UNIT 1

Unit – I: INTRODUCTION Database system, Characteristics (Database Vs File System), Database Users(Actors on Scene, Workers behind the scene), Advantages of Data base systems, Database applications.
Brief introduction of different Data Models; Concepts of Schema, Instance and data independence; Three tier schema architecture for data independence; Database system structure, environment, Centralized and Client Server architecture for the database.

Data: Data is meaningful known raw facts that can be processed and stored as information.

Database: Database is a collection of interrelated and organized data. In general, it is a collection of files.

Any database which is collection of data, typically describes the activities of one or more related organizations. A database management system (DBMS) is a software designed to assist in maintaining and utilizing large collections of data. A DBMS contains information pertaining to a particular enterprise.

Database Management System: Any Database Management System is

- A collection of interrelated data
- A set of programs to access the data
- An environment, which is efficient and convenient to use.

History:

First general-purpose DBMS, designed by Charles Bachman in the early 1960s which formed as basis for network data model which strongly influenced database systems through the 1960s. In the late 1960s, IBM developed the Information Management System (IMS) DBMS, which formed the basis for an alternative data representation framework called the hierarchical data model.

In 1970, Edgar Codd, at IBM's San Jose Research Laboratory, proposed a new data representation framework called the relational data model.

In the 1980s, the relational model consolidated its position as the dominant DBMS paradigm, and database systems continued to gain widespread use. The SQL query language for relational databases, developed as part of IBM's System R project, is now the standard query language.

In the late 1980s and the 1990s, advances were made in many areas of database systems. Considerable research was carried out into more powerful query languages and richer data models, with emphasis placed on supporting complex analysis of data from all parts of an enterprise.

Database management continues to gain importance as more and more data is brought online and made ever more accessible through computer networking. Commercially, database management systems represent one of the largest and most vigorous market segments.

Applications of DBMS:

Databases touch every area of our life. The list of few areas where they are extensively used is:

- Banks
- Universities
- Airlines
- E-Commerce
- Stock Exchanges
- Weather Forecast
- Manufacturing Assemblies
- Human Resource

CHARACTERISTICS (DATABASE vs FILE SYSTEMS):

FILE SYSTEMS: In earlier days, the databases were created directly on top of file systems. File system has many disadvantages.

- Not enough primary memory to process large data sets. If data is maintained in other storage devices like disks, tapes and bringing relevant data to main memory, it increases the cost of performance.
- Problem in accessing the large data due to addressing the data using 32 bit or 64 bit mode addressing mechanism.
- Programs must be written to process the user request to process the data stored in files which are complex in nature because of large volume of data to be searched.
- Inconsistent data and complexity in providing concurrent accesses.
- Not sufficiently flexible to enforce security policies in which different users have permission to access different subsets of the data.

DATABASE MANAGEMENT SYSTEM: The database management system with the help of application programs, is able to address all the issues or problems of a file system. By storing data in a DBMS rather than as a collection of operating system files, we can use the DBMS's features to manage the data in a robust and efficient manner. As the volume of data and the number of users grow hundreds of gigabytes of data and thousands of users are common in current corporate databases DBMS support becomes indispensable.

- **Data Independence:** Application programs should not, ideally, be exposed to details of data representation and storage, The DBMS provides an abstract view of the data that hides such details.
- **Efficient Data Access:** A DBMS utilizes a variety of sophisticated techniques to store and retrieve data efficiently. This feature is especially important if the data is stored on external storage devices.
- **Data Integrity and Security:** If data is always accessed through the DBMS, the DBMS can enforce integrity constraints. For example, before inserting salary information for an employee, the DBMS can check that the department budget is not exceeded. Also, it can enforce *access controls* that govern what data is visible to different classes of users.
- **Data Administration:** Experienced professionals who understand the nature of the data being managed, and how different groups of users use it, can be responsible for organizing the data representation to minimize redundancy and for fine-tuning the storage of the data to make retrieval efficient.
- **Concurrent Access and Crash Recovery:** A DBMS schedules concurrent accesses to the data in such a manner that users can think of the data as being accessed by only one user at a time. Further, the DBMS protects users from the effects of system failures.

Reasons for not choosing DBMS: Though there are several advantages with DBMS, some applications, with tight real-time constraints or just a few well-defined critical operations for which efficient custom code must

be written are not choosing DBMS. The reasons are:

1. A DBMS is a complex piece of software, optimized for certain kinds of workloads and its performance may not be adequate for certain specialized applications.
2. An application may need to manipulate the data in ways not supported by the query language. In such a situation, the abstract view of the data presented by the DBMS does not match the application's needs and actually gets in the way.

DATABASE USERS:

These apply to "large" databases, not "personal" databases that are defined, constructed, and used by a single person like Microsoft Access. Users may be divided into 2 categories.

1. Actors on the scene
2. Workers behind the scene

ACTORS ON THE SCENE: Those who actually use and control the database content, and those who design, develop and maintain database applications

1. **Database Administrator** (DBA): This is the chief administrator, who oversees and manages the database system (including the data and software). Duties include authorizing users to access the database, coordinating/monitoring its use, acquiring hardware/software for upgrades, etc. In large organizations, the DBA might have a support staff.

2. **Database Designers**: They are responsible for identifying the data to be stored and for choosing an appropriate way to organize it. They also define **views** for different categories of users. The final design must be able to support the requirements of all the user sub-groups.

3. **End Users**: These are persons who access the database for **querying**, **updating**, and **report generation**. They are main reason for database's existence.

