SPE
Society of Petroleum Engineers

SPE 24282

Object-Oriented Data Management

Steve Trythall, J.N. Greenlee, and Dawson Martin, Petroleum Science & Technology Inst.

This paper was prepared for presentation at the SPE European Petroleum Computer Conference held in Stavanger, Norway, 25–27 May 1992.

Abstract

It is increasingly recognised that good data management is essential to the petroleum industry. Relational databases are prevalent within the exploration and production industry and are increasingly applied to the storage of exploration data.

In this paper we argue that the need to describe and manage complex exploration data is better served by the object oriented database management system. The OODBMS provides richer data modelling facilities which are important for representing the objects which are common in petroleum exploration, e.g. three dimensional volumes, images, etc. The facilities of the OODBMS will be introduced and contrasted with the relational database management system.

An example of the use of databases in spatial data modelling will be given and the object and relational implementations contrasted. A benchmark figure for the relational database (ORACLE) and the object oriented database (ONTOS) will be given. We will show that the OODBMS is superior in speed, efficiency and data modelling capabilities.

Introduction.

The 1990s are likely to be the decade of the object. A vast number of software vendors are using the tag "object oriented" to describe their products. Software engineering publications are continually advocating the benefits of using object orientation in the analysis, design and construction of our applications. Object oriented operating systems are starting to be released and there is considerable work being carried out on object broker technology.

One more recent development (as far as commercial products are concerned) is the object oriented database management system (OODBMS). This development has come from two groups of researchers: the object oriented programming camp who have seen the need for persistent objects; and the database camp who have realised the benefits that can be gained from object orientation. The user community of OODBMSs is slowly expanding. It has been shown that the object database is very useful where the application demands a complex data model (eg CAD/CAM and knowledge engineering). They are also useful as they remove the impedance mismatch that exists between the application which uses complex data structures and relational databases that only provide tables.

However, a question remains about the applicability of this emerging technology. *Is the hydrocarbon exploration and production industry a domain whose features merit the use of an object oriented database?* In this paper we will introduce the object database and discuss whether we consider it to be a technology appropriate for our industry and present a case study.

What Distinguishes the Object Database?

Database management system have become very prevalent in our industry over the last decade. The object database shares many features with the relational database. These common features are essential for a commercial database management system, i.e.:

Persistence

Disk Management

Concurrency Control

Recovery

Querying Facility

In addition to these features the object database also has a number of distinctive attributes:

Object Identity

A system wide unique key is available for every object. This key is automatically generated by the database system rather than being left to the user to create. This means that it is not necessary for the user to intoduce artificial attributes to the data model such as `triangle_id` (`TID` in fig 1). This unique key also allows us to have object existence independent of value, i.e. two objects can exist with the same value.

User Defined Types

Data models are a representation of the domain we are dealing with. The systems we use provide primitives as a language for representation and the set of primitives provided vary in their semantic richness. The OODB can be viewed as a superset of the relational database as one of its primitives could be rows of tuples in a table. However, the use of user defined types allows us to increase its semantic expressiveness. The user defined types allow us to introduce geological objects such as sedimentary layers; each layer has a lithology which could be an enumerated type. This contrasts with the relational database where everything is represented using a parameterised type i.e. the homogeneous list of flat tuples. Within the tuples the columns can only be made up of primitive types (ints, chars, etc).

Many objects in the real world are formed from a number of sub-objects. The object database allows us to represent composite types. The stratigraphic column object could be represented as an ordered list of heterogeneous objects,e.g., layers, unconformities, hiatuses and faults. One of the strengths of the object database is that the decomposition of an object is not restricted to involving a single object type and it is not necessary to know the type of every sub-object.

Encapsulation

As in object oriented programming, it is possible to specify an interface to the data members of an object. This interface will allow the implementation details of the data members to be hidden from the user.

Inheritance Hierarchies

We are accustomed to the idea of real world objects fitting into natural hierarchies. For example, geological event, unconformity and erosion form a hierarchy with geological event being the most general and erosion being the most specific. The object database allows us to capture the

hierarchical nature of many domains. The objects in the hierarchies of an object database are linked together by inheritance relationships allowing more specific objects to reuse both structure and methods (procedures) from more general objects.

