Chapter 16. Object-Oriented Database Systems

Table of contents

Objectives

At the end of this chapter you should be able to:

- Describe the essential characteristics of Object databases.
- Critically assess the strengths and weaknesses of Object databases with respect to Relational systems.
- Describe why Object databases appear to be such a good fit for a number of major growth areas in computing, such as Web-based and multimedia

information systems.

- Describe the strategy being adopted by major database supplier Oracle to address the apparent threat of Object database systems, and critically compare this approach with a pure Object technology approach.

Introduction

In parallel with this chapter, you should read Chapter 25 and Chapter 26 of Thomas Connolly and Carolyn Begg, “Database Systems A Practical Approach to Design, Implementation, and Management”, (5th edn.).

The Object data model provides a richer set of semantics than the Relational model. Most of the major database vendors are extending the Relational model to include some of the mechanisms available in Object databases. These extended Relational databases are often called Object-Relational. In this sense the Object data model can be seen as an enriching of the Relational model, giving a wider range of modelling capabilities. The topics of design, concurrency control, performance tuning and distribution are just as relevant for Object databases as for Relational systems.

Relational database systems have been the mainstay of commercial systems since the 80s. Around about the same time, however, developments in programming languages were giving rise to a new approach to system development. These developments lead to the widespread use of Object technology, and in particular, Object-oriented programming languages such as C++ and Java. Many people expected a similar growth in the commercial use of Object database systems, but these have been relatively slow to be adopted in industry and commerce. In this chapter we will explore the reasons why Object databases have not so far had a major impact in the commercial arena, and examine whether the continuing growth of the World Wide Web and multimedia information systems could lead to a major expansion in the use of Object database technology.

Motivation

The Relational database model has many advantages that make it ideally suited to numerous business applications. Its ability to efficiently handle simple data types, its powerful and highly optimisable standard query language, and its good protection of data from programming errors make it an effective model. However, a number of limitations exist with the model, which have become increasingly clear as more developers and users of database systems seek to extend the application of DBMS technology beyond traditional transaction processing applications, such as order processing, financial applications, stock control, etc.

Among the applications that have proved difficult to support within Relational environments are those involving the storage and manipulation of design data.

Design data is often complex and variable in length, may be highly interrelated, and its actual structure, as well as its values, may evolve rapidly over time, though previous versions may be required to be maintained. This is quite different to the typically fixed-length, slowly evolving data structures which characterise transaction processing applications.

The query languages used to manipulate Relational databases are computationally incomplete; that is, they cannot be used to perform any arbitrary calculation that might be needed. The SQL language standard, and its derivative languages, are essentially limited to Relational Algebra-based operations, providing very little in the way of computational power to handle numerically complex applications.

Further to the problems that have been associated with Relational databases since their inception, a significant problem that has come to light relatively recently is the need to be able to store and manipulate ever more complicated data types, such as video, sound, complex documents, etc. This is putting an increasing strain on the model and restricting the kinds of business solutions that can be provided. One reason for this increase in data complexity is the explosion in popularity of the Internet and Web, where it is necessary to store large quantities of unstructured text, multimedia, images and spatial data.

Other examples of applications that have proved difficult to implement in Relational systems include:

- Geographical information systems
- Applications for processing large and inter-related documents
- Databases to support CASE tools
- Image processing applications

What is Object database technology?

Capturing semantics

Although the Relational model enforces referential integrity, as we saw in the chapter on integrity constraints and database triggers, it has no mechanism for distinguishing and enforcing the different kinds of relationship which may exist between entities. Examples of these relationships include:

- **Existence-dependency:** Where one entity can only exist in relation to another. An example of such a relationship is that of an order-line entity instance, which only makes sense within the context of its corresponding parent order entity.
- **Associations:** When entities of different types are associated with one another; for example, when a car entity is associated with a particular

person through an ‘owns’ relationship.

- **Categorisations:** When a number of different entity types are classified into a particular overall grouping; for example, lecturers, administrators, deans, professors and administrators are all categorised as university employees.

