IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661,p-ISSN: 2278-8727, Volume 19, Issue 6, Ver. IV (Nov.- Dec. 2017), PP 32-37
www.iosrjournals.org

Approach Of Object Oriented Data Modeling

Loie Naser Mahmoud Nimrawi
Assistant Professor of Computer systems & Complex Networks
Dept . of Computer Science - Sajir College of Science and Arts - Shaqra University

Abstract: *In This section explains some basic concepts such as Object-oriented programming, objects, database system, and object-oriented database. Also there is some explanations of the basic object-oriented modeling .*

Keywords: *complex objects, object identity, classes, attributes, behaviors, encapsulation, inheritance, overriding behaviors, late binding, and naming*

Date of Submission: 12-12-2017 Date of acceptance: 30-12-2017

I. Introduction

1.2 Object Oriented Programming

Object-oriented programming (OOP) is a programming language model organized around "objects" rather than "actions" and data rather than logic data, and Object-oriented programming (OOP) is a method of creating a software application from a group of software components called objects and the application itself is made up of messages that are passed between these objects. OOP contrasts with procedural programming. In procedural programming data and the functions that manipulate the data are separate from one another, because much of the data is available to multiple functions, this can make debugging difficult. Particularly as applications become large, the procedural model begins to break down. By contrast, because data and functions (called methods in OO-speak) are contained within objects, the data is more protected and the application can grow with fewer problems. Objects also model real-world objects more accurately, so conceptually even complex problems can be simpler to address through software.

1.3 Object

Objects are key to understanding Object-oriented technology, Objects are self-contained software components that are used to build object oriented applications, also objects contain data (usually referred to as attributes) ,also ways of working with the data (methods) , and Software objects are often used to model the real-world objects .
Another term associated with this is class, which could be thought as a generic version of the object.

1.4 Database System

A Database system is basically just a computerized record keeping system. The database itself can be regarded as a kind of electronic filling cabinet, i.e., it is a repository or container for collection of computerized data files [5]. Users of the system can perform a variety of operations on such files for example:

1. Add new, empty files to the database.
2. Inserting data into existing files.
3. Retrieving data from existing files.
4. Changing data in existing files.
5. Deleting data from existing files.
6. Removing existing files from the database.

So the definition for a database system is basically a computerized record keeping system. i.e., it is a computerized system whose overall purpose is to store information and to allow users to retrieve and update that information on demand. The information in question can be anything that is of significance to the individual or organization concerned. In order words, that is needed to assist in the general process of running the business of that individual or organization.

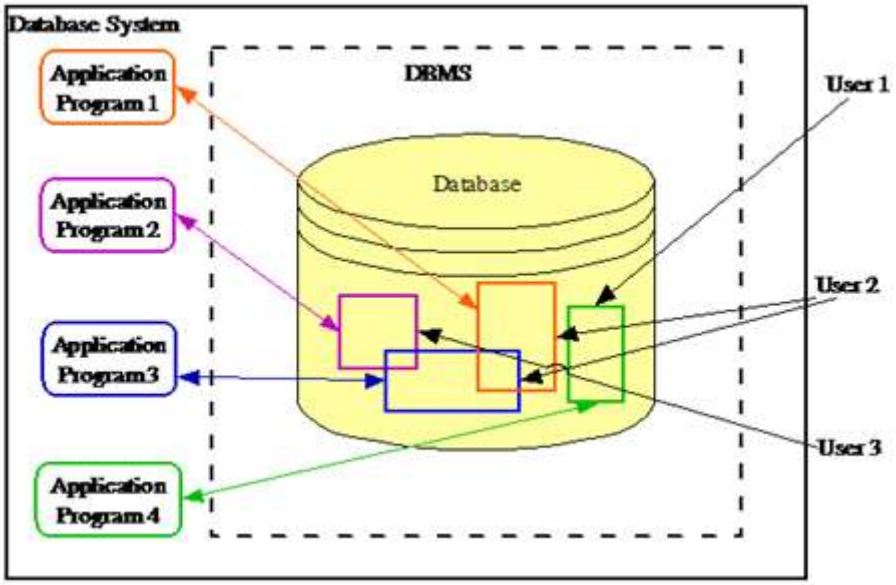

Figure 1.1 *Simplified picture of a database system*

It is meant to show that a database system involves three major components: data, hardware, and users. There are three components briefly considered as below [5].

