Perfect Dimensions

Gregor N. Purdy and Stephen A. Brobst
Intelligent Enterprise, 1999 June 1

Note: This online version is substantially the same as the version in the printed magazine. However, a few small editing errors have been corrected. A PDF file of the original article is available at http://www.tanning.com/pressroom/publications/IE1June.Brobst.pdf. In case something happens to that version, a copy has been cached here.

The right dimensional modeling technique makes all the difference in implementing analytic data structures that best suit your Relational OLAP tool.

Today's advanced business intelligence applications combine using large volumes of detailed data with sophisticated analytic capabilities. A common implementation technique is to store data using a relational database and support applications with a relational online analytical processing (ROLAP) tool providing *ad hoc* and standardized reporting. These ROLAP systems are a common way to jumpstart your company's Web-based information access and regular report scheduling. Most importantly, you're able to specify complex calculations your ROLAP tool will automatically translate into SQL statements necessary for a report. Each tool is built around an underlying reporting model reflected in the tool's database design and report construction requirements and limitations. But all tools focus on business dimensions. We'll discuss the techniques for dimensional modeling and the tradeoffs you might encounter when arriving upon an appropriate physical representation implemented with a relational database.

Dimensional modeling is a technique used to model databases for analytical applications, especially those you can create using ROLAP tools. Dimensional and traditional entity/relationship (E-R) modeling are logical modeling techniques used to capture essential details of the model's subject. However, dimensional modeling rules are more restrictive. The dimensional technique results in a simpler model, which you can use with a ROLAP tool after appropriate physical translation into database tables.

E-R diagrams usually represent conceptual models, and you can use such diagrams to capture complex relationships among entities. E-R models' representative power are stronger than that of dimensional models because of the multiple constructs supported, such as many-to-many, one-to-many, or optional relationships. A dimensional model diagram looks somewhat like an E-R diagram of a normalized conceptual model but has the following additional characteristics:

- Every relationship must be mandatory (one-to-one or one-to-many)
- There must be no more than one possible path between any two entities (this rule prevents cyclic structures)
- You can use groups of optional attributes in place of optional relationships to outboard entities in a conceptual model.

Figure 1 shows three dimensions that appear in the conceptual model for a retail environment. The levels reachable from the Day level comprise the Period dimension; from the UPC level, the Product dimension; and from the Store level, the Geography dimension. The boxes with dotted borders highlight hierarchies within the dimension.

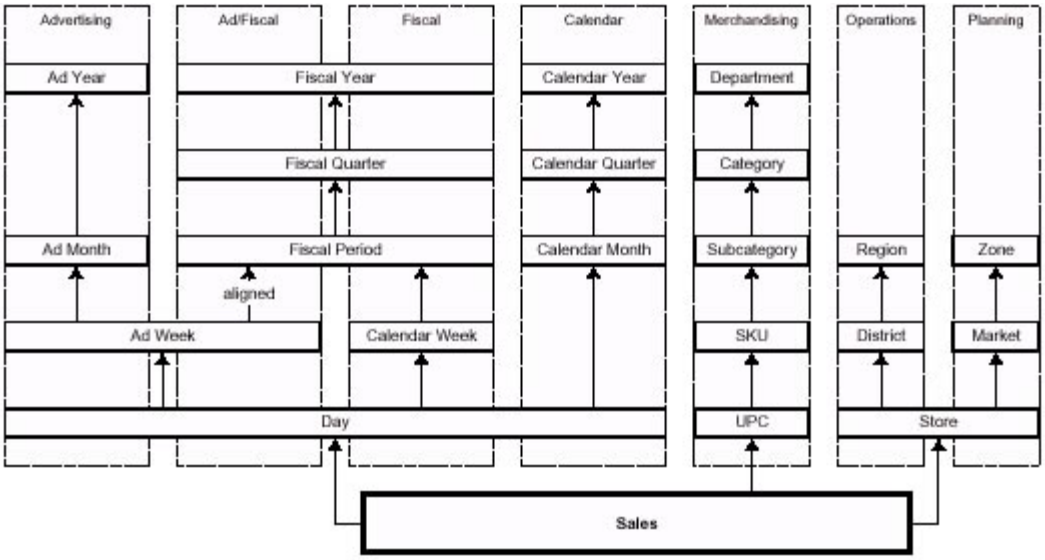

FIGURE 1: A simple dimensional model for a basic retail application.

