

Design Tip #130 Accumulating Snapshots for Complex Workflows

By Margy Ross

As Ralph described in [Design Tip #37 Modeling a Pipeline with an Accumulating Snapshot](#), accumulating snapshots are one of the three fundamental types of fact tables. We often state that accumulating snapshot fact tables are appropriate for predictable workflows with well-established milestones. They typically have five to ten key milestone dates representing the workflow/pipeline start, completion, and the key event dates in between.

Our students and clients sometimes ask for guidance about monitoring cycle performance for a less predictable workflow process. These more complex workflows have a definite start and end date, but the milestones in between are often numerous and less stable. Some occurrences may skip over some intermediate milestones, but there's no reliable pattern.

Be forewarned that the design for tackling these less predictable workflows is not for the faint of heart! The first task is to identify the key dates that will link to role-playing date dimensions. These dates represent key milestones; the start and end dates for the process would certainly qualify. In addition, you'd want to consider other commonly-occurring, critical milestones. These dates (and their associated dimensions) will be used for report and analyses filtering. For example, if you want to see cycle activity for all workflows where a milestone date fell in a given work week, calendar month, fiscal period, or other standard date dimension attribute, then it should be identified as a key date with a corresponding date dimension table. The same holds true if you want to create a time series trend based on the milestone date. While selecting specific milestones as the critical ones in a complex process may be challenging for IT, business users can typically identify these key milestones fairly readily. But they're often interested in a slew of additional lags which is where things get thorny.

For example, let's assume there are six critical milestone dates, plus an additional 20 less critical event dates associated with a given process/workflow. If we labeled each of these dates alphabetically, you could imagine analysts being interested in any of the following date lags:

A-to-B, A-to-C, ..., A-to-Z (total of 25 possible lags from event A)

B-to-C, ..., B-to-Z (total of 24 possible lags from event B)

C-to-D, ..., C-to-Z (total of 23 possible lags from event C)

...

Y-to-Z

Using this example, there would be 325 ($25+24+23+\dots+1$) possible lag calculations between milestone A and milestone Z. That's an unrealistic number of facts for a single fact table! Instead of physically storing all 325 date lags, you could get away with just storing 25 of them, and then calculate the others. Since every cycle occurrence starts by passing through milestone A (workflow begin date), you could store all 25 lags from the anchor event A, then calculate the other 300 variations.

Let's take a simpler example with actual dates to work through the calculations:

Event A (process begin date) - Occurred on November 1

Event B - Occurred on November 2

Event C - Occurred on November 5

Event D - Occurred on November 11

Event E - Didn't happen

Event F (process end date) - Occurred on November 16

In the corresponding accumulating snapshot fact table row for this example, you'd physically store the following facts and their values:

- A-to-B days lag - 1
- A-to-C days lag - 4
- A-to-D days lag - 10
- A-to-E days lag - null
- A-to-F days lag - 15

To calculate the days lag from B-to-C, you'd take the A-to-C lag value (4) and subtract the A-to-B lag value (1) to arrive at 3 days. To calculate the days lag from C-to-F, you'd take the A-to-F value (15) and subtract the A-to-C value (4) to arrive at 11 days. Things get a little trickier when an event doesn't occur, like E in our example. When there's a null involved in the calculation, like the lag from B-to-E or E-to-F, the result needs to also be null because one of the events never happened.

This technique works even if the interim dates are not in sequential order. In our example, let's assume the dates for events C and D were swapped: event C occurred on November 11 and D occurred on November 5. In this case, the A-to-C days lag is 10 and the A-to-D lag is 4. To calculate the C-to-D lag, you'd take the A-to-D lag (4) and subtract the A-to-C lag (10) to arrive at a -6 days.

In our simplified example, storing all the possible lags would have resulted in 15 total facts (5 lags from event A, plus 4 lags from event B, plus 3 lags from event C, plus 2 lags from event D, plus 1 lag from event E). That's not an unreasonable number of facts to just physically store. This tip makes more sense when there are dozens of potential event milestones in a cycle. Of course, you'd want to hide the complexity of these lag calculations under the covers from your users, like in a view declaration.

As I warned earlier, this design pattern is not simplistic; however, it's a viable approach for addressing a really tricky problem.