with the design of the ETL process, it is important to ensure that all the base facts are accounted for. The dimensional model is not complete until all the base facts required to support the calculation of the necessary derived facts have been identified.

When the business users have specific ideas about what they want and need, the derived facts can be identified during a dedicated design session. In some cases, the users may not have concrete ideas, but you still need to develop a starting set of derived facts. Keep in mind that the initial list of facts is not definitive — once the users get their hands on the DW/BI system, they will want more. You will need to decide who has the authority to add new derived facts to the library.

Note Too often, the dimensional design is completed with little thought toward the derived facts required until the team is ready to begin developing the BI applications. Unfortunately, this is late in the process. Often there are dozens and perhaps hundreds of business calculations required that need to be defined and consensus developed. Likely, this could have been readily accomplished during the initial design process when all of the interested stakeholders were already gathered together working through the preliminary design.

If you have a small number of derived facts, simply use the dimensional design worksheet to capture information regarding the derived facts. For each derived fact, you will need to capture the base facts that are required to support the derived fact and the mathematical formula for deriving the fact. If you have a large number of derived facts, it may be beneficial to capture and document them separately in a document such as the derived fact worksheet shown in Figure 7-5.

Chg Flag	**Fact Group**	**Measure Name**	**Measure Description**	**Agg Rule**	**Formula**	**Constraints**	**Transformations**
	POS	$ Sales	The dollar amount of the goods sold through the retail channel.	Sum	Sum (Dollar Sales)	None	None
	POS	Total US $ Sales	The dollar amount sold for the total US geography.	Sum	Sum (Dollar Sales)	Geography = Total US	None
	POS	% of Total US $	Dollar sales as a percentage of total US geography.	Recalc	($ Sales/ Total US $ Sales) * 100	NA	NA
	POS	Prev. $ Sales	The dollar amount of the goods sold through the retail channel during the previous period.	Sum	Sum (Dollar Sales)	None	Previous period
	POS	Prev. Tot US $ Sales	The dollar amount sold for the total US during the previous period.	Sum	Sum (Dollar Sales)	Geography = Total US	Previous period
POS	Prev.	The	Recalc	(Prev $	Tot US $	NA	NA

Chg Flag	Fact Group	Measure Name	Measure Description	Agg Rule	Formula	Constraints	Transformations
	% of Total US $	previous period dollars as a percentage of the previous period total US dollars.		Sales/ Prev	Sales) * 100		
	POS	$ Chg vs. Prev	The actual change in dollars from previous period	Sum	$ Sales — Prev $ Sales	NA	NA
	POS	Units	The number of consumer units sold.	Sum	Sum (Units)	None	None
	POS	Avg Retail Price	The average price at the register.	Recalc	$ Sales/ Units	None	None
	Inv	Inventory $	The dollar value of units in inventory.	Sum w/Limit	Sum (inv Units) expect across time, then take value from max date.		
	Fcst	Forecast $	The dollar amount of the expected sales through the retail sales channel.	Sum	Sum (Forecast Dollars)	None	None
	Multi Group	% Var to Forecast $	The percentage difference between actual and forecast sales dollars.	Recalc	(Sum (Dollar Sales) Sum (Forecast Dollars))/Sum (Dollar Sales)	None	None

Figure 7-5: Example derived fact worksheet

Document the Detailed Table Designs

The attribute and metrics list discussed earlier and shown in Figure 7-2 is used as a working deliverable in the early phases of the detail design process. This tool provides an easy place to capture the basic information about each table and column. We prefer this spreadsheet tool because it allows the data modeler to quickly move or copy attributes around the model as needed, so the documentation is efficiently updated between modeling sessions. Providing freshly updated documentation for each design session helps move the process along more quickly because there is less time spent rehashing changes already agreed upon. We usually use the simplified attributes list during the early design discussions.

The key deliverable of the detailed modeling phase is the detailed design worksheet, shown in Figure 7-6. The purpose of this design documentation is to capture the details of the design for communication to other interested stakeholders including the business users, BI application developers, and very importantly, the ETL developers who will be tasked with populating the design. The final dimensional design worksheet is the first step toward creating the source-to-target mapping document. The physical design team will further flesh out the mapping. Once the model is stable, it is appropriate to move the model from the spreadsheet tool into your organization's data modeling tool of choice.

