# A Dimensional Modeling Manifesto

By Ralph
Kimball

**Drawing the Line Between Dimensional Modeling and ER Modeling Techniques**

Dimensional modeling (DM) is the name of a logical design technique often used for data warehouses. It is different from, and contrasts with, entity-relation modeling (ER). This article points out the many differences between the two techniques and draws a line in the sand. DM is the only viable technique for databases that are designed to support end-user queries in a data warehouse. ER is very useful for the transaction capture and the data administration phases of constructing a data warehouse, but it should be avoided for end-user delivery.

## What is ER?

ER is a logical design technique that seeks to remove the redundancy in data. Imagine that we have a business that takes orders and sells products to customers. In the early days of computing (long before relational databases) when we first transferred this data to a computer, we probably captured the original paper order as a single fat record with many fields. Such a record could easily have been 1,000 bytes distributed across 50 fields. The line items of the order were probably represented as a repeating group of fields embedded in the master record. Having this data on the computer was very useful, but we quickly learned some basic lessons about storing and manipulating data. One of the lessons we learned was that data in this form was difficult to keep consistent because each record stood on its own. The customer's name and address appeared many times, because this data was repeated whenever a new order was taken. Inconsistencies in the data were rampant, because all of the instances of the customer address were independent, and updating the customer's address was a messy transaction.

Even in the early days, we learned to separate out the redundant data into distinct tables, such as a customer master and a product master — but we paid a price. Our software systems for retrieving and manipulating the data became complex and inefficient because they required careful attention to the processing algorithms for linking these sets of tables together. We needed a database system that was very good at linking tables. This paved the way for the relational database revolution, where the database was devoted to just this task.

The relational database revolution bloomed in the mid 1980s. Most of us learned what a relational database was by reading Chris Date's seminal book on the subject, *An Introduction to Relational Databases* (Addison-Wesley), first published in the early 1980s. As we paged through Chris's book, we worked through all of his Parts, Suppliers, and Cities database examples. It didn't occur to most of us to ask whether the data was completely "normalized" or whether any of the tables could be "snowflaked," and Chris didn't develop these topics. In my opinion, Chris was trying to explain the more fundamental concepts of how to think about tables that were relationally joined. ER modeling and normalization were developed in later years as the industry shifted its attention to transaction processing.

The ER modeling technique is a discipline used to illuminate the microscopic relationships among data elements. The highest art form of ER modeling is to remove all redundancy in the data. This is immensely beneficial to transaction processing because transactions are made very simple and deterministic. The transaction of updating a customer's address may devolve to a single record lookup in a customer address master table. This lookup is controlled by a customer address key, which defines uniqueness of the customer address record and allows an indexed lookup that is extremely fast. It is safe to say that the success of transaction processing in relational databases is mostly due to the discipline of ER modeling.

