EPAM Systems, RD Dep., RD Dep.

INTRODUCTION TO DWH AND ETL

Data Modeling Basics

CONTENTS

1 DIMENSIONAL MODELING

Dimensional modeling is a design discipline that straddles the formal relational model and the engineering realities of text and number data. Compared to third normal form entity-relationship modeling, it is less rigorous (allowing the designer more discretion in organizing the tables), but more practical because it accommodates database complexity and improves performance. Dimensional modeling has an extensive portfolio of techniques for handling real-world situations.

1.1 MEASUREMENTS AND CONTEXT

Dimensional modeling begins by dividing the world into measurements and context. Measurements are usually numeric and taken repeatedly. Numeric measurements are facts. Facts are always surrounded by mostly textual context that is true at the moment the fact is recorded. Facts are very specific, well-defined numeric attributes. By contrast, the context surrounding the facts is open ended and verbose. It is not uncommon for the designer to add context to a set of facts partway through the implementation.

1.1.1 Fact Tables for Measurements

The fact table in a dimensional model stores the performance measurements resulting from an organization's business process events. You should strive to store the low-level measurement data resulting from a business process in a single dimensional model. Because measurement data is overwhelmingly the largest set of data, it should not be replicated in multiple places for multiple organizational functions around the enterprise. Allowing business users from multiple organizations to access a single centralized repository for each set of measurement data ensures the use of consistent data throughout the enterprise.

The term fact represents a business measure. Imagine standing in the marketplace watching products being sold and writing down the unit quantity and dollar sales amount for each product in each sales transaction. These measurements are captured as products are scanned at the register. Each row in a fact table corresponds to a measurement event. The data on each row is at a specific level of detail, referred to as the grain, such as one row per product sold on a sales transaction. One of the core tenets of dimensional modeling is that all the measurement rows in a fact table must be at the same grain. Having the discipline to create fact tables with a single level of detail ensures that measurements are not inappropriately double-counted.

The idea that a measurement event in the physical world has a one-to-one relationship to a single row in the corresponding fact table is a bedrock principle for dimensional modeling. Everything else builds from this foundation.

The most useful facts are numeric and additive, such as dollar sales amount. Additivity is crucial because BI applications rarely retrieve a single fact table row. Rather, they bring back hundreds, thousands, or even millions of fact rows at a time, and the most useful thing to do with so many rows is to add them up. You will see that facts are sometimes semi-additive or even nonadditive. Semi-additive facts, such as account balances, cannot be summed across the time dimension. Non-additive facts, such as unit prices, can never be added. You are forced to use counts and averages or are reduced to printing out the fact rows one at a time—an impractical exercise with a billion-row fact table.

Facts are often described as continuously valued to help sort out what is a fact versus a dimension attribute. The dollar sales amount fact is continuously valued in this example because it can take on virtually any value within a broad range. As an observer, you must stand out in the marketplace and wait for the measurement before you have any idea what the value will be.

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

If there is no sales activity for a given product, you do not put any rows in the table. It is important that you do not try to fill the fact table with zeroes representing no activity because these zeroes would overwhelm most fact tables. By including only true activity, fact tables tend to be quite sparse. Despite their sparsity, fact tables usually make up 90 percent or more of the total space consumed by a dimensional model. Fact tables tend to be deep in terms of the number of rows, but narrow in terms of the number of columns. Given their size, you should be judicious about fact table space utilization.

As examples are developed throughout this book, you will see that all fact table grains fall into one of three categories: transaction, periodic snapshot, and accumulating snapshot. Transaction grain fact tables are the most common.

All fact tables have two or more foreign keys that connect to the dimension tables' primary keys. For example, the product key in the fact table always matches a specific product key in the product dimension table. When all the keys in the fact table correctly match their respective primary keys in the corresponding dimension tables, the tables satisfy referential integrity. You access the fact table via the dimension tables joined to it.

The fact table generally has its own primary key composed of a subset of the foreign keys. This key is often called a composite key. Every table that has a composite key is a fact table. Fact tables express many-to-many relationships. All others are dimension tables.

