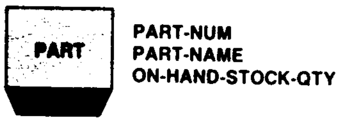
We continue our exploration of database design in parts 10 and 11 of this 14-part series, Process-Driven Data Design.

PAST AND FUTURE EVENTS

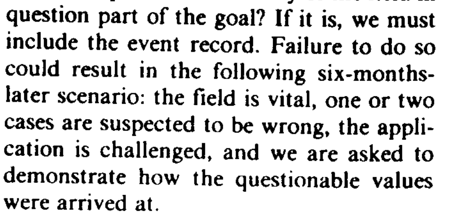
by Frank Sweet

To refresh your memory, our sample application is an imaginary Maintenance Equipment Spare Parts Inventory System. MESPIS's first scope goal is to "Keep track of the on-hand stock quantity of each item in our firm's spare parts warehouse." So far, our database design contains only one record.



Applications that record the past need past-event records. In the context of meeting our first goal, what is the single most vital data field (as opposed to ID field) in the part-item record? On-hand-stock-qty, of course. The field is actually named in the scope statement. With it, we can begin to address the other goals. Without it, we haven't a prayer.

The past-event pattern tells us that when a field is so important that the whole application hinges on it, an update audit trail isn't a luxury; it's a bare necessity. We must create a volatile entity (an event) to record every occasion when the on-hand-stock-qty was updated.



Some users might deny that audit-trail records are needed. The decisive criterion is simple: is the accuracy of the field in question part of the goal? If it is, we must include the event record. Failure to do so could result in the following six-months-later scenario: the field is vital, one or two cases are suspected to be wrong, the application is challenged, and we are asked to demonstrate how the questionable values were arrived at.

The point is not that past-event records make our demonstration easier. They avoid the challenge altogether. Consider your own experience. How often do you phone your bank because of a questionable account balance? How often would you call if its monthly statement showed only your current balance and did not list every check and deposit?

What fields should the past-event records carry? Data fields and ID fields, naturally. Data fields include quantity, date, and description. Quantity (in or out, as the case may be) is essential since the records' purpose is to justify the balance carried in the part-item record. Date is also needed for the same reason.

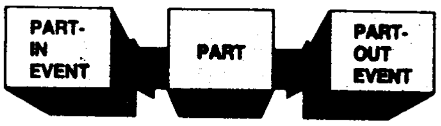
Description is not crucial since we're now dealing with events, not objects. It's not forbidden, of course, just not mandatory. The user might want to write somewhere: "This is when Harry replaced the steam trap on unit 12 because he backed the truck into it." If so, include a description. (Alternatively, since the above sentence tells us the individual withdrawing the part, the cost center to be charged, and the reason for the withdrawal, each of these elements could be codified and made separate fields.)

An identifying serial number of some sort is also needed. Without such a number, there is no way to tell a program (or a person) which specific event we are referring to in any particular case. If some law of nature decreed that there be a maximum of one "in" and one "out" for a given part on any given date, then the date itself would seem capable of doubling as the record's identifier. There are two reasons why this would be unwise. First, it's unlikely that such a law exists. Second, mixing meaningful data into identifiers leads to unresolvable confusion if those data must themselves be modified (Harry didn't take the part on Thursday after all; it was Friday). Ideally, record keyfields should be unique, unambiguous, unchanging, and dataless.

A final thought on our past events: two different boxes are shown—part-in and part-out. They might be merged into just one type of record in physical database design. The decision will pivot on the similarity of their fields and their relationships to other records. For now, it's best to keep them separate. After all, part-comes-in and part-goes-out are fundamentally different happenings in the real world.

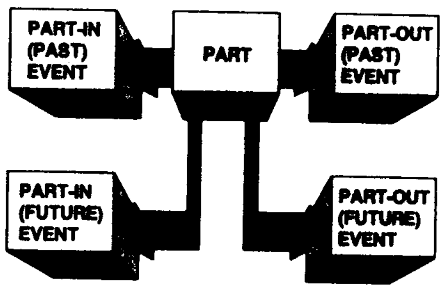
PART 11: FUTURE EVENTS

Now, in part 11, let's investigate a close relative of the past event, the future event. Systems that are meant to manipulate the future (to make something happen) need future-event records. Unfortunately, dp historically grew out of accounting, so most older systems simply record the past. So far, MESPIS looks to the past, and our design reflects this.



But MESPIS has three goals. Is one of them intended to make something happen? Consider the second goal: "Produce a report when it is time to reorder a spare part, based on its stock quantity being too low." Evidently, the report is meant to get the part resupplied; its ultimate goal is to avoid running out. This is important because it hints that our system looks to the future as well as the past. Hence, we turn our attention to the third database design pattern, the future event. It tells us that, to manipulate the future, we need future-event entities.

Adding these to our design results in the following diagram. The picture now shows one involatile object (part) and four volatile events (in and out, past and future).



The fields in the two new records will be the same as those in the past events: date, serial number, quantity, and (optionally) description, since the same rationale