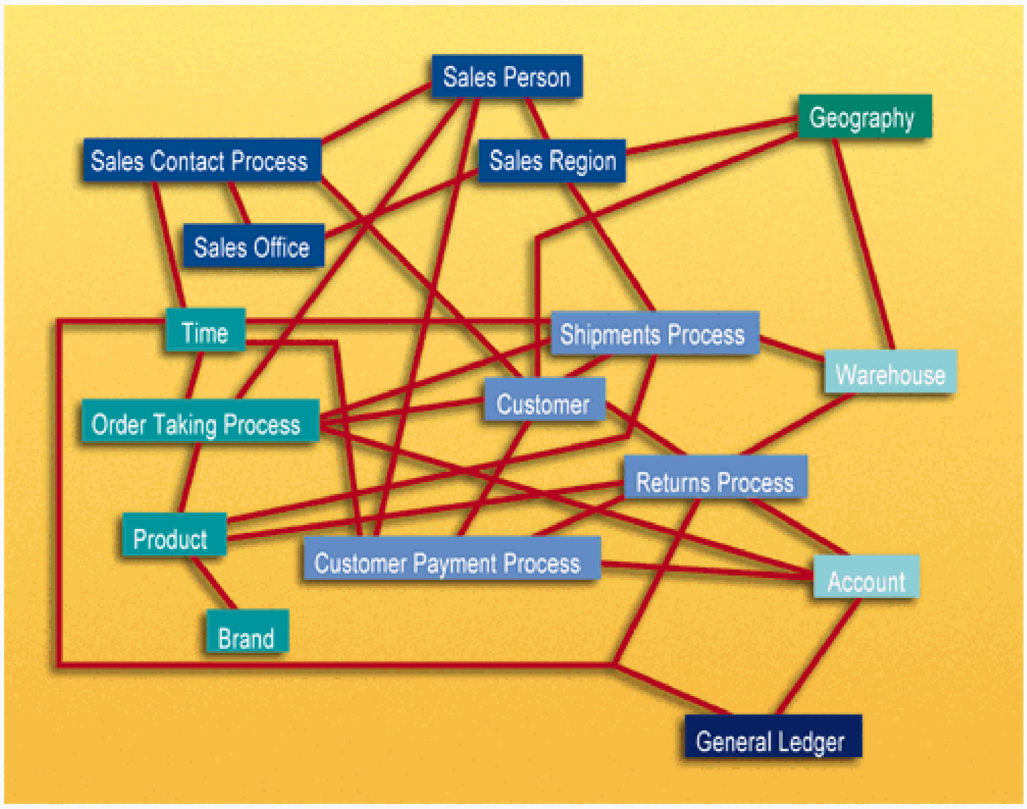

Figure 1. A high-level overview of an ER model. Each box on the diagram is actually many entities. Each business process shown is probably a separate legacy application. The equivalent DM design would isolate each business process and surround it with just its relevant dimensions.

However, in our zeal to make transaction processing efficient, we have lost sight of our original, most important goal. We have created databases that cannot be queried! Even our simple order-taking example creates a database of dozens of tables that are linked together by a bewildering spider web of joins. (See Figure 1) All of us are familiar with the big chart on the wall of the IS database designer's cubicle. The ER model for the enterprise has hundreds of logical entities! High-end systems such as SAP have thousands of entities. Each of these entities usually turns into a physical table when the database is implemented. This situation is not just an annoyance, it is a showstopper:

- End users cannot understand or remember an ER model. End users cannot navigate an ER model. There is no graphical user interface (GUI) that takes a general ER model and makes it usable by end users.
- Software cannot usefully query a general ER model. Cost-based optimizers that attempt to do this are notorious for making the wrong choices, with disastrous consequences for performance.
- Use of the ER modeling technique defeats the basic allure of data warehousing, namely intuitive and high-performance retrieval of data.

Ever since the beginning of the relational database revolution, IS shops have noticed this problem. Many of them that have tried to deliver data to end users have recognized the impossibility of presenting these immensely complex schemas to end users, and many of these IS shops have stepped back to attempt "simpler designs." I find it striking that these "simpler" designs all look very similar! Almost all of these simpler designs can be thought of as "dimensional." In a natural, almost unconscious way, hundreds of IS designers have returned to the roots of the original relational model because they know the database cannot be used unless it is packaged simply. It is probably accurate to say that this natural dimensional approach was not invented by any single person. It is an irresistible force in the design of databases that will always appear when the designer places understandability and performance as the highest goals. We are now ready to define the DM approach.

## What is DM?

DM is a logical design technique that seeks to present the data in a standard, intuitive framework that allows for high-performance access. It is inherently dimensional, and it adheres to a discipline that uses the relational model with some important restrictions. Every dimensional model is composed of one table with a multipart key, called the fact table, and a set of smaller tables called dimension tables. Each dimension table has a single-part primary key that corresponds exactly to one of the components of the multipart key in the fact table. (See Figure 2.) This characteristic "star-like" structure is often called a star join. The term star join dates back to the earliest days of relational databases.

