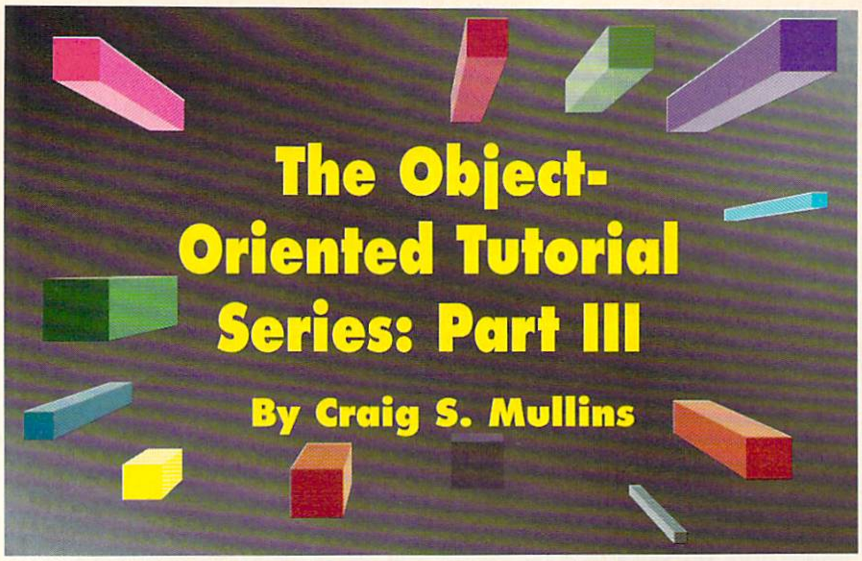


The Object-Oriented Tutorial Series: Part III

By Craig S. Mullins

OBJECT VS. RELATIONAL

In the May issue of *Data Management Review*, we continued our exploration of object-oriented data base management techniques. The information in the first two installments of this series provided an overview of the basic ingredients in the Object-Oriented (OO) paradigm. But the data base world these days is decidedly relational. What are the major differences between Object Data Base Management System (ODBMS) and Relational Data Base Management System (RDBMS) technologies? Are there any similarities? Can the two coexist or should they be combined? Let's take a closer look at these questions as we compare OO technology to relational technology.

WHY OBJECT-ORIENTED INSTEAD OF RELATIONAL?

When OO proponents seek to promote ODBMS over RDBMS, several arguments are heard over and over. First and foremost is the ability of an ODBMS to support complex objects in an efficient and easy-to-manipulate form. Examples of complex objects would be bill of materials hierarchies, CAD diagrams or multimedia BLOBs. Although some relational data bases can process these types of objects, seldom is it easy or efficient to do so. Imagine trying to use DB2 to store pictures in VARGRAPHIC columns or to explode a BOM hierarchy from a fully normalized table and you'll get the idea.

OO proponents also tout ODBMS products as able to resolve the impedance mismatch problems encountered with RDBMS products. Impedance mismatch refers to the difference between the declarative, set-level operation of relational data base query languages (such as SQL) and the procedural, record-level operation of typical third generation programming languages, such as COBOL or C. When accessing a relational data base, a typical application will require embedding SQL within a 3GL. Because the two languages operate at different levels, a mechanism is required to resolve the difference. (In DB2, this mechanism is called a cursor.) On the other hand, access to data in an ODBMS is coded using an OO programming language (such as C++ or Smalltalk), so no impedance mismatch is encountered. All operations are at the record level.

And objects within an ODBMS are organized more closely to the way in which we view them in the real world. For example, consider the portion of a data model shown in Figure 1. This is an entity type hierarchy. The physical implementation of such a hierarchy is often difficult. The following characterize how you would view this relationship as an ODBMS or a RDBMS:

- ODBMS: implement one object of class VEHICLE, and three objects, AUTOMOBILE, PLANE and MOTORCYCLE based upon the class VEHICLE; relationships are established by inheritance from VEHICLE to the subordinate classes and each class will inherit the methods coded for vehicle; or
- RDBMS: implement four separate tables, one each for VEHICLE, AUTOMOBILE, PLANE and MOTORCYCLE; code the appropriate referential integrity; and code different algorithms to access each.

Finally, RDBMS products do not provide encapsulation or abstract data types. With encapsulation, reusability is increased. And abstract data types enable data base designers to easily implement correct and appropriate data stores for the applications being developed. See Figure 1.

In general, by raising the level of abstraction, ODBMS products make it easier for developers to design and physically implement a logical data model using an ODBMS.

THE RELATIONAL ARGUMENTS

The relational proponents have counterpoints to each of the above OO points. Let's take them one by one. First, there is no reason why RDBMS products cannot be extended to support complex objects. For example:

- Couldn't multiple tables be connected together to form a complex object? Consider an object such as an ORDER in an Order Entry system. Typically, orders are composed of a single ORDER_HEADER row, multiple ORDER_DETAIL rows and possibly ORDER_DESCRIPTION rows. If an RDBMS could enable users to access all of this information with a single operation, lock all tables at the object (ORDER) level, and enforce integrity across all rows for each ORDER key, wouldn't this serve the same purpose as complex objects?
- Do multimedia objects such as BLOBs or sound have to be stored directly in the data base? Can't the data base point to some external medium on which the multimedia objects are stored? And wouldn't relational access be relatively easy to implement for this?