There are usually a handful of dimensions that together uniquely identify each fact table row. After this subset of the overall dimension list has been identified, the rest of the dimensions take on a single value in the context of the fact table row's primary key. In other words, they go along for the ride.

Although you could lump all context into a wide, logical record associated with each measured fact, you will usually find it convenient and intuitive to divide the context into independent logical clumps. When you record facts—dollar sales for a grocery store purchase of an individual product, for example—you naturally divide the context into clumps named product, store, time, customer, clerk, and several others. We call these logical clumps dimensions and assume informally that these dimensions are independent. Figure 3 shows the dimensional model for a typical grocery store fact.

1.1.2 Additive Facts

At the heart of every fact table is the list of facts that represent the measurements. Because most fact tables are huge, with millions or even billions of rows, you almost never fetch a single record into your answer set. Rather, you fetch a very large number of records, which you compress into digestible form by adding, counting, averaging, or taking the min or max. But for practical purposes, the most common choice, by far, is adding. Applications are simpler if they store facts in an additive format as often as possible. Thus, in the grocery example, you don't need to store the unit price. You merely compute the unit price by dividing the dollar sales by the unit sales whenever necessary.

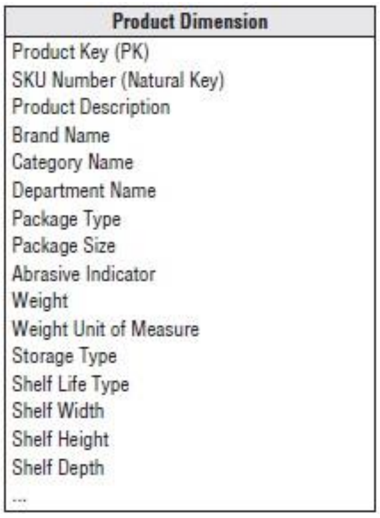
Some facts, like bank balances and inventory levels, represent intensities that are awkward to express in an additive format. You can treat these semi-additive facts as if they were additive—but just before presenting the results to the business user; divide the answer by the number of time periods to get the right result. This technique is called averaging over time.

Some perfectly good fact tables represent measurement events with no facts, so we call them factless fact tables. The classic example of a factless fact table is a record representing a student attending a class on a specific day. The dimensions are day, student, professor, course, and location, but there are no obvious numeric facts. The tuition paid and grade received are good facts, but not at the grain of the daily attendance.

1.1.3 Dimension Tables for Descriptive Context

Dimension tables are integral companions to a fact table. The dimension tables contain the textual context associated with a business process measurement event. They describe the "who, what, where, when, how, and why" associated with the event. As illustrated in Figure 1, dimension tables often have many columns or attributes. It is not uncommon for a dimension table to have 50 to 100 attributes; although, some dimension tables naturally have only a handful of attributes.

Dimension tables tend to have fewer rows than fact tables, but can be wide with many large text columns. Each dimension is defined by a single primary key (refer to the PK notation in Figure 1), which serves as the basis for referential integrity with any given fact table to which it is joined.



Product Dimension
Product Key (PK)
SKU Number (Natural Key)
Product Description
Brand Name
Category Name
Department Name
Package Type
Package Size
Abrasive Indicator
Weight
Weight Unit of Measure
Storage Type
Shelf Life Type
Shelf Width
Shelf Height
Shelf Depth
...

Figure 1 Dimension tables contain descriptive characteristics of business process nouns

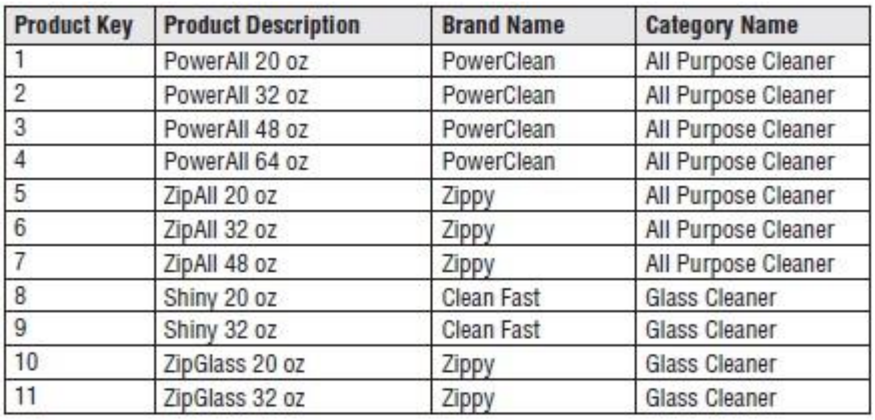
Dimension attributes serve as the primary source of query constraints, groupings, and report labels. In a query or report request, attributes are identified as the by words. For example, when a user wants to see dollar sales by brand, brand must be available as a dimension attribute.

