We continue our exploration of database design in part three of this 14-part series, Process-Driven Data Design.

OBJECTS AND EVENTS

by Frank Sweet

I left us with a problem last time, "The Case of the Unreserved Work Orders." We were offered values for all three terms in the equation: POOLTIME × FLOWRATE = POPULATION. We observed 200 orders awaiting parts and were informed in interviews that 40 new ones were issued each month and that their average wait time was 10 days. The question was, "What is the average pooltime?" The answer is five months, no less.

Our challenge, to estimate the average wait time, involved deciding whether to accept the numbers as given or to cross-check them against one another. The interview-given pooltime, in months (0.3), times the flowrate (40) is by no means equal to the observed population (200). The three figures we collected (flowrate, pooltime, population) are contradictory. We must decide which one is wrong.

We observed the 200 orders sitting around—that number cannot be challenged. The 40 new orders per month would be easy to verify, thus unlikely to be misstated. Besides, what motive would there be for understating it? It's more likely to be overstated ("See how hard we work"). By elimination, we come to the 10-day wait time. The number is suspect for two reasons. First, our verifying it would seem impossible. Second, it's the sort of thing higher management puts into goals or management objectives ("Orders must be processed within 10 days"). When this happens, many will unconsciously adopt a convenient fiction rather than admit an unpleasant truth. Discarding the fishy 10-day datum, we compute average pooltime to be population (200) divided by flowrate (40/month), or five months.

Incidentally, in the real case, our curiosity was so aroused by the five-month delay in getting jobs onto the floor that we launched another study just to find the cause. This eventually led to a vendor quality/performance history system.

An even more important number in design is a record's volatility, the reciprocal of pooltime. It's defined as: VOLATILITY = FLOWRATE ÷ POPULATION. Earlier, we pointed out that volatility is binary and nearly every type of record falls into one of two monthly-volatility groups: 1% or 100%. The underlying reason is that we computerize data about two classes of real-world phenomena: objects and events.

Objects are tangible things that exist independent of time. Vendors, customers, products, employees, warehouses, and ocean-going freighters are all objects. The volatility of their records is low—around 1%. Often called "base data" or "master files," involatile records store reference information such as address, location, and name or description. Objects, in other words, just sit there and don't do much.

Events are happenings; each occurs at a specific instant. Freight deliveries, shipments, labor charges, receipts, and disbursements are events. Their volatility is high—around 100%. Called "transaction data" or "permanent work files," they keep track of what's going on out there. They hold fields like date (when did it happen), responsibility (who did it), cause (why), and the like. Events, in other words, keep happening.

It's vital that we recognize which boxes (records) in our Bachman diagrams represent involatile objects and which reflect volatile events. The distinction is so important, in fact, that I once proposed using different data-diagram symbols for the two—rectangular boxes for objects, and hexagonal ones for events. Sadly, I cannot draw a nice-looking hexagon, and the technique never caught on.

But distinguishing between classes of entities is not difficult. The computation is straightforward and, after doing it a few times, we can recognize them at a glance. We notice, for instance, that the fields—shipment-date, order-date, and patient-admission-date—make sense in that the words convey clear images. Employee-date, freighter-date, and vendor-date, on the other hand, are somehow unsatisfactory and incomplete. This is because the essence of an event is that it occurs at a specific instant while an object is not so transient.

SEPERATE OBJECTS, EVENTS

There are three reasons to distinguish between objects and events. First, we can derive events from objects but not vice versa. This means that in constructing a long-range plan—an overall database architecture—for a pool of shared data, we begin with the easily identified object records and then derive events by Bachman manipulation.

Second, shared files imply standard data names and, most especially, standard keyfield formats. We sell both ideas more easily if we first apply them to involatile reference data, rather than to less widely familiar events.

Third, how we physically implement a design into our database management system depends on each record's volatility in many ways. Multiple access paths, backup/recovery procedures, and migration techniques, all vary with volatility.

Two warnings: First, don't confuse records in another system with objects. Objects have an external reality while records model either objects or events. We would err, for example, in considering a purchase order record involatile on the grounds that it models a physical thing—a document. That document is simply another form of record which, in turn, models an event—an agreement to buy something. When in doubt, compute the volatility. Second, don't be overly strict in interpreting "tangible" reality. Nations, sales districts, and colors aren't strictly tangible, yet they would be involatile object entities in a database. Again, compute volatility when in doubt.

Next time, we'll investigate keyfield design. ◉

Frank Sweet is corporate manager of data administration for the Charter Co., Jacksonville, Fla.