Dimensions

A dimension is a collection of conceptually related entities. For example, a Period dimension would consist of entities representing temporal intervals, such as day, week, month, and year. A dimension's entities are called its *levels*. The submodel consisting of only a dimension's levels is called its *dimension structure*, which must be connected. That is, every level must be reachable from every other level.

A level that is the source of more than one arc is a *split* in the structure, and a level that's the target of more than one reference is a *merge*. Splits and merges are collectively called *branches*. In Figure 1, Fiscal Period is an example of a merge between Ad Week and Calendar Week. Also in Figure 1, Day demonstrates a split into Ad Week and Calendar Week. A *hierarchy* is a substructure of a dimension structure with no branches. If the entire structure has no branches, then the dimension is strictly hierarchical. Within a hierarchy, there's no ambiguity in the interactive analysis terms *roll up* (tracing arcs) and *drill down* (retracing arcs), because at the most, there's one possible next level up or down. Dimensional diagrams are drawn conventionally with the arrows between levels pointing up wherever possible. This style presents natural physical analogs for the concepts 'roll up' and 'drill down.'

Fact Groups

An entity not in a hierarchy is a *fact group*. Every entity in a dimensional model must be either a level or a fact group. The attributes of a fact group are its facts.

Facts represent stored data, and *metrics* represent reported data. Facts are represented in the dimensional model, and metrics are described in the application's metadata. Metrics corresponding directly to facts are called *simple metrics*. Metrics involving calculations, such as standard arithmetic operations or other, aggregate operations, numeric constants, and filter criteria are called *calculated metrics*.

Because most reporting tools are built on the premise that all facts correspond to simple metrics, it's best to avoid storing calculation results as facts. Doing so splits the application's calculations into two classes: those performed in data management (with definitions captured in one location), and those performed by the application (with definitions captured in a different location). Choosing to store calculated facts means accepting the limitation that, as far as reporting applications are concerned, the resulting fact is disconnected from the other facts on which it is based.

For example, gross profit is defined as sales dollars minus cost. This calculation belongs in the application's metadata so users can look up the definition along with those of other metrics. Storing gross profit in addition to sales dollars and cost might save some report processing time, but at the expense of being unable to capture the calculation's definition in the application metadata (other than as a descriptive annotation).

The dimensions related to a fact group define its *dimensionality*. The particular levels to which fact group refers define its *granularity*. Generally, a fact group is permitted to have at most one reference to any dimension.

A fact is *additive over* a particular dimension if adding it through or over the dimension results in a fact with the same essential meaning as the original fact, but now relative to the new granularity. A fact is *fully additive* if it is additive over every dimension of its dimensionality, *partially additive* if additive over at least one but not all of the dimensions of its dimensionality, and *non-additive* if not additive over any dimension. In general, facts representing individual business transactions are fully additive, although cumulative totals are partially additive. Non-additive facts are usually the result of ratio or other calculations; where possible, you should replace such facts with the underlying calculation facts so you can capture the calculation in the application as a metric.

For example, a fact representing cumulative sales units over a day at a store for a product is a fully additive fact: Adding it over any of its dimensions yields a fact with the same basic meaning except it's for a coarser granularity However, a fact representing ending inventory on hand for a week at a store for a product is a partially additive fact: Adding it over the dimensions containing the store and product levels doesn't fundamentally change its meaning. But adding over the time dimension provides a result no longer representing ending inventory on hand. (The ending inventory on hand for each new sample would have to be the sum of the on-hand values for only those samples corresponding to the latest week in consideration.)

