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

THE TWO-HEADED ARROW

by Frank Sweet

Bachman data-structure diagrams consist of boxes and arrows. The boxes represent data entities or types of records. The arrows depict relationships between records. Each arrowhead marks the "many" end of a relationship.

Bachman diagrams are useful to designers for two reasons. First, they pack so much information into a succinct, easily reproduced form. A few lines sketched on a blackboard or notepad replace tedious, easily misunderstood explanations. Also, we can manipulate the symbols, like the terms of an equation, to derive a detailed conceptual database design from a high-level summary. Diagram manipulations let us conclude things about our design. We can test it for consistency and examine alternatives before spending days and dollars physically building it in our shop's database management system.

The three basic diagram manipulations are these: two-headed arrows produce intersection entities, headless arrows will merge entities, and optional arrows will split entities. We'll cover these rules and others in the next few issues.

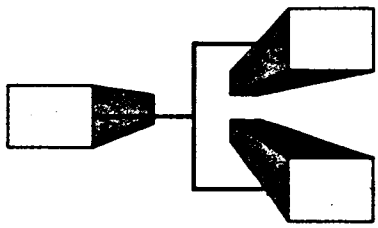
First, though, consider the symbols themselves. Boxes model data entities, the things about which we'll store data. They are important because once implemented, they become different types of records in the database. Arrows model interrecord relationships. They are important for three reasons.

First, they embody referential data integrity. In other words, a "vendor" box pointing to a "purchase order" box means that we must not store a new purchase order unless it is associated with a valid vendor. Neither should we erase a vendor as long as it has purchase orders associated with it.

Second, arrows show important access paths (predefined JOINS, in relational terms) that the finished application will use (for example, given a vendor, find its purchase orders). Finally, many database management systems use disk address pointer chains or arrays to relate records to one another. Conceptual relationships are a starting point for defining these physical relationships.

Here are six sample Bachman diagrams. The first three are not valid in a finished conceptual database design.

No arrowhead. Which of the two is the "many" end? We'll talk about that next time.



Conceptually meaningless, although this can be implemented in some database management systems.

This one is conceptually meaningful but theoretically impossible. This situation does not occur in real life. We will discuss it in a moment.

The next three, though bizarre, do make sense.

Quite common actually, a bill-of-material structure.

Weird but legitimate. I've seen only one like this. It belongs to a large-city rescue squad's mapping database.

Also common in real life—each organizational unit reports to one and only one other unit, but each may have several units reporting to it. It is not directly implementable in most database packages.

Let's examine the first diagram manipulation: the two-headed arrow and the missing intersection. A two-headed arrow means an entity is missing from our design. There's a record out there that we must identify and include before translating conceptual into physical design.

The above figure tells us that the relationship between the two records is not simply one of header detail. Since an arrowhead is the many end of a one-to-many relationship, twin arrowheads don't tell us which is the many end. Say we wanted a part-number catalog master file as well as a file of purchase orders. Obviously, purchased parts and their purchase orders are related in some way, but where does the arrowhead go? One PO can include many different parts, putting the arrowhead on the right. But wait, any one part can be included in many different purchase orders. This means that the arrowhead goes on the left.

Our problem is caused by lack of an entity. The many-to-many situation indicates that we are missing a record. There is a box, a thing about which we need to keep data, that we have not yet identified.

Think about it. The PURCHORD record carries data about a purchase order (independent of what individual items are in the PO). The PARTNUM record holds a part's catalog data (regardless of any POS that exist). Where does "quantity ordered" go? Not in PARTNUM, because any one part could be ordered in different quantities on many POS, but not in PURCHORD either: a PO can include different quantities of many