Such distinctions between relationship types can be made in a conceptual entity-relationship model, but not explicitly when mapped to the Relational model. If such distinctions are made, it is possible to define the semantics of operations to create, update and delete instances of relationships differently for each case.

Semantic data models are data models that attempt to capture more of the semantics of the application domain, and are frequently defined as extensions to the Relational model. Such models enable the representation of different types of entity, and the description of different types of relationship between entity types, such as those described above.

Semantic models therefore aim to support a higher level of ‘understanding’ of the data within the system; however, these models do not increase support for the manipulation of data. The extended data structuring mechanisms are accompanied by the same general set of operators (create entity, delete entity and update entity). We would be able to constrain the data structures more naturally if we recognised that the data structures that have been defined are accessed and updated through a fixed set of data-type specific operators. On creating a new entity it is often necessary to carry out a number of checks on other entities before allowing the new entity to be created. It may be necessary to invoke other operations as a consequence of the new entity’s creation. These checks and operations are entity-type specific.

The next stage in semantic data modelling, is the integration of operator definition with the data structuring facilities, such that operator definitions are entity-type specific. The Object-oriented paradigm is one possible way to attempt this integration, by providing a mechanism for progressing from a purely structural model of data towards a more behavioural model, combining facilities for both the representation and manipulation of data within the same model.

Review questions 1

- Describe some of the shortcomings in the Relational approach to database systems, which have lead people to look for alternative database technologies for some applications.
- Identify a further example of each of the three types of relationship mentioned in the text: existence dependency, association and categorisation.

Object-oriented concepts

Combining structure and behaviour

A basic difference between traditional databases and Object databases, is the way in which the passive and active elements of the underlying system are implemented. Traditional databases are seen as passive, storing data which is retrieved by an application, manipulated and then updated on the database. This is in contrast to the active, Object-oriented approach where the manipulation occurs within the database itself. It is also possible to use Object-oriented (OO) databases passively; however, this means that they are not necessarily being used to their full potential.

The inclusion of the behaviour, or processing, related to an object, along with the definition of the structure of the object, stored within the database itself, is what distinguishes the Object-oriented approach from semantic data models, which purely try to improve the level of meaning supported by the data model of the database system. The way in which active behaviour is supported within Object databases, is via the message/method feature.

Messages

If object A in the database wants object B to do something, it sends B a message. The success or failure of the requested operation may be conveyed back from object B to object A, via a further message. In general, each object has a set of messages that it can receive and a set of replies it can send. An object does not need to know anything about the other objects it interacts with, other than what messages can be sent to them, and what replies it can receive from them. The internal workings are thus encapsulated into the definition for each object.

Methods

Methods are procedures, internal to each object, which alter an object's private state. State here means the values of the data items of the object in question.

Examples of methods

Some examples of commonly found methods are as follows:

- **Constructors:** These are used whenever a new instance of an object is created. They initialise the data items contained within the object instance. It is possible for objects to have a number of different constructors, if it is required that they should sometimes be created with different starting values for their data items.

- **Destructors:** These methods are used when an instance of an object is deleted. They ensure that any resources that are held by the object instance, such as storage space, are released.
- **Transformers:** These methods are used to change an object's internal state. There may be a number of transformer methods used to bring about changes to the data items of an object instance.

The Object-oriented approach, therefore, provides the ability to deal with objects and operations on those objects, that are more closely related to the real world. This has the effect of raising the level of abstraction from that used in Relational constructs, such as tables, theoretically making the data model easier to understand and use.

Defining objects - Class definitions

In the Object-oriented approach, everything can, in some way, be described as an object. The term usually applies to a person, place or thing that a computer application may need to deal with. In traditional database terms, an object can be likened to an entity in an E-R diagram, but instead of the entity merely containing attributes, it can also contain methods, sometimes known as operations. These methods are fragments of program code, which are used to carry out operations relevant to the object in question. For example, a Customer object, as well as having the traditional data items we might expect to see in a Customer table, may include operations such as CREATE A NEW CUSTOMER INSTANCE (constructor), REMOVE A CUSTOMER INSTANCE (destructor), CHANGE CUSTOMER DETAILS (transformer), etc.