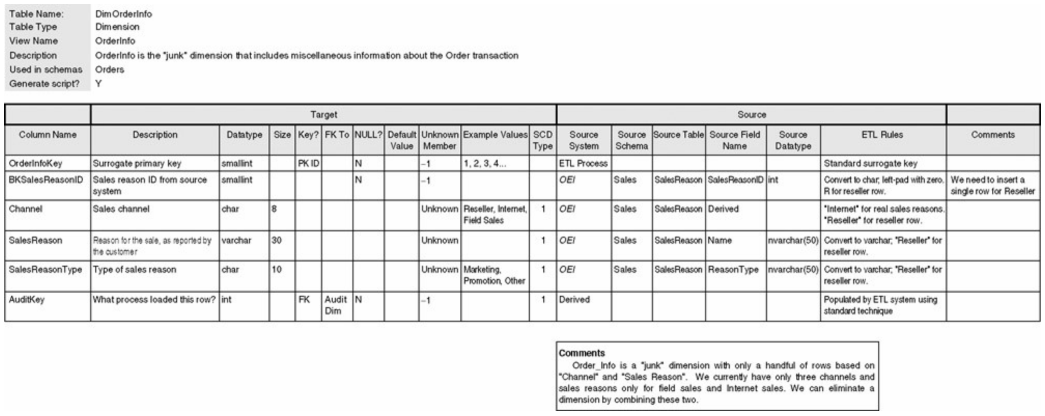


Table Name:	DimOrderInfo
Table Type	Dimension
View Name	OrderInfo
Description	OrderInfo is the "junk" dimension that includes miscellaneous information about the Order transaction
Used in schemas	Orders
Generate script?	Y

	Target										Source						
Column Name	Description	Datatype	Size	Key?	FK To	NULL?	Default Value	Unknown Member	Example Values	SCD Type	Source System	Source Schema	Source Table	Source Field Name	Source Datatype	ETL Rules	Comments
OrderInfoKey	Surrogate primary key	smallint		PK ID		N		-1	1, 2, 3, 4...		ETL Process					Standard surrogate key	
BKSalesReasonID	Sales reason ID from source system	smallint				N		-1			*OEI*	Sales	SalesReason	SalesReasonID	int	Convert to char; left-pad with zero. R for reseller row.	We need to insert a single row for Reseller
Channel	Sales channel	char	8					Unknown	Reseller, Internet, Field Sales	1	*OEI*	Sales	SalesReason	Derived		"Internet" for real sales reasons. "Reseller" for reseller row.	
SalesReason	Reason for the sale, as reported by the customer	varchar	30					Unknown		1	*OEI*	Sales	SalesReason	Name	nvarchar(50)	Convert to varchar; "Reseller" for reseller row.	
SalesReasonType	Type of sales reason	char	10					Unknown	Marketing, Promotion, Other	1	*OEI*	Sales	SalesReason	ReasonType	nvarchar(50)	Convert to varchar; "Reseller" for reseller row.	
AuditKey	What process loaded this row?	int		FK	Audit Dim	N		-1		1	Derived					Populated by ETL system using standard technique	

Comments
Order_Info is a "junk" dimension with only a handful of rows based on "Channel" and "Sales Reason". We currently have only three channels and sales reasons only for field sales and Internet sales. We can eliminate a dimension by combining these two.