1.5 Object Oriented Databases

Object oriented databases are also called Object Database Management Systems (ODBMS), Object-oriented database technology are of object-oriented modeling and database technologies. Figure 1.2 illustrates how these programming and database concepts have come together to provide what we now call object-oriented databases.

An Object Oriented Database is the are of Object Oriented modeling and Database Technology	
Object Oriented Modeling	**Database Technology**
Object identity **Encapsulation** **Overriding** **Behaviors** **Naming** **Attributes** **Inheritance** **Classes** **Complex** **Objects**	**Transactions** **Security** **Integrity** **Persistence** **Query** **Recovery** **Concurrency**

Figure 1.2 *Makeup of an Object-Oriented Database*

Perhaps the most significant characteristic of object-oriented database technology is that it combines object-oriented modeling with database technology to provide an integrated application development system.

1.6 Basic Object-Oriented Modeling

An object-oriented database system must satisfy two Standard: it should be a DBMS, and it should be an object-oriented system, i.e., to the extent possible, it should be consistent with the current crop of object-oriented programming languages.

The Standard have ten features :

Attributes, object Identity .complex , Objects ,classes, Types .behaviors , encapsulation. Overriding Behaviors and Late Binding, inheritance, and naming

1.6.1 Attributes

Attributes represent data components that make up the content of a class. Attributes are called data members in the C++ programming language. Instance attributes are data components that are stored by each instance of the class. Class attributes (static data members in C++) are data values stored once for all instances of the class. Attributes may or may not be visible to external users of the class. Attribute types are typically a subset of the basic data types supported by the programming language that interfaces to the OODBMS. Typically this includes enumeration types such as characters and booleans, numeric types such as integers and

floats, and fixed length arrays of these types such as strings. The OODBMS may allow variable length arrays, structures (i.e. records) and classes as attribute types .

1.6.2 Object Identity

Object oriented databases (and programming languages) provide the concept of an object identifier (OID) as a means of uniquely identifying a particular object. OIDs are system generated. A database application does not have direct access to the OID. The OID of an object never changes, even across application executions. The OID is not based on the value stored within the object. This differs from relational databases, which use the concept of primary keys to identify a particular table row (i.e., tuple). Primary keys are based upon data stored in the identified row. The concept of OIDs makes it easier to control the storage of objects (e.g., not based on value) and to build links between objects (e.g., they are based on the never changing OID). Complex objects often include references to other objects, directly or indirectly stored as OIDs, the size of an OID can substantially affect the overall database size due to the large number of inter-object references typically found within an OO application. When an object is deleted, its OID may or may not be reused. Reuse of OIDs reduces the chance of running out of unique OIDs but introduces the potential for invalid object access due to dangling references. A dangling reference occurs if an object is deleted, and some other object retains the deleted object's OID, typically as an inter-object reference. This second object may later use the OID of the deleted object with unpredictable results. The OID may be marked as invalid or may have been re-assigned. Typically, an OODBMS will provide mechanisms to ensure dangling references between objects are avoided .

