

## Design Tip #87 Combining SCD Techniques: Having It Both Ways

By Joy Mundy

I've spoken with many warehouse designers who are torn between conflicting business requirements when structuring their dimensions. Some business users have a compelling requirement for a dimension attribute to track history as a Type 2 attribute, while other users are equally clear they want that same attribute to restate history as Type 1. What to do?

One approach is to construct queries so that we join the dimension table to itself, returning the dimensional attributes for the current row together with the measured numeric facts for all history, such as:

```
SELECT CurrCust.CustomerName AS CurrentCustomerName,
CurrCust.City AS CurrentCustomerCity,
SUM(SalesFacts.SalesAmount) AS TotalSales
FROM Customer CurrCust
INNER JOIN Customer CustType2 ON
    (CurrCust.CustBusinessKey = CustType2.CustBusinessKey)
    AND CurrCust.IsCurrentInd = 'Y'
    INNER JOIN SalesFacts ON
        SalesFacts.CustomerKey = CustType2.CustomerKey
GROUP BY CurrCust.CustomerName, CurrCust.City
```

This query returns an answer set with three columns: current customer name, current customer city and sales for that customer across all time. This SQL has been tested and is correct! This is the easiest solution to implement on the design and ETL side because it requires no additional effort. But our design goal is seldom centered on what's easiest for the ETL system to implement, and this approach has two significant downsides. First and most important, it's not something we'd expect most business users to be able to do. Second, that self-join on the dimension table will slow down query performance.

What we're really doing in this self-join approach is adding the current version of each Type 2 attribute as an additional column in the dimension table. We're doing this virtually at query time, but a nicer approach is simply to do it once at ETL time. In other words, if you have a Type 2 attribute that some people want to see as Type 1, then put it in your dimension table twice. Although this may seem to violate the dictum that we have a single version of the truth, that problem is neatly solved by renaming the Type 1 attribute as *current*, such as Current Customer City. It's easy to educate the business users that the current attributes are the versions of the attribute as they are today; the other attribute with the similar name tracks history. This is a very simple solution.

Most warehouse designers are comfortable with this approach as long as there are only a handful of Type 2 attributes in the dimension. But what if your dimension table is very wide, for example a customer dimension with a hundred attributes, and many or most of those attributes are tracked as Type 2? Doubling the width of the dimension table with a hundred additional *current* versions of each attribute is unappealing. It's particularly awkward if your user access technology doesn't have a way to group attributes into display folders when the user is browsing the data structures.

In the extreme situation where your dimension table is long and wide – you have millions of

customers and hundreds of attributes about those customers – you may choose to create two dimensions: Customer and Current Customer. Customer would be the normal dimension with many Type 2 attributes, and would contain multiple rows for each customer as their attributes change over time. Current Customer would contain only the current view of each customer's attributes, and would have one row per customer. Current Customer would be a pure Type 1 dimension. Fact tables would need to have a key for both Customer and Current Customer. This technique is further described in our [“Slowly Changing Dimensions are not always as Easy as 1, 2, 3”](#) article published in Intelligent Enterprise.

These two approaches roll neatly into an OLAP solution. The simple approach of adding the current attributes to the dimension table would translate to additional attributes of the Customer dimension in the OLAP database. If your only access into the data is through the OLAP layer, you can actually get away with the first technique we mentioned: use a query or define a view that performs the self-join on the dimension table. Most OLAP tools would only execute this query at the time the dimension is processed, and it would probably perform adequately. If instead you decide to create and manage a separate Current Customer dimension table, it would show up in your OLAP tool as a separate dimension. It should work as well in OLAP as it does in the relational database.