Unit 3
Emerging Database Management System Technologies
By
Bhupendra Singh Saud

Overview of Object Oriented Concepts

Object oriented databases try to maintain a direct correspondence between real-world and database objects so that the objects do not lose their integrity and identity and can easily be identified and operated upon. Object has two compounds and they are:

- State (value) and
- Behavior (operations)

In an object-oriented database, objects may have an object structure of arbitrary complexity in order to contain all of the necessary information that is required describes the object. On the contrary in traditional database systems, information about a complex object is often scattered over many relations or records leading to loss of direct correspondence between a real-world object and its database representation. The internal structure of an object in OOPLs includes the specification of instance variables which holds the value that defines the internal state of the object. An instance variable is similar to the concept of an attribute except for the instance variables may be encapsulated within the object and thus are not necessarily visible to external users. Some OO models insist that all operations a user can apply to an object must be predefined. This forces to a complete encapsulation of objects.

Why we need object oriented database?

We need object oriented database due to mainly following two reasons:

- Application which required modeling of complex objects:
  Traditional data models and systems have been quite successful in developing the database technologies required for many traditional business database applications. However, they have certain shortcomings when more complex database applications must be designed and implemented—for example, databases for engineering design and manufacturing (CAD/CAM and CIM1), scientific experiments, telecommunications, geographic information systems, and multimedia. These newer applications have requirements and characteristics that differ from

those of traditional business applications, such as more complex structures for stored objects; the need for new data types for storing images, videos, or large textual items; longer-duration transactions; and the need to define nonstandard application-specific operations.
- Vast increase in the use of object-oriented programming languages for developing software applications.
- To maintain a direct correspondence between real-world and database objects

Classes and objects

A class is understood as an entity that has a well-defined role in the application domain. An object is referred to as a particular instance of a class. An object has structure or state (variables) and methods (behavior/operations). An object is described by four characters and they are:

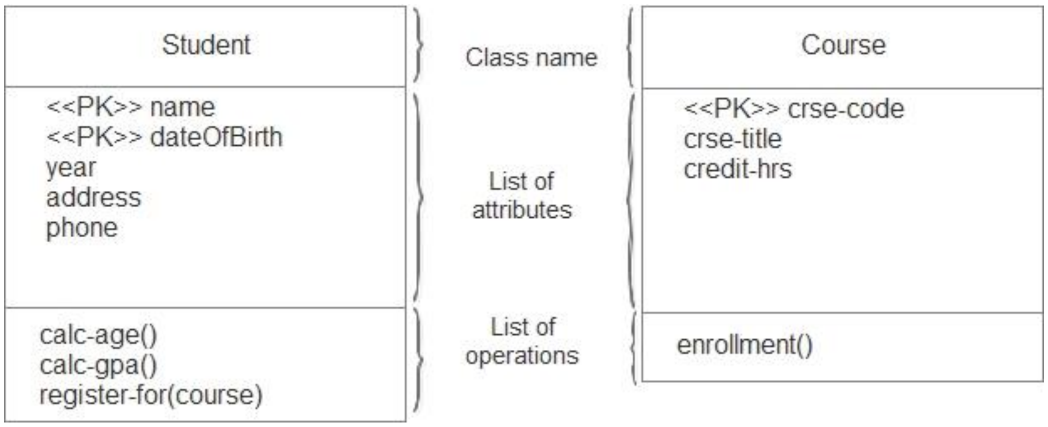
- **Identifier:** A system-wide unique id for an object.
- **Name**: An object may also have a unique name in database which is optional.
- **Lifetime:** It determines if the object is persistent or transient.
- **Structure:** The construction of objects using type constructors.

Object Oriented Database Model

This model represents an entity as a class. A class captures the attributes as well as the behavior. Instances of the class are called objects. Within an object, the class attributes take on specific values which distinguish one object from another. Object Oriented Examples:

- **Class:** Cat.
- **Attributes**: Color, weight, breed.
- **Behavior:** Scratches, sleeps, purrs.

An instance of the cat class is an object with specific attributes.



[TU Question]:- What are the advantages and disadvantages of OODBMS?

Advantages of OODBMS

- Easier Design-Reflect Applications
- Modularity and Reusability
- Incremental refinement and abstraction
- Multiple inheritance
- Support for multiple version and Alternatives
- Designer can specify the structure of objects and their behavior (methods).
- Better interaction with object-oriented languages such as Java and C++
- Definition of complex and user-defined types.
- Encapsulation of operations and user-defined methods.

Disadvantages of OODBMSs

There are following disadvantages of OODBMSs:

- **Lack of universal data model:** There is no universally agreed data model for an OODBMS, and most models lack a theoretical foundation. This .disadvantage is seen as a significant drawback, and is comparable to pre-relational systems.
- **Lack of experience:** In comparison to RDBMSs the use of OODBMS is still relatively limited. This means that we do not yet have the level of experience that we have with traditional systems. OODBMSs are still very much geared towards the programmer, rather than the naïve end-user. Also there is a resistance to the acceptance of the technology. While the OODBMS is limited to a small niche market, this problem will continue to exist
- **Lack of standards:** There is a general lack of standards of OODBMSs. We have already mentioned that there is not universally agreed data model. Similarly, there is no standard object-oriented query language.
- **Competition:** Perhaps one of the most significant issues that face OODBMS vendors is the competition posed by the RDBMS and the emerging ORDBMS products. These products have an established user base with significant experience available. SQL is an approved standard and the relational data model has a solid theoretical formation and relational products have many supporting tools to help .both end-users and developers.
- **Query optimization compromises encapsulations:** Query optimization requires. An understanding of the underlying implementation to access the database efficiently. However, this compromises the concept of incrassation.

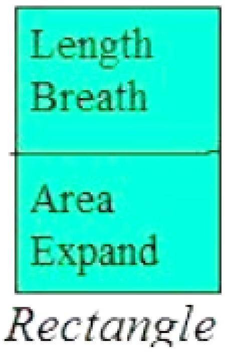
- **Locking at object level may impact performance** Many OODBMSs use locking as the basis for concurrency control protocol. However, if locking is applied at the object level, locking of an inheritance hierarchy may be problematic, as well as impacting performance.
- **Complexity:** The increased functionality provided by the OODBMS (such as the illusion of a single-level storage model, pointer sizzling, version management, and schema evolution--makes the system more complex than that of traditional DBMSs. In complexity leads to products that are more expensive and more difficult to use.
- **Lack of support for views:** Currently, most OODBMSs do not provide a view mechanism, which, as we have seen previously, provides many advantages such as data independence, security, reduced complexity, and customization.
- **Lack of support for security:** Currently, OODBMSs do not provide adequate security mechanisms. The user cannot grant access rights on individual objects or classes.

Features of object oriented (O-O) Databases

- Object oriented databases store persistent objects permanently on secondary storage, and allow the sharing of these objects among multiple programs and applications.
- Object oriented databases provide a unique system-generated object identifier for each object. Object oriented databases maintain a direct correspondence between real-world and database objects so that objects don't lose their integrity and identify and can be easily be identified and operated upon.
- In Object Oriented databases, objects can be very complex in order to contain all significant information that may be required to describe the object completely.
- Object oriented databases allow us to store both the class and state of an object between programs. They take the responsibility for maintaining the links between stored object behavior and state away from the programmer, and manage objects outside of programs with their public and private elements intact. They also simplify the whole process of rendering objects persistent by performing such tasks invisibly.

State and Behavior

Object normally have property and some behavior. State of an object is the values of attributes and behavior of object means operations that operates on attributes of given object.
Example: for the rectangle object,



Length, breadth acts as states of rectangle and calculating area of rectangle act as behavior of rectangle object.

Type Constructors

Type constructor is the collection of multiple similar basic type under a common name. It determines how the object is constructed. The type constructors can be used to define the data structures for an object oriented database schema. The three most basic constructors are **atom, tuple,** and **set**. Other commonly used constructors include **list, bag,** and **array.**

The type constructors set, list, array, and bag are called collection types (or bulk types), and to distinguish them from basic types and tuple types. Here, the state of the object will be a collection of objects that may be unordered or ordered.