1.6.3 Complex Objects

Object oriented systems and applications are unique in that the information being maintained is organized in terms of the real-world entities being modeled. This differs from relational database applications that require a translation from the real-world information structure to the table formats used to store data in a relational database. Normalizations upon the relational database tables result in further perturbation of the data from the user's perceptual viewpoint. Object oriented systems provide the concept of complex objects to enable modeling of real-world entities. A complex object contains an arbitrary number of fields, each storing atomic data values or references to other objects (of arbitrary types). A complex object exactly models the user perception of some real-world entity. Complex objects are built from simpler ones by applying constructors to them. The simple objects are objects such as integers, characters, byte strings of any length, booleans and floats (one might add other atomic types). There are various complex object constructors such as tuples, sets, bags, lists, and arrays. The minimal set of constructors that the system should have is set, list and tuple. Sets are critical because they are a natural way of representing collections from the real world. Tuples are critical because they are a natural way of representing properties of an entity. Of course, both sets and tuples are important because they gained wide acceptance as object constructors through the relational model. Lists or arrays are important because they capture order, which occurs in the real world, and they also arise in many scientific applications, where people need matrices or time series data .

1.6.4 Classes and Types

OO modeling is based on the concept of a class. A class defines the data values stored by, and the functionality associated with, an object of that class. One of the primary advantages of OO data modeling is this tight integration of data and behavior through the class mechanism. Each object belongs to one, and only one, class. An object is often referred to as an instance of a class. A class specification provides the external view of the instances of that class.

A class has an extent (sometimes called an extension), which is the set of all instances of the class. Implementation of the extent may be transparent to an application, but minimally provides the ability to visit every instance of the class. Within an OODBMS, the class construct is normally used to define the database schema. Some OODBMS use the term type instead of class.

A type, in an object-oriented system, summarizes the common features of a set of objects with the same characteristics. It corresponds to the notion of an abstract data type. It has two parts: the interface and the implementation (or implementations). Only the interface part is visible to the users of the type, the implementation of the object is seen only by the type designer. The interface consists of a list of operations together with their signatures (i.e., the type of the input parameters and the type of the result). The type implementation consists of a data part and an operation part. In the data part, one describes the internal structure of the object's data. Depending on the power of the system, the structure of this data part can be more or less complex. The operation part consists of procedures which implement the operations of the interface part.

In programming languages, types are tools to increase programmer productivity, by insuring program correctness. By forcing the user to declare the types of the variables and expressions he/she manipulates, the system reasons about the correctness of programs based on this typing information. If the type system is

designed carefully, the system can do the type checking at compile-time, otherwise some of it might have to be deferred at compile time. Thus types are mainly used at compile time to check the correctness of the programs. In general, in type-based systems, a type is not a first class citizen and has a special status and cannot be modified at run-time.

The notion of class is different from that of type. Its specification is the same as that of a type, but it is more of a run-time notion. It contains two aspects: an object factory and an object warehouse. The object factory can be used to create new objects, by performing the operation new on the class, or by cloning some prototype object representative of the class.

The object warehouse means that attached to the class is its extension, i.e., the set of objects that are instances of the class. The user can manipulate the warehouse by applying operations on all elements of the class. Of course, there are strong similarities between classes and types, the names have been used with both meanings and the differences can be subtle in some systems. It do not feel that can be should choose one of these two approaches and it can be consider the choice between the two should be left to the designer of the system. It can be require, however, that the system should offer some form of data structuring mechanism, be it classes or types. Thus the classical notion of database schema will be replaced by that of a set of classes or a set of types .

1.6.5 Encapsulation

The idea of encapsulation comes from (i) the need to cleanly distinguish between the specification and the implementation of an operation and (ii) the need for modularity. Modularity is necessary to structure complex applications designed and implemented by a team of programmers. It is also necessary as a tool for protection and authorization. There are two views of encapsulation: the programming language view (which is the original view since the concept originated there) and the database adaptation of that view.

The idea of encapsulation in programming languages comes from abstract data types. In this view, an object has an interface part and an implementation part. The interface part is the specification of the set of operations that can be performed on the object. It is the only visible part of the object. The implementation part has a data part and a procedural part.

The data part is the representation or state of the object and the procedure part describes, in some programming language, the implementation of each operation.