- **Casual end users:** use database occasionally, needing different information each time; use query language to specify their requests; typically middle- or high-level managers.
- **Naive/Parametric end users**: Typically the biggest group of users; frequently query/update the database using standard **canned transactions** that have been carefully programmed and tested in advance. Examples:
  - bank tellers check account balances, post withdrawals/deposits.
  - reservation clerks for airlines, hotels, etc., check availability of seats/rooms and make reservations.
  - shipping clerks (e.g., at UPS) who use buttons, bar code scanners, etc., to update status of in-transit packages.
- **Sophisticated end users**: engineers, scientists, business analysts who implement their own applications to meet their complex needs.
- **Stand-alone users**: Use "personal" databases, possibly employing a special-purpose (e.g., financial) software package. Mostly maintain personal databases using ready-to-use packaged applications. An example is a tax program user that creates its own internal database. Another example is maintaining an address book.

4. **System Analysts, Application Programmers, Software Engineers**:

- **System Analysts**: determine needs of end users, especially naive and parametric users, and develop specifications for canned transactions that meet these needs.
- **Application Programmers**: Implement, test, document, and maintain programs that satisfy the specifications mentioned above.
- **Software Engineers:** Analysts and programmers are commonly referred to as software engineers-should be familiar with the full range of capabilities provided by the DBMS to accomplish their tasks.

WORKERS BEHIND THE SCENE: Those who design and develop the DBMS software and related tools, and the computer systems operators.

1. **DBMS system designers/ implementers:** provide the DBMS software that is at the foundation of all.
2. **Tool developers**: design and implement software tools facilitating database system design, performance monitoring, creation of graphical user interfaces, prototyping, etc.
3. **Operators and maintenance personnel**: responsible for the day-to-day operation of the system.

ADVANTAGES OR CAPABILITIES OF DATABASE SYSTEMS:

1. **Controlling Redundancy:** Data redundancy leads to wasted storage space, duplication of effort (when multiple copies of a datum need to be updated), and a higher likelihood of the introduction of inconsistency. On the other hand, redundancy can be used to improve performance of queries. Indexes, for example, are entirely redundant, but help the DBMS in processing queries more quickly.

2. **Restricting Unauthorized Access:** A DBMS should provide a security and authorization subsystem, which is used for specifying restrictions on user accounts. Common kinds of restrictions are to allow read-only access (no updating), or access only to a subset of the data

3. **Providing Persistent Storage for Program Objects:** Object-oriented database systems make it easier for complex runtime objects (e.g., lists, trees) to be saved in secondary storage so as to survive beyond program termination and to be retrievable at a later time.

4. **Providing Storage Structures for Efficient Query Processing:** The DBMS maintains indexes (typically in the form of trees and/or hash tables) that are utilized to improve the execution time of queries and updates. The choice of which indexes to create and maintain is part of physical database design and tuning is the responsibility of the DBA.

The **query processing and optimization** module is responsible for choosing an efficient query execution plan for each query submitted to the system.

5. **Providing Backup and Recovery:** The subsystem having this responsibility ensures that recovery is possible in the case of a system crash during execution of one or more transactions.

6. **Providing Multiple User Interfaces:** For example, query languages for casual users, programming language interfaces for application programmers, forms and/or command codes for parametric users, menu-driven interfaces for stand-alone users.

7. **Representing Complex Relationships Among Data:** A DBMS should have the capability to represent such relationships and to retrieve related data quickly.

8. **Enforcing Integrity Constraints:** Most database applications are such that the of the data require that it satisfy certain restrictions in order to make sense. Perhaps the most fundamental constraint on a data item is its data type, which specifies the universe of values from which its value may be drawn.
Another kind of constraint is referential integrity, which says that if the database includes an entity that refers to another one, the latter entity must exist in the database.

9. **Permitting Inferencing and Actions Via Rules:** In a deductive database system, one may specify declarative rules that allow the database to infer new data. E.g., Figure out which students are on academic probation. Such capabilities would take the place of application programs that would be used to ascertain such information otherwise. **Active** database systems go one step further by allowing "active rules" that can be used to initiate actions automatically.

DATABASE APPLICATIONS

1. Early Database Applications Using Hierarchical and Network Systems

Many early database applications, such as corporations, universities, hospitals, and banks. In many of these applications, there were large numbers of records of similar structure. One of the main problems with early database systems was the intermixing of conceptual relationships with the physical storage and placement of records on disk.

Although this provided very efficient access for the original queries and transactions that the database was designed to handle, it did not provide enough flexibility to access records efficiently when new queries and transactions were identified. In particular, new queries that required a different storage organization for efficient processing were quite difficult to implement efficiently. It was also quite difficult to reorganize the database when changes were made to the requirements of the application. They provided only programming language interfaces. This made it time-consuming and expensive to implement new queries and transactions, since new programs had to be written, tested, and debugged.

2. Providing Application Flexibility with Rational Databases

Relational databases were originally proposed to separate the physical storage of data from its conceptual representation and to provide a mathematical foundation for databases. The relational data model also introduced high-level query languages that provided an alternative to programming language interfaces; hence, it was a lot quicker to write new

queries. Early experimental relational systems and the commercial RDBMSs (relational database management systems) were quite slow. With the development of new storage and indexing techniques and better query processing and optimization, their performance improved. Eventually, relational databases became the dominant type of database systems for traditional database applications.

3. Object-Oriented Applications and need for more Complex Databases

The emergence of object-oriented programming languages in the 1980s and the need to store and share complex-structured objects led to the development of object-oriented databases. Initially, they were considered a competitor to relational databases, since they provided more general data structures. They also incorporated many of the useful object oriented paradigms, such as abstract data types, encapsulation of operations, inheritance, and object identity.

