MASTER OF COMPUTER APPLICATIONS

SEMESTER 1

RELATIONAL DATABASE MANAGEMENT SYSTEM

Unit 10

Object Oriented DBMS

Table of Contents

1. INTRODUCTION

In the previous unit, you studied the concept of parallel database architectures. You also studied the concept of parallel query evaluation, parallelising individual operations, I/O Parallelism etc.

In today's world, Client-Server applications that rely on a database on the server as a data store while servicing requests from multiple clients are quite commonplace. Majority of these applications use Relational Database Management System (RDBMS) as their data store simultaneously with an object oriented programming language for development. This causes certain inefficiencies as objects must be mapped to tuples in the database and vice versa instead of the data being stored in a way that is consistent with the programming model. To overcome this problem Object Oriented Database Management Systems (OODBMS) have been developed.

In this unit, you will study the concept of object oriented DBMS. You will learn about object oriented paradigm and architectural approaches of OODBMS. Also you will recognise the concept of object oriented data model, OODBMS terminology, type hierarchies and inheritance. We will also discuss the concept of type extents and persistent programming languages.

1.1 Objectives

After studying this unit, you should be able to:

- ❖ *Discuss object oriented paradigm*
- ❖ *Recognise OODBMS architectural approaches*
- ❖ *Describe object identity, its procedures and encapsulation*
- ❖ *Explain object oriented data model*
- ❖ *Describe relationships and identifiers*
- ❖ *Discuss basic OODBMS Terminology*
- ❖ *Recognise basic interface and class structure*
- ❖ *Explain the concept of type hierarchies and inheritance*
- ❖ *Discuss the concept of type extents and persistent programming languages*

2. OBJECT ORIENTED PARADIGM

The Object Oriented model or paradigm relies on the encapsulation of the data and code into one single unit. All the interactions among one object and its system are performed through messages. Therefore, a set of allowed messages defines the interface among an object and its system. Generally, an object has association with:

- A group (set) of methods, where every method is a set of code to execute every message. A method submits a value as reply of the message.
- A group (set) of variables that include the object data. The measure of every variable is by itself one object.
- A group (set) of messages and the object reacts to these messages.

Now let us discuss the incentive of using messages & methods: As an example consider employees as objects and annual-wage as message. Every employee object reacts to the yearly-wage message but in dissimilar calculations for managers, back-end employees, etc.

As the sole external interface portrayed by one object is pack of messages, to which it reacts, it is feasible to:

- modify the variables and methods' definition and not having any affect on the remaining system
- substitute a variable with a method that calculates a value

The main benefit of the object oriented paradigm is the capability to modify an object definition without having an effect on the remaining system. You can classify the methods of any object as either 'readonly' or 'update'. Also, you can classify message as 'readonly' or 'update'. You can express the entity's derived attributes in the E.R model as readonly messages.

Another major benefit of the object oriented paradigm is its ability to understand easily. It facilitates natural illustration of real-world objects, their mutual relationships and behaviour and is thus close to customers. An object oriented application comprises of a set of objects with their own private state, having an interaction among themselves. Object oriented systems can be maintained easily since they are modular and objects are independent of each other.

Other objects in the system should not be affected by change in one object. Object oriented paradigm removes the requirement for shared data areas, hence diminishing system coupling. The paradigm assists reusability. Objects are self-reliant and may be utilised in other suitably similar applications.

SELF-ASSESSMENT QUESTIONS – 1

1. ______________________ are said to be self-reliant and may be utilised in other suitably similar applications.
2. The methods of an object cannot be classified as either read-only or update. (True/ False)

3. OODBMS ARCHITECTURAL APPROACHES

Now you will learn about various architectural approaches relevant to an OODBMS. These approaches are as follows:

- Distributed Client - Server Approach
- Data Access Mechanism
- Object Clustering
- Heterogeneous Operation Let's discuss them in detail.