The attributes and methods for groups or classes of objects of the same type are described in a class definition. Each particular object is known as an instance of that class. The class definition is like a template, therefore, which defines the set of data items and methods available to all instances of that class of object. Some Object database systems also permit the definition of database constraints within class definitions, a feature which might be considered to be a specific case of method definition.

Example of class definition

Consider the object type 'book' as might exist in a library database. Information to be held on a book include its title, date of publication, publisher and author. Typical operations on a book might be:

- Take a book out on loan.
- Reserve a book for taking out on loan when available.
- A Boolean function which returns true if the book is currently on loan and false otherwise. The above operations will be implemented as methods of class 'book'.

The class book may be defined by the following structure:

class book

properties

title : string;

date_of_Publication : date;

published_by : publisher;

written_by : author;

operations

create () -> book;

loan (book, borrower, date_due);

reserve (book, borrower, date_reserved);

on_loan (book) -> boolean;

end book;

A method can receive additional information, called parameters, to perform its task. In the above class, loan method expects a book, borrower and date due for it to perform the loan operation. Parameters are put in the parenthesis of a method. When a method performs its task, it can return data back to the caller method.

An important point to note here is that data abstraction as provided by the class mechanism allows one to define properties of entities in terms of other entities. Thus we see from the above example that the properties published_by and written_by are defined in terms of the classes 'publisher' and 'author' respectively. Outline class definitions for author and publisher could be as follows:

class author

properties

surname : string;

initials : string;

nationality : country;

year_of_birth : integer;

year_of_death : integer;

operations

create () -> author;

end author.

class publisher

properties

name : string;

location : city;

operations

create () -> publisher;

end publisher.

Inheritance

When defining a new class, it can either be designed from scratch, or it can extend or modify other classes - this is known as inheritance. For example, the class ‘manager’ could inherit all the characteristics of the class ‘employee’, but also be extended to encompass features specific to managers. This is a very powerful feature, as it allows the reuse and easy extension of existing data definitions and methods (note that inheritance is not just restricted to data; it can apply equally to the methods of a class). Some systems only permit the inheritance of the data items (sometimes called the state or properties) of a class definition, while others allow inheritance of both state and behaviour (the methods of a class). Inheritance is a powerful mechanism, as it provides a natural way for applications or systems to evolve. For example, if we wish to create a new class of product, we can easily make use of any previous development work that has gone into the definition of data structures and methods for existing products, by allowing the definition of the new class to inherit them.

Example of class definitions to illustrate inheritance:

As an example, we might take the object classes ‘mammal’, ‘bird’ and ‘insect’, which may be defined as subclasses of ‘creature’. The object class ‘person’ is a subclass of ‘mammal’, and ‘man’ and ‘woman’ are subclasses of ‘person’. Class definitions for this hierarchy might take the following form:

class creature

properties

type : string; weight : real;

habitat : (... some habitat type such as swamp, jungle, urban);

operations

create () -> creature;

predators (creature) -> set (creature);

life_expectancy (creature) -> integer;

end creature.

class mammal inherit creature;

properties

gestation_period : real;

operations

end mammal.

class person inherit mammal;

properties

surname, firstname : string;

date_of_birth : date;

origin : country;

end person.

class man inherit person;

properties

wife : woman;

operations

end man.

class woman inherit person;

properties

husband : man;

operations

end woman.

The inheritance mechanism may be used not only for specialisation as described above, but for extending software modules to provide additional services (operations). For example, if we have a class (or module) A with subclass B, then B provides the services of A as well as its own. Thus B may be considered as an extension of A, since the properties and operations applicable to instances of A are a subset of those applicable to instances of B.

This ability of inheritance to specify system evolution in a flexible manner is invaluable for the construction of large software systems. For database applications, inheritance has the added advantage of providing the facility to model natural structure and behaviour.

It is possible in some systems, to inherit state and/or behaviour from more than one class. This is known as multiple inheritance; it is only supported in some Object-oriented systems.

Encapsulation

Encapsulation in object oriented means an object contains both the data structures and the methods to manipulate the data structures. The data structures are internal to the object and are only accessed by other objects through the public methods. Encapsulation ensures that changes in the internal data structure of an object does not affect other objects provided the public methods remains the same. Encapsulation provides a form of data independence.

