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

Dimensional Modeling Introduction

Now that you understand the DW/BI system's goals, let's consider the basics of dimensional modeling. *Dimensional modeling* is widely accepted as the preferred technique for presenting analytic data because it addresses two simultaneous requirements:

- Deliver data that's understandable to the business users.
- Deliver fast query performance.

Dimensional modeling is a longstanding technique for making databases simple. In case after case, for more than five decades, IT organizations, consultants, and business users have naturally gravitated to a simple dimensional structure to match the fundamental human need for simplicity. Simplicity is critical because it ensures that users can easily understand the data, as well as allows software to navigate and deliver results quickly and efficiently.

Imagine an executive who describes her business as, "We sell products in various markets and measure our performance over time." Dimensional designers listen carefully to the emphasis on product, market, and time. Most people find it intuitive to think of such a business as a cube of data, with the edges labeled product, market, and time. Imagine slicing and dicing along each of these dimensions. Points inside the cube are where the measurements, such as sales volume or profit, for that combination of product, market, and time are stored. The ability to visualize something as abstract as a set of data in a concrete and tangible way is the secret of understandability. If this perspective seems too simple, good! A data model that starts simple has a chance of remaining simple at the end of the design. A model that starts complicated surely will be overly complicated at the end, resulting in slow query performance and business user rejection. Albert Einstein captured the basic philosophy driving dimensional design when he said, "Make everything as simple as possible, but not simpler."

Although dimensional models are often instantiated in relational database management systems, they are quite different from *third normal form (3NF) models* which seek to remove data redundancies. Normalized 3NF structures divide data into many discrete entities, each of which becomes a relational table. A database of sales orders might start with a record for each order line but turn into a complex spider web diagram as a 3NF model, perhaps consisting of hundreds of normalized tables.

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

Normalized 3NF structures are immensely useful in operational processing because an update or insert transaction touches the database in only one place. Normalized models, however, are too complicated for BI queries. Users can't understand, navigate, or remember normalized models that resemble a map of the Los Angeles freeway system. Likewise, most relational database management systems can't efficiently query a normalized model; the complexity of users' unpredictable queries overwhelms the database optimizers, resulting in disastrous query performance. The use of normalized modeling in the DW/BI presentation area defeats the intuitive and high-performance retrieval of data. Fortunately, dimensional modeling addresses the problem of overly complex schemas in the presentation area.

NOTE A dimensional model contains the same information as a normalized model, but packages the data in a format that delivers user understandability, query performance, and resilience to change.

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 1-1.

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.

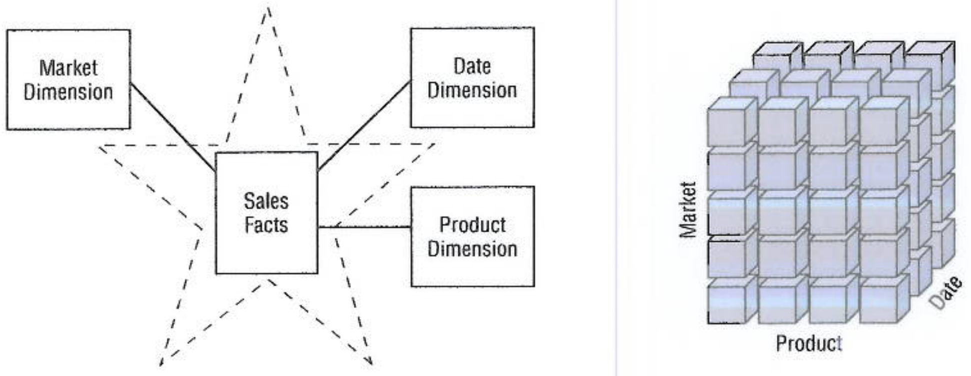


Figure 1-1: Star schema versus OLAP cube.

Fortunately, most of the recommendations in this book pertain regardless of the relational versus multidimensional database platform. Although the capabilities of OLAP technology are continuously improving, we generally recommend that detailed, atomic information be loaded into a star schema; optional OLAP cubes are then populated from the star schema. For this reason, most dimensional modeling techniques in this book are couched in terms of a relational star schema.

OLAP Deployment Considerations

Here are some things to keep in mind if you deploy data into OLAP cubes:

- A star schema hosted in a relational database is a good physical foundation for building an OLAP cube, and is generally regarded as a more stable basis for backup and recovery.
- OLAP cubes have traditionally been noted for extreme performance advantages over RDBMSs, but that distinction has become less important with advances in computer hardware, such as appliances and in-memory databases, and RDBMS software, such as columnar databases.
- OLAP cube data structures are more variable across different vendors than relational DBMSs, thus the final deployment details often depend on which OLAP vendor is chosen. It is typically more difficult to port BI applications between different OLAP tools than to port BI applications across different relational databases.