3.1 Distributed Client – Server Approach

Enhancements in technologies of local area network and workstation have given rise to group design type applications fulfilling the need for OODBMS For example, Electronic Offices, CASE, CAD, etc. OODBMS are usually implemented in a multiple process distributed environment. Various services of database are offered by server processes. These services may be managing secondary storage, controlling transaction, etc.

Client processes manage application specific activities like utilisation and updation of separate objects. These processes may be situated on the same workstation or on dissimilar workstations. Usually, a single server will communicate with numerous clients providing simultaneous requests for data which is managed by that server. A client may interact with numerous servers to use data distributed all through the network.

There are three different workstation-server architectures that have been proposed for use with OODBMS. There are discussed as below:

Object server approach

An object is considered as the unit of transfer from server to client. Both machines store objects and are competent of performing methods on objects. Object-level locking is carried out easily.

The main disadvantage of this approach is the overhead related with the server interaction needed to access each object.

Another disadvantage is the added complexity of the server software which must offer whole OODBMS functionality.

Page server approach

In this approach, we consider page as the unit of transfer from server to client. The overhead of object access is decreased by the transfers of page level since it does not need server interaction at all times. You can simplify the architecture and implementation of the server as it needs only executing the services of backend databases.

A probable disadvantage of this approach is that methods can be assessed only on the client. Therefore all objects that an application uses must be transported to the client. Here, it is difficult to implement object level locking.

File server approach

In this approach, the client processes of OODBMS have an interaction with a network file service for reading and writing database pages. This approach makes the process of the server implementation simpler because there is no need to manage secondary storage. The main disadvantage of this approach is that it requires two network interactions for accessing data.

From the three different approaches discussed above, the page server approach provides buffer pools and efficient clustering algorithms. If large amount of data is scanned by applications, the object server approach performs badly, but is better as compared to the page server approach for applications executing numerous updates and executing on workstations with small buffer pools.

3.2 Data Access Mechanism

Assessment of Object oriented DBMS products should take into account the procedure required to shift data from secondary store unit into a consumer application. Usually this necessitates interaction with the server process, probably across one network.

Objects that are stored into a consumer's memory may need more processing. The cost and procedure of releasing locks, and updated objects that are returned to the server should be considered.

3.3 Object Clustering

The process of transferring units larger as compared to an object is done under the supposition that an access of an application to a specified object signifies a high possibility that it may also access other related objects.

When transferring number of objects, further server interaction may not be required to assure these further object accesses.

Object clustering can be defined as the capability for an application to offer information to the object oriented DBMS. This is done so that objects which are usually accessed mutually can be accumulated close to each other and therefore benefits from bulk transfers of data.

3.4 Heterogeneous Operation

In this approach, an object oriented DBMS offers a method in which applications can work together. This is done by sharing access to a common group of objects. Numerous concurrent applications are supported by a usual OODBMS, these applications are executed on numerous processors which are connected through a local area network.

Frequently, the processors will be from dissimilar computer companies where each company comprises its own data representation formats. To make applications work together in this kind of an environment, data must be converted to the representation format appropriate for the processor.

Then the data is accumulated enduringly by a server and momentarily by a client who desires to access the data. To make object oriented DBMS an efficient integration method, it must support data access in heterogeneous processing surroundings.

SELF-ASSESSMENT QUESTIONS – 2

3. In which of the following approach, the unit of transfer to client from server is regarded as a page?
   a) Page Server
   b) File Server
   c) Object Server
   d) Blade server
4. You can define object clustering as the potential of an application to offer information to object oriented DBMS. (True/ False)

4. OBJECT IDENTITY

An identity of an object is maintained even when all or some values of variables or even definitions of methods vary with time. The object identity concept is essential in applications, however, it does not relate to relational database tuples.

Object identity is considered as a powerful concept of identity as compared to the ones that are usually seen in the programming languages or in the data models which are not based on the object orientation.

Various forms of identity which exist are defined as below:

- ***Name:*** This signifies a user supplied name which is utilised for identity; e.g., name of a file in the file system.
- ***Value:*** This signifies a value of data which is utilised for identity; e.g., primary key of one row in single relational database.
- ***Built-in:*** This signifies that an idea of identity is built (created) into programming languages or the data model or and it does not need any user supplied identifier.

