CIS 315 - Reading Packet: "A few words on OODBMS's and ORDBMS's"

Sources:

* C. Schahczenski, "Object-Oriented Databases in our Curricula", in "The Journal of Computing in Small Colleges", Volume 16, Number 1, November 2000, including "The Proceedings of the Ninth Annual CCSC Rocky Mountain Conference", October 20-21, 2000, Utah Valley State College, Orem, Utah; pp. 170-176.
* "The Object-Oriented Database System Manifesto", Atkinson, Bancilhon, DeWitt, Dittrich, Maier, Zdonick, Proceedings of the First International Conference on Deductive and Object-Oriented Databases, pages 223-40, Kyoto, Japan, December 1989.
    * URL for a version on the web [confirmed was working, 11-29-2010]

      http://www.cs.cmu.edu/afs/cs.cmu.edu/user/clamen/OODBMS/Manifesto/
* Chapter 11, pp. 417-474, “Obect-Oriented Databases”, in Database Systems: Design, Implementation, and Management, by Rob, Coronel, Wadsworth Publishing, 1993.
* Chapter 15, pp.425-438, “Object-oriented Databases”, in Database Analysis and Design, 2nd Edition, by Hawryszkiewycz, MacMillan Publishing, 1991.
* sources that are no longer web-accessible:
    * [Haines] severall web pages from Jason Haines' thesis (from a University in Australia...?), at http://demos.anu.edu.au/jason/thesis/thesis/WWW/node16.html (node17,node18); (no longer a good web address)
    * [Feeley] a web page by Mike Feeley, U. Washington, http://www.cs.washington.edu/homes/feeley/generals/DataManagement/node8.html (no longer a good web address)
    * [ObjectStore] part of a review of ObjectStore, another OODBMS, from web page http://www.eu.sun.com/sunworldonline/asm-04-1995/asm-04-lead.rev.html (no longer a good web address)
    * some “Objectivity” pages from the web; (Objectivity must be one OODBMS...)! (no longer can find them on the web)

A few words on OODBMS's and ORDBMS's

First:

* OODBMS - object-oriented database management system
* ORDBMS - object-relational database management system

Note that object-oriented and objest-relational DBMS's are still an area IN FLUX -- people/companies are still defining what it really means for a DBMS to be an OODBMS or an ORDBMS.

IN THEORY... an OODBMS maintains/sets up a database of OBJECTS (in the Java/C++ sense: you define a class, instances of that class are objects). Objects, then, are what you should recall from your first object-oriented programming class, whether it was in C++ or in Java: objects are data (data fields) PLUS methods (how to act on that data). And, you should also recall that classes can be declared as subclasses of other classes, and a class that is a subclass of another class inherits methods and data fields from that superclass. (That is, if class A is a subclass of class B, then A inherits methods and data fields from class B.)

What does it mean, exactly, to have an object-oriented DBMS, then? Be careful -- there really isn't a standard definition for this yet, although several groups are working on coming up with such a definition.

Why even bother with this at all? A lot depends on your perspective, and where you are coming from on this. For example, if you have a large store of object-oriented data that you would like to be persistent between runs of programs in a more convenient format than via files, the idea of an OODBMS could be quite appealing (in the same way that early DBMS's looked like a step up from files in the early days of databases). An OODBMS might also make it easier to create your own data types than is the case in a typical RDBMS, which might be appealing if you need more data type flexibility (rather than just using the data types provided by the DBMS for columns).

There are issues in OODBMS's that are sticky, however. We know that, at least in concept, objects are data PLUS methods. Would a OODBMS then store an object's methods along with its data in the OODBMS? Should it? If so, how?

Some of the aspects of RDBMS's (that we perhaps are so used to that we take them for granted) are not necessarily the case for OODBMS's. Remember how one of the big advantages of RDBMS's is program/data independence (the data are not dependent on the application program USING the data)? So that, when you change the form of the data (add a column, perhaps), you DON'T have to change the application programs, because the RDBMS is hiding the file storage details from the application program? Well -- what happens with this when methods are PART of the data? Have we lost some of the benefit of RDBMS's that came from deliberately separating the program and the data? (The issue here is that program and data are deliberately linked in an object-oriented program!)