- Couldn't an operation be added to SQL (or any other relational data base query language for that matter), to explode a BOM hierarchy from a normalized table? And with the appropriate structure, say a linked list that traverses the hierarchy, wouldn't performance be just fine?

And what of impedance mismatch? A relational proponent would claim that OO technology "solves" the impedance mismatch "problem" by going backwards in time to the days of COBOL and record-by-record access. It is no longer possible then, to access many rows (or records, or whatever) with one operation; instead, the programmer would have to go back to coding a read loop. Is this a solution or does it take power away from the developer?

The whole abstract data type issue is simply another name for domains. The relational model has incorporated the domain concept since its inception in 1969. Simply because RDBMS vendors do not support a concept as key to relational technology as domains, is no reason to knock the relational model. When RDBMS products fully support domains (including support for user-defined data types) then they will be every bit as effective as ODBMS products with abstract data types. Probably more so!

Finally, aren't referential integrity (RI), FIELDPROCs and EDITPROCs a type of encapsulation? The issue is "at what level of abstraction are we talking?" And RDBMS products can (and will) be enhanced to support a more complete form of encapsulation in the future.

THE PROBLEMS WITH OO

What about the problems with ODBMS? The relational model is founded upon the rigorous mathematics of set theory. ODBMS products are not based upon any standard OO data model. In fact, there is no OO data model. Oh, sure, there are certain OO techniques that are standard in most ODBMS, but there is no data model in the way that there is a Relational Data Model. And there is no single OO guru such as Dr. Ted Codd, the originator of relational theory.

And what about simple querying in an ODBMS? It is not as easy as with SQL. Why should I have to code a C++ program just to produce a simple report? An RDBMS will allow me to produce a quick and dirty report with a single SQL query.

Finally, do we have to go back to the days of having the programmer decide which way is best to access data? RDBMS products provide systematic optimization. The data base knows the most efficient way to access the data. With an ODBMS, we go backwards yet again.

OO STRIKES BACK

Well, if RDBMS products can be extended to incorporate OO techniques, then ODBMS products can be extended to provide components of relational technology. Why can't an ODBMS provide systematic optimization or a query language?

In fact, several ODBMS products on the market today do, in fact, provide these capabilities. However, the jury remains out on just how useful and efficient systematic OO optimization will be. If systematic optimization co-exists with explicitly programmed, non-systematic optimization (as is typical in ODBMS products), then confusion will result. What if a programmer indicates that a specific index is to be used for a certain query, but that index was subsequently dropped? Will the query fail? Will systematic optimization "kick in" at run time? Supporting both systematic and explicit access path specification will result in a horrifying mess for maintenance programmers simply trying to determine how data is actually being accessed.

Object SQL is also problematic. For example, one of the basic tenets of the relational model is the non-subversion rule. Loosely translated, this rule states that data base updates must not be able to bypass system-defined integrity rules. An ODBMS defines data integrity rules using encapsulated methods. If OSQL permits updates, deletes and inserts, then it must also enforce all of the "integrity-related" encapsulated methods (written in C++, Smalltalk or some other OOPL) for the objects upon which it acts. This could prove to be a difficult task. If modification is not permitted, then arguably one of the basic benefits of a relational DBMS (set level modification) will be lost when moving to an ODBMS.

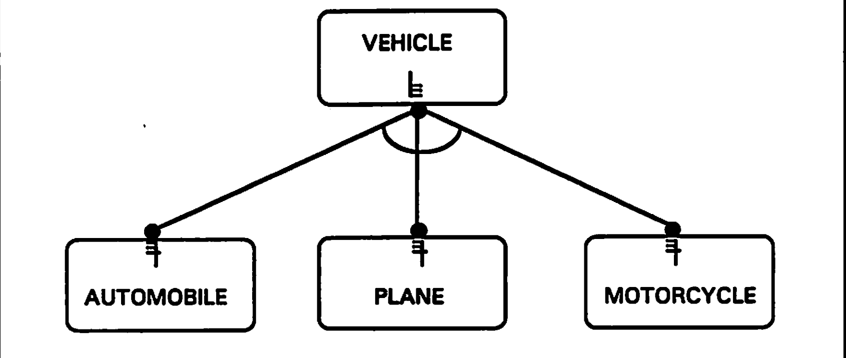
EMOTIONAL ISSUES

Indeed, these are very emotional issues. OO technology threatens those of us content with relational technology in much the same way that relational technology threatened IMS and IDMS experts. So, let's try to remove the emotion and discuss the basic issues.

The first problem is that there are currently no complete implementations of the relational model. For example, complete domain support is missing from every RDBMS product on the market. Do we compare relational theory with OO theory or relational reality to OO reality? Truly, neither is fair, but then life is not fair.