You can implement object identity through an exclusive, system generated object identifier (OID). The external user cannot see value of OID. However, it can be utilised by a system internally to recognise every object in a unique manner and to generate and handle the references of inter-object.

In numerous circumstances, the automatic generation of identifiers by the system is considered as an advantage. This is because it does not require people to do that task. But, people should make use of this ability with caution.

Identifiers produced by the system are generally particular to the system. If data are moved to another database system, then there is a need to translate identifiers. If entities that are being modelled previously contain distinctive identifiers which are from outside the system, then identifiers produced by system may not be necessary.

SELF-ASSESSMENT QUESTIONS – 3

5. The _____________________ of object identity can't be seen by the user.
6. Value of OID can't be viewed by the external user. (True/ False)

5. PROCEDURES AND ENCAPSULATION

Procedures are used to illustrate an object's behaviour and they are also known as functions or methods. The high level of abstraction can be viewed by the user in an OODBMS. In OODBMS, data is encapsulated inside the object.

This data can only be accessed by OODBMS procedures which are related with that particular object. To be an appropriate OODBMS, it is necessary for the database to comprise procedures beside just data.

Encapsulation basically signifies hiding the data inside the object from the outside classes. Classes perform the encapsulation of the attributes and behaviours of their objects.

By means of behaviour encapsulation, the users of the class are not allowed to view the internal implementation of behaviour. This process offers some amount of data independence in order that users are not required to be modified when behaviour implementations are modified. Attributes of a class may or may not be encapsulated.

Changing the definition of the attributes of a class that are not encapsulated needs variation of all users that use them. The attributes that users of a class cannot use are encapsulated. Attributes that are encapsulated generally comprise of behaviours that offer some kind of access to the attribute by the users. Variations to these attributes usually do not need variation to users of the class.

SELF-ASSESSMENT QUESTIONS – 4

7. Which of the following process hides the internal data of the object from the outside classes?
   a) Implementation
   b) Encapsulation
   c) Attribute hiding
   d) Inheritance
8. By means of behaviour encapsulation, the users of the class are allowed to view the internal implementation of behaviour. (True/False)

6. OBJECT ORIENTED DATA MODEL

A data model is defined as an organisation of the real world objects (entities), restrictions on them, and the relationships between the objects. A database language is considered as a tangible syntax for a data model. A data model is implemented by the database system.

Object oriented data model comprises various Object Oriented concepts:

1. ***Object and object identifier:*** Entity of any real world is known as an object (related with a unique id: utilised to refer to an object for retrieval). In object oriented databases, OID identifies the objects uniquely. OID format is particular for every system.
2. ***Attributes and methods:*** Each object comprises of a state and behaviour, where a state is the group of values for the object's attributes and behaviour is the group of methods, that is, code which functions on the state of the object. Only clear message passing can access or invoke the state and behaviour which are encapsulated in an object.

   [An instance variable whose domain can be any class, that is, user- defined or primitive is known as an attribute.]
3. ***Class:*** A class includes a set of all objects, which partake (share) the same group of methods and attributes. Object must relate to just one class. An object is considered as example for that class.

4. ***Class hierarchy and inheritance:*** This includes deriving a new class (which is known as subclass) from a current class (which is known as superclass). All the methods and attributes of the current class are inherited by a subclass. Also subclass may comprise additional attributes &methods. The concept of single inheritance (class hierarchy) and multiple inheritances will be discussed further in this unit.

SELF-ASSESSMENT QUESTIONS – 5

9. You cannot use database system to implement a data model. (True/ False)
10. Which of the following can be defined as the group of values for the object's attributes?
    a) State
    b) Class
    c) Behaviour
    d) Method

7. RELATIONSHIPS

Relationships are one of the significant constituents of the Object Oriented paradigm. Relationships permit objects to consider each other and effect in networks of inter-related objects. Relationships are considered as the paths utilised to carry out navigation-based data access.