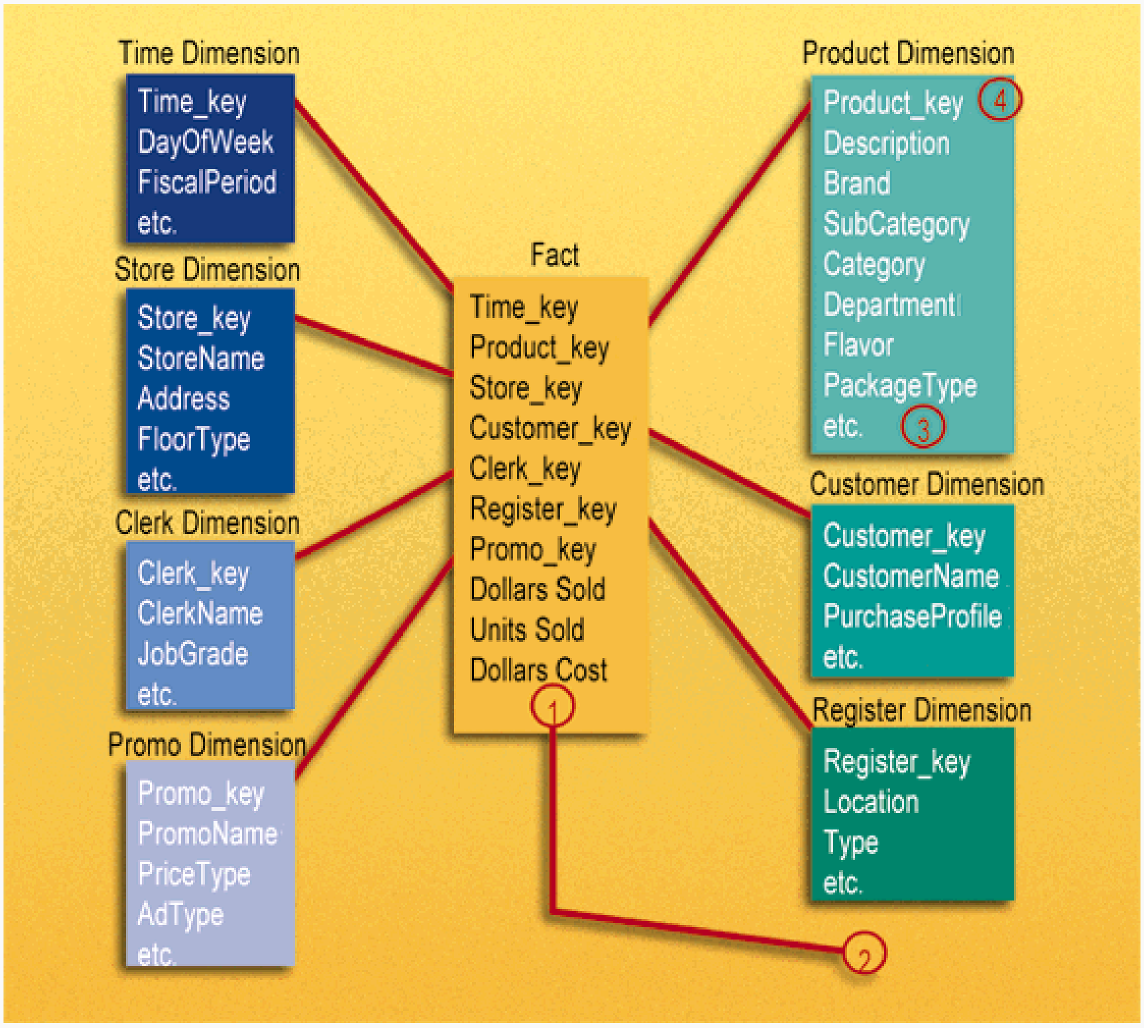

Figure 2. A detailed dimensional model for retail point of sales. Numbers 1 through 4 show places where the design may be gracefully extended.

A fact table, because it has a multipart primary key made up of two or more foreign keys, always expresses a many-to-many relationship. The most useful fact tables also contain one or more numerical measures, or "facts," that occur for the combination of keys that define each record. In Figure 2, the facts are Dollars Sold, Units Sold, and Dollars Cost. The most useful facts in a fact table are numeric and additive. Additivity is crucial because data warehouse applications almost never retrieve a single fact table record; rather, they fetch back hundreds, thousands, or even millions of these records at a time, and the only useful thing to do with so many records is to add them up.

