The ODMG· Object Model

The ODMG object model is the data model upon which the object definition language (ODL) and object query language (OQL) are based. It is meant to provide a standard data model for object databases, just as SQL describes a standard data model for relational databases. It also provides a standard terminology in a field where the same terms were sometimes used to describe different concepts.

Objects and Literals

Objects and literals are the basic building blocks of the object model. The main difference between the two is that an object has both an object identifier and a state (or current value), whereas a literal has a value (state) but no object identifier. In either case, the value can have a complex structure. The object state can change over time by modifying the object value. A literal is basically a constant value, possibly having a complex structure, but it does not change.

An object has five aspects:

- identifier,
- name,
- lifetime,
- structure,
- creation.

1. The **object identifier** is a unique system-wide identifier (or Object_id). Every object must have an object identifier.

2. Some objects may optionally be given a **unique name** within a particular ODMS—this name can be used to locate the object, and the system should return the object given that name. Obviously, not all individual objects will have unique names. Typically, a few objects, mainly those that hold collections of objects of a particular object type—such as extents—will have a name.

These names are used as entry points to the database; that is, by locating these objects by their unique name, the user can then locate other objects that are referenced from these objects. Other important objects in the application may also have unique names, and it is possible to give more than one name to an object. All names within a particular ODMS must be unique.

3. **The lifetime** of an object specifies whether it is a persistent object (that is, a database object) or transient object (that is, an object in an executing pro-gram that disappears after the program terminates). Lifetimes are independent of types—that is, some objects of a particular type may be transient whereas others may be persistent.

4. The **structure** of an object specifies how the object is constructed by using the type constructors. The structure specifies whether an object is atomic or not. An atomic object refers to a single object that follows a user-defined type, such as Employee or Department. If an object is not atomic, then it will be composed of other objects. For example, a collection object is not an atomic object, since its state will be a collection of other objects. In the ODMG model, an atomic object is any individual user-defined object. All values of the basic built-in data types are considered to be literals.

5. **Object creation** refers to the manner in which an object can be created. This is typically accomplished via an operation new for a special Object_Factory interface. In the object model, a literal is a value that does not have an object identifier. However, the value may have a simple or complex structure.

There are three types of literals:

atomic, structured, and collection.

1. **Atomic literals**

Correspond to the values of basic data types and are predefined. The basic data types of the object model include long, short, and unsigned integer numbers (these are specified by the keywords long, short, unsigned long, and unsigned short in ODL), regular and double precision floating point numbers (float, double), Boolean values (boolean), single characters (char), character strings (string), and enumeration types (enum), among others.

2. **Structured literals**

Correspond roughly to values that are constructed using the tuple constructor. The built-in structured lit-erals include Date, Interval,Time, and Timestamp.

3. **Collection literals**

specify a literal value that is a collection of objects or values but the collection itself does not have an Object_id. The collections in the object model can be defined by the type generators set<T>, bag<T>, list<T>, and array<T>, where T is the type of objects or values in the collection.28 Another collection type is dictionary<K, V>, which is a collection of associations <K,

V>, where each K is a key (a unique search value) associated with a value V; this can be used to create an index on a collection of values V.

The notation of ODMG uses three concepts: interface, literal, and class. Following the ODMG terminology, we use the word behavior to refer to operations and state to refer to properties (attributes and relationships).

An **interface** specifies only behavior of an object type and is typically non instantiable (that is, no objects are created corresponding to an interface). Although an interface may have state properties (attributes and relationships) as part of its specifications, these cannot be inherited from the interface. Hence, an interface serves to define operations that can be inherited by other interfaces, as well as by classes that define the user-defined objects for a particular application.

A **class** specifies both state (attributes) and behavior (operations) of an object type, and is instantiable. Hence, database and application objects are typically created based on the user-specified class declarations that form a database schema.

Finally, a **literal** declaration specifies state but no behavior. Thus, a literal instance holds a simple or complex structured value but has neither an object identifier nor encapsulated operations.