The complexity of the model and the lack of an early standard contributed to their limited use.

Now mainly used in specialized applications, such as engineering design, multimedia publishing, and manufacturing systems.

4. Interchanging Data in the web for E-Commerce

In the 1990s, electronic commerce (e-commerce) emerged as a major application on the Web. It quickly became apparent that parts of the information on e-commerce Web pages were often dynamically extracted data from DBMSs. A variety of techniques were developed to allow the interchange of data on the Web.

5. Extending Database Capabilities for new Applications

The success of database systems in traditional applications encouraged developers of other types of applications to attempt to use them. The following are examples of these applications:

- Scientific applications that store large amounts of data resulting from scientific experiments in areas such as high-energy physics or the mapping of the human genome.

• Storage and retrieval of images, from scanned news or personal photographs to satellite photograph images and images from medical procedures such as X-rays or MRI (magnetic resonance imaging).

• Storage and retrieval of videos, such as movies, or video clips from news or personal digital cameras.

• Data mining applications that analyze large amounts of data searching for the occurrences of specific patterns or relationships.

• Spatial applications that store spatial locations of data such as weather information or maps used in geographical information systems.

• Time series applications that store information such as economic data at regular points in time, for example, daily sales or monthly gross national product figures.

It was quickly apparent that basic relational systems were not very suitable for many of these applications, usually for one or more of the reasons like need for more complex data structures, new data types, new operations and query language constructs, new storage and indexing structures.

This led DBMS developers to add some general purpose and special purpose functionality to their systems.

INTRODUCTION OF DIFFERENT DATA MODELS

A data model is a collection of concepts that can be used to describe the structure of a database and provides the necessary means to achieve this abstraction whereas structure of a database means the data types, relationships and constraints that should hold on the data.

Collection of conceptual tools for describing data, data relationships, data semantics and consistency constraints. The various data models that have been proposed fall into three different groups.

1. Object based logical models
2. Record-based logical models
3. Physical models

1. OBJECT BASED LOGICAL MODELS: They are used in describing data at the logical and view levels. They are characterized by the fact that they provide fairly flexible structuring capabilities and allow data constraints to be specified explicitly. There are many different models mostly known ones are:

- The E-R model
- The object-oriented model
- The semantic data model
- The functional data model

The E-R model: The (E-R) data model is based on a perception of a real worker that consists of a collection of basic objects, called entities, and of relationships among these objects. The overall logical structure of a database can be expressed graphically by an E-R diagram. Which is built up by the following components:

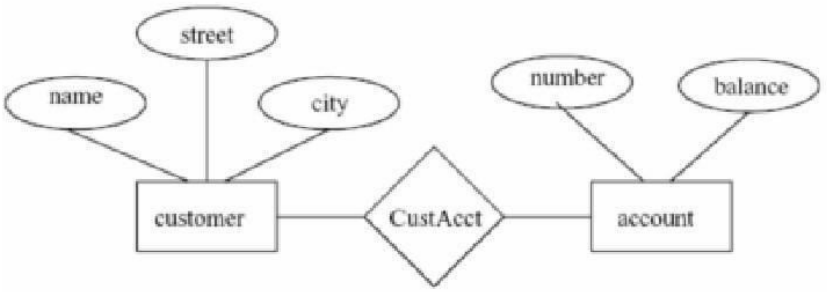
Rectangles, which represent entity sets

Ellipses, which represent attributes

Diamonds, which represent relationships among entity sets

Lines, which link attributes to entity sets and entity sets to relationships

E.g. suppose we have two entities like customer and account, then these two entities can be modeled as follows.



The Object-Oriented Model:

Like the E-R model the object-oriented model is based on a collection of objects. An object contains values stored in instance variables within the object. An object also contains bodies of code that operate on the object. These bodies of code are called methods. Classes: It is the collection of objects which consist of the same types of values and the same methods.

E.g. account number & balance are instance variables; pay-interest is a method that uses the above two variables and adds interest to the balance.

Semantic Models:

These include the extended relational, the semantic network and the functional models. They are characterized by their provision of richer facilities for capturing the meaning of data objects and hence of maintaining database integrity. Systems based on these models exist in monotype for at the time of writing and will begin to filter through the next decade.

2. RECORD-BASED LOGICAL MODELS: Record based logical models are also used in describing data at the logical and view levels. In contrast to object-based data models, they are used both to specify the overall logical structures of the database, and to provide a higher-level description of the implementation.

Record-based models are so named because the database is structured in fixed-format records of several types. Each record type defines a fixed number of fields, or attributes, and each field is usually of a fixed length. The three most widely accepted record based data models are:

- The relational Model
- Network Model
- Hierarchical Model

Relational Model: The relational model uses a collection of tables to represent both data and the relationships among those data. Each table has multiple columns, and each column has a unique name as follows:

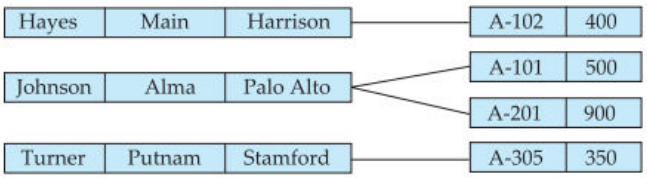
Customer (table name)

Customer–name	Customer-street	Customer-City	Account-number
Johnsons	Alma	Pala Alto	A-101
Smith	North	Ryc	A-215
Hayes	Main	Harrison	A-102
Turner	Dutnam	Stanford	A-305
Johnson	Alma	PalaAlto	A-201
Jones	Main	Harrison	A-217
Lindsay	Park	Pittifield	A-222
Smith	North	Rye	A-201

Sample Relational Database

Network Model:

Data in the network model is represented by collection of records, and relationship among data is represented by links, which can be viewed as pointers. The records in the database are organized as collections of arbitrary graphs.