Dimension tables, by contrast, most often contain descriptive textual information. Dimension attributes are used as the source of most of the interesting constraints in data warehouse queries, and they are virtually always the source of the row headers in the SQL answer set. In Figure 2, we constrain on the Lemon flavored products via the Flavor attribute in the Product table, and on Radio promotions via the AdType attribute in the Promotion table. It should be obvious that the power of the database in Figure 2 is proportional to the quality and depth of the dimension tables.

The charm of the database design in Figure 2 is that it is highly recognizable to the end users in the particular business. I have observed literally hundreds of instances where end users agree immediately that this is "their business."

## DM vs. ER

Obviously, Figure 1 and Figure 2 look quite different. Many designers react to this by saying, “There must be less information in the star join,” or “The star join is only used for high-level summaries.” Both of these statements are false.

The key to understanding the relationship between DM and ER is that a single ER diagram breaks down into multiple DM diagrams. Think of a large ER diagram as representing every possible business process in the enterprise. The master ER diagram may have Sales Calls, Order Entries, Shipment Invoices, Customer Payments, and Product Returns, all on the same diagram. In a way, the ER diagram does itself a disservice by representing on one diagram multiple processes that never coexist in a single data set at a single consistent point in time. It’s no wonder the ER diagram is overly complex. Thus the first step in converting an ER diagram to a set of DM diagrams is to separate the ER diagram into its discrete business processes and to model each one separately.

The second step is to select those many-to-many relationships in the ER model containing numeric and additive nonkey facts and to designate them as fact tables. The third step is to denormalize all of the remaining tables into flat tables with single-part keys that connect directly to the fact tables. These tables become the dimension tables. In cases where a dimension table connects to more than one fact table, we represent this same dimension table in both schemas, and we refer to the dimension tables as “conformed” between the two dimensional models.

The resulting master DM model of a data warehouse for a large enterprise will consist of somewhere between 10 and 25 very similar-looking star join schemas. Each star join will have four to 12 dimension tables. If the design has been done correctly, many of these dimension tables will be shared from fact table to fact table. Applications that drill down will simply be adding more dimension attributes to the SQL answer set from within a single star join. Applications that drill across will simply be linking separate fact tables together through the conformed (shared) dimensions. Even though the overall suite of star join schemas in the enterprise dimensional model is complex, the query processing is very predictable because at the lowest level, I recommend that each fact table should be queried independently.

## The Strengths of DM

The dimensional model has a number of important data warehouse advantages that the ER model lacks. First, the dimensional model is a predictable, standard framework. Report writers, query tools, and user interfaces can all make strong assumptions about the dimensional model to make the user interfaces more understandable and to make processing more efficient. For instance, because nearly all of the constraints set up by the end user come from the dimension tables, an end-user tool can provide high-performance “browsing” across the attributes within a dimension via the use of bit vector indexes. Metadata can use the known cardinality of values in a dimension to guide the user-interface behavior. The predictable framework offers immense advantages in processing. Rather than using a cost-based optimizer, a database engine can make very strong assumptions about first constraining the dimension tables and then “attacking” the fact table all at once with the Cartesian product of those dimension table keys satisfying the user’s constraints. Amazingly, by using this approach it is possible to evaluate arbitrary n-way joins to a fact table in a single pass through the fact table’s index. We are so used to thinking of n-way joins as “hard” that a whole generation of DBAs doesn’t realize that the n-way join problem is formally equivalent to a single sort-merge. Really.

A second strength of the dimensional model is that the predictable framework of the star join schema withstands unexpected changes in user behavior. Every dimension is equivalent. All dimensions can be thought of as symmetrically equal entry points into the fact table. The logical design can be done independent of expected query patterns. The user interfaces are symmetrical, the query strategies are symmetrical, and the SQL generated against the dimensional model is symmetrical.