Kinds of Type constructor:

- **Atom:** says that an object is storing atomic values.
  e.g.: “Aarav“.
- **Set:** set of values of same type with duplication allowed.
  e.g.: {123,456,123}.
- **Bag:** set with no duplicate items.
  e.g.: {123,456,678}
- **List:** ordered collection of items of the same type with infinite size.
  e.g.: [123,456,678]
- **Array:** similar to list but fixed size.

[TU Question]:- What is OID? How persistent objects are maintained in OO Database?
[TU Question]:- What is the difference between persistent and transient objects? How persistence is handled in typical OO database systems?

OIDs (Object Identifiers)

When a program terminates then all the objects that is in the primary memory are lost. These kind of objects called **transient object**. They do not live forever.

However, OO allows us to store objects permanently in the database, these objects are called **persistent objects**. Because they persist the beyond the life of program.

OIDs is the mechanism to refer to persistent objects. An ODMS provides a unique identity to each independent object stored in the database. This unique identity is typically implemented via a unique, system-generated object identifier (OID). The value of an OID is not visible to the external user, but is used internally by the system to identify each object uniquely and to create and manage inter-object references.

The main property required of an OID is that it be immutable; that is, the OID value of a particular object should not change. This preserves the identity of the real-world object being 4rdrepresented.

Example: In Figure below, the attributes that refer to other objects—such as **Dept** of EMPLOYEE or **Projects** of DEPARTMENT are basically OIDs that serve as references to other objects to represent relationships among the objects. For example, the attribute **Dept** of EMPLOYEE is of type DEPARTMENT, and hence is used to refer to a specific DEPARTMENT object (the DEPARTMENT object where the employee works). The value of such an attribute would be an OID for a specific DEPARTMENT object.

```
Define type EMPLOYEE
        Tuple (Fname: string;
        Minit: char;
        Lname: string;
        Ssn: string;
        Birth_date: DATE;
        Address: string;
        Sex: char;
        Salary: float;
        Supervisor: EMPLOYEE;
        Dept: DEPARTMENT ;);

Define type DATE
        Tuple (Year: integer;
        Month: integer;
        Day: integer ;);

Define type DEPARTMENT
        Tuple (Dname: string;
        Dnumber: integer;
```

```
                    Mgr: tuple (Manager: EMPLOYEE;
                    Locations: set (string);
                    Start_date: DATE ;);
                    Employees: set (EMPLOYEE);
                    Projects: set (PROJECT) ;);
```

Fig: Specifying the object types EMPLOYEE, DATE, and DEPARTMENT using type constructors.

Similarities and dissimilarities of Objects and Literals

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

Every object must have an immutable OID, whereas a literal value has no OID and its value just stands for itself. Thus, a literal value is typically stored within an object and cannot be referenced from other objects. An object has five aspects: **identifier, name, lifetime, structure,** and **creation**.

- The object identifier is a unique system-wide identifier (or Object_id). Every object must have an object identifier.
- Some objects may optionally be given a unique name within a particular ODMS—this name can be used to locate the object, and the system should return the object given that name.
- The lifetime of an object specifies whether it is a persistent object (that is, a database object) or transient object (that is, an object in an executing program that disappears after the program terminates). Lifetimes are independent of types—that is, some objects of a particular type may be transient whereas others may be persistent.
- The structure of an object specifies how the object is constructed by using the type constructors. The structure specifies whether an object is atomic or not. An atomic object refers to a single object that follows a user-defined type, such as Employee or Department. If an object is not atomic, then it will be composed of other objects.
- Object creation refers to the manner in which an object can be created. This is typically accomplished via an operation new for a special Object_Factory interface.

In the object model, a literal is a value that does not have an object identifier. However, the value may have a simple or complex structure. There are three types of literals: **atomic, structured,** and **collection.**

- **Atomic literals** correspond to the values of basic data types and are predefined. The basic data types of the object model include long, short, and unsigned integer numbers, regular and double precision floating point numbers, Boolean values, single characters, character strings, and enumeration types.

- **Structured literals** correspond roughly to values that are constructed using the tuple constructor. The built-in structured literals include Date, Interval, Time, and Timestamp. User-defined structures are created using the STRUCT keyword in ODL, as in the C and C++ programming languages.
- **Collection literals** specify a literal value that is a collection of objects or values but the collection itself does not have an Object_id. The collections in the object model can be defined by the type generators set<T>, bag<T>, list<T>, and array<T>, where T is the type of objects or values in the collection.

[TU Question]:- What is the difference between structured and unstructured complex object? Differentiate identical versus equal objects with examples.

Complex objects (Object structure)

It means there is no restriction in the structure in the object. In ODBs, the value of a complex object can be constructed from other objects. Each object is represented by triple. Complex objects are built from simpler ones by applying constructors to them. The simplest 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: tuples, sets, bags, lists, and arrays are examples.

An object is defined by a triple (OID, type constructor, state) or (i, c, v) where OID is the unique object identifier, type constructor is its type (such as atom, tuple, set, list, array, bag, etc.) and state is its actual value.

Example:

- (i1, atom, 'John')
- (i2, atom, 30)
- (i3, atom, 'Mary')
- (i4, atom, 'Mark')
- (i5, atom 'Vicki')
- (i6, tuple, [Name: i1, Age: i3])
- (i7, set, {i4, i5})
- (i8, tuple, [Name: i3, Friends: i7])
- (i9, set, {i6, i8})

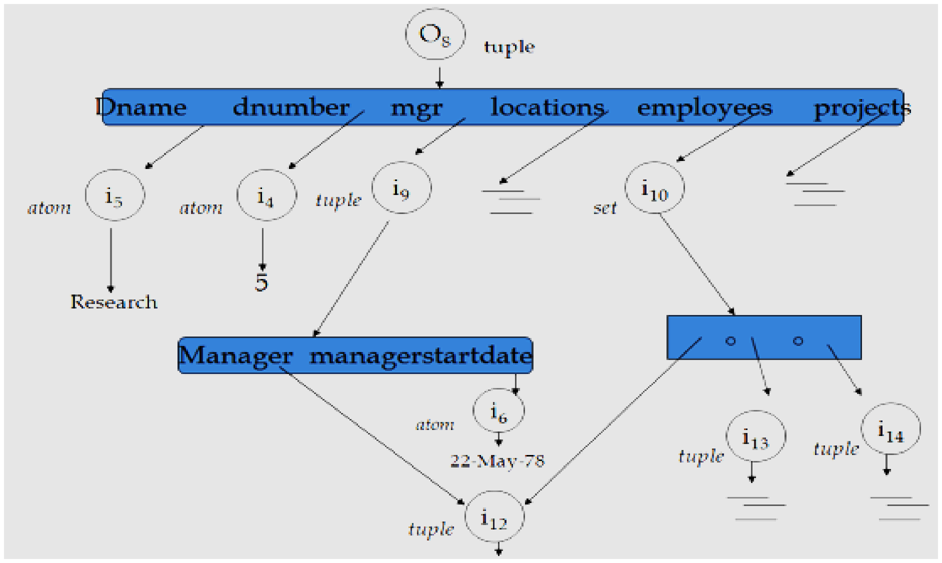


Fig: Structure of Complex object

There are two types of complex objects in object oriented database system which are:

1. Structured complex object and
2. Unstructured complex object

1. Structured Complex object

Structured complex object is defined by repeated application of the type constructors provided by the OODBMS. Simply structured complex objects are constructed by using type constructors (set, atom, tuple etc.). Hence, the object structure is defined and known to the OODBMS. The OODBMS also defines methods or operations on it.

Two types of reference semantics (**ownership semantics** and **reference semantics**) exist between a complex object and its components at each level.

- **Ownership semantics** applies when the sub-objects of a complex object are encapsulated within the complex object and are hence considered part of the complex object. It is also called is-part-of or is-component-of relationship. e.g., "Arjun" atomic value owned by employee. Means that 'Arjun' is dependent on owner.
- **Reference semantics** applies when the components of the complex object are themselves independent objects but may be referenced from the complex object. It is also called is-associated-with relationship. e.g., Department referenced by the employee object.

