International Data Corporation

Object Database vs. Object-Relational Databases

Steve McClure

IDC Bulletin #14821E - August 1997

Table of Contents - **Abstract** - **Document**

IDC Opinion

When you don't know where you want to go, any road will take you there. Users need to understand the differences between object database management systems and the newer object relational database management systems (ORDBMSs). ORDBMSs will support some of the object extensions needed by today's more complex applications. Because of the relative size of the RDBMS vendors' marketing infrastructures, the ORDBMS market will surpass the size of the ODBMS market in the next five years. Nevertheless, putting object extensions on RDBMSs is tantamount to adding stereo radios and global navigation systems to horse-drawn carriages. You will have interesting enhancements, but the wrong base vehicle. In the end, it won't be the appropriate vehicle for the information superhighway.

The Database Market Is in Transition

We are in a period of intensive change and innovation regarding database technology and related products. Nineteen ninety-six ended with a flurry of product announcements from database vendors that highlighted the trend toward extended support for additional data types -- also being called complex data. The data types to which we refer are multimedia types (text, image, audio, video), other new data types like time series or geospatial, and any data type a user may wish to define. These are extensions from the very limited, simple, traditional data supported in the mainstream relational database products.

Several of the recent announcements had the theme of "universal" in their product names or marketing messages: Universal Server, Universal Database, Universal Warehouse, among others. Some are middleware-oriented architectures reflecting future directions for their product development. Prominent among this flurry of activity is the promotion of a new extended version of relational database technology surfacing under the name "object relational" database management system (ORDBMS). It is

this new class of database we want to discuss and contrast with the relational databases from which they are evolving and also with the pure object databases they will never replace. Table 1 shows a list of representative vendors and their products.

Table 1

Database Management System Products by Vendor, 1997

	Products		
Vendor	**RDBMS**	**ORDBMS**	**ODBMS**
Oracle	Oracle 7.x	Oracle 8.x	
Sybase	System 10/11		
Informix	Dynamic Server	Universal Server (Illustra)	
IBM	DB/2	Universal Database (DB/2 Extenders)	
UniSQL		UniSQL/X	
Unisys		OSMOS	
Computer Associates	OpenIngres		Jasmine
Gemstone			Gemstone
O2			O2
Object Design			Object Store
Objectivity			Objectivity/DB
Versant			Versant ODBMS

Source: International Data Corporation, 1997

The Market Driver -- Future Competitive Advantage

Competitive Advantage by Using Information Technology

The vast amount of information in today's enterprises is not the type of data traditionally stored in relational databases. These other types of information are instead kept in files, spreadsheets, or, mostly likely, in nondigital form. By now the relational databases have been employed to automate most of the obvious back-office and, more recently, front-office applications for today's enterprises. Any competitive advantages derived from that automation activity is diminishing. To find other information technologies to leverage for competitive advantage, organizations are turning to the Internet/intranet and to a richer set of data types. To keep pace with their customers' needs, almost all relational database vendors are scrambling to extend the capabilities of their product lines to support Internet-enabled applications and the multimedia data types typically found on the Web. The World Wide Web promises global access

from a "universal client." Why not then a universal database or server?

Virtual Organizations

Once the automation of the front office and field force is accomplished, the next big wave of application demand will be to support a variety of "virtual" organizations that encompass external stakeholders, such as suppliers, channel partners, development partners, and customers. This is the future and the context for the turmoil in the database market. IDC has referred to this future as the "wired marketplace."

Objects Help Manage Complexity

It is not just about new data types. The new distributed applications will be built more and more on modular, object-oriented architectures, especially in the context of components (i.e., software components). Object-oriented or object-based architectures are very appropriate for managing complexity (e.g., complex data relationships). Competitive business pressures today are increasing the level of complexity that business software must model and support. As more and more applications are implemented in these object-oriented or object-based architectures, there will be increasing pressures on application developers to have high-performance storage mechanisms that are fundamentally compatible with the object model. Hence, this demand makes object database management systems (ODBMSs) necessary. ODBMSs are defined and implemented explicitly to provide efficient storage for object-oriented applications.

TOYOTA DRIVES SALES WITH JASMINE

Multimedia Application Brings Exciting New Capabilities To Auto Dealerships Throughout Australia

---

Once the domain of high tech ad agencies and elite software development shops, multimedia applications are entering mainstream business use. As companies realize the tremendous impact that multimedia presentations have on viewers, corporate developers are scrambling to discover accessible, easy to use tools for constructing applications that incorporate complex graphics, audio and sound. Many of them are finding the answer with Jasmine, the industry's first visual development environment designed explicitly for object-oriented, multimedia applications. Toyota of Australia is a prime example.

"We wanted to create a dealership kiosk that would absolutely captivate the customers' attention," says Kevin Smith, National Parts & Accessories Systems Manager at Toyota. "This kiosk, to be located near the sales desks, would greet buyers with an eye-catching multimedia display that would draw them in. Then, beginning with a bare-bones image of the

car they were interested in, customers would be able to literally build the car of their dreams. If they wanted to see the car equipped with a rear spoiler, or all dressed up with the sports package, a mere click of the mouse would change the image accordingly.

Smith and his colleagues were driven by a common business motivation: money, and the opportunity to expand the sales of genuine accessories at the time of vehicle purchase. Customers were buying the standard Toyota vehicles and later purchasing accessories from auto supply stores selling non-genuine accessories.

"It is important to us that Toyota vehicles are equipped with genuine Toyota accessories," says Smith. "They are the highest quality goods, designed to meet modern day safety requirements."

Taking definitive action, Toyota decided to build a multimedia application prototype that would allow customers to browse and purchase accessories in an engaging and interactive environment. Toyota's goal was to create an application so compelling that customers would not even dream of leaving the dealership without a fully-loaded vehicle.