Sometimes, careful consideration can provide alternatives to non-additive facts. For example, a sales fact representing cost is useful. It lets you calculate gross profit. However, per-unit cost is nonadditive, which would present some difficulty in summarizing the report data. If you instead store the cost as an extended cost (per-unit cost times units sold), it will be fully additive. Furthermore, you can use a calculation (sum of extended cost divided by sum of sales units) to obtain average unit cost for any coarser granularity.

A fact group's *potential cardinality* is the product of the expected cardinalities of its granularity levels. The *expected cardinality* of a fact group is an estimate of the number of samples that will actually occur. A fact group's *expected density* is the expected cardinality divided by the potential cardinality; you can think of it as a percentage-filled estimate. Finally, the *expected sparsity* of a fact group is a percentage equal to 100 percent minus the expected density. You can't determine the three analogous physical concepts — actual cardinality, actual density, and actual sparsity — without first building and populating the corresponding table.

Some facts, such as the sales facts in our example, represent actual events. You can trace an *actual event* back to at least one business transaction (such as sales, orders, or inventory). The tie to a business transaction may be direct (as with sales and orders, where the fact represents a single aspect of a single transaction), or indirect (as with the case of the aforementioned inventory where the fact represents the net effect of some transactions on the state of the business). Other facts represent *potential events* — those representing the ability of the business to perform a transaction. (For example, the set of products a particular store sells during a particular day is typically much smaller than the set of products it carries on that day.)

Facts for actual events tend to be sparser than facts for potential events. A given fact can appear in more than one fact group as long as there's a single finest granularity with which all the others are compatible. The fact appearing in the finest granularity fact group is called the *base fact*. The others are called *aggregate facts* and

must have rules defined for how they'll be constructed from the base fact. (Usually, summation is the method.)

There are three primary reasons for creating aggregate facts when translating the dimensional model to a physical representation:

- Enhancing performance on common queries at coarser granularities
- Storing more historical data than is practical at the finer granularities
- Taking advantage of the need to store other facts with similar sparsity and other characteristics at the coarser granularity.

When contemplating the introduction of aggregates, keep in mind that although aggregating to coarser granularities decreases potential cardinality, it tends to also increase density (and therefore decrease sparsity). This set of relationships can cause an aggregate fact group's expected cardinality to be comparable to the base fact group's expected cardinality, which means they'd be equally costly in terms of storage. If a candidate aggregate fact entity's expected cardinality is more than 10 percent of that of the next finer granularity with the same facts, strongly consider omitting the aggregate from the design. Also, note that aggregate management introduces operational complexities and data dependencies that you shouldn't underestimate.

Physical Dimension Representations

There are many options for translating a dimensional model into a physical representation appropriate for implementation in a relational database product. When deciding which of the possible representations you should select, consider the specifics of the front-end tool youi're deploying, as well as the underlying relational database management system (RDBMS). We'll focus on the four major alternatives for representing a dimensional model: *flattened*, *normalized*, *expanded*, and *levelized*. Note that specific front-end tools will often impose additional nuances in the physical representation of a dimensional model (especially in the Period dimension), and you should investigate these requirements prior to finalizing a physical design.