Review question 2

- Describe the difference between methods and messages in Object-oriented systems.
- Describe a situation in which it may be necessary to provide two different constructor methods for instances of an object.
- Describe the main advantages of inheritance.
- Describe the concept of encapsulation in Object-oriented systems.

Implementing an application of Object databases

Implementing Object databases

An important difference between databases and OO languages is that OO languages create objects in memory, and when an OO application ends, all objects created by the application are destroyed and the data must be written to files in order to be used at a later date. Conversely, databases require access to persistent data. Pure Object-oriented databases make use of Object technology by adding persistence to existing Object-oriented languages; this allows data to be stored as objects even when a program is not running.

In order to implement and manipulate an OO database, it is necessary to use a language that is capable of handling OO concepts. According to Silberschatz (1997) there are several ways in which to do this:

- An existing Relational data-manipulation language can be extended to handle complex data-types and Object-orientation. This leads to Object-Relational systems, discussed later in this chapter.
- A purer OO alternative is to extend an existing OO language to deal with databases, and so it becomes a persistent programming language. C++ and other languages have all had persistent versions implemented.

- The Object database system may be built as such from the beginning. db4objects, DTS/S1, Perst, etc, are examples of pure Object database systems which have been built using this approach.

The use of OO languages allows programmers to directly manipulate data without having to use an embedded data manipulation language such as SQL. This gives programmers a language that is computationally complete and therefore provides greater scope for creating effective business solutions.

Applications for OO databases

There are many fields where it is believed that the OO model can be used to overcome some of the limitations of Relational technology, where the use of complex data types and the need for high performance are essential. These applications include:

- Computer-aided design and manufacturing (CAD/CAM)
- Computer-integrated manufacturing (CIM)
- Computer-aided software engineering (CASE)
- Geographic information systems (GIS)
- Many applications in science and medicine
- Document storage and retrieval

Problems with the OO model

One of the key arguments against OO databases is that databases are usually not designed to solve specific problems, but need the ability to be used to solve many different problems not always apparent at the design stage of the database. It is for this reason that OO technology, and its use of encapsulation, can often limit its flexibility. Indeed the ability to perform ad hoc queries can be made quite difficult, although some vendors do provide a query language to facilitate this.

The use of the same language for both database operations and system operations can provide many advantages, including that of reducing the impedance mismatch: the difference in level between set-at-a-time and record-at-a-time processing. Date (2000), however, does not agree that this is best achieved by making the database language record-at-a-time; he even goes as far as to say that "record-at-a-time is a throwback to the days of pre-Relational systems such as IMS and IDMS". Instead, he proposes that set-at-a-time facilities be added to programming languages. Nonetheless, it could be argued that one of the advantages of pre-Relational systems was their speed. The procedural nature of

OO languages can still lead to serious difficulties when it comes to optimisation, however.

Another problem associated with pure OO databases is that in many cases its use is comparable to that of using a sledgehammer to crack a nut. A large proportion of organisations do not currently deal with the complex data types that OO technology is ideally suited too, and therefore do not require complex data processing. For these companies, there is little incentive for them to move towards Object technology when Relational databases and online analytical processing tools will be sufficient to satisfy their data processing requirements for several years to come. Of course, it is always possible that these companies will find a use for the technology as its popularity becomes more widespread.

The future of OO databases

Many applications falling into the categories cited earlier have been successfully implemented using pure OO techniques. However, the aforementioned problems associated with the OO database model have led to some people doubting as to whether pure OO really is the way forward for databases, particularly with regard to mainstream business applications. Date (2000) is a particularly vehement opponent of pure OO technology, arguing instead that the existing Relational model should evolve to include the best features of Object-orientation and that OO in itself does not herald the dawn of the third generation of database technology.

