Chapter 3  
ODS Data Model: An Informal View

3.1 Preamble.

The goal of a database system is to store and manipulate information about the application domain. In the preceding chapters, the fundamental concepts of object oriented database models have been narrated. This chapter and the subsequent chapters present how ODS handles these basic concepts of designing bibliographical database management system and related issues. 

The Object Oriented database management system (ODS) supports multiple object systems simultaneously. One object system represents one integrated and independent unit of database. An object system consists of an object hierarchy, the specification and instances which constitute the database. The scope of a query or communication is always restricted within an object system. The user has to check-in an object system before starting any task and he/she cannot interact with any part of any other object system unless he/she finishes the current task in the object system and checks out of it.

The constructs and the operations of the database system are described precisely to specify how this information is mapped in a database. These constructs and operations are specified in terms of a data model and this chapter describes the one underlying of the system. The model can be briefly described as a generic, object oriented data model, offering a rich classification scheme, object and schema evolution, and strong constraint assurance. In this context, the term generic means that the model can be used for both conceptual modeling of the application domain and for the implementation of the resulting database system. This is a considerable advantage over the traditional method, where the design is usually done with the Entity-Relationship model and the implementation is then done with another model, namely the relational model. The concept of typing by classification makes proposed model distinctive from most currently existing models.
3.2 Object System.

An object system consists of an ODS schema\(^{1,2,3,4}\) and a database instance subsequently may evolve to rivet the changes in the schema or changes in the database instances or both for database model.

A. Database model is a logical organization of the real world objects (entities), constraints on them, and the relationship among objects. A database language is a concrete system implements a data model.

B. A core object oriented data model consists,

i. **Object and object identifier**: Any real world entity is uniformly modeled as an object (i.e. associated with a unique id for pinpointing an object to retrieval).

ii. **Attributes and methods**: Every object has a state (the set of values for the attributes of the object and a behavior, the set of methods program code - which operate on the state of the object). The state and behavior encapsulated in an object is accessed or invoked from outside the object only through explicit message passing.
C. Class means grouping all objects which share the same set of attributes and methods. An object must belong to only one class as an instance of that class (instance of relationship). A class is similar to an abstract data type, a class may also be primitive (no attributes) e.g. integer, string or boolean.

D. Class hierarchy and inheritance: A new class (subclass) is derived from an existing class (superclass). The subclass inherits all the attributes and methods of the existing class and beside having additional attributes and methods of single inheritance (class hierarchy) vs multiple inheritance (class lattice) to form a object structure.

3.2.1 Object Structure.

A. The object oriented paradigm is based on encapsulating code and data into a single unit. Conceptually all the interactions between an object and the rest of the system are via messages. Thus the interface between an object and the rest of the system is defined by a set of allowed messages.

B. In general an object has associated with:

- A set of variables that contain the data for the object. The value of each variable is itself an object.
- A set of messages to which the object responds.
- A set of methods each of which is a body of code to implement each message a method returns a value as the response to the message.

C. Motivation for using messages and methods: All user objects respond to the issue message but in different computations for entitlement. By getting encapsulated within the user object itself, information about how to check the category and the entitlement of all user objects presents the same interface. Since the only external interface presented by an object is the set of messages to which it responds, it is possible to (i) modify the definition of methods and variables without affecting the
rest of the system and (ii) replace a variable with the method that computes a value e.g. entitlement from category of borrowers. The ability to modify the definition of an object without affecting the rest of the system is considered to be one of the major advantages of the object oriented programming paradigm.

D. The method of an object may be classified as either 'read only' or 'update'. The corresponding message can also be classified as 'read only' or 'update'. Derived attributes of an entity in the ER model can be expressed as read only messages for a object class.

3.2.2 Object Classes.

A. Usually there are many similar objects in a database. By similar it is impelled that they respond to the same messages, use the same methods and consist of variables of the same name and type. Similar objects are grouped to form a class. Each such object is called an instance of its class e.g. in a library database users information and borrow/issue are classes and user id is an instance.

B. The definition of the class user is written in code. The definition shows the variables and the messages to which the objects of the class respond but not the methods that handle the messages.

```java
class user {
    / *Variables */
    Varchar2 u_code
    Varchar2 u_name
    Varchar2 u_email
    Varchar2 u_catagory
    Date Reg_data
    Date Exp_date
    / * Messages */
    Varchar2 u_catagory();
```
Varchar2 u_entitle();
Varchar2 loan_period();
Varchar2 U-fineday();
}

It may be wondered why the user field is used as VARCHAR2 in u_code tables INT should be used even though the field is made up of digits however VARCHAR2 is preferred since it will allow dashes and space and alphanumeric character as well as textual numbers (like 01IE2653). Here IE stand for the Industrial Engineering and Management Department.

