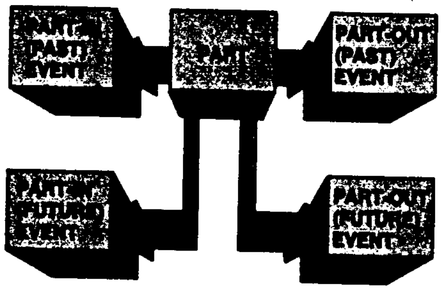
The final three installments of our 14-part series, Process-Driven Data Design.

THE TEMPLATE

by Frank Sweet

Future-event records tend to be quite volatile. Their short lives and high flowrate often call for a cookie-cutter entity to help stamp them out in assembly-line fashion.

We've been designing a maintenance equipment spare parts inventory system called MESPIS. Its goal is to report when we must reorder spare parts, based on their stock balance being too low. So far, our database looks like this:



Part, the hub of our design, contains ID-number, description, and on-hand-stock-qty for each spare part. The two past-event records document every occasion when the part's on-hand balance was updated. The two future events predict arrivals of ordered parts and planned usage. Now, we'll examine the template, or cookie-cutter, pattern.

Compute the volatility of our part-out future event. Say 10 maintenance work orders are done each month. They are planned 30 days in advance, and each requires about 100 different kinds of spare parts. Once the parts are actually consumed, their future-event records vanish, replaced by past events. Hence, some 1,000 new future part-out records are born every month, each with a life span of about one month. This 1,000-record population has a volatility of 100% per month. Work it out, and you'll see that if maintenance work orders were scheduled three months in advance, the volatility would be only 33% per month. In other words, future-event volatility is inversely proportional to the user's planning horizon; the less farseeing the forecast, the higher the volatility.

Contrast this with past events. They appear when parts are consumed and last for however long we need their audit trail. For example, keeping records in a system for six months results in a 17% monthly volatility. Past-event volatility is inversely proportional to the user's need for history; the more history needed, the lower the volatility.

Since hindsight is sharper than prophecy, any application's future is less certain than its past. Consequently, future-event records are, by far, the most volatile entities in a database. They are produced in a steady stream, live out their short lives, and vanish.

Precisely because they emerge in a steady stream, loading them with data can be tiresome. A model or template record helps. This pattern tells us that when we have a highly volatile entity, we should plan how we'll produce its fields. One way is to find a record wherein we can house standard default values for the fields.

Consider lead time. We saw last issue that our future part-in record carries a date telling when the event (part's arrival) is anticipated. Notice that this information has value even beyond the scope of our system. With it, for instance, we could produce a report comparing the expected arrival dates of those framis bearings we spoke of with their planned consumption dates. True, such an expediter's report is beyond the scope of our development contract, but it's nice to know we could respond quickly to such a request if called upon.

The problem is, where does the information come from? We could ask the user to enter it manually each time. This is more work for our unfortunate friend in purchasing. In addition to bombarding her with reorder warnings, we ask her to guess the date each part will arrive. How would she go about it? Framis bearings take four weeks, fernst gaskets take six, and light bulbs come in overnight. Knowing the nature of the part, she would add its typical lead time to today's date, giving a likely arrival date. Such lead time characterizes the part itself. It is a template datum because it helps compute a field (arrival date) in the volatile part-in future event. Hence, we should add estimated-lead time to the record layout of part-item. Similarly, we should inspect all fields in future-event records: how will each be produced? Would a template help?

Notice that we design the template to help the user, not replace him. The computed date is simply a first guess, offered as a suggestion. Final responsibility for accuracy remains with the user and we must enable him to manually overlay the computer's estimate with his own.

A productive-skepticism warning: users do not always see a template's usefulness as clearly as they do its threat. A template's goal is to handle the routine that makes up 80% of any activity, enabling users to override exceptions. But some confuse importance with ease of automation. With data such as lead time or price, users have been known to refuse template defaults and insist on having it done by hand: "Delivery-date is too important to be entrusted to the computer." Only time and