The capability to directly and proficiently display relationships is one of the main enhancements of the Object Oriented data model over the relational data model. This decreases data independence by depending on the occurrence of particular relationships and indexes.

Theoretically, you can consider relationships as abstract entities that permit objects to refer to each other. An OODBMS may select to symbolise relationships as attributes of the class (from which the relationships originate), as independent objects (where case relationships may be extensible and permit attributes to be added to a relationship), or as hidden data structures connected to the owning object in some way.

We frequently call relationships as references, associations, or links. At times, we use the term relation to signify the schema definition of the potential for inter-connections among objects. We use the term relationship to signify actual incidences of an inter-connection among objects. Here, the term relationship is used interchangeably for both the schema definition as well as the object level existence of connections among objects.

Though we can discover much regarding an object by observing its attributes, at times a significant fact regarding an object is the manner in which it connects to other objects in the same or another class.

Example: Let us consider a class known as movie. We have given below the declarations of four attributes that are comprised by all movie objects.

1. class Movie {
2. attribute string Name ;
3. attribute integer Year ;
4. attribute integer length ;

5. attribute enum Film {colour, black and white } film Type;

Now, assume that you want to add a property (that is a set of stars) to the declaration of the Movie class. More specifically, we would like to connect each Movie object to the set of Star objects. The best manner to symbolise this connection among the classes, Movie and Star, is with a relationship. This relationship can be represented in Movie by the following line: relationship Set<Star> stars;

The above line is represented in the declaration of class Movie. This line may emerge after any of the lines numbered (1) to (5). It signifies that in every object of class Movie, there is a group of references to Star objects. The set of references is known as stars. Here, the keyword relationship indicates that stars enclose references to other objects, whereas the keyword Set previous to<Star> indicates that stars refers to a set of Star objects, instead of a single object.

SELF-ASSESSMENT QUESTIONS – 6

11. Relationships can be considered as ________________ entities that permit objects to refer to each other.
12. Which of the following component of Object Oriented paradigm is used to carry out navigation-based data access?
    a) Identifiers
    b) Relationships
    c) Inheritance
    d) Attributes

Activity 1

Illustrate the concept of relationships in OODBMS with example.

8. IDENTIFIERS

Object identifiers can uniquely identify objects.

- You can store object identifiers as a field of an object, and they are referred to another object. For example, the field of a person object named as spouse can be considered as an identifier of another person object.
- Object identifiers can be considered as system generated (that is produced by database) or external (such as social-security number).

To identify the object in some of the systems, only 4 byte with object position or object index in file is sufficient. However, in some other systems, object identifiers are considered to be more complicated and maintain exclusiveness even outside the local computer's range.

SELF-ASSESSMENT QUESTIONS – 7

13. Object identifiers cannot be accumulated as a field of an object. (True/ False)
14. For identifying the object in some systems, only _______________ bytes with object index or object position in the file is sufficient.

9. BASIC OODMS TERMINOLOGY

OODBMS Terminology includes the following:

- ***Object Identity:*** We have already discussed this concept as above.
- ***Classes:*** A class defines the data values accumulated by an object of that class. Every object is related to just one class. An object is frequently considered as an instance of a class. Specification of a class offers the outside view of class's instances. In case of OODBMS, the class construct is usually utilised to define the schema of a database. Make note that some object oriented databases make use of the term type in place of a class. The objects that are to be accumulated inside the database are defined by OODBMS.
- ***Encapsulation:*** We have already discussed this concept in the section above.
- ***Inheritance:*** Inheritance is defined as the process where the behaviour and properties of parent object are inherited by the child object. If class 'A' is derived from class 'B', all methods of class 'B' are inherited by class 'A'. Also it can be applied all over where class 'B' can be employed.

SELF-ASSESSMENT QUESTIONS – 8

15. Object is considered as an ____________________ of a class.
16. What do you call the process where the behaviour and properties of parent object are inherited by the child object?
    a) Encapsulation
    b) Inheritance
    c) Polymorphism
    d) Message passing

10. BASIC INTERFACE AND CLASS STRUCTURE

An interface is used to illustrate the behaviour or ability of a class without performing to a specific implementation. Interface symbolises an agreement between a supplier and its users, which defines what's needed from every implementer. This is done in terms of the services they must offer, regardless of how they handle to do it.