Figure 3.2- Creation of user database.

Figure 3.3 - Creation of Category and Entitlement.
C. Class (i) captures the instance of relationship (ii) the basis on which a query may be formulated (iii) enhances the integrity of object oriented systems by introducing type checking and (iv) reduces replications of users and integrity-related specifications among objects in the same class.

D. The concept of classes is similar to the concept of abstract data types (ADT). There are several additional aspects to the class concept apart from ADTs. To represent these properties, each class is treated as an object. Metaclass denotes the class of a class. However, most OODB systems do not support the strict notion of metaclass. In ORION\textsuperscript{19} class is the root of the class hierarchy. A class object includes

- a set-valued variable whose value is the set of all objects that are instances of the class.
- implementation of a method for the message new, which creates a new instance of the class.

Since the attributes of an object can not be used to differentiate it from the others, the objects only have attributes when used in a given context and also because, there is no intention to prevent two different objects from having exactly the same attributes in the same context. For example, it is quite possible to think of a library where all copies of a book are represented by the corresponding accession number of objects; in this case, all objects will have the same attributes, but they are still different objects. The obvious solution for solving object identity is to assign a unique object identifier Figure 3.1, or OID, to each object. During the design, system generates the incremental value of these OIDs e.g. data about the books in temp\_BK system, instead of manual entry and the uniqueness of OIDs is assured (the code follows).

```sql
$sql_connect(webdb ,$username,$password);
@$sql_select_db($database= softlib ) or die( "Unable to select database ");
$query="CREATE TABLE contacts (id int(6) NOT NULL auto_increment,first
varchar(15) NOT NULL, last varchar(15) NOT NULL, phone varchar(20) NOT
NULL, mobile varchar(20) NOT NULL, fax varchar(20) NOT NULL, email varchar(30)
```

57
With adoption of system generated incremental value for OIDs, it is now possible to determine the duplicate OIDs and the system is able to determine if two OIDs are equal when their values are equal.

Another important characteristic of OIDs is that they are guaranteed to be an absolutely immutable attribute of the object when an object is created, it obtains its OID, and the same keeps it during whole life. Neither the user nor the system administrator may be able to change the OID of an object. Only when the object is wiped out of the system its OID be recycled and re-assigned to a newly created object.

Object identity is certainly an important feature of an object oriented database management system, but it is not the only one, contrary to the relational DBMS, OODBMS must also be able to manage objects that are more complex than simple flat records. Such complex objects arise not only in large CAE (computer aided engineering) databases, GIS(geographic information systems) or multimedia databases, but also in modern payroll systems, library management, and many others.

Earlier the designer were not comfortable to handle complex objects for the modeling /designing of such application models because non availability of the database management systems for supporting such complex objects. But with the object oriented approach the designer able to juggle around with the complex objects and accomplish cleaner, simpler and better than the previous ones.

Another feature of OODBMS is that objects should not remain passive. On the contrary, they must be active in the sense that they should be able to accept messages and
respond to those messages by calling a routine or, in the object oriented terminology, a method. In proposed design, this concept of active objects is provided the message handler metaphor Inform of LOV (Large Object Volume). LOV are used during the design as it is very simple and at the same time powerful enough to show the principle of active objects. But nothing prevents the system from being extended with a more complex method for calling mechanism. Last but not the least another important feature of an object oriented system is the support for inheritance. With inheritance to mean a mechanism that allows a class to be defined as an extension of a previously defined ODS schema.

3.3 ODS Schema.

The basis of an ODS schema is a class hierarchy. A class specification is the specification of the intentional properties that its instances\textsuperscript{17, 18, 19} must possess. The properties may be structural as well as procedural.

3.3.1 Primitive Class.

There are certain primitive classes of objects having knowing properties e.g. the class int of integers, the class float of floating point numbers, the class bool of truth values etc. These classes are invariant over time with neither their specifications nor the instances can changing with time. The primitive classes may belong to two categories-the primitive classes which can have only a finite set of instances, e.g. bool (finite domain primitive classes) and others, e.g. int (infinite domain primitive classes). Each primitive class must be associated with an effective constructor, i.e. a computational procedure must exist to construct the instances of a primitive class. ODS has in-built primitive classes int, float, bool, string along with basic behavioral properties like arithmetic and relational operations for int, float logic operations for bool and basic string operations for string.
At the time of initialization of an object system, additional primitive classes may be defined by structure definition as in the programming language C. Behavioral properties, however, may later be added to primitive classes. There is a primitive class root, with extensional property id and operations equal (value equality) and equal (identity) over object identifiers. This class is the most primitive class in the sense that every object must be derived from it to construct a class.

