

Fact Tables and Dimension Tables

**The logical foundation of dimensional modeling**

by Ralph Kimball Dimensional modeling is a design discipline that straddles the formal relational model and the

engineering realities of text and number data. Compared to entity/relation modeling, it's less rigorous (allowing the designer more discretion in organizing the tables) but more practical because it accommodates database complexity and improves performance. Contrasted with other modeling disciplines, dimensional modeling has developed an extensive portfolio of techniques for handling real-world situations.

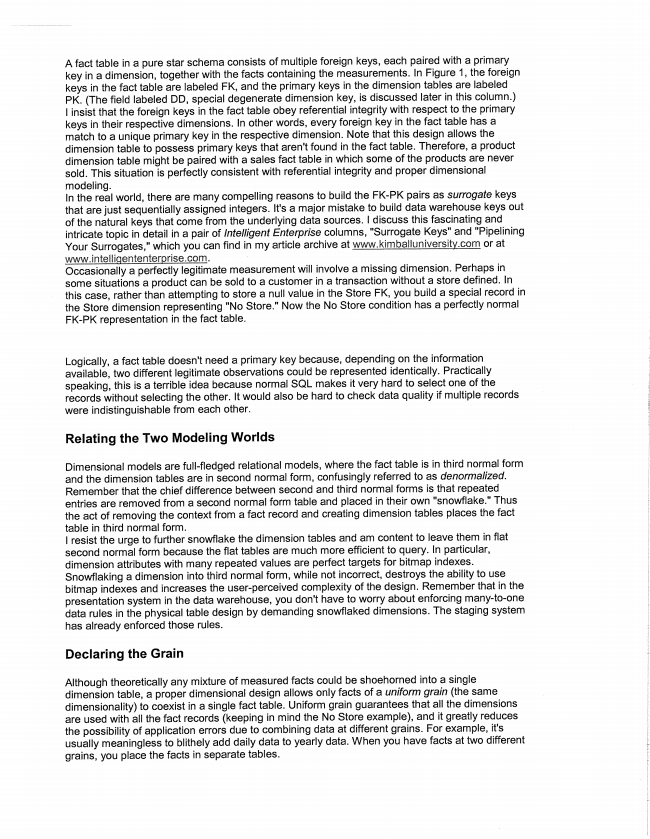
**Measurements and Context**

Dimensional modeling begins by dividing the world into measurements and context. Measurements are usually numeric and taken repeatedly. Numeric measurements are facts. Facts are always surrounded by mostly textual context that's true at the moment the fact is recorded. Facts are very specific, well-defined numeric attributes. By contrast, the context surrounding the facts is open-ended and verbose. It's not uncommon for the designer to add context to a set of facts partway through the implementation. Although you could lump all context into a wide, logical record associated with each measured fact, you'll usually find it convenient and intuitive to divide the context into independent logical clumps. When you record facts — dollar sales of a grocery store purchase of an individual product, for example — you naturally divide the context into clumps named Product, Store, Time, Customer, Clerk, and several others. We call these logical clumps dimensions and assume informally that these dimensions are independent. Figure 1 shows the dimensional model for a typical grocery store fact. In truth, dimensions rarely are completely independent in a strong statistical sense. In the grocery store example, Customer and Store clearly will show a statistical correlation. But it's usually the right decision to model Customer and Store as separate dimensions. A single, combined dimension would likely be unwieldy with tens of millions of rows. And the record of when a given customer shopped in a given store would be expressed more naturally in a fact table that also showed the Time dimension. The assumption of dimension independence would mean that all the dimensions, such as Product, Store, and Customer, are independent of Time. But you have to account for the slow, episodic change of these dimensions in the way you handle them. In effect, as keepers of the data warehouse, we have taken a pledge to faithfully represent these changes. This predicament gives rise to the technique of slowly changing dimensions, the subject of the next column in this

Series.

**Dimensional Keys**

If the facts are truly measures taken repeatedly, you find that fact tables always create a characteristic many-to-many relationship among the dimensions. Many customers buy many products in many stores at many times. Therefore, you logically model measurements as fact tables with multiple foreign keys referring to the contextual entities. And the contextual entities are each dimensions with a single primary key. (See Figure 1.) Although you can separate the logical design from the physical design, in a relational database fact tables and dimension tables are most often explicit tables. Actually, a real relational database has two levels of physical design. At the higher level, tables are explicitly declared together with their fields and keys. The lower level of physical design describes the way the bits are organized on the disk and in memory. Not only is this design highly dependent on the particular database, but some implementations may even "invert" the database beneath the level of table declarations and store the bits in ways that are not directly related to the higher-level physical records. What follows is a discussion of the higher level only.



A fact table in a pure star schema consists of multiple foreign keys, each paired with a primary key in a dimension, together with the facts containing the measurements. In Figure 1, the foreign keys in the fact table are labeled FK, and the primary keys in the dimension tables are labeled PK. (The field labeled DD, special degenerate dimension key, is discussed later in this column.) | insist that the foreign keys in the fact table obey referential integrity with respect to the primary keys in their respective dimensions. In other words, every foreign key in the fact table has a match to a unique primary key in the respective dimension. Note that this design allows the dimension table to possess primary keys that aren't found in the fact table. Therefore, a product dimension table might be paired with a sales fact table in which some of the products are never sold. This situation is perfectly consistent with referential integrity and proper dimensional modeling. In the real world, there are many compelling reasons to build the FK-PK pairs as surrogate keys that are just sequentially assigned integers. It's a major mistake to build data warehouse keys out of the natural keys that come from the underlying data sources. I discuss this fascinating and intricate topic in detail in a pair of Intelligent Enterprise columns, "Surrogate Keys" and "Pipelining Your Surrogates," which you can find in my article archive at www.kimballuniversity.com or at www.intelligententerprise.com. Occasionally a perfectly legitimate measurement will involve a missing dimension. Perhaps in some situations a product can be sold to a customer in a transaction without a store defined. In this case, rather than attempting to store a null value in the Store FK, you build a special record in the Store dimension representing "No Store." Now the No Store condition has a perfectly normal FK-PK representation in the fact table.