- OLAP cubes typically offer more sophisticated security options than RDBMSs, such as limiting access to detailed data but providing more open access to summary data.
- OLAP cubes offer significantly richer analysis capabilities than RDBMSs, which are saddled by the constraints of SQL. This may be the main justification for using an OLAP product.
- OLAP cubes gracefully support slowly changing dimension type 2 changes (which are discussed in Chapter 5: Procurement), but cubes often need to be reprocessed partially or totally whenever data is overwritten using alternative slowly changing dimension techniques.
- OLAP cubes gracefully support transaction and periodic snapshot fact tables, but do not handle accumulating snapshot fact tables because of the limitations on overwriting data described in the previous point.
- OLAP cubes typically support complex ragged hierarchies of indeterminate depth, such as organization charts or bills of material, using native query syntax that is superior to the approaches required for RDBMSs.
- OLAP cubes may impose detailed constraints on the structure of dimension keys that implement drill-down hierarchies compared to relational databases.
- Some OLAP products do not enable dimensional roles or aliases, thus requiring separate physical dimensions to be defined.

We'll return to the world of dimensional modeling in a relational platform as we consider the two key components of a star schema.

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, as illustrated in Figure 1-2.

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 aren't inappropriately double-counted.

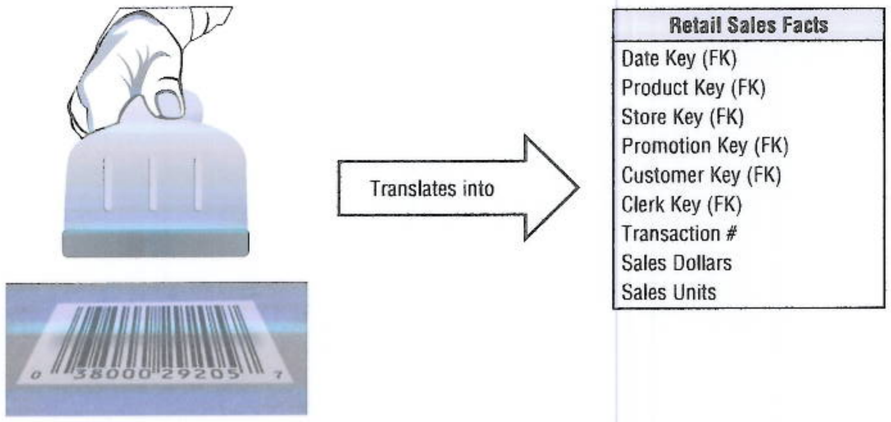


Figure 1-2: Business process measurement events translate into fact tables.

NOTE 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. Throughout this book we will use dollars as the standard currency to make the case study examples more tangible—you can substitute your own local currency if it isn't dollars.

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. No matter how the user slices the data in Figure 1-2, the sales units and dollars sum to a valid total. You will see that facts are sometimes semi-additive or even non-additive. 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.

Referring to the sample fact table in Figure 1-2, if there is no sales activity for a given product, you don't put any rows in the table. It is important that you do not try to fill the fact table with zeros representing no activity because these zeros 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. We will introduce transaction fact tables in Chapter 3: Retail Sales, and both periodic and accumulating snapshots in Chapter 4: Inventory.

All fact tables have two or more foreign keys (refer to the FK notation in Figure 1-2) that connect to the dimension tables' primary keys. For example, the product key in the fact table always 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.

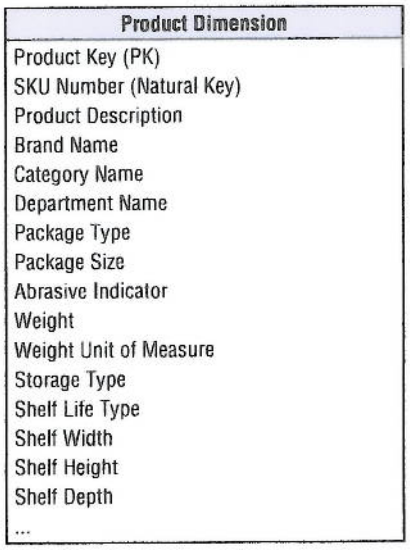
The fact table generally has its own primary key composed of a subset of the foreign keys. This key is often called a *composite key*. Every table that has a composite key is a fact table. Fact tables express many-to-many relationships. All others are dimension tables.

