# Unit-1 Databases-Basic Concepts

Structure

## 1. Introduction

All of us use databases implicitly or explicitly, knowingly or unknowingly, directly or indirectly in our daily lives. And most of us are ignorant about this fact. Let us take some examples to understand this: when we go to a bank counter, the bank clerk uses database on our behalf, on the other hand when we go to an ATM machine we use the database directly. Again when we book air or rail ticket- we use the database either directly or indirectly. When we go to a shopping Mall- we use the Inventory (Although indirectly) and the POS (Point of sale) machine that accesses the database.

Anyway, this is how we are living with databases all around us. Now the question arises what exactly a database is. A database, simply put is a collection of related data. It can also be defined as an organized collection of related data stored for some specific purpose. A DBMS is a software package that is used to create, manipulate and manage a database. By this we mean that DBMS provides us tools to create a database, insert data into it, delete, modify or update data in the database. Examples of DBMS include licensed packages like ORACLE, SQL Server and Open source packages like MySQL, PostgreSQL etc.

This unit discusses the basic concepts in Databases. Lesson 1 talks about the limitations of the file based systems and the characteristics, advantages and disadvantages of the database approach. Lesson 2 discusses the logical DBMS architecture and the concept of data independence. Lesson 3 deals with the physical DBMS architecture and the related concepts. Lesson 4 deals with database languages and the different interfaces provided by DBMS. Lesson 5 introduces the concept of data modeling using ER approach.

## 2. Objectives

After going through this unit you should be able to:

- Describe the File Based system and its limitations;
- Describe the structure and characteristics of DBMS;
- Define the functions of DBA;
- Explain the three-tier/three schema architecture of DBMS;
- Explain ER modeling and
- Explain database languages and interface.

# Lesson-1

## Why use DBMS

Why do we need the database management system? To explain this, let us first discuss the alternative to DBMS that is the file-based system.

## Limitations of file based system

File based systems are an early attempt to computerize the manual filing system. At homes files relating to bank statements, receipts, tax payments, etc can be maintained. What do we do to find information from these files? For retrieval of information, the entries could be searched sequentially. Alternatively, an indexing system could be used to locate the information.

The manual filing system works well when the number of items to be stored is small. It even works quite well when the number of items stored is quite large and they are only needed to be stored and retrieved. However, a manual file system crashes when cross-referencing and processing of information in the files is carried out. For example, in a college a number of students are enrolled who have the options of doing various courses. The college may have separate files for the personal details of students, fees paid by them, the number and details of the courses taught, the number and details of each faculty member in various departments. Consider the effort to answer the following queries.

- Annual fees paid by the B.Com students.
- Number of students requiring transport facility from a particular area.
- This year’s turnover of students as compared to last year.
- Number of students opting for different courses from different departments.
- Courses taught by various teachers

The answers to all these questions would be cumbersome and time consuming in the file based system. The file-based systems have certain limitations, listed below:

- **Separation and isolation of data:** When the data is stored in separate files it becomes difficult to access. It becomes extremely complex when the data has to be retrieved from more than two files as a large amount of data has to be searched.
- **Duplication (**Rather Multiplication**) of data:** Due to the decentralized approach, the file system leads to uncontrolled duplication of data. This is undesirable as the duplication leads to wastage of a lot of storage space. It also costs time and money to enter the data more than once. For example, the address information of student may have to be duplicated in bus list file data.
- **Inconsistent Data:** The data in a file system can become inconsistent if more than one person modifies the data concurrently, for example, if any student changes the residence and the change is notified to only his/her file and not to bus list. Entering wrong data is also another reason for inconsistencies.
- **Data dependence:** The physical structure and storage of data files and records are defined in the application code. This means that it is extremely difficult to make changes to the existing structure. The programmer would have to identify all the affected programs, modify them and retest them. This characteristic of the File Based system is called program data dependence.
- **Incompatible File Formats:** Since the structure of the files is embedded in application programs, the structure is dependent on application programming languages. Hence the structure of a file generated by COBOL programming language may be quite different from a file generated by 'C' programming language. This incompatibility makes them difficult to process jointly. The application developer may have to develop software to convert the files to some common format for processing. However, this may be time consuming and expensive.
- **Fixed Queries:** File based systems are very much dependent on application programs. Any query or report needed by the organization has to be developed by the application programmer. With time, the type and number of queries or reports

increases. Producing different types of queries or reports is not possible in File Based Systems. As a result, in some organizations the type of queries or reports to be produced is fixed. No new query or report of the data could be generated.

Besides the above, the maintenance of the File Based System is difficult and there is no provision for security. Recovery is inadequate or non-existent. Because of all these limitations, file based systems are seldom used now.

## The Database Approach

In order to overcome the limitations of a file system, a new approach called database approach emerged. A database is a persistent collection of logically related data. The initial attempts were to provide a centralized collection of data. A database has a self-describing nature. It contains not only the data but also the complete definition of the database structure and constraints, which are stored in a system catalog. A DBMS manages this data. It allows data sharing and integration of data of an organization in a single database. DBMS controls access to this data and thus needs to provide features for database creation, data manipulation such as data value modification, data retrieval, data integrity and security etc. Some of the advantages of database approach are discussed here.

- **Reduced Redundancy**

  In a file processing system, each user group maintains its own files resulting in a considerable amount of redundancy of the stored data. This results in wastage of storage space but more importantly may result in data inconsistencies. Also, the same data has to be updated more than once resulting in duplication of effort. The files that represent the same data may become inconsistent as some may be updated whereas others may not be. In database approach data can be stored at a single place or with controlled redundancy under DBMS, which saves space and does not permit inconsistency.

- **Shared Data**

  A DBMS allows the sharing of database under its control by any number of application programs or users. A database belongs to the entire organization and is shared by all authorized users.

- **Data Independence**

  In the file-based system, the descriptions of data and logic for accessing the data are built into each application program making the program more dependent on data. A change in the structure of data may require alterations to programs. Database Management systems separates data descriptions from data. Hence it is not affected by changes. This is called Data Independence, where details of data are not exposed. DBMS provides an abstract view and hides details. For example, logically we can say that the interface or window to data provided by DBMS to a user may still be the same although the internal structure of the data may be changed (physical Data Independence).

