Chapter 1 Introduction:

1.1 Concept and Applications:

Data: Collection of numbers, characters/ Data are values of qualitative or quantitative variables, belonging to a set of items.

Database: A database is an organized collection of data. The data are typically organized to model relevant aspects of reality in a way that supports processing requiring this information. For example, modeling the availability of rooms in hotels in a way that supports finding a hotel with vacancies.

Database Management System: A Database Management System is a collection of interrelated data and a set of programs to access those data. The main purpose of DBMS is to provide a way to store and retrieve database information that is both convenient and efficient. Management of data involves both defining structures for storage of information and providing mechanisms for the manipulation of information. In addition, the database system must ensure the safety of the information stored, despite system crashes or attempts at unauthorized access.

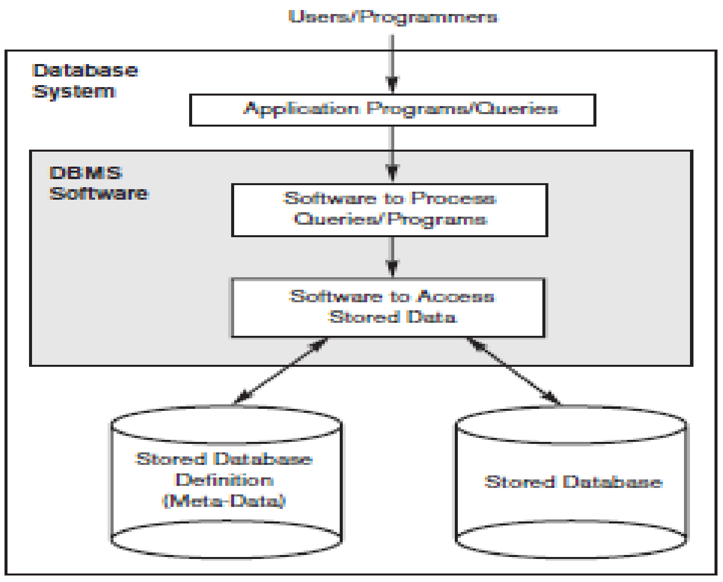

Users/Programmers
Database System
Application Programs/Queries
DBMS Software
Software to Process Queries/Programs
Software to Access Stored Data
Stored Database Definition (Meta-Data)
Stored Database

Figure 1.1
A simplified database system environment.

Applications of Database System:

Databases are widely used. Here are some representative applications:
• *Banking*: For customer information, accounts, and loans, and banking transactions.
• *Airlines*: For reservations and schedule information. Airlines were among the first to use databases in a geographically distributed manner—terminals situated around the world accessed the central database system through phone lines and other data networks.
• *Universities*: For student information, course registrations, and grades.
• *Credit card transactions*: For purchases on credit cards and generation of monthly statements.

• *Telecommunication*: For keeping records of calls made, generating monthly bills, maintaining balances on prepaid calling cards, and storing information about the communication networks.
• *Finance*: For storing information about holdings, sales, and purchases of financial instruments such as stocks and bonds.
• *Sales*: For customer, product, and purchase information.
• *Manufacturing*: For management of supply chain and for tracking production of items in factories, inventories of items in warehouses/stores, and orders for items.
• *Human resources*: For information about employees, salaries, payroll taxes and benefits, and for generation of paychecks.

1.2 Objectives of DBMS:

The objectives that the management should keep in mind when they design and organize their data base management systems are:
(i) Provide for mass storage of relevant data,
(ii) Make access to the data easy for the user,
(iii) Provide prompt response to user requests for data,
(iv) Make the latest modifications to the database available immediately,
(v) Eliminate redundant data,
(vi) Allow for multiple users to be active at one time,
(vii) Allow for growth in the database system,
(viii) Protect the data from physical harm and unauthorised access.

1.3 Needs of DBMS:

a) The ability to update and retrieve data :
This is a fundamental component of a DBMS and essential to database management. Without the ability to view or manipulate data, there would be no point to using a database system.
Updating data in a database includes adding new records, deleting existing records and changing information within a record. The user does not need to be aware of how DBMS structures this data, all the user needs to be aware of is the availability of updating and/or pulling up information, the DBMS handles the processes and the structure of the data on a disk.

b) Support Concurrent Updates :
Concurrent updates occur when multiple users make updates to the database simultaneously.
Supporting concurrent updates is also crucial to database management as this component ensures that updates are made correctly and the end result is accurate. Without DBMS intervention, important data could be lost and/or inaccurate data stored.
DBMS uses features to support concurrent updates such as batch processing, locking, two-phase locking, and time stamping to help make certain that updates are done accurately. Again, the user is not aware all this is happening as it is the database management system's responsibility to make sure all updates are stored properly.

