# 1 *The Reader* at a Glance

Beginning in late 2007, Ralph wrote a series of articles for *DM Review* magazine (now called *Information Management*). Published over a 16-month time frame, this sequence systematically describes the Kimball approach and classic best practices in a cohesive manner. Rather than scattering these articles topically throughout the book, we opted to present the series nearly in its entirety because it provides an overview of the content that follows in subsequent chapters. You can think of Chapter 1 as *CliffsNotes* for *The Kimball Group Reader*.

The chapter begins with several articles encouraging you to practice restraint and establish appropriate boundaries with other stakeholders when embarking on a data warehouse/business intelligence (DW/BI) project. From there, the series turns its attention to bringing operational data into the data warehouse and then leveraging core dimensional modeling principles to deliver robust analytic capabilities to the business users.

In addition to the articles in this chapter, Ralph also wrote a very detailed article on data quality for *DM Review*. Due to its in-depth coverage, this article is presented in Chapter 11 with other back room extract, transform, and load (ETL) topics.

## Setting Up for Success

Before diving into implementing the DW/BI system, make sure you assess the complete set of related requirements, while avoiding the risks of overpromising.

### 1.1 Resist the Urge to Start Coding

*Ralph Kimball,* DM Review, *Nov 2007*

The most important first step in designing a DW/BI system, paradoxically, is to stop. Step back for a week, and be absolutely sure you have a sufficiently broad perspective on all the requirements that surround your project. The DW/BI design task is a daunting intellectual challenge, and it is not easy to step far enough back from the problem to protect yourself from embarrassing or career-threatening problems discovered after the project is underway.

Before cutting any code, designing any tables, or making a major hardware or software purchase, take a week to write down thoughtful, high quality answers to the following 10 questions, each of which is a reality that will come to control your project at some point. These define the classic set of simultaneous constraints faced by every DW/BI effort.

1. *Business requirements*. Are you in touch with the key performance indicators (KPIs) your users actually need to make the decisions currently important to their enterprise? Although all 10 questions are important, understanding the business requirements is the most fundamental and far reaching. If you have a positive answer to this question, you can identify the data assets needed to support decision making, and you will be able to decide which measurement process to tackle first.
2. *Strategic data profiling*. Have you verified that your available data assets are capable of supporting the answers to question number one? The goal of strategic data profiling is to make "go/no go" decisions very early in the DW/BI project as to whether to proceed with a subject area.
3. *Tactical data profiling*. Is there a clear executive mandate to support the necessary business process re-engineering required for an effective data quality culture, perhaps even driving for Six Sigma data quality? The only real way to improve data quality is to go back to the source and figure out why better data isn't being entered. Data entry clerks are not the cause of poor data quality! Rather, the fixes require an end-to-end awareness of the need for better quality data and a commitment from the highest levels to change how business processes work.
4. *Integration*. Is there a clear executive mandate in your organization to define common descriptors and measures across all your customer-facing processes? All of the organizations within your enterprise that participate in data integration must come to agreement on key descriptors and measures. Have your executives made it clear that this must happen?
5. *Latency*. Do you have a realistic set of requirements from business users for how quickly data must be published by the data warehouse, including as-of-yesterday, many times per day, and truly instantaneous?
6. *Compliance*. Have you received clear guidance from senior management as to which data is compliance-sensitive, and where you must guarantee that you have protected the chain of custody?
7. *Security*. Do you know how you are going to protect confidential as well as proprietary data in the ETL back room, at the users' desktops, over the web, and on all permanent media?
8. *Archiving*. Do you have a realistic plan for very long term archiving of important data, and do you know what data should be archived?
9. *Supporting business users*. Have you profiled all your user communities to determine their abilities to use spreadsheets, construct database requests in ad hoc query tools, or just view reports on their screens?
10. *IT licenses and skill sets*. Are you prepared to rely on the major technology site licenses your organization has already committed to, and do you have enough staff with advanced skills to exploit the technical choices you make?

Time spent answering these classic DW questions is enormously valuable. Every one of the answers will affect the architecture, choice of approaches, and even the feasibility of your DW/BI project. You dare not start coding before all the team members understand what these answers mean!

The big news is that business users have seized control of the DW. They may not be building the technical infrastructure, but they are quite sure that they own the data warehouse and the BI tools and those tools must meet their needs. This transfer of initiative from IT to the users has been very obvious in the past two or three years. Witness the soul-searching articles and industry speeches exhorting CIOs to show more business leadership and the high CIO turnover as reported in *CIO Magazine* (see the April 1, 2004 issue at `www.cio.com`).

Many of the 10 questions in this article are brought into much clearer focus by increased user ownership of the DW/BI system. Let's focus on the top five new urgent topics, in some cases coalescing our questions:

- *Business requirements*. The DW/BI system needs a permanent "KPI team" continuously in touch with business users' analytic needs and the consequent demand for new data sources to support new KPIs. Also, the system should increasingly support the full gamut of analytic applications, which include not only data delivery, but alerting the users to problems and opportunities, exploring causal factors with additional data sources, testing what-if scenarios to evaluate possible decisions, and tracking the decisions made. The DW/BI system is not just about displaying reports, but rather must be a platform for decision making in the broadest sense. The oldest label for data warehousing, *decision support*, remains surprisingly apt.
- *Strategic data profiling*. The earlier you tell the users bad news about the viability of a proposed data source, the more they will appreciate you. Develop the ability to assess a data source within a day or two. Elevate the data profiling tool to a strategic, must-have status.
- *Tactical data profiling*. The increased awareness of data quality is one of the most remarkable new DW perspectives, certainly driven by business users. But all is for naught if the business is not willing to support a quality culture and the end-to-end business process re-engineering required.
- *Integration and latency*. The user demand for the 360-degree integrated view of the business has been more like an approaching express train than a shock wave. We have been talking about it for almost a decade. But now the demands of integration, coupled with real-time access to information, have combined these two issues into a significant new architectural challenge.
- *Compliance and security*. DW/BI folks in IT often don't have the right instincts for protecting data because the system is supposed to be about exposing data. But this new emphasis on compliance and security can be built systematically into the data flows and the BI tools across the entire DW/BI solution.

The purpose of this first article has been to expose the fundamental design issues every DW/BI design team faces and to bring to the surface the urgent new requirements. In this ongoing series of articles, I probe each of these areas in some depth, reminding us of the remarkably unchanging aspects of data warehousing, while at the same time trying to catch the winds of change.

## 1.2 Set Your Boundaries

*Ralph Kimball,* DM Review, *Dec 2007*

In article 1.1, *Resist the Urge to Start Coding*, I encouraged you to pause briefly before charging forward on your ambitious DW/BI project. You were supposed to use this pause to answer a checklist of major environmental questions regarding business requirements, quality data, and whether your organization is ready to attack the hard issues of integration, compliance, and security.

While answering the questions, I hope you talked to all your business user clients and sponsors who may have a stake or a responsibility in the DW/BI system. Before the memory of these conversations fades away, I suggest you make a thorough list of all the promises you made as you were selling the concept of the DW/BI system. It wouldn't surprise me if you said, "Yes, we will…"

- Tie the rolling operational results to the general ledger (GL).
- Implement effective compliance.
- Identify and implement all the key performance indicators (KPIs) needed by marketing, sales, and finance and make them available in the executive dashboard.
- Encourage the business community to add new cost drivers to our system requirements so that they can calculate activity-based costing and accurate profit across the enterprise. And while we are adding these cost drivers, we'll work out all the necessary allocation factors to assign these costs against various categories of revenue.
- Identify and implement all the customer satisfaction indicators needed by marketing.
- Seamlessly integrate all the customer-facing operational processes into a single coherent system.
- Promise to use exclusively the front end, middleware, and back end tools provided by the enterprise resource planning (ERP) vendor whose worldwide license was just signed by our CEO.
- Be the first showcase application for the new service-oriented architecture (SOA) initiative, and we'll implement, manage, and validate the new infrastructure.
- Implement and manage server virtualization for the DW/BI system. And this new system will be "green."
- Implement and manage the storage area network (SAN) for the DW/BI system.
- Implement and manage security and privacy for all data in the DW/BI system, including responsibility for the lightweight directory access protocol (LDAP) directory server and its associated authentication and authorization functions. We'll also make sure that all data accesses by the sales force in the field are secure.
- Define the requirements for long term archiving and recovery of data looking forward 20 years.

Looking at this list of promises all at once, you might wonder who in their right mind would agree to them. Actually, I am much more sympathetic than it may seem. You must address these topics because they are all key facets of the DW/BI challenge. But if you gave the answers as literally stated, you have lost control of your boundaries. You have taken on far too much, you have made promises you can't deliver, and your business clients and enterprise bosses have abrogated or avoided key responsibilities that they must own. More seriously, even if you think you can deliver all these promises, you are not in a powerful enough position in your enterprise to make all these results happen.

You don't have to be a curmudgeon to be a good DW/BI system manager. This isn't about saying no to every possible responsibility. You will be doing your enterprise a favor by alerting and educating your business users and management to the appropriate boundaries of responsibilities. You can still be an enthusiastic advocate, as long as your boundaries are clear. Let's describe the key boundaries.

- *Boundaries with the business users*. Your job is to find the business users, interview them, and interpret what they tell you into specific DW/BI deliverables. You must assemble a findings document that describes the results of the interviews and how you interpreted what the business users told you. Their responsibility is to be available for the interviews and to put energy into describing how they make decisions. Later in the process, the business users have a responsibility to provide feedback on your findings. You cannot attempt to define business requirements unless the business user community is an equal partner with IT.