The Object-Relational model

Perhaps the best hope for the immediate future of database objects is the Object-Relational model. A recent development, stimulated by the advent of the Object-oriented model, the Object-Relational model aims to address some of the problems of pure OO technology - such as the poor support for ad hoc query languages - and open database technology, and provide better support for existing relational products, by extending the Relational model to incorporate the key features of Object-orientation. The Object-Relational model also provides scope for those using existing Relational databases to migrate towards the incorporation of objects, and this perhaps is its key strength, in that it provides a path for the vast number of existing Relational database users gradually to migrate to an Object database platform, while maintaining the support of their Relational vendor.

A major addition to the Relational model is the introduction of a stronger type of system that can accommodate the use of complex data types, which still allow the Relational model to be preserved. Several large database suppliers, including IBM Informix and Oracle, have embraced the Object-Relational model as the way forward.

DB2 Relational Extenders

IBM DB2 Relational Extenders are built on the Object/Relational facilities first introduced in DB2 version2. These facilities form the first part of IBM's implementation of the emerging SQL3 standard. It includes UDTs (User Defined Types), UDFs (User Defined Functions), large objects (LOBs), triggers, stored procedure and checks.

The DB2 Relational Extenders are used to define and implement new complex data types. The Relational Extenders encapsulate the attribute structure and behaviour of these new data types, storing them in table columns of a DB2 database. The new data types can be accessed through SQL statements in the same manner as the standard DB2 data types. The DBMS treats these data types in a strongly typed manner, ensuring that they are only used where data items or columns of the particular data type are anticipated. A DB2 Relational Extender is therefore a package consisting of a number of UDTs, UDFs, triggers, stored procedures and constraints.

When installing a Relational Extender on a database, various files are copied into the server's directories, including the function library containing the UDFs. Then an application is run against the database to define the Relational Extender's database definition to the server. These include scripts to define the UDTs and UDFs making up the Relational Extender.

IBM Informix DataBlades

The DataBlades are standard software modules that plug into the database and extend its capabilities. A DataBlade is like an Object-oriented package, similar to a C++ class library that encapsulates a data object's class definition. The DataBlade not only allows the addition of new and advanced data types to the DBMS, but it also enables specification of new, efficient and optimised access and processing methods for these data types.

A DataBlade includes the data type definition (or structure) as well as the methods (or operations) through which it can be processed. It also includes the rules (or integrity constraints) that should be enforced, similar to a standard built-in data type.

A DataBlade is composed of UDT, a number of UDFs, access methods, interfaces, tables, indexes and client code.

Object-Relational features in Oracle 11

Important

The object features described in the following can only be used with Oracle Enterprise edition. In particular, if you are using Personal Oracle edition for this module, you will not be able to create the objects described. You will however be able to perform the required activities, as these involve examining sample scripts that are included in the Oracle Personal Edition package. If your Learning Support Centre has a version of Oracle running on a mainframe or minicomputer, it is possible that access to the Enterprise Edition of Oracle can be provided. This is not necessary for completion of the activities and exercises of this chapter, but would be necessary if you wish to consolidate the information given here with some practical experience of Oracle's object features.

We shall examine in some detail the facilities incorporated in Oracle11, as these provide a good example of how one of the major database vendors is seeking to increase the level of Object support within the DBMS, while maintaining support for the Relational model.

Abstract data types

Abstract data types (ADTs) are provided to enable users to define complex data types, which are structures consisting of a number of different elements, each of which uses one of the base data types provided within the Oracle product. For example, an abstract data type could be created to store addresses. Such a data type might consist of three separate base attributes, each of which is of type varchar(30). From the time of its creation, an ADT can be referred to when creating tables in which the ADT is to be used. The address ADT would be established with the following definition in Oracle 8:

CREATE TYPE ADDRESS_TYPE AS OBJECT (STREET VARCHAR2(30),

CITY VARCHAR2(30),

COUNTRY VARCHAR2(30));

ADTs can be nested (their definitions can make use of other ADTs). For example, if we wished to set up an ADT to describe customers, we could make use of the address ADT above as follows:

CREATE TYPE CUSTOMER_TYPE AS OBJECT (CUST_NO NUMBER(6),

NAME VARCHAR2(50),

BIRTHDATE DATE,

GENDER CHAR,

ADDRESS ADDRESS_TYPE);

The advantages of ADTs are that they provide a standard mechanism for defining complex data types within an application, and facilitate reuse of complex data definitions.