2. Unstructured complex objects

It is a data type provided by a DBMS and permits the storage and retrieval of large objects that are needed by the database application. These objects are unstructured in the sense that the DBMS does not know what their structure is, only the application programs that uses them can interpret

their meaning. These objects are considered complex because they require large area of storage and are not part of the standard data types provided by traditional DBMSs.
Typical examples of such objects are **bitmap images** and **long text strings** (such as documents); they are also known as binary large objects, or BLOBs for short. Character strings are also known as character large objects, or CLOBs for short.

Identical vs. Equal objects

Two objects are said to have **identical states** (deep equality) if the graphs representing their states are identical in every respect, including the OIDs at every level.

Two objects are said to have **equal states** (shallow equality) if the graphs representing their states are same, including all the corresponding atomic values. However, some corresponding internal nodes in the two graphs may have objects with different OIDs.

Example: This example illustrates the difference between the two definitions for comparing object states for equality.

$o_1 = (i_1, \text{tuple}, <a_1:i_4, a_2:i_6>)$
$o_2 = (i_2, \text{tuple}, <a_1:i_5, a_2:i_6>)$
$o_3 = (i_3, \text{tuple}, <a_1:i_4, a_2:i_6>)$
$o_4 = (i_4, \text{atom}, 10)$
$o_5 = (i_5, \text{atom}, 10)$
$o_6 = (i_6, \text{atom}, 20)$

In this example, the objects o_1 and o_2 have **equal states** (shallow equality), since their states at the atomic level are the same but the values are reached through distinct objects o_4 and o_5.

However, the objects o_1 and o_3 have **identical states** (deep equality), even though the objects themselves are not because they have distinct OIDs. Similarly, although the states of o_4 and o_5 are identical, the actual objects o_4 and o_5 are equal but not identical, because they have distinct OIDs.

Encapsulation of Operations, Methods, and Persistence

Encapsulation means that an object contains both the data structure and the set of operations that can be used to manipulate it. Often cases, adopting encapsulation hides the implementation from the users do not necessarily have to know the detail of it.

Encapsulation is related to the concepts of abstract data types and information hiding in programming languages. Here the main idea is to define the behavior of a type of object based on the operations that can be externally applied to objects of that type. The internal structure of the object is hidden, and the object is only accessible through a number of predefined operations. Some operations may be used to create or destroy objects; other operations may update the object value and other may be used to retrieve parts of the object value or to apply some calculations to the object value.

The external users of the object are only made aware of the interface of the object, which defines the names and arguments of each operation. The implementation of the object is hidden from the external users; it includes the definition of the internal data structure of the object and the implementation of the operations that access these structures.

In object oriented - OO terminology, the interface part of each operation is called the signature, and the operation implementation is called a method. A method is invoked by sending a message to the object to execute the corresponding method.

Not all objects are meant to be stored permanently in the database. Transient objects exist in the executing program and disappear once the program terminates.

Persistent objects are stored in the database and persist after program terminates. The typical mechanism for persistence involves giving an object a unique persistent name through which it can be retrieved.

In traditional database models and systems, this concept was not applied, since it is usual to make the structure of database objects visible to users and external programs.

Inheritance

Inheritance is deriving objects from existing objects. The derived objects inherit properties from their parent object. Parent objects are those objects from which other objects are derived. Inheritance is a way of reusing the existing code.

Polymorphism

Polymorphism concept allows the same operator name or symbol to be bound to two or more different implementation of the operator, depending on the type of objects to which the operator is applied.

Multiple Inheritance and Selective Inheritance

Multiple inheritance in a type hierarchy occurs when a certain subtype T is a subtype of two (or more) types and hence inherits the functions (attributes and methods) of both super types.
For example, we may create a subtype ENGINEERING_MANAGER that is a subtype of both MANAGER and ENGINEER. This leads to the creation of a type lattice rather than a type hierarchy.

One problem that can occur with multiple inheritance is that the super types from which the subtype inherits may have different functions of the same name, creating an ambiguity.

If a function is inherited from some common super type, then it is inherited only once. In such a case, there is no ambiguity; the problem only arises if the functions are distinct in the two super types. Some languages do not allow multiple inheritance.

Selective inheritance occurs when a subtype inherits only some of the functions of a super type. Other functions are not inherited. This mechanism is not typically provide in OO database system.

Versions and configurations

Versions:

- Ability to maintain several versions of an object
- Commonly found in many software engineering and concurrent engineering environments
- Merging and reconciliation of various versions is left to the application program
- Some systems maintain a version graph

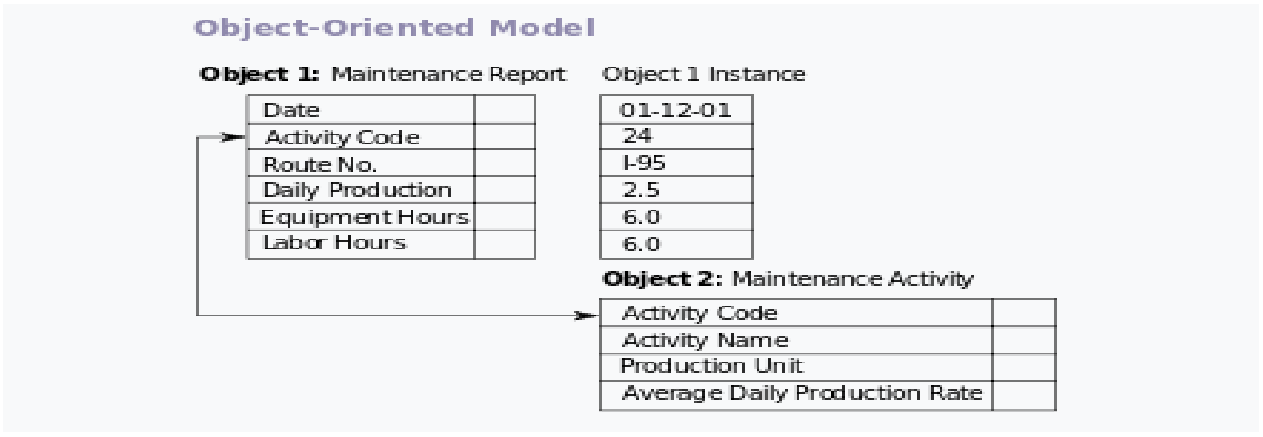
Configuration:

- A configuration is a collection compatible versions of modules of a software system (a version per module)

Object Relational Database concepts

Object relational DBMSs (ORDBMSs) are also called object-relational or enhanced systems. These systems emerged as a way of enhancing the capabilities of relational DBMSs (RDBMSs) with some of the features that appeared in object oriented DBMSs (ODBMSs). Object-relational database systems provide a convenient migration path for users of relational database who wish to use object-oriented features

An object-relational database (ORD), or object-relational database management system (ORDBMS), is a database management system (DBMS) similar to a relational database, but with an object-oriented database model: objects, classes and inheritance are directly supported in database schemas and in the query language. In addition, just as with pure relational systems, it supports extension of the data model with custom data-types and methods.



Example of an object-oriented database model

An object-relational database can be said to provide a middle ground between relational databases and object-oriented databases (object database).

Comparison of RDBs vs. ORDBs

- Very easy to compare because both are based on relational model.
- An RDB does not support abstract data types (ADT), all attribute values must be atomic and relations must be in first normal form (flat relation).
- An ORDB supports ADTs, attributes can be multivalued, and does not require first normal form.
- The underlying basic data structures of RDBs are much simpler but less versatile than ORDBs.
- ORDBs support complex data whereas RDBs don't.
- ORDBs support wide range of applications.
- RDBs have only one construct i.e. Relation, whereas the type system of ODBs is much richer and complex.
- RDBs require primary keys and foreign keys for implementing relationships, ODBs simply don't.
- ODBs support complex data whereas RDBs don't.
- ODBs support wide range of applications.
- ODBs are much faster than RDBs but are less mature to handle large volumes of data.
- There is more acceptance and domination of RDBs in the market than that for ODBs.
- Both support ADTs, collections, OIDs, and inheritance, though philosophically quite different.
- ORDBs extended RDBs whereas ODBs add persistence and database capabilities to OO languages.
- Both support query languages for manipulating collections and nested and complex data.
- SQL3 is inspired from OO concepts and is converging towards OQL (object query language).
- ORDBs carries all the benefits of RDBs, whereas ODBs are less benefited from the technology of RDBs.