The car manufacturer knew that to be effective the application had to be appealing. Video, audio, animation-the wide spectrum of multimedia-were essential to capture and retain customer interest. It also had to be highly intuitive, and even fun to use. Buyers would quickly become frustrated with an application that was confusing or, even worse, boring.

Technically speaking, the application had to be database-driven to facilitate management of an enormous volume of continually changing information. And, most importantly, the application had to be built fast. Toyota did not want to waste one minute in taking the fight to the off-brand competitors.

"Clearly, an object database was essential for what we wanted to build," Smith explains. "There is no other way to efficiently store the kind of elaborate multimedia we intended to use. And we didn't have the time to waste fiddling around with difficult programming languages like C++. We needed an intuitive, visual development environment that was seamlessly integrated with the database to meet our prototype development schedule."

After attending a three-day Jasmine technical training class at CA world headquarters, Smith and a colleague went to work on their solution, which they developed for the Windows NT platform.

"The Jasmine Application Development System was easy to grasp," Smith says. "It is so easy to use that we could actually start building the application as soon as we returned from the class. Jasmine Studio's drag-and-drop, point-and-click development scheme allowed us to create a large, sophisticated application very quickly."

Today, Smith and his colleagues are continuing to refine the prototype with a view to the possible deployment on the World Wide Web. This would enable their overseas customers in Middle East, SE Asia and the Pacific to have access to the latest available accessories.

"That's another one of Jasmine's strengths," Smith says. "The same version of an application can be deployed wherever we choose -- in an application, on client/server systems and on the Web."

Jasmine and CA were instrumental in helping Toyota explore the opportunities of using a multimedia application to increase genuine accessory sales." says Smith.

Competing Data Models

Definitions

There is more debate than consensus on the definitions of database management system categories. For the purposes of this discussion, we will define three basic categories and try to differentiate them. The description of database type is separated into three parts:

- Data model
- Query language
- Computational model (navigational access)

In the interest of brevity, we are not attempting an exhaustive discussion of all possible aspects of database technology that could be used to evaluate products. It should also be noted that any one product may only partially conform to the full potential of the category of database to which it belongs.

Relational

Relational database technology was originally described by E. F. Codd and later implemented by IBM and others in products. The eventual standard to describe relational databases is published by ANSI as the X3H2 specification and its ISO counterpart. More popularly, it is referred to as SQL plus a version number, the latest being SQL2. RDBMS technology has a certain mathematical purity that some academics would argue has not been faithfully implemented in the major vendors' products and is being further vitiated by the object extensions now being discussed for SQL3.

Data model -- An RDBMS stores data in a database consisting of one or more tables of rows and columns. The rows correspond to a record (tuple); the columns correspond to attributes (fields in the record). Each column has a data type (e.g., date). The types of data that can be stored are confined to a very limited (six or so) number of data types -- for example, character, string, time, date, numbers (fixed and floating point), and currency. Any attribute (field) of a record can store only a single value. Variable length fields are not supported. Relationships are not explicit, but rather implied by values in specific fields, foreign keys in one table (e.g., departments) that match those of records in a second table (e.g., employees). Many-to-many relationships typically require an intermediate table that contains just the relationships.

Query language -- A view is a subset of a database that is the result of the evaluation of a query. In an RDBMS, the view is a table. RDBMSs use SQL (SQL2 currently) for data definition, data management, and data access and retrieval. Data is retrieved based on the **value** in a certain field in record. The types of queries supported run the gamut from simple single-table queries to very complicated multitable queries involving joins, nesting, set union/differences, and others.

Computational model -- All processing is based on values in fields of records. Records do not have unique identifiers that are immutable during the life of the tuple. There are no provisions for references from one record to another. Examining the result of a query is done under the control of a cursor that allows the user to step through the result set one record at a time. The same is true for updates.

JASMINE GIVES L'OREAL A MULTIMEDIA MAKEOVER

Advertising and Marketing Collateral Will Be Available Worldwide Via Corporate Intranet

L'Oreal, one of the largest international cosmetics companies, is famous for its colorful advertising and marketing materials. Each year, the company produces numerous television commercials, as well as vivid full-page ads that appear in magazines around the world. The volume and diversity of these materials, combined with the many L'Oreal offices that need access to them, makes managing all this creative output extremely challenging.

"A global firm must have consistency in its message," said Michel Barry, L'Oreal Technical Director. "This is very hard to do when the materials are in traditional form--film, videotape, advertising slicks, etc. These must be reproduced and distributed, which is a very time-consuming process."

L'Oreal's business isn't standing still, either, which means each time the creative material is modified, the whole process has to be repeated. "We currently use a database-driven solution for multimedia management, but we are very interested in moving to next-generation technology."

Searching for a way to better manage this plethora of multimedia data, L'Oreal decided to investigate CA's Jasmine. "Jasmine's top-to-bottom, object-oriented architecture is very appealing to us. The database stores video as video, audio as audio, images as images. Jasmine provides the optimum solution for centralizing and managing complex information. And, because it is completely Web-enabled, it provides a solid foundation for an Intranet project," he said.

Jasmine combines a pure object database with a powerful multimedia development system. Using the Jasmine™ Studio, L'Oreal quickly built a prototype application.

Building the application was a matter of dragging objects from the database on to application "scenes" and assigning behavior with a few clicks of the mouse. To make the application especially powerful, L'Oreal added a search engine from Verity, a Jasmine Development Partner. The Verity technology allows users to capture key concepts and features of documents, rapidly searching and organizing results into categories that are either user-defined or discovered on the fly. Verity's knowledge-based indexing and retrieval offer a combination of excellent performance and concept-based searching.