- **Improved Integrity**

  Data Integrity refers to validity and consistency of data. Data Integrity means that the data should be accurate and consistent. This is done by providing some checks or constraints. These are consistency rules that the database is not permitted to violate. Constraints may apply to data items within a record or relationships between records. For example, the age of an employee can be between 18 and 70 years only. While entering the data for the age of an employee, the database should check this.

- **Efficient Data Access**

DBMS utilizes techniques to store and retrieve the data efficiently. A complex DBMS provides services to end users, where they efficiently retrieve the data almost immediately.

- **Multiple User Interfaces**

Since many users with varying levels of technical knowledge use a database, DBMS's provide a variety of interfaces. These include-

a) A query language for casual users,
b) A programming language interface for application programmers,
c) Forms and codes for parametric users,
d) Menu driven interfaces, and
e) Natural language interfaces for standalone users, these interfaces are still not available in standard form with commercial database.

- **Representing complex relationship among data**

A database may include varieties of data, interrelated to each other in many ways. Modern and advanced DBMS's have the capability to represent a variety of relationships among the data as well as to retrieve and update related data easily and efficiently.

- **Improved Security**

Data is vital to the existence of any organization. In a shared system where multiple users share the data, all information should not be shared by all users. For example, the salary of the employees should not be visible to anyone other than the department dealing in this. Hence, database should be protected from unauthorized users. This is done by Database Administrator (DBA) by providing the usernames and passwords only to authorized users as well as granting privileges or the type of operation allowed. This is done by using security and authorization subsystem. Only authorized users may use the database and their access types can be restricted to only retrieval, insert, update or delete or any of these. For example, the Branch Manager of any company may have access to all data whereas the Sales Assistant may not have access to salary details.

- **Improved Backup and Recovery**

  A file-based system may fail to provide measures to protect data from system failures. This lies solely on the user by taking backups periodically. DBMS provides facilities for recovering the hardware and software failures. A backup and recovery subsystem is responsible for this. In case a program fails, it restores the database to a state (consistent and safe) in which it was before the execution of the program.

- **Support for concurrent transactions**

  A transaction is defined as a unit of work. For example, a bank may be involved in a transaction where an amount of Rs.5000/- is transferred from account X to account Y. A DBMS also allows multiple transactions to occur simultaneously.

# Lesson-2

## THE LOGICAL DBMS ARCHITECTURE

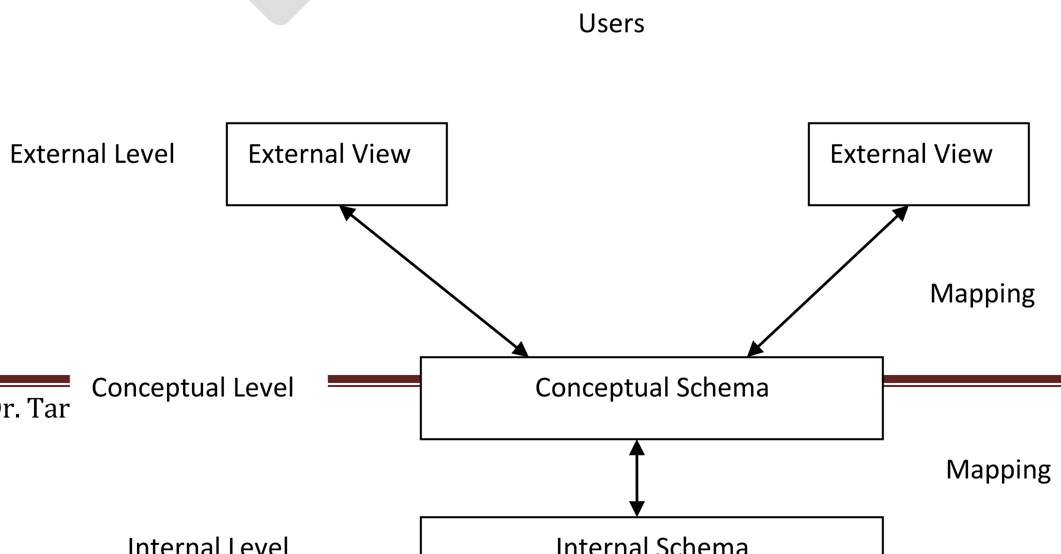
Database Management Systems are very complex, sophisticated software applications that provide reliable management of large amounts of data. To describe general database concepts and the structure and capabilities of a DBMS better, we should study the architecture of a typical database management system.

There are two different ways to look at the architecture of a DBMS: the logical DBMS architecture and the physical DBMS architecture. The logical architecture deals with the way data is stored and presented to users, while the physical architecture is concerned with the software and hardware components that make up a DBMS.

## LOGICAL DBMS ARCHITECTURE/THREE SCHEMA ARCHITECTURE

The logical DBMS architecture describes how data in the database is perceived by users. It is not concerned with how the data is handled and processed by the DBMS, but only with how it looks. The method of data storage on the underlying file system is not revealed, and the users can manipulate the data without worrying about where it is located or how it is actually stored. This results in the database having different levels of abstraction.

The majority of commercial Database Management Systems available today are based on the ANSI/SPARC generalized DBMS architecture, as proposed by the ANSI/SPARC Study Group on Data Base Management Systems. Hence this is also called as the ANSI/SPARC model. It divides the system into three levels of abstraction: the internal or physical level, the conceptual or logical level, and the external or view level. The diagram below shows the logical architecture for a typical DBMS.



The external or view level: It is the highest level of abstraction of database. It provides a window on the conceptual view, which allows the user to see only the data of interest to them. The user can be either an application program or an end user. There can be many external views as any number of external schemas can be defined and they can overlap each other. It consists of the definition of logical records and relationships in the external view. It also contains the methods for deriving the objects such as entities, attributes and relationships in the external view from the Conceptual view.

The Conceptual Level or logical level: The conceptual level presents a logical view of the entire database as a unified whole. It allows the user to bring all the data in the database together and see it in a consistent manner. Hence, there is only one conceptual schema per database. The first stage in the design of a database is to define the conceptual view, and a DBMS provides a data definition language (DDL) for this purpose. It describes all the records and relationships included in the database. The data definition language used to create the conceptual level must not specify any physical storage considerations that should be handled by the physical level. It does not provide any storage or access details, but defines the information content only.

The Internal or Physical Level: The collection of files permanently stored on secondary storage devices is known as the physical database. The physical or internal level is the one closest to physical storage devices, and it provides a low-level description of the physical database, and an interface between the operating systems file system and the record structures used in higher levels of abstraction. It is at this level that record types and methods of storage are defined, as well as how stored fields are represented, what physical sequence the stored records are in, and what other physical structures exist.