The second problem we encounter is that there is no standard OO model. This must be remedied in order to compare relational theory to OO theory.

Figure 1: Data Model - A Sample Entity Type Hierarchy



And what if we decide to compare relational reality to OO reality? What is an ODBMS? Not every DBMS that states it is object-oriented, actually is an ODBMS. This is somewhat analogous to the period in time where every DBMS was rushing to claim that it was relational. For example, Adabas, an excellent inverted list DBMS and IDMS, a CODASYL/Network DBMS, both claimed to be relational. Adabas with very few changes, IDMS with many changes, going so far as to rename itself IDMS/R. IBM, on the other hand, took the appropriate approach at the time by not tinkering with IMS to "relationalize" it, but by creating a second DBMS, DB2, that adhered to relational tenets.

Maybe even more difficult is the question: What is an RDBMS? True, we have the relational model to fall back on. But, if we do that, then our answer must be that there are no RDBMS products because no DBMS supports every relational capability described in the model. But surely DB2, SQL/DS, Informix, Sybase and Ingres are RDBMS products! But is dBase IV? Probably not. There is a gray area here that is difficult to clarify.

BACK TO THE FUTURE

I suspect that it is impossible to completely and fairly compare RDBMS and ODBMS. The differences are just too great. Although OOP has its roots in the 1960s, just as the relational model does, ODBMS is a relatively new idea that really didn't begin until the 1980s. Is it fair to compare a mature discipline with an immature one? We can, however, make some basic assumptions that will help data base developers.

But we can say this: relational is not, and never can be OO. Likewise, OO is not and never can be relational. But, if OO cannot be relational, and relational cannot be OO, what will happen? Here are my predications:

- There will always be both true relational and true object-oriented DBMS.
- Pure relational data bases will continue to prosper and thrive in the world of business data processing where applications like payroll and accounting do not generally require complex objects.
- Usage of pure object-oriented data bases will continue to grow in those fields requiring complex objects, such as CAD and manufacturing.
- Successful relational DBMS products will seek to incorporate the best components of OO technology without compromising the relational model.
- As RDBMS products begin to support more and more relational and OO features, relational data base technology will maintain its status as the industry leader, and increase their user base even more dramatically.
- Successful object-oriented DBMS products will seek to incorporate the best features of the relational model, without compromising the benefits derived from classes, inheritance and encapsulation.
- Desperate relational DBMS vendors will seek to incorporate OO technology in a haphazard fashion, not caring whether they subvert the relational model. Beware of these DBMS products.

OBJECT RELATIONAL

The immediate future of the DBMS is what IBM has coined object relational (OR). A division within IBM, the Data Base Technology Institute (DBTI), has been working on a new OR data base model. Although no products have been announced, it is a safe bet that IBM will use the research being done by DBTI in all of their relational DBMS products (DB2, DB2/2, DB2/6000, SQL/DS and SQL/400).

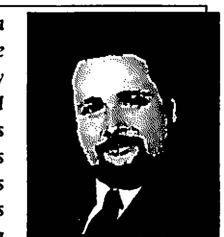
What object relational provides is a marriage of the best relational and object-oriented concepts. In the long run, DB2 will be enhanced to incorporate object-oriented features. So, instead of a new ODBMS, say DB3, IBM will be promoting the extension of DB2 to Object Relational, making, say DB2++. This is most likely the same path that other RDBMS vendors (such as Oracle, Ask and Sybase) will follow.

The best course of action for data processing professionals who must work with DBMS software is to keep up-to-date with the industry. To do that you will need to read technical publications (such as *Data Management Review*) but also additional literature on both object-oriented and relational technology. Even though the RDBMS comprise the majority of DBMS products on the market, few of us who work with relational technology understand the entire relational model and what it implies.

THE ODD COUPLE

The merging of OO and relational technology promises to be very entertaining over the course of the next few years. Hopefully, it will be done properly. If not, we will definitely be living with more of what I call "Odd Couple" products like IDMS/R. If those OO techniques which do not conflict with the relational model are incorporated into RDBMS products, then the object relational approach will be able to deliver significant gains in the SDLC. However, there are many questions that remain to be answered. Which OO techniques do not conflict with the relational model? Can tables inherit properties from one another? If they do, how should it be done? This is making relational more hierarchial! Can code be encapsulated within tables, within tablespaces, within data bases? Almost certainly, but how? And, if we incorporate the best relational techniques into ODBMS products, which relational techniques do not conflict with OO theory?

Well, for the answers to these and other questions you'll have to wait for Part IV of the Object-Oriented Tutorial Series in a future issue of *Data Management Review.*



Craig S. Mullins is a member of the Technical Advisory Group at PLATINUM technology, inc. He has more than seven years experience in all facets of data base systems development, including developing and teaching DB2 classes, systems analysis and design, data base and system administration and data analysis.

Was this article of value to you? If so, please let us know by circling Reader Service No. 35.