Given L'Oreal's reputation for sophistication, the company wanted its future application to look good. "For all its simplicity, Jasmine Studio is an extremely powerful tool that will enable us to build a complex, highly-stylized application," he said. "With Jasmine Studio, we'll leave behind the Windows world of pop-up boxes and pull-down menus and build a new, more intuitive system."

The main scene of the L'Oreal prototype application features a detailed and richly-appointed image of an executive office, complete with desk, rolodex, bookshelf, television and CD player. By clicking on the various items, users are able to search for and view specific items, then use them in new ways. A video marketing clip, for example, might be downloaded on to a laptop and used during a sales call.

"A user-friendly interface is essential. This application is designed to be deployed on an intranet. We want an application that will not require a steep learning curve. That's one of the things Jasmine Studio can give us," the L'Oreal Technical Director said.

"In addition to its elegant user interface, the database underlying the application offers a complete array of mission-critical features. Jasmine supports all the features that give object-oriented databases their power, such as inheritance, instance-level and class-level properties, instance methods and class methods. It has the ability to store complex data objects as true objects and not just blobs of data. This greatly simplifies the development of complex applications," he said.

"This Intranet is an important system for us. With Jasmine, we're confident that our new application will be grounded in solid, extensible technology. The database, the development tool and the execution environment form a seamless whole that is perfectly geared to next generation enterprise application development."

Relational Variations on a Theme

Post-relational (or nested relational) is sometimes thought of as a special case of RDBMS that can store and retrieve (but not query) multiple values (e.g., a list) in a single field of a single tuple. For example, a field in an employee record could contain a list of hobbies.

Another special case of RDBMS is **BLOBS**, in which support has been added for the storage and retrieval of additional data types, especially multimedia data types (e.g., text, images, audio clips, videostreams, etc.). Storage is in the form of binary large objects (BLOBs). There are no provisions for querying the content of the BLOB. Thus, one can save and retrieve an image, but cannot query the database for all images that match a certain color.

Object Database

There is no *official* standard for object databases. The de facto standard is contained in a book published

by Morgan Kaufmann, *The Object Database Standard: ODMG-V2.0*, under the sponsorship of the Object Database Management Group (http://www.odmg.com). See *The Object Database Management Group* (IDC #11892, July 1996) for a discussion of the ODMG and its standard. The emphasis of ODBMSs is on direct (one-to-one, we hope) correspondence between the following:

- The objects and object relationships within an application written in object-oriented languages
- The storage of those in the database

Data model -- Object databases employ a data model that has object-oriented aspects like class, with attributes and methods and integrity constraints; provides object identifiers (OIDs) for any persistent instance of a class; supports encapsulation (data and methods); multiple inheritance; and supports abstract data types.

Object databases combine the elements of object orientation and object-oriented programming languages with database capabilities. They provide more than persistent storage of programming language objects. Object databases extend the functionality of object programming languages (e.g., C++, Smalltalk, Java) to provide full-featured database programming capability. The result is a high level of congruence between the data model for the application and the data model of the database, resulting in less code (25% to 35% less), more natural data structures, and better maintainability and reusability of code. C++, Java, and Smalltalk programmers can write complete database applications with a modest amount of additional effort.

The ODMG-93 object model specifies two types of what it calls denotable objects: objects and literals. It further defines two types of characteristics of objects: operations and properties. Properties can be either attributes or relationships. Object databases can efficiently support any type of structure, including trees and composite objects.

Query language -- An object-oriented language (C++, Smalltalk, Java) is the language for both the application and the database. It provides a very direct relationship between the application object and the stored object. Data definition, manipulation, and query reflect this relationship. ODBMSs have been integrated with C++, C, Smalltalk, Java, and LISP.

The ODMG-93 also defines a declarative language, OQL, for query of database objects. OQL is not 100% compliant with SQL (something the SQL3 developers are attempting to do with complete backwards compatibility with SQL2 and possibly OQL compatibility). SQL2 lacks the type concept of object technology. The result of an OQL query can be an atom, a structure, a literal, one object, or a set of objects. Most object databases also support SQL, primarily in the context of ODBC.

Computational model -- In the RDBMS the query language is the means to create, access, and update objects, but in an ODBMS, although declarative queries are still possible, the primary interface for creating and modifying objects is directly via the object language (C++, Java, Smalltalk) using the native language syntax. Moreover, every object in the system is automatically given an identifier (OID) that is

unique and immutable during the object's life. One object can contain an OID that logically references, or points to, another object. These references prove valuable when associating objects with real-world entities, such as products or customers or business processes; they also form the basis of features such as bidirectional relationships, versioning, composite objects, and distribution. In most ODBMSs, the OIDs become physical (the logical identifier is converted to pointers to specific memory addresses) once the data is loaded into memory (cached) for use by the object-oriented application. This conversion of references to memory pointers, sometimes called pointer swizzling, allows access to cached objects at native memory speeds (hundredths of microseconds) instead of the traditional approach of messages to the server, which take milliseconds.

Architecture Differences Between RDB and ODB

Primarily, RDBMSs have been built around central server architectures, which are much the same as mainframe architectures. ODBMSs often assume a network of computers, with processing on the back or front end, as well as intermediate tiers, caching on each level, and clustering capabilities independent of type. In terms of computation model, although RDBMSs typically confine all processing to the SQL language and its operations (SELECT/PROJECT/JOIN and INSERT/UPDATE/DELETE), ODBMSs allow the use of host object languages like C++, Java, and Smalltalk directly on the objects "in the database"; that is, instead of translating back and forth between application language structures (COBOL, C, etc.) and database structures (SQL), application programmers can simply use the object language to create and access objects through the methods. The database system maintains the persistence, integrity, recoverability, and concurrency of those same objects.