Sample Network Database

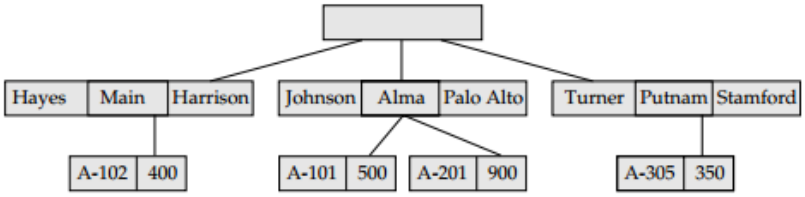
Disadvantages of Network Data Model:

a. System complexity – In a network model, data are accessed one record at a time. This makes it essential for the database designers, administrators, and programmers to be familiar with the internal data structures to gain access to the data. Therefore, a user-friendly database management system cannot be created using the network model.

b. Lack of structural independence – Making structural modifications to the database is very difficult in the network database model as the data access method is navigational. Any changes made to the database structure require the application programs to be modified before they can access data. Though the network database model achieves data independence, it still fails to achieve structural independence.

Hierarchical Model:

The hierarchical model is similar to the network model in the sense that data and relationships among data are represented by records and links, respectively. It differs from the network model in that records are organized as collection of trees rather than arbitrary graphs.



Sample Hierarchical Database

Advantages of hierarchical Model:

1. Simplicity – Since the database is based on the hierarchical structure, the relationship between the various layers is logically simple. Thus, the design of a hierarchical database is simple.

2. Data Security–Hierarchical model was the first database that offered the data security that is provided and enforced by the DBMS.

3. Data Integrity – Since the hierarchical model is based on the parent/child relationship, there is always a link between the parent segment and the child segment under it. The child segments are always automatically referenced to its parent, this model promotes data integrity.

4. Efficiency – The hierarchical database model is a very efficient one when the database contains a large number of one-to-many relationships and when the users require large number of transactions, using data whose relationships are fixed.

Disadvantages of hierarchical Model:

1. Implementation Complexity – Although the hierarchical database model is conceptually simple and easy to design, it is quite complex to implement. The database designers should have very good knowledge of the physical storage characteristics.

2. Database management problems

3. Lack of structural independence – Structural independence exists when the changes made to the database structure does not affect the DBMS's ability to access data. Hierarchical database systems use physical storage paths to navigate to the different data segments. So the application programmer should have a good knowledge of the relevant access paths to access the data. So if the physical structure is changed the applications will also have to be altered. us, in a hierarchical database the benefits of data independence are limited by structural dependence.

4. Programming complexity – Due to the structural dependence and the navigational structure, the application programmers and the end users must know precisely how the data is distributed physically in the database in order to access data. This requires knowledge of complex pointer systems, which is difficult for users who have little or no Programming knowledge.

5. Implementation limitation – Many of the common relationships do not conform to the one-to-many format required by the hierarchical model. The many-to-many relationships, which are more common in real life, are very difficult to implement in a hierarchical model.

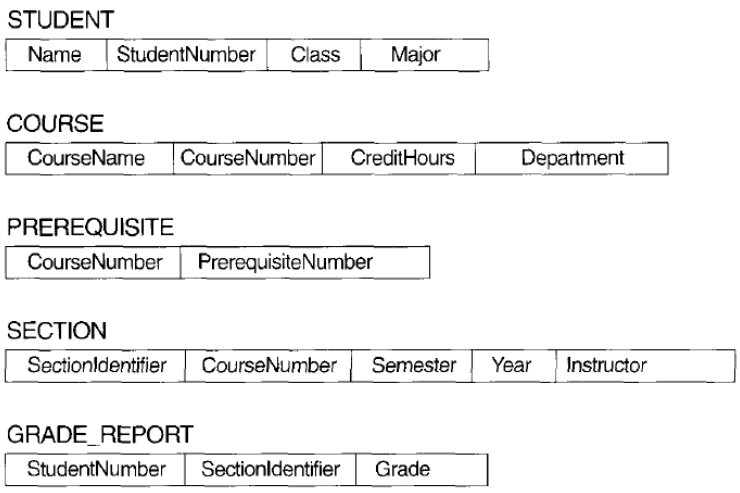
3. PHYSICAL DATA MODELS

Physical data models are used to describe data at the lowest level. In contrast to logical data models, there are few physical data models in use. Two of the widely known ones are the unifying model and the frame-memory model.

CONCEPTS OF SCHEMA AND INSTANCE

It is important to distinguish between the *description* of the database and the *database itself.*

Database Schema: The description of a database is called the database schema, which is specified during database design and is not expected to change frequently. Most models have certain conventions for displaying schemas as diagrams. A displayed schema is called a schema diagram.



Schema Diagram

The diagram displays the structure of each record type but not the actual instances of records. Each object in the schema is known as **schema construct.**

A schema diagram displays only *some aspects* of a schema, such as the names of record types and data items, and some types of constraints. Other aspects are not specified in the schema diagram. for example, It shows neither the data type of each data item nor the relationships among the various files. Many types of constraints are not represented in schema diagrams.

Database Instance: The data in the database at a particular moment in time is called a database state or database instance or snapshot. It is also called the *current* set of occurrences or instances in the database. In a given database state, each schema construct has its own *current set* of instances; for example, the STUDENT construct will contain the set of individual student entities (records) as its instances. Many database states can be constructed to correspond to a particular database schema. Every time we insert or delete a record or change the value of a data item in a record, we change one state of the database into another state.