Figure 7-6: Example detailed dimensional design worksheet

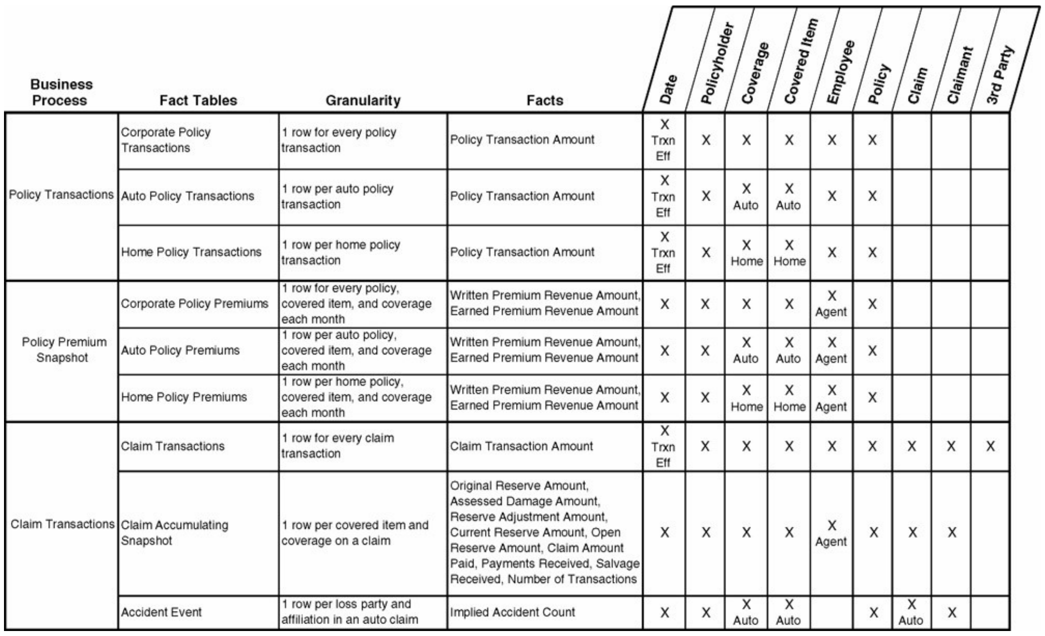
Each dimension and fact table should have a separate detailed table design. At a minimum, the supporting information required includes the attribute/fact name, description, sample values, and a slowly changing dimension type indicator for every dimension attribute. In addition, the detailed fact table design should identify each foreign key relationship, appropriate degenerate dimensions, and rules for each fact to indicate whether it's additive, semi-additive, or non-additive. The detailed table design should also capture attributes and facts that are desired by the users, but are not available or are outside the initial project scope. They should be clearly labeled as not available. Capturing these additional requirements in the design allows you to clearly communicate that you understood the requirement, while helping manage the business users' expectations.

Any issues, definitions, transformation rules, and data quality challenges discovered during the detailed design process should be captured. Also, any special design techniques employed such as junk dimensions, minidimensions, or bridge tables should be identified.

Tip In addition to the physical database attributes, much of what you learn about the data during the modeling process is not part of the logical model at all, but it is important to the ETL process. Capture this information in the dimensional design worksheet to save the ETL team from reinventing the wheel. Having someone from the ETL team participate in the data modeling sessions can prove valuable in identifying the most pertinent issues.

Update the Bus Matrix

During the detailed modeling process, there is often new learning about the business process being modeled. Frequently, these findings result in the introduction of new fact tables to support the business process, new dimensions, or the splitting or combining of dimensions. It is important to keep the initial bus matrix updated throughout the design process because the bus matrix is a key communication and planning device that will be relied upon by project sponsors and other team members. It also serves as a communication tool for the design team in discussions with other designers, administrators, and business users. The matrix is very useful as a high level introduction to the overall DW/BI system design and implementation roadmap. The matrix shown in Figure 7-7 is an illustration of a more detailed bus matrix that has been updated and extended by the design team to communicate additional information about the associated fact table's granularity and metrics.