3.4 Constructed Class.

Once an object system is initialized, the primitive classes and their instances are fixed. However, new classes may be defined in an object system using existing classes, semantic constructors supported by ODS and message-method definitions. The fundamental semantic constructs supported by ODS are specialization, aggregation and grouping (SET construction). These constructors are more or less similar to those in classical object oriented paradigm. The specifications of these constructs in the ODS model are:

**Aggregation:** It defines a class as a composition of other classes. A class may be defined as an aggregation of named components which may belong to other classes or may be of the same class. For example, the following aggregation declarations of Employee and Dept.

**Example 3.4**

Employee

- Name: string
- Emp_no: string
- Works_in: Dept
- Books_issued: Book
- Recommended_by Employee: Employee

Dept

- Name: string
- Head: Employee
- Location: Address
In this declaration Employee is an aggregation of four components namely name, works_in, books_issued and recommended_by. Any instance of Employee, say emp1, will possess all these properties, i.e. emp1 will have a name, let "B. Majumdar", an emp_no let "88032", the department where emp1 works, let central library, the accession number of the book issued, let TB37049 and an employee, let emp3 (head of the department), who was the introducer of emp1 at the time of issuing/registration of the library borrower card.

Two aggregations is shown diagrammatically as in figure 3.4, the nodes drawn as \( \otimes \) denote aggregations, the nodes drawn \( \square \) as denote the primitive classes and the nodes drawn as \( O \) denote other nodes which are not of primitive types and also not detailed out in the diagram.

In this diagram the aggregation of address and book are not shown. The ISA relationship is also hidden. However, it must be clear that the ISA lattice and the aggregation network do not interfere than for inheritance. If X ISA Y, then the components in aggregation of Y is inherited by X.
3.4.1 Inheritance.

A staff class might be treated differently than a student class, even though they all belong to the class of library user. To extend the behavior of the base class, while creating a specialized class called student, tailor code segment could be used to give the student class special privilege. An object oriented database schema typically requires a large number of classes often, however several classes are similar. For example institute staff member are similar to student users. In the designed two type of inheritance are used. i. Data inheritance (to inherit additional attributes from another class by the data). ii) Function/object inheritance (to inherit data and attributes from another object in its class by the object). In order to allow the direct representation of similarities among classes which need to place classes in a specialization hierarchy.

3.4.1.1 Specialization:

When a class X is defined as a specialization of Y (defined as X ISA Y), the instances of class X inherit all the properties (both structural and behavioral) of class Y. Hence, by virtue of inheritance, any instance of X is logically an instance of Y also. NIL is a special class in ODS which is a specialization of any class. It may be noted that the relation ISA is reflexive and transitive. ODS does not allow multiple generalization which can be expressed by the following proposition.

Proposition 3.4.1.1 If X ISA Y and X ISA Z then either Y ISA Z or Z ISA Y, or X is NIL.

The set of instances in NIL is a null set. Moreover, X ISA root, for every class X. ISA is an antisymmetric relation.
The concept of class hierarchy is similar to that of specialization in the ER model. The corresponding class hierarchy is shown in Fig 3.7.
Fact 3.4.1.1 The class root is an upper bound and NIL is a lower bound, for any subset of classes in an ODS schema.

Lemma 3.4.1.1 ISA forms a lattice over the classes in an ODS schema.

Prof: ISA is reflexive, transitive and antisymmetric. The lub (lower bound) and glb (greatest lower bound) of any two classes in an ODS schema exists. The follows from the fact figure 3.7 and the proposition 3.4.1.1

The keyword 'isa' is used to indicate that a class is a specialization of another class. The specializations of a class are called subclasses e.g. institute staff is a subclass of user and Gr. A staff is a subclass of institute staff. Conversely, institute staff is a superclass of Gr. A staff. Class hierarchy and inheritance of properties form more general classes e.g. an object representing an institute staff contains all the variables of classes Gr. A, Gr. B and Gr. C. Methods are inherited in a manner identical to inheritance of variables.

An important feature of inheritance in OO systems is the notion of substitutability. Any method of a class ‘A’ can be equally will be invoked with an object belonging to any subclass B of ‘A’. This characteristic leads to code-reuse methods and functions in class A (such as u_category in class user do not have to be rewritten again for objects of class B). These are probable ways of associating objects with nonleaf classes to ensure a grouping.

Figure 3.7- ISA hierarchy.
Class user {
    Varchar2 u_name;
    Varchar2 u_address;
};
Class Institute Staff isa user{
    Varchar2 Gr. A;
};

3.5 Grouping.

The grouping construct allows construction sets of instances. In the example 3.4 the class Dept has a name, a head and a location. A department has a SET of employees who work in the Dept. In order to include this information the class Dept. may be modified using the grouping construct SET OF as show below.

