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

DATA INTEGRITY AND THE IDENTIFIER

by Frank Sweet

Database design is full of compromises. Data integrity and ease of use are fundamentally at cross-purposes. We cannot avoid trading one for the other. The problem isn't a software limitation; its roots lie in the very reasons we collect and store information. We want it to be right. We want it to be on time. We can never have both. Today we'll look at two examples of compromise, the difference between mandatory and optional interrecord relationships—hard sets vs. soft sets—and the desirability of summary fields.

Consider the following Bachman data structure diagram where we may have any number of invoices on file for a vendor, but each invoice must be related to only one vendor.

There are two ways to physically implement such a relationship between records (called a "set" in Codasyl-COBOL) in most databases. One is where each vendor record is the starting point of a chain of disk address pointers. (The vendor record points to the first of its invoices, which points to the second, etc.) In the other, each invoice simply contains its vendor number as a data field and pointer chains are not used.

Both approaches enable a program to retrieve the vendor record, given an invoice. With the first, it would read the invoice record and then use the database management system's obtain-owner or get-parent command. Under the second, the program would still read the invoice first, but then it would move vendor number from the invoice record to the vendor record's direct-access key and read it directly.

Both approaches let us extract all the invoices for a given vendor, although the pointer chains do it more efficiently. Here, the program would read the vendor file, then use its chain-following command (get next within parent or obtain next within set). Without pointer chains, the program would simply read every invoice in the file and selectively process only those that contain the given vendor number.

Where the efficiency of given-a-vendor-get-the-invoices activities is unimportant, the major difference between the approaches is in the strictness of data validation that the database management system can provide. "Hard set" is where we use the DBMS itself to guarantee data integrity—to ensure that each invoice is related to a valid vendor. "Soft set" is where we build integrity checking into the application. Pointer chains enable hard sets.

With a hard set, each invoice must be connected to a valid vendor. The rule is simple and strict. Look at four sample situations:

- Initial oversight. Imagine that a programmer fails to follow specifications or the specs don't say vendor number must be checked before storing a new invoice record. The DBMS would refuse the command and return an error code unless a valid vendor had been accessed first.
- Subsequent maintenance. Say postimplementation maintenance requires that users have the ability to change the vendor with which an invoice is associated. Again, if the changed program neglects to check the new vendor's validity, the DBMS would disallow the operation.
- Owner deletion. Let's look at it from the other angle. If a user or program attempts to erase a vendor record from the file, the DBMS won't allow the operation if any invoices are attached to the vendor.
- User override. Perhaps the user needs the ability to put invoices into the system before assigning them to a vendor (maybe identifying the vendor takes several days). The DBMS will not allow it.

Don't misunderstand. We aren't saying that any particular database package compels this degree of strictness; none does. But they enable it, and we're describing what happens when we take advantage of the capability.

DANGER OF ORPHANED INVOICES

With a soft set, the application program—not the DBMS—controls data integrity. Look again at the sample situations. In the case of initial oversight, a new invoice could be stored without a valid vendor number or with no vendor number at all. In subsequent maintenance, if the update program moves a nonvalid vendor number into the invoice and modifies it, it will remain inaccurate. Without hard-set integrity, a vendor could be deleted, leaving dozens of invoices orphaned and no longer meaningful. And, if the user needs to put invoices into the system before assigning them vendor numbers, there's nothing to prevent it.

Which is best? There is no answer. The trade-off, as we warned, is between data integrity and ease of use. Look at it this way: the power of the hard set is that it enforces strict referential integrity whether analysts, programmers, or users want it or not. That is also the hard set's weakness.

Summary fields are another area of compromise. Consider a hierarchical database. Not a hierarchical DBMS—that's just another name for IMS/DL1. I mean really hierarchical. You know, with records for parishes, dioceses, archdioceses, and so forth, right on up to the Pope. Each parish record contains a field, number-of-faithful, telling how many members it has. Each bishop's diocesan record holds the same data element, enumerating total parishioners in the diocese, and so on. These are evidently redundant and are called "summary fields" because each contains the sum of the same field in subordinate records. Summary fields are sometimes useful. They are always risky.

Summary fields are often useful because they enable one program to do the tedious summarization work for many others. For example, say many programs need