Business Process	Fact Tables	Granularity	Facts	Date	Policyholder	Coverage	Covered Item	Employee	Policy	Claim	Claimant	3rd Party
Policy Transactions	Corporate Policy Transactions	1 row for every policy transaction	Policy Transaction Amount	X Trxn Eff	X	X	X	X	X			
	Auto Policy Transactions	1 row per auto policy transaction	Policy Transaction Amount	X Trxn Eff	X	X Auto	X Auto	X	X			
	Home Policy Transactions	1 row per home policy transaction	Policy Transaction Amount	X Trxn Eff	X	X Home	X Home	X	X			
Policy Premium Snapshot	Corporate Policy Premiums	1 row for every policy, covered item, and coverage each month	Written Premium Revenue Amount, Earned Premium Revenue Amount	X	X	X	X	X Agent	X			
	Auto Policy Premiums	1 row per auto policy, covered item, and coverage each month	Written Premium Revenue Amount, Earned Premium Revenue Amount	X	X	X Auto	X Auto	X Agent	X			
	Home Policy Premiums	1 row per home policy, covered item, and coverage each month	Written Premium Revenue Amount, Earned Premium Revenue Amount	X	X	X Home	X Home	X Agent	X			
Claim Transactions	Claim Transactions	1 row for every claim transaction	Claim Transaction Amount	X Trxn Eff	X	X	X	X	X	X	X	X
	Claim Accumulating Snapshot	1 row per covered item and coverage on a claim	Original Reserve Amount, Assessed Damage Amount, Reserve Adjustment Amount, Current Reserve Amount, Open Reserve Amount, Claim Amount Paid, Payments Received, Salvage Received, Number of Transactions	X	X	X	X	X Agent	X	X	X	
	Accident Event	1 row per loss party and affiliation in an auto claim	Implied Accident Count	X	X	X Auto	X Auto		X	X Auto	X	

Figure 7-7: Example detailed bus matrix

Identify and Resolve Modeling Issues

It is important to capture every data related issue as it arises to keep track of the issue and its resolution. It would not be unusual for the high level model to change during the detailed design as additional information is uncovered. Source system analysis may uncover opportunities or constraints not known earlier or design decisions may result in more or fewer dimensions. Deeper analysis may even result in changes to the fact table grain or spawn new fact tables. It's also possible that new business requirements will surface and need to be addressed.

The design session facilitator should reserve adequate time at the end of every design session to review and validate new issue entries and their assignments. Assign someone to track and capture issues for the issue list in every meeting. We often see this role become the responsibility of the project manager, who is typically adept at keeping the list updated and encouraging progress on resolving open issues, if they're participating in design sessions. Figure 7-8 is an example issues list that is useful for tracking issue minutia during the process.

Open table as spreadsheet

Chng Flag	**Issue #**	**Task/Topic**	**Issue**	**ID Date**	**Rptd By**	**Resp**	**Date Closed**	**Status**	**Priority**
	27	Employee	Research availability of historical data.	2/18/2008	Team	BH	-	Open	High
	28	Sales Territory	Research relationships and history among Sales Territory, Sales Rep, and Customer tables in the source system.	2/18/2008	Team	RB	-	Open	High
	29	Customer	Assess projected impact of combined Field Sales, Internet and Reseller on data size and growth of Customer table.	2/18/2008	Team	BB	-	Open	High
	30	Customer	Verify understanding of and support for	2/18/2008	Team	BB	-	Open	High

Chng Flag	Issue #	Task/Topic	Issue	ID Date	Rptd By	Resp	Date Closed	Status	Priority
			combined Customer table with core users.						
	31	Promotions	Discuss special offer / promotions with marketing to understand how these might change	2/18/2008	Team	LB	-	Open	High
	32	Order_Info	Verify concept of Channel with Marketing and Sales.	2/18/2008	Team	CK	-	Open	High
	33	Order_Info	Verify usability of combined Channel and Sales Reason fields in the same table.	2/18/2008	Team	CK	-	Open	High
	34	Order_Info	Determine list of possible combinations between Channel and Sales Reason.	2/18/2008	Team	CK	-	Open	High

Figure 7-8: Example data model issues list

Between design sessions, the core design team is typically busy profiling data, seeking clarification and agreement on common definitions, and meeting with source system experts to resolve data challenges. As the design matures, these activities tend to become very focused on the remaining open issues. Some issues may be outside the design team's sphere of influence or responsibility. These issues need to be reviewed closely and escalated to the appropriate person for closure. Do not be afraid to solicit additional assistance from the project manager or even the project sponsor. In many cases, the primary obstacle to getting closure is freeing up enough time to work on the issues. The project manager may need to work with both IT and business management to get the required time allocated. Make sure that management understands what is at stake — if the modeling process is delayed, the entire project will be delayed.