Example 3.5
Dept
    name: string
    head: Employee
    location: Address
    employees: SET OF Employee

The grouping construct may be used along with aggregation only as illustrated in example 3.4. It cannot be used to create a distinct named class, which is allowed in the IFO7 model.

ODS supports another specific grouping construct, a linear array indexed by integers. This feature is illustrated in the following aggregation model of a user_entitle

Example 3.5.1
U_catgory : Varchar2
U_entitle Number(2) means upto 99 books can me define
If the library policy is allowed the specific category say 'AL' to borrow maximum of 10 books then it can be identified 2 digit number being an integer index (<99). It may be mentioned that the array construct can also realized with the SET OF construct. This construct is separately included because it provides a better user interface and is amenable to more efficient implementation than as set.

![Vendor and Books Diagram](image)

**Figure 3.8- Classifications.**

### 3.5.1 Classification by Collections.

The primary role of classification is to organize and structure a database by disposing or classifying objects in different collections. In proposed model, a collection represents a semantic grouping of objects, i.e. a set of objects with a common role and common properties. A collection is an object itself. For example, a collection of vendor could contain a set of persons supplying books for the library, and another collection of users could contain a set of persons issuing the books. Figure 3.8 shows a graphical representation of such a classification.

In figure 3.8, collections are represented by grey ovals and the objects inside collections are represented by black dots. In developed model terminology, the objects that belong to a collection are called the members of the collection. The number of members of a collection C is called the cardinality of C and two basic operations that are applicable to collections are insertion and deletion. The semantics of these functions is obvious,
insertion adds a new object to a collection and deletion removes one from it. From a formal point of view, the insertion operation can be seen as a function with the following property:

```php
<?
    sql_connect(localhost,$username,$password);
    @sql_select_db($database) or die( "Unable to select database");
    $query = "INSERT INTO contacts VALUES (,,,,$first,1  '$lasf, '$phone', '$mobile', '$fax', '$email', '$web');"
    mysql_query($query); sql_close(); ?>
```

Since collections are object, it is even possible to make collections of collections and allow nested structures of classification to be built. The set of all collections in a database, together with the rules that control these collections, is called the schema of the database. In many available DBMSs, a schema is a rather rigid structure in the sense that once it is defined, the schema is never changed.

Through this study much more flexible model is introduced. Actually designed model allows collections in the database to be added or removed, or in other terms to change the schema of the database and enabled the user to add new classifications which leads new ways of designing database schemas.

For example to illustrate this idea that need to design a bibliographical database for library to sort information about the books to classification of books supplied by a specific vendor by defining a collection of books and a collection of vendor and a relation between these two collections to show which books are supplied by which vendor.
Besides these constructs, the ODS model specification requires the definition of message and methods. With each class one may define zero or more messages (other than those which are inherited). Each message may be associated with one or more methods inherited). Each message possesses an identifier (name) which is unique for the particular class. Each method also possesses a name unique over the set of methods in the class. A method is specified by a sort signature, formal parameters, local variables and NODAL (New Object Oriented Database Language) statements.

In ODS model each class is treated as a sort. Object variables may be declared to be of such sorts. The ISA relationship provides the subsort relationship. The type checking for message invocation is done using subtyping information inferred from the ISA relationship. Statements written in C can be embedded in the NODAL statements. However, the scoping rule of NODAL does not allow the user to deviate from the object oriented style of programming. A statement in a method may refer to only the formal parameters and to local variables other than the object components. Therefore, the only global environment that exists is the database. An ordinary user can never write a statement in C which can directly operate on the database. Thus side-effects associated with the C-statements are controlled by ODS.
ODS supports queries expressed in the same object oriented style. The ODS model attempts to look uniformly at both specification and manipulation of the database. In order to capture the structural as well as behavioral features the existing models advocate two different formalisms. Other models\textsuperscript{1, 3, 18} use a relational model for the methods and a graph model for the structure. Few models\textsuperscript{5, 19} advocate a complex object model for structure and a procedural transaction model for methods and queries. Models like deductive database\textsuperscript{20} use complex object models for structure and relational model for queries.

This work demonstrated that it is possible to model both the structural as well as the behavioral aspects of objects with the help of accumulated arrows. The concept of accumulated arrows was introduced by Heselink\textsuperscript{14} for modeling non-determinism in data types. Heselink used an accumulated arrow for construction of models and modeling their interactions. The same concept had been adopted here for modeling operations involving objects in an object oriented database. The accumulated arrows can be used for modeling both the schematic as well as operational aspects of an object oriented database. Further, interpretation of accumulated arrows as multi-functions helps to satisfy the database requirement of finding all possible answers to a query and batch operations. The algebra of accumulated arrows had been extended to satisfy the purpose of modeling ODS.
3.6 Reference.