c) Recovery of Data :
In the event a catastrophe occurs, DBMS must provide ways to recover a database so that data is not permanently lost. There are times computers may crash, a fire or other natural disaster may occur, or a user may enter incorrect information invalidating or making records inconsistent.
If the database is destroyed or damaged in any way, the DBMS must be able to recover the correct state of the database, and this process is called Recovery. The easiest way to do this is to make regular

backups of information. This can be done at a set structured time so in the event a disaster occurs, the database can be restored to the state that it was last at prior to backup

1.4 Database Systems vs File Systems:

File System	DBMS
Used to manage and organise the files stored in the hard disk of the computer	A software to store and retrieve the user's data
Redundant data is present	No presence of redundant data
Query processing is not so efficient	Query processing is efficient
Data consistency is low	Due to the process of normalization, the data consistency is high
Less complex, does not support complicated transactions	More complexity in managing the data, easier to implement complicated transactions
Less security	Supports more security mechanisms
Less expensive in comparison to DBMS	Higher cost than the File system
Does not support crash recovery	Crash recovery mechanism is highly supported

1.5 Data Abstraction:

For the system to be usable, it must retrieve data efficiently. The need for efficiency has led designers to use complex data structures to represent data in the database. Since many database-systems users are not computer trained, developers hide the complexity from users through several levels of abstraction, to simplify users' interactions with the system:

• **Physical level**. The lowest level of abstraction describes *how* the data are actually stored. The physical level describes complex low-level data structures in detail.

• **Logical level**. The next-higher level of abstraction describes *what* data are stored in the database, and what relationships exist among those data. The logical level thus describes the entire database in terms of a small number of relatively simple structures. Although implementation of the simple structures at the logical level may involve complex physical-level structures, the user of the logical level does not need to be aware of this complexity. Database administrators, who must decide what information to keep in the database, use the logical level of abstraction.

• **View level**. The highest level of abstraction describes only part of the entire database. Even though the logical level uses simpler structures, complexity remains because of the variety of information stored in a large database. Many users of the database system do not need all this information; instead, they need to access only a part of the database. The view level of abstraction exists to simplify their interaction with the system. The system may provide many views for the same database.

Figure 1.1 shows the relationship among the three levels of abstraction.

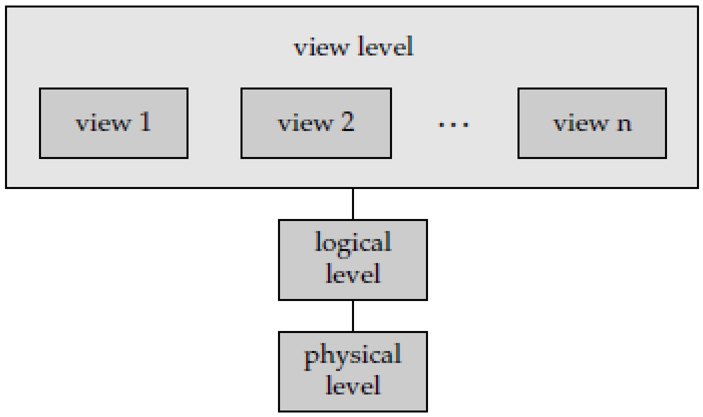


Figure 1.1 The three levels of data abstraction.

1.6 Data Independence:

IT can be defined as the capacity to change the schema at one level of a database system without having to change schema at the next higher level. There are mainly two types of Data Independence

1. Logical data independence: The ability to change the logical (conceptual) schema without changing the External schema (User View) is called logical data independence. For example, the addition or removal of new entities, attributes, or relationships to the conceptual schema should be possible without having to change existing external schemas or having to rewrite existing application programs.
2. Physical data independence: The ability to change the physical schema without changing the logical schema is called physical data independence. For example, a change to the internal schema, such as using different file organization or storage structures, storage devices, or indexing strategy, should be possible without having to change the conceptual or external schemas.