  Your job is not over after the first round of interviews. You must encourage ongoing business user feedback and suggestions, and also educate the business users as to the realities of system development. View this as a mutual learning process. In the latter stages of DW/BI system development, you simply cannot add new KPIs and especially new data sources to the project without slipping the delivery date. You cannot suddenly change a batch-oriented system into a real-time pipeline. Your business users must be understanding and trusting partners of the DW/BI system development, and they have to understand the costs of sudden new requirements. Bottom line—business users must become sophisticated observers of the DW/BI development process and know when it is inappropriate to change the scope by adding new KPIs, new data sources, or new real-time requirements.
- *Boundaries with finance*. Of the promises you made, several should be the responsibility of finance. You should never agree to implement cost allocations, even if the "profit system" is your main responsibility. Not only are cost allocations very complex, but the assignment of costs to various revenue-producing departments is bad news politically. In this case, finance should work out the logical and political implications of the cost allocations, and you can quietly implement them.

  You also should never agree to tie rolling operational results to the GL. In dimensional modeling parlance, you can't make this happen because the GL dimensions, such as organization and account, can't be conformed to the operational dimensions, such as customer and product. Also, special GL transactions, such as journal adjustments done at the end of the month, often cannot be put into an operational context. Again, you need to hand this issue back to finance and wait for a solution from them.

- *Boundaries across organizations.* These days it is hard to find anyone who argues against integration of all your data assets under the DW/BI umbrella. But this challenge is 70 percent political and only 30 percent technical. Your executives must establish a corporate culture that sends a very clear message to all the separate departments that they must come together to agree on common dimensional attributes, key performance metrics, and calendars. Your executives must lead the way before you can do your job.
- *Boundaries with legal.* In the early '90s, we often lamented that the data warehouse wasn't seeing widespread use. Well, now we have the opposite problem. A big piece, shall I say headache, of being taken very seriously is providing adequate security, privacy, archiving, and compliance across the DW/BI system. But you can't do anything until you understand your enterprise's policies. You must not define these policies yourself. You can lose your job and go to jail if you get these wrong. Go to your legal department with a list of areas where you need firm guidance.
- *Boundaries with IT.* Strangely, one of the most important boundaries you must maintain is with IT. You should be able to rely on other groups within IT for storage (either SAN or networked attached storage), server virtualization, LDAP server maintenance, authentication technologies, and providing new infrastructure such as SOA.

Most of us in the DW/BI business are natural salespeople. We are evangelists for the use of our systems because we really believe they will benefit the business. But we need to be conscious of trying to please the client too much. Ultimately, the DW/BI system will be much more successful if all the other parties described in this article are equal, responsible partners.

## Tackling DW/BI Design and Development

This group of articles focuses on the big issues that are part of every DW/BI system design.

### 1.3 Data Wrangling

*Ralph Kimball,* DM Review, *Jan 2008*

In this article, we are ready to design the first stage of the data pipeline leading from the operational sources to the final BI user interfaces. I call this stage "data wrangling" because we must lasso the data and get it under our control. Successful data wrangling includes change data capture, extraction, data staging, archiving, and the first step of data warehouse compliance. Let's examine these narrow data wrangling responsibilities.

The amount of data processed in each data extract should be kept to the bare minimum. You should strive not to download complete copies of source tables, but sometimes you must. Limiting the data extract to the bare minimum is a fascinating challenge and can be harder than it appears. The first architectural choice is whether to perform change data capture on the production source computer or after extraction to a machine owned by the data warehouse. From the data warehousing point of view, the more attractive alternative is doing change data capture at the production source. For this you need cooperation from the production source database administrators (DBAs), adequate processing resources on the production machine, and a very high quality scheme for identifying 100 percent of the changes that have occurred since the previous load.

To design the change data capture system on the production source, you need to have a very candid conversation with the production system DBAs. You need to identify every situation in which a change to source data could happen. These include normal applications posting transactions, special administrative overrides, and emergency scenarios, such as restoration of a data set.

One popular way to look for source data changes is to query a `change_date_time` field in the source table. This is a pretty strong approach if this field is populated by database triggers that are not circumvented by any process. But many production applications prohibit the use of triggers for performance reasons. Also, how does such an approach handle record deletes? If the record simply vanishes, you won't find it by querying the `change_date_time` field. But maybe you can collect the deletes in a separate feed.

Another approach is a special production system daemon that captures every input command by reading the production system transaction log or by intercepting message queue traffic. The daemon approach solves the delete problem but is still vulnerable to special administrative override commands performed manually by the DBAs. Some of you may think such overrides are crazy, but I have seen some very well-run shops resort to doing these overrides occasionally because of weird business rules that are simply too complicated to program into the normal transaction processing applications.

If you have figured out an acceptable scheme for isolating all data changes at the source, you still need to ask for one more favor, if you have any political capital left with the source system DBAs. You need to get a reason code for all changes to the major dimensional entities, such as customer or product. In dimensional modeling parlance, these reason codes will tell you whether the change to an individual dimensional attribute should be treated as a slowly changing dimension (SCD) type 1, 2, or 3. These distinctions are a big deal. The ETL pipelines required to process these three SCD choices are completely different.

If your production system presents too many objections, consider doing change data capture after extraction. Now you must download much larger data sets, perhaps complete dimensions or even complete fact tables. But you are guaranteed to find every change in the source system, as long as you keep a complete prior copy of the source system tables against which to compare.

If you download a complete source table today, you can find all the changes by performing a record-by-record and field-by-field comparison against a copy of yesterday's source table. You will indeed find every change, including the deletes. But in this case, you are probably missing reason codes for dimensional attribute changes. If so, you may need to impose unilaterally a reason code policy crafted for each attribute. In other words, if the package type of an existing product suddenly is changed, you could always assume that manufacturing is correcting a data error, and hence the change is always type 1.

If the table you are comparing is very large, the brute force approach of comparing each field can take too long. You can often improve this comparison step by a factor of 10 using a special hash code, called a *cyclic redundancy checksum* (CRC). For a discussion of this advanced technique, see the discussion of cyclic redundancy check on Wikipedia.

Finally, even if you are sure you have accounted for 100 percent of the source system changes, you should periodically check the DW totals against totals computed directly on the source. This is like balancing your checkbook when you have to manually investigate a discrepancy between the two data sets.

Extraction, whether it occurs before or after change data capture, is the transfer of data from the source system into the DW/BI environment. Besides actually moving the data, you have two main responsibilities in this step. First, you need to rid yourself of all narrowly proprietary data formats during the transfer itself. Change EBCDIC character formatting to ASCII. Unravel all IBM mainframe data formats (e.g., packed decimals and `OCCURS` statements) into standard relational database management system table and column formats. I also suggest unraveling XML hierarchical structures at this stage, although perhaps at some point XML structures will be fully supported at a semantic level by relational databases.

Your second responsibility is to direct the flow of incoming data either into simple flat files or relational tables. Both choices are equally valid. You can process flat files very efficiently with sort utilities and sequential processing commands like `grep` and `tr`. Of course, you will eventually load everything into relational tables for joining and random access operations.

I recommend immediately *staging* all data received by the DW/BI system. In other words, save the data you just received in the original target format you have chosen before you do anything else to it. I am very conservative. Staging the extracted data means keeping it forever, either offline or online. Data staging is meant to support all the types of backup.

A special form of archiving serves as an important step when you are forced to deal with compliance-sensitive data: proving that the data you received has not been tampered with. In this case, the data staging is augmented with a strong hash code that you use to show that the data has not changed. You should also write this staged data and hash code to permanent media and store this media with a bonded third party who can verify that the data was sent to them on a certain date.

Now that you have wrangled the data into your DW/BI environment, it is time to tame the beast by making the data support the business users' decision making.

## 1.4 Myth Busters

*Ralph Kimball,* DM Review, *Feb 2008*

Dimensional modeling is an old discipline, dating from the late 1970s when ACNielsen introduced its Inf*Act syndicated data reporting service, organized around dimensions and facts. It is, therefore, surprising that some consultants and industry pundits consistently state myths and misrepresentations about dimensional modeling that have been debunked multiple times. It is time (once again) to address these myths.

*Myth: A dimensional view could be missing key relationships that exist only in a true relational view.*

Myth buster: This is perhaps the best place to start debunking dimensional modeling misrepresentations and myths. A dimensional model contains all the data relationships that a normalized model contains. There is no data relationship expressible in a normalized model that cannot be expressed in a dimensional model. Note that dimensional models are fully relational. Fact tables are generally in third normal form and dimension tables are generally in second normal form. The major difference between the two approaches is that the many-to-one relationships in the dimensions have been denormalized to flatten the dimensions for user understandability and query performance. But all the data relationships and data content are otherwise identical.

*Myth: A very real issue with a dimensional enterprise data model (EDM) is the possibility that the model may not be extensible and easily accommodate changing business needs. Although a logical representation of the business can be achieved using dimensional structures, using these structures could have negative effects on extensibility and industry data integration.*

Myth buster: This myth about extensibility is a strange one; dimensional models are significantly more robust than normalized models when data relationships change. For more than 10 years, we have been teaching the "graceful extensibility" of dimensional models. Five types of change have no effect on the business intelligence applications running on dimensional models:

1. Adding a new dimension to a fact table
2. Adding a new fact to a fact table
3. Adding a new dimension attribute to a dimension table
4. Altering the relationship of two dimension attributes to form a hierarchy (many-to-one relationship)
5. Increasing the granularity of a dimension

In the normalized world, such changes often involve altering the relationship between separate tables. Altering the relationship between tables exposed to the BI tools forces recoding of applications. With dimensional models, the applications keep on running without the need to recode because the dimensional schemas are inherently more robust when confronted with new content and new business rules.

