

Design Tip #80 Adding a Row Change Reason Attribute

By Warren Thornthwaite

We are firm believers in the principle that business requirements drive the data model. Occasionally, we'll work with an organization that needs to analyze Type 2 changes in a dimension. They need to answer questions like "How many customers moved last year?", or "How many new customers did we get by month?" which can be difficult with the standard Type 2 control columns. When this happens, we add a control column called RowChangeReason to the design. I was recently approached by someone who realized their business could use a RowChangeReason column and asked if it was possible to add one on to an existing dimension. In this design tip, we'll describe a bulk update technique for retroactively adding a RowChangeReason column to any of your dimensions.

In its simplest version, the RowChangeReason column contains a two character abbreviation for each Type 2 column that changed in a given row. For example if LastName and ZipCode changed, the RowChangeReason would be "LN ZP". You can certainly use more characters if you like, but make sure to put a space between the abbreviations. Although this is not meant to be a user queryable column, it does let us easily identify and report on change events. A question like "How many people changed zip codes last year?" could be answered with the following SELECT statement:

```
SELECT COUNT(DISTINCT CustomerBusinessKey)
FROM Customer
WHERE RowChangeReason LIKE '%ZP%'
AND RowEffectiveDate BETWEEN '20050101' AND '20051231'
```

The LIKE operator and wildcards make the order of the entries unimportant. The RowChangeReason column allows us to answer a lot of interesting questions about behaviors in the dimension table.

We'll use the simple customer dimension shown in Figure 1 as our example. In this table, FirstName is tracked as Type 1, and the other business attributes are tracked as Type 2. Since the table has Type 2 attributes, it also has the necessary Type 2 control columns: RowEffectiveDate, RowEndDate and IsRowCurrent (which we realize is redundant, but we like the convenience). It also includes the new column we are adding on called RowChangeReason.

Customer
CustomerKey
CustomerBusinessKey
FirstName
LastName
City
State
Zip
RowEffectiveDate
RowEndDate
IsRowCurrent
RowChangeReason

Figure 1 – A simple customer dimension

The process to bulk update the new RowChangeReason column takes two passes: One for the new row for each business key, and one for all subsequent changed rows. The first pass joins the dimension to itself using an outer join, treating the table as the current row and the alias as the prior row. The query then finds all the new rows by constraining on all those entries that don't have prior rows using the IS NULL limit in the WHERE clause.

```
UPDATE Customer
SET RowChangeReason = 'NW'
FROM Customer LEFT OUTER JOIN
    Customer PE ON -- Prior Entry Customer table alias
    Customer.CustomerBusinessKey =
        PE.CustomerBusinessKey
    AND
    Customer.RowEffectiveDate = PE.RowEndDate+1
WHERE PE.CustomerBusinessKey IS NULL
```

The second pass is a bit more complicated because it needs to create a RowChangeReason code for all the rows that have been added to the dimension due to a change in a Type 2 attribute. In this case, we use an inner join between the current row and the prior row which automatically excludes the new rows. We'll also use a CASE statement to generate the string of abbreviations which identifies each of the four Type 2 columns that actually had a change from their prior values. Finally, we'll concatenate the abbreviations together to make the entry for the RowChangeReason column.

```
UPDATE Customer
SET RowChangeReason = Query1.RowChangeReason
FROM
    (SELECT NE.CustomerBusinessKey, NE.RowEffectiveDate,
        (CASE WHEN NE.LastName <> PE.LastName
            THEN 'LN ' ELSE " END) +
        (CASE WHEN NE.City <> PE.City
            THEN 'CT ' ELSE " END) +
        (CASE WHEN NE.State <> PE.State
            THEN 'ST ' ELSE " END) +
        (CASE WHEN NE.Zip <> PE.Zip
            THEN 'ZP ' ELSE " END) RowChangeReason
    FROM Customer NE INNER JOIN -- NE for new entry
        Customer PE ON -- PE for prior entry
        NE.CustomerBusinessKey = PE.CustomerBusinessKey AND
        NE.RowEffectiveDate = PE.RowEndDate + 1
    ) Query1
INNER JOIN Customer ON
    Customer.CustomerBusinessKey = Query1.CustomerBusinessKey AND
    Customer.RowEffectiveDate = Query1.RowEffectiveDate
```

This SQL will work for the one-time process of adding on a RowChangeReason column if you left it off your initial design. Of course, you will still need to add the appropriate comparison logic to your ETL programs to fill the RowChangeReason column moving forward.