Polymorphism
The object database also allows us to overload procedures and operators so that more specialised object can have different behaviour to their ancestors. The reimplemented version of the method or operator will have the same name as the original.

It is often useful to be able to treat a number of objects as if they were all members of the same class and to be able to ignore whether they are more specialised versions of that class. For example, we may have a dictionary of geological events which contains both depositional layers and erosional unconformities. We may iterate over the dictionary and invoke the backstrip method on each element. The appropriate method will be called for the two types of geological events. In order to carry out this, the database system has to determine at runtime which method should be called. This is known as *late-binding*.

An Appropriate Technology?

We strongly believe that our industry will benefit from the introduction of object oriented technology. This belief results from our experience of the data types that are prevalent within in the industry. We believe that the main benefits will come in the following areas:

Modelling Primitives
Many of the data types that are in common usage within the industry would benefit from the enhanced set of modelling promitives that an OODBMS can provide. The ability to efficiently store images, text, composite structures and highly interconnected spatial data will benefit from this technology.

Navigational Querying
The relational model represents a relationship using elements which are shared between two tables. This approach requires a join operation to be carried out in order to locate the tuple with which a relationship is held. In the object database a relationship can be modelled using a simple attribute containing a reference. This reference can be both a pointer to the object and its unique identifier. The option then exists of carrying out navigational queries by following object pointers. This is "computationally cheaper" than a join as it does not require a search through a table. Many of the data models which are used in our industry would benefit from this approach.

Example Application

In geology one method for representing a three dimensional area or model is as a set of surfaces, one on top of the other. These surfaces can be modelled as a set of connected triangles. Each triangle has three points. Two triangles are said to be connected if they share a point. A point has its coordinate in space and some attributes such as porosity at this coordinate.

Two data structures were built for this example, one using the relational paradigm and the other using the object oriented paradigm. For the relational model we show in fig 1 the structure of the tables and for the object model we show in fig 2 the objects and some of the associated methods.

If we study the first paragraph of this section we can quickly identify the objects required : Model, Surface, Triangle, Point. These are exactly the objects we model in our object database.

With the relational model it is necessary to create tables for each of these objects and one further table is required, a connections table. It is interesting to note the amount of extra data required in the relational model. In order to allow table joins it is necessary to create unique identifiers for each tuple in a table (e.g. MID, TID, SID, PID in fig. 1). In the object model there is no such requirement. It follows that we would expect the object-oriented database to be smaller than the equivalent relational database.

Consider printing out the points of a given triangle using both data models. With the relational model it is necessary to join the Triangles and Points tables using the TID. So the TID of the triangle in question is noted and searched for in the Points table. The tuples in the Points table with a matching TID are then printed. In the object model things are much simpler. The given instance of triangle contains three points. There is no need to search for these points. Printing the points is a simple matter of sending the print message to each of these points by invoking the printPoint() method on each one.

The elimination of the need for joins and so searches in the object-oriented database would suggest that the object-oriented database would be faster than the equivalent relational database. Figures for this difference in speed are generally around a two fold increase but some specialised applications have shown an increase of one to two orders of magnitude.

Looking at the two data models side by side the object data model is more intuitive. It models the real world objects as objects with all their real world attributes and behaviours. The relational model is not as simple to build or understand. It requires additional attributes (unique identifiers) which have no meaning in the real world. It also requires the duplication of data, for example the connections table, which is not required in the object model. In the relational model there is no means of attaching behaviour to a table or its tuples. In the object database it is possible to associate a method with Triangle to tell us if one triangle is connected to another (fig 2. `is_connected(triangle)`) there is no direct way of doing this with the relational database. This would suggest that the object-oriented database is easier to program and use.

Benchmark

We wanted to compare the performance of the relational and the object-oriented databases using the example above. We implemented the relational model using the RDBMS ORACLE and the object-oriented model using the OODBMS ONTOS.

The OODBMS ONTOS is supplied by Ontologic Inc.. It is a multi user, distributed object database with a C++ class library interface. Its fundamental purpose as a database is to provide a reliable persistent storage facility for C++ objects. It allows objects denoted by C++ program variables to have a lifetime that is longer than that of the program that created them and allows C++ programs to retrieve persistent objects into their program variables. It also provides within its class library useful classes such as aggregate classes and the exception handling facilities absent in C++.