Object-Relational DBMS's try to combine object-oriented and relational database management systems trying to get the best of both worlds. But it appears that, here too, the definition of ORDBMS is not completely "settled" yet. One might call something an ORDBMS just because the user can define his/her own data types, but otherwise he/she uses relations whose attributes can happen to have that new data type for its rows. You would be wise to dig deeper to see just what is intended if a vendor makes the claim that a DBMS is either an OODBMS or an ORDBMS.

Another interesting difference between both OODBMS's and ORDBMS's and RDBMS's is that OODBMS's and ORDBMS's are not based on an underlying mathematical model like RDBMS's are; there isn't an analogy for relational algebra here.

One interesting possibility for an OODBMS is that it might actually be part of an object-oriented language, rather than a separate, independent component. It might be a seamless part of a C++ or a Java or an object-oriented language -- you might just ask to save some object (within your C++ or Java or object-oriented code), and it would simply get stored within an OODBMS automatically. That is, the OODBMS would be part of the application language development environment; obviously, in this approach, this is inherently linking application and data. Data would be related via their object inheritance hierarchy in an otherwise "flat" object-store.

Some potential **advantages** of OODBMS's:

* might be integrated with an object-oriented programming language (as described above); in that kind of approach, you would only need to learn that one language, you wouldn't need a separate, second "database" language (such as SQL)
* might have methods and data stored together automatically (easier than files)
* user-defined types
* more-complex data might be more easily represented (more complex than 1:1, 1:N, N:M)

Some potential **disadvantages** of OODBMS's:

* you might have to involve that object-oriented programming language (used to define the object); or, it might require object-oriented programming
* you've now re-introduced more program/data dependence (reduced or eliminated program/data independence)
* lacks a strong underlying mathematical model
* currently, immature query and reporting tools
* relatively less data is currently in object form
* currently, immature concurrency control and transaction management support;
* currently, unproven performance

"The Object-Oriented Database System Manifesto" (second source listed at the beginning of today's notes) threw down some interesting gauntlets with regard to what that collection of authors thought an OODBMS "should" be, to be able to reasonably call itself an OODBMS. Some of these features are related to the claim that such a system is indeed object-oriented, and some of these features are related to the claim that such a system is indeed a DBMS. (These are the IDEALS as judged by these authors, note!)

(related to actually being a DBMS)

* persistence of data
* secondary storage management (the OODBMS should handle the details of secondary storage management, as a DBMS does)
* concurrency support
* recovery support
* an ad-hoc query facility (that should be high-level, efficient, and application independent)

(related to actually being object-oriented)

support for complex objects: user-defined types, constructors, methods, inheritance

* support for object identity
    * when you want to be able to uniquely identify an object, even if it happens to have the same

values for its data fields as another object;

(do two pointers/references point to/refer to the SAME object in memory, or to two objects with EQUIVALENT data fields? consider (if you happen to be familiar with Java) the difference between == and the `equals` method in Java...)

* support for encapsulation
    * if you are calling something object-oriented, it better support encapsulation (combination) of data and methods;
    * (although, yes, this DOES affect the separation of data and application that we've enjoyed in an RDBMS...)
* support one of the major object-oriented models; (C++/Java style classes or Smalltalk's approach, for example)
* support for inheritance (especially since that is a major way that data would likely be related in such a system)
* support for overriding, overloading, polymorphism (late binding)
* (and the paper goes on from there...)

So, how about an Object-Relational DBMS, an ORDBMS? What would that mean? People are still discussing that as well. Is it as simple as providing user-defined types, and then a relation's attribute can be defined to have that type (as its domain)? Does it mean layering object capabilities on top of a "traditional" RDBMS? But references and pointers are still not consistently handled in such systems, and there is little standardization yet of what it means for something to be an ORDBMS, so read the specifications carefully for anything claiming to be one of these!