STUDENT	Name	StudentNumber	Class	Major
	Smith	17	1	CS
	Brown	8	2	CS

COURSE	CourseName	CourseNumber	CreditHours	Department
	Intro to Computer Science	CS1310	4	CS
	Data Structures	CS3320	4	CS
	Discrete Mathematics	MATH2410	3	MATH
	Database	CS3380	3	CS

SECTION	SectionIdentifier	CourseNumber	Semester	Year	Instructor
	85	MATH2410	Fall	98	King
	92	CS1310	Fall	98	Anderson
	102	CS3320	Spring	99	Knuth
	112	MATH2410	Fall	99	Chang
	119	CS1310	Fall	99	Anderson
	135	CS3380	Fall	99	Stone

GRADE_REPORT	StudentNumber	SectionIdentifier	Grade
	17	112	B
	17	119	C
	8	85	A
	8	92	A
	8	102	B
	8	135	A

PREREQUISITE	CourseNumber	PrerequisiteNumber
	CS3380	CS3320
	CS3380	MATH2410
	CS3320	CS1310

Database Instance for the above schema

When we define a new database, we specify its database schema only to the DBMS. At this point, the corresponding database state is the *empty state* with no data. We get the *initial state* of the database when the database is first populated or loaded with the initial data. From then on, every time an update operation is applied to the database, we get another database state. At any point in time, the database has a *current state.* The DBMS is partly responsible for ensuring that *every* state of the database is a valid state i.e., a state that satisfies the structure and constraints specified in the schema.

The DBMS stores the descriptions of the schema constructs and constraints, also called the **meta-data** in the DBMS catalog so that DBMS software can refer to the schema whenever it needs to. The schema is sometimes called the **intension**, and a database state an **extension** of the schema.

THREE TIRE SCHEMA ARCHITECTURE FOR DATA INDEPENDENCE

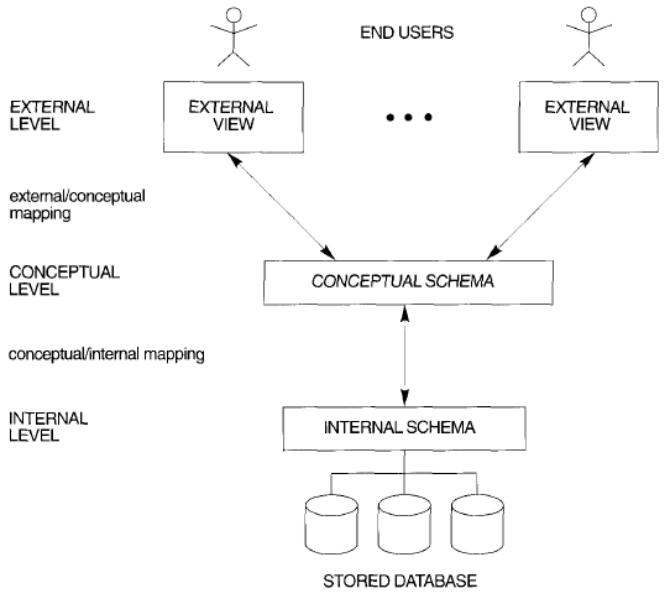
The goal of the three-schema architecture is to separate the user applications and the physical database. In this architecture, schemas can be defined at the following three levels:

1. Internal Level
2. Conceptual Level
3. External or View Level

1. The **internal level** has an internal schema, which describes the physical storage structure of the database. The internal schema uses a physical data model and describes the complete details of data storage and access paths for the database.

2. The **conceptual level** has a conceptual schema, which describes the structure of the whole database for a community of users. The conceptual schema hides the details of physical storage structures and concentrates on describing entities, data types, relationships, user operations, and constraints. Usually, a representational data model is used to describe the conceptual schema when a database system is implemented. This *implementation conceptual schema* is often based on a *conceptual schema design* in a high-level data model.

3. The **external or view level** includes a number of external schemas or user views. Each external schema describes the part of the database that a particular user group is interested in and hides the rest of the database from that user group. As in the previous case, each external schema is typically implemented using a representational data model, possibly based on an external schema design in a high level data model.



THREE TIER SCHEMA ARCHITECTURE

The three-schema architecture is a convenient tool with which the user can visualize the schema levels in a database system.

The three schemas are only *descriptions* of data; the only data that *actually* exists is at the physical level. In a DBMS based on the three-schema architecture, each user group refers only to its own external schema. Hence, the DBMS must transform a request specified on an external schema into a request against the conceptual schema, and then into a request on the internal schema for processing over the stored database. If the request is a database retrieval, the data extracted from the stored database must be reformatted to match the user's external view. The processes of transforming requests and results between levels are called mappings. These mappings may be time-consuming, so some DBMSs-especially those that are meant to support small databases-do not support external views.

DATA INDEPENDENCE:

Data independence can be defined as the capacity to change the schema at one level of a database system without having to change the schema at the next higher level. We can define two types of data independence:

1. **Logical data independence** is the capacity to change the conceptual schema without having to change external schemas or application programs. We may change the conceptual schema to expand the database (by adding a record type or data item), to change constraints, or to reduce the database (by removing a record type or data item). After the conceptual schema undergoes a logical reorganization, application programs that reference the external schema constructs must work as before. Changes to constraints can be applied to the conceptual schema without affecting the external schemas or application programs.

2. **Physical data independence** is the capacity to change the internal schema without having to change the conceptual schema. Hence, the external schemas need not be changed as well. Changes to the internal schema may be needed because some physical files had to be reorganized-for example, by creating additional access structures-to improve the performance of retrieval or update. If the same data as before remains in the database, we should not have to change the conceptual schema.