Review and Validate the Model

Once you're confident in the model, the process moves into the review and validation phase identified in Figure 7-1. This phase involves reviewing the model with successive audiences, each with different levels of technical expertise and business understanding to get feedback from interested people across the organization. At a minimum, the design team should plan on talking to three groups:

- Source system developers and DBAs who can often spot errors in the model very quickly
- Core business or power users who were not directly involved in the model development process
- Finally, the broader user community.

The core modeling team will get valuable feedback from the review and validation process. The greater DW/BI team also benefits from these reviews in the form of a more informed and engaged business user community. Feedback from these sessions should be incorporated into the final design documentation.

Of course, throughout the design process, the modeling team should occasionally lift itself from the detailed design activities to step back and review the overall design within the team. This provides a forum for reviewing decisions and exploring alternative ways to model the data.

Perform IT Data Model Review

Typically the first public design review of the detailed dimensional model is with your peers in the IT organization. This audience is often comprised of reviewers who are intimately familiar with the target business process because they wrote or manage the system that runs it. They are also at least partly familiar with the target data model because you've already been pestering them with source data questions. Good preparation and a carefully orchestrated presentation will help make this a successful session.

Note These IT folks know a lot about the transaction system and the business processes it represents. The IT data model review may require more than one session if you have more than 8 or 10 dimensions to cover or if your audience is particularly interactive.

Preparation

The IT review can be challenging because the reviewers often lack an understanding of dimensional modeling. In fact, most of them probably fancy themselves as pretty proficient third normal form modelers. Their tendency will be to apply transaction processing-oriented modeling rules to the dimensional model. Rather than spending the bulk of your time debating the merits of different modeling disciplines, it is best to be prepared to provide some dimensional modeling education as part of the review process.

Session Flow

Start the session with a review of basic dimensional modeling concepts. Don't spend too much time on this, but make sure everyone knows that analytics are different from transactions and therefore need a different type of data model. Make sure they know the differences between fact and dimension tables and that dimensions are denormalized for good reason.

When everyone has the basic concepts down, begin with a review of the bus matrix. This will give everyone a sense for the project scope and overall data architecture, demonstrate the role of conformed dimensions, and show the relative business process priorities. Next, illustrate how the selected row on the matrix translates directly into the high level dimensional model diagram, like the example in Figure 7-3. This gives everyone the entity-level map of the model and serves as the guide for the rest of the discussion.

Most of the review session should be spent going through the dimension and fact tables one by one. This group can handle the spreadsheet version of the detailed data model, so make copies of all the detailed worksheets.

It is also a good idea to review any remaining open issues for each table as you go through the model. Often someone in the group has the answer, or at least knows where to find it. Discuss the availability of dimension change history for all type 2 slowly changing dimension attributes and for the fact table itself — where is this historical data and how can it be extracted?

Several changes to the model will come out of this meeting. Remember to assign the task of capturing the issues and recommendations to someone on the team.

Review with Core Users

In many projects, this review is not required because the core business users are members of the modeling team and are already intimately knowledgeable about the data model. Otherwise, this review meeting is very similar in scope and structure to the IT review meeting. The core business users are more technical than typical business users and can handle more detail about the model. Often, especially in smaller organizations, we combine the IT review and core user review into one session. This works if the participants already know each other well and work together on a regular basis.

Present to the Business Users

This session is as much education as it is design review. You want to educate people without overwhelming them, while at the same time illustrating how the dimensional model supports their business requirements. In addition, you want them to think closely about how they will use the data so they can help highlight any shortcomings in the model.

Create a presentation that follows the same outline as the IT review. Start with basic dimensional concepts and definitions, and then describe the bus matrix as your enterprise DW/BI data roadmap. Review the high level model diagram, and finally, review the important dimensions, like customer and product.

We typically use the high level model diagram to present the individual dimension tables to the users rather than the detailed design worksheets. Certainly the high level model in Figure 7-3 is a great place to start. You will find that the model makes sense to the users once they understand how to interpret the diagrams.