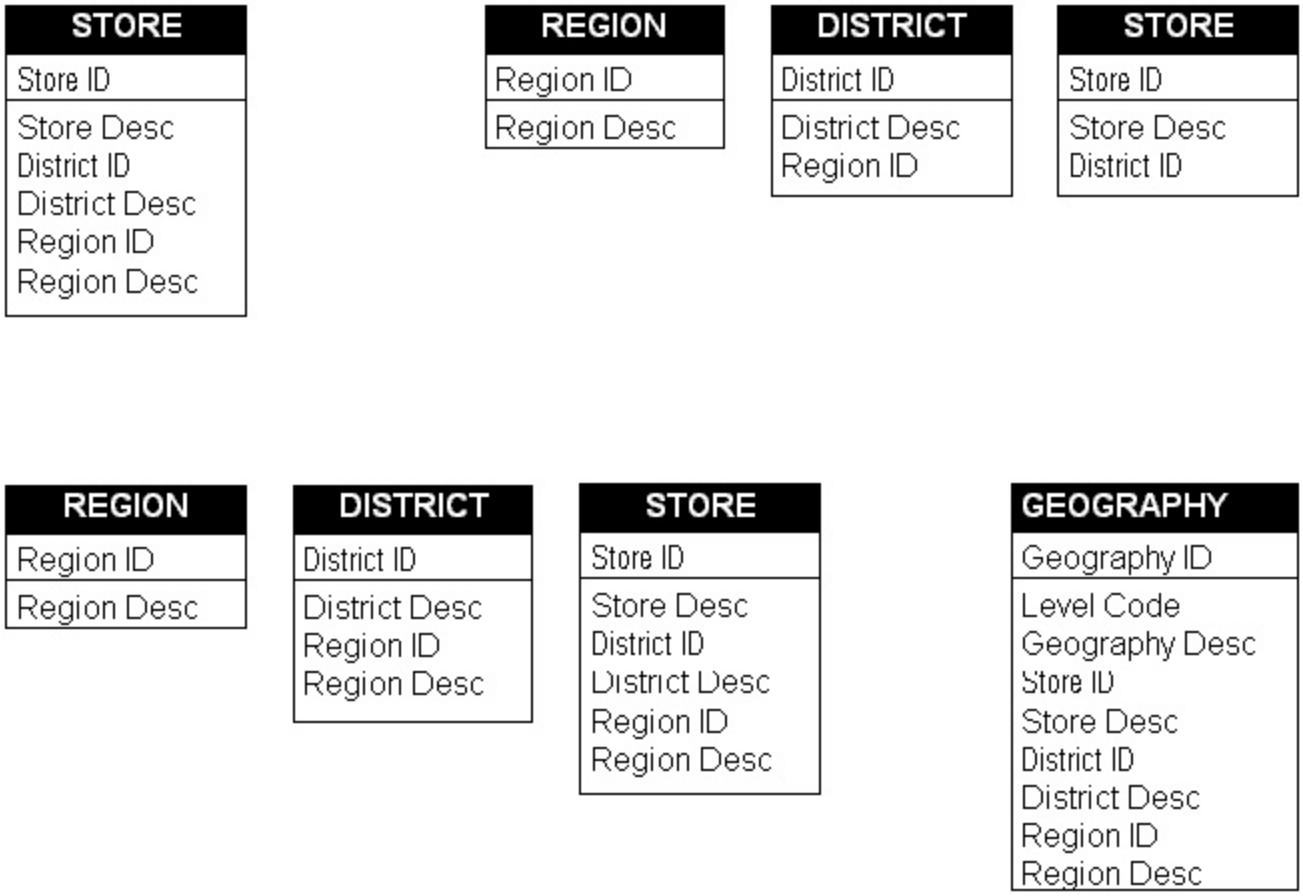

FIGURE 2 The four alternatives for physical representation of a dimensional model: (a) flattened, (b) normalized, (c) expanded, and (d) levelized.

Flattened. A flattened dimension consists of just one table. The table has one column for each attribute of each dimension level and contains one row for each of the rows in the lowest (most detailed) level in the dimensional model. Attributes from levels above the lowest are flattened into the bottom level as separate columns. Figure 2a shows the flattened representation for the example model, where Store is the lowest level.

If the dimensional model has fact groups involving levels other than the lowest dimension levels, you'll have to create additional flattened dimension tables with these other levels; their lowest levels will serve as targets for the foreign keys of these other fact tables.

Normalized. Normalized dimensions consist of one table for each level of the dimension. Each table has one column for each corresponding level attribute (including a primary key), and one foreign key column for each of its direct parent levels. Each table contains one row for each instance of the corresponding level.

For example, each of the tables in Figure 2b has ID and Description columns, but only the Store and District tables have additional foreign-key columns to represent the relationships to the next levels to which they roll up in the dimensional model.

