OBJECT ORIENTED DATABASES

Databases for engineering design and manufacturing (CAD/CAM and CIM), scientific experiments, telecommunications, geographic information systems, and multimedia such newer applications have requirements and characteristics that differ from those of traditional business applications, such as more complex structures for objects, longer-duration transactions, new data types for storing images or large textual items, and the need to define nonstandard application-specific operations. Object-oriented databases were proposed to meet the needs of these more complex applications. The object-oriented approach offers the flexibility to handle some of these requirements without being limited by the data types and query languages available in traditional database systems. A key feature of object-oriented databases is the power they give the designer to specify both the structure of complex objects and the operations that can be applied to these objects.

Object Identity

An OO database system provides a unique identity to each independent object stored in the database. This unique identity is typically implemented via a unique, system-generated object identifier, or OID. The value of an OID is not visible to the external user, but it is used internally by the system to identify each object uniquely and to create and manage inter-object references. OID. The value of an OID is not visible to the external user, but it is used internally by the system to identify each object uniquely and to create and manage inter-object references.

The main property required of an OID is that it be immutable; that is, the OID value of a particular object should not change. This preserves the identity of the real-world object being represented.

Object Structure

In OO databases, the state (current value) of a complex object may be constructed from other objects (or other values) by using certain type constructors. One formal way of representing such objects is to view each object as a triple (i, c, v), where i is a unique object identifier (the OID), c is a type constructor (that is, an indication of how the object state is constructed), and v is the object state (or current value). The data model will typically include several type constructors. The three most basic constructors are atom, tuple, and set. Other commonly used constructors include list, bag, and array. The atom constructor is used to represent all basic atomic values, such as integers, real numbers, character strings, Booleans, and any other basic data types that the system supports directly.

Type Constructors

An object definition language (ODL) that incorporates the preceding type constructors can be used to define the object types for a particular database application. The type constructors can be used to define the data structures for an OO database schema.

Encapsulation of Operations, Methods, and Persistence

The concept of encapsulation is one of the main characteristics of OO languages and systems. It is also related to the concepts of abstract data types and information hiding in programming languages. In traditional database models and systems, this concept was not applied, since it is customary to make the structure of database objects visible to users and external programs.

<u>Specifying Object Behavior via Class Operations</u>

The concepts of information hiding and encapsulation can be applied to database objects. The main idea is to define the behavior of a type of object based on the operations that can be externally applied to objects of that type. The internal structure of the object is hidden, and the object is accessible only through a number of predefined operations.

The external users of the object are only made aware of the interface of the object type, which defines the name and arguments (parameters) of each operation. The implementation is hidden from the external users; it includes the definition of the internal data structures of the object and the implementation of the operations that access these structures. In OO terminology, the interface part of each operation is called the signature, and the operation implementation is called a method. Typically, a method is invoked by sending a message to the object to execute the corresponding method. Notice that, as part of executing a method, a subsequent message to

another object may be sent, and this mechanism may be used to return values from the objects to the external environment or to other objects.

Specifying Object Persistence via Naming and Reachability

Transient objects exist in the executing program and disappear once the program terminates. Persistent objects are stored in the database and persist after program termination. The typical mechanisms for making an object persistent are naming and reachability.

The naming mechanism involves giving an object a unique persistent name through which it can be retrieved by this and other programs. This persistent object name can be given via a specific statement or operation in the program. All such names given to objects must be unique within a particular database. Hence, the named persistent objects are used as entry points to the database through which users and applications can start their database access. Obviously, it is not practical to give names to all objects in a large database that includes thousands of objects, so most objects are made persistent by using the second mechanism, called reachability. The reachability mechanism works by making the object reachable from some persistent object. An object B is said to be reachable from an object A if a sequence of references in the object graph lead from object A to object B.

Type Hierarchies and Inheritance

In most database applications, there are numerous objects of the same type or class. Hence, OO databases must provide a capability for classifying objects based on their type, as do other database systems. But in OO databases, a further requirement is that the system permit the definition of new types based on other predefined types, leading to a type (or class) hierarchy.

Typically, a type is defined by assigning it a type name and then defining a number of attributes (instance variables) and operations (methods) for the type. In some cases, the attributes and operations are together called functions, since attributes resemble functions with zero arguments. A function name can be used to refer to the value of an attribute or to refer to the resulting value of an operation (method). In this section, we use the term function to refer to both attributes and operations of an object type, since they are treated similarly in a basic introduction to inheritance.