Dimension table attributes play a vital role in the DW/BI system. Because they are the source of virtually all constraints and report labels, dimension attributes are critical to making the DW/BI system usable and understandable. Attributes should consist of real words rather than cryptic abbreviations. You should strive to minimize the use of codes in dimension tables by replacing them with more verbose textual attributes. You may have already trained the business users to memorize operational codes, but going forward, minimize their reliance on miniature notes attached to their monitor for code translations. You should make standard decodes for the operational codes available as dimension attributes to provide consistent labeling on queries, reports, and BI applications. The decode values should never be buried in the reporting applications where inconsistency is inevitable.

Sometimes operational codes or identifiers have legitimate business significance to users or are required to communicate back to the operational world. In these cases, the codes should appear as explicit dimension attributes, in addition to the corresponding user-friendly textual descriptors. Operational codes sometimes have intelligence embedded in them. For example, the first two digits may identify the line of business, whereas the next two digits may identify the global region. Rather than forcing users to interrogate or filter on substrings within the operational codes, pull out the embedded meanings and present them to users as separate dimension attributes that can easily be filtered, grouped, or reported.

In many ways, the data warehouse is only as good as the dimension attributes; the analytic power of the DW/BI environment is directly proportional to the quality and depth of the dimension attributes. The more time spent providing attributes with verbose business terminology, the better. The more time spent populating the domain values in an attribute column, the better. The more time spent ensuring the quality of the values in an attribute column, the better. Robust dimension attributes deliver robust analytic slicing-and-dicing capabilities.

Figure 2 shows that dimension tables often represent hierarchical relationships. For example, products roll up into brands and then into categories. For each row in the product dimension, you should store the associated brand and category description. The hierarchical descriptive information is stored redundantly in the spirit of ease of use and query performance. You should resist the perhaps habitual urge to normalize data by storing only the brand code in the product dimension and creating a separate brand lookup table, and likewise for the category description in a separate category lookup table. This normalization is called snowflaking. Instead of third normal form, dimension tables typically are highly denormalized with flattened many-to-one relationships within a single dimension table. Because dimension tables typically are geometrically smaller than fact tables, improving storage efficiency by normalizing or snowflaking has virtually no impact on the overall database size. You should usually trade off dimension table space for simplicity and accessibility.



Product Key	Product Description	Brand Name	Category Name
1	PowerAll 20 oz	PowerClean	All Purpose Cleaner
2	PowerAll 32 oz	PowerClean	All Purpose Cleaner
3	PowerAll 48 oz	PowerClean	All Purpose Cleaner
4	PowerAll 64 oz	PowerClean	All Purpose Cleaner
5	ZipAll 20 oz	Zippy	All Purpose Cleaner
6	ZipAll 32 oz	Zippy	All Purpose Cleaner
7	ZipAll 48 oz	Zippy	All Purpose Cleaner
8	Shiny 20 oz	Clean Fast	Glass Cleaner
9	Shiny 32 oz	Clean Fast	Glass Cleaner
10	ZipGlass 20 oz	Zippy	Glass Cleaner
11	ZipGlass 32 oz	Zippy	Glass Cleaner

Figure 2 Sample rows from a dimension table with denormalized hierarchies

1.2 FACTS AND DIMENSIONS JOINED IN A STAR SCHEMA

Now that you understand fact and dimension tables, it is time to bring the building blocks together in a dimensional model, as shown in Figure 3. Each business process is represented by a dimensional model that consists of a fact table containing the event's numeric measurements surrounded by a halo of dimension tables that contain the textual context that was true at the moment the event occurred. This characteristic star-like structure is often called a star join, a term dating back to the earliest days of relational databases.