1.7 Instances and Schemas:

Databases change over time as information is inserted and deleted. The collection of information stored in the database at a particular moment is called an **instance** of the database. The overall design of the database is called the database **schema**. Schemas are changed infrequently, if at all. The concept of database schemas and instances can be understood by analogy to a program written in a programming language. A database schema corresponds to the variable declarations (along with associated type definitions) in a program. Each variable has a particular value at a given instant. The values of the variables in a program at a point in time correspond to an *instance* of a database schema. Database systems have several schemas, partitioned according to the levels of abstraction. The **physical schema** describes the database design at the physical level, while the **logical schema** describes the database design at the logical level. A database may also have several schemas at the view level, sometimes called **subschemas**, that describe different views of the database. Of these, the logical schema is by far the most important, in terms of its effect on application programs, since programmers construct applications by using the logical schema. The physical schema is hidden beneath the logical schema, and can usually be changed easily without affecting application programs. Application programs are said to exhibit **physical data independence** if they do not depend on the physical schema, and thus need not be rewritten if the physical schema changes.

1.8 Database Languages(Concept of DDL, DML, DCL)

A database system provides a **data definition language** to specify the database schema and a **data manipulation language** to express database queries and updates.

1.8.1 Data-Definition Language

We specify a database schema by a set of definitions expressed by a special language called a **data-definition language** (**DDL**).

For instance, the following statement in the SQL language defines the *account* table:

create table *account* (*account-number* **char**(10), *balance* **integer**)

Execution of the above DDL statement creates the *account* table. In addition, it updates a special set of tables called the **data dictionary** or **data directory**.

A data dictionary contains **metadata**—that is, data about data. The schema of a table is an example of metadata. A database system consults the data dictionary before reading or modifying actual data.

1.8.2 Data-Manipulation Language

Data manipulation is

- The retrieval of information stored in the database
- The insertion of new information into the database
- The deletion of information from the database
- The modification of information stored in the database

A **data-manipulation language (DML)** is a language that enables users to access or manipulate data as organized by the appropriate data model. There are basically two types:

- **Procedural DMLs** require a user to specify *what* data are needed and *how* to get those data.
- **Declarative DMLs** (also referred to as **nonprocedural** DMLs) require a user to specify *what* data are needed *without* specifying how to get those data.

The DML component of the SQL language is nonprocedural.

A **query** is a statement requesting the retrieval of information. The portion of a DML that involves information retrieval is called a **query language**. Although technically incorrect, it is common practice to use the terms *query language* and *data-manipulation language* synonymously.

This query in the SQL language finds the name of the customer whose customer-id is 192-83-7465:

select *customer.customer-name*
from *customer*
where *customer.customer-id* = 192-83-7465

1.8.3 Data Control Language(DCL): DCL languages are used to control the user access to the database, tables, views, procedures, functions and packages. They give different levels of access to the objects in the database. Some Examples:

Grant- gives user's access privileges to database.
Revoke- Withdraw access privileges given with GRANT command.

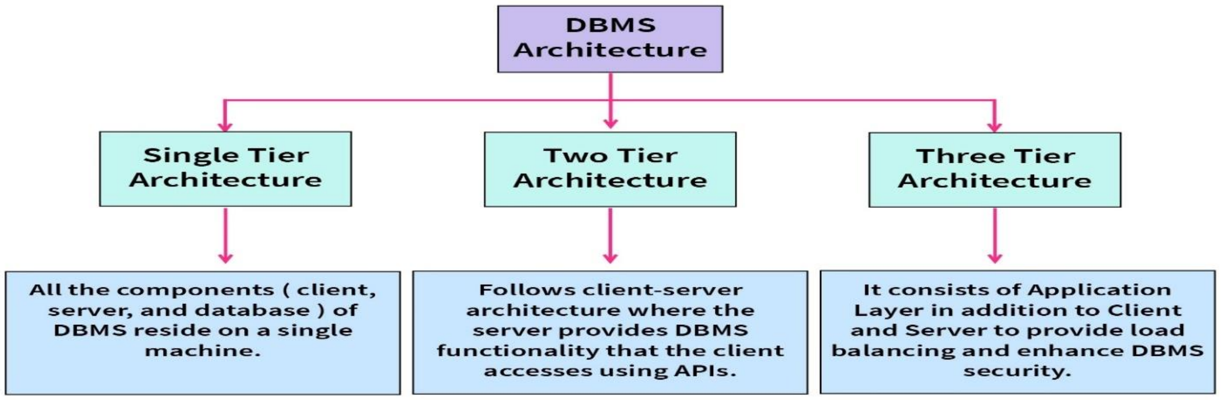
1.9 DBMS Architecture and its Types:

DBMS architecture describes the structure and how the users are connected to a specific database system. It also affects the performance of the database as it helps to design, develop, implement, and maintain the database management system. Database management systems are divided into multiple levels of abstraction for proper functioning. These modules/layers describe the functioning and the design of the DBMS.