Overview of SQL and its object-relational features

SQL stands for structured query language. It is a database query language used to communicate and manage data stored on the database. SQL was specified in 1970s. It was enhanced substantially

in 1989 and 1992. A new standard called SQL3 added object-oriented features. A subset of SQL3 standard, now known as SQL-99 has been approved.
The following are some of the object database features that have been included in SQL:

- Some type constructors have been added to specify complex objects. These include the row type, which corresponds to the tuple (or struct) constructor. An array type for specifying collections is also provided. Other collection type constructors, such as set, list, and bag constructors, were not part of the original SQL/Object specifications but were later included in the standard.
- A mechanism for specifying object identity through the use of reference type is included.
- Encapsulation of operations is provided through the mechanism of user defined types (UDTs) that may include operations as part of their declaration. These are somewhat similar to the concept of abstract data types that were developed in programming languages. In addition, the concept of user defined routines (UDRs) allows the definition of general methods (operations).
- Inheritance mechanisms are provided using the keyword UNDER.

[TU Model Question):- Describe the steps of the algorithm for object database design by EER-to-OO mapping.

Mapping an EER Schema to an ODB Schema

It is relatively straightforward to design the type declarations of object classes for an ODBMS from an EER schema that contains neither categories nor n-ary relationships with n > 2. However, the operations of classes are not specified in the EER diagram and must be added to the class declarations after the structural mapping is completed.
The outline of the mapping from EER to ODL is as follows:
Step 1. Create an ODL class for each EER entity type or subclass. The type of the ODL class should include all the attributes of the EER class. Multivalued attributes are typically declared by using the **set, bag, or list** constructors. If the values of the multivalued attribute for an object should be ordered, the list constructor is chosen; if duplicates are allowed, the bag constructor should be chosen; otherwise, the set constructor is chosen. Composite attributes are mapped into a tuple constructor (by using a struct declaration in ODL). Declare an extent for each class, and specify any key attributes as keys of the extent.

Step 2. Add relationship properties or reference attributes for each binary relationship into the ODL classes that participate in the relationship. These may be created in one or both directions. If a binary relationship is represented by references in both directions, declare the references to be relationship properties that are inverses of one another, if such a facility exists. If a binary relationship is represented by a reference in only one direction, declare the reference to be an attribute in the referencing class whose type is the referenced class name. Depending on the cardinality ratio of the binary relationship, the relationship properties or reference attributes may be single-valued or collection types. They will be single-valued for binary relationships in the 1:1 or N:1 directions; they are collection types (set-valued or list-valued) for relationships in the 1:N or M:N direction. If relationship attributes exist, a tuple constructor can be used to create a structure of the form < reference, relationship attributes>, which may be included instead of the reference attribute.

Step 3. A constructor method should include program code that checks any constraints that must hold when a new object is created. A destructor method should check any constraints that may be violated when an object is deleted. Other methods should include any further constraint checks that are relevant.

Step 4. An ODL class that corresponds to a subclass in the EER schema inherits (via extends) the type and methods of its superclass in the ODL schema.

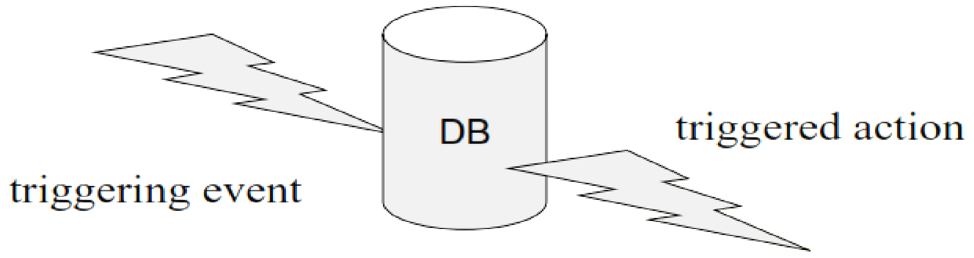
Step 5. Weak entity types can be mapped in the same way as regular entity types. An alternative mapping is possible for weak entity types that do not participate in any relationships except their identifying relationship; these can be mapped as though they were composite multivalued attributes of the owner entity type, by using the set < struct <... >> or list <struct <... >> constructors. The attributes of the weak entity are included in the struct <... > construct, which corresponds to a tuple constructor.

Step 6. Categories (union types) in an EER schema are difficult to map to ODL. It is possible to create a mapping similar to the EER-to-relational mapping by declaring a class to represent the category and defining 1:1 relationships between the category and each of its super classes. Another option is to use a union type, if it is available

Step 7. An n-ary relationship with degree n > 2 can be mapped into a separate class, with appropriate references to each participating class. These references are based on mapping a 1:N relationship from each class that represents a participating entity type to the class that represents the n-ary relationship. An M:N binary relationship especially if it contains relationship attributes, may also use this mapping option, if desired.

Active database concepts

A **trigger** is a procedure that is automatically invoked by the DBMS in response to specified changes to the database, and is typically specified by the DBA. A database that has a set of associated triggers is called an **active database**.



Generalized Model for Active Databases and Oracle Triggers

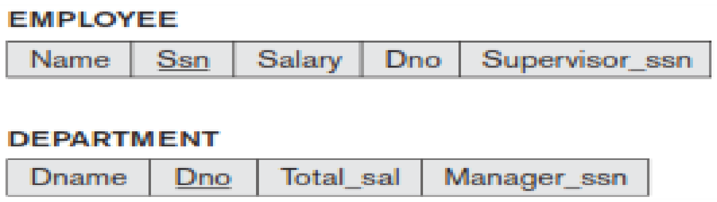
The model that has been used to specify active database rules is referred to as the Event-Condition-Action (ECA) model. A rule in the ECA model has three components:

1. **The event(s) that triggers the rule:**
   These events are usually database update operations that are explicitly applied to the database. However, in the general model, they could also be temporal events or other kinds of external events.
2. **The condition that determines whether the rule action should be executed**:
   Once the triggering event has occurred, an optional condition may be evaluated. If no condition is specified, the action will be executed once the event occurs. If a condition is specified, it is first evaluated, and only if it evaluates to true will the rule action be executed.
3. **The action to be taken:**
   The action is usually a sequence of SQL statements, but it could also be a database transaction or an external program that will be automatically executed.

A syntax summary for specifying triggers in the Oracle system

```
CREATE TRIGGER <trigger name>
      AFTER / BEFORE <triggering events>
            ON <table name>
                  [FOR EACH ROW]
                        [WHEN <condition>]
                              <Trigger actions>;
```

Example: For the following schema diagram;