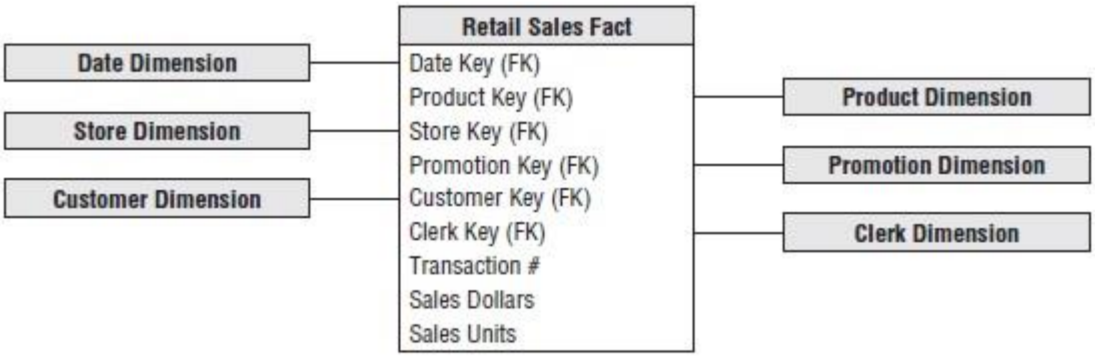


Figure 3 Grocery Store Dimensional Model

In truth, dimensions rarely are completely independent in a strong statistical sense. In the grocery store example, customer and store clearly will show a statistical correlation. Nevertheless, it is usually the right decision to model customer and store as separate dimensions. A single, combined dimension would likely be unwieldy with tens of millions of rows. In addition, the record of when a given customer shopped in a given store would be expressed more naturally in a fact table that also showed the time dimension.

The assumption of dimension independence would mean that all the dimensions, such as product, store, and customer, are independent of time. But you have to account for the slow, episodic change of these dimensions in the way you handle them. In effect, as keepers of the data warehouse, we have taken a pledge to faithfully represent these changes. This predicament gives rise to the technique of slowly changing dimensions.

1.3 DIMENSIONAL KEYS

If the facts are truly measures taken repeatedly, you find that fact tables always create a characteristic many-to-many relationship among the dimensions. Many customers buy many products in many stores at many times.

Therefore, you logically model measurements as fact tables with multiple foreign keys referring to the contextual entities. And the contextual entities are each dimensions with a single primary key, e.g., for the customer dimension, as in Figure 3. Although you can separate the logical design from the physical design, in a relational database fact tables and dimension tables are most often explicit tables.

Actually, a real relational database has two levels of physical design. At the higher level, tables are explicitly declared together with their fields and keys. The lower level of physical design describes the way the bits are organized on the disk and in memory. Not only is this design highly dependent on the particular database, but some implementations may even “invert” the database beneath the level of table declarations and store the bits in ways that are not directly related to the higher level physical records. What follows is a discussion of the higher-level physical design only.

A fact table in a dimensional star schema consists of multiple foreign keys, each paired with a primary key in a dimension, together with the facts containing the measurements. In Figure 3, the foreign keys in the fact table are labeled FK, and the primary keys in the dimension tables are labeled PK, as shown for the customer dimension. I insist that the foreign keys in the fact table obey referential integrity with respect to the primary keys in their respective dimensions. In other words, every foreign key in the fact table has a match to a unique primary key in the respective dimension. Note that this design allows the dimension table to possess primary keys that are not found in the fact table. Therefore, a product dimension table might be paired with a sales fact table in which some of the products are never sold. This situation is perfectly consistent with referential integrity and proper dimensional modeling.

In the real world, there are many compelling reasons to build the FK-PK pairs as surrogate keys that are just sequentially assigned integers. It is a major mistake to build data warehouse keys out of the natural keys that come from the underlying operational data sources. Occasionally a perfectly legitimate measurement will involve a missing dimension. Perhaps in some situations a product can be sold to a customer in a transaction without a store defined. In this case, rather than attempting to store a null value in the store FK, you build a special record in the store dimension representing “No Store.” Now this condition has a perfectly normal FK-PK representation in the fact table.