Declaration of interfaces and classes is comparable to C++ and java syntax, but not quite the same. But the restrictions of a class or interface declaration are taken directly from C++. We have shown the declaration as below.

```
Class class_name
{
// class methods
};
Interface interface_name
{
// interface methods
};
```

We begin every declaration with either the keyword class or interface to recognise the element which is being declared. After writing the keyword, we write the name of the interface or class. This is to note that the class or interface names start with uppercase letters.

If one or more interfaces are implemented by a class, a separation from a class name is provided between those interfaces by a colon:

```
Class class_name : interface interface_name
{
//class methods
};
```

Now, if a class inherits from a superclass, it extends the class, as shown below:

```
Class class_name extends superclass_name:interface_name
{
```

```
//class methods
};
```

SELF-ASSESSMENT QUESTIONS – 9

17. If you want to implement one or more interfaces by a class, then a separation between those interfaces from a class name is provided by a ________.
18. An interface illustrates the behaviour or ability of a class without performing to a specific implementation. (True/ False)

11. TYPE HIERARCHIES AND INHERITANCE

You can specify the object types by means of a type hierarchy. Type hierarchy permits the inheritance of both attributes and methods of types defined earlier.

In the simplest manner, a type could be defined by providing it a type name and after that giving the names of its public (visible) functions.

An object oriented database usually needs numerous classes. Frequently, however, various classes are analogous. For instance, bank employee is analogous to consumer.

To permit the direct demonstration of similarities between classes, it is required to put classes in a specialisation hierarchy. For example we have shown a specialisation hierarchy for E-R model in Figure 10.1.

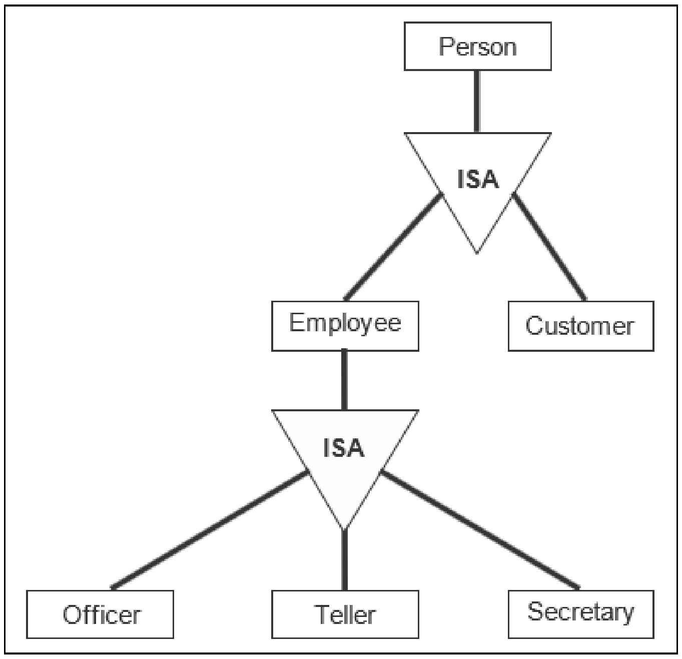

Fig 10.1: Specialisation Hierarchy for Banking Example

Class hierarchy concept is said to be analogous to the specialisation hierarchy. We have shown the consequent class hierarchy in Figure 10.2.