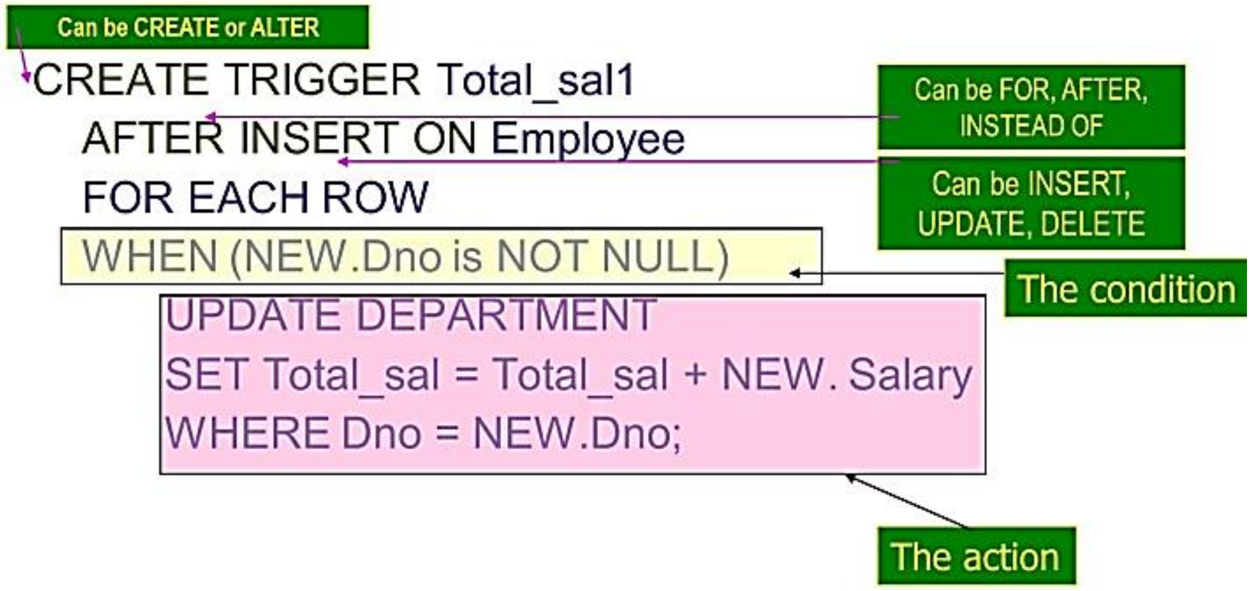
EMPLOYEE

Name	Ssn	Salary	Dno	Supervisor_ssn

DEPARTMENT

Dname	Dno	Total_sal	Manager_ssn

Suppose, we need to compute Total_sal if the new employee is immediately assigned to a department—that is, if the value of the Dno attribute for the new employee tuple is not NULL



```
CREATE TRIGGER Total_sal1
    AFTER INSERT ON Employee
    FOR EACH ROW
    WHEN (NEW.Dno is NOT NULL)
        UPDATE DEPARTMENT
        SET Total_sal = Total_sal + NEW. Salary
        WHERE Dno = NEW.Dno;
```

Design and Implement Issues for Active Databases

In addition to creating rules, an active database allows users to **activate**, **deactivate**, **drop,** and **group** rules by refereeing their rules

- ❖ A **deactivated rule** will not be triggered by the triggering event. This feature allows users to selectively deactivate rules for certain periods of time when they are not needed
- ❖ The **active command** will make the rule active again
- ❖ The drop command deletes the rule from the system
- ❖ Another option is to group rules into named **rule sets**, so the whole set of rules could be activated, deactivated, or dropped. It is also useful to have a command that can trigger a rule or rule set via an explicit **PROCESS RULE** command issued by the user.
- ❖ Whether the triggered action should be executed **before**, **after**, or **concurrently** with the triggering event. Whether the action being executed should be considered as a **separate transaction** or whether it should be part of the same transaction that triggered the rule

There are three main possibilities for *condition evaluation* (also known as *rule consideration*)

- ❖ **Immediate consideration:** The condition is evaluated as part of the same transaction as the triggering event, and is evaluated immediately. This case can be further categorized into three options:
  - ■ Evaluate the condition before executing the triggering event
  - ■ Evaluate the condition after executing the triggering event
  - ■ Evaluate the condition instead of executing the triggering event
- ❖ **Deferred consideration**. The condition is evaluated at the end of the transaction that included the triggering event. In this case, there could be many triggered rules waiting to have their conditions evaluated.
- ❖ **Detached consideration.** The condition is evaluated as a separate transaction, spawned from the triggering transaction.

Potential Applications for Active Databases

- ❖ **Notification** – allow notification of certain conditions that occur
- ❖ **Enforce integrity constraint** – specify the types of events that may cause the constraints to be violated and then evaluating appropriate condition that check whether the constraints are actually violated by the event or not
- ❖ Automatic **maintenance of derived data**
- ❖ Maintain the consistency of **materialized views** whenever the base relations are modified
- ❖ Maintain **replicated tables** whenever the master table is modified

Temporal database concepts

Temporal databases, in the broadest sense, encompass all database applications that require some aspect of time when organizing their information. It encompass all DB applications that require some aspect of time when organizing their information. A **temporal** database is a database that has certain features that support time-sensitive status for entries. Where some databases are considered current databases and only support factual data considered valid at the time of use, a temporal database can establish at what times certain entries are accurate.

Temporal DB applications have been developed since the early days of database usage. However, in creating these applications, it was mainly left to the application developers to discover, design, program, and implement the temporal concepts.

There are many examples of applications where some aspect of time is needed to maintain the information in a DB.

- ***Health care*:** patient histories need to be maintained
- ***Insurance*:** claims and accident histories are required

- ***Finance*:** stock price histories need to be maintained.
- ***Personnel management*:** salary and position history need to be maintained
- ***Banking*:** credit histories

Valid Time and Transaction Time Dimensions

Valid time

Given a particular event or fact that is associated with a particular time point or time period in the database, the association may be interpreted to mean different things. The most natural interpretation is that the associated time is the time that the event occurred, or the period during which the fact was considered to be true in the real world. If this interpretation is used, the associated time is often referred to as the **valid time**.

Simply, the **valid time** denotes the time period during which an event occur or a fact is true with respect to the real world. A temporal database using this interpretation is called a **valid time** database.

Example:

Imagine that we come up with a temporal database storing data about the 18th century. The valid time of these facts is somewhere between 1700 and 1799, whereas the transaction time starts when we insert the facts into the database, for example, January 21, 1998. Assume we would like to store data about our employees with respect to the real world. Then, the following table could result:

EmpID	Name	Department	Salary	ValidTimeStart	ValidTimeEnd
10	John	Research	11000	1985	1990
10	John	Sales	11000	1990	1993
10	John	Sales	12000	1993	Now
11	Paul	Research	10000	1988	1995
12	George	Research	10500	1991	Now
13	Ringo	Sales	15500	1988	Now

The above valid-time table stores the history of the employees with respect to the real world. The attributes **ValidTimeStart** and **ValidTimeEnd** actually represent a time interval which is closed at its lower and open at its upper bound. Thus, we see that during the time period [1985 – 1990], employee John was working in the research department, having a salary of 11000. Then he changed to the sales department, still earning 11000. In 1993, he got a salary raise to 12000. Note that it is now possible to store information about past states. We see that Paul was employed from 1988 until 1995. In the corresponding non-temporal table, this information was (physically) deleted when Paul left the company.

Transaction Time

However, a different interpretation can be used, where the associated time refers to the time when the information was actually stored in the database; that is, it is the value of the system time clock when the information is valid in the system. In this case, the associated time is called the **transaction time**. A temporal database using this interpretation is called a **transaction time database.**

Simply, the **Transaction time** is the time period during which a fact is stored in the database.

Example:

EmpID	Name	Department	Salary	TransactionTimeStart	TransactionTimeEnd
10	John	Research	11000	1985-10-2, 10:02:33	1990-10-2, 11:33:04
10	John	Sales	11000	1990-04-11, 05:04:33	1993-05-11, 06:22:55
10	John	Sales	12000	1993-7-30, 5:33:05	Now
11	Paul	Research	10000	1988-5-23, 7:23:34	1995-8-30, 5:33:55
12	George	Research	10500	1991-8-23, 12:44:34	Now
13	Ringo	Sales	15500	1988-7-24, 11:23:55	Now

The above valid-time table stores the history of the employees with respect to the real world. The attributes **TransactionTimeStart** and **TransactionTimeEnd** actually represent a transaction time interval which is closed at its lower and open at its upper bound.

Bitemporal Database

In some applications, only one of the dimensions is needed and in other cases both time dimensions are required, in which case the temporal database is called a **bitemporal database.** It uses both valid time and transaction time in a single database.

Deductive databases

A deductive database system typically specify rules through a **declarative language** – a language in which we specify what to achieve rather than how to achieve it. It is a database system that can make deductions (i.e., conclude additional facts) based on rules and facts stored in the (deductive) database. It is also related to the field of logic programming and the **Prolog** language.

A variation of Prolog called **Datalog** can also be used to define rules declaratively in conjunction with an existing set of relations. A deductive database used two main types of specifications: **facts** and **rules.**

Facts are specified in a manner similar to the way relations are specified, except that it is not necessary to include the attribute names.

Rules are somewhat similar to relational views. They specify virtual relations that are not actually stored but that can be formed from the facts by applying inference mechanisms based on the rule specifications.

The deductive database work based on logic has used Prolog as a starting point. A variation of Prolog called Datalog is used to define rules declaratively in conjunction with an existing set of relations, which are themselves treated as literals in the language.

The notation used in Prolog/Datalog is based on providing predicates with unique names.