Logically, a fact table does not need a primary key because, depending on the information available, two different legitimate observations could be represented identically. Practically speaking, this is a terrible idea because normal SQL makes it very hard to select one of the records without selecting the other. It would also be hard to check data quality if multiple records were indistinguishable from each other.

1.4 RELATING THE TWO MODELING WORLDS

Dimensional models are full-fledged relational models, where the fact table is in third normal form and the dimension tables are in second normal form, confusingly referred to as denormalized. Remember that the chief difference between second and third normal forms is that repeated entries are removed from a second normal form table and placed in their own “snowflake.” Thus, the act of removing the context from a fact record and creating dimension tables places the fact table in third normal form.

I resist the urge to further snowflake the dimension tables and am content to leave them in flat second normal form because the flat tables are much more efficient to query. In particular, dimension attributes with many repeated values are perfect targets for bitmap indexes. Snowflaking a dimension into third normal form, while not incorrect, destroys the ability to use bitmap indexes and increases the user perceived complexity of the design. Remember that in the presentation area of the data warehouse, you don’t have to worry about enforcing many-to-one data rules in the physical table design by demanding snowflaked dimensions. The ETL staging system has already enforced those rules.

1.5 STAR SCHEMAS VERSUS OLAP CUBES

Dimensional models implemented in relational database management systems are referred to as star schemas because of their resemblance to a star-like structure. Dimensional models implemented in multidimensional database environments are referred to as online analytical processing (OLAP) cubes, as illustrated in Figure 4.

If your DW/BI environment includes either star schemas or OLAP cubes, it leverages dimensional concepts. Both stars and cubes have a common logical design with recognizable dimensions; however, the physical implementation differs.

When data is loaded into an OLAP cube, it is stored and indexed using formats and techniques that are designed for dimensional data. Performance aggregations or precalculated summary tables are often created and managed by the OLAP cube engine. Consequently, cubes deliver superior query performance because of the precalculations, indexing strategies, and other optimizations. Business users can drill down or up by adding or removing attributes from their analyses with excellent performance without issuing new queries. OLAP cubes also provide more analytically robust functions that exceed those available with SQL. The downside is that you pay a load performance price for these capabilities, especially with large data sets.

Although the capabilities of OLAP technology are continuously improving, it is generally recommended that detailed, atomic information be loaded into a star schema; optional OLAP cubes are then populated from the star schema.

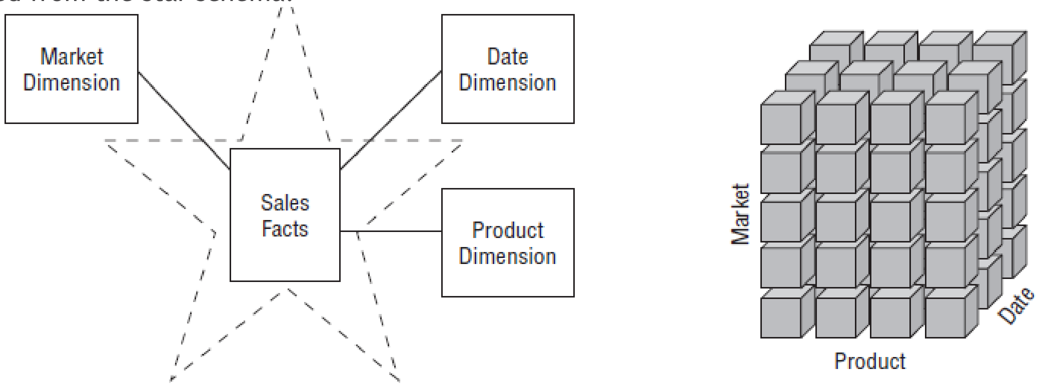


Figure 4 Star schema versus OLAP cube

1.6 SNOWFLAKE SCHEMA

"Snowflaking" is a method of normalizing the dimension tables in a star schema. When it is completely normalized along all the dimension tables, the resultant structure resembles a snowflake with the fact table in the middle. The principle behind snowflaking is normalization of the dimension tables by removing low cardinality attributes and forming separate tables.