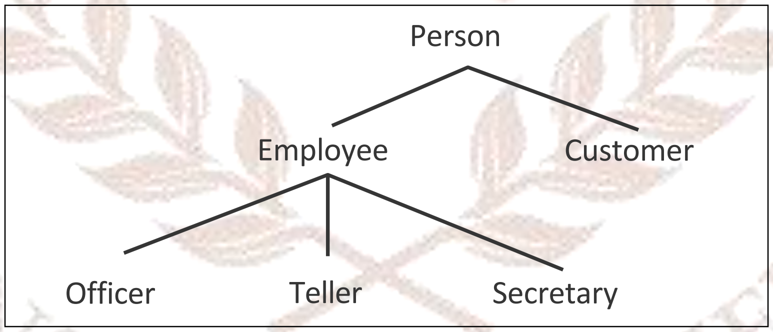

Fig 10.2: Class Hierarchy matching the Banking Example

We can define the class hierarchy in pseudo-code. Now we will show the definition of class hierarchy in pseudo code. Also we have shown the variables related with each class. This is shown as below:

```
Class person
{
String name; string address:
};
Class customer isa person
 {
Int credit-rating:
};
Class employee isa person
{
Date start-date; int salary:
};
class officer isa employee
{
Int office-number; int expense-account-number:
};
class teller isa employee
{
Int hours-per-week; int station-number:
};
```

Now let us define subtypes and supertypes.

When a new type is generated by the user that is analogous but not the same to a previously defined type, it is known as a subtype. All the functions of the subtype are inherited by a supertype.

The supertype which is situated at the top of the type hierarchy includes a set of fields that is inherited by all related subtypes. Before creating a subtype, an existence of supertype is must.

We use the keyword ‘isa’ to signify that one class is considered as a specialisation of another class. We call the specialisation of a class as sub- classes. For example, *a sub-class of person (super-class) can be employee; a subclass of employee (super-class) can be cashier (or teller). On the other hand, employee is considered as a super-class of cashier (or teller).*

The concept of *substitutability* is considered as a significant benefit of inheritance in object oriented systems. Consider a class ‘A’. You can call any method of a class, ‘A’, by means of an object relating to any subclass ‘B’ of ‘A’. Thus you can reuse the code. For example, there is no need to rewrite the methods &functions in class ‘A’ (like getname in class person) for objects of class ‘B’.

There are two probable methods of relating objects with non-leaf classes:

- Associate all employee objects comprising the instances of officer, teller, and secretary. with the employee class
- Associate merely those employee objects that are instance neither teller nor officer, nor secretary, with the employee class.

Usually the second option is made in object oriented systems. In this case, it is possible to identify the set of all employee objects by combing those objects related with all classes in the subtree rooted at employee.

Many object oriented systems permit specialisation to be partial, that is, they permit objects that relate to a class like employee that do not relate to any of subclasses of that class.

Multiple inheritance

The capability of class to inherit the variables and methods from numerous superclasses is known as Multiple inheritance.

Mostly, tree-structured organisation of classes is sufficient to illustrate applications. There are circumstances that can’t be displayed substantially in one tree structured class hierarchy.

An example: In Figure 10.3, we have created subclasses like *part-time- secretary, full-time-secretary,* etc. But there are some problems:

(a) Redundancy results in possible irregularities on updates

(b) The full time /part time employees which are not considered as secretaries and tellers both cannot be demonstrated in the hierarchy.

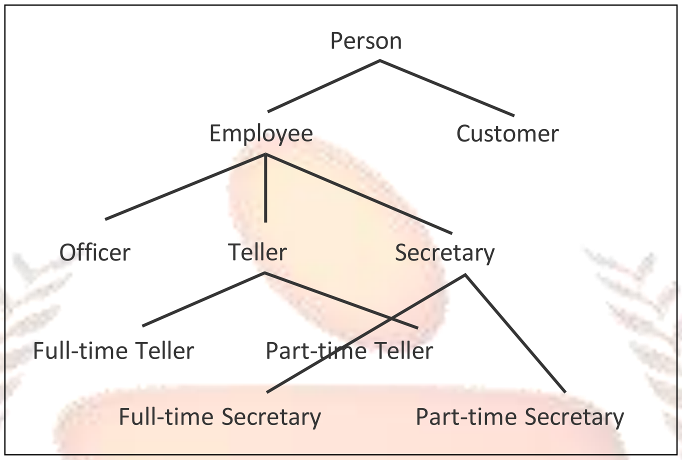

Fig 10.3: Class Hierarchy for Part Time and Full Time Employees

In Figure 10.4, we have shown the relationship of class and subclass. This is shown by a rooted DAG (directed acyclic graph) where a class might comprise of extra superclass than one.