Since a database management system is not always directly accessible by the user or an application, we can maintain it with the help of various architectures based on how the user is connected to the database. These architectures follow a **tier-based classification**, i.e., the DBMS architecture is classified depending upon how many layers are present in the structure of the DBMS.

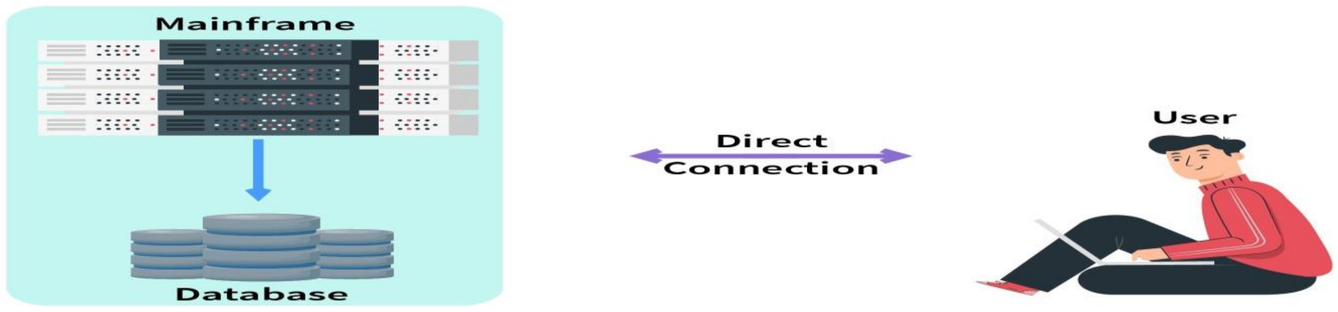
The most common DBMS architectures:

- Single Tier Architecture (One-Tier Architecture)
- Two-Tier Architecture
- Three-Tier Architecture


DBMS Architecture
Single Tier Architecture
Two Tier Architecture
Three Tier Architecture
All the components (client, server, and database) of DBMS reside on a single machine.
Follows client-server architecture where the server provides DBMS functionality that the client accesses using APIs.
It consists of Application Layer in addition to Client and Server to provide load balancing and enhance DBMS security.

1. Single Tier Architecture:

- In this architecture, the database is directly available to the user. It means the user can directly sit on the DBMS and uses it.
- Any changes done here will directly be done on the database itself. It doesn't provide a handy tool for end users.
- The 1-Tier architecture is used for development of the local application, where programmers can directly communicate with the database for the quick response.



Single Tier DBMS Architecture is used whenever:

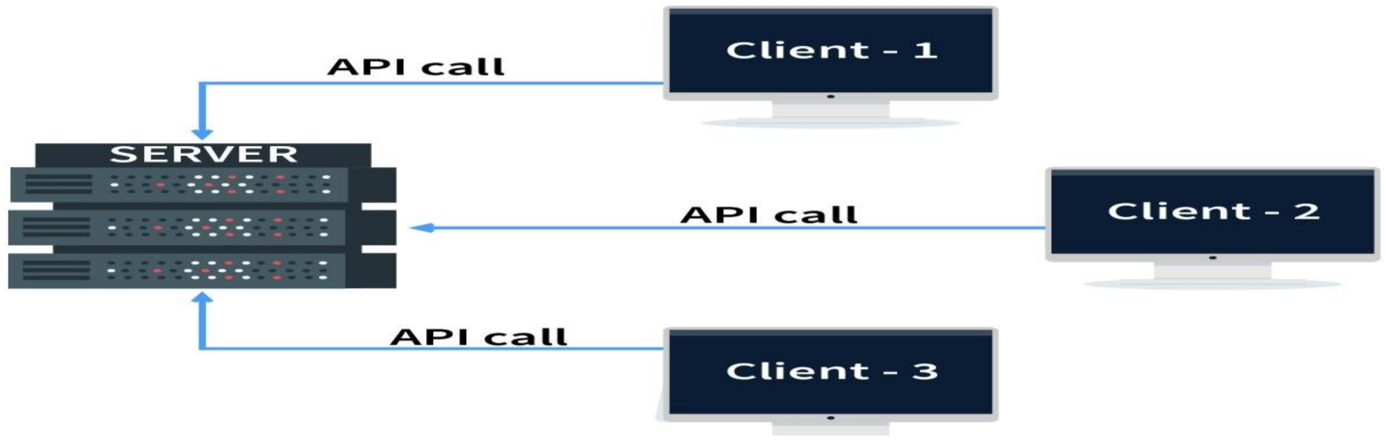
- The data isn't changed frequently.
- No multiple users are accessing the database system.
- We need a direct and simple way to modify or access the database for application development.