Allocate about a third of the time to illustrate how the model can be used to answer a broad range of questions about the business process. Pull some interesting examples from the requirements document and walk through how they would be answered. The power users will get this immediately. Reassure the rest of your audience that most of this complexity will be hidden behind a set of easy-to-use structured reports. The point is to show you can answer just about every question they might ask about this business process.

There are usually only minor adjustments to the model once you get to this point. After working so hard to develop the model, the users may not show what you consider to be appropriate enthusiasm; they may not ooh and aah. The model seems obvious to the users and makes sense; after all, it is a reflection of their business. This is a good thing; it means you have done your job well! The oohs and aahs will come when the business users see the real data for the first time.

Finalize the Design Documentation

You can consider yourself finished with the dimensional modeling design effort when you have exhausted the issues list or the project team, whichever comes first. Realistically, the project team will be exhausted long before the dimensional model is complete. The most important issues are the model's ability to support the business requirements and the identification of source data that can reasonably populate the model.

As with other data modeling, you could spend months trying to track down every single issue. It is simply not feasible to take that much time. Over the years, it has been proven that the last several issues can take as much time to resolve as the first 50 or 60, and the final result has minimal actual business impact. Once most of the issues have been addressed, the remaining should be prioritized. As time goes on, you will find that more and more of the issues are either resolved or determined to be future enhancements. If you are waiting on input from other people to continue, then you must escalate this to the project manager because it can impact the entire project schedule.

Once the model is in its final form, the design documentation should be compiled. This document typically includes:

- Brief description of the business process(es) included in the design.
- High level discussion of the business requirements to be supported pointing back to the detailed requirements document.
- High level data model diagram.
- Detailed dimensional design worksheet for each fact and dimension table.
- Open issues list highlighting the unresolved issues.
- Discussion of any known limitations of the design to support the project scope and business requirements.

- Other items of interest, such as design compromises or source data concerns.

Because the design team has been capturing source profiling information, data issues, and transformation rules on the detailed dimensional design worksheet (Figure 7-6), it forms the basis for the project's source-to-target mapping document. The source-to-target map describes how each target table and column in the data warehouse will be populated. It's one of the key recipes used by the ETL kitchen developers. The mapping outlines *what* needs to happen, without going into the details of *how*.

The final detailed dimensional design worksheet captures everything the modeling team learned about the data during the design process, along with guidance about valid values, decodes, slowly changing dimension rules, and more complex transformations. Information about the target tables is further embellished during the physical design process, as we describe in Chapter 8, including physical names, data types, and key declarations. Ultimately, the details about the target data structures are completely fleshed out in the final source-to-target map, which is the key hand-off from the project's modeling phase to the ETL design phase detailed in Chapters 9 and 10.

Embrace Data Stewardship

We've been focused on the tactics of the dimensional design process, but we can't leave this topic without getting on our soapbox again about the importance of establishing a data stewardship program in your organization. Achieving data consistency is a critical objective for most DW/BI programs. Establishing responsibility for data quality and integrity can be extremely difficult in many organizations. Most operational systems effectively capture key operational data. A line in the order entry system will typically identify a valid customer, product, and quantity. Optional fields that may be captured at that point, such as user name or customer SIC code, are not usually validated, if they get filled in at all. Operational system owners are not measured on the accuracy or completeness of these fields; they are measured on whether or not the orders get taken, filled, and billed. Unfortunately, many of these operationally optional fields are important analytic attributes. Quality issues or missing values become significantly magnified under the scrutiny of hundreds of analytic business users with high powered query tools.

Identifying and dealing with these issues requires an organizational commitment to a continuous quality improvement process. Establishing an effective data stewardship program is critical to facilitating this effort. The primary goal of a data stewardship program is the creation of corporate knowledge about its data resources to provide legible, consistent, accurate, documented, and timely information to the enterprise. Stewardship is also tasked with ensuring that corporate data asset is used correctly and to its fullest extent, but only by those individuals authorized to leverage the data.