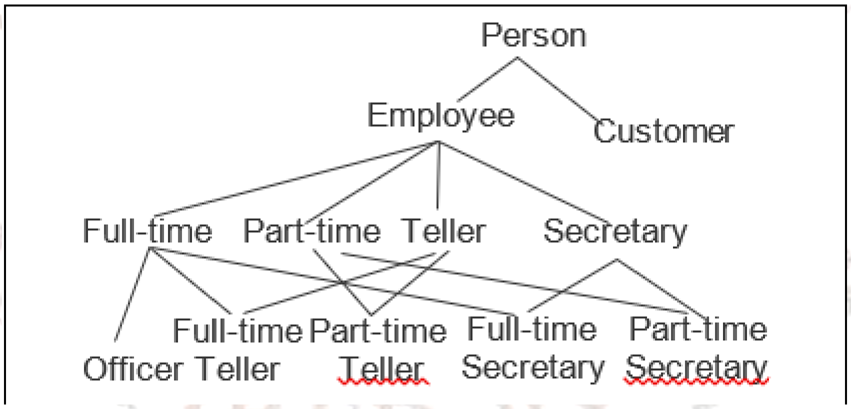

Fig 10.4: Class Directed Acyclic Graph for Banking Example

Dealing with name conflicts

On the utilisation of multiple inheritance, there is possible uncertainty that whether you can inherit the same method or variable from more than 1 superclass.

Example: Consider the example of banking, where a variable pay is defined for each full-time, part-time, teller & secretary as given below:

- ***full-time:*** salary is one integer from 0-100000 comprising of yearly wages.
- ***part-time:*** salary is one integer from 0-30 comprising of an per-hour rate of wages.
- ***teller:*** salary is one integer from 0-15000 comprising of the yearly wages.
- ***secretary:*** salary is one integer from 0-20000 comprising of the yearly wages.

You can inherit the description of salary from either from part-time or from secretary. In case of part-time secretary, we have the following alternatives:

- Comprise both variables, renaming them to part-time-pay and secretary- pay.
- Select one or the other depending on the order of creation.
- Tell the consumer to make an option at the time of class definition.
- Consider the situation as an error.

Till now, no solution has been considered as best, and different systems make different selections.

All the cases in multiple inheritances do not lead to uncertainty. If, rather than defining salary, we maintain the variable salary definition in class employee, & do not define it anywhere, then salary is inherited by all the subclasses from employee (no uncertainty).

Multiple inheritances can be used to illustrate the concept of roles. For example, consider the subclasses student, teacher and football Player. For these subclasses an object can relate to numerous categories at once and we call each of these categories as a role.

Multiple inheritances can be used to create subclasses, like student-teacher, student-football-player, etc. to illustrate the chances of an object concurrently comprising multiple roles.

SELF-ASSESSMENT QUESTIONS – 10

19. A class can be considered as a specialisation of another class by using the keyword ___________.
20. The object types can be specified by means of a type hierarchy. (True/ False)

12. TYPE EXTENTS AND PERSISTENT PROGRAMMING LANGUAGES

In many object oriented databases, the collection of objects in an extent comprises of the same type or class. But, since types are supported by most of the object oriented databases, it is assumed that extents are collections of objects of the same type.

Persistent data is defined as the data that continue to occur even after program that generated it, has finished.

A programming language which is expanded by means of constructs to manage persistent data is known as a persistent programming language. It is differentiated from embedded SQL by the following ways:

- In the language of a persistent program, language is completely incorporated with host language and same type of system is used by both. Any changes in format that are needed in the databases are performed clearly.
- By means of Embedded SQL, the programmer is accountable for writing unambiguous code to retrieve data from memory or send data back to database.
- In the language of persistent program, one programmer could use persistent data without writing such code in an unambiguous manner.