Data independence occurs because when the schema is changed at some level, the schema at the next higher level remains unchanged; only the *mapping* between the two levels is changed. Hence, application programs referring to the higher-level schema need not be changed. The three-schema architecture can make it easier to achieve true data independence, both physical and logical.

DATABASE SYSTEM STRUCTURE

We can explain the overall structure of DBMS/System structure and its components by the diagram given below:

Database systems are partitioned into modules for different functions. Some functions (e.g. file systems) may be provided by the operating system. Broadly the functional components of a database system are:

a. Query Processor: It is one of the functional components of DBMS. It translates statements in a query language into low-level instructions the database manager understands. It may also attempt to find an equivalent but more efficient form. It contains following components:

i. DML compiler - It converts DDL statements to a set of tables containing metadata stored in a data dictionary.

It also performs query optimization.

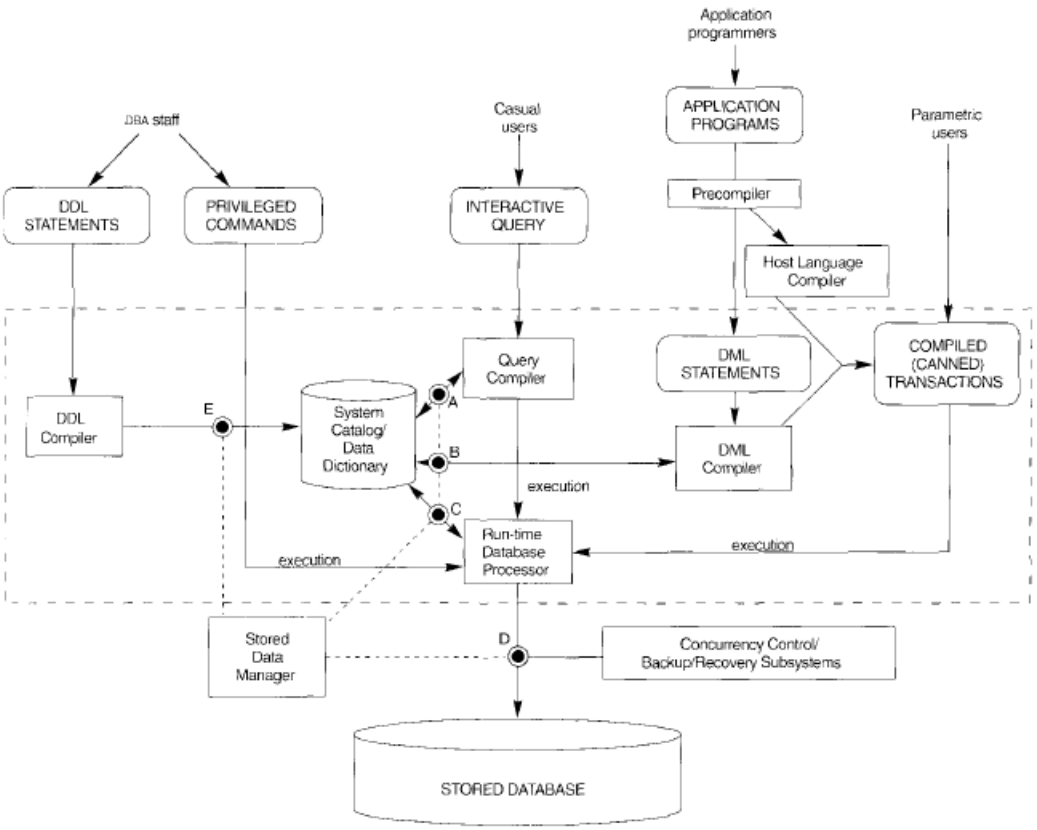
ii. DDL interpreter – It interprets DDL statements and records definitions into data dictionary.

iii. Query evaluation engine – It executes low-level instructions generated by DML compiler. They mainly deal with solving all problems related to queries and query processing. It helps database system simplify and facilitate access to data.

b. Storage Manger (Database Manager)

Storage manager is a program module that provides the interface between the low-level data stored in the database and the application programs and queries submitted to the system. The storage manager is responsible to the following tasks:

1. interaction with the file manager
2. efficient storing, retrieving and updating of data.



Database System Structure

The important components include:

i. File manager: It manages allocation of disk space and data structures used to represent information on disk.

ii. Database manager: It is the interface between low-level data and remains in a consistent (correct) state despite system failures (e.g., power failures and operating system crash s) and transaction failures.

iii. DML pre compiler: It converts DML statements embedded in an application program to normal procedure calls in a host language. The pre compiler interacts with the query processor.

vi. DDL compiler: It converts DDL statements to a set of tables containing metadata stored in a data dictionary.

v. Authorization and integrity manger – It conducts integrity checks and user authority to access data.

vi. Buff r manger – It is critical part of DB and stores temporary data.

In addition, several data structures are required for physical system implementation:

a. Data files: They store the database itself.

b. Data dictionary: It stores information about the structure of the database. It is used heavily. Great emphasis should be placed on developing a good design and efficient implementation of the dictionary. In short, it stores metadata.

c. Indices: They provide fast access to data items holding particular values.