**Mappings between Levels and Data Independence**

The three levels of abstraction do not exist independent of each other. There must be some interface/correspondence or mapping between the three levels. There are two types of mappings: the conceptual/internal mapping and the external/conceptual mapping (Refer Figure 1).

The conceptual/internal mapping lies between the conceptual and internal levels, and defines the correspondence between the records and the fields of the conceptual view and the files and data structures of the internal view. If the structure of the stored database is changed, then the conceptual/ internal mapping must also be changed accordingly so that the view from the conceptual level remains constant. It is this mapping that provides physical data independence for the database. For example, we may change the internal view of student relation by breaking the student file into two files, one containing regno, name and address and other containing registration programme. However, the mapping will make sure that the conceptual view is restored as original and does not change.

The external/conceptual view lies between the external and conceptual levels, and defines the correspondence between a particular external view and the conceptual view. Although these two levels are similar, some elements found in a particular external view may be different from the conceptual view. For example, several fields can be combined into a single (virtual) field, which can also have different names from the original fields. If the structure of the database at the conceptual level is changed, then the external/conceptual mapping must change accordingly so that the view from the external level remains constant. It is this mapping that provides logical data independence for the database. For example, we may change the student relation to have more fields at conceptual level, yet this will not change the two user views at all.

It is also possible to have another mapping, where one external view is expressed in terms of other external views (this could be called an external/external mapping). This is useful if several external views are closely related to one another, as it allows you to avoid mapping each of the similar external views directly to the conceptual level.

**The need for Three level architecture**

The objective of three level architecture is to separate each user's view of the database from the way the database is physically represented. This is achieved in the following way:

- Support of multiple user views: Each user is able to access the same data, but have a different customized view of the data. Each user is able to change the way he or she views the data and this change does not affect other users.
- Insulation between user programs and data: Users do not directly deal with physical storage details, such as indexing or hashing. The user's interaction with the database is independent of storage considerations. Insulation between conceptual and physical structures can be defined as:
    1. The Database Administrator should be able to change the storage structures without affecting users' views.
    2. The internal structure of the database should be unaffected by the changes to the physical aspects of the storage, such as changing to a new storage device.
    3. The DBA should be able to change the conceptual structure of the database without affecting all users.

# Lesson-3

## PHYSICAL DBMS ARCHITECTURE

The physical architecture describes the software components (including the underlying hardware) used to enter and process data, and how these software components are related and interconnected. Although it is not possible to generalize the component structure of a DBMS, it is possible to identify a number of key functions which are common to most database management systems. The components that normally implement these functions are shown in Figure below, which depicts the physical architecture of a typical DBMS.

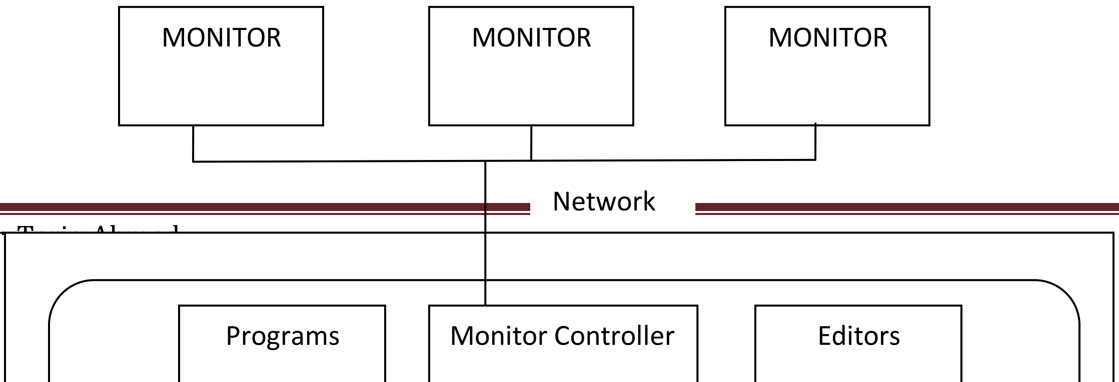


Figure 2. Physical DBMS architecture

Based on various functions, the database system may be partitioned into the following modules. Some functions (for example, file systems) may be provided by the operating system and network protocols etc.

**DML Precompiler**

All the Database Management systems have two basic sets of Languages ─ Data Definition Language (DDL) that contains the set of commands required to define the format of the data that is being stored and Data Manipulation Language (DML) which defines the set of commands that modify, process data to create user definable output. The DML statements can also be written in an application program. The DML Precompiler converts DML statements (such as SELECT …. FROM in Structured Query Language (SQL) covered in Unit-3) embedded in an application program to normal procedural calls in the host language. The Precompiler interacts with the query processor in order to generate the appropriate code.

**DDL Compiler**

The DDL compiler converts the data definition statements (such as CREATE TABLE ... in SQL) into a set of tables containing metadata tables. These tables contain information concerning the database and are in a form that can be used by other components of the DBMS. These tables are then stored in a system catalog or data dictionary.

**File Manager**

File manager manages the allocation of space on disk storage. It establishes and maintains the list of structures and indices defined in the internal schema that is used to represent information stored on disk. However, the file manager does not directly manage the physical input and output of data. It passes the requests on to the appropriate access methods, which either read data from or write data into the system buffer or cache. The file manager can be implemented using an interface to the existing file subsystem provided by the operating system of the host computer or it can include a file subsystem written especially for the DBMS.

**Database Manager**

It is the interface between low-level data, application programs and queries. Databases typically require a large amount of storage space. It is stored on disks, as the main memory of computers cannot store this information. Data is moved between disk storage and main memory (into SGA-system global area) as needed. Since the movement of data to and from disk is slow relative to the speed of the central processing unit of computers, it is imperative that database system structure data in such a way so as to minimize the need to move data between disk and main memory.

A database manager is a program module responsible for interfacing the database file system to the user queries. In addition, the tasks of enforcing constraints to maintain the consistency and integrity of the data as well as its security are also performed by database manager. Synchronizing the simultaneous operations performed by concurrent users is under the control of the data manager. It also performs backup and recovery operations. Now let us summarize the important responsibilities of Database manager.