Object Relational

Extended relational and object relational are synonyms for database management products that try to unify aspects of both the relational and object databases. (Note: There is no official definition of what an object relational database management system is.) Won Kim, founder of UniSQL, recently published a white paper describing a framework with which to evaluate the completeness of a product's compliance with seven major categories of capabilities of object relational databases.

Although this white paper is interesting and well written, the criteria it describes is not yet widely accepted. Morgan Kaufman Publishers (San Francisco, California) recently published a book, *Object-Relational DBMSs, The Next Great Wave*, written by Michael Stonebraker, which also sets out a very lengthy discussion on what defines an object relational database product. Stonebraker uses Illustra (now Informix) to illustrate his points. He discusses various options for vendors that want to migrate their products into this category.

The vendors that have products in this space include Informix, IBM, Hewlett-Packard, Unisys, Oracle, and UniSQL. For most of them, "extended relational" is probably the more appropriate term. SQL is a given. ORDMSs will be specified by the extensions to the SQL standard, SQL3/4, which is now under development. It will be two or more years, however, before SQL3 is approved and implemented. In the

meantime, the ORDBMS vendors are anticipating these extensions in their current products, which are therefore proprietary. Thus, this category of database product is really a no-man's land between object and relational. At least one ORDMS, UniSQL/X, has client-side caching, making it more like an object database, but it still is SQL centric.

Data model -- ORDBMS employ a data model that -- to quote the SQL3 standard -- attempts to "add OO-ness to tables." All persistent (database) information is still in tables, but some of the tabular entries can have richer data structure, termed abstract data types (ADTs). An ADT is a data type that is constructed by combining basic alphanumeric data types. The support for abstract data types is attractive because the operations and functions associated with the new data type can be used to index, store, and retrieve records based on the content of the new (e.g., multimedia) data type. Per Won Kim's definition, an ORDBMS should be a superset of an RDBMS, and default to SQL2 (or its successors) if no object capabilities are used. This approach, though it provides more structures for modeling more complex data and free-standing functions, lacks the fundamental object requirement of encapsulation of operations with data. It also has limited support for relationships, identity, inheritance, polymorphism, composition, extensibility or creating user-defined persistent classes, and integration with host object languages like Java.

Query language -- An ORDBMS supports an extended form of SQL, sometimes referred to as an ObjectSQL. The point of the extensions is to support the object model (e.g., queries involving object attributes). Typical extensions include queries involving nested objects, set-valued attributes, inclusion of methods/functions in search predicates, and queries involving abstract data types. The ORDBMS is still relational because the data is stored in tables of rows and columns, and SQL, with the extensions mentioned, is the language for data definition, manipulation, and query. The target and result of a query are still tables or tuples, although SQL4 will generalize this to collections of objects, incorporating OQL, or close to it, for its read-only ad hoc query. The key here is searchable content.

Computational model -- The SQL language, with extensions for accessing ADTs, is still the primary interface to the database. The direct support of host object languages and their objects is missing, forcing programmers to continue to translate between objects and tables.

Unifying Object and Relational

Vendors supplying ORDBMSs, such as UniSQL, Informix, and IBM, would like to have us believe that they have unified the two data models into single products that integrate processing in a highly efficient way. This raises many questions about the depth of the merge. At a superficial level, a unifying interface could be placed over two different storage managers -- one for the relational data model and one for the object data model. This is similar to the federated database mentioned in the next paragraph. A more comprehensive solution would extend the integration well down into the database storage engine itself, which is no mean task. This affects many important aspects of the DBMS, such as query optimization, performance, and scalability. The major RDBMS vendors know a great deal about how to optimize a traditional RDMS, but their investment into these areas is confined to table-based data structures. Much

less is known about how to do it when a data blade is added, for image processing, for example.

Gateways and Multidatabase (Federated) Systems

Enterprise data rarely resides in only one database or one type of database; therefore, most vendors provide ways to integrate processing against multiple, heterogeneous databases. This process may involve multiple vendors of one data model (e.g., relational) or mixed (e.g., relational and object) data models. In the latter case, the primary support is in the data model support by the vendor's own database product. If the vendor can provide a fairly transparent view of the integrated databases, especially for transaction processing, then the integration is often referred to as a multidatabase system or federated database (not to be confused with distributed or replicated databases). If the support is less transparent, then the connectivity is more in the gateway category. Such products fall in the category of middleware.

Why Should Anyone Care?

Ignoring gateways for the moment, why should anyone care which category of database product they are using? How can we position one against another? We will use a few key dimensions to focus our discussion on this question. The relative importance of each of the following dimensions will vary with the application(s) that must use the database(s):

- Support for object-oriented programming and languages
- Simplicity of use
- Simplicity of development
- Extensibility
- Complex data relationships
- Interoperability and middleware
- Performance versus safety
- Distribution, replication, and federated databases
- Scalability
- Product maturity
- Legacy people, programs, and databases, and the universality of SQL
- Software ecosystems
- Vendor viability

Table 2 summarizes some of the following discussions of these dimensions.

Support for Object-Oriented Programming and Languages

ODBMSs have been optimized to directly support object-oriented applications. All OO concepts are supported (e.g., encapsulation, inheritance, polymorphism). The ODBMS model is congruent with the object model in the running application. Because relationships are directly represented, referential

integrity is inherently maintained by the ODBMS. The object model used by the Object Database Management Group in its ODMG-93 standard is derived from the OMG Common Object Model, and the model has been supplemented with bindings for C++, Smalltalk, and Java as well as Object Definition Language (ODL, based on OMG's IDL), the aforementioned OQL, meta-object access, and object interchange.