*Myth: A dimensional model by its definition is built to address a very specific business need. Relational modeling mimics business processes, while dimensional modeling captures how people monitor their business.*

Myth buster: A dimensional model is built in response to a measurement process, never a specific business need or a desired final report for a specific department. A fact record in a dimensional model is created as a 1:1 response to a measurement event in a specific business process. Fact tables are defined by the physics of the real world. Our job as modelers is to carefully understand the grain of the physical measurement event and to faithfully attach facts and dimensions to that event that are "true to the grain." A dimensional model satisfies a business requirement only if the business happens to need the measurement events represented in the fact table. The format and content of a dimensional model has no dependence on a final report desired by the business users because it is determined only by the physics of the measurement process. A dimensional model is never crafted to meet the needs of a specific department, but rather is a single representation of a business process that looks the same to all observers.

*Myth: In a dimensional model, usually only one date is associated with time. The other dates (e.g., in an order) are not captured, and, therefore, valuable data can be lost.*

Myth buster: If you understand the previous myth buster, then you can appreciate that a measurement involving a line item on an order will naturally expose many dates. Each of these dates is represented by a foreign key to a copy or view of the date dimension. I first described this technique of using dimension "roles" in article 9.8, *Data Warehouse Role Models*. Role playing dimensions are an old standard dimensional modeling technique we have described hundreds of times.

*Myth: Relational is preferred because an EDM should capture data at a very low granular level—preferably individual transactions.*

Myth buster: From the very beginning, I have urged designers to capture measurement events in fact tables at the lowest possible (e.g., transaction) grain. In my 1996 book, *The Data Warehouse Toolkit*, I wrote, "A data warehouse almost always demands data expressed at the lowest possible grain of each dimension, not because queries want to see individual records, but because queries need to cut through the database in very precise ways." If we have consistently urged dimensional models to be built at the most expressive granular grain for the past 11 years, through 300,000 books, more than 250 articles, and 10,000 students in our classes, where do people come up with myths like this?

Stepping back from these specific myths, I urge you to think critically. When you read or hear strong statements, circle around the issues and educate yourself. Challenge the assumptions. Look for defendable positions, detailed logic, and clear thinking. I expect to be held to such high standards, and I hope you will do the same to others.

## 1.5 Dividing the World

*Ralph Kimball,* DM Review, *Mar 2008*

In the last four *DM Review* articles, I laid a solid foundation for building a data warehouse. We have done a careful job of gathering all the overlapping design constraints; we have established a good set of boundaries with all the groups we interact with; we have captured a perfect subset of changed data to feed our data extraction; and we have described common misunderstandings about dimensional models.

Our next big task is to divide the data into dimensions and facts. *Dimensions* are the basic stable entities in our environment, such as customers, products, locations, marketing promotions, and calendars. *Facts* are the numeric measurements or observations gathered by all of our transaction processing systems and other systems. Business users instinctively understand the difference between dimensions and facts. When we deliver data to the BI tools, we take great care to make dimensions and facts visible at the user interface level in order to exploit the users' understanding and familiarity with these concepts. Perhaps another way to say this is the dimensional data warehouse is the platform for BI.

Dimensions and facts drive the user interface experience in the BI tools. Dimensions are overwhelmingly the target for constraints and the source of "row headers" in the BI tool results. Facts are overwhelmingly the grist for computations. Separating the dimensions and facts structurally in the data is very helpful because it encourages consistency in application development and the BI tool user interfaces.

Dividing the world of data into dimensions and facts is a fundamental and powerful idea. Ninety-eight percent of all data items are immediately and obviously categorized as one or the other. Discrete textual data items that describe the attributes of our stable entities belong to dimensions. Repeated numeric measurements whose values are not fully predictable are facts. Thus, if we sell a red ballpoint pen for $1.79, then "red" is an attribute in the ballpoint pen row in the product dimension, and $1.79 is an observed fact.

The foundation of the data warehouse is the measurement event that produces a fact record. This is a very physical, tangible result. A fact record exists if and only if a measurement event takes place. This physical result is used by the data warehouse designer to make sure that the design sits on solid rock. When we describe the measurement in physical, real-world terms, we call this the grain of the fact table. If you are quite sure of the grain, you will have a relatively easy time designing the fact table. That is why I keep telling students to "keep to the grain."

When a measurement event creates a fact record, we scramble to attach contemporary versions of all the relevant dimensional entities to this fact record. When we sell the red ballpoint pen for $1.79, the flash bulb goes off, and from this snapshot we assemble an impressive set of dimensional entities, including customer, product, store location, employee (cashier), employee (store manager), marketing promotion, calendar, and maybe even the weather. We are careful to use up-to-date versions of the dimensions so that we are describing this sales measurement event correctly. Notice that the grain of this measurement is the cash register "beep" when the item is scanned. Later in the design process, we implement this grain with various foreign keys connecting to the dimensions, but we don't start the design process with the keys. We start with the physical event.

Once we have the grain of the fact table firmly in mind, we make sure that the only facts introduced into our fact records are defined by the scope of the measurement event. In our cash register example, the instantaneous price of the product and the number of units sold are good facts that are true to the grain. But total sales for the month or the sales on the same day last year are not true to the grain and must not be included in the physical fact record. Sometimes it is hard to resist adding facts that are not true to the grain because they provide a shortcut for a specific query, but these rogue facts always introduce complexities, asymmetries, and confusion for the application developer and the business user. Once again—keep to the grain.

Whenever possible, we strive to make facts additive. In other words, it makes sense to add the fact across records. In our retail sales example, although the price is true to the grain, it is not additive. But if we instead store the extended price (unit price multiplied by quantity sold) and the quantity sold, then both these facts are additive. We can instantaneously recover the unit price with a simple division. Forcing facts to be additive whenever possible seems like a small point, but it is one of the many ways we make our BI platform simple. Like the famous Japanese auto manufacturer example of quality, a thousand little improvements eventually become a sustainable strategic advantage. Conversely, a thousand little "gadgets" shoehorned into a database to make certain queries simpler will produce an unworkable, unmaintainable design.

In a similar vein, we resist taking normalized data models all the way into the BI environment. Normalized data models are essential for efficient transaction processing, and are helpful for storing data after the data has been cleaned. But normalized models are not understandable by business users. Before you lapse into religious wars with your colleagues, please recognize that when correctly designed, normalized models and dimensional models contain exactly the same data and reflect exactly the same business rules. Any and all data relationships can be accurately represented using either methodology. Thus, the reason for using dimensional models is that they form a proven, workable basis for BI.

Earlier in this article, I stated that 98 percent of all data items are immediately and obviously categorized as either a fact or a dimension attribute. What about the remaining 2 percent? Perhaps you have been thinking that in our retail sales example the price of the product should actually be in the product dimension, not in the fact table. In my opinion, upon a little reflection, this is an easy choice. Because the price of a product often varies over time and over location, it becomes very cumbersome to model the price as a dimension attribute. It should be a fact. But it is normal to recognize this rather late in the design process.

A more ambiguous example is the limit on coverage within an automobile insurance policy. The limit is a numerical data item, perhaps $300,000 for collision liability. The limit may not change over the life of the policy, or it changes very infrequently. Furthermore, many queries would group or constrain on this limit data item. This sounds like a slam dunk for the limit being an attribute in the coverage dimension. But the limit is a numeric observation, and it can change over time, albeit slowly. One could pose some important queries summing or averaging all the limits on many policies and coverages. This sounds like a slam dunk for the limit being a numeric fact in a fact table.

Rather than agonizing over the dimension versus fact choice, simply model it both ways! Include the limit in the coverage dimension so that it participates in the usual way as a target for constraints and the content for row headers, but also put the limit in the fact table so it can participate in the usual way within complex computations.

This example allows me to summarize this article with an important principle: Your design goal is ease of use, not methodological correctness. In the final step of building dimensional models that are intended for consumption by business users, we should be willing to stand on our heads to make our BI systems understandable and fast. That often means transferring work into the extract, transform, and load (ETL) back room and tolerating more storage overhead to simplify the final data presentation.

## 1.6 Essential Steps for the Integrated Enterprise Data Warehouse

*Ralph Kimball,* DM Review, *Apr 2008 and May 2008*

*This content was originally published as two consecutive articles in the* DM Review *series.*

In this article, I propose a specific architecture for building an integrated enterprise data warehouse (EDW). This architecture directly supports master data management (MDM) efforts and provides the platform for consistent business analysis across the enterprise. I describe the scope and challenges of building an integrated EDW, and provide detailed guidance for designing and administering the necessary processes that support integration. This article has been written in response to a lack of specific guidance in the industry as to what an integrated EDW actually is and what necessary design elements are needed to achieve integration.