Disadvantages of persistent programming languages

- Persistent programming language is influential; however, it is easy to make mistakes that can harm the database.
- Performing routine high level optimisation is tough.
- Declarative querying is not supported properly.

SELF-ASSESSMENT QUESTIONS – 11

21. A programming language which is expanded by means of constructs to manage persistent data is known as a ___________________.
22. Performing routine high level optimisation in persistent programming language is

13. SUMMARY

Let us recapitulate the important points discussed in this unit:

- The Object Oriented model or paradigm relies on the encapsulation of the data and code into one single unit.
- A set of allowed messages defines the interface among an object and its system.
- Enhancements in technologies of local area network and workstation have given rise to group design type applications fulfilling the need for OODBMS.
- An object is considered as the unit of transfer from server to client.
- In page server approach, we consider page as the unit of transfer from server to client.
- In file server approach, the client processes of OODBMS have an interaction with a network file service for reading and writing database pages.
- The ability of class to inherit the methods and variables and from several superclasses is multiple inheritance.
- A programming language which is expanded by means of constructs to manage persistent data is known as a persistent programming language.

14. GLOSSARY

- ***Data model:*** It is an organisation of the real world entities, restrictions on them, & the relationships between the objects.
- ***Directed acyclic graph (DAG):*** It is a directed graph with no directed cycles. It is formed by a collection of vertices and directed edges, each edge linking one vertex to another.
- ***Inheritance:*** Inheritance is defined as the process where the behaviour and properties of parent object are inherited by the child object.
- ***Multiple inheritance:*** The capability of class to inherit the variables and methods from numerous superclasses is known as Multiple inheritance.
- ***Non-leaf classes:*** Non-leaf classes are abstract and can have further subclasses (child classes).
- ***Persistent programming language:*** A programming language which is expanded by means of constructs to manage persistent data is known as a persistent programming language.
- ***Relationships:*** Relationships permit objects to consider each other and effect in networks of inter-related objects.

15. TERMINAL QUESTIONS

1. Explain various architectural approaches of OODBMS.
2. Illustrate the concept of object oriented data model.
3. Discuss the concept of Type Hierarchies and Inheritance with example. Also illustrate multiple inheritance.
4. What is persistent programming language? How can it be differentiated with embedded SQL? Illustrate.
5. Discuss the declaration of interfaces and classes. Also illustrate how to implement one or more interfaces by a class.

16. ANSWERS

Self-Assessment Questions

1. Objects
2. False
3. Page server
4. True
5. Value
6. True
7. Encapsulation
8. False
9. False
10. State
11. Abstract
12. Relationships
13. False
14. Four
15. Instance
16. Inheritance
17. Colon
18. True
19. isa

20. True
21. Persistent programming language
22. True

Terminal Questions

1. The various architectural approaches of OODBMS include Distributed Client - Server Approach, Data Access Mechanism, Object Clustering, and Heterogeneous Operation. Refer Section 3 for more details.
2. Object oriented data model include various object oriented concepts such as attributes & methods, objects & objects identifiers, class hierarchy & inheritance. Refer Section 6 for more details.
3. Type hierarchy allows the inheritance of both attributes & methods. The capability of class to inherit the variables and methods from numerous superclasses is known as Multiple inheritance. Refer Section 11 for more details.
4. A programming language which is extended with constructs to manage persistent data is known as a persistent programming language. Refer Section 12 for more details.
5. An interface is used to illustrate the behaviour or ability of a class without performing to a specific implementation. Refer Section 10 for more details.

17. REFERENCES

References:

- Prabhu, C.S.R. (2005), *Object Oriented Database Systems,* (2nd Ed.), PHI Learning Pvt. Ltd.
- Khoshafian S. (1993), *Object Oriented databases,(*1stEd.), John Wiley.

E-references

- http://fria.fri.uniza.sk/~kmat/dbs/oodbs/OODBS1b.htm.
- www.cs.cityu.edu.hk/~jfong/cs3462/Lectures/Lecture9.ppt.