The database translation of the principle is that an object encapsulates both program and data. In the database world, it is not clear whether the structural part of the type is or is not part of the interface (this depends on the system), while in the programming language world, the data structure is clearly part of the implementation and not of the interface.

Consider, for instance, an Employee. In a relational system, an employee is represented by some tuple. It is queried using a relational language and, later, an application programmer writes programs to update this record such as to raise an Employee's salary or to fire an Employee. These are generally either written in a imperative programming language with embedded DML statements or in a fourth generation language and are stored in a traditional file system and not in the database. Thus, in this approach, there is a sharp distinction between program and data, and between the query language (for ad hoc queries) and the programming language (for application programs).

In an object-oriented system, the employee can be defined as an object that has a data part and an operation part, which consists of the raise and fire operations and other operations to access the Employee data. When storing a set of Employees, both the data and the operations are stored in the database. Thus, there is a single model for data and operations, and information can be hidden. No operations, outside those specified in the interface, can be performed. This restriction holds for both update and retrieval operations .

1.6.6 Behaviors

Behaviors represent the functional component of a class. A behavior describes how an object operates upon its attributes and how it interacts with other related objects. Behaviors are called member functions in the C++ programming language. Behaviors hide their implementation details from users of a class .

1.6.7 Overriding Behaviors and Late Binding

OO applications are typically structured to perform work on generic classes (e.g., a vehicle) and at runtime invoke behaviors appropriate for the specific vehicle being executed upon. Applications constructed in such a manner are more easily maintained and extended since additional vehicle classes may be added without requiring modification of application code. Overriding behaviors is the ability for each class to define the functionality unique to itself for a given behavior. Late binding is the ability for behavior invocation to be selected at runtime based on the class of an object (instead of at compile time) .

1.6.8 Inheritance

Inheritance in the object model is a means of defining one class in terms of another. This is common usage for most of us. For example, a conifer is a type of tree. There are certain characteristics that are true for all trees, yet there are specific characteristics for conifers .

In an object model, there is no distinction in usage between system predefined types and user-defined types. This is known as extensibility. So it is possible to define a type as a sub-type of a system type or as a sub-type of a user-define type. So the inheritance is a way to construct new classes from existing classes. It defines what attributes and methods are available in the new class. Classes may inherit from one or more classes.

As seen in figure 1.3. A class that inherits from exactly one class is said to use single inheritance (sometimes called simple inheritance).

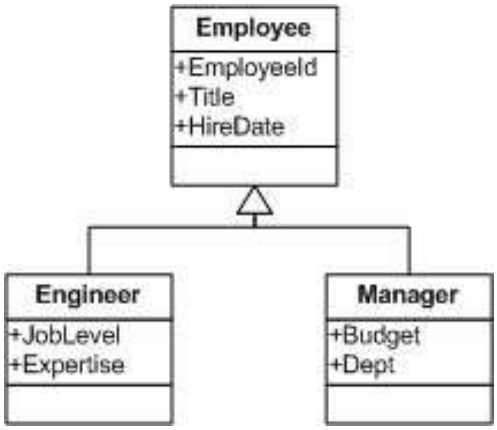

Figure 1.3 *Simple Inheritance Process*

Employee is a base class whereas Engineer and Manager are derived classes. An Engineer is an Employee and so is a Manager. Therefore, both Engineer and Manager inherit all the attributes and methods of an Employee.