ONTOS is relatively easy for a C++ programmer to use. Once a programmer has designed the application, identified the basic class structure and created the C++ class definitions, it is a simple matter of passing these definitions to the ONTOS schema compiler to generate the database schema in an ONTOS database. In an application the ONTOS database is accessed through a C++ interface which ensures that objects are transparently activated (retrieved from) the database server to the C++ client as they are needed.

We populated each of the databases with the same dataset. This consisted of one model containing one surface. The surface was made up of 4802 connected triangles, the connections also being stored in the database.

The task we chose to use as our benchmark was to extract from the database each triangle in the surface and each of its connected triangles. In surface modelling this type of transaction is common, for example it may be required to estimate the average porosity of a triangle by looking at the porosity of each of the connecting triangles.

The timing of this task for each of the databases is given below:

ORACLE	ONTOS
139 sec	69sec

i.e. the ONTOS database was more than twice as fast as the ORACLE database.

Conclusions

We believe that the exploration and production industry would benefit from the use of object oriented technology in a number of application areas. Both the increases in modelling capabilities and performance gained by the use of an object oriented database management system over the existing relational systems is sufficient reason for our industry to reconsider its existing database technologies.

References

1.Jones, C. B., "Data Structures for three-dimensional spatial information systems in geology," International Journal of Geographical Information Systems, Vol. 3, No. 1, pp. 15-31 (1989).

2.Dittrich, K. R., "Object-Oriented Database Systems: The Notions and the Issues," On Object-Oriented Database Systems, Dittrich, K. R. et al, Springer-Verlag, pp. 3-10.

3.Ontologic Inc., Ontos Developer's Guide.

fig 1. relational data model

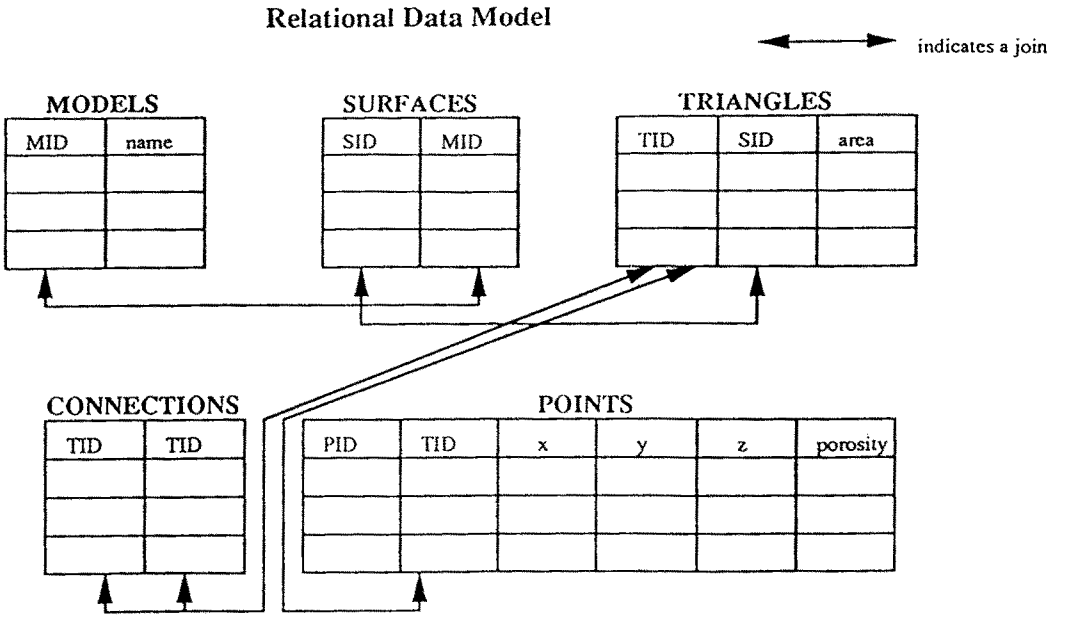

fig 2. object data model