The SQL3 object model is its own definition, a compromise that adds some object support while maintaining backward compatibility with SQL2. ORDBMSs have limited support for inheritance, with no consistent definition between vendors or with respect to SQL3 (e.g., constructed types). For all vendors, base types are abstract types, supporting inheritance of properties and functions. There is no such agreement for constructed types to be abstract types.

Table 2

A Comparison of Database Management Systems

Criteria	RDBMS	ORDBMS	ODBMS
Defining standard	SQL2 (ANSI X3H2)	SQL3/4 (in process)	ODMG-V2.0
Support for object-oriented programming	Poor; programmers spend 25% of coding time mapping the program object to the database	Limited mostly to new data types	Direct and extensive
Simplicity of use	Table structures easy to understand; many end-user tools available	Same as RDBMS, with some confusing extensions	OK for programmers; some SQL access for end users
Simplicity of development	Provides independence of data from application, good for simple relationships	Provides independence of data from application, good for simple relationships	Objects are a natural way to model; can accommodate a wide variety of types and relationships
Extensibility and content	None	Limited mostly to new data types	Can handle arbitrary complexity; users can write methods and on any structure
Complex data relationships	Difficult to model	Difficult to model	Can handle arbitrary complexity; users can write methods and on any structure

Performance versus interoperability	Level of safety varies with vendor, must be traded off; achieving both requires extensive testing	Level of safety varies with vendor, must be traded off; achieving both requires extensive testing	Level of safety varies with vendor; most ODBMSs allow programmers to extend DBMS functionality by defining new classes
Distribution, replication, and federated databases	Extensive	Extensive	Varies with vendor; a few provide extensive support
Product maturity	Very mature	Immature; extensions are new, are still being defined, and are relatively unproven	Relatively mature
Legacy people and the universality of SQL	Extensive supply of tools and trained developers	Can take advantages of RDBMS tools and developers	SQL accommodated, but intended for object-oriented programmers.
Software ecosystems	Provided by major RDBMS companies	Provided by major RDBMS companies	ODBMS vendors beginning to emulate RDBMS vendors, but none offers large markets to other ISVs
Vendor viability	Expected for the major established RDBMS vendors	Expected for the major RDBMS vendors; UniSQL is struggling	Less of an issue than it was; some shakeout still expected
Source: International Data Corporation, 1997			

For an object-oriented application involving complex data relationships to work with a relational or object relational database, code must be written to map the complex object structure into the simple storage model of the relational database. The analogy (attributed to Ester Dyson) would be the requirement of disassembling your car before storing it in your garage each night and then reassembling it before using it each morning. This has both development productivity and runtime performance issues. This coding effort can amount to 25% of the development effort; thus, the savings are substantial. Application maintenance is also easier. At runtime, similar performance penalties are paid moving data between the database and the running application. Because this mapping from database to application structures is left to the application, rather than the DBMS, it opens up the risk of integrity violation. Different applications might map differently, creating inconsistencies, and end-user GUI tools bypass any such mapping, going directly to the primitives stored in tables, missing any encapsulated operations of the objects.

Simplicity of Use

Relational databases, with their tabular, row-and-column metaphor, provide a simple, easy-to-learn user interface. However, we should quickly note that very few users actually use SQL directly. The relational database vendors and their partners have provided a plethora of tools, mostly forms based, that hide the SQL from the user and generate it in the background. Tools of this type are much less prevalent with the object database products, although we expect this to improve over time.

This model of tables with simple data is easy to use, but only if it maps well to the application's data structures. If the application's structures are complex, the mapping to tables is like forcing a round peg into a square hole. Moreover, this traditional approach has led to a specialization of "database programmers" who are experts in translating back and forth from tables to application structures, who understand SQL, and who effectively limit the accessibility of DBMS technology to a small minority of application programmers. Instead, object programmers find it simpler to directly use objects without having to force them into tables. All programmers today are being trained in object programming, which opens up the use of database technology to a much broader base of programmers.

Simplicity of Development

Object databases are a natural for organizations that have embraced object-oriented programming. As noted earlier in this bulletin, object databases were developed to provide direct correspondence between the application object and the stored object; development is very efficient. Object definition is done only once. In the ODMG specification, for example, object persistence can be provided simply by declaring an application class "persistent capable," and then object instances may be either persistent or transient, but are otherwise the same.

The object-oriented development paradigm has the advantage of being a more natural way to model the "real world" of the application domain. Objects in the real world are modeled directly, one for one, in the software. The object referred to by the end user in the statement of requirements can be the same object represented in the model of the application in an analysis and design tool (e.g., Rational Rose), in the software code, and -- if an ODBMSs is used -- in the database. This one-to-one correspondence greatly simplifies development throughout the application development life cycle and afterwards during ongoing maintenance. It also simplifies the integration of different vendors' tools across the phases of the software development life cycle.

The RDBMS model does provide independence of the data from applications that use the data. With SQL as a query language, any application can access and use data in an independent fashion. This procedure reduces maintenance costs, because each new application does not require any modification of the database -- unless the new application has additional, unanticipated requirements.

Extensibility and Content

The traditional use of DBMSs has been limited by the rather minimal set of data types (string, integer, floating-point number, date, currency) supported by the mainstream data management systems. For

ORDBMSs, extensibility means that the ORDBMS has provisions for the user (actually the user's programmers) to define and support new data types. Initially, it may mean the vendor, or its value-added-resellers, supply the extended support (e.g., for multimedia data types such as image, audio, and video).

One issue concerns who (vendors versus VARs or other ISVs versus users) creates and qualifies the extensions. IBM, for the time being, is retaining all control over any type extensions to DB/2. Informix allows both VAR, other ISVs, and users, but they must work closely with Informix as they add the extension to ensure appropriate optimization and integration into the kernel.