### What Does an Integrated EDW Deliver?

The mission statement for the integrated EDW is to provide the platform for business analysis to be applied consistently across the enterprise. Above all, this mission statement demands consistency across business process subject areas and their associated databases. Consistency requires:

- Detailed textual descriptions of entities such as customers, products, locations, and calendars be applied uniformly across subject areas, using standardized data values. This is a fundamental tenet of MDM.
- Aggregated groupings such as types, categories, flavors, colors, and zones defined within entities have the same interpretations across subject areas. This can be viewed as a higher level requirement on the textual descriptions.
- Constraints posed by business intelligence applications, which attempt to harvest the value of consistent text descriptions and groupings, be applied with identical application logic across subject areas. For instance, constraining on a product category should always be driven from a field named "category" found in the product dimension.
- Numeric facts be represented consistently across subject areas so that it makes sense to combine them in computations and compare them to each other, perhaps with ratios or differences. For example, if revenue is a numeric fact reported from multiple subject areas, the definitions of each of these revenue instances must be the same.
- International differences in languages, location descriptions, time zones, currencies, and business rules be resolved to allow all of the previous consistency requirements to be achieved.
- Auditing, compliance, authentication, and authorization functions be applied in the same way across subject areas.
- Coordination with industry standards be adopted for data content, data exchange, and reporting, where those standards impact the enterprise. Typical standards include ACORD (insurance), MISMO (mortgages), SWIFT and NACHA (financial services), HIPAA and HL7 (health care), RosettaNet (manufacturing), and EDI (procurement).

## Ultimate Litmus Test for Integration

Even an EDW that meets all of the consistency requirements must additionally provide a mechanism for delivering integrated reports and analyses from BI tools, attached to many database instances, possibly hosted on remote, incompatible systems. This is called *drilling across* and is the essential act of the integrated EDW. When we drill across, we gather results from separate business process subject areas and then align or combine these results into a single analysis.

For example, suppose the integrated EDW spans manufacturing, distribution, and retail sales in a business that sells audio/visual systems. Assume that each of these subject areas is supported by a separate transaction processing system. A properly constructed drill-across report could look like Figure 1-1.

| Product Category | Fiscal Period | Manufacturing Finished Inventory (Units) | Distribution Waiting to Return (Units) | Retail Revenue (US Dollars) |
|---|---|---|---|---|
| Consumer Audio | 2008 FP1 | 14,386 | 283 | $15,824,600 |
| Consumer Audio | 2008 FP2 | 17,299 | 177 | $19,028,900 |
| Consumer Video | 2008 FP1 | 8,477 | 85 | $16,106,300 |
| Consumer Video | 2008 FP2 | 9,011 | 60 | $17,120,900 |
| Pro Audio | 2008 FP1 | 2,643 | 18 | $14,536,500 |
| Pro Audio | 2008 FP2 | 2,884 | 24 | $15,862,000 |
| Pro Video | 2008 FP1 | 873 | 13 | $7,158,600 |
| Pro Video | 2008 FP2 | 905 | 11 | $7,421,000 |
| Storage Media | 2008 FP1 | 35,386 | 258 | $1,380,054 |
| Storage Media | 2008 FP2 | 44,207 | 89 | $1,724,073 |

**Figure 1-1:** Drill-across report combining data from three subject area fact tables.

The first two columns are row headers from the product and calendar conformed dimensions, respectively. The remaining three columns each come from separate business process fact tables, namely manufacturing inventory, distribution, and retail sales. This deceptively simple report can only be produced in a properly integrated EDW. In particular, the product and calendar dimensions must be available in all three separate databases, and the category and period attributes within those dimensions must have identical contents and interpretations. Although the metrics in the three fact columns are different, the meaning of the metrics must be consistent across product categories and times.

You must understand and appreciate the tight constraints on the integrated EDW environment demanded by the preceding report. If you don't, you won't understand this article, and you won't have the patience to study the detailed steps described next. Or to put the design challenge in other terms, if you eventually build a successful integrated EDW, you will have visited every issue that follows. With those warnings, read on.

## Organizational Challenges

The integrated EDW deliverables I've described are a daunting list indeed. But for these deliverables to even be possible, the enterprise must make a profound commitment, starting from the executive suite. The separate divisions of the enterprise must have a shared vision of the value of data integration, and they must anticipate the steps of compromise and decision making that will be required. This vision can only come from the senior executives of the enterprise, who must speak very clearly on the value of data integration.

Existing MDM projects provide an enormous boost for the integrated EDW, because presumably the executive team already understands and approves the commitment to building and maintaining master data. A good MDM resource greatly simplifies, but does not eliminate, the need for the EDW team to build the structures necessary for data warehouse integration.

In many organizations, a chicken-and-egg dilemma exists as to whether MDM is required before an integrated EDW is possible or whether the EDW team creates the MDM resources. Often, a low profile EDW effort to build conformed dimensions solely for data warehouse purposes morphs into a full-fledged MDM effort that is on the critical path to supporting mainline operational systems. In my classes since 1993, I have shown a backward pointing arrow leading from cleansed data warehouse data to operational systems. In the early days, we sighed wistfully and wished that the source systems cared about clean, consistent data. Now, more than 15 years later, we seem to be getting our wish!

## Conformed Dimensions and Facts

Since the earliest days of data warehousing, conformed dimensions have been used to consistently label and constrain separate data sources. The idea behind conformed dimensions is very simple: Two dimensions are conformed if they contain one or more common fields whose contents are drawn from the same domains. The result is that constraints and labels have the same content and meaning when applied against separate data sources.

Conformed facts are simply numeric measures that have the same business and mathematical interpretations so that they may be compared and computed against each other consistently.

## Using the Bus Matrix to Communicate with Executives

When you combine the list of EDW subject areas with the notion of conformed dimensions, a powerful diagram emerges, which we call the enterprise data warehouse bus matrix. A typical bus matrix is shown in Figure 1-2.

The business process subject areas are shown along the left side of the matrix and the dimensions are shown across the top. An X marks where a subject area uses the dimension. Note that "subject area" in our vocabulary corresponds to a business process, typically revolving around a transactional data source. Thus, "customer" is not a subject area.

| | Date | Raw Material | Supplier | Plant | Product | Shipper | Warehouse | Customer | Sales Rep | Promotion Deal |
|---|---|---|---|---|---|---|---|---|---|---|
| Raw Material Purchasing | | 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 | X |
| Returns | X | | | | X | X | | X | X | X |

**Figure 1-2:** Bus matrix for a manufacturer's EDW.

At the beginning of an EDW implementation, this bus matrix is very useful as a guide, both to prioritize the development of separate subject areas and to identify the potential scope of the conformed dimensions. The columns of the bus matrix are the invitation list to the conformed dimension design meeting.

Before the conformed dimension design meeting occurs, this bus matrix should be presented to senior management, perhaps in exactly the form of Figure 1-2. Senior management must be able to visualize why these dimensions (master entities) attach to the various business process subject areas, and they must appreciate the organizational challenges of assembling the diverse interest groups together to agree on the conformed dimension content. If senior management is not interested in what the bus matrix implies, then to make a long story short, you have no hope of building an integrated EDW.

It is worth repeating the definition of a conformed dimension at this point to take some of the pressure off of the conforming challenge. Two instances of a dimension are conformed if they contain one or more common fields whose contents are drawn from the same domains. This means that the individual subject area proponents do not have to give up their cherished private descriptive attributes. It merely means that a set of master, universally agreed upon attributes must be established. These master attributes then become the contents of the conformed dimension and become the basis for drilling across.

## Managing the Integrated EDW Backbone

The backbone of the integrated EDW is the set of conformed dimensions and conformed facts. Even if the enterprise executives support the integration initiative and the conformed dimension design meetings go well, there is a lot to the operational management of this backbone. This management can be visualized most clearly by describing two personality archetypes: the dimension manager and the fact provider. Briefly, the dimension manager is a centralized authority who builds and distributes a conformed dimension to the rest of the enterprise, and the fact provider is the client who receives and utilizes the conformed dimension, almost always while managing one or more fact tables within a subject area.

At this point, I must make three fundamental architectural claims to prevent false arguments from arising:

1. The need for dimension managers and fact providers arises solely from the natural reuse of dimensions across multiple fact tables or online analytical processing (OLAP) cubes. Once the EDW community has committed to supporting cross-process analysis, there is no way to avoid all the steps described in this article.
2. Although I describe the handoff from the dimension manager to the fact provider as if it were occurring in a distributed environment where they are remote from each other, their respective roles and responsibilities are the same whether the EDW is fully centralized on a single machine or profoundly distributed across many diverse machines in different locations.
3. The roles of dimension manager and fact provider, although obviously couched in dimension modeling terms, do not arise from a particular modeling persuasion. All of the steps described in this article would be needed in a fully normalized environment.

We are now ready to roll up our sleeves and describe exactly what the dimension manager and fact provider do.

## The Dimension Manager

The dimension manager defines the content and structure of a conformed dimension and delivers that conformed dimension to downstream clients known as fact providers. This role can definitely exist within a master data management (MDM) framework, but the role is much more focused than just being the keeper of the single truth about an entity. The dimension manager has a list of deliverables and responsibilities, all oriented around creating and distributing physical versions of the dimension tables that represent the major entities of the enterprise. In many enterprises, key conformed dimensions include customer, product, service, location, employee, promotion, vendor, and calendar. As I describe the dimension manager's tasks, I will use customer as the example to keep the discussion from being too abstract. The tasks of the customer dimension manager include:

- *Defining the content of the customer dimension.* The dimension manager chairs the design meeting for the conformed customer dimension. At that meeting, all the stakeholders from the customer-facing transaction systems come to agreement on a set of dimensional attributes that everyone will use when drilling across separate subject areas. Remember that these attributes are used as the basis for constraining and grouping customers. Typical conformed customer attributes include type, category, location (multiple fields implementing an address), primary contact (name, title, address), first contact date, credit worthiness, demographic category, and others. Every customer of the enterprise appears in the conformed customer dimension.

