A productive-skepticism warning: users do not always see a template's usefulness as clearly as they do its threat.

familiarity can alleviate fear. In this situation, our wisest course is to include templates in the design but temporarily leave them out of the processing. Thus, a typical sequence of change requests reads:

January (system newly installed): "Delivery dates must be entered by hand. They are too important to be left to the computer."

April: "We need a list of standard lead times we can refer to (while manually computing and entering all those delivery dates)."

July: "The resupply report should automatically compute each order's estimated delivery date. But this must not go directly into the database. Instead, we shall transcribe the dates from report to database, correcting each as needed."

October: "Computed delivery dates should go automatically into the database. A separate Executive Review Report must be provided, however. It should list each day's computed delivery dates so we can carefully review them and correct those in error."

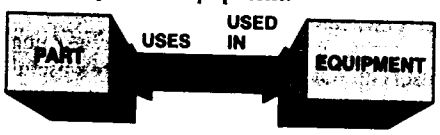
January: "Executive Review Report? Never heard of it. Oh yes, now I remember—that's the one Joe binds and files. Nobody knows what it's for."

THE SELF-RELATED RECORD

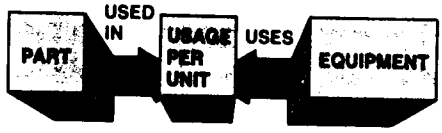
In previous issues we addressed two of the three goals in the scope statement for a maintenance equipment spare parts inventory system. We've determined that the hub of our database is a part-item record with one occurrence for each different kind of part we stock. The record holds the part's ID number, description, on-hand quantity, reorder point, and delivery lead time.

Now, in part 13, we examine the third goal, that is, to "identify, for each part, those pieces of equipment in which it is used." The statement also implies the converse: to identify, for each piece of equipment, the spare parts it uses.

One approach is to add an equipment box, that is, a file containing a record for each piece of equipment.



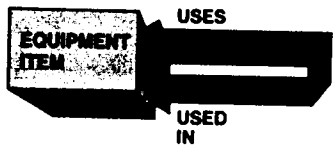
The arrow is two-headed because, while any part (e.g., lubricant) could be used in many different kinds of equipment, a given piece of equipment could require many different types of parts. Recall that a two-headed arrow warns of a missing intersection entity. Replacing it with the intersection, we have:



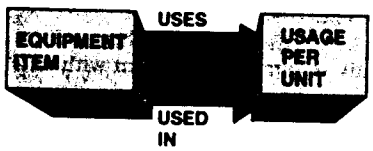
This approach is satisfactory and might do the job. Its main flaw lies in the need to discriminate between a spare part and a piece of equipment. Terminology, it turns out, often depends on context, and something that's called a spare part one moment might be termed a piece of equipment the next.

Looking at an automobile, we might consider the entire engine assembly a spare part. But if we consider the engine as equipment, its ignition cabling group (i.e., coil, distributor, wires) might be termed a spare. And, while a faulty ignition system (equipment) is being repaired, a single spark plug cable (spare part) could be replaced.

If such context-dependent terminology were the case in our application, then treating part-item and equipment as two different entities would be a mistake. A less redundant solution is to have just one entity and call it equipment/item as a compromise.



The situation regarding this two-headed arrow is precisely the same as in the prior one. It means we need an intersection entity. What makes it confusing is that the same record lies at both ends of the arrow. But, though the relationships are harder to visualize, the same rules apply. The resulting pattern is so widely used that it has a name: bill of material.



NUMBERS, KEYFIELDS, ETC.

It is dangerously easy to think a conceptual database is designed when, in fact, major issues are still unresolved. The topics keyfields, numbers, and real things comprise a form of checklist we find useful in deciding whether we are really finished. They are not new. We have mentioned all three before and in part 14 we review our prior discussions.

Keyfields—unique, unambiguous, unchanging, and dataless. Every entity should have a keyfield—an identifier that will tell a person or program which occurrence is at hand. Keyfields should be:

- Unique. Each real-world object or event should be represented by only one occurrence of its entity. Don't have two item records for the same part.
- Unambiguous. Each occurrence of an entity should model only one real-world object or event. Don't mix light bulbs and gaskets in a single time record.
- Unchanging. Once they are assigned, an entity occurrence's keyfields should remain unchanged.
- Dataless. Keyfields identify. Data fields describe entity attributes. Don't mix the two functions.

Numbers—population and volatility. An easily avoided error in database design is to neglect numerical analysis. To an intern, it may seem that the experienced surgeon takes risks. To an apprentice, the seasoned engineer may appear to guess at pressure vessel stress. Similarly, to novice database designers, veterans can seem to shortcut numerical analysis of the data. In all three cases, appearances are deceptive.

A database designer working on his or her tenth materials management system might give the illusion of being unaware that the bill of material is a complete template for supply requisitions, or that maintenance work-order volatility is between 50% and 100%. But, like the swan's effortless glide, it is an illusion that conceals frantic paddling beneath the surface. Before signing off a conceptual design, we must know every entity's population and volatility. Until we do, our design is unfinished.

Real things—objects and events. Application systems analysis usually begins by studying the existing system, automated or not. This is the easiest way to find out what it's all about. But our database design would be less than professional if it simply modeled the existing system. Our goal, after all, is to model underlying physical reality.

Every box in our design should represent an identifiable entity in the real world. Nonvolatile boxes simulate objects (or intersection data about pairs of objects). Volatile boxes model events or happenings that actually take place. No box should simply model a record in another data processing system, automated or manual.

Concluding a conceptual database design, we ask ourselves three questions:

Does every box have a unique, unambiguous, unchanging dataless keyfield?

Can we produce reasonable population and volatility estimates for every box?

Does every box represent either a real-world object or a real-world event?

Yes? Then we're done. ◉