A type in its simplest form can be defined by giving it a type name and then listing the names of its visible (public) functions. When specifying a type in this section, we use the following format, which does not specify arguments of functions, to simplify the discussion:

TYPE_NAME: function, function, . . . , function

For example, a type that describes characteristics of a PERSON may be defined as follows:

PERSON: Name, Address, Birthdate, Age, SSN

In the PERSON type, the Name, Address, SSN, and Birthdate functions can be implemented as stored attributes, whereas the Age function can be implemented as a method that calculates the Age from the value of the Birthdate attribute and the current date.

The concept of subtype is useful when the designer or user must create a new type that is similar but not identical to an already defined type. The subtype then inherits all the functions of the predefined type, which we shall call the supertype. For example, suppose that we want to define two new types EMPLOYEE and STUDENT as follows:

EMPLOYEE: Name, Address, Birthdate, Age, SSN, Salary, HireDate, Seniority

STUDENT: Name, Address, Birthdate, Age, SSN, Major, GPA

EMPLOYEE subtype-of PERSON: Salary, HireDate, Seniority

STUDENT subtype-of PERSON: Major, GPA

Complex Objects

Unstructured Complex Objects and Type Extensibility

An unstructured complex object facility provided by a DBMS permits the storage and retrieval of large objects that are needed by the database application. Typical examples of such objects are bitmap images and long text strings (such as documents); they are also known as binary large objects, or BLOBs for short. These objects are unstructured in the sense that the DBMS does not know what their structure is—only the application that uses them can interpret their meaning. For example, the application may have functions to display an image or to search for certain keywords in a long text string. The objects are considered complex because they require a large area of storage and are not part of the standard data types provided by traditional DBMSs. Because the object size is quite large, a DBMS may retrieve a portion of the object and provide it to the application program before the whole object is retrieved. The DBMS may also use buffering and caching techniques to prefetch portions of the object before the application program needs to access them.

Structured Complex Objects

A structured complex object differs from an unstructured complex object in that the object's structure is defined by repeated application of the type constructors provided by the OODBMS. Hence, the object structure is defined and known to the OODBMS.

Polymorphism

Polymorphism is the ability to assume many forms. In short it refers to several methods / operators sharing the same name but having different parameter list and different implementations. Variables also can be polymorphic. There are two ways polymorphism can be achieved:

- **Polymorphism through scope:** For example, variables polymorphism.

- **Polymorphism through inheritance:**
    - **Overloading:** Say class Shape has a function CalcArea(int, int). Say a class Circle() is derived from Shape. It overloads the CalcArea() method by specifying only one int parameter, radius.
    - **Overriding:** Continuing the same example suppose a class rectangle is derived from Shape. It overrides the method CalcArea(int, int) to provide its implementation. Class Triangle also overrides the method and provides its implementation.
    - A derived class declares a data member having same name as the base class data member.

Polymorphism means takes more than one form. Means one function or operator can have same name but differnet parameters and different implementations. There are two types of polymorphism.

1. Static polymorphism: It is also known as Early binding. Means binding/polymorphism takes place at the complie time. Eg: method overloading and operator overloading

2. Dynamic ploymorphism: It is also known as Late binding. Means binding/polymorphism takes place at the rum time:

Multiple inheritance and Selective Inheritance

Multiple inheritance is a feature in which an object or class can inherit characteristics and features from more than one parent object or parent class. Multiple inheritance can have the 'diamond problem' that arises when two classes B and C inherit from A, and class D inherits from both B and C. If there is a method in A that B and/or C has overridden, and D does not override it, then which version of the method does D inherit: that of B, or that of C?

Selective inheritance is a feature of some programming languages that allows a subclass to inherit a limited number of properties from the superclass. In the case of multiple inheritance especially, it seems reasonable to allow the selective inheritance of methods. This way if more than one superclass provides a method, the subclass can explicitly name which superclass is to provide the method.

Versions and Configurations

Many database applications that use OO systems require the existence of several versions of the same object. There may be more than two versions of an object.

A configuration of the complex object is a collection consisting of one version of each module arranged in such a way that the module versions in the configuration are compatible and together form a valid version of the complex object.