

## Kimball Design Tip #22: Variable Depth Customer Dimensions

By Ralph Kimball

The customer dimension is probably the most challenging dimension in a data warehouse. In a large organization, the customer dimension can be

- 1) huge, with millions of records
- 2) wide, with dozens of attributes
- 3) slowly changing, but sometimes quickly changing

To make matters worse, in the biggest customer dimensions we often have two categories of customers which I will call Visitor and Customer.

The Visitor is anonymous. We may see them more than once, but we don't know their name or anything about them. On a web site, all we have for a Visitor is a cookie that tells us they have returned. In a retail operation, a Visitor engages in an anonymous transaction. Perhaps we have a credit card number or a simple shopper identity card, but we assume here that no meaningful demographic data accompanies a Visitor.

The Customer, on the other hand, is reliably registered with us. We know the Customer's name, address, and as much demographics as we care to elicit directly from them or purchase from third parties. We have a shipping address, a payment history, and perhaps a credit history with each Customer.

Let's assume that at the most granular level of our data collection, 80% of the fact table measurements involve Visitors, and 20% involve Customers. Let us further assume that we accumulate simple behavior scores for Visitors consisting only of Recency (when was the last time we saw them), Frequency (how many times have we seen them), and Intensity (how much business have we done with them). So in this simple design we only have three attributes/measures for a Visitor.

On the other hand let's assume we have 50 attributes/measures for a Customer, covering all the components of location, payment behavior, credit behavior, directly elicited demographic attributes, and third party purchased demographic attributes.

Let's make some rather specific and limiting assumptions for the sake of a clean design. We can relax some of these assumptions later...

First, let's combine Visitors and Customers into a single logical dimension called Shopper. We will give the true physical Visitor/Customer a single permanent Shopper ID, but we will make the key to the table a surrogate key so that we can track changes to the Shopper over time. Logically, our dimension looks like

**attributes for both Visitors and Customers:**

Shopper Surrogate Key	<==	simple integer assigned sequentially with each change
Shopper ID	<==	permanent fixed ID for each physical shopper

Recency Date	<== date of last visit, Type 1: overwritten
Frequency	<== number of visits, Type 1: overwritten
Intensity	<== total amount of business, e.g., sales dollars, Type 1

**attributes for Customers only:**

5 name attributes	<== first, middle, last, gender, greeting
10 location attributes	<== address components
5 payment behavior attributes	
5 credit behavior attributes	
10 direct demographic attributes	
15 purchased demographic attributes	

One strong assumption we have made here is to include the Recency, Frequency, and Intensity information as dimensional attributes rather than as facts, and also to continuously overwrite them as time progresses (Type 1 slowly changing dimension). This assumption makes our Shopper dimension very powerful. We can do classic shopper segmentation directly off the dimension without navigating a fact table in a complex application. See the discussion of Recency-Frequency-Intensity segmentation in my Webhouse Toolkit book, starting on page 73.

If we assume that many of the final 50 Customer attributes are textual, we could have a total record width of 500 bytes, or more.

Suppose we have 20 million Shoppers (16 million Visitors and 4 million registered Customers). Obviously we are worried that in 80% of our records, the trailing 50 fields have no data! In a 10 gigabyte dimension, this gets our attention.

This is a clear case where, depending on the database, we may wish to introduce a snowflake.

In databases with variable width records, like Oracle, we can simply build a single shopper dimension with all the above fields, disregarding the empty fields issue. The majority of the shopper records, which are simple Visitors, remain narrow, because in these databases, the null fields take up zero disk space.

But in fixed width databases, we probably don't want to live with the empty fields for all the Visitors, and so we break the dimension into a base dimension and a snowflaked subdimension:

**Base:**

Shopper Surrogate Key	<== simple integer assigned sequentially with each change
Shopper ID	<== permanent fixed ID for each physical shopper
Recency Date	<== date of last visit, Type 1: overwritten
Frequency	<== number of visits, Type 1: overwritten
Intensity	
Customer Surrogate Key	<== new field to link to the snowflake

**Snowflake:**

Customer Surrogate Key	<== 1:1 matching field for those shoppers who are Customers
5 name attributes	
10 location attributes	
5 payment behavior attributes	
5 credit behavior attributes	
10 direct demographic attributes	
15 purchased demographic attributes	

In a fixed width database, using our previous assumptions, the base Shopper dimension is 20 million X 25 bytes = 500 MB, and the snowflake dimension is 4 million X 475 bytes = 1.9 gigabytes. We have saved 8 gigabytes by using the snowflake.

If you have a query tool that insists on a classic star schema with no snowflakes, then hide the snowflake under a view declaration.

This is the basic foundation for a variable depth customer dimension. I have left lots of issues on the table, including

- how to administer the changing attributes in the Customer portion of the dimension
- how to not lose the history of the recency-frequency-intensity measures given we are overwriting them.
- how to add hierarchical relationships to all of this if the customers are organizations
- how to deal with security and privacy design constraints that may be imposed on top of all this

Stay tuned for the next Design Tip...

Write to me with questions or comments about variable depth customer dimensions.