Lack of consistent data across the organization is the bane of many DW/BI system efforts. Data stewardship is a key element in overcoming consistency issues. Unfortunately, you can't purchase a wonder product to create conformed dimensions and miraculously solve your organization's master data management issues. Defining master conformed dimensions to be used across the enterprise is a cultural and geopolitical challenge. Technology can facilitate and enable data integration, but it doesn't fix the problem. Data stewardship must be a key component of your solution.

In our experience, the most effective data stewards come from the business community. As with technology, the DW/BI team facilitates and enables stewardship by identifying problems and opportunities and then implementing the agreed upon decisions to create, maintain, and distribute consistent data. But the subject matter experts in the business are the ones rationalizing the diverse business perspectives and driving to common reference data. To reach a consensus, senior business and IT executives must openly promote and support the stewardship process and its outcomes, including the inevitable compromises.

Conclusion

Dimensional modeling is an iterative design process requiring the cooperative effort of people with a diverse set of skills. The design effort begins with an initial graphical model pulled from the bus matrix and presented at the entity level. This model is created and critically scrutinized in an early set of high level design sessions that also yield an initial list of attributes or metrics for each table and issues requiring additional investigation. At this point, the detailed modeling process takes the model, table by table, and drills down into the definitions, sources, relationships, data quality problems, and required transformations. The primary goals are to create a model that meets the business requirements, verify that the data is available to populate the model, and provide the ETL team with a solid starting point and clear direction.

The task of determining column and table names is interwoven into the design process. The organization as a whole must agree on the names, definitions, and derivations of every column and table in the dimensional model. This is more of a political process than a technical one, and will require the full efforts of your most diplomatic team member.

The detailed modeling process is followed by several reviews. The three most common review sessions involve interested members of the IT organization, the more technically sophisticated business users, and the general business user community. This last session is as much an educational event as it is a review.

The end result is a dimensional model that has been tested against the business needs and the data realities and found capable of meeting the challenge.

BLUEPRINT FOR ACTION

Managing the Effort and Reducing Risk

Follow these best practices to keep the dimensional modeling effort and risk in check:

- Assure the data modeler leading the design process is an expert dimensional modeler. If not, consider supplementing with an outside resource.

- Every member of the design team must thoroughly and completely understand the business requirements.
- Be sure to include power users as part of the design team.
- Continuously probe the proposed source data with your data profiling tools to assure the data required to support the proposed data model is available.

Assuring Quality

Key steps to assure quality include:

- Adhere to dimensional modeling best practices.
- Insist on active participation of power users in the design process.
- Conduct extensive data profiling.
- Do not skip the IT and user review sessions.

Key Roles

Key roles for designing the dimensional model include:

- The data modeler leads the dimensional modeling efforts.
- The business analyst, power users, and BI application developers represent the business users' analytic needs.
- Data stewards help drive to organizational agreement on the dimensional model's names, definitions, and business rules.
- Source system experts bring knowledge of the operational systems.
- The ETL team learns about the sources, targets, and gets a sense of the heavy lifting they'll need to do to convert from one to the other.
- Interested parties in the IT and broader business communities will review and provide feedback on the design.
- The database and data access tool vendors provide principles for database design to optimize their products.

Key Deliverables

Key deliverables for designing the dimensional model include:

- High level model diagram
- Attribute and metrics list
- Detailed dimensional design worksheet
- Issues list