- Interaction with file manager: The raw data is stored on the disk using the file system which is usually provided by a conventional operating system. The database manager translates the various DML statements into low-level file system commands. Thus, the database manager is responsible for the actual storing, retrieving and updating of data in the database.
- Integrity enforcement: The data values stored in the database must satisfy certain types of consistency constraints. For example, the balance of a bank account may never fall below a prescribed amount (for example, Rs. 500/-). Similarly the number of holidays per year an employee may be having should not exceed 8 days. These constraints must be specified explicitly by the DBA. If such constraints are specified, then the database manager can check whether updates to the database result in the violation of any of these constraints and if so appropriate action may be imposed.
- Security enforcement: As discussed above, not every user of the database needs to have access to the entire content of the database. It is the job of the database manager to enforce these security requirements.
- Backup and recovery: A computer system is subject to failures. There are a variety of causes of such failure, including disk crash, power failure and software errors. In each of these cases, information concerning the database is lost. It is the responsibility of database manager to detect such failures and restore the database to a state (safe and consistent) that existed prior to the occurrence of the failure. This is usually accomplished through the backup and recovery procedures.
- Concurrency control: When several users update the database concurrently, the consistency of data may no longer be preserved. It is necessary for the system to

control the interaction among the concurrent users, and achieving such a control is one of the responsibilities of database manager (using various techniques).

The above functions are achieved through the database manager. The major components of a database manager are:

- **Authorization control:** This module checks that the user has necessary authorization to carry out the required function.
- **Command Processor:** Once the system has checked that the user has authority to carry out the operation, control is passed to the command processor, which converts commands to a logical sequence of steps.
- **Integrity checker:** For an operation that changes the database, the integrity checker checks that the requested operation satisfies all necessary integrity constraints such as key constraints.
- **Query Optimizer:** This module determines an optimal strategy for the query execution.
- **Transaction Manager:** This module performs the required processing of operations of various transactions. The transaction manager maintains tables of authorization. The DBMS may use authorization tables to allow the transaction manager to ensure that the user has permission to execute the desired operation on the database. The authorization tables can only be modified by properly authorized user commands, which are also checked against the authorization tables.
- **Scheduler:** This module is responsible for ensuring that concurrent operations or transactions on the database proceed without conflicting with one another. It controls the relative order in which transaction operations are executed. A database may also support concurrency control tables to prevent conflicts when simultaneous, conflicting commands are executed. The DBMS checks the concurrency control tables before executing an operation to ensure that the data used by it is not locked by another statement.

- **Recovery Manager:** This module ensures that the database remains in a consistent state in the presence of failures. It is responsible for transaction commit and abort that is success or failure of transaction.
- **Buffer Manager:** This module is responsible for the transfer of data between main memory and secondary storage, such as disk and tape. The recovery manager and the buffer manager are sometimes collectively referred to as data manager. The buffer manager is sometimes known as cache manager.

## Query Processor

The query language processor is responsible for receiving query language statements and changing them from the English-like syntax of the query language to a form the DBMS can understand. The query language processor usually consists of two separate parts: the parser and the query optimizer. The parser receives query language statements from application programs or command-line utilities and examines the syntax of the statements to ensure they are correct. To do this, the parser breaks a statement down into basic units of syntax called tokens and examines them to make sure each statement consists of the proper component parts. If the statements follow the syntax rules, the tokens are passed to the query optimizer.

The query optimizer examines the query language statement, and tries to choose the best and most efficient way of executing the query. To do this, the query optimizer will generate several query plans in which operations are performed in different orders, and then try to estimate which plan will execute most efficiently. When making this estimate, the query optimizer may examine factors such as: CPU time, disk time, network time, sorting methods, and scanning methods etc.

## Data Dictionary

A Data Dictionary or catalogue stores information about the structure of the database. It is used heavily. Hence a good data dictionary should have a good design and efficient implementation. It is seen that when a program becomes somewhat large in size, keeping track of all the available names that are used and the purpose for which they were used becomes more and more difficult. After a significant time if the same or another programmer has to modify the program, it becomes extremely difficult.

The problem becomes even more difficult when the number of data types that an organization has in its database increases. The data of an organization is a valuable corporate resource and therefore some kind of inventory and catalog of it must be maintained so as to assist in both the utilization and management of the resource. It is for this purpose that a data dictionary or dictionary directory is emerging as a major tool. A dictionary provides definitions of things. A directory tells you where to find them. A data dictionary/directory contains information (or data) about the data or what we call metadata.

A comprehensive data dictionary would provide the definition of data items, how they fit into the data structure and how they relate to other entities in the database. In DBMS, the data dictionary stores the information concerning the external, conceptual and internal levels of the databases. It would combine the source of each data field value, that is from where the authenticate value is obtained. The frequency of its use and audit trail regarding the updates including user identification with the time of each update is also recorded in Data dictionary.

The Database administrator uses the data dictionary in every phase of a database life cycle, starting from the data gathering phase to the design, implementation and maintenance phases. Documentation provided by a data dictionary is as valuable to end users and managers, as it is essential to the programmers. Users can plan their applications with the database only if they know exactly what is stored in it. For example, the description of a data item in a data dictionary may include its origin and other text description in plain English, in addition to its data format. Thus, users and managers will be able to see exactly what is available in the database. A data dictionary is a road map which guides users to access information within a large database. An ideal data dictionary should include everything a DBA wants to know about the database including the following:

1. External, conceptual and internal database descriptions.
2. Descriptions of entities (record types), attributes (fields), as well as cross-references, origin and meaning of data elements.
3. Synonyms, authorization and security codes.
4. Which external schemas are used by which programs, who the users are, and what their authorizations are.
5. Statistics about database and its usage including number of records, etc.

A data dictionary is implemented as a database (tables in RDBMS) so that users can query its contents. The cost effectiveness of a data dictionary increases as the complexity of an information system increases. A data dictionary can be a great asset not only to the DBA for database design, implementation and maintenance, but also to managers or end users in their project planning.

# Lesson-4

## Database Languages and Interface

At its most basic level the DBMS architecture can be broken down into two parts: the back end and the front end. The back end (Usually a database) is responsible for managing the physical database and providing the necessary support and mappings for the internal, conceptual, and external levels. Other benefits of a DBMS, such as security, integrity, and access control, are also the responsibility of the back end.

The front end is really just any application that runs on top of the DBMS (like VB, oracle Developer etc). These may be applications provided by the DBMS vendor, the user, or a third party. The user interacts with the front end, and may not even be aware that the back end exists. This interaction is done through Applications and Utilities which are the main interface to the DBMS for most users.