Example of Single Tier DBMS Architecture:

In order to learn the Structure Query Language (SQL), we set up our SQL server and the database on our local system. This SQL server enables us to directly interact with the relational database and execute certain operations without requiring any network connection. This whole setup to learn SQL queries is an example of Single-Tier DBMS architecture.

2. Two Tier Architecture:

- The 2-Tier architecture is the same as the basic client-server. In the two-tier architecture, applications on the client end can directly communicate with the database on the server side. For this interaction, APIs like ODBC, and JDBC are used.
- The user interfaces and application programs are run on the client side.
- The server side is responsible to provide the functionalities like query processing and transaction management. To communicate with the DBMS, the client-side application establishes a connection with the server side.



The main advantages of having a two-tier architecture over a single tier are:

- Multiple users can use it at the same time. Hence, it can be used in an organization.
- It has high processing ability as the database functionality is handled by the server alone.
- Faster access to the database due to the direct connection and improved performance.
- Because of the two independent layers, it's easier to maintain.

Example of Two-Tier DBMS Architecture:

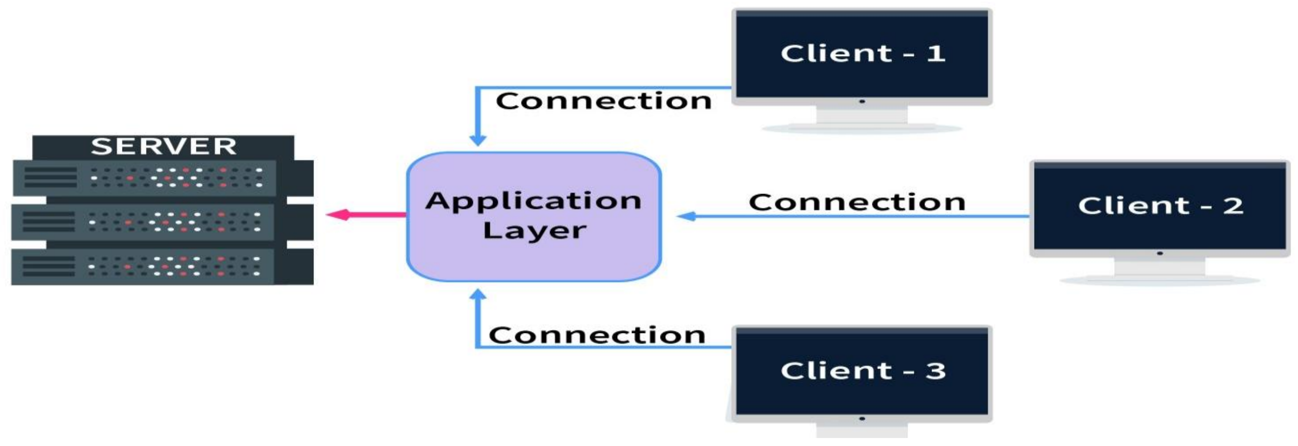
Consider a situation where you went to a bank to withdraw some cash. After entering the withdrawal amount and the account details on the withdrawal slip, the banker will go through the server-side database via his credential (**API call**) and will check whether there is enough balance present or not. This client-server model is an example of Two-Tier DBMS architecture.

Note - The main disadvantages of Two-Tier DBMS Architecture are:

- **Scalability -** As the number of clients increases, the load on the server increases. Thereby declining the performance of the DBMS and, in turn, the client-side application.
- **Security -** The Direct connection between the client and server systems makes this architecture vulnerable to attacks.

3. Three Tier Architecture:

- The 3-Tier architecture contains another layer between the client and server. In this architecture, the client can't directly communicate with the server.
- The application on the client end interacts with an application server which further communicates with the database system.
- The end-user has no idea about the existence of the database beyond the application server. The database also has no idea about any other user beyond the application. The 3-Tier architecture is used in the case of the large web application.



The main advantages of Three Tier DBMS Architecture are:

- **Scalability -** Since the database server isn't aware of any users beyond the application layer and the application layer implements load balancing, there can be as many clients as you want.
- **Data Integrity -** Data corruption and bad requests can be avoided because of the checks performed in the application layer on each client request.
- **Security -** The removal of the direct connection between the client and server systems via abstraction reduces unauthorized access to the database.

Note - In Three Tier DBMS Architecture, an additional layer (Application Layer) is added between the Client and the Server. This increases the number of layers present between the DBMS and the end-users, making the implementation of the DBMS structure complex and difficult to maintain.