In all cases for ORDBMSs, there are a number of constraints developers need to be aware of when considering user-defined types. For example, constructed types may not be allowed to be abstract types directly (they may have to be values of attributes in an abstract base type). References (i.e., direct references to system-created IDs) can be used to reference a row in a table, but the table's row type must be named. However, such types cannot be abstract types with user-defined functions. Thus, there is a trade-off between taking advantage of fast referencing for high performance versus the extensibility of user-defined types.

Object database management systems are inherently extensible because the classes defined in the application can optionally be definitions of new persistent types. New classes may be added by users, with full DBMS functionality for persistent instances of those classes.

The bottom line is that an ODBMS can handle arbitrary complexity. Users can write methods --on any structure.

Complex Data Relationships

The relational and object relational vendors like to talk about complex data. What they mean primarily is new data types like multimedia types. However, an important distinction for the ODBMS vendors is the notion of complex data relationships -- something the ORDBMS vendors don't want to discuss as much. Applications, especially complex applications, sometimes have many-to-many relationships. An example would be in hospital administration. A patient can have many doctors; a doctor can have many patients. Ditto for procedures, operating rooms, other facilities, and other staff. University administration is similar. Some applications have hierarchical relationships, such as the assembly hierarchies in a manufacturing resource planning system. These are some examples of complex data relationships.

Because of the use of OIDs in object databases, they are much more suitable for applications that have to process complex relationships. This is because of the ability to navigate by following references, bidirectional many-to-many relationships, and propagation of operations across relationships to form composite objects. ORDBMSs have little support for these capabilities; relational has none. Modeling relationships may require additional tables in the RDBMS. Processing usually involves multiple, time-consuming joins.

As an illustration, the following is a quote from a Versant press release concerning an Air France reservation and seat-tracking application. "It's not the complexity of the data, but the complexity of the interrelationships between data points that the solution had to manage," said Steve Clampett, SDT's senior vice president. "Using a relational database, a single query could require 30 or more joins. Even Oracle was an order of magnitude slower than Versant in our tests of this application."

Performance tests at Air France reveal that the Availability Processor application performs at a level consistent with or exceeding industry requirements. For example, sell transaction traffic for a recent day reached more than 207,000 transactions. Of that number, 99.48% of those transactions were performed in less than 30 milliseconds.

JASMINE MAKES CHILD'S PLAY OF TOYS R US MULTIMEDIA PROJECT

User-Friendly Development Environment Allows Marketing Personnel to Participate in Object-Oriented Development

Toys have traditionally been among the most difficult products to depict in paper catalogues. Static photos fail to capture the "fun quotient" of modern toys, which often include sophisticated electronics, moving parts, and eye-catching special effects. Realizing this, Linea GIG S.P.A., whose main partner is Toys R Us, decided to build a sophisticated electronic catalogue that would display the entire GIG product line using multimedia presentations.

"Have you ever watched a child play with a favorite toy?" asked Dott. Giorgio Pezzin, Managing Director of GIG. "In building our electronic catalogue, we wanted to capture that same sense of wonder and delight."

Supplying children with toys is a rewarding business, but it's not all fun and games. To build the multimedia solution, GIG chose CA's Jasmine, a pure object database that is seamlessly integrated with an intuitive development system and deployment environment for both Web and client/server systems.

Jasmine is a complete object database and multimedia development environment that includes all of the necessary tools for companies to build and deliver next-generation business systems. Its ease of use is enhanced by an integrated graphical user interface (GUI) and a drag-and-drop environment for developing applications and managing databases. Jasmine Studio developers-called "composers"-drag objects from the database onto a "scene," then add complex behavior to the objects simply by pointing and clicking with the mouse.

"Jasmine Studio makes it easy to create multimedia scenes in an intuitive way," said Dott. Giorgio Pezzin. "It gives developers a high-level, visual environment for working with objects and defining their

properties and behaviors. Traditional programming languages require more time and expertise to achieve similar results."

Jasmine Studio has such a low learning curve that GIG was able to involve its marketing and design specialists in the development of the multimedia catalogue. "Jasmine allowed us to put our creative marketing people on the initiative, right along side of our programmers and technologists," said Dott. Giorgio Pezzin. "As a result, the finished product more accurately reflects the look and feel we are after."

GIG composers quickly built an object model, then used the Jasmine Studio environment to define object classes, properties and methods. The entire development effort required less than three months. Dott. Giorgio Pezzin attributes this fast development cycle to Jasmine's object-oriented approach. "By storing information as objects, we can compose next-generation applications very quickly, maximizing object application code reuse," he said. "We have created a great deal of multimedia information and established complex relationships among business objects. Jasmine's object database manages this kind of information much better than a relational or hybrid database can."

Jasmine is based on a pure object model, not merely a relational model that has been extended to support complex data types. Furthermore, Jasmine's "Create Once, Deploy Everywhere" architecture gives developers the ability to roll out the electronic catalogue on many different computing platforms, including Intranet, Internet, CD/ROM and client/server systems. "Jasmine lets us take full advantage of object technology and the Internet," he added.

The finished electronic catalogue is a marvel of sleek Italian design, allowing GIG sales representatives around the world to demonstrate thousands of toys in brilliant multimedia detail. Full-motion video, photographs, sound, and animation are interwoven with textual information to describe the toys and make them come alive. Currently deployed on a secure Intranet at GIG's Florence office, the catalogue is tied to an online ordering system that gives customers direct access to order status, stock supplies, prices, and other important information. Because the catalogue sports a graphical user interface, sales reps say it is intuitive and does not require the extensive training often associated with other new software systems.