A predicate has an implicit meaning, which is suggested by the predicate name, and a fixed number of arguments. The predicate‘s type is determined by its arguments:

- If the arguments are all constant values, the predicate simply states that a certain fact is true
- If the predicate has variables as arguments, it is either considered as a query or as part of a rule or constraint.

[TU Question]:- Describe multimedia database and what are the different types of multimedia data that are available in current systems?

[TU Question]:- Enumerate the limitations of conventional database compared to multimedia database.

Multimedia Databases

Multimedia databases provide features that allow users to store and query different types of multimedia information, which includes **images**, **video clips**, **audio clips**, and **documents.**

Simply, a Multimedia database (MMDB) is a collection of related multimedia data. The multimedia data include one or more primary media data types such as text, images, graphic objects (including drawings, sketches and illustrations) animation sequences, audio and video.

Multimedia databases provide features that allow users to store and query different types of multimedia information, which includes images (such as photos or drawings), video clips (such as movies, newsreels, or home videos), audio clips (such as songs, phone messages, or speeches), and documents (such as books or articles). The main types of database queries that are needed involve locating multimedia sources that contain certain objects of interest. For example, one may want to locate all video clips in a video database that include a certain person, say Michael Jackson. One may also want to retrieve video clips based on certain activities included in them, such as video clips where a soccer goal is scored by a certain player or team.

Advantages of Multimedia database

- It is very user-friendly. It doesn’t take much energy out of the user, in the sense that you can sit and watch the presentation, you can read the text and hear the audio.

- It is multi sensorial. It uses a lot of the user's senses while making use of multimedia, for example hearing, seeing and talking.
- It is integrated and interactive. All the different mediums are integrated through the digitization process. Interactivity is heightened by the possibility of easy feedback.
- It is flexible. Being digital, this media can easily be changed to fit different situations and audiences.
- It can be used for a wide variety of audiences, ranging from one person to a whole group.

Disadvantages

- Information overload. Because it is so easy to use, it can contain too much information at once.
- It takes time to compile. Even though it is flexible, it takes time to put the original draft together.
- It can be expensive. Multimedia makes use of a wide range of resources, which can cost you a large amount of money.
- Too much makes it unpractical. Large files like video and audio has an effect of the time it takes for your presentation to load. Adding too much can mean that you have to use alarger computer to store the files.

Characteristics of different multimedia sources

Image

- Typically stored either in raw form as a set of pixel or cell values, or in compressed form to save space
- Each image can be represented by an m by n grid of cells and each cell contains a pixel value that describe the cell content
- Compression standards, such as GIF or JPEG, use various mathematical transformations to reduce the number of cells stored.

Video

- Typically represented as a sequence of frames, where each frame is a still image
- The video is divided into video segments, where each segment is made up of sequence of contiguous frames that includes the same objects/activities. Each segment is identified by its starting and ending frames
- An indexing technique called frame segment trees has been proposed for video indexing. The index includes both objects and activities
- Videos are also often compressed using standards such as MPEG

Text/Document

- The full text of some article, book, or magazine
- Typically indexed by identifying the keywords that appear in the text and their relative frequencies. Because there are too many keywords when attempting to index, a technique called singular value decomposition (SVD) can be used to reduce the number of keywords
- An indexing technique called telescoping vector trees, or TV-trees can be used to group similar documents together

Audio

- Include stored recorded messages
- Discrete transforms can be used to identify the main characteristics of a certain person's voice in order to have similarity based indexing and retrieval. Futures include loudness, intensity, pitch, and clarity

Types of Multimedia are available in current systems:

DBMSs have been constantly adding to the types of data they support. Today many types of multimedia data are available in current systems.

- **Text:** May be formatted or unformatted. For ease of parsing structured documents, standards like SGML and variations such as HTML are being used.
- **Graphics:** Examples include drawings and illustrations that are encoded using some descriptive standards (e.g. CGM, PICT, postscript).
- **Images:** Includes drawings, photographs, and so forth, encoded in standard formats such as bitmap, JPEG, and MPEG. Compression is built into JPEG and MPEG. These images are not subdivided into components. Hence querying them by content (e.g., find all images containing circles) is nontrivial.
- **Animations:** Temporal sequences of image or graphic data.
- **Video:** A set of temporally sequenced photographic data for presentation at specified rates– for example, 30 frames per second.
- **Structured audio**: A sequence of audio components comprising note, tone, duration, and so forth.
- **Audio**: Sample data generated from aural recordings in a string of bits in digitized form. Analog recordings are typically converted into digital form before storage.
- **Composite or mixed multimedia data:** A combination of multimedia data types such as audio and video which may be physically mixed to yield a new storage format or logically mixed while retaining original types and formats. Composite data also contains additional control information describing how the information should be rendered.

[TU Question]:- Discuss some applications of active database. How do spatial databases differ from regular database?

Spatial Database

A spatial database is a database that is enhanced to store and access spatial data or data that defines a geometric space. The special data stored in the form of co-ordinate form. These data are often associated with geographic locations and features, or constructed features like cities. Data on spatial databases are stored as coordinates, points, lines, polygons and topology. Some spatial databases handle more complex data like three-dimensional objects, topological coverage and linear networks.

The main goal of a spatial database system is the effective and efficient handling of spatial data types in two, three or higher dimensional spaces, and the ability to answer queries taking into consideration the spatial data properties.

A common example of spatial data can be seen in a road map. A road map is a 2-dimensional object that contains points, lines, and polygons that can represent cities, roads, and political boundaries such as states or provinces. A road map is a visualization of geographic information.

Examples of spatial data types are:

- **Point:** characterized by a pair of (x, y) values,
- **Line segment:** characterized by a pair of points,
- **Rectangle:** characterized by its lower-left and upper-right corners,
- **Polygon:** comprised by a set of points, defining its corners.

Examples of spatial datasets

Three types of spatial queries

- **Range query** – Finds the objects of a particular type that are within a particular distance from a given location. For example, finds all hospitals within the Kathmandu city area, or finds all ambulances within five miles of an accident location

- **Nearest neighbor query –** Finds an object of a particular type that is closest to a given location. For example, finds the police car that is closest to a particular location
- **Spatial joins or overlays –** Typically joins the objects of two types based on some spatial condition, such as the objects intersecting or overlapping spatially or being within a certain distance of one another. For example, find all cities that fall on a major highway or finds all homes that are within two miles of a lake

Geographic Information Systems (GIS)

Geographic information systems (GIS) are used to collect, model, store, and analyze information describing physical properties of the geographical world. GIS technology integrates common database operations such as query and statistical analysis with the unique visualization and geographic analysis benefits offered by maps. Map making and geographic analysis are not new, but a GIS performs these tasks better and faster than do the old manual methods. And, before GIS technology, only a few people had the skills necessary to use geographic information to help with decision making and problem solving. Today, Professionals in every field are increasingly aware of the advantages of thinking and working geographically.

The scope of GIS broadly encompasses two types of data:

1. **Spatial data**, originating from maps, digital images, administrative and political boundaries, roads, transportation networks; physical data such as rivers, soil characteristics, climatic regions, land elevations
2. **Non-spatial data**, such as socio-economic data (like census data), economic data, or sales or marketing information.

Components of GIS

A working Geographic Information System seamlessly integrates five key components: hardware, software, data, people, and methods.

Hardware: Hardware includes the computer on which a GIS operates, the monitor on which results are displayed, and a printer for making hard copies of the results.

GIS software: It provides the functions and tools needed to store, analyze, and display geographic information. Key software components include tools for the input and manipulation of geographic information, a database management system (DBMS), tools that support geographic query, analysis, and visualization, and a graphical user interface (GUI) for easy access to tools.

Data: Possibly the most important component of a GIS is the data. A GIS will integrate spatial data with other data resources and can even use a database management system, used by most organizations to organize and maintain their data, to manage spatial data. There are three ways to obtain the data to be used in a GIS. Geographic data and related tabular data can be collected in-house or produced by digitizing images from aerial photographs or published maps.