- *Receiving notification of new customers*. The dimension manager is the keeper of the master list of dimension members, in this case, customers. The dimension manager must be notified whenever a new customer is registered.
- *Deduplicating the customer dimension*. The dimension manager must deduplicate the master list of customers. Customer lists in the real world are nearly impossible to deduplicate completely. Even when customers are registered through a central MDM process, it is often possible to create duplicates, either for individual customers or business entities.
- *Assigning a unique durable key to each customer*. The dimension manager must identify and keep track of a unique durable key for each customer. Many DBAs automatically assume that this is the "natural key," but quickly choosing the natural key may be the wrong choice. A natural key may not be durable! Using the customer example, if there is any conceivable business rule that could change the natural key over time, then it is not durable. Also, in the absence of a formal MDM process, natural keys can arise from more than one customer-facing process. In this case, different customers could have natural keys of very different formats. Finally, a source system's natural key may be a complex, multifield data structure. For all these reasons, the dimension manager needs to step back from literal natural keys and assign a unique durable key that is completely under the control of the dimension manager. I recommend that this unique, durable key be a simple sequentially assigned integer, with no structure or semantics embedded in the key value.
- *Tracking time variance of customers with type 1, 2, and 3 slowly changing dimensions (SCDs)*. The dimension manager must respond to changes in the conformed attributes describing a customer. Much has been written about tracking the time variance of dimension members using SCDs. A type 1 change overwrites the changed attribute and therefore destroys history. A type 2 change creates a new dimension record for that customer, properly time stamped as of the effective moment of the change. A type 3 change creates a new field in the customer dimension that allows an "alternate reality" to be tracked. The dimension manager updates the customer dimension in response to change notifications received from various sources.
- *Assigning surrogate keys for the customer dimension*. Type 2 is the most common and powerful of the SCD techniques because it provides precise synchronization of a customer description with that customer's transaction history. Because type 2 creates a new record for the same customer, the dimension manager is forced to generalize the customer dimension primary key beyond the unique, durable key. The primary key should be a simple surrogate key, sequentially assigned as needed, with no structure or semantics in the key value. This primary key is separate from the unique durable key, which simply appears

in the dimension as a normal field. The unique, durable key is the glue that binds the separate SCD type 2 records for a single customer together.

- *Handling late-arriving dimension data*. When the dimension manager receives late notification of a type 2 change affecting a customer, special processing is needed. A new dimension record must be created and the effective dates of the change adjusted. The changed attribute must be propagated forward in time through existing dimension records.
- *Providing version numbers for the dimension*. Before releasing a changed dimension to the downstream fact providers, the dimension manager must update the dimension version number if type 1 or type 3 changes have occurred or if late arriving type 2 changes have occurred. The dimension version number does not change if only contemporary type 2 changes have been made since the previous release of the dimension.
- *Adding private attributes to dimensions*. The dimension manager must incorporate private departmental attributes in the release of the dimensions to the fact providers. These attributes are of interest to only a part of the EDW community, perhaps a single department.
- *Building shrunken dimensions as needed*. The dimension manager is responsible for building shrunken dimensions that are needed by fact tables at higher levels of granularity. For example, a customer dimension might be rolled up to the demographic category to support a fact table that reports sales at this level. The dimension manager is responsible for creating this shrunken dimension and assigning its keys.
- *Replicating dimensions to fact providers*. The dimension manager periodically replicates the dimension and its shrunken versions to all the downstream fact providers. All the fact providers should attach the new dimensions to their fact tables at the same time, especially if the version number has changed.
- *Documenting and communicating changes*. The dimension manager maintains metadata and documentation describing all the changes made to the dimension with each release.
- *Coordinating with other dimension managers*. Although each conformed dimension can be administered separately, it makes sense for the dimension managers to coordinate their releases to lessen the impact on the fact providers.

## The Fact Provider

The fact provider sits downstream from the dimension manager and responds to each release of each dimension that is attached to a fact table under the provider's control. Tasks include:

- *Avoiding changes to conformed attributes*. The fact provider must not alter the values of any conformed dimension attributes, or the whole logic of drilling across diverse business process subject areas will be corrupted.
- *Responding to late-arriving dimension updates*. When the fact provider receives late-arriving updates to a dimension, the primary keys of the newly created dimension records must be inserted into all fact tables using that dimension whose time spans overlap the date of the change. If these newly created keys are not inserted into the affected fact tables, the new

dimension record will not tie to the transactional history. The new dimension key must overwrite existing dimension keys in the affected fact tables from the time of the dimension change up to the next dimension change that was already correctly administered.

- *Tying the conformed dimension release to the local dimension*. The dimension manager must deliver to the fact provider a mapping that ties the fact provider's local natural key to the primary surrogate key assigned by the dimension manager. In the surrogate key pipeline (next task), the fact provider replaces the local natural keys in the relevant fact tables with the conformed dimension primary surrogate keys using this mapping.
- *Processing dimensions through surrogate key pipeline*. The fact provider converts the natural keys attached to contemporary transaction records into the correct primary surrogate keys and loads the fact records into the final tables with these surrogate keys.
- *Handling late arriving facts*. The surrogate key pipeline mentioned in the previous paragraph can be implemented in two different ways. Traditionally, the fact provider maintains a current key lookup table for each dimension that ties the natural keys to the contemporary surrogate keys. This works for the most current fact table data where you can be sure that the contemporary surrogate key is the one to use. But the lookup tables cannot be used for late-arriving fact data because it is possible that one or more old surrogate keys must be used. In this traditional approach, the fact provider must revert to an inefficient dimension table lookup to figure out which old surrogate key applies.

  A more modern approach to the surrogate key pipeline implements a dynamic cache of records looked up in the dimension table rather than a separately maintained lookup table. This cache handles contemporary fact records as well as late-arriving fact records with a single mechanism.
- *Synchronizing dimension releases with other fact providers*. It is critically important for all the fact providers to respond to dimension releases at the same time. Otherwise a client application attempting to drill across subject areas will encounter dimensions with different version numbers. See the description of using dimension version numbers in the next section.

## Configuring Business Intelligence (BI) Tools

There is no point in going to all the trouble of setting up dimension managers, fact providers, and conformed dimensions if you aren't going to perform drill-across queries. In other words, you need to "merge sort" separate answer sets on the row headers defined by the values from the conformed dimension attributes. There are many ways to do this in standard BI tools and straight SQL.

You should use dimension version numbers when performing drill-across queries. If the requesting application does not include the version number in the select list, erroneous results are possible because dimension attributes may not be consistent across subject areas. If the requesting application does include the version number in the select list, then at least the results from the fact table queries will end up on separate rows of the answer set, properly labeled by the dimension version. This isn't much consolation to the user, but at least the problem is diagnosed in an obvious way.

Figure 1-3 shows a report drilling across the same three databases as in Figure 1-1, but where a dimension version mismatch occurs. Perhaps the definition of certain product categories has been adjusted between product dimension version 7 and version 8. In this case, the retail sales fact table is using version 8, whereas the other two fact tables are still using version 7. By including the product dimension version attribute in the SQL select list, we automatically avoid merging potentially incompatible data. Such an error would be particularly insidious because without the rows being separated, the result would look perfectly reasonable but could be disastrously misleading.

| Product Category | Product Dimension Version | Manufacturing Finished Inventory (Units) | Distribution Waiting to Return (Units) | Retail Revenue (US Dollars) |
|---|---|---|---|---|
| Consumer Audio | 7 | 14,386 | 283 | |
| Consumer Audio | 8 | | | $15,824,600 |
| Consumer Video | 7 | 8,477 | 85 | |
| Consumer Video | 8 | | | $17,120,900 |

**Figure 1-3:** Drill-across report with a dimension version mismatch.

## Joint Responsibilities

Dimension managers and fact providers must ensure that auditing, compliance, authentication, authorization, and usage tracking functions are applied uniformly for all BI clients. This set of responsibilities is especially challenging because it is outside the scope of the steps described in this article. Even when modern role-enabled authentication and authorization safeguards are in place when using the EDW, subtle differences in the definition of roles may give rise to inconsistency. For example, a role named “senior analyst” may have different interpretations at different entry points to the EDW. The best that can be said for this difficult design challenge is that personnel responsible for defining the LDAP-enabled roles should be invited to the original dimension conforming meetings so that they become aware of the scope of EDW integration.

The integrated EDW promises a rational, consistent view of enterprise data. This promise has been repeated endlessly in the trade literature. But until now, there has been no specific design for actually implementing the integrated EDW. Although this implementation of the integrated EDW must seem daunting, I believe that the steps and responsibilities I have described are basic and unavoidable, no matter how your data warehouse environment is organized. Finally, this architecture represents a distillation of more than two decades’ experience in building data warehouses based on conformed dimensions and facts. If you carefully consider the detailed recommendations in these articles, you should avoid reinventing the wheel when you are building your integrated EDW.

## 1.7 Drill Down to Ask Why

*Ralph Kimball,* DM Review, *Jul 2008 and Aug 2008*

*This content was originally published as two separate articles in the series.*

Boiled down to its essence, the real purpose of a data warehouse is to be the perfect platform for decision making. Most DW and BI architects accept this view, but how many stop and think carefully about what is decision making, exactly? Every DW/BI architect can describe his or her technical architecture, but how many can describe the architecture of decision making? If there even is an architecture of decision making, how does the DW/BI system interact with its components, and what specific demands does decision making place on the DW/BI system?

