

Design Tip #112 Creating Historical Dimension Rows

By Warren Thornthwaite

The business requirement for tracking dimension attribute changes over time is almost universal for DW/BI systems. Tracking attribute changes supports accurate causal analysis by associating the attribute values that were in effect when an event occurred with the event itself. You will need a set of ETL tasks to assign surrogate keys and manage the change tracking control fields for this Type 2 tracking of historical changes.

One big challenge many ETL developers face is recreating historical dimension change events during the initial historical data load. This is often a problem because the operational system may not retain history. In the worst case, attribute values are simply overwritten with no history at all. You need to carefully search the source systems for evidence of historical values. If you can't find all the historical data, or the effort to recreate it is significant, you will need to discuss the implications of incomplete history with business folks. Once the decisions are made, you will need to load what you have and properly set the effective date and end date control columns. We'll talk more about each of these three steps below.

Dig for History

First look at the source tables that directly feed your dimension along with any associated tables. You may find some change tracking system for audit or compliance purposes that isn't obvious from the direct transaction source tables. We worked on one system that wrote the old customer row into a separate history table anytime a value changed. Once we found that table, it was fairly straightforward to recreate the historical dimension rows.

You may need to broaden your search for evidence of change. Any dates in the dimension's source tables might be helpful. Your customer dimension source tables may have fields like registration date, cancel date, or contact date. Transaction events might also be helpful. Sometimes the shipment or customer service systems keep a copy of the customer address in their records each time a fact event occurs.

In the worst case, you may need to pull data from the backup system. Pulling daily backups for a set of tables for the last five years is usually not feasible. Explore the possibility of pulling one backup per month, so you will at least be able to identify changes within a 30 day window of when they actually occurred.

Discuss the Options and Implications

The business folks often don't comprehend the implications of incomplete dimension history without careful explanation and detailed examples. You need to help them understand so they can help make informed decisions. In the worst case, you may not have any change history. Simply associating current dimension values with historical fact events is usually not acceptable from the business perspective. In a case like this, it's not uncommon to only load data from the point where dimension history is available.

Build the Dimension

You may have to pull several different change events for a given dimension together from several sources. Combine all these rows into a single table and order them by the customer transaction key and the dimension change date which becomes the effective date of the row. The expiration date of

each row will depend on the effective date of the next row. You can accomplish the expiration date assignment with a looping function in your language of choice. It's also possible to do it in your database with a cursor or other control structures. Don't worry too much about performance because this is a one-time effort.

Choose a Day-Grain or Minute-Second-Grain

For years we have recommended augmenting a Type 2 dimension with begin and end effective time stamps to allow for precise time slicing of history as described in this design tip. But there is a fussy detail that must be considered when creating these time stamps. You must decide if the dimension changes must be tracked on a daily grain (i.e., no more than one change per day) or on a minute-second grain where many changes could occur on a given day. In the first case, you can set the end expiration time stamp to be one day less than the time stamp of the next dimension change, and your BI queries can constrain a candidate date BETWEEN the begin and end time stamps. But if your dimension changes are to be tracked on a minute-second basis, you don't dare set the end time stamp to be "one tick" less than the next dimension change's time stamp because the "tick" is machine and DBMS dependent, and you could end up with gaps in your time sequence. In this case, you must set the end time stamp to be exactly equal to the time stamp of the next change, and forgo the use of the BETWEEN syntax in your BI queries in favor of GREATER-THAN and LESS-THAN-OR-EQUAL syntax.

Here is a set-based example to assign the expiration data for a day grain customer dimension table that uses the ROW_NUMBER function in Oracle to group all the rows for a given customer in the right order:

```
UPDATE Customer_Master T
    SET T.Exp_Date =
        (SELECT NVL(TabB.Real_Exp_Date, '31-DEC-9999')
 FROM
    (SELECT ROW_NUMBER() OVER(Partition by Source_Cust_ID
    Order by Eff_Date) AS RowNumA, Customer_Key, Source_Cust_ID,
    Eff_Date, Exp_Date
    FROM Customer_Master ) TabA -- target row
LEFT OUTER JOIN
    (SELECT ROW_NUMBER() OVER(Partition by Source_Cust_ID
    Order by Eff_Date) AS RowNumB, Source_Cust_ID,
    Eff_Date -1 AS
    Real_Exp_Date, Exp_Date -- assumes day grain
    FROM Customer_Master) TabB -- next row after the target row
ON TabA.RowNumA = TabB.RowNumB - 1
AND TabA.Source_Cust_ID = TabB.Source_Cust_ID
WHERE T.Customer_Key = TabA.Customer_Key);
```

Here is a set-based example of how to assign the expiration data for a minute-second grain customer dimension table that uses the ROW_NUMBER function in Oracle to group all the rows for a given customer in the right order. Note that date field references in the first example have been replaced with date-time field references below:

```
UPDATE Customer_Master T
    SET T.Exp_Date_Time =
        (SELECT NVL(TabB.Real_Exp_Date_Time, '31-DEC-9999 0:0:0')
 FROM
    (SELECT ROW_NUMBER() OVER(Partition by Source_Cust_ID
    Order by Eff_Date_Time) AS RowNumA, Customer_Key, Source_Cust_ID,
    Eff_Date_Time, Exp_Date_Time
    FROM Customer_Master ) TabA -- The target row
LEFT OUTER JOIN
```

```
(SELECT ROW_NUMBER() OVER(Partition by Source_Cust_ID
Order by Eff_Date_Time) AS RowNumB, Source_Cust_ID,
Eff_Date_Time AS
Real_Exp_Date_Time, Exp_Date_Time -- assumes minute-second grain
FROM Customer_Master) TabB -- next row after the target row
ON TabA.RowNumA = TabB.RowNumB - 1
AND TabA.Source_Cust_ID = TabB.Source_Cust_ID
WHERE T.Customer_Key = TabA.Customer_Key);
```

This design tip gives you detailed guidance to instrument your slowly changing dimensions with precise time stamps during an initial historical load or routine batch load of large numbers of records.