The Soul of the Data Warehouse, Part 3: Handling Time

By Ralph
Kimball

The three most fundamental maneuvers in every data warehouse are drilling down, drilling across, and handling time. I discussed the first two maneuvers in the previous two installments of this Fundamentals series. (See Resources.) The third, handling time, makes good on a pledge that every data warehouse provider implicitly takes: *The data warehouse shall preserve history.* In practice, this pledge generates three main requirements for the data warehouse:

First, every piece of data in the data warehouse must have a clearly understood time validity. In other words, when dic the data become valid, and when did it cease to be valid?

Second, if the detailed description of a data warehouse entity has changed over time, you must correctly associate each version of that entity with the contemporary versions of other measurements and entities in the data warehouse. In other words, if a customer made a purchase a year ago, the description of the customer attached to that purchase must be correct for that time frame.

Last, the data warehouse must support the natural ways people have of viewing data over time. These natural ways include seeing instantaneous events, regular periodic reports, and latest status.

Dimensional modeling provides a convenient framework for dealing with each of these requirements. Remember that dimensional models are organized around measurements. Measurements, which are usually numeric, occupy fact tables in a dimensional model. The contexts of the measurements are in dimension tables, which surround the fact tables and connect to them through a series of simple relationships between foreign keys and primary keys.

Time Validity

A measurement is usually a physical act that takes place at a specific time, so it’s natural, even irresistible, to attach a time stamp to each fact table record. Every fact table has a time dimension.

Time stamps are commonly recorded at a daily grain because many legacy systems don’t record time of day when posting a measurement. In a dimensional schema, the daily time stamp consists of a surrogate (vanilla integer) foreign key in the fact table joined to a corresponding primary key in the daily time dimension table. You want the time stamp in the fact table to be a surrogate key rather than a real date for three reasons: First, the rare time stamp that is inapplicable, corrupted, or hasn’t happened yet needs a value that cannot be a real date. Second, most end-user calendar navigation constraints, such as fiscal periods, end-of-periods, holidays, day numbers, and week numbers aren’t supported by database time stamps. Therefore, they need to come from a table with a verbose time dimension, rather than computed in the requesting query tool. Third, integer time keys take up much less disk space than full dates.

When the source system provides a detailed time stamp for the measurement down to the minute or the second, the time of day needs to be a separate dimension. Otherwise, a combined day and time-of-day dimension would be impractically large.

In multinational applications, there are often two time-stamp perspectives: the remote party's and the home office's. Certainly, when recording the time of day, it usually makes sense to include two pairs of time stamps (day and time-of-day as well as remote and local) rather than leaving it up to the query tool to work out time zones. Mainly because of daylight savings time rules, the calculation of time zone differences is horrendously complicated. (For more on this topic, see my book, *Data Webhouse Toolkit*.)

Correct Association

Dimensional modeling assumes that dimensions are largely independent. This assumption, combined with the fact that time is its own dimension, implies that other entities, such as customer and product, are independent of time. In the real world, this inference isn't quite true. Entities such as customer and product slowly change over time, usually in episodic, unpredictable ways.

When the data warehouse encounters a legitimate revised description of, for example, a customer, there are three fundamental choices for handling this slowly changing dimension:

1. Overwrite the changed attribute, thereby destroying previous history. This approach is justifiable only when correcting an error, if you're living by the pledge.
2. Issue a new record for the customer, keeping the customer natural key, but creating (by necessity) a new surrogate primary key.
3. Create an additional field in the existing customer record, and store the old value of the attribute in the additional field. Overwrite the original attribute field. This strategy is called for when the attribute can have simultaneous "alternate realities."

I've written voluminously about these three types of slowly changing dimensions (SCDs), how they're modeled, and how they're administered. Please see any of my books for the details. But the point I wish to make here is that all three choices need at least one embedded time stamp stating when the record was updated as well as a companion field describing that change. For the primary Type 2 SCD, where a new record is created, you need a pair of time stamps as well as a change description field. The pair of time stamps define a span of time from the begin-effective time to the end-effective time when the complete customer description remains valid.

The most sophisticated treatment of a Type 2 SCD record involves five fields:

- Begin effective date/time stamp (not a surrogate key pointer)
- End effective date/time stamp
- Effective date surrogate key (daily grain) connecting to date dimension as a snowflake
- Change description field (text)
- Most recent flag.

The first two fields are what conventional `BETWEEN` constraints use to profile the dimension at specific points in time. They need to be single fields with full date and time stamps in order to make the `BETWEEN` machinery work. The third field constrains specific records in the dimension that changed on days that can be described only via the organization's calendar date table (such as employee demotions that occurred the day before payday). The fourth field lets you find all changes in the dimension meeting a particular description.The fifth field is a quick way to find all the current records in a dimension without using `BETWEEN`.

Natural Views

In 30 years of analyzing and modeling data, I've found that fact-table measurements all fall into just three classes. These types correspond to instantaneous events, regular periodic reports, and latest status. In dimensional modeling, these three fact-table types are the transaction grain, the periodic-snapshot grain, and the accumulating-snapshot grain.

The transaction grain represents a point in space and time, and is meant for a measurement event defined at a particular instant. A scanner event at a grocery store is the classic example of a transaction event. In this case, the time stamp in the fact table is very simple. It's either a single daily-time-grain foreign key or a pair consisting of a daily-time-grain foreign key together with a time-of-day foreign key, depending on what the source system provides. The facts in this transaction-grain table must be true to the grain and should describe only what took place in that instant.

The periodic-snapshot grain represents a regular repeating measurement, like a bank account monthly statement. This fact table also has a single time stamp, representing the overall period. Usually the time stamp is the end of the period, and often is expressed at the daily grain, even if it's understood to represent a month or a fiscal period. The facts in this periodic-snapshot grain table must be true to the grain and should describe only measures appropriate to the specific period.

The accumulating-snapshot grain represents the current evolving status of a process that has a finite beginning and end. Usually these processes are of short duration and therefore don't lend themselves to the periodic snapshot. Order processing is the classic example of an accumulating snapshot.

The design and administration of the accumulating snapshot is quite different from the first two fact-table types. All accumulating-snapshot fact tables have a set of as many as four to 12 dates describing the typical scenario of the process being modeled. For instance, an order has a set of characteristic dates: original order date, actual ship date, delivery date, final payment date, and return date. In this example, these five dates appear as five, separate, date-valued foreign (surrogate) keys. When the order record is first created, the first of these dates is well defined, but perhaps none of the others have yet happened. This same fact record is subsequently revisited as the order wends its way through the pipeline. Each time something happens, the accumulating-snapshot fact record is destructively modified. The date foreign keys are overwritten, and various facts are updated. Often the first date remains inviolate because that describes when the record was created, but all the other dates may well be overwritten, sometimes more than once.

Have You Lived Up to Your Pledges?

This column has been a brief overview of the central techniques of handling time in a data warehouse. If you've systematically employed surrogate keys for all your connections to your master time dimensions, faithfully tracked changes in dimension entities with the three types of SCDs, and supported your users' reporting needs with transaction-grain, periodic-snapshot-grain, and accumulating-snapshot-grain fact tables, then you have indeed lived up to your pledge.