Chapter one

OBJECT-BASED DATABASES

Traditional database applications like airline reservation and employee management consist of des processing tasks with relatively simpler data types such as integers, dates, and strings, which an well suited to the relational model. However, complex database applications such as Computer Auded Design (CAD), Computer Aided Software Engineering (CASE), and Geographical Information tem (GIS) demand the use of complex data types to represent multivalued attributes, composite m butes, and inheritance. Such features can be easily represented in the E-R and Extended E-R model but are difficult to translate into relational model using simpler SQL data types. Thus, the need of new database model that allows us to deal with complex data types arose.

The new database model proposed was based on the concept of object, since objects represent complex data types naturally. The concept of object was implemented in the database system in he ways, which resulted in object-relational and object-oriented systems. These object-based database systems provide a richer type system including complex data types and object orientation. This chapter discusses both of them in detail.

1.1 NEED FOR OBJECT-BASED DATABASES

Relational databases systems suffer from certain limitations, which led to the development of object based database systems. The main disadvantage of relational databases is that they represent entities and relationships among them in the form of two- dimensional tables. Thus, any kind of complex data such as multivalued and composite attributes are difficult to represent in such databases. For example, consider the multivalued attributes Phone and Email_id from the E-R model of Online Book database. Representation of such attributes in relational model results in the decom- position of relation into several relations (because INF states that all the attributes must have atomic values). However, creation of a new relation for each multivalued attribute may be expensive and artificial in some cases

Another limitation of relational model is that concepts like inheritance are difficult to rep- sent in it. The inheritance hierarchy needs to be represented with a series of tables. In order to data consistency and integrity, referential integrity constraints must also be set up. With complex type systems. E-R model concepts such as multivalued and composite attributes, gen- elization and specialization can be directly represented without a complex translation to the relational model.

Relational database systems may not be suitable for certain applications, but they have so many advantages. That is why many commercial DBMS products are basically relational but also support object-oriented concepts.

Another major reason for the development of object-based databases is the increasing use of object-oriented programming languages in software development. Traditional databases seem to be difficult to use with software applications that are developed in object-oriented languages such as C++, Smalltalk, or Java. Object-oriented databases are designed so that they can be directly integrated with the applications that are developed using object-oriented programming language.

1.2 OBJECT RELATIONAL DATABASE SYSTEMS

Object relational database systems (ORDBMS) are the relational database systems that have been tended to include the features of object-oriented paradigm. The SQL:1999 extends the SQL to support the complex data types and object-oriented features such as inheritance. In this section, we dass implementation of these features in SQL.

1.2.1 Structured Types

In addition to built-in data types, SQL:1999 allows us to create new types which are called user-defined types in SQL. Structured type is a form of user-defined type that allows representing complex data types. Using structured types, composite attributes as well as multivalued attributes of E-R dia- grams are represented efficiently.

1.3Using type constructor

Structure data type are defined using type constructors, namely, row and array. Row type is used to specify the type of a composite attribute of a relation. A row type can be specified using the syntax given here.

CREATE TYPE <Type Name> AS [ROW] (<attribute> <data_type$_1$>,
<attribute> <data_type$_2$>,
:
<attribute> <data_type$_n$>

NOTE: The keyword ROW is optional.

For example, a row type to represent a composite attribute Address with component attributes HouseNo, Street, City, State, and zip can be specified as follows.

CREATE TYPE AddressType AS

(State VARCHAR (30),
City VARCHAR (15),
Street VARCHAR (6),
HouseNo VARCHAR (12),
Zip INTEGER (6));

We can now use the defined type (that is, AddressType) as a type for an attribute while creat ing a relation. For example, a relation PUBLISHER can be created by declaring an attribute Pad dress of type Address Type as shown here

CREATE TABLE PUBLISHER (
PID VARCHAR (20),
Pname VARCHAR (50),
PAddress AdressType);

The attribute Paddress in the PUBLISHER relation is now a composite attribute having HouseNo, Street, City, State, and Zip as its components.

The row types discussed so far hav bee names associated with them. SQL provides an alternative way of defining composite attributes by using unnamed row types. To illustrate this, consider the following declaration.

```
CREATE TABLE PUBLISHER (
PID             VARCHAR (20),
Pname VARCHAR (50),
Paddress        ROW (HouseNo  VARCHAR (12),
                     Street       VARCHAR (12),
                     City          VARCHAR (12),
                     Zip           INTEGER (6));
```

Note that the attribute Paddress has unnamed type and rows of the relation also have an unnamed type.

Now in order to access the components of a composite attribute "dot" notation is used. For example. Paddress. City returns the city component of the Paddress attribute. For example, consider the following query.

```
SELECT Paddress. Housello, Paddress.City
FROM PUBLISHER;
```

In addition to creating an attribute of user-defined type in a relation, SQL also allows creating a relation whose rows are of a user-defined type. For example, we can define a type PublisherType, which can be further used to create a relation PUBLISHER as follows

```
CREATE TYPE   PublisherType AS (
   PID          VARCHAR (20),
  Pname        VARCHAR (20),
  Padress      AdressType);

CREATE TABLE PUBLISHER OF PublisherType;
```

An array type is used to specify an attribute (multivalued attribute) whose value will be a collection. For example, an author can have many phone numbers, thus, the attribute Phone can be repre- sented using array type