Based on the success of the electronic catalogue, GIG is now establishing the design for a number of additional Jasmine systems, primarily specialty catalogues and electronic commerce applications.

Technically, it's impressive, but Mr. Pezzin tends to emphasize the intangible benefits--like ensuring GIG can continue to spark the boundless creative energy of children. "All of our products must be of the highest quality and have substantial educational and play value," he said. "We build systems to help children grow, and CA builds systems to help us grow. The power and flexibility of the Jasmine solution has made our work a lot more effective--and a lot more fun."

Performance Versus Interoperability

Introducing extensions in a loosely coupled product architecture, with clearly defined interfaces facilitates introducing new capabilities. Oracle's Network Computing Architecture, with its data cartridges, fits in this context. This product architecture also sets the stage for support for a more universal data access and support for multitiered application architecture. This proprietary approach implies more of a vendor lock-in for the user.

Isolating the software associated with supporting additional data types from the more fundamental, underlying database engine functionality decreases the chance of the new extensions introducing a bug that will compromise the engine (e.g., the storage engine). This DBMS product design approach is safer than one that is tightly integrated. The trade-off is performance for safety. Separate processes probably means context switching, which hurts performance, especially in certain hardware architectures, such as those that are pipelined.

The data cartridge (Oracle) or data blades (Informix) approaches require modifications to the DBMS kernel. This task is not for the fainthearted; there is an associated danger of corruption and this approach still forces the context-switching performance overhead in communicating between applications and this DBMS-resident code. Instead, ODBMSs allow any application programmers to extend the DBMS functionality by defining any new classes, including structures, operations, and relationships, with such code executing either on the back end or front end for optimization of caching. Different ODBMSs provide different levels of safety, or protection of the DBMS itself from application bugs.

Distribution, Replication, and Federated Databases

A distributed database transparently stores its data across multiple volumes and even different locations. A replicated database has all or portions of its data replicated at one or more different sites and periodically synchronizes the contents of the replicated data. Data replication is the foundation for data warehousing. A federated database integrates isolated, heterogeneous databases into a single virtual schema for use by applications, including transaction processing. Such use does not preclude the use of any one database in its normal operational context.

Although the RDBMS approach extends servers to communicate with other servers and can support some level of replication, especially for read-only replicates, ODBMSs such as Objectivity/DB, provide forms of distribution and replication beyond that of the RDBMS world, with complete transparency across multiple servers and databases in a federation, multiple schemas in a hierarchy, and replication with automatic synchronization and integrity. In addition, ODBMSs support features not typically present in the RDBMSs, such as dynamic schema evolution with automatic instance migration, versioning, and long transactions.

Product Maturity

Relational database products have been under development and used much longer than object database products. The RDBMSs are more mature products. The more mature products have been fine-tuned for

optimized performance (albeit on a very limited set of data types) and provide a very rich set of functionality, including support of advanced features like parallel processing, replication, high availability, security, and distribution.

There are a wide variety of tools and applications that support the RDBMSs and work with SQL. Ostensibly, the ORDBMSs should be able to take advantage of this support because they are extensions of the RDBMSs their vendors have been marketing for years. However, the experience and investment in RDBMS products have all been built around tables. The newer support for abstract data types requires new capabilities in the engines for the basic DBMS capabilities such as recovery, concurrency, indexing, and caching; thus, these data types do not automatically benefit from the RDBMS maturity.

We expect that the completion and implementation of SQL3/4 ADT extensions to take years. In the meantime, the ODBMS products are now also maturing, some with nearly a decade of experience in production applications and often with more advanced functionality than their RDBMS competitors. In addition, it will be difficult for ORDMSs to be both extensible and retain their legacy advantages, especially highly optimized performance because indexing -- specifically B-tree support -- must be made generic. In the legacy RDBMSs, B-tree indexing is highly tuned to a limited set of types.

Ultimately, as ORDBMSs evolve to support more of the ODBMS-like capabilities, they will do so based on new extensions to the products, thereby becoming the new, untried products compared with the ODBMS that are already mature in those areas, with DBMS engines natively designed for objects. Furthermore, although the ORDBMSs struggle with and try to extend inherited architectures and product implementations that assume only the tabular relational model, ODBMSs benefit from foundations built directly to support objects.

Legacy People and the Universality of SQL

The other advantage that RDBMSs and the SQL-based ORDBMSs have is the availability of experienced developers and the plethora of SQL-based developer tools, books, and consultants. ODBMSs won't be in a similar position for another five years. SQL is the most universal database language. As a result of the investments made by organizations during the last 15 years, most developers are familiar with SQL and have SGL tools with which to develop. Accommodating SQL or related access mechanisms like ODBC minimizes the cost of adoption of proposed extensions or new database capabilities.

Recognizing this fact, most ODBMS products now support at least some of SQL, and some have supported full SQL and ODBC for several years now, including extensions to support object queries using methods, relationships, inheritance, and others. This allows RDBMS-trained users to start using ODBMSs via familiar SQL and GUI tools, share the same objects created in Java or C++, and gradually learn to take more advantage of the additional object capabilities.

Software Ecosystem

Product architectures have a direct impact on the vendor's channel strategy. An architecture that is more open, possibly with published APIs, and portable across multiple platforms, sets the stage for other potential partners to expand the overall market for the database vendor's product line. Conversely, a vendor's desire to prevent outside partners from creating extensions to its products can severely limit the vendor's channel strategy. The more open the database vendor is, the more likely it is to create a *software ecosystem* that is founded on its DBMS. Although the vendor must incur additional costs (and technical risks) to support such an ecosystem, such partners can work synergistically to create a market that is larger and expands more rapidly. The issue is control -- control over the product definition and control over the evolution of the total solution provided in the context of the overall ecosystem.