The snowflake schema is similar to the star schema. However, in the snowflake schema, dimensions are normalized into multiple related tables, whereas the star schema's dimensions are normalized with each dimension represented by a single table.

“When can I use a snowflake?” is a common question for data warehouse designers. And usually answer is that it is a bad idea to expose the users to a physical snowflake design because it usually compromises understandability and performance. Nevertheless, in certain situations a snowflake design is not only acceptable, but also recommended.

1.6.1 Classic Snowflake

The way to create a classic snowflake is to remove low cardinality attributes from a dimension table and place these attributes in a secondary dimension table connected by a snowflake key. In cases where a set of attributes form a multilevel hierarchy, the resulting string of tables looks a little like a snowflake—hence the name.

Hence, it is not recommended to use snowflakes, there are three cases where variations on a snowflake are not only acceptable, but are the keys to a successful design. These three cases are large customer dimensions, financial product dimensions, and multi-enterprise calendar dimensions.

1.6.2 Large Customer Dimensions

The customer dimension is probably the most challenging dimension in a data warehouse. In a large organization, the customer dimension can be huge, with millions of records, and wide, with dozens of attributes. To make matters worse, the biggest customer dimensions commonly contain two categories of customers, which I will call "visitor" and "customer."

Visitors are anonymous. You may see them more than once, but you do not know their names or anything else about them. On a web site, the only knowledge you have about visitors is a cookie indicating they have returned. In a retail operation, a visitor with a physical shopper tag engages in an anonymous transaction.

Customers, conversely, are reliably registered with your company. You know customers' names, addresses, and as much demographic and historical data as you care to elicit directly from them or purchase from third parties.

Let us assume that at the most granular level of your data collection, 80 percent of the fact table measurements involve visitors and 20 percent involve customers. You accumulate just two simple behavior scores for visitors consisting only of recency (when they last visited you) and frequency (how many times they have visited).

On the other hand, let us assume you have 50 attributes and measures for a customer, covering all the components of location, payment behavior, credit behavior, directly elicited demographic attributes, and purchased demographic attributes.

Now you combine visitors and customers into a single logical dimension called shopper, as shown in Figure 5. You give the visitor or customer a single, permanent shopper ID, but make the key to the table a surrogate key so that you can track changes to the shopper over time.

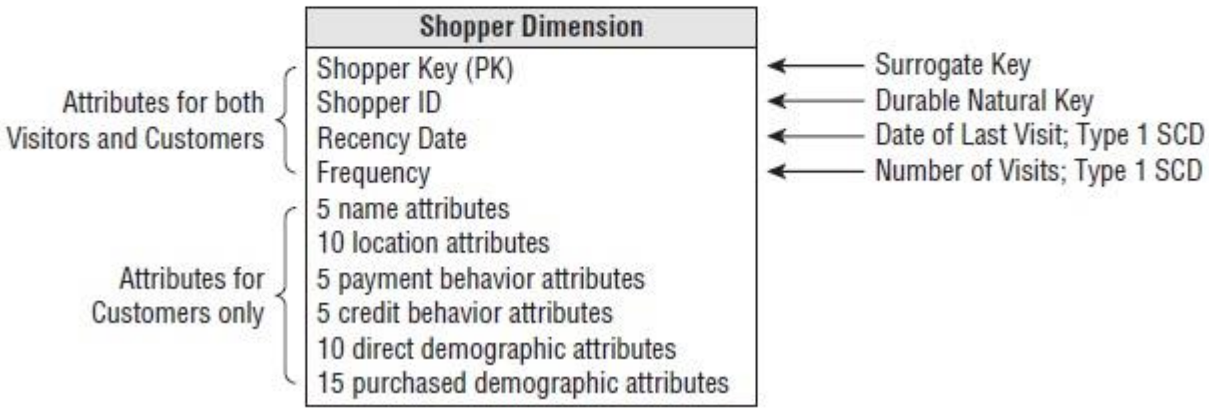

Shopper Dimension
Shopper Key (PK)
Shopper ID
Recency Date
Frequency
5 name attributes
10 location attributes
5 payment behavior attributes
5 credit behavior attributes
10 direct demographic attributes
15 purchased demographic attributes
Attributes for both Visitors and Customers
Attributes for Customers only
Surrogate Key
Durable Natural Key
Date of Last Visit; Type 1 SCD
Number of Visits; Type 1 SCD