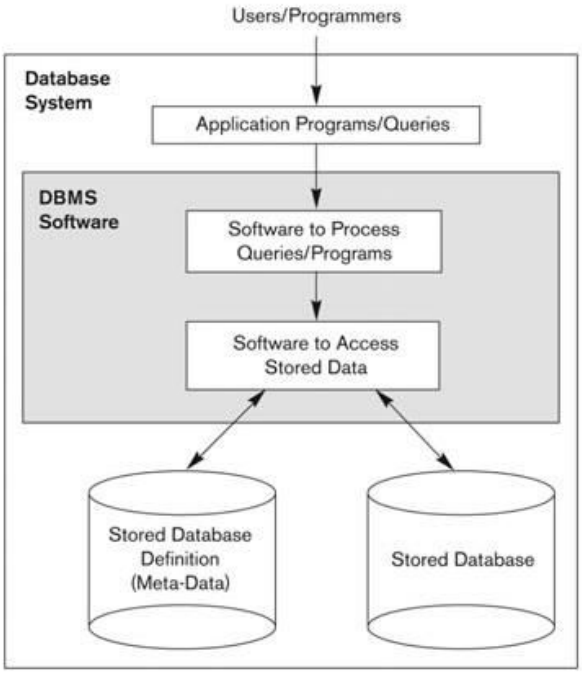
DATABASE ENVIRONMENT

A database management system (DBMS) is a collection of programs that enables users to create and maintain a database. The DBMS is hence a *general-purpose software system* that facilitates the processes of *defining, constructing, manipulating,* and *sharing* databases among various users and applications.

- **Defining** a database involves specifying the data types, structures, and constraints for the data to be stored in the database.
- **Constructing** the database is the process of storing the data itself on some storage medium that is controlled by the DBMS.
- **Manipulating** a database includes such functions as querying the database to retrieve specific data, updating the database to reflect changes in the mini world, and generating reports from the data.
- **Sharing** a database allows multiple users and programs to access the database concurrently.

Other important functions provided by the DBMS include *protecting* the database and *maintaining* it over a long period of time.

Protection includes both *system protection* against hardware or software malfunction (or crashes), and *security protection* against unauthorized or malicious access. A typical large database may have a life cycle of many years, so the DBMS must be able to maintain the database system by allowing the system to evolve as requirements change over time. we can call the database and DBMS software together a database system.



Database System Environment

CENTRALIZED AND CLIENT/SERVER ARCHITECTURE FOR DATABASE:

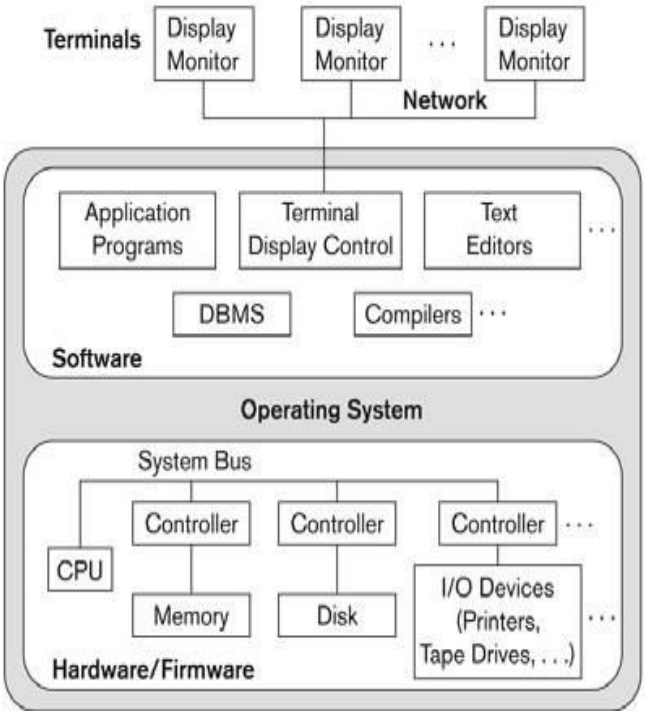
CENTRALIZED DBMS ARCHITECTURE:

Architectures for DBMSs have followed trends similar to those for general computer system architectures. Earlier architectures used mainframe computers to provide the main processing for all functions of the system, including user application programs and user interface programs, as well as all the DBMS functionality.

As prices of hardware declined, most users replaced their terminals with personal computers (PCs) and workstations. At first, database systems used these computers in the same way as they had used display

terminals, so that the DBMS itself was still a centralized DBMS in which all the DBMS functionality, application program execution, and user interface processing were carried out on one machine.

Gradually, DBMS systems started to exploit the available processing power at the user side, which led to client/server DBMS architectures.



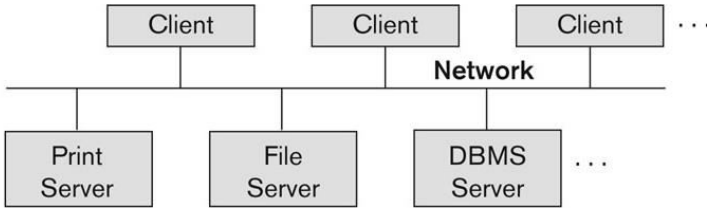
Centralized Architecture For DBMS

CLIENT/SERVER ARCHITECTURE:

Basic Client/Server Architecture:

The client/server architecture was developed to deal with computing environments in which a large number of PCs, workstations, file servers, printers, database servers, Web servers, and other equipment are connected via a network. The idea is to define specialized servers with specific functionalities.