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.

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 1-3, 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 1-3), 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 1-3: 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.

NOTE Dimensions provide the entry points to the data, and the final labels and groupings on all DW/BI analyses.

When triaging operational source data, it is sometimes unclear whether a numeric data element is a fact or dimension attribute. You often make the decision by asking whether the column is a measurement that takes on lots of values and participates in calculations (making it a fact) or is a discretely valued description that is more or less constant and participates in constraints and row labels (making it a dimensional attribute). For example, the standard cost for a product seems like a constant attribute of the product but may be changed so often that you decide it is more like a measured fact. Occasionally, you can't be certain of the classification; it is possible to model the data element either way (or both ways) as a matter of the designer's prerogative.

NOTE The designer's dilemma of whether a numeric quantity is a fact or a dimension attribute is rarely a difficult decision. Continuously valued numeric observations are almost always facts; discrete numeric observations drawn from a small list are almost always dimension attributes.

Figure 1-4 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 almost always 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 1-4: Sample rows from a dimension table with denormalized hierarchies.

Contrary to popular folklore, Ralph Kimball didn't invent the terms fact and dimension. As best as can be determined, the dimension and fact terminology originated from a joint research project conducted by General Mills and Dartmouth University in the 1960s. In the 1970s, both AC Nielsen and IRI used the terms consistently to describe their syndicated data offerings and gravitated to dimensional models for simplifying the presentation of their analytic information. They understood that their data wouldn't be used unless it was packaged simply. It is probably accurate to say that no single person invented the dimensional approach. It is an irresistible force in designing databases that always results when the designer places understandability and performance as the highest goals.

Facts and Dimensions Joined in a Star Schema

Now that you understand fact and dimension tables, it's time to bring the building blocks together in a dimensional model, as shown in Figure 1-5. 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.

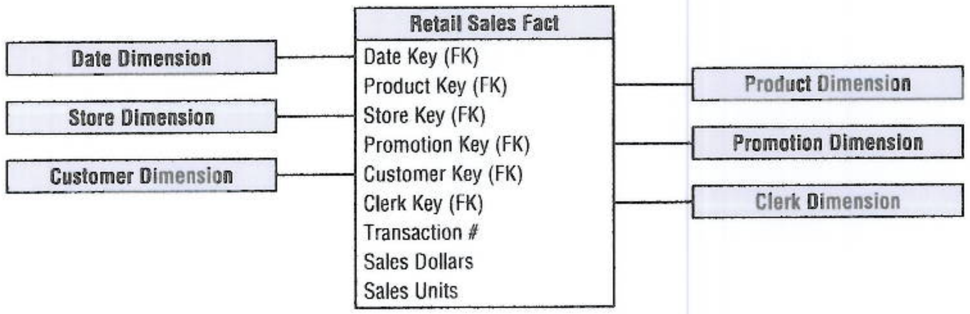


Figure 1-5: Fact and dimension tables in a dimensional model.

