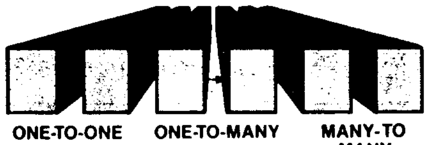
We continue our exploration of database design in parts six and seven of this 14-part series, Process-Driven Data Design.

HEADLESS AND OPTIONAL ARROWS

by Frank Sweet

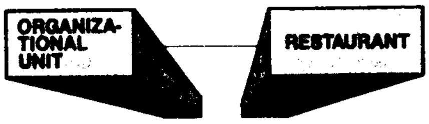
Bachman diagrams show relationships (arrows) among information entities or records (boxes) in conceptual database design. Now, symmetry seems to demand that real-world relationships should come in three flavors:



The idea is so appealing, in fact, that to say only the one-to-many flavor really exists seems to imply an irrational universe. Yet this is the case. Douglas Adams, author of *A Hitchhiker's Guide to the Galaxy,* theorizes that if anyone should discover the purpose of the universe (what it's good for), it will instantly vanish and be replaced by something even more bizarre and inexplicable. I believe this has already happened. In conceptual database design, only one-to-many relationships actually exist for long. The other two are simply intermediate design steps that must be resolved into one-to-manys.

We looked at the many-to-many relationship last time. We showed that such a two-headed arrow tells us we are not finished. It points out that there's an intersection record missing which, when identified, resolves the two-headed arrow into two one-to-many relationships.

The one-to-one headless arrow tells us that there are too many boxes present. Consider the following design modeling a restaurant chain:



Every restaurant is also an organizational unit of the firm. Moreover, each one is only one such unit. Not all units are restaurants, of course; there are also offices, warehouses, districts, and so on. But for every unit that is a restaurant, it is only one restaurant. Some units, like districts or regions, may have several restaurants reporting to them but these aren't the same thing as the restaurants themselves. We first model the situation with a one-to-one relationship, and then consider: are retail stores and organizational units really different entities? No. The terms used are merely context-dependent names for the same class of real-world tangible objects.

The general rule is that when we find a one-to-one relationship between two boxes, we replace them with just one box. The headless arrow means that we are really dealing with one entity. The final criterion, based on normalization, is that if every data element in both boxes car be uniquely and unambiguously determine ' by the key-field of either box, then you really just have one record type.

For example, the four boxes, "CICS User," "Insurance Claimant," "Credcard Holder," and "Computer Programmer" are all just different views of "Employee." "Inbound Shipment" and "Outbound Shipment" just mean "Shipment."

In physical database design, we do sometimes implement one-to-one relationships in order to conserve core, disk-space, I/O, or cpu cycles. Consider an employee record with fields for executive stock options and bonuses. Since these data apply to only a few employees, the record would have much empty space for most personnel. Wasted disk space at $2 per megabyte per month (3350 rental) is not as costly as it once was, but it's still irritating. We could compress the employee record, but that costs cpu cycles and makes restructuring more difficult. A cleaner solution is to hang a smaller record off it, to hold the fields that apply only to executives. Only an employee record that needs those fields would own such a subordinate record and, at most, it would own just one.

As another example, Ken Thorn, of Giant Food Inc. in Washington, D.C. asks, "What's the relationship between 'states of the union' and 'governors of states'? Clearly, they bear a one-to-one relationship with each other yet they are not the same entity (one is more organic than the other)."

Part of the problem yields to better definition: if it's an American history database, for example, where we store political biographies, a one-to-many relationship is revealed. Each state, since its entry into the Union, has had many governors. If, on the other hand, we only mean to capture data about current officeholders, then normalization tells us that they are truly the same information entity, despite intuition.

But now we're sailing the treacherous shoals between theory and reality. For, even if our users earnestly promise that all they'll ever want is current data, experienced database designers would still make them separate physical records. History-keeping (audit trail) is eventually required by almost every application and it would be easier to add it if the volatile portion (elections in this case) were separate. That way, you could add an effective-date field to the governor record and keep historical occurrences alongside the current one.

But these one-to-ones are made to fit the conceptual design into the limitations of our software, hardware, or planning ability. In a theoretically perfect conceptual design, neither the headless arrow nor the two-headed arrow would exist.

PART 7: SPLITTING A BOX

Now, as we begin part seven, let's look into how we go about splitting a box in two when an optional arrow appears.

We've been manipulating data structure, and operating conceptually on Bachman diagrams while designing a database. In part five we saw how to derive a new data entity (a box) with the two-headed arrow rule. Just above, in the headless arrow discussion, we showed how to eliminate a box by merging two entities into one. Both rules are reliably objective; like arithmetic, they always work.

Next, we'll inspect two more manipulations: splitting entities with optional arrows and merging those with similar relationships. These rules are more subjective than the prior two. In other words, they