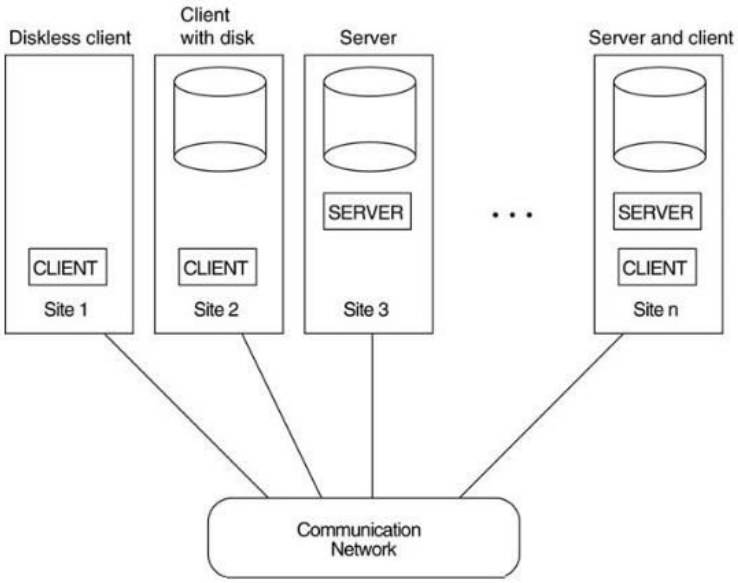
The resources provided by specialized servers can be accessed by many client machines. The client machines provide the user with the appropriate interfaces to utilize these servers, as well as with local processing power to run local applications. This concept can be carried over to software, with specialized software-such as a DBMS or a CAD (computer-aided design) package being stored on specific server machines and being made accessible to multiple clients.



Logical two-tier architecture

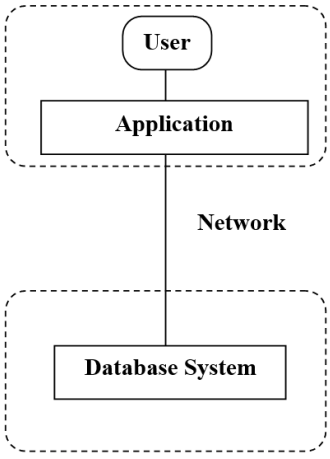
The concept of client/server architecture assumes an underlying framework that consists of many PCs and workstations as well as a smaller number of mainframe machines, connected via local area networks and other types of computer networks. A client in this framework is typically a user machine that provides user interface capabilities and local processing. When a client requires access to additional functionality-such as database access-that does not exist at that machine, it connects to a server that provides the needed functionality. A server is a machine that can provide services to the client machines, such as file access,

printing, archiving, or database access. In the general case, some machines install only client software, others only server software, and still others may include both client and server software. However, it is more common that client and server software usually run on separate machines.



Physical two-tier Architecture

In a client/server architecture, the user interface programs and application programs can run on the client side. When DBMS access is required, the program establishes a connection to the DBMS (which is on the server side); once the connection is created, the client program can communicate with the DBMS. A standard called Open Database Connectivity (ODBC) provides an application programming interface (API), which allows client-side programs to call the DBMS, as long as both client and server machines have the necessary software installed. Most DBMS vendors provide ODBC drivers for their systems. Hence, a client program can actually connect to several RDBMSs and send query and transaction requests using the ODBC API, which are then processed at the server sites. Any query results are sent back to the client program, which can process or display the results as needed. A related standard for the Java programming language, called JDBC, allows Java client programs to access the DBMS through a standard interface.



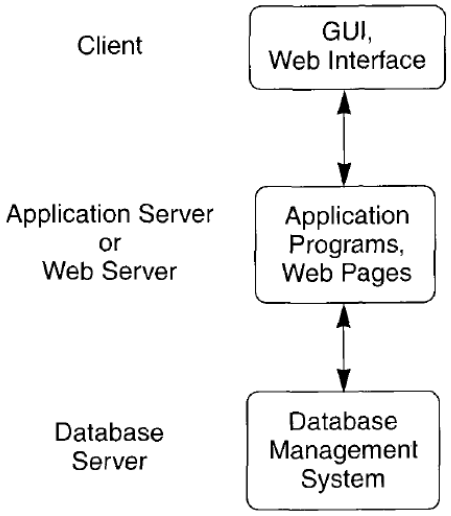
Two Tier Architecture

Three-Tier Client/Server Architectures for Web Applications:

Many Web applications use an architecture called the three-tier architecture, which adds an intermediate layer between the client and the database server. This intermediate layer or middle tier is sometimes called the application server and sometimes the Web server, depending on the application.

This server plays an intermediary role by storing business rules (procedures or constraints) that are used to access data from the database server. It can also improve database security by checking a client's credentials before forwarding a request to the database server.

Clients contain GUI interfaces and some additional application-specific business rules. The intermediate server accepts requests from the client, processes the request and sends database commands to the database server, and then acts as a conduit for passing (partially) processed data from the database server to the clients, where it may be processed further and filtered to be presented to users in GUI format. Thus, the *user interface, application rules,* and *data access* act as the three tiers.



Logical Three tier Architecture

A 3-tier architecture separates its tiers from each other based on the complexity of the users and how they use the data present in the database. It is the most widely used architecture to design a DBMS. The layers in the figure are named as Database Tier, Application Tier and Presentation Tier (in bottom to up manner).

- **Database (Data) Tier** − At this tier, the database resides along with its query processing languages. We also have the relations that define the data and their constraints at this level.
- **Application (Middle) Tier** − At this tier reside the application server and the programs that access the database. For a user, this application tier presents an abstracted view of the database. End-users are unaware of any existence of the database beyond the application. At the other end, the database tier is not aware of any other user beyond the application tier. Hence, the application layer sits in the middle and acts as a mediator between the end-user and the database.
- **Presentation (User) Tier** − End-users operate on this tier and they know nothing about any existence of the database beyond this layer. At this layer, multiple views of the database can be provided by the application. All views are generated by applications that reside in the application tier.