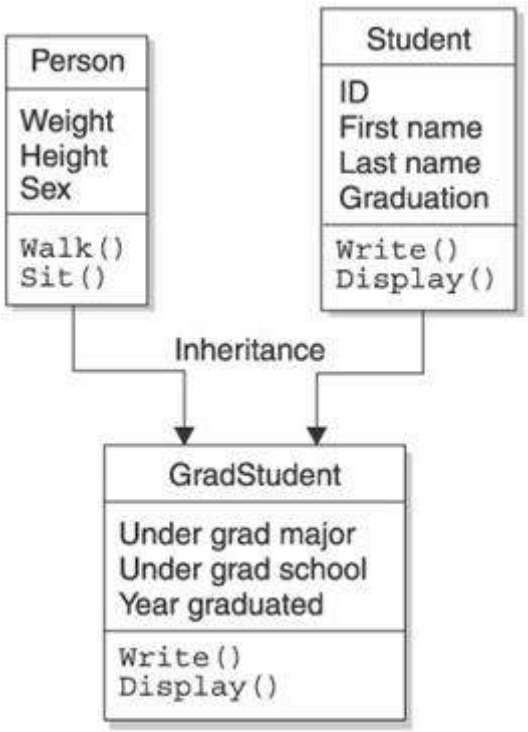

Figure 1.4 *Multiple Inheritance Process*

Multiple inheritance means that inheriting from more than one class. Shown in figure 1.4.In this example, a grad student is a type of Person. Also a grad student is a type of student.and grad student inherits all the attributes and methods of Person and student ,there is certain factors in mind when implementing multiple inheritance:

- Each class that is inherited must pass the "is a" test.

- Parent classes are independent of each other.
- Inheritance occurs in one direction from the parent to the child, which is identical to simple inheritance.
- Any number of parent classes can be inherited by a child class as long as they pass the "is a" test.
- Multiple inheritance can lead to an interesting and potentially confusing issue referred to as the diamond problem

1.6.9 Naming

OO applications are characterized as being composed of a network of inter-connected objects. An application begins by accessing a few known objects and then traverses to additional objects via relationships from the known objects.

As objects are created they are linked (i.e. related) to other existing objects. Given this scenario, the database must provide some mechanism for identifying one or more objects at application start-up without using relations from existing objects. This is typically accomplished by allowing objects to be named and providing a retrieval mechanism based upon name.

An application begins by loading one or two 'high-level' objects that it knows by name and then traverses to other reachable objects. Object names apply within some name scope. Within a given scope, names must be unique (i.e. the same name can not refer to two objects). The simplest scope model is for the entire database to act as a single name scope. An alternative scope model is for the application to identify name scopes. Using multiple name scopes will reduce the chance for name conflicts .

1.7 Summary

Object oriented programming (OOP) is a programming language model organized around "objects" rather than "actions" and data rather than logic data. Objects are key to understanding Object Oriented technology, Objects are self-contained software to build object oriented applications, An Object Oriented Database is the are of Object Oriented modeling and Database Technology and Object-oriented database system must satisfy two Standard and explain first Standard have ten features :

Attributes, objectIdentity.complex, Objects, classes, Types.behaviors, encapsulation. overriding Behaviors and Late Binding, inheritance, and naming

References

[1] Database Management Systems Revisited, An Updated DACS State-of-the-Art Report. Prepared by: Gregory McFarland, Andres Rudmik, and David Lange Modus Operandi, Inc, 1999.
[2] Cattell, R.G.G., Object Data Management: Object-Oriented and Extended Relational Database Systems. Massachusetts: Addison-Wesley, 1997.
[3] C. J DATE An Introduction to Database System – 7th ed, Addison Wesley, 2005.
[4] Database System Manifesto "Malcolm Atkinson" http://www.cs.cmu. edu/People/clamen/OODBMS/Manifesto/htManifesto/Manifesto.html
[5] Elisa Bertino and Lorenzo: Object Database Systems: Concepts and Architectures. Reading, Mass.: Addison-wesley (1993).
[6] Wirfs-Brock, Rebecca, et al., Designing Object-Oriented Software. New Jersey: Prentice-Hall, 1998.

IOSR Journal of Computer Engineering (IOSR-JCE) is UGC approved Journal with Sl. No. 5019, Journal no. 49102.

Loie Naser Mahmoud Nimrawi "Approach Object Oriented Data Modeling." IOSR Journal of Computer Engineering (IOSR-JCE) 19.6 (2017): 32-37.