There are three main sources of applications and utilities for a DBMS:

a) Vendor applications and utilities are provided for working with or maintaining the database, and usually allow users to create and manipulate a database without the need to write custom applications.
b) User applications are generally custom-made application programs written for a specific purpose using a conventional programming language. This programming language is coupled to the DBMS query language through the application program interface (API). This allows the user to utilize the power of the DBMS query language with the flexibility of a custom application.
c) Third party applications may be similar to those provided by the vendor, but with enhancements, or they may fill a perceived need that the vendor hasn't created an application for. They can also be similar to user applications, being written for a specific purpose they think a large majority of users will need.

The most common applications and utilities used with a database can be divided into several well-defined categories. These are:

- **Command Line Interfaces:** These are character-based, interactive interfaces that let us use the full power and functionality of the DBMS query language directly. They allow us to manipulate the database and perform ad-hoc queries and see the results immediately. They are often the only method of exploiting the full power of the database without creating programs using a conventional programming language.
- **Graphical User Interface (GUI) tools:** These are graphical, interactive interfaces that hide the complexity of the DBMS and query language behind an intuitive, easy to understand, and convenient interface. This allows casual users the ability to access the database without having to learn the query language, and it allows advanced users to quickly manage and manipulate the database without the trouble of entering formal commands using the query language. This may include Menu and Form based interfaces. However, graphical interfaces usually do not provide

the same level of functionality as a command line interface because it is not always possible to implement all commands or options using a graphical interface.

- **Backup/Restore Utilities:** These are designed to minimize the effects of a database failure and ensure a database is restored to a consistent state if a failure occurs. Manual backup/restore utilities require the user to initiate the backup, while automatic utilities will back up the database at regular intervals without any intervention from the user. Proper use of a backup/restore utility allows a DBMS to recover from a system failure correctly and reliably.
- **Load/Unload Utilities:** These allow the user to unload a database or parts of a database and reload the data on the same machine, or on another machine in a different location. This can be useful in several situations, such as for creating backup copies of a database at a specific point in time, or for loading data into a new version of the database or into a completely different database. These load/unload utilities may also be used for rearranging the data in the database to improve performance, such as clustering data together in a particular way or reclaiming space occupied by data that has become obsolete.
- **Reporting/Analysis Utilities:** These are used to analyze and report on the data contained in the database. This may include analyzing trends in data, computing values from data, or displaying data that meets some specified criteria, and then displaying or printing a report containing this information.

## Database Administrator (DBA)

DBMS provides control of both data and programs accessing that data. The person having such control over the system is called the database administrator (DBA). The DBA administers the three levels of the database and defines the global view or conceptual level of the database. The DBA also specifies the external view of the various users and applications and is responsible for the definition and implementation of the internal level, including the storage structure and access methods to be used for the optimum performance of the DBMS. Changes to any of the three levels due to changes in the organization and/or emerging technology are under the control of the DBA.

Mappings between the internal and the conceptual levels, as well as between the conceptual and external levels, are also defined by the DBA. The DBA is responsible for granting permission to the users of the database and stores the profile of each user in the database. This profile describes the permissible activities of a user on that portion of the database accessible to the user via one or more user views. The user profile can be used by the database system to verify that a particular user can perform a given operation on the database.

The DBA is also responsible for defining procedures to recover the database from failures due to human, natural, or hardware causes with minimal loss of data. This recovery procedure should enable the organization to continue to function and the intact portion of the database should continue to be available. the functions of DBA can be summarized as:

- **Schema definition:** Creation of the original database schema is accomplished by writing a set of definitions which are translated by the DDL compiler to a set of tables that are permanently stored in the data dictionary.
- **Storage Structure and access method definition:** The creation of appropriate storage structures and access methods is accomplished by writing a set of definitions which are translated by the data storage and definition language compiler.
- **Schema and Physical organization modification** (Schema Evolution)**:** DBA performs either modification of the database schema or the description of the physical storage organization. These changes, although relatively rare, are accomplished by writing a set of definitions which are used by either the DDL compiler or the data storage and definition language compiler to generate

modification to the appropriate internal system tables (for example the data dictionary).

- **Granting of authorization for data access:** DBA grants and allows the granting (grant with grant option) of different types of authorization for data access to the various users of the database.

**Integrity constraint specification:** The DBA specifies the constraints. These constraints are kept in a special system structure, the data dictionary that is consulted by the database manager prior to any data manipulation. Data Dictionary is one of the valuable tools that the DBA uses to carry out data administration.

# Lesson-5

## DATABASE MODELS

After going through the database architecture, let us now dwell on an important question: how is the data organized in a database? There are many basic structures that exist in a database system. They are called the database models. A database model defines

• The logical data structure

• Data relationships

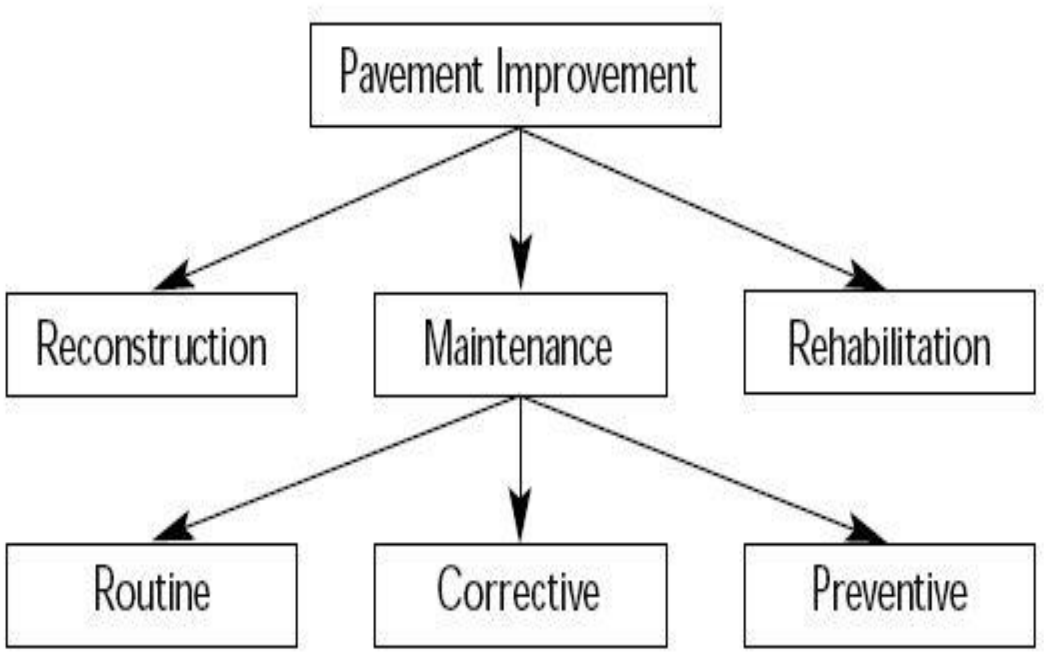
• Data consistency constraints.

