# 11. Advanced Database Concepts

## 11.1 Object-Oriented Model:

A data model is a logic organization of the real world objects (entities), constraints on them, and the relationships among objects. A DB language is a concrete syntax for a data model. A DB system implements a data model.

A core object-oriented data model consists of the following basic object-oriented concepts:

(1) **object and object identifier**: Any real world entity is uniformly modeled as an object (associated with a unique id: used to pinpoint an object to retrieve).

(2) **attributes and methods**: every object has a state (the set of values for the attributes of the object) and a behavior (the set of methods - program code - which operate on the state of the object). The state and behavior encapsulated in an object are accessed or invoked from outside the object only through explicit message passing.

[ An attribute is an instance variable, whose domain may be any class: user-defined or primitive. A class composition hierarchy (aggregation relationship) is orthogonal to the concept of a class hierarchy. The link in a class composition hierarchy may form cycles. ]

(3) **class**: a means of grouping all the objects which share the same set of attributes and methods. An object must belong to only one class as an instance of that class (instance-of relationship). A class is similar to an abstract data type. A class may also be primitive (no attributes), e.g., integer, string, Boolean.

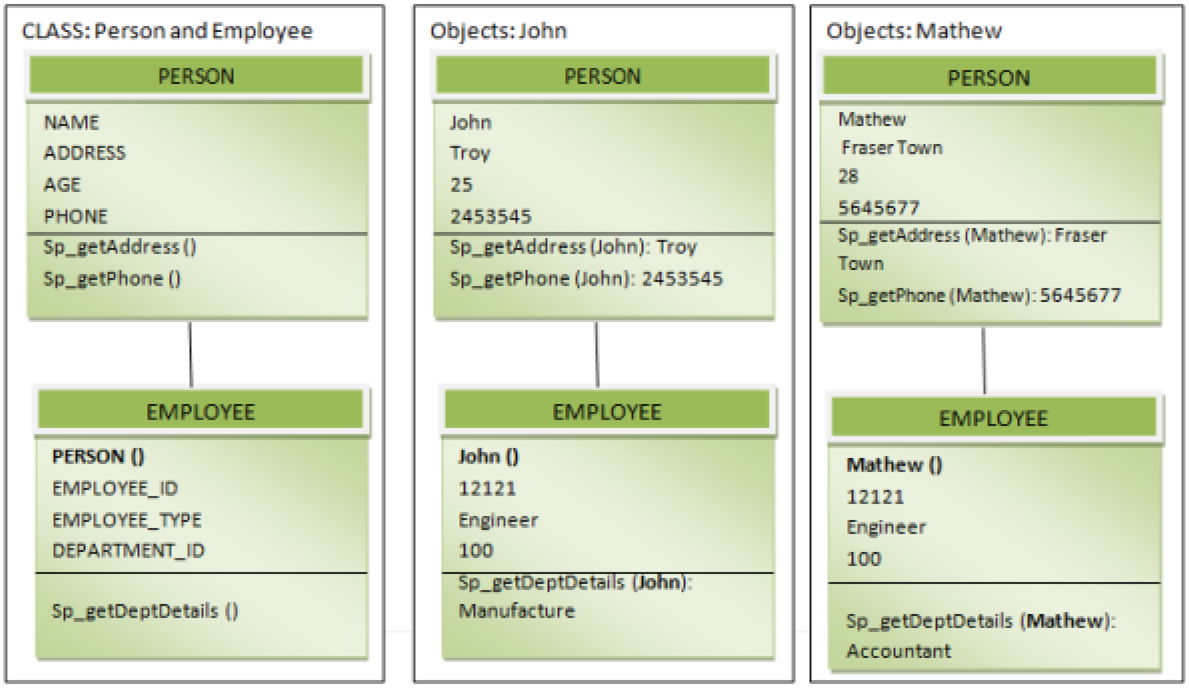
(4) **Class hierarchy and inheritance**: derive a new class (subclass) from an existing class (superclass). The subclass inherits all the attributes and methods of the existing class and may have additional attributes and methods. single inheritance (class hierarchy) vs. multiple inheritance (class lattice).