In 2002, Bill Schmarzo, a former member of the Kimball Group, proposed a very useful architecture for decision making, which he called the *analytic application process*. According to Bill, an analytic application consists of five stages:

1. *Publish reports*. Provide standard operational and managerial "report cards" on the current state of a business.
2. *Identify exceptions*. Reveal the exceptional performance situations to focus attention.
3. *Determine causal factors*. Seek to understand the "why" or root causes behind the identified exceptions.
4. *Model alternatives*. Provide a backdrop to evaluate different decision alternatives.
5. *Track actions*. Evaluate the effectiveness of the recommended actions and feed the decisions back to both the operational systems and DW, against which published reporting will occur, thereby closing the loop.

I have found these analytic application stages to be very useful when I think about the architecture of a DW/BI system. Publishing reports (stage one) is the traditional legacy view of the data warehouse. We pump out reports and we stack them on the business users' desks. There isn't a lot of interactive BI in stage one! We also have been identifying exceptions (stage two) with thresholds, alerts, and red/green blinking graphics for many years. At least in stage two, the choice of which alerts and thresholds we want on our desktops implies some judgment and involvement by the user.

But it is in stage three, where we determine the causal factors behind the exceptions that life really gets interesting. A good DW/BI system should let the decision maker bring his or her full intellectual capital to bear on understanding what the system is bringing to our attention. This stage can be summarized by one all-important word: *why*.

Suppose that you work for an airline as a fare planner. In this role, a critical key performance indicator (KPI) is the yield, which according to Wikipedia is "the revenue or profits from a fixed, perishable resource such as airline seats or hotel room reservations. The challenge is to sell the right resources to the right customer at the right time for the right price."

This morning, in your job as a fare planner, the DW/BI system produces a yield report (stage one) and highlights a number of airline routes for which the yield has dropped significantly (stage two). So, how does the DW/BI system support the all-important stage three? How does the DW/BI system support the fare planner when you ask, "Why are my yields down?"

Imagine five ways in which the fare planner might ask why. I'll arrange these in order of increasing breadth and complexity:

1. *Give me more detail*. Run the same yield report, but break down the high level routes by dates, time of day, aircraft type, fare class, and other attributes of the original yield calculation.

   In a dimensional data warehouse environment where all business process subject areas are built as fact tables and dimension tables on the lowest level atomic data, drilling down is easily accomplished by adding a row header (grouping column) to the report query from any of the attached dimensions. In your yield report, if you assume the report is calculated from a database of individual boarding passes, then drilling down by date, time of day, aircraft type, or fare class is simply a user interface gesture that drags the new grouping attribute from the relevant dimension into the query. Note that this general form of drilling down does not require a predetermined or predeclared hierarchy. As I have been pointing out for 15 years, drilling down has nothing to do with hierarchies!
2. *Give me a comparison*. Run the same yield report, but this time compare to a previous time period or to competitive yield data if it is available.

   In the example, if the route is the row header of the yield report, a common comparison mode would be to add one or more numeric columns to the report, each perhaps with a different date. Thus, the yields at different dates could be compared. This can be accomplished in several different ways by your BI tool, but as the comparisons become more complex, a drill-across approach to sort merging separate queries is the most practical and scalable. See article 1.6, *Essential Steps for the Integrated Enterprise Data Warehouse*, for more information on drill-across techniques. Keep in mind that graphical comparisons may be more effective than straight text reports, especially for time series data.
3. *Let me search for other factors*. Jump to non-yield databases, such as a weather database, holiday/special events database, marketing promotions database, or competitive pricing database to see if any of these exogenous factors could have played a role.

   To jump to a separate database, the context of the current report query must be captured and used as input to the new database. When the user selects a specific row or cell in the original report, that detailed context should be used. For instance, if the June 2008 yield on the San Jose to Chicago route is selected by the user, then June 2008, San Jose, and Chicago should be carried as constraints on the next query. Perhaps the next query focuses on weather. This scenario presents some challenges to the DW/BI designer. The user can't jump to another database if it isn't there. Thus, the designer needs to anticipate and provide possible targets for this kind of jump. Also, the BI tool should support the context capture steps described in this section so that the process of jumping to a new database is as effortless as possible. We describe this capability in article 1.11, *Exploit Your Fact Tables*.

4. *Tell me what explains the variance*. Perform a data mining analysis, perhaps using decision trees, examining hundreds of marketplace conditions to see which of these conditions correlates most strongly with the drop in yield ("explaining the variance" in data mining terminology).

   To feed data mining, the DW/BI designer must again anticipate the database resources that will be needed when these kinds of analyses are performed. Practical information about how to build these kinds of data mining interfaces, including decision trees, is available in Michael Berry and Gordon Linoff's *Data Mining Techniques for Marketing, Sales and Customer Relationship Management* (Wiley, 1997).

5. *Search the web for information about the problem*. Google or Yahoo! the web for "airline yield 2008 versus 2007."

   Finally, if you have successfully searched for other factors (number three), you need to be able to transfer the context of the exception you have identified into a Google or Yahoo! query. If you are dubious that this would be of much value, search for "airline yield 2007 versus 2006." This can be a paradigm shift kind of experience. What we need is a one-button jump from a report cell to a browser.

Twenty years ago, when we drilled down in a data warehouse to ask why, we rarely provided more than the first capability. I like to think of the longer list of all five capabilities as 2008's definition of drilling down to ask why.

In this article, we have reminded ourselves of the true goal of the data warehouse/business intelligence system: assisting the business user in making decisions. The key capability is providing the most flexible and comprehensive ways to drill down and ask why.

## 1.8 Slowly Changing Dimensions

*Ralph Kimball,* DM Review, *Sep 1, 2008 and Oct 1, 2008*

*This content was originally published as two separate articles in the series.*

The notion of time pervades every corner of the data warehouse. Most of the fundamental measurements we store in our fact tables are time series, which we carefully annotate with time stamps and foreign keys connecting to calendar date dimensions. But the effects of time are not isolated just to these activity-based time stamps. All of the other dimensions that connect to fact tables, including fundamental entities such as customer, product, service, terms, facility, and employee, are also affected by the passage of time. As data warehouse managers, we are routinely confronted with revised descriptions of these entities. Sometimes the revised description merely corrects an error in the data. But many times the revised description represents a true change at a point in time of the description of a particular dimension member, such as customer or product. Because these changes arrive unexpectedly, sporadically, and far less frequently than fact table measurements, we call this topic *slowly changing dimensions* (SCDs).

## Three Types of Slowly Changing Dimensions

In more than 30 years of studying the time variance of dimensions, amazingly I have found that the data warehouse only needs three basic responses when confronted with a revised or updated description of a dimension member. I call these, appropriately, types 1, 2, and 3. I'll start with type 1, using the employee dimension to keep the discussion from being too abstract.

### Type 1: Overwrite

Suppose we are notified that the home city attribute for Ralph Kimball in the employee dimension has changed from Santa Cruz to Boulder Creek as of today. Furthermore, we are advised that this is an error correction, not an actual change of location. In this case, we may decide to overwrite the home city field in the employee dimension with the new value. This is a classic type 1 change. Type 1 changes are appropriate for correcting errors and for situations where a conscious choice is made not to track history. And of course, most data warehouses start out with type 1 as the default.

Although the type 1 SCD is the simplest and seemingly cleanest change, there are a number of fine points to think about:

1. Type 1 destroys the history of a particular field. In our example, reports that constrain or group on the home city field will change. Business users will need to be aware that this can happen. The data warehouse needs an explicit, visible policy for type 1 fields that says, "We will correct errors" and/or "We do not maintain history on this field even if it changes."
2. Precomputed aggregates (including materialized views and automatic summary tables) that depend on the home city field must be taken offline at the moment of the overwrite and be recomputed before being brought back online. Aggregates that do not depend on the home city field are unaffected.
3. In financial reporting environments with month-end close processes and in any environment subject to regulatory or legal compliance, type 1 changes may be outlawed. In these cases, the type 2 technique must be used.
4. Overwriting a single dimension field in a relational environment has a pretty small impact but can be disastrous in an OLAP environment if the overwrite causes the cube to be rebuilt. Carefully study your OLAP system reference manual to see how to avoid unexpected cube rebuilds.
5. All distributed copies of the employee dimension, as well as aggregates, must be updated simultaneously across the enterprise when type 1 changes occur, or else the logic of drilling across will be corrupted. In a distributed environment, type 1 (and type 3) changes should force the dimension version number to be updated, and all drill-across applications must include the dimension version number in their queries. This process was described in detail in article 1.6, *Essential Steps for the Integrated Enterprise Data Warehouse*.
6. In a pure type 1 dimension where all fields in the dimension are subject to overwriting, a type 1 change like the home city change for Ralph Kimball will typically affect only one

record (the record for Ralph Kimball). But in a more typical complex environment, where some fields are type 1 and other fields are type 2, the act of overwriting the home city field must overwrite all the records for Ralph Kimball. In other words, type 1 affects all history, not just the current perspective.

### Type 2: Add a New Dimension Record

Let's alter the previous scenario where I overwrote the home city field in Ralph Kimball's employee record to assume that Ralph Kimball actually moved from Santa Cruz to Boulder Creek on July 18, 2008. Assume our policy is to accurately track the employee home addresses in the data warehouse. This is a classic type 2 change.