Following are the different types of database models used in database systems:

## Hierarchical Model

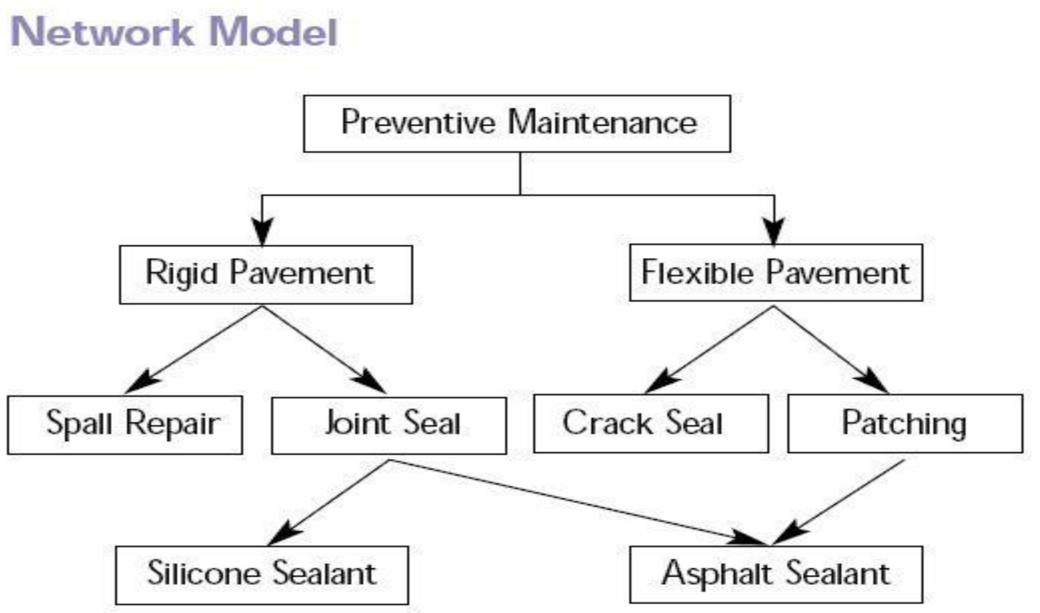
The hierarchical data model organizes data in a tree structure. There is a hierarchy of parent and child data segments. This structure implies that a record can have repeating information, generally in the child data segments. Data is a series of records, which have a set of field values attached to it. It collects all the instances of a specific record together as a record type. These record types are the equivalent of tables in the relational model, and with the individual records being the equivalent of rows. To create links between these record types, the hierarchical model uses Parent Child Relationships. These are a 1:N mapping between record types. This is done by using trees, like set theory used in the relational model, "borrowed" from mathematics. For example, an organization might store information about an employee, such as name, employee number, department, and salary. The organization might also store information about an employee's children, such as name and date of birth. The employee and children data forms a hierarchy, where the employee data represents the parent segment and the children data represents the child segment. If an employee has three children, then there would be three child segments associated with one employee segment. In a hierarchical database the parent-child relationship is one to many. This restricts a child segment to having only one parent segment. Hierarchical DBMSs were popular from the late 1960s, with the introduction of IBM's Information Management System (IMS) DBMS, through the 1970s. Following is a diagrammatic representation of hierarchical model showing an example also.

Hierarchical Model



## Network Model

The popularity of the network data model coincided with the popularity of the hierarchical data model. Some data were more naturally modeled with more than one parent per child. So, the network model permitted the modeling of many-to-many relationships in data. In 1971, the Conference on Data Systems Languages (CODASYL) formally defined the network model. The basic data modeling construct in the network model is the set construct. A set consists of an owner record type, a set name, and a member record type. A member record type can have that role in more than one set; hence the multi-parent concept is supported. An owner record type can also be a member or owner in another set. The data model is a simple network, and link and intersection record types may exist, as well as sets between them. Thus, the complete network of relationships is represented by several pair wise sets; in each set some (one) record type is owner (at the tail of the network arrow) and one or more record types are members (at the head of the relationship arrow). Usually, a set defines a 1:M relationship, although 1:1 is permitted. The CODASYL network model is based on mathematical set theory. The figure below shows the diagrammatic representation of the model.



## Relational Model

RDBMS - relational database management system- A database based on the relational model developed by E.F. Codd. A relational database allows the definition of data structures, storage and retrieval operations and integrity constraints. In such a database the data and relations between them are organized in tables. A table is a collection of records and each record in a table contains the same fields.

Properties of Relational Tables:

- Values Are Atomic
- Each Row is Unique
- Column Values Are of the Same Kind
- The Sequence of Columns is Insignificant
- The Sequence of Rows is Insignificant

- Each Column has a unique name

  Certain fields may be designated as keys, which means that searches for specific values of that field will use indexing to speed them up. Where fields in two different tables take values from the same set, a join operation can be performed to select related records in the two tables by matching values in those fields. Often, but not always, the fields will have the same name in both tables. For example, an "orders" table might contain (customer-ID, product-code) pairs and a "products" table might contain (product-code, price) pairs so to calculate a given customer's bill you would sum the prices of all products ordered by that customer by joining on the product-code fields of the two tables. This can be extended to joining multiple tables on multiple fields. Because these relationships are only specified at retrieval time, relational databases are classed as dynamic database management system. The Relational database model is based on the Relational Algebra. This model has been dealt with great detail in the next unit.