Let us consider an Employee database to understand this model better. In this database we have different types of employees – Engineer, Accountant, Manager, Clark. But all these employees belong to Person group. Person can have different attributes like name, address, age and phone. What do we do if we want to get a person's address and phone number? We write two separate procedure sp_getAddress and sp_getPhone.

What about all the employees above? They too have all the attributes what a person has. In addition, they have their EMPLOYEE_ID, EMPLOYEE_TYPE and DEPARTMENT_ID attributes to identify them in the organization and their department. We have to retrieve their department details, and hence we sp_getDeptDetails procedure. Currently, say we need to have only these attributes and functionality.

Since all employees inherit the attributes and functionalities of Person, we can re-use those features in Employee. But do we do that? We group the features of person together into class. Hence a class has all the attributes and functionalities. For example, we would create a person class and it will have name, address, age and phone as its attribute, and sp_getAddress and sp_getPhone as procedures in it. The values for these attributes at any instance of time are object. i.e. ; {John, Troy, 25, 2453545 : sp_getAddress (John), sp_getPhone (John)} forms on person object. {Mathew, Fraser Town, 28, 5645677: sp_getAddress (Mathew), sp_getPhone (Mathew} forms another person object.

Now, we will create another class called Employee which will inherit all the functionalities of Person class. In addition it will have attributes EMPLOYEE_ID, EMPLOYEE_TYPE and DEPARTMENT_ID, and sp_getDeptDetails procedure. Different objects of Employee class are Engineer, Accountant, Manager and Clerk.




Here we can observe that the features of Person are available only if other class is inherited from it. It would be a black box to any other classes. This feature of this model is called encapsulation. It binds the features in one class and hides it from other classes. It is only visible to its objects and any inherited classes.

**Advantages:**

- Because of its inheritance property, we can re-use the attributes and functionalities. It reduces the cost of maintaining the same data multiple times. Also, these informations are encapsulated and, there is no fear being misused by other objects. If we need any new feature we can easily add new class inherited from parent class and adds new features. Hence it reduces the overhead and maintenance costs.
- Because of the above feature, it becomes more flexible in the case of any changes.
- Codes are re-used because of inheritance.
- Since each class binds its attributes and its functionality, it is same as representing the real world object. We can see each object as a real entity. Hence it is more understandable.

**Disadvantages**

- It is not widely developed and complete to use it in the database systems. Hence it is not accepted by the users.
- It is an approach for solving the requirement. It is not a technology. Hence it fails to put it in the database management systems.

## 11.2 Object- Relational Model (ORM)

The object-relational model is designed to provide a relational database management that allows developers to integrate databases with their data types and methods. It is essentially a relational model that allows users to integrate object-oriented features into it.
This design is most recently shown in the Nordic Object/Relational Model. The primary function of this new object-relational model is to more power, greater flexibility, better performance, and greater data integrity then those that came before it.
Some of the **benefits** that are offered by the Object-Relational Model include:

- **Extensibility** – Users are able to extend the capability of the database server; this can be done by defining new data types, as well as user-defined patterns. This allows the user to store and manage data.

- **Complex types** – It allows users to define new data types that combine one or more of the currently existing data types. Complex types aid in better flexibility in organizing the data on a structure made up of columns and tables.

- **Inheritance** – Users are able to define objects or types and tables that procure the properties of other objects, as well as add new properties that are specific to the object that has been defined.

- A field may also contain an object with attributes and operations.

- Complex objects can be stored in relational tables.

The object-relational database management systems which are also known as ORDBMS, these systems provide an addition of new and extensive object storage capabilities to the relational models at the center of the more modern information systems of today.

These services assimilate the management of conventional fielded data, more complex objects such as a time-series or more detailed geospatial data and varied dualistic media such as audio, video, images, and applets.

This can be done due to the model working to summarize methods with data structures, the ORDBMS server can implement complex analytical data and data management operations to explore and change multimedia and other more complex objects.

**Disadvantages of ORDBMSs**

- The ORDBMS approach has the obvious disadvantages of complexity and associated increased costs. Further, there are the proponents of the relational approach that believe the· essential simplicity' and purity of the .relational model are lost with these types of extension.