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

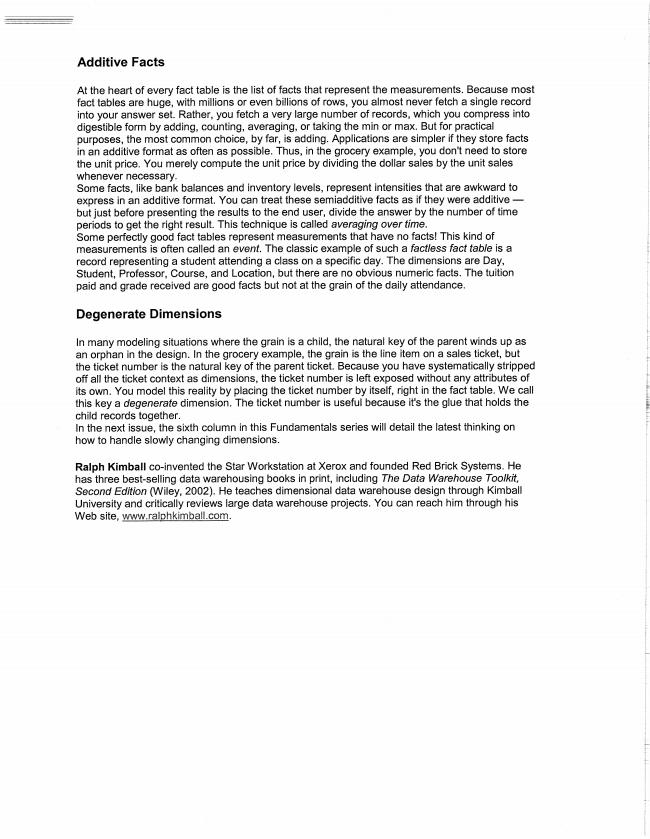
were indistinguishable from each other.

**Relating the Two Modeling Worlds**

Dimensional models are full-fledged relational models, where the fact table is in third normal form and the dimension tables are in second normal form, confusingly referred to as denormalized. Remember that the chief difference between second and third normal forms is that repeated entries are removed from a second normal form table and placed in their own "snowflake." Thus the act of removing the context from a fact record and creating dimension tables places the fact table in third normal form. | resist the urge to further snowflake the dimension tables and am content to leave them in flat second normal form because the flat tables are much more efficient to query. In particular, dimension attributes with many repeated values are perfect targets for bitmap indexes. Snowflaking a dimension into third normal form, while not incorrect, destroys the ability to use bitmap indexes and increases the user-perceived complexity of the design. Remember that in the presentation system in the data warehouse, you don't have to worry about enforcing many-to-one data rules in the physical table design by demanding snowflaked dimensions. The staging system has already enforced those rules.

**Declaring the Grain**

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



**Additive Facts**

At the heart of every fact table is the list of facts that represent the measurements. Because most fact tables are huge, with millions or even billions of rows, you almost never fetch a single record into your answer set. Rather, you fetch a very large number of records, which you compress into digestible form by adding, counting, averaging, or taking the min or max. But for practical purposes, the most common choice, by far, is adding. Applications are simpler if they store facts in an additive format as often as possible. Thus, in the grocery example, you don't need to store the unit price. You merely compute the unit price by dividing the dollar sales by the unit sales whenever necessary. Some facts, like bank balances and inventory levels, represent intensities that are awkward to express in an additive format. You can treat these semiadditive facts as if they were additive — but just before presenting the results to the end user, divide the answer by the number of time periods to get the right result. This technique is called averaging over time. Some perfectly good fact tables represent measurements that have no facts! This kind of measurements is often called an event. The classic example of such a factless fact table is a record representing a student attending a class on a specific day. The dimensions are Day, Student, Professor, Course, and Location, but there are no obvious numeric facts. The tuition paid and grade received are good facts but not at the grain of the daily attendance.

**Degenerate Dimensions**

In many modeling situations where the grain is a child, the natural key of the parent winds up as an orphan in the design. In the grocery example, the grain is the line item on a sales ticket, but the ticket number is the natural key of the parent ticket. Because you have systematically stripped off all the ticket context as dimensions, the ticket number is left exposed without any attributes of its own. You model this reality by placing the ticket number by itself, right in the fact table. We call this key a degenerate dimension. The ticket number is useful because it's the glue that holds the child records together. In the next issue, the sixth column in this Fundamentals series will detail the latest thinking on how to handle slowly changing dimensions.

Ralph Kimball co-invented the Star Workstation at Xerox and founded Red Brick Systems. He has three best-selling data warehousing books in print, including The Data Warehouse Toolkit, Second Edition (Wiley, 2002). He teaches dimensional data warehouse design through Kimball University and critically reviews large data warehouse projects. You can reach him through his

Web site, www.ralphkimball.com.

三