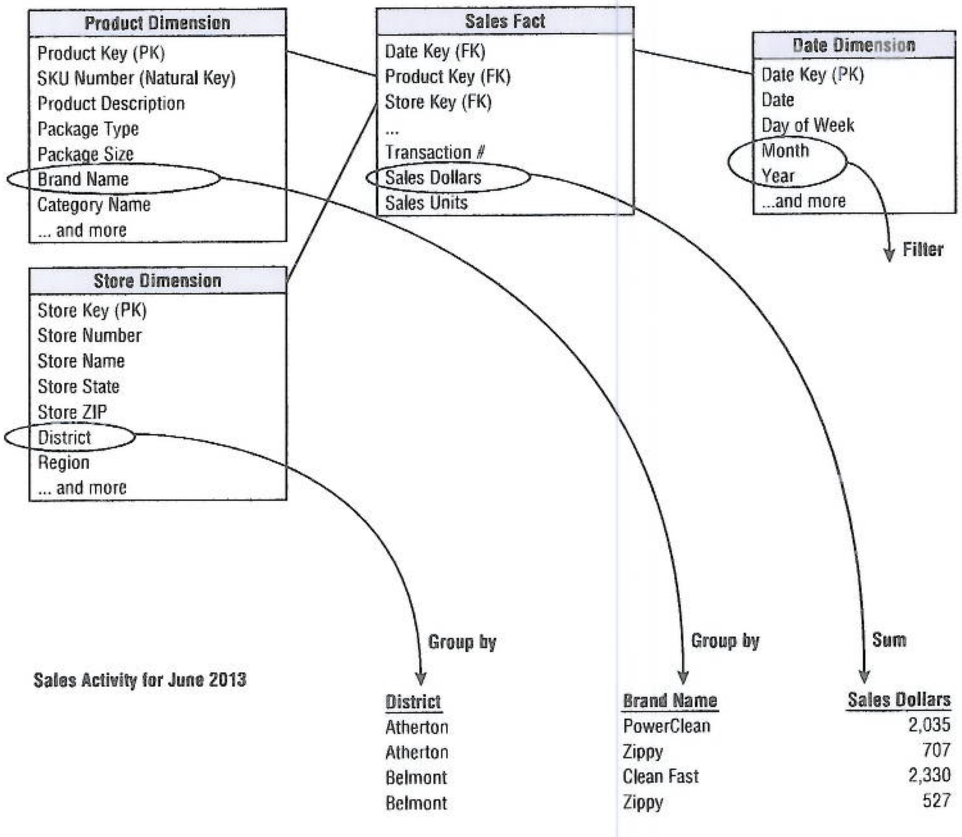
The first thing to notice about the dimensional schema is its simplicity and symmetry. Obviously, business users benefit from the simplicity because the data is easier to understand and navigate. The charm of the design in Figure 1-5 is that it is highly recognizable to business users. We have observed literally hundreds of instances in which users immediately agree that the dimensional model is their business. Furthermore, the reduced number of tables and use of meaningful business descriptors make it easy to navigate and less likely that mistakes will occur.

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

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

This book illustrates repeatedly that the most granular or atomic data has the most dimensionality. Atomic data that has not been aggregated is the most expressive data; this atomic data should be the foundation for every fact table design to withstand business users' ad hoc attacks in which they pose unexpected queries. With dimensional models, you can add completely new dimensions to the schema as long as a single value of that dimension is defined for each existing fact row. Likewise, you can add new facts to the fact table, assuming that the level of detail is consistent with the existing fact table. You can supplement preexisting dimension tables with new, unanticipated attributes. In each case, existing tables can be changed in place either by simply adding new data rows in the table or by executing an SQL `ALTER TABLE` command. Data would not need to be reloaded, and existing BI applications would continue to run without yielding different results. We examine this graceful extensibility of dimensional models more fully in Chapter 3.

Another way to think about the complementary nature of fact and dimension tables is to see them translated into a report. As illustrated in Figure 1-6, dimension attributes supply the report filters and labeling, whereas the fact tables supply the report's numeric values.



Sales Activity for June 2013

District	Brand Name	Sales Dollars
Atherton	PowerClean	2,035
Atherton	Zippy	707
Belmont	Clean Fast	2,330
Belmont	Zippy	527

Figure 1-6: Dimensional attributes and facts form a simple report.

You can easily envision the SQL that's written (or more likely generated by a BI tool) to create this report:

```
SELECT
      store.district_name,
      product.brand,
      sum(sales_facts.sales_dollars) AS "Sales Dollars"
FROM
      store,
      product,
      date,
      sales_facts
WHERE
      date.month_name="January" AND
      date.year=2013 AND
      store.store_key = sales_facts.store_key AND
      product.product_key = sales_facts.product_key AND
      date.date_key = sales_facts.date_key
GROUP BY
      store.district_name,
      product.brand
```

If you study this code snippet line-by-line, the first two lines under the `SELECT` statement identify the dimension attributes in the report, followed by the aggregated metric from the fact table. The `FROM` clause identifies all the tables involved in the query. The first two lines in the `WHERE` clause declare the report's filter, and the remainder declare the joins between the dimension and fact tables. Finally, the `GROUP BY` clause establishes the aggregation within the report.

Kimball's DW/BI Architecture

Let's build on your understanding of DW/BI systems and dimensional modeling fundamentals by investigating the components of a DW/BI environment based on the Kimball architecture. You need to learn the strategic significance of each component to avoid confusing their role and function.

As illustrated in Figure 1-7, there are four separate and distinct components to consider in the DW/BI environment: operational source systems, ETL system, data presentation area, and business intelligence applications.

Operational Source Systems

These are the operational systems of record that capture the business's transactions. Think of the source systems as outside the data warehouse because presumably you have little or no control over the content and format of the data in these operational systems. The main priorities of the source systems are processing performance and availability. Operational queries against source systems are narrow, one-record-at-a-time