## 11.3 Distributed Databases:

A distributed database system consists of a collection of sites, each of which maintains a local database system. Each site is able to process local transactions: those transactions that access data in only that single site. In addition, a site may participate in the execution of global transactions; those transactions that access data in several sites. The execution of global transactions requires communication among the sites. The general structure of a distributed system appears in Figure below:

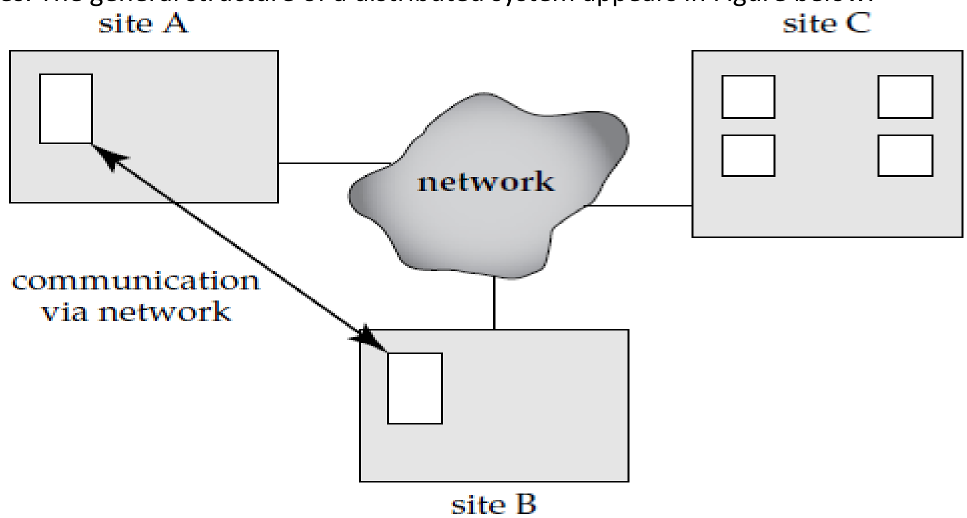



**Figure 18.9** A distributed system.

There are several reasons for building distributed database systems, including sharing of data, autonomy, and availability.

• **Sharing data.** The major advantage in building a distributed database system is the provision of an environment where users at one site may be able to access the data residing at other sites. For instance, in a distributed banking system, where each branch stores data related to that branch, it is possible for a user in one branch to access data in another branch. Without this capability, a user wishing to transfer funds from one branch to another would have to resort to some external mechanism that would couple existing systems.

• **Autonomy.** The primary advantage of sharing data by means of data distribution is that each site is able to retain a degree of control over data that are stored locally. In a centralized system, the database administrator of the central site controls the database. In a distributed system, there is a global database administrator responsible for the entire system. A part of these responsibilities is delegated to the local database administrator for each site. Depending on the design of the distributed database system, each administrator may have a different degree of **local autonomy**. The possibility of local autonomy is often a major advantage of distributed databases.

• **Availability.** If one site fails in a distributed system, the remaining sites may be able to continue operating. In particular, if data items are **replicated** in several sites, a transaction needing a particular data item may find that item in any of several sites. Thus, the failure of a site does not necessarily imply the shutdown of the system.

## Homogeneous and Heterogeneous Databases:

In a **homogeneous distributed database**, all sites have identical database management system software, are aware of one another, and agree to cooperate in processing users' requests. In such a system, local sites surrender a portion of their autonomy in terms of their right to change schemas or database management system software. That software must also cooperate with other sites in exchanging information about transactions, to make transaction processing possible across multiple sites.

In contrast, in a **heterogeneous distributed database**, different sites may use different schemas, and different database management system software. The sites may not be aware of one another, and they may provide only limited facilities for cooperation in transaction processing. The differences in schemas are often a major problem for query processing, while the divergence in software becomes a hindrance for processing transactions that access multiple sites.

## Disadvantages of Distributed Database:

- Complexity — extra work must be done by the DBAs to ensure that the distributed nature of the system is transparent. Extra work must also be done to maintain multiple disparate systems, instead of one big one. Extra database design work must also be done to account for the disconnected nature of the database — for example, joins become prohibitively expensive when performed across multiple systems.
- Economics — increased complexity and a more extensive infrastructure means extra labour costs.
- Security — remote database fragments must be secured, and they are not centralized so the remote sites must be secured as well. The infrastructure must also be secured (e.g., by encrypting the network links between remote sites).
- Difficult to maintain integrity — but in a distributed database, enforcing integrity over a network may require too much of the network's resources to be feasible.
- Additional software is required.
- Concurrency control: it is a major issue. It can be solved by locking and timestamping.