Object tables

These are tables created within Oracle11 which have column values that are based on ADTs. Therefore, if we create a table which makes use of the customer and address ADTs described above, the table will be an object table. The code to create such a table would be as follows:

CREATE TABLE CUSTOMER OF CUSTOMER_TYPE;

Note that this CREATE TABLE statement looks rather different to those encountered in the chapter on SQL Data Definition Language (DDL). It is very brief, because it makes use of the previous work we have done in establishing the customer and address ADTs.

It is extremely important to bear in mind the distinction between object tables and ADTs.

ADTs are the building blocks on which object tables can be created. ADTs themselves cannot be queried, in the same way that the built-in data types in Oracle such as number and varchar2 cannot be queried. ADTs simply provide the structure which will be used when objects are inserted into an object table. Object tables are the element which is queried, and these are established using a combination of base data types such as varchar2, date, number and any relevant ADTs as required.

Nested tables

A nested table is a table within a table. It is a collection of rows, represented as a column in the main table. For each record in the main table, the nested table may contain multiple rows. This can be considered as a way of storing a one-to-many relationship within one table. For example, if we have a table storing the details of departments, and each department is associated with a number of projects, we can use a nested table to store details about projects within the department table. The project records can be accessed directly through the corresponding row of the department table, without needing to do a join. Note that the nested table mechanism sacrifices first normal form, as we are now storing a repeating group of projects associated with each department record. This may be acceptable, if it is likely to be a frequent requirement to access departments with their associated projects in this way.

Varying arrays

A varying array, or varray, is a collection of objects, each with the same data type. The size of the array is preset when it is created. The varying array is treated like a column in a main table. Conceptually, it is a nested table, with a preset limit on its number of rows. Varrays also then allow us to store up to a preset number of repeating values in a table. The data type for a varray is determined by the type of data to be stored.

Support for large objects

Large objects, or LOBs as they are known in Oracle8, are provided for by a number of different predefined data types within Oracle11. These predefined data types are as follows:

- Blob: Stores any kind of data in binary format. Typically used for multimedia data such as images, audio and video.
- Clob: Stores string data in the database character set format. Used for large strings or documents that use the database character set exclusively. Characters in the database character set are in a fixed-width format.
- Nclob: Stores string data in National Character Set format. Used for large strings or documents in the National Character Set. Supports characters of varying-width format.
- Bfile: Is a pointer to a binary file stored outside of the database in the host operating system file system, but accessible from database tables.

It is possible to have multiple large objects (including different types) per table.

Summary

Despite the advances made in OO technology and its widespread acceptance in general programming use, pure Object-orientation has only achieved serious acceptance in a limited number of specialised fields and not general, industrial-strength applications. The two main reasons for this appear to be the problems that moving to OO introduces, in addition to the fact that Relational technology still has a great deal to offer. The way forward for the use of objects in databases seems to be the Object-Relational model, extending the existing Relational model to incorporate the best features of OO technology, thus delivering the best of both worlds.

Discussion topic

There are a number of applications, such as engineering design, for which Object-oriented database systems are clearly superior to Relational systems. For a number of commercial applications, however, the advantage is perhaps less clear. Imagine you are starting up a company, which requires to keep data about customers, orders, products and sales. Discuss with your colleagues whether you would prefer to go for a Relational, Object-Relational or Object-oriented database solution. Factors you should take into account are as follows:

- The nature of the products to be sold.
- Whether the database is to be connected to the Internet.

- The volume of the data (both in terms of the numbers of records of each type, and the frequency of transactions to be supported).

Consider in your discussions the way in which each of these factors might affect your decision.

Further work

Polymorphism

Object-orientation contains a number of new concepts and terminology, most of which have been introduced to some extent in this chapter. One important area that has not been covered in detail, is the ability to provide alternative implementations of computer processing. For example, it may be required to calculate the salary of full-time employees in one way, and of part-time employees in another. This facility can be provided in Object-oriented systems through the mechanism of polymorphism. Using the core text for the module, investigate the concept of polymorphism, and identify two further situations where it might be applied.