## Object/Relational Model

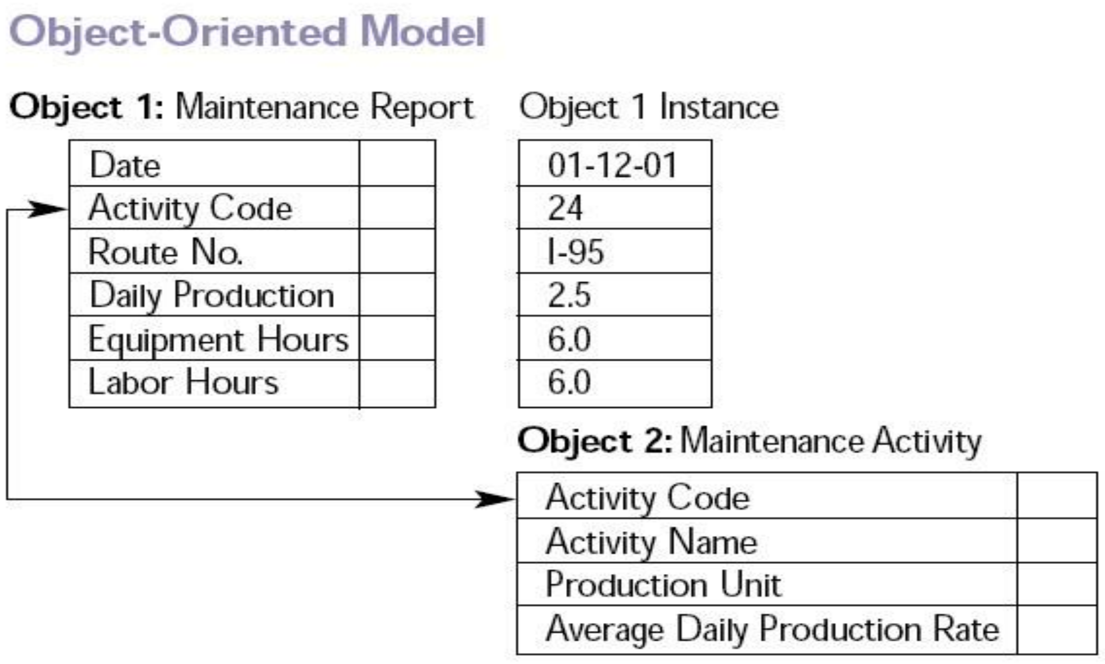
Object/relational database management systems (ORDBMSs) add new object storage capabilities to the relational systems. These new facilities integrate management of traditional fielded data, complex objects such as time-series and geospatial data and diverse binary media such as audio, video, images, and applets. By encapsulating methods with data structures, an ORDBMS server can execute complex analytical and data manipulation operations to search and transform multimedia and other complex objects.

As an evolutionary technology, the object/relational (OR) approach has inherited the robust transaction- and performance-management features of relational DBMS and the flexibility of object-oriented DBMS. Database designers can work with familiar tabular structures and data definition languages (DDLs) while assimilating new object-management possibilities. Query and procedural languages and call interfaces in

ORDBMSs are familiar: SQL3, vendor procedural languages, and ODBC, JDBC, and proprietary call interfaces are all extensions of RDBMS languages and interfaces. And the leading vendors of ORDBMS are IBM, Informix, and Oracle.

## Object-Oriented Model

Object DBMSs add database functionality to object programming languages. They bring much more than persistent storage of programming language objects. Object DBMSs extend the semantics of the C++, Smalltalk and Java object programming languages to provide full-featured database programming capability, while retaining native language compatibility. A major benefit of this approach is the unification of the application and database development into a seamless data model and language environment. As a result, applications require less code, use more natural data modeling, and code bases are easier to maintain. Object developers can write complete database applications with a modest amount of additional effort.
According to Rao (1994), "The object-oriented database (OODB) paradigm is the combination of object-oriented programming language (OOPL) systems and persistent systems. The power of the OODB comes from the seamless treatment of both persistent data, as found in databases, and transient data, as found in executing programs."
In contrast to a relational DBMS where a complex data structure must be flattened out to fit into tables or joined together from those tables to form the in-memory structure, object DBMSs have no performance overhead to store or retrieve a web or hierarchy of interrelated objects. This one-to-one mapping of object programming language objects to database objects has two benefits over other storage approaches: it provides higher performance management of objects, and it enables better management of the complex interrelationships between objects. This makes object DBMSs better suited to support applications such as financial portfolio risk analysis systems, telecommunications service applications, World Wide Web document structures, design and manufacturing systems, and hospital patient record systems, which have complex relationships between data. Figure below illustrates the model.



## Semi-structured Model

In semi-structured data model, the information that is normally associated with a schema is contained within the data, which is sometimes called ``self-describing". In such database there is no clear separation between the data and the schema, and the degree to which it is structured depends on the application. In some forms of semi-structured data there is no separate schema, in others it exists but only places loose constraints on the data. Semi-structured data is naturally modeled in terms of graphs which contain labels which give semantics to its underlying structure. Such databases subsume the modeling power of recent extensions of flat relational databases, to nested databases which allow the nesting (or encapsulation) of entities, and to object databases which, in addition, allow cyclic references between objects.
Semi-structured data has recently emerged as an important topic of study for a variety of reasons. First, there are data sources such as the Web, which we would like to treat as databases but which cannot be constrained by a schema. Second, it may be desirable to have an extremely flexible format for data exchange between disparate databases. Third,

even when dealing with structured data, it may be helpful to view it as semi-structured for the purposes of browsing.

## Associative Model

The associative model divides the real-world things about which data is to be recorded into two sorts:
Entities are things that have discrete, independent existence. An entity's existence does not depend on any other thing. Associations are things whose existence depends on one or more other things, such that if any of those things ceases to exist, then the thing itself ceases to exist or becomes meaningless.
An associative database comprises two data structures:
1. A set of items, each of which has a unique identifier, a name and a type.
2. A set of links, each of which has a unique identifier, together with the unique identifiers of three other things, that represent the source source, verb and target of a fact that is recorded about the source in the database. Each of the three things identified by the source, verb and target may be either a link or an item.

## Entity-Attribute-Value (EAV) data model

The best way to understand the rationale of EAV design is to understand row modeling (of which EAV is a generalized form). Consider a supermarket database that must manage thousands of products and brands, many of which have a transitory existence. Here, it is intuitively obvious that product names should not be hard-coded as names of columns in tables. Instead, one stores product descriptions in a Products table: purchases/sales of individual items are recorded in other tables as separate rows with a product ID referencing this table. Conceptually an EAV design involves a single table with three columns, an entity (such as an olfactory receptor ID), an attribute (such as species, which is actually a pointer into the metadata table) and a value for the attribute (e.g., rat). In EAV design, one row stores a single fact. In a conventional table that has one column per attribute, by contrast, one row stores a set of facts. EAV design is appropriate when the number of parameters that potentially apply to an entity is vastly more than those that actually apply to an individual entity.