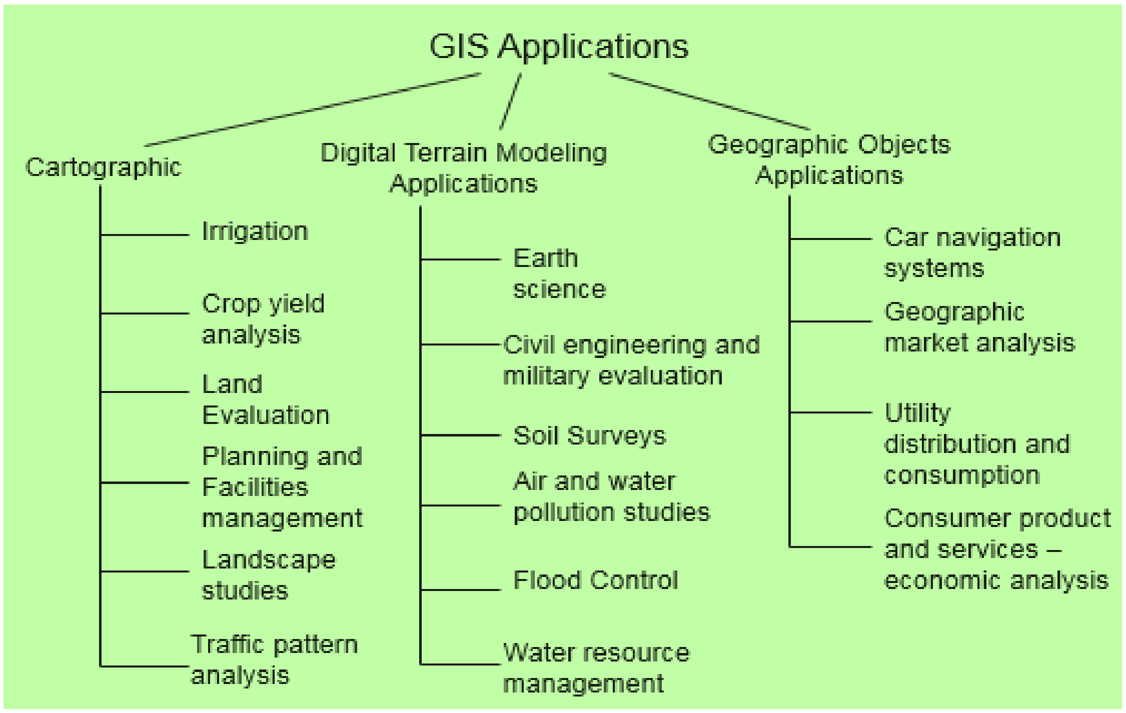
People: GIS users range from technical specialists who design and maintain the system to those who use it to help them perform their everyday work.

Methods: A successful GIS operates according to a well-designed plan and business rules, which are the models and operating practices unique to each organization.

GIS applications

It is possible to divide GISs into three categories:

1. Cartographic applications
2. Digital terrain modeling applications, and
3. Geographic objects applications


GIS Applications
Cartographic
Irrigation
Crop yield analysis
Land Evaluation
Planning and Facilities management
Landscape studies
Traffic pattern analysis
Digital Terrain Modeling Applications
Earth science
Civil engineering and military evaluation
Soil Surveys
Air and water pollution studies
Flood Control
Water resource management
Geographic Objects Applications
Car navigation systems
Geographic market analysis
Utility distribution and consumption
Consumer product and services – economic analysis

In cartographic and terrain modeling applications, variations in spatial attributes are captured – for example, soil characteristics, crop density, and air quality.
In geographic object applications, objects of interest are identified from a physical domain – for example, power plants, electoral districts, property parcels, product distribution districts, and city

landmarks; These objects are related with pertinent application data – for example, power consumption, voting patterns, property sales volumes, product sales volume, and traffic density.
The first two categories of GIS applications require a field-based representation, whereas the third category requires an object-based one. The cartographic approach involves special functions that can include the overlapping of layers of maps to combine attribute data. Digital terrain modeling requires a digital representation of parts of earth's surface using land elevations at sample points that are connected to yield a surface model showing the surface terrain. In object-based geographic applications, additional spatial functions are needed to deal with data.

Data management requirements of GIS

The functional requirements of the GIS applications translate into the following database requirements.

- Data Modeling and Representation.
- Data Analysis
- Data Integration
- Data Capture

1. Data Modeling and Representation

GIS data can be broadly represented in tow formats:

- vector and
- raster

Vector data represents geometric objects such as points, lines, and polygons
Raster data is characterized as an array of points, where each point represents the value of an attribute for a real-world location. Informally, raster images are n-dimensional array where each entry is a unit of the image and represents an attribute. Two-dimensional units are called pixels, while three-dimensional units are called voxels. Three-dimensional elevation data is stored in a raster-based digital elevation model (DEM) format. This model is commonly used for analytical applications such as modeling, map algebra, and more advanced techniques of feature extraction and clustering.

2. Data Analysis

GIS data undergoes various types of analysis for example, in applications such as soil erosion studies, environmental impact studies, or hydrological runoff simulations, data may undergo various types of **geomorphometric analysis** – measurements such as slope values, gradients (the rate of change in altitude), aspect (the compass direction of the gradient), profile convexity (the rate of change of gradient), plan convexity (the convexity of contours and other parameters).

3. Data Integration

GISs must integrate both vector and raster data from a variety of sources. Sometimes edges and regions are inferred from a raster image to form a vector model, or conversely, raster images such as aerial photographs are used to update vector models

4. Data Capture

The first step in developing a spatial database for cartographic modeling is to capture the two-dimensional or three-dimensional geographical information in digital form – a process that is sometimes impeded by source map characteristics such as resolution, type of projection, map scales, cartographic licensing, diversity of measurement techniques, and coordinate system differences. Spatial data can also be captured from remote sensors in satellites such as Landsat, NORA, and Advanced Very High Resolution Radiometer as well as SPOT HRV (High Resolution Visible Range Instrument.

Specific GIS data operations

GIS applications are conducted through the use of special operators such as the following:

- **Interpolation** – derives elevation data
- **Interpretation** – involves the interpretation of operations on terrain data such as editing, smoothing, reducing details, and enhancing.
- **Proximity analysis** – computation of "zones of interest" around objects
- **Raster image processing** – can be divided into (1) map algebra and (2) digital image analysis
- **Analysis of networks** – analysis of networks for segmentation, overlays, and so on

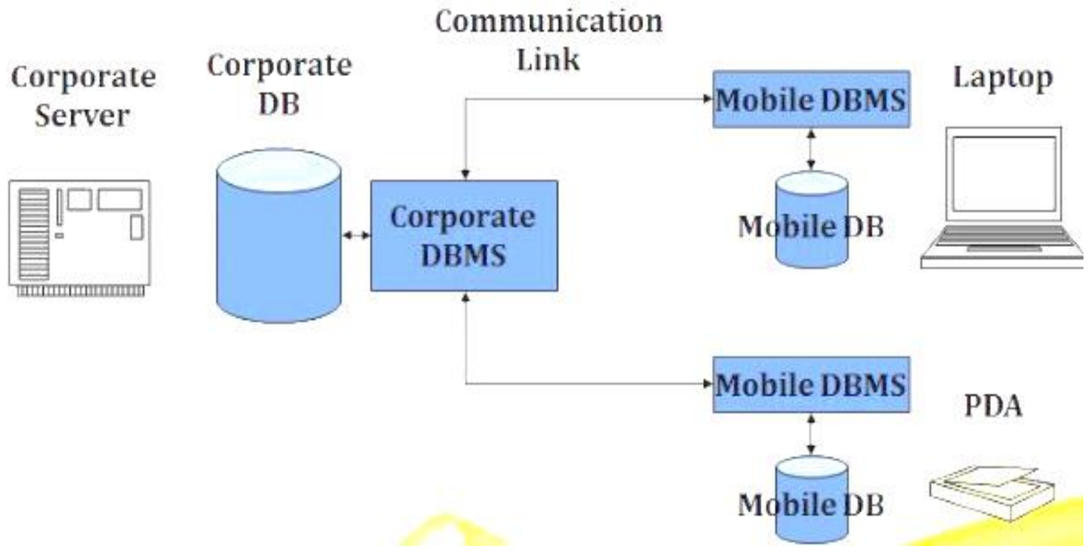
Mobile Databases

Recent advances in portable and wireless technology have led to **mobile computing**, a new dimension in data communication and processing. A mobile database is a database that can be connected to by a mobile computing device over a mobile network. It is portable and physically separate from the corporate database server. But Mobile Database is capable of communicating with that corporate database server from remote sites allowing the sharing of corporate database.