Figure 5 Sample Shopper Dimension

Note the importance of including the recency and frequency information as dimensional attributes rather than as facts and overwriting them as time progresses. This decision makes the shopper dimension very powerful. You can do classic shopper segmentation directly off the dimension without navigating a fact table in a complex application.

Assuming that many of the final 50 customer attributes are textual, you could have a total record width of 500 bytes or more. Suppose you have 20 million shoppers (16 million visitors and four million registered customers). Obviously, you are worried that in 80 percent of your records, the trailing 50 fields contain no data! In a 10 GB dimension, this condition gets your attention.

This is a clear case where, depending on the database, you may wish to introduce a snowflake. In databases with variable-width records, you can simply build a single shopper dimension with all the preceding fields, disregarding the empty field issue. The majority of the shopper records that are simple visitors remain narrow because in these databases, the null fields take up zero disk space.

But in fixed-width databases, you probably do not want to live with the empty fields for all the visitors, and so you break the dimension into a base dimension and a snowflaked subdimension, as shown in Figure 6. All the visitors share a single record in the subdimension, which contains special null attribute values.

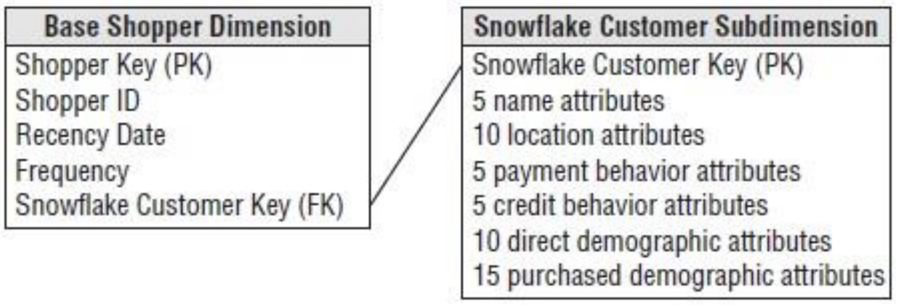

Base Shopper Dimension
Shopper Key (PK)
Shopper ID
Recency Date
Frequency
Snowflake Customer Key (FK)
Snowflake Customer Subdimension
Snowflake Customer Key (PK)
5 name attributes
10 location attributes
5 payment behavior attributes
5 credit behavior attributes
10 direct demographic attributes
15 purchased demographic attributes

Figure 6 A shopper dimension with a snowflake for customer attributes

In a fixed-width database, using our previous assumptions, the base shopper dimension is 20 million × 25 bytes=500 MB, and the snowflake dimension is 4 million × 475 bytes=1.9 GB. You save 8 GB by using the snowflake. If you have a query tool that insists on a classic star schema with no snowflakes, then you can hide the snowflake under a view declaration.

2 FOUR-STEP DIMENSIONAL DESIGN PROCESS

The most common approach to design the dimensional model begins by consistently considering four steps, as the following sections discuss in more detail.

2.1 STEP 1: SELECT THE BUSINESS PROCESS

A business process is a low-level activity performed by an organization, such as taking orders, invoicing, receiving payments, handling service calls, registering students, performing a medical procedure, or processing claims. To identify your organization’s business processes, it is helpful to understand several common characteristics:

- Business processes are frequently expressed as action verbs because they represent activities that the business performs. The companion dimensions describe descriptive context associated with each business process event.
- Business processes are typically supported by an operational system, such as the billing or purchasing system.
- Business processes generate or capture key performance metrics. Sometimes the metrics are a direct result of the business process; the measurements are derivations at other times. Analysts invariably want to scrutinize and evaluate these metrics by a seemingly limitless combination of filters and constraints.
- Business processes are usually triggered by an input and result in output metrics. In many organizations, there’s a series of processes in which the outputs from one process become the inputs to the next. In the parlance of a dimensional modeler, this series of processes results in a series of fact tables.