## Context Model

The context data model combines features of all the above models. It can be considered as a collection of object-oriented, network and semi-structured models or as some kind of object database. In other words this is a flexible model; you can use any type of database structure depending on task. Such data model has been implementedinDBMSConteXt. The fundamental unit of information storage of ConteXt is a CLASS. Class contains METHODS and describes OBJECT. The Object contains FIELDS and PROPERTY. The field may be composite, in this case the field contains Sub Fields etc. The property is a set of fields that belongs to particular Object. (Similar to AVL database). In other words, fields are permanent part of Object but Property is itsvariablepart. The header of Class contains the definition of the internal structure of the Object, which includes the description of each field, such as their type, length, attributes and name. Context data model has a set of predefined types as well as user defined types. The predefined types include not only character strings, texts and digits but also pointers (references) and aggregate types (structures).

A context model comprises three main data types: REGULAR, VIRTUAL and REFERENCE. A regular (local) field can be ATOMIC or COMPOSITE. The atomic field has no inner structure. In contrast, a composite field may have a complex structure, and its type is described in the header of Class. The composite fields are divided into STATIC and DYNAMIC. The type of a static composite field is stored in the header and is permanent. Description of the type of a dynamic composite field is stored within the Object and can vary from Object to Object.

Like a NETWORK database, apart from the fields containing the information directly, context database has fields storing a place where this information can be found, i.e. POINTER (link, reference) which can point to an Object in this or another Class. Because main addressed unit of context database is an Object, the pointer is made to Object instead of a field of this Object. The pointers are divided on STATIC and DYNAMIC. All pointers that belong to a particular static pointer type point to the same Class (albeit, possibly, to different Object). In this case, the Class name is an integral part of the that pointer type. A dynamic pointer type describes pointers that may refer to different Classes. The Class, which may be linked through a pointer, can reside on the same or any other computer on the local area network. There is no hierarchy between Classes and the pointer can link to any Class,includingitsown.

In contrast to pure object-oriented databases, context databases is not so coupled to the programming language and doesn't support methods directly. Instead, method invocation is partially supported through the concept of VIRTUAL fields.

A VIRTUAL field is like a regular field: it can be read or written into. However, this field is not physically stored in the database, and in it does not have a type described in the scheme. A read operation on a virtual field is intercepted by the DBMS, which invokes a method associated with the field and the result produced by that method is returned. If no method is defined for the virtual field, the field will be blank. The METHODS is a subroutine written in C++ by an application programmer. Similarly, a write operation on a virtual field invokes an appropriate method, which can changes the value of the field. The current value of virtual fields is maintained by a run-time process; it is not preserved between sessions. In object-oriented terms, virtual fields represent just two public methods: reading and writing. Experience shows, however, that this is often enough in practical applications. From the DBMS point of view, virtual fields provide transparent interface to such methods via an application written by application programmer.

A context database that does not have composite or pointer fields and property is essentially RELATIONAL. With static composite and pointer fields, context database become OBJECT-ORIENTED. If the context database has only Property in this case it is an ENTITY-ATTRIBUTE-VALUE database. With dynamic composite fields, a context database becomes what is now known as a SEMISTRUCTURED database. If the database has all available types... in this case it is ConteXt database!

## SUMMARY

Databases and database systems have become an essential part of our everyday life. We encounter several activities that involve some interaction with a database almost daily. File based systems were an early attempt to computerize the manual filing system. This system has certain limitations. In order to overcome the limitations of file-based system, a new approach, a database approach, emerged. A database is a collection of logically related data. This has a large number of advantages over the file-based approach. These systems are very complex, sophisticated software applications that provide reliable management of large amounts of data.

There are two different ways to look at the architecture of a DBMS: the logical DBMS architecture and the physical DBMS architecture. The logical architecture deals with the way data is stored and presented to users, while the physical architecture is concerned with the software components that make up a DBMS. The physical architecture describes the software components used to enter and process data, and how these software components are related and interconnected. At its most basic level the physical DBMS architecture can be broken down into two parts: the back end and the front end.

The logical architecture describes how data in the database is perceived by users. It is not concerned with how the data is handled and processed by the DBMS, but only with how it looks. The method of data storage on the underlying file system is not revealed, and the users can manipulate the data without worrying about where it is located or how it is actually stored. This results in the database having different levels of abstraction such as the internal or physical level, the conceptual level, and the external or view level. The objective of the three level architecture is to separate each user's view of the database from the way the database is physically represented. Finally, we have a brief introduction to the concepts of database Models.

## Exercise:

**(A). State whether the following are True or False.**

1. The external schema defines how and where data are organized in physical data storage.
2. A schema separates the physical aspects of data storage from the logical aspects of data representation.
3. The conceptual schema defines a view or views of the database for particular users.
4. A collection of data designed to be used by different people is called a database.
5. In a database, the data are stored in such a fashion that they are independent of the programs of people using the data.
6. Using a database redundancy can be reduced.
7. The data in a database cannot be shared.
8. Security restrictions are impossible to apply in a database.
9. In a database data integrity can be maintained.
10. Independence means that the three levels in the schema (internal, conceptual and external) should be independent of each other so that the changes in the schema at one level should not affect the other levels.

**(B). Answer the following questions.**

1. What is a database? What is a Database Management System?

2. Illustrate and Explain Three schema architecture of DBMS?
3. Explain and illustrate the Physical architecture of a DBMS?
4. What are the components of a commercial architecture of a DBMS? Explain each.
5. Explain different types of Data Models used in Database Management Systems?
6. What is Data Dictionary? What is it used for.
7. What are the limitations of a file based approach for data manipulation?
8. What are the characteristic features of a DBMS?
9. Explain advantages and disadvantages of a DBMS?
10. Why do we need databases? Explain giving real life examples

**Further/Suggested Readings**

**1.** Elmasri R., Navathe B., “Fundamentals of Database Systems”, Pearson Education.

**2.** Desai Bipin C., “An Introduction to Database Systems”, Galgotia Publications Pvt. Ltd.

**3.** Silberschatz A., Korth Henry, Sudarshan S., “Database System Concepts”, McGraw Hill Publications.