

Kimball Design Tip #11: Accurate Counts Within A Dimension

By Warren Thornthwaite

Any dimension table with rich descriptive attributes often becomes the direct target of queries independent of any fact table. For example, we do various counts against our Customer table to answer questions like how many customers do we have by payment type, or state, or gender, or service plan, etc. on a daily basis. Simple counts against a static dimension are obvious, but our lives become more interesting when we try to retrieve these counts from a slowly changing dimension.

Counts on a Slowly Changing Dimension (SCD)

The problem is making sure we don't over count. In the customer SCD, we'll have multiple rows per customer, so a simple query that counts customer by state will end up over counting any customer who has had any changes (and therefore more than one record). One might be inclined to do a COUNT DISTINCT of the production customer key if it is available. The problem with this is if the attribute you are counting by has changed, like a move from one state to another, you may still over count because the customer key may be unique by state. We need some way to limit the SCD to one row per customer. We can do this for the most recent version of each customer record if there is a 'Current_Row' flag in the dimension. This approach will give us counts for the most current status of our customers. You can even make a standalone Customer Count table, view, or pre-defined query that only returns current rows for generating current counts.

Counts Over Time

Current counts are always useful, but the real trick is to get counts as of any particular date in history. Understanding how values change over time is one of the primary purposes of the data warehouse. Knowing we currently have 2,311 customers is good; being able to compare this with the count from a year ago is even better. You must have a slowly changing dimension to get these kinds of historical counts. For example, if you needed to know how many customers you had at the end of 1999, you could constrain the Row_Begin_Date <= '12/31/1999' AND Row_End_Date >= '12/31/1999'. This limits the results set to only those rows that were "active" on 12/31/1999. (The choice of comparison operators depends on how you set your begin and end dates.) We have assumed in this example that when a customer dimension record is changed, the row_end_date of the first record is one day less than the row_begin_date of the second record, and multiple changes within a single day are not allowed in the data.

If you really want to get fancy, instead of directly constraining date fields in the customer dimension table, you can use the full power of the Period dimension table to supply the target date, or even multiple target dates. To do this, you use the same comparison operators to create two unequal joins between the Date field of your Period table and the begin and end dates of the customer table, and constrain the Date field in the Period table equal to the target date. You can then include the Date field in the select list to see the date the counts apply to. To return counts for multiple dates in the same query, like the last day of the month for a year, remove the limit on the Date field of the Period table and add a constraint on the Month_End_Flag = 'y'. The SQL would look something like this:

```
SELECT Period.Date, Customer.State, COUNT(Customer.Customer_Key) FROM Customer,
Period WHERE Customer.Row_Begin_Date <= Period.Date AND Customer.Row_End_Date
```

`>= Period.Date AND Period.Month_End_Flag = 'y'`

Note that this type of unequal join between the Period table and dimension table can pose a real challenge for the database engine with large dimensions. In our case, we have bitmapped indexes on both date fields, and we get pretty good performance.

Clearly, these types of queries are non-trivial to construct, and we would encourage you to shield most of your user community from query gymnastics of this nature. In fact, if requests for counts by certain attributes are common, it may make sense to create a summary fact table that includes a count for every existing combination of attributes by day. Then we are back to a simple dimensional model where each attribute is a dimension, and the fact table has at least one field that is the count of customers by attribute combination by day.