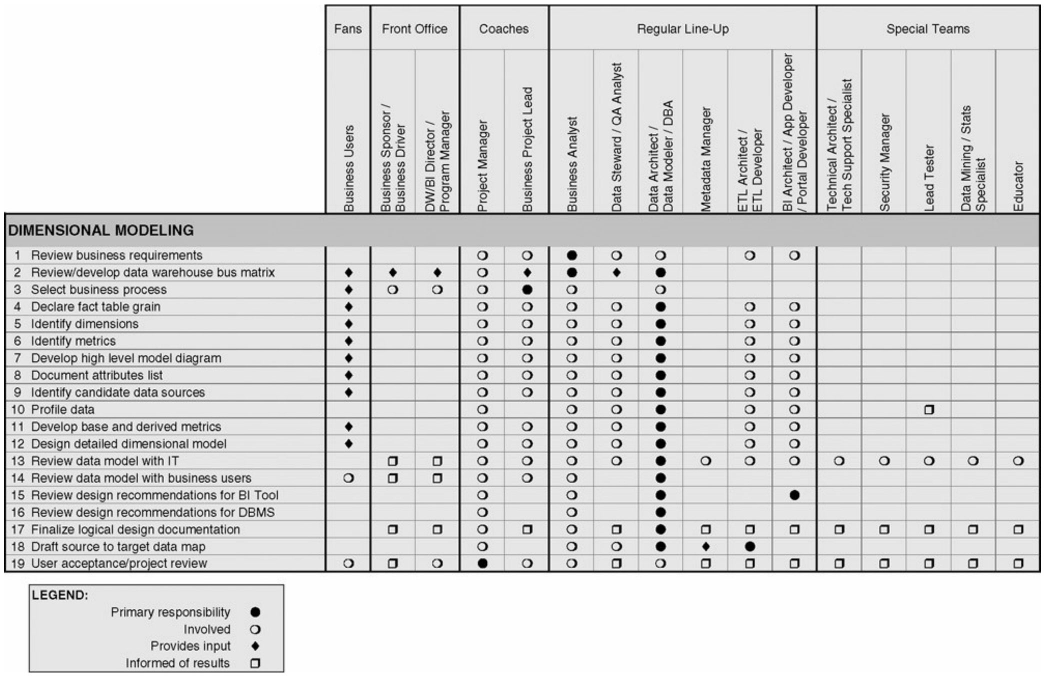
Estimating Considerations

It can take from two weeks to several months to build a model, depending on the complexity of the data and industry, access to key business personnel to make decisions, and their willingness to participate. The scope of the project is also a factor. In general, the logical dimensional data model design typically takes 3–4 weeks for a single fact table and its related dimensions. If the number of dimensions is relatively small, or the dimensions already exist, then the design time can be reduced.

Website Resources

The following templates for designing dimensional models are found on the book's website at http://www.kimballgroup.com.

- Example high level model diagram
- Example detailed dimensional design worksheet



	Fans	Front Office		Coaches		Regular Line-Up						Special Teams				
	Business Users	Business Sponsor / Business Driver	DW/BI Director / Program Manager	Project Manager	Business Project Lead	Business Analyst	Data Steward / QA Analyst	Data Architect / Data Modeler / DBA	Metadata Manager	ETL Architect / ETL Developer	BI Architect / App Developer / Portal Developer	Technical Architect / Tech Support Specialist	Security Manager	Lead Tester	Data Mining / Stats Specialist	Educator
DIMENSIONAL MODELING																
1 Review business requirements				○	○	●	○	○		○	○					
2 Review/develop data warehouse bus matrix	◆	◆	◆	○	◆	●	◆	●								
3 Select business process	◆	○	○	○	●	○		○								
4 Declare fact table grain	◆			○	○	○	○	●		○	○					
5 Identify dimensions	◆			○	○	○	○	●		○	○					
6 Identify metrics	◆			○	○	○	○	●		○	○					
7 Develop high level model diagram	◆			○	○	○	○	●		○	○					
8 Document attributes list	◆			○	○	○	○	●		○	○					
9 Identify candidate data sources	◆			○	○	○	○	●		○	○					
10 Profile data				○		○	○	●		○	○			□		
11 Develop base and derived metrics	◆			○	○	○	○	●		○	○					
12 Design detailed dimensional model	◆			○	○	○	○	●		○	○					
13 Review data model with IT		□	□	○	○	○	○	●	○	○	○	○	○	○	○	○
14 Review data model with business users	○	□	□	○	○	○		●								
15 Review design recommendations for BI Tool				○		○		●			●					
16 Review design recommendations for DBMS				○		○		●								
17 Finalize logical design documentation		□	□	○	□	○	□	●	□	□	□	□	□	□	□	□
18 Draft source to target data map				○		○	○	●	◆	●						
19 User acceptance/project review	○	□	○	●	○	○	□	○	□	□	□	□	□	□	□	□

LEGEND:	
Primary responsibility	●
Involved	○
Provides input	◆
Informed of results	□