Mobile computing devices (e.g., smartphones and PDAs) store and share data over a mobile network, or a database which is actually stored by the mobile device. A mobile database is a database that resides on a mobile device such as a PDA, a smart phone, or a laptop. Such devices are often limited in resources such as memory, computing power, and battery power.

Mobile computing architecture, characteristics of mobile environments, data management issues. It can also be defined as a system with the following structural and functional properties.

- Distributed system with mobile connectivity (A mode in which a client or a server can establish communication with each other whenever needed)
- Full database system capability
- Complete spatial mobility
- Wireless and wired communication capability



Ability of mobile DBMS

- Communicate with centralized database server through modes such as wireless or Internet access
- Replicate data on centralized database server and mobile device
- Synchronize data on centralized database server and mobile device
- Capture data from various sources such as Internet
- Manage/analyze data on the mobile device
- Create customized mobile applications

[**TU Question: - Describe the characteristics of mobile computing environment in detail]**

[**TU Question: - Explain mobile computing architecture with suitable diagram**]

Mobile Computing Architecture

The general architecture of a mobile platform is a distributed architecture where a number of computers, generally referred to as **Fixed Hosts** and **Base Stations**, are interconnected through a high speed wired network. Fixed hosts are general purpose computers configured to manage mobile units. Base stations function as a gateways to the fixed network for the **Mobile Units**; they are equipped with wireless interfaces and offer network access services of which mobile units are clients.

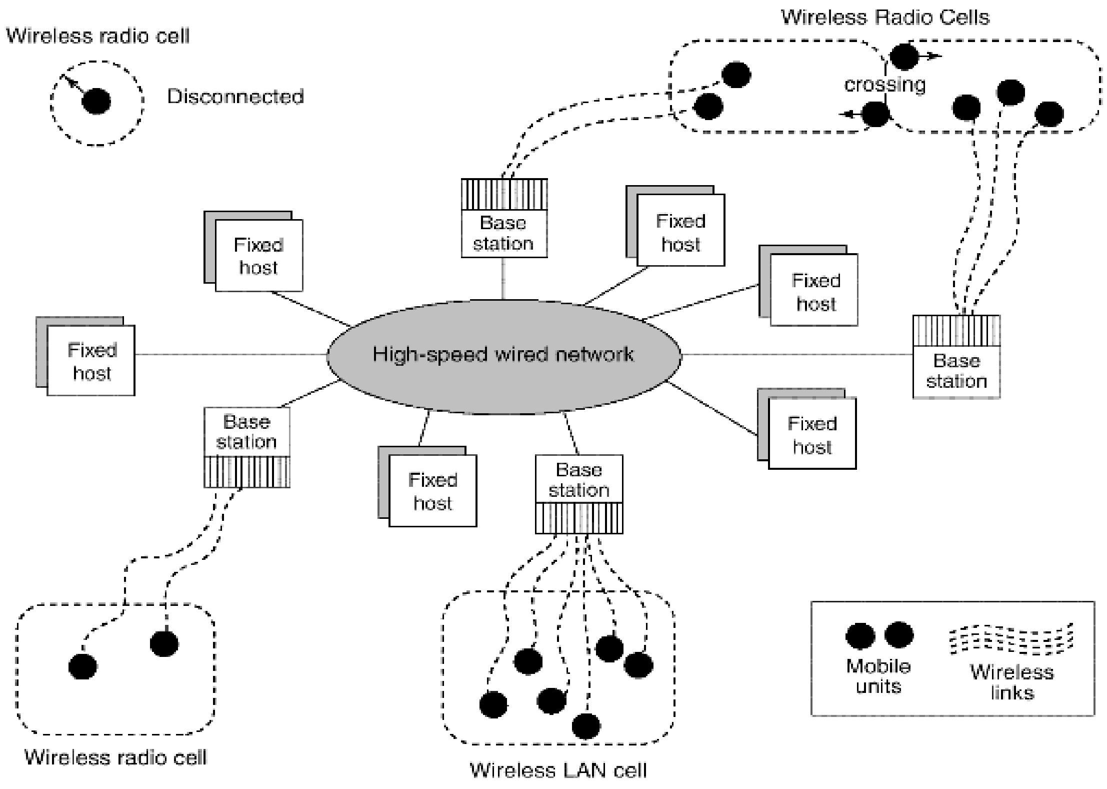


Fig: A general architecture of a mobile platform

Wireless Communications

The wireless medium on which mobile units and base stations communicate have bandwidths significantly lower than those of a wired network. Some wireless access options allow seamless roaming throughout a geographical region (e.g., cellular networks), whereas Wi-Fi networks are localized around a base station; Some wireless networks, such as Wi-Fi and Bluetooth, use unlicensed areas of frequency spectrum, which may cause interference with other appliances, such as cordless telephones; Modern wireless networks can transfer data in units called packets, that are commonly used in wired networks in order to conserve bandwidth; Wireless applications must consider these characteristics when choosing a communication option

Client/Network Relationships

Mobile units can move freely in a **geographic mobility domain**, an area that is circumscribed by wireless network coverage; To manage the mobility of units, the entire geographic mobility domain is divided into one or more smaller domains, called **cells**, each of which is supported by at least one base station; The mobile discipline requires that the movement of mobile units be unrestricted throughout the cells of a geographic mobility domain, while maintaining information **access contiguity** – i.e., movement, especially intercell movement, does not negatively affect the data retrieval process; This architecture is designed for a fixed network, emulating a traditional client-server architecture

Characteristics of Mobile Computing Environments

The characteristics of mobile computing include **high communication latency**, **intermittent wireless connectivity**, **limited battery life**, and **changing client location**

- **Latency** is caused by the processes unique to the wireless medium, such as coding data for wireless transfer, and tracking and filtering wireless signals at the receiver.
- **Intermittent connectivity** can be intentional or unintentional; unintentional disconnections happen in areas where wireless signals cannot reach, e.g., elevator shafts or subway tunnels; Intentional disconnections occur by user intent, e.g., during an airplane takeoff, or when the mobile device is powered down.
- **Battery life** is directly related to battery size, and indirectly related to the mobile device's capabilities and overall efficiency.
- **Client locations** are expected to change, which alters the network topology and may cause their data requirements to change.

  First, servers must keep track of client locations in order to route messages to them efficiently. Second, client data should be stored in network location that minimize the traffic necessary to access it. Keeping data in a fixed location increases access latency if the client moves far away from it.

All these characteristics impact data management, and robust mobile applications must consider them. To compensate for high latencies and unreliable connectivity, clients' cache replicas of important, frequently accessed data, and work offline, if necessary; besides increasing data availability and response time, caching can also reduce client power consumption by eliminating the need to make energy-consuming wireless data transmission for each data access

Data Management Issues

From a data management standpoint, mobile computing may be considered a variation of distributed computing. Mobile databases can be distributed under two possible scenarios:

1. The entire database is distributed mainly among the wired components, possibly with full or partial replication; a base station or fixed host manages its own database with a DBMS-like functionality, with additional functionality for locating mobile units and additional query and transaction management features to meet the requirements of mobile environments.
2. The database is distributed among wired and wireless components; Data management responsibility is shared among base stations or fixed hosts and mobile units.

The distributed data management issues can also be applied to mobile databases with the following additional considerations and variations:

- **Data distribution and replication** – Data is unevenly distributed among the base stations and mobile units.
- **Transactions models** – Issues of fault tolerance and correctness of transactions are aggravated.
- **Query processing** – Awareness of where data is located is important and affects the cost/benefit analysis of query processing; Query optimization is more complicated because of mobility and rapid resource changes of mobile units.
- **Recovery and fault tolerance** – The mobile database environment must deal with site, media, transaction, and communication failure.
- **Mobile database design** – The global name resolution problem for handling queries is compounded because of mobility and frequent shutdown.
- **Location-based service** – As clients move, location-dependent cache information may become stale.
- **Division of labor** – Certain characteristics of the mobile environment force a change in the division of labor in query processing.
- **Security** – Mobile data is less secure than that which is left at the fixed location.