The type 2 SCD requires that we issue a new employee record for Ralph Kimball effective July 18, 2008. This has many interesting side effects:

1. Type 2 requires that we generalize the primary key of the employee dimension. If Ralph Kimball's employee natural key is G446, then that natural key will be the "glue" that holds Ralph Kimball's multiple records together. I do not recommend creating a smart primary key for type 2 SCDs that contains the literal natural key. The problems with smart keys become especially obvious if you are integrating several incompatible HR systems with differently formatted natural keys. Rather, you should create completely artificial primary keys that are simply sequentially assigned integers. We call these keys *surrogate keys*. You must make a new surrogate primary key whenever you process a type 2 change in a dimension.
2. In addition to the primary surrogate key, I recommend adding five additional fields to a dimension that is undergoing type 2 processing. These fields are shown in Figure 1-4. The date/times are full time stamps that represent the span of time between when the change became effective and when the next change becomes effective. The end effective date/time stamp of a type 2 dimension record must be exactly equal to the begin effective date/time stamp of the next change for that dimension member. The most current dimension record must have an end effective date/time stamp equal to a fictitious date/time far in the future. The effective date key links to a standard calendar date dimension table to view dimension changes filtered on attributes such as fiscal period. The change reason text attribute should be drawn from a preplanned list of reasons for a change, in our example, to the employee attributes (such as employee moved, employee left company, etc.). Finally, the current flag provides a rapid way to isolate exactly the set of dimension members that is in effect at the moment of the query. These five administrative fields allow users and applications to perform many powerful queries.
3. With a dimension undergoing type 2 processing, great care must be taken to use the correct contemporary surrogate keys from this dimension in every affected fact table. This assures that the correct dimension profiles are associated with fact table activity. The extract, transform, and load (ETL) process for aligning the dimension tables with fact tables at load time is called the *surrogate key pipeline* and is covered extensively in Chapter 11.

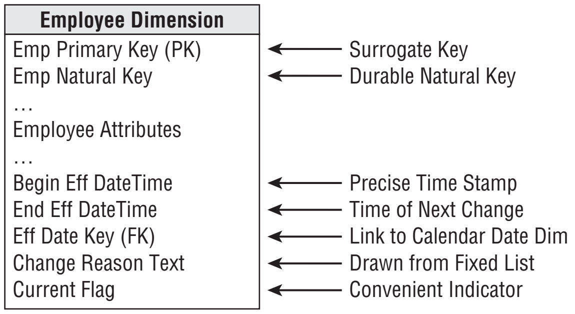

**Figure 1-4:** Employee dimension designed for type 2 SCD.

### Type 3: Add a New Field

Although the type 1 and 2 SCDs are the primary workhorse techniques for responding to changes in a dimension, we need a third technique for handling alternate realities. Unlike physical attributes that can have only one value at a given point in time, some user-assigned attributes can legitimately have more than one assigned value depending on the observer's point of view. For example, a product category can have more than one interpretation. In a stationery store, a marking pen could be assigned to the household goods category or the art supplies category. Users and BI applications need to be able to choose at query time which of these alternate realities applies.

The requirement for an alternate reality view of a dimension attribute usually is accompanied by a subtle requirement that separate versions of reality be available at all times in the past and in the future, even though the request to make these realities visible arrived at the data warehouse today.

In the simplest variation, there is only one alternate reality. In this case, for the product category example, we add a new field in the dimension, perhaps called Alternate Category. If the primary category of our marking pen used to be household goods and now should be art supplies, then in a type 3 treatment, we push the household goods label into the alternate category field and we update the regular category field with art supplies by overwriting. The overwriting step is similar to a type 1 SCD and provokes all the same earlier caveats.

With type 3 machinery in place, users and BI applications can switch seamlessly between these alternate realities. If the environment requires more than one alternate reality, this approach can be generalized by adding more alternate fields, although obviously this approach does not scale gracefully beyond a few choices.

The three SCD approaches to handling time variance in dimensions have enormous applicability in the real-world situations encountered by the data warehouse. Type 2, in particular, allows us to make good on the data warehouse pledge to preserve history faithfully.

## 1.9 Judge Your BI Tool through Your Dimensions

*Ralph Kimball,* DM Review, *Nov 1, 2008*

Dimensions implement the user interface (UI) in a BI tool. In a dimensional DW/BI system, the textual descriptors of all the data warehouse entities like customer, product, location, and time reside in dimension tables. My previous articles carefully described three major types of dimensions according to how the DW/BI system responds to their slowly changing characteristics. Why all this fuss about dimensions? They are the smallest tables in the data warehouse, and the real "meat" is actually the set of numeric measurements in the fact tables. But that argument misses the point that the DW/BI system is always accessed through the dimensions. The dimensions are the gatekeepers, the entry points, the labels, the groupings, the drill-down paths and, ultimately, the texture of the DW/BI system. The actual content of the dimensions determines what is shown on the screen of a BI tool and what UI gestures are possible. That is why we say that the dimensions implement the UI.

This article points out what a good BI tool must be able to do with the dimensions in order to implement an expressive, easy-to-use DW/BI system. I invite you to compare this list against your own BI tools. Almost all of the functions described should be accomplished with single UI gestures, such as dragging an item from a list and dropping it onto a target.

- *Assemble a BI query or report request by first selecting dimension attributes and then selecting facts to be summarized.* This requirement is so basic that it is easy to overlook. One has to be able to see the dimensions and the facts in order to use them. Look for a clean, linear list of all the attributes in your dimensions. Do not accept a normalized snowflaked portrayal, which may appeal to data modelers but is notoriously intimidating to business users. The dimension attributes and the numeric facts should be added to a query or report request with simple UI gestures, as suggested by Figure 1-5.
- *Drill down by adding a row header*. The most fundamental maneuver in a BI tool is drilling down to a more detailed perspective. In almost all cases, drilling down has nothing to do with declared hierarchies. For instance, in Figure 1-5, you can drill down by dragging the promotion type attribute into the results set from the promotion dimension. This would let you see how the individual brands did under different types of promotions. This is drilling down in its most effective form.
- *Browse a dimension to preview permissible values and set constraints*. In Figure 1-5, you should be able to see a list of all the product categories by double-clicking the category attribute in the product dimension. This list should serve both as a preview of what the row labels will be when you use the attribute as a row header, and as a place to set constraints, such as `category = "Candy"`. Make sure you can set multiple picks if you want a short list of simultaneous constraints.
- *Restrict the results of a dimension browse with other constraints in effect*. Sometimes, the list of dimension attribute values is too long to be useful. Make sure that the list can be shortened by applying the constraints you have already set on other attributes in that dimension. An

advanced feature would let you shorten this list even further by traversing the fact table and applying constraints that have been set on other dimensions. For instance, maybe you want a list of product categories that were sold in a particular store in January.

- *Drill across by "accreting" measures under labels defined by conformed dimension attributes*. In article 1.6, *Essential Steps for the Integrated Enterprise Data Warehouse*, I described the architecture of drill-across reports that delivered integrated results from multiple separate fact tables. Such a report is shown in Figure 1-6, where we are drilling across three separate fact tables: shipments, inventory, and sales.

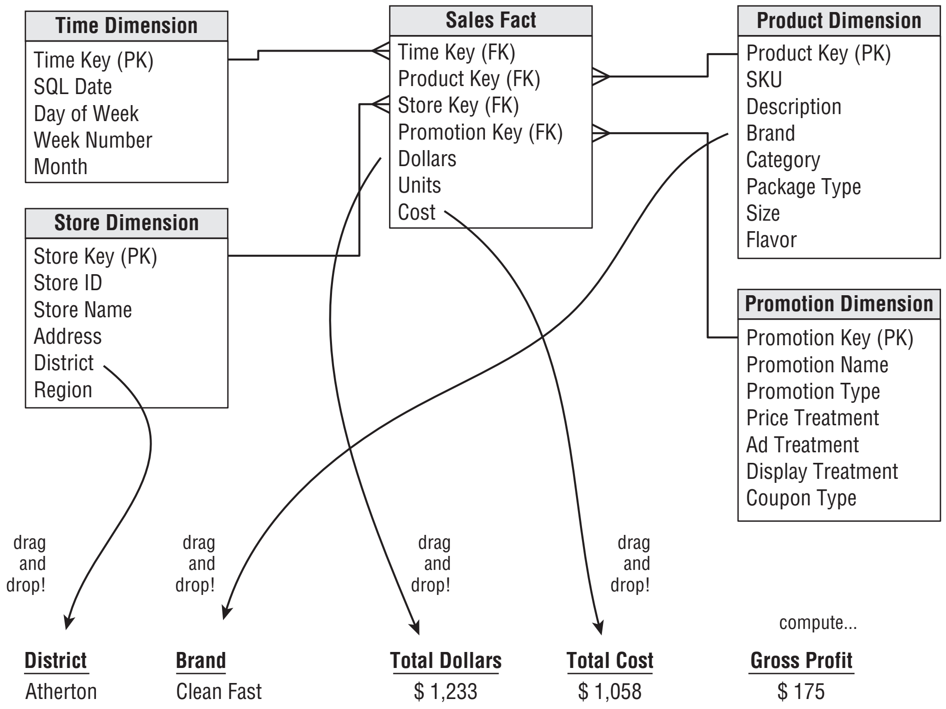

**Figure 1-5:** Adding dimension attributes and numeric facts through simple user interface gestures.

| Product Name | Manufacturing Shipments | Warehouse Inventory | Retail Sales |
|---|---|---|---|
| Frame | 2,940 | 1,887 | $761 |
| Toggle | 13,338 | 9,376 | $2,448 |
| Widget | 7,566 | 5,748 | $2,559 |

**Figure 1-6:** Sample drill-across report.

The first column in Figure 1-6 is the product name, which comes from a product dimension that must be attached to each fact table separately. The product name is a conformed attribute because it has the same column name and content in all three product dimensions, which could well be located on physically separate machines. The challenge for a BI tool is to allow the user or report designer to "pin" the product name as a row header and then systematically visit various possible fact tables, dragging separate fact columns into the report.