Taking a lead from its Illustra acquisition, Informix is busy creating a software ecosystem around its Universal Server. Dozens of vendors are writing data blades -- some minor, some major -- for use with the Informix Universal Server, thereby extending the apparent range of capabilities available to developers. IBM has taken a more cautious approach, choosing to be the only provider of DB2 extensions for now.

Although the object database vendors, in general, are in a much better technical position to support the type of extensibility required to build an ecosystem, they can't deliver much market size. For example, Object Design Inc. (ODI) is in a similar position to Informix; ODI added data blade-like support to its ODBMS, ObjectStore. Some of ODI's partners are the same as Informix's partners (e.g., Virage for image processing). The difference is that Virage's own software, written in C++, was interfaced with ObjectStore in a matter of days instead of the months it took with Informix. The native object class support of all ODBMSs allows direct support of a much broader set of object functionality, without requiring database specialists to modify the kernel for each such addition.

Vendor Viability

For certain applications, ODBMSs are clearly superior to RDBMSs or their extended cousins, the ORDBMSs. Despite this clear advantage, some user organizations, and even ISVs, feel safer using a relational database in such applications because the RDBMS vendors are very large and have a higher probability of being around for the long haul. With the exception of Computer Associates, all the ODBMS vendors are small -- a fact that early adopters of ODBMSs were willing to accept because they were trading off risk against significant (orders of magnitude) benefits. The ODBMS vendors realized early on that this issue was an inhibitor to adoption. Therefore, they formed the Object Database Management Group and published a standard for ODBMSs to ensure a degree of portability between vendors for applications that used an ODBMS.

Nevertheless, the more risk-averse users continue to trial new applications with complex data models on top of RDBMSs. They are actually fooling themselves. Moreover, the RDBMS vendors are fooling the public. Putting object extensions on RDBMSs is tantamount to adding stereo radios and global navigation systems to horse-drawn carriages. You will have interesting enhancements, but the wrong base vehicle. In the end, it won't be the appropriate vehicle for the information superhighway. Among the

larger traditional database companies, only Computer Associates, seems to understand this. The native object class support of all ODBMSs allows direct support of a much broader set of object functionality, without requiring database specialists to modify the kernel for each such addition. End users who misapply RDBMS technology, ultimately run the risk of wasting time developing solutions that won't scale and will perform too slowly.

Conclusion

The ORDBMS vendors are much larger and have huge entrenched marketing infrastructure. By comparison, the ODBMS vendors are much smaller. For this reason alone, IDC believes that during the next four years, the ORDBMS market is expected to become larger than the ODBMS market, which itself is growing at a respectable rate. Users will make choices of database vendors based on many criteria, some of which have been addressed here. It is clear that in today's complex, rapidly changing world, ODBMSs provide the more flexible, extensible alternative for companies that must act quickly to match the capabilities of their information systems with the needs of their organizations.

Topics: ODBC, Smalltalk, Relational Database Management Systems, Lisp, SQL, Object Database Management Systems, C++, Java, Object Relational Database Management System, BLOB
Companies: UniSQL, OMG, Object Design, Unisys, Ingres, Oracle, IBM, Computer Associates, Hewlett-Packard, Informix Software
Document Number: 14821E
Published under Services: Government: Application Development, Object Tools
Note: For related research please search on the above listed topics.

Table of Contents - **Abstract** - **Document**

database relational-database object-oriented-database.Â The relational model is more powerful than object models in terms of the kinds of questions you can ask of your data. Unfortunately, SQL threw out much of the expressive power that the relational model is capable of, but even in this diluted form, it is still easier to express queries in SQL than in a typical OO database (be it ORM or OODBMS). Queries in an OODBMS are predominantly driven by navigational operators, which means that if your sales database has sales people owning their sales, then querying for the monthly sales for a given SKU is not only likely to be cripplingly slow, but very Relational Database vs Object Oriented Database. Summary: Difference Between Relational Database and Object Oriented Database is that relational database is a database that stores data in tables that consist of rows and columns. Each row has a primary key and each column has a unique name. A file processing environment uses the terms file, record, and field to represent data.Â Object-oriented databases have several advantages compared with relational databases: they can store more types of data, access this data faster, and allow programmers to reuse objects. An object-oriented database stores unstructured data more efficiently than a relational database. Unstructured data includes photos, video clips, audio clips, and documents. Comparison of object database management systems. From Wikipedia, the free encyclopedia. This is a comparison of notable object database management systems, showing what fundamental object database features are implemented natively. Name. Current Stable Version.Â âœª 09- Object Oriented Database Model In Database Management System In HINDI - Overview Of Data Models. âœª hierarchical vs Network vs Relational data model/DBMS. âœª Object Relational Database Model. Transcription. Benchmarks. With an object database, data can be managed as objects with methods and attributes. But what exactly are object-oriented databases?Â Relational vs. object-oriented databases. For a long time, relational databases have been the standard in web and software development. In this model, information is stored in interconnected tables. Here, too, the links between complex pieces of information with different components can be stored and retrieved. With an object database, though, all of the unitâ€™s components are available immediately. That also means that the data sets can be much more complex. With a relational database we tend to try to accommodate simple information. OBJECT ORIENTED DATABASE Object oriented databases are also called Object Database Management Systems (ODBMS). Object databases store objects rather than data such as integers, strings or real numbers. Objects are used in object oriented languages such as Smalltalk, C++, Java, and others.Â ODBMS vs. Relational. J. Object-Oriented Programming Focus on ODBMS, pp: 35. [9]Brodie, M., J. Mylopoulos and J. Schmidt, 1985.On Conceptual Modeling.