A third strength of the dimensional model is that it is gracefully extensible to accommodate unexpected new data elements and new design decisions. When we say gracefully extensible, we mean several things. First, all existing tables (both fact and dimension) can be changed in place by simply adding new data rows in the table, or the table can be changed in place with a SQL alter table command. Data should not have to be reloaded. Graceful extensibility also means that that no query tool or reporting tool needs to be reprogrammed to accommodate the change. And finally, graceful extensibility means that all old applications continue to run without yielding different results. In Figure 2, I labeled the schema with the numbers 1 through 4 indicating where you can, respectively, make the following graceful changes to the design after the data warehouse is up and running by:

1. Adding new unanticipated facts (that is, new additive numeric fields in the fact table), as long as they are consistent with the fundamental grain of the existing fact table
2. Adding completely new dimensions, as long as there is a single value of that dimension defined for each existing fact record
3. Adding new, unanticipated dimensional attributes
4. Breaking existing dimension records down to a lower level of granularity from a certain point in time forward.

A fourth strength of the dimensional model is that there is a body of standard approaches for handling common modeling situations in the business world. Each of these situations has a well-understood set of alternatives that can be specifically programmed in report writers, query tools, and other user interfaces. These modeling situations include:

- Slowly changing dimensions, where a “constant” dimension such as Product or Customer actually evolves slowly and asynchronously. Dimensional modeling provides specific techniques for handling slowly changing dimensions, depending on the business environment. See my DBMS article of April 1996 on slowly changing dimensions.
- Heterogeneous products, where a business such as a bank needs to track a number of different lines of business together within a single common set of attributes and facts, but at the same time it needs to describe and measure the individual lines of business in highly idiosyncratic ways using incompatible measures.
- Pay-in-advance databases, where the transactions of a business are not little pieces of revenue, but the business needs to look at the individual transactions as well as report on revenue on a regular basis. For this and the previous bullet, see my DBMS article of December 1995, the insurance company case study.
- Event-handling databases, where the fact table usually turns out to be “factless.” See my DBMS article of September 1996 on factless fact tables.

A final strength of the dimensional model is the growing body of administrative utilities and software processes tha manage and use aggregates. Recall that aggregates are summary records that are logically redundant with base data already in the data warehouse, but they are used to enhance query performance. A comprehensive aggregate strategy is required in every medium- and large-sized data warehouse implementation. To put it another way, if you don’t have aggregates, then you are potentially wasting millions of dollars on hardware upgrades to solve performance problems that could be otherwise addressed by aggregates.

All of the aggregate management software packages and aggregate navigation utilities depend on a very specific single structure of fact and dimension tables that is absolutely dependent on the dimensional model. If you don’t adhere to the dimensional approach, you cannot benefit from these tools. Please see my DBMS articles on aggregate navigation and the various products serving aggregate navigation in the September 1995 and August 1996 issues.

## Myths About DM

A few myths floating around about dimensional modeling deserve to be addressed. Myth number one is “Implementing a dimensional data model will lead to stovepipe decision-support systems.” This myth sometimes goes on to blame denormalization for supporting only specific applications that therefore cannot be changed. This myth is a short-sighted interpretation of dimensional modeling that has managed to get the message exactly backwards! First, we have argued that every ER model has an equivalent set of DM models that contain the same information. Second, we have shown that even in the presence of organizational change and end-user adaptation, the dimensional model extends gracefully without altering its form. It is in fact the ER model that whipsaws the application designers and the end users!

A source of this myth, in my opinion, is the designer who is struggling with fact tables that have been prematurely aggregated. For instance, the design in Figure 2 is expressed at the individual sales-ticket line-item level. This is the correct starting point for this retail database because this is the lowest possible grain of data. There just isn’t any further breakdown of the sales transaction. If the designer had started with a fact table that had been aggregated up to weekly sales totals by store, then there would be all sorts of problems in adding new dimensions, new attributes, and new facts. However, this isn’t a problem with the design technique, this is a problem with the database being prematurely aggregated.