Stepping back from these BI tool requirements, we can judge the maturity and experience of the BI tool by first asking whether these maneuvers are even possible, and then asking how easy they are to use. I think it is amazing that some BI tools still don't "get it"; I have talked about these capabilities for 20 years, and we had most of these functions in the Metaphor tool suite starting in 1984.

Judge the ease of use with a simple test—count the clicks. A double-click or click-and-drag counts as a single click. To my way of thinking, one click counts as outstanding, two clicks is pretty good, three clicks is marginal, and more than three clicks is unacceptable.

Modern BI tools have lots more features than those I have described in this article, but what good are advanced features if the BI tool can't support the basic maneuvers? Put your tool to the test.

## 1.10 Fact Tables

*Ralph Kimball,* DM Review, *Dec 1, 2008*

Fact tables are the foundation of the data warehouse. They contain the fundamental measurements of the enterprise, and they are the ultimate target of most data warehouse queries. Perhaps you are wondering why it took me so long to get to fact tables in this series of articles. Well, there is no point in hoisting fact tables up the flagpole unless they have been chosen to reflect urgent business priorities, have been carefully quality assured, and are surrounded by dimensions that provide a wealth of entry points for constraining and grouping. Now that we have paved the way for fact tables, let's see how to build them and use them.

### Stay True to the Grain

The first and most important design step is declaring the fact table grain. The grain is the business definition of what a single fact table record represents. The grain declaration is not a list of dimensional foreign keys that implement a primary key for the fact table. Rather the grain is the description of the measurement event in the physical world that gives rise to a measurement. When the grocery store scanner measures the quantity and the charged price of a product being purchased, the grain is literally the beep of the scanner. That is a great grain definition!

Immediately after declaring the grain, it is possible to list the dimensional foreign keys that exist at that grain. By declaring the grain first, the discussion of foreign keys remains grounded and precise.

The real purpose of the fact table is to be the repository of the numeric facts that are observed during the measurement event. It is critically important for these facts to be true to the grain. The grocery store "beep" measures the quantity and extended price of the product being scanned. We never include other numeric measurements that violate the grain, such as the overall category sales or the sales of this product last month. Even though these other measurements might be narrowly helpful for selected calculations, they cannot be combined across fact records and they introduce weird asymmetries in the design of applications. We let our BI tools compute these off-topic values at query time rather than hard coding them into our fact tables.

We always strive to make the facts additive across the dimensions and exactly consistent with the grain. Notice that we don't store the price of the product being scanned because the price is nonadditive. Rather, we store the extended price (unit price multiplied by quantity sold), which can be added freely across products, stores, times, and all the other dimensions.

## Build Up from the Lowest Possible Grain

The data warehouse should always be built on fact tables expressed at the lowest possible grain. In the example, the beep of the grocery store cash register is the lowest possible grain because it cannot be divided any further. Fact tables at the lowest grain are the most expressive because they have the most complete set of possible dimensions for that business process. The beep grain fact table could have date, store, product, cashier, manager, customer, promotion, competition, basket, and even weather if all these data sources can be marshaled when the fact records are created. Higher grain aggregated tables such as category sales by district cannot support all these dimensions and therefore are much less expressive. It is a fundamental mistake to publish only aggregated tables to the business users without making the lowest grain fact tables smoothly accessible by drilling down. Most of the false notions that dimensional tables presuppose the business question come from making this fundamental mistake.

## Three Kinds of Fact Tables

If you stay true to the grain, all of your fact tables can be grouped into just three types: transaction grain, periodic snapshot grain, and accumulating snapshot grain, as shown in Figure 1-7. In Figure 1-7, the dimensions are designated by foreign keys (FK) and the numeric facts are italicized.

| Transaction Grain |
|---|
| Date Key (FK) |
| Product Key (FK) |
| Store Key (FK) |
| Customer Key (FK) |
| Cashier Key (FK) |
| Manager Key (FK) |
| Promotion Key (FK) |
| Weather Key (FK) |
| Transaction # (DD) |
| *Quantity* |
| *Extended Price* |

| Periodic Snapshot Grain |
|---|
| Month Key (FK) |
| Account Key (FK) |
| Branch Key (FK) |
| Household Key (FK) |
| *Balance* |
| *Fees Paid* |
| *Interest Earned* |
| *Transaction Count* |

| Accumulating Snapshot Grain |
|---|
| Order Date Key (FK) |
| Ship Date Key (FK) |
| Delivery Date Key (FK) |
| Payment Date Key (FK) |
| Return Date Key (FK) |
| Warehouse Key (FK) |
| Customer Key (FK) |
| Promotion Key (FK) |
| Order Status Key (FK) |
| *Quantity* |
| *Extended List Price* |
| *Discount* |
| *Extended Net Price* |

**Figure 1-7:** The three different types of fact tables.

The transaction grain corresponds to a measurement taken at a single instant. The grocery store beep is a transaction grain. The measured facts are valid only for that instant and event. The next measurement event could happen one millisecond later or next month or never. Thus, transaction grain fact tables are unpredictably sparse or dense. We have no guarantee that all the possible foreign keys will be represented. Transaction grain fact tables can be enormous, with the largest containing many billions of records.

The periodic snapshot grain corresponds to a predefined span of time, often a financial reporting period. Figure 1-7 illustrates a monthly account periodic snapshot. The measured facts summarize activity during or at the end of the time span. The periodic snapshot grain carries a powerful guarantee that all of the reporting entities will appear in each snapshot, even if there is no activity. The periodic snapshot is predictably dense, and applications can rely on combinations of keys always being present. Periodic snapshot fact tables can also get large. A bank with 20 million accounts and a 10-year history would have 2.4 billion records in the monthly account periodic snapshot!

The accumulating snapshot grain fact table corresponds to a predictable process that has a well-defined beginning and end. Order processing, claims processing, service call resolution, and college admissions are typical candidates. The grain of an accumulating snapshot for order processing, for example, is usually the line item on the order. Notice in Figure 1-7 that there are multiple dates representing the standard scenario that an order undergoes. Accumulating snapshot records are revisited and overwritten as the process progresses through its steps from beginning to end. Accumulating snapshot fact tables generally are much smaller than the other two types because of this overwriting strategy.

## 1.11 Exploit Your Fact Tables

*Ralph Kimball,* DM Review, *Jan/Feb 2009*

Article 1.10, *Fact Tables*, described the three types of fact tables that are all you will ever need in your data warehouse. The secret to this simple observation is adhering fanatically to the grain. A fact table records measurement events, and as long as we only record one kind of measurement event in a given fact table, we only need the three basic types: transaction grain, periodic snapshot grain, and accumulating snapshot grain. In this article, I describe basic ways to exploit these clean fact table designs in the front room and in the back room.

### Front Room: Aggregate Navigation

I've previously described that the most atomic grain of a measurement process is the most expressive. More dimensions can be attached to the atomic grain than can be attached to higher aggregated levels. The DW/BI team should expose the atomic grain of a business process to the business users and application designers, allowing the data warehouse to choose aggregated levels of the data at run time, not design time. Thus, in our grocery store "beep" grain example, if the user's query asks for a category total, the database chooses a category level aggregated fact table silently and behind the scenes at run time. In this way, aggregate tables are like indexes. Reports and queries should be designed without specific reference to the aggregate tables. If the user asks for a category total of specific products packaged in glass containers, the database cannot use the simple category aggregate table and gracefully assembles the answer from the atomic fact table.

## Front Room: Drilling Across Different Grains

If you are good at visualizing dimensional schemas, you will understand that you can drill across multiple fact tables at different grains as long as you choose conformed dimension attributes for the answer set row headers that exist for all the fact tables in your integrated query. For example, if you have a sales fact table in the grocery store whose grain is every individual transaction, and you also have a monthly brand forecast table whose grain is product brand by month, you can perform a drill-across query on these two tables as long as all the dimensional references in the SELECT list and in the query constraints refer only to brands and months.

## Front Room: Exporting Constraints to Different Business Processes

If you have a sophisticated data warehouse, you may be able to support the capabilities in the previous two sections. But there's one absolute bedrock of decision support I bet you can't do. Suppose you have drilled down in the grocery store database and found a product brand that seems to be selling poorly in certain regions. Your instinct is aroused, and you ask, "What were the merchandising deals we did with that manufacturer in those regions and in the time period I am concerned with?" In most data warehouses, you would have to scribble down the brand, region, and time period information on a piece of paper, close the current application, open another user interface, and reenter this information. But why can't you simply select one or more offending rows in the first application and copy/paste them with a single user interface gesture into the second application? The context of the selection would apply the appropriate constraints directly. I know this is possible because I have built query tools that do this.

## Back Room: Fact Table Surrogate Keys

Surrogate keys are a staple of dimension table design, but surprisingly, there are times when we want a surrogate key in a fact table. Remember that a surrogate key is just a simple integer key, assigned in sequence as records are created. Although fact table surrogate keys (FSKs) have no business meaning in the front room, they have a number of uses in the back room:

- FSKs uniquely and immediately identify single fact records.
- Because FSKs are assigned sequentially, a load job inserting new records will have FSKs in a contiguous range.
- An FSK allows updates to be replaced by insert-deletes.
- Finally, an FSK can become a foreign key in a fact table at a lower grain.