# 11.4 Concepts of Data Warehouses:

A **data warehouse** is a repository (or archive) of information gathered from multiple sources, stored under a unified schema, at a single site. Once gathered, the data are stored for a long time, permitting access to historical data. Thus, data warehouses provide the user a single consolidated interface to data, making decision-support queries easier to write. Moreover, by accessing information for decision support from a data warehouse, the decision maker ensures that online transaction-Processing systems are not affected by the decision-support workload.

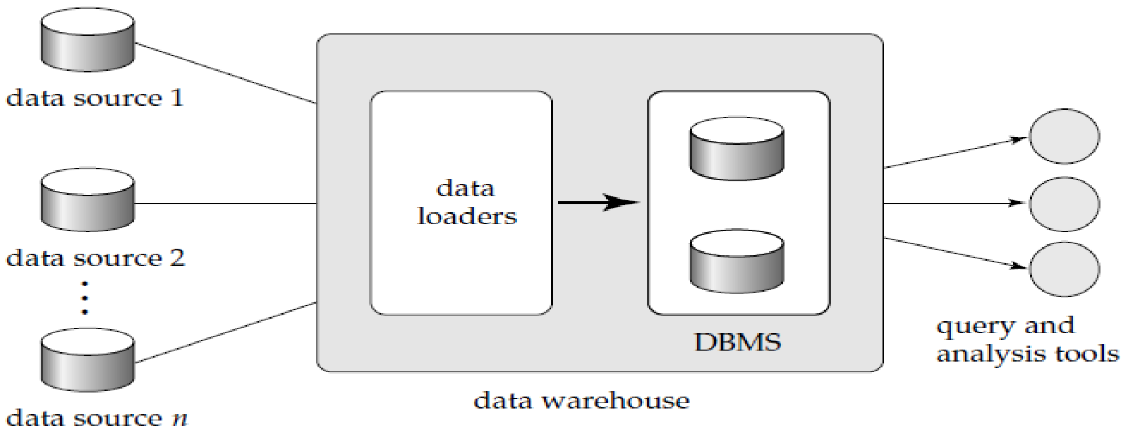



**Figure 22.8** Data-warehouse architecture.

**Components of a Data Warehouse:**

Figure 22.8 shows the architecture of a typical data warehouse, and illustrates the gathering of data, the storage of data, and the querying and data-analysis support. Among the issues to be addressed in building a warehouse are the following:

• **When and how to gather data.** In a **source-driven architecture** for gathering data, the data sources transmit new information, either continually (as transaction processing takes place), or periodically (nightly, for example). In a **destination-driven architecture**, the data warehouse periodically sends requests for new data to the sources.

• **What schema to use.** Data sources that have been constructed independently are likely to have different schemas. In fact, they may even use different data models. Part of the task of a warehouse is to perform schema integration, and to convert data to the integrated schema before they are stored. As a result, the data stored in the warehouse are not just a copy of the data at the sources. Instead, they can be thought of as a materialized view of the data at the sources.

• **Data cleansing.** The task of correcting and preprocessing data is called **data cleansing**. Data sources often deliver data with numerous minor inconsistencies, that can be corrected. For example, names are often misspelled, and addresses may have street/area/city names misspelled, or zip codes entered incorrectly. These can be corrected to a reasonable extent by consulting a database of street names and zip codes in each city. Address lists collected from multiple sources may have duplicates that need to be eliminated in a **merge– purge operation**. Records for multiple individuals in a house may be grouped together so only one mailing is sent to each house; this operation is called **householding**.

• **How to propagate updates**. Updates on relations at the data sources must be propagated to the data warehouse. If the relations at the data warehouse are exactly the same as those at the data source, the propagation is straightforward. If they are not, the problem of propagating updates is basically the *view-maintenance* problem.

• **What data to summarize.** The raw data generated by a transaction-processing system may be too large to store online. However, we can answer many queries by maintaining just summary data obtained by aggregation on a relation, rather than maintaining the entire relation. For example, instead of storing data about every sale of clothing, we can store total sales of clothing by item name and category.