Myth number two is “No one understands dimensional modeling.” This myth is absurd. I have seen hundreds of excellent dimensional designs created by people I have never met or had in my classes. A whole generation of designers from the packaged-goods retail and manufacturing industries has been using and designing dimensional databases for the last 15 years. I personally learned about dimensional models from existing A.C. Nielsen and IRI applications that were installed and working in such places as Procter & Gamble and The Clorox Company as early as 1982.

Incidentally, although this article has been couched in terms of relational databases, nearly all of the arguments in favor of the power of dimensional modeling hold perfectly well for proprietary multidimensional databases such as Oracle Express and Arbor Essbase.

Myth number three is “Dimensional models only work with retail databases.” This myth is rooted in the historical origins of dimensional modeling but not in its current-day reality. Dimensional modeling has been applied to many different business areas including retail banking, commercial banking, property and casualty insurance, health insurance, life insurance, brokerage customer analysis, telephone company operations, newspaper advertising, oi company fuel sales, government agency spending, and manufacturing shipments.

Myth number four is “Snowflaking is an alternative to dimensional modeling.” Snowflaking is the removal of low-cardinality textual attributes from dimension tables and the placement of these attributes in “secondary” dimension tables. For instance, a product category can be treated this way and physically removed from the low-level product dimension table. I believe that this method compromises cross-attribute browsing performance and may interfere with the legibility of the database, but I know that some designers are convinced that this is a good approach. Snowflaking is certainly not at odds with dimensional modeling. I regard snowflaking as an embellishment to the cleanliness of the basic dimensional model. I think that a designer can snowflake with a clea conscience if this technique improves user understandability and improves overall performance. The argument that snowflaking helps the maintainability of the dimension table is specious. Maintenance issues are indeed leveraged by ER-like disciplines, but all of this happens in the operational data store, before the data is loaded intc the dimensional schema.

The final myth is “Dimensional modeling only works for certain kinds of single-subject data marts.” This myth is an attempt to marginalize dimensional modeling by individuals who do not understand its fundamental power and applicability. Dimensional modeling is the appropriate technique for the overall design of a complete enterprise-level data warehouse. Such a dimensional design consists of families of dimensional models, where each family describes a business process. The families are linked together in an effective way by insisting on the use of conformed dimensions.

## In Defense of DM

Now it’s time to take off the gloves. I firmly believe that dimensional modeling is the only viable technique for designing end-user delivery databases. ER modeling defeats end-user delivery and should not be used for this purpose.

ER modeling does not really model a business; rather, it models the micro relationships among data elements. ER modeling does not have “business rules,” it has “data rules.” Few if any global design requirements in the ER modeling methodology speak to the completeness of the overall design. For instance, does your ER CASE tool try to tell you if all of the possible join paths are represented and how many there are? Are you even concerned with such issues in an ER design? What does ER have to say about standard business modeling situations such as slowly changing dimensions?

ER models are wildly variable in structure. Tell me in advance how to optimize the querying of hundreds of interrelated tables in a big ER model. By contrast, even a big suite of dimensional models has an overall deterministic strategy for evaluating every possible query, even those crossing many fact tables. (Hint: You control performance by querying each fact table separately. If you actually believe that you can join many fact tables together in a single query and trust a cost-based optimizer to decide on the execution plan, then you haven't implemented a data warehouse for real end users.)

The wild variability of the structure of ER models means that each data warehouse needs custom, handwritten and tuned SQL. It also means that each schema, once it is tuned, is very vulnerable to changes in the user's querying habits, because such schemas are asymmetrical. By contrast, in a dimensional model all dimensions serve as equal entry points to the fact table. Changes in users' querying habits don't change the structure of the SQL or the standard ways of measuring and controlling performance.

ER models do have their place in the data warehouse. First, the ER model should be used in all legacy OLTP applications based on relational technology. This is the best way to achieve the highest transaction performance and the highest ongoing data integrity. Second, the ER model can be used very successfully in the back-room data cleaning and combining steps of the data warehouse. This is the ODS, or operational data store.

However, before data is packaged into its final queryable format, it must be loaded into a dimensional model. The dimensional model is the only viable technique for achieving both user understandability and high query performance in the face of ever-changing user questions.