You need to listen carefully to the business to identify the organization’s business processes because business users cannot readily answer the question, “What business process are you interested in?” The performance measurements users want to analyze in the DW/BI system result from business process events.

Sometimes business users talk about strategic business initiatives instead of business processes. These initiatives are typically broad enterprise plans championed by executive leadership to deliver competitive advantage. In order to tie a business initiative to a business process representing a project-sized unit of work for the DW/BI team, you need to decompose the business initiative into the underlying processes. This means digging a bit deeper to understand the data and operational systems that support the initiative’s analytic requirements.

It is also worth noting what a business process is not. Organizational business departments or functions do not equate to business processes. By focusing on processes, rather than on functional departments, consistent information is delivered more economically throughout the organization. If you design departmentally bound dimensional models, you inevitably duplicate data with different labels and data values. The best way to ensure consistency is to publish the data once.

2.2 STEP 2: DECLARE THE GRAIN

Although theoretically any mixture of measured facts could be shoehorned into a single table, a proper dimensional design allows only facts of a uniform grain (the same dimensionality) to coexist in a single fact table. Uniform grain guarantees that all the dimensions are used with all the fact records (keeping in mind the "No Store" example) and greatly reduces the possibility of application errors due to combining data at different grains. For example, it is usually meaningless to blithely add daily data to yearly data. When you have facts at two different grains, place the facts in separate tables.

Well, declaring the grain means specifying exactly what an individual fact table row represents. The grain conveys the level of detail associated with the fact table measurements. It provides the answer to the question, "How do you describe a single row in the fact table?" The grain is determined by the physical realities of the operational system that captures the business process's events.

Example grain declarations include:

- One row per scan of an individual product on a customer's sales transaction
- One row per line item on a bill from a doctor
- One row per individual boarding pass scanned at an airport gate
- One row per daily snapshot of the inventory levels for each item in a warehouse
- One row per bank account each month

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 is a mistake to list a set of dimensions and then assume this list is the grain declaration. Whenever possible, you should express the grain in business terms.

Dimensional modelers sometimes try to bypass this seemingly unnecessary step of the four-step design process. Please do not! Declaring the grain is a critical step that cannot be taken lightly. In debugging thousands of dimensional designs over the years, the most frequent error is not declaring the grain of the fact table at the beginning of the design process. If the grain is not clearly defined, the whole design rests on quicksand; discussions about candidate dimensions go around in circles, and rogue facts sneak into the design. An inappropriate grain haunts a DW/BI implementation!

It is extremely important that everyone on the design team reach agreement on the fact table's granularity. Having said this, you may discover in steps 3 or 4 of the design process that the grain statement is wrong. This is okay, but then you must return to step two, restate the grain correctly, and revisit steps 3 and 4 again.

2.3 STEP 3: IDENTIFY THE DIMENSIONS

Dimensions fall out of the question, "How do business people describe the data resulting from the business process measurement events?" You need to decorate fact tables with a robust set of dimensions representing all possible descriptions that take on single values in the context of each measurement. If you are clear about the grain, the dimensions typically can easily be identified as they represent the "who, what, where, when, why, and how" associated with the event. Examples of common dimensions include date, product, customer, employee, and facility. With the choice of each dimension, you then list all the discrete, text-like attributes that flesh out each dimension table.

2.4 STEP 4: IDENTIFY THE FACTS

Facts are determined by answering the question, “What is the process measuring?” Business users are keenly interested in analyzing these performance metrics. All candidate facts in a design must be true to the grain defined in step 2. Facts that clearly belong to a different grain must be in a separate fact table. Typical facts are numeric additive figures, such as quantity ordered or dollar cost amount. You need to consider both your business users’ requirements and the realities of your source data in tandem to make decisions regarding the four steps. We strongly encourage you to resist the temptation to model the data by looking at source data alone. It may be less intimidating to dive into the data rather than interview a businessperson; however, the data is no substitute for business user input. Unfortunately, many organizations have attempted this path of least-resistance data-driven approach but without much success.