

Kimball Design Tip #29: Graceful Modifications To Existing Fact and Dimension Tables

By Ralph Kimball

Despite the best plans and the best intentions, the data warehouse designer must often face the problem of adding new data types or altering the relationships among data after the data warehouse is up and running. In an ideal world we would like such changes to be "graceful" so that existing query and reporting applications continue to run without being recoded, and existing user interfaces "wake up" to the new data and allow the data to be added to queries and reports.

Obviously, there are some changes that can never be handled gracefully. If a data source ceases to be available and there is no compatible substitute, then the applications depending in this source will stop working.

But can we describe a class of situations where changes to our data environment can be handled gracefully?

The predictable symmetry of our dimensional models comes to our rescue. Dimensional models are able to absorb some significant changes in the source data and in our modeling assumptions without invalidating existing applications. Let's list as many of these changes as we can, starting with the simplest.

1. NEW DIMENSIONAL ATTRIBUTES. If, for example, we discover new textual descriptors of a product or a customer, we add these attributes to the dimension as new fields. All existing applications will be oblivious to the new attributes and will continue to function. Most user interfaces should notice the new attributes at query time. Conceptually, the list of attributes available for constraining and grouping should be displayed in a query tool or a reporting tool via an underlying query of the form `SELECT COLUMN_NAME FROM SYS_TABLES WHERE TABLE_NAME = 'PRODUCT'`. This kind of user interface will continuously "adjust" when new dimension attributes are added to the schema. In a slowly changing dimension (SCD) environment, where slightly changed versions of the dimension are being maintained, care must be taken to assign the values of the new attributes correctly to the various versions of the dimension records. If the new attributes are available only after a specific point in time, then "N.A." (not available) or its equivalent must be supplied for old dimension records.

2. NEW TYPES OF MEASURED FACTS. Similarly, if new measured facts become available we can add them to the fact table gracefully. The simplest case is when the new facts are available in the same measurement event and at the same grain as the existing facts. In this case, the fact table is altered to add the new fact fields, and the values are populated into the table. In an ideal world, an `ALTER TABLE` statement can be issued against the existing fact table to add the new fields. If that is not possible, then a second fact table must be defined with the new fields and the records copied from the first. For truly huge fact tables, large groups of records may have to be moved from one table to the other to keep from storing the giant table in its entirety twice. If the new facts are only available from a point in time forward, then true null values need to be placed in the older fact records. If we have done all of this, old applications will continue to run undisturbed. New applications using the new facts should behave reasonably even if the null values are encountered. The users may have to be trained that the new facts are only available from a specific point in time

forward.

A more complex situation arises when new measured facts are not available in the same measurement event as the old facts, or if the new facts occur naturally at a different grain. If the new facts cannot be allocated or assigned to the original grain of the fact table, it is very likely that the new facts belong in their own fact table. It is almost always a mistake to mix grains of measurements or mix disjoint kinds of measurements in the same fact table. If you have this situation, you need to bite the bullet and find a query tool or report writer that is capable of multi-pass SQL so that it can access multiple fact tables in the same user request. Tools like Cognos, Business Objects, and Microstrategy are quite capable of handling multi-pass SQL.

3. NEW DIMENSIONS. A dimension can be added to an existing fact table by adding a new foreign key field and populating it correctly with values of the primary key from the new dimension. For example, a weather dimension can be added to a retail sales fact table if a source describing the weather is available, for instance, at each selling location each day. Note that we are not changing the grain of the fact table. If the weather information is only available from a point in time forward, then the foreign key value for the weather dimension must point to a record in the weather dimension whose description is "weather unavailable".

4. A DIMENSION BECOMING MORE GRANULAR. Sometimes it is desirable to increase the granularity of a dimension. For instance, a retail sales fact table with a store dimension could be modified to replace the store dimension with an individual cash register dimension. If we had 100 stores, each with an average of 10 cash registers, the new cash register dimension would have 1000 records. All of the original store attributes would be included in the cash register dimension because cash registers roll up perfectly in a many to 1 relationship to stores. The store attributes could also be modeled physically as a snowflaked outrigger dimension connected to the cash register dimension. Notice that when we increase the granularity of the store==>cash register dimension, we must increase the granularity of the fact table. The fact table will become 10 times as large, in our example. There is probably no alternative but to drop the fact table and rebuild it. Although this change is a big complex change, it is graceful! All the original applications are unaffected. The store totals all look the same and all queries will return the same results. It may run more slowly because there are ten times as many records in the fact table, but we would probably build a store aggregate anyway! This would be the original fact table, now playing the role of an anonymous aggregate table.

5. ADDITION OF AN EXPLICIT HIERARCHY RELATING TWO DIMENSIONS. We may have a situation where two dimensions turn out to have a hierarchical many to 1 relationship but for various reasons we keep the dimensions separate. Normally we would combine hierarchically related entities into the same dimension but if one or both of the dimensions exists in its own right as an independent "conformed" dimension, we may wish to keep them separate. For instance, in insurance we may have a policy dimension and a customer dimension. If every policy has exactly one customer, then policy rolls up to customer. However we probably would not embed the customer information in the policy dimension because the customer dimension will certainly be a major dimension in its own right in our overall data warehouse bus architecture. The customer dimension will need to connect to many other fact tables, in many cases where there is no policy dimension possible. In spite of this need to keep the dimensions separate, some designers will add a very useful CustomerPolicy key to some of their fact tables where the new CustomerPolicy dimension is exactly the marriage of Customer and Policy. This dimension contains the combination of Customers and Policies and can be reliably queried at all times to explore this relationship, independent of the residency of any fact table. The addition of this new combined dimension key poses the same administrative issues as the addition of a new dimension, described in paragraph #3 above. Since it is simply a new dimension, it passes the gracefulness test.

6. ADDITION OF A COMPLETELY NEW SOURCE OF DATA INVOLVING EXISTING DIMENSIONS AS WELL AS UNEXPECTED NEW DIMENSIONS. Almost always, a new source of data has its own granularity and its own dimensions. All of you dimensional designers know the answer to this one. We sprout a brand new fact table. Since any existing fact tables and dimension

tables are untouched, by definition, all the existing applications keep chugging along. Although this case seems almost trivial, the point here is to avoid cramming the new measurements into the existing fact tables. A single fact table always owns a single kind of measurement expressed with a uniform grain.

This design tip has tried to define a taxonomy of unexpected changes to your data environment to give you a way to sort through various responses, and to recognize those situations where a graceful change is possible. Since redoing queries and reports is hugely expensive and probably disruptive to the end users, our goal is to stay on the graceful side of the line.