Normalized dimensions use space efficiently, because the minimum possible set of attributes is defined for each entity. However, some queries against this sort of model would require more joins than would other representations. For example, selecting data from a store-level fact table and grouping it by an attribute of regions requires all three dimension tables referred to in Figure 1 — Store, District, and Region — simultaneously. This operation would necessitate a multitable join and would possibly affect overall query

efficiency.

Expanded. Similar to normalized dimensions, expanded dimensions consist of one table per dimension. However, each of the tables has columns for the corresponding level attributes as well as for all the levels above it in the dimension structure. As shown in Figure 2c, an expanded dimension has the maximum number of tables, with the maximum number of columns per table. The tables each have one row for each instance of the corresponding level. Expanded dimensions are even less space efficient than flattened dimensions because of the larger number of tables they spawn. However, the tables let you perform efficient queries against all possible fact tables involving the dimension, because a table with the right key and the right columns for any filter will already be available for a join; there's no need to resort to multi-table joins within a single dimension.

Levelized. Levelized dimensions consist of just one table that has all the columns of a flattened dimension and a few additional special columns to identify the source level. The table has one row for each instance of each level. In other words, the number of columns in a 1evelized design will be roughly equivalent to what's required in a flattened design, yet the number of rows will be equal to the sum of the number of rows from all levels in a normalized design.

In Figure 2d, the table would contain one row for each Store, District, and Region. For each row, the value of the Level Code column indicates the corresponding level, which would be Store, District, or Region. An artificial key, such as Geography ID, serves to keep all the rows distinct, because it's possible the domains of the natural keys for the levels might overlap (such as when there's both a Store 1 and a District 1). For each row, the columns that don't apply are NULL. For example, a District row would have NULLs for the store information.

Among the four physical dimension representations, flattened is the one most often discussed in reference to dimensional modeling. However, dimensional modeling is a logical technique, and the dimension representation is a physical decision most often determined by the choice of reporting tool. The other three representations are equally valid, and there are situations in which they are required. Dimensional modeling *per se* doesn't dictate dimension flattening.

Knowing Design Parameters

The dimensional modeling techniques we've described facilitate creating data models to support ROLAP and other analytical access against a data warehouse. High-level, tool-based access to data warehouses usually requires relatively simple data models because of the structural assumptions reporting tool developers make. These assumptions determine the database schema patterns the tool will support, which are usually closely related to the patterns appearing in a dimensional model.

By using a modeling method that accounts for the restrictions the reporting tool places on the physical model, it's easier to arrive at a final model you can use through the tool. Of course, as with any effective modeling technique, the diagrams are a small part of the overall deliverable. In addition, it's vital to produce documentation that also includes extensive conceptual information: precise definitions of the business terms used in the model, detailed explanations of design decisions, the alternatives considered, and the factors and reasoning behind the alternatives chosen. All this is necessary to let project contributors do qualified reviews and support ongoing discussions with business users. Fortunately, because the dimensional diagrams tend to be relatively simple, they serve as an excellent frame of reference for both these activities.

Space considerations for the dimensional tables in a physical representation are typically inconsequential compared to their corresponding fact tables (which are generally orders of magnitude larger in size than the dimensional tables). The real trade-offs in choosing physical representation usually relate to the particular

front-end reporting tool selected and the ability of your database to handle the implemented table structures efficiently. Before making database and front-end reporting tool selections, you have to understand the design parameters your choice will influence. Only then can you implement a physical representation with a relational database optimized for your tool of choice. Dimensional modeling techniques are independent of the physical representation you ultimately select, and they provide a robust design vehicle for describing the analytic structure you want to implement in your ROLAP system.

Gregor Purdy is a senior consultant at Strategic Technologies & Systems in Boston. He specializes in dimensional modeling techniques and ROLAP application design. You can reach him at *gregor@strattech.com*.

Stephen Brobst is a founder and managing partner at Strategic Technologies & Systems. He is also a cofounder and senior consultant at Tanning Technology Corp. in Denver. You can reach him at *sabrobst@strattech.com*.

2001-07-23 @ 10:56:23 (EDT)