Defining Methods

We can also define methods on the structured types. The methods are declared as part of the type definition as shown here

```
CREATE TYPE  PublisherType AS (
        P ID        VARCHAR (20))
        Pname       VARCHAR (20))
        Paddress    VARCHAR (20))
      METHOD ShowName (P_ID VARCHAR (20))
     RETURNS    VARCHAR(50));
```

Here method ShowName is declared that takes P ID as parameter and it is supposed to return the name of the publisher. The method body is created separately as shown here.

```
CREATE INSTANCE METHOD ShowName (P_ID VARCHAR (20))
        RETURNS VARCHAR (50)
        FOR PublisherType
BEGIN
        RETURN SELF. Pname;
END
```

In this declaration, the FOR clause indicates that the method is declared for the type PublisherType, and the keyword INSTANCE indicates that the method executes on an instance of PublisherType. Note that the method returns Pname by using the variable SELF, which refers to the current instance of PUBLISHER on which the method is invoked.

2. Inheritance

Unlike relational system, object-relational database system supports inheritance directly. In chips relational database system, inheritance can be used in two ways: for reusing and refining types i inheritance) and for creating hierarchies of collection of similar objects (table inheritance).

Unlike relational system, object-relational database system supports inheritance directly. In chips relational database system, inheritance can be used in two ways: for reusing and refining types i inheritance) and for creating hierarchies of collection of similar objects (table inheritance).

NOTE SQL:1999 and SQL:2003 supports only single inheritance. Though it may possible that future versions of SQL support multiple inheritance also.

2.1 Inheritance

Type Inheritance

Consider the following type definition for BookType.

CREATE TYPE BookType (

Book title	VARCHAR (50),
Category	VARCHAR (50),
Price	NUMERIC (4),
Copyright_date	NUMERIC (4),
Year	NUMERIC (4),
Page_count	NUMERIC (4),
P ID	VARCHAR (50)) NOT FINAL;

Further, suppose that want store additional information in the database about the books that are textbooks. Instead of creating separate type for text book we can inherit BookType to define type TextBookType shown here.

CREATE TYPE TextBookType **UNDER** BookType (subject VARCHAR (30));

This statement creates a new type TextBookType, which inherits the attributes of BookType, plus it has one additional attribute Subject. Here, BookType is a supertype of TextBookType, and TextBookType is a subtype of BookType.

Note that the keyword **NOT FINAL** states that the subtypes can be created from the given type. In addition, SQL also provides **FINAL** keyword, which states that subtypes cannot be created from the given type.

16.3 OBJECT-ORIENTED DATABASE MANAGEMENT SYSTEMS

Object-oriented database Management systems (OODBMS) provide a closer integration with an object-oriented programming language such as Python, C#, C++, Java, or Smalltalk. An object-oriented database system extends the concept of object-oriented programming language with persistence, concurrency control, data recovery, security, and other capabilities as shown in

Object oriented programming

- ✓ Data encapsulation
- ✓ Inheritance
- ✓ Object identity
- ✓ Polymorphism

Database capabilities

- ✓ Integrity
- ✓ Security
- ✓ Versioning
- ✓ Transaction
- ✓ Concurrency
- ✓ Recovery

Note: it is object oriented DBMS

1.3 Characteristics of Object-Oriented Databases

Object-oriented databases combine the object-oriented programming concepts and database capabilities to provide an integrated application development system. In addition to basic object-oriented programming concepts such as encapsulation, inheritance, polymorphism, and dynamic binding, object-oriented
Database also supports persistence and versioning.

Persistence is one of the most important characteristics of object-oriented database systems. In an object-oriented programming language (OOPL), objects are transient in nature, that is, they exist only during program execution and disappear after the program terminates. In order to convert an OOPL into a persistent programming language (or database programming language), the objects need to be made persistent, that is, objects should persist even after the program termination. OO databases store persistent objects permanently on the secondary storage so that they can be retrieved and shared by multiple programs. The data stored in OO database is accessed directly from the object-oriented programming language using the native type system of the language. Whenever a persistent object is created, the system returns a persistent object identifier. Persistent OID is implemented through a persistent pointer, which points to an object in the database and remains valid even after the termination of program.

Another important feature of OODBMS **is versioning. Versioning** allows maintaining multiple versions of an object, and OODBMS provide capabilities for dealing with all the versions of the object. This feature is especially useful for designing and engineering applications in which the older version of the object that contains tested and verified design should be retained until its new version is tested and released.

Difference between ODBMS and ORDBMS

OODBMS

- ✓ It is created on the basis of persistent program- ming paradigm.
- ✓ It supports ODL and OQL for defining and manipulating complex data types.
- ✓ It aims to achieve seamless integration with object-oriented programming languages such as C++, Java, or Smalltalk.
- ✓ Query optimization is difficult to achieve in these databases.
- ✓ The query facilities of OQL are not supported efficiently in most OODBMS.

- ✓ It is based on object-oriented programming languages; any error of data type made by programmer may affect many users.

ORDBMS

- ✓ It is built by creating object-oriented extensions of a relational database system.
- ✓ It supports an extended form of SQL.
- ✓ Such an integration is not required as SQL:1999 allows us to embed SQL commands in a host language.
- ✓ The relational model has a very strong foundation for query optimization, which helps in reducing the time taken to execute a query.
- ✓ The query facilities are the main focus of ORDBMS. The querying in these databases is as simple as in relational database system, even for complex data types and multimedia data.
- ✓ It provides good protection against programming errors