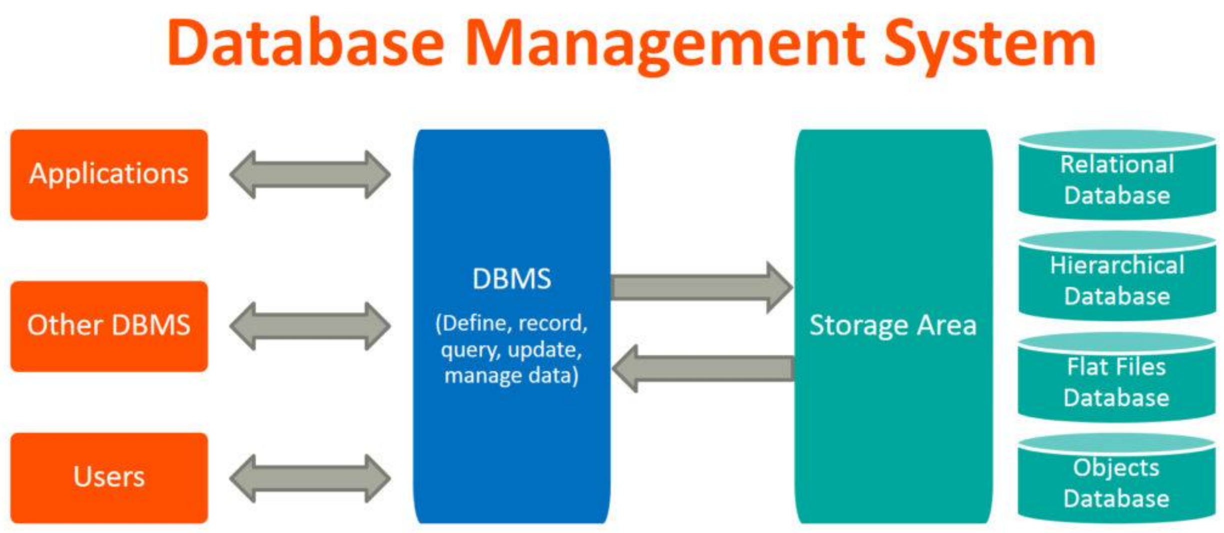
DBMS

A **database management system** (**DBMS**) is the software that interacts with end users, applications, and the database itself to capture and analyze the data. The DBMS software additionally encompasses the core facilities provided to administer the database. The sum total of the database, the DBMS and the associated applications can be referred to as a **database system**. Often the term "database" is also used loosely to refer to any of the DBMS, the database system or an application associated with the database.

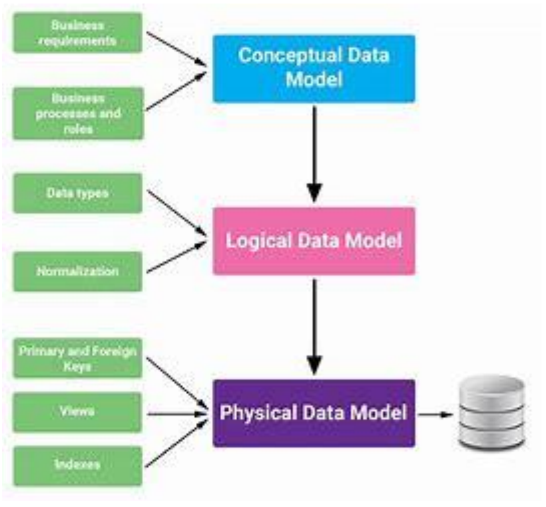


Database System Concepts and Architecture

Database system concepts and architecture are the **fundamental ideas** and **design principles** that underlie the structure and functionality of a **database management system (DBMS)**. A DBMS is a software system that **stores, organizes and manages** a large amount of data within a single software application. A DBMS can have different **architectures**, such as **client/server**, where the database system is divided into two parts: a server that runs the DBMS software and handles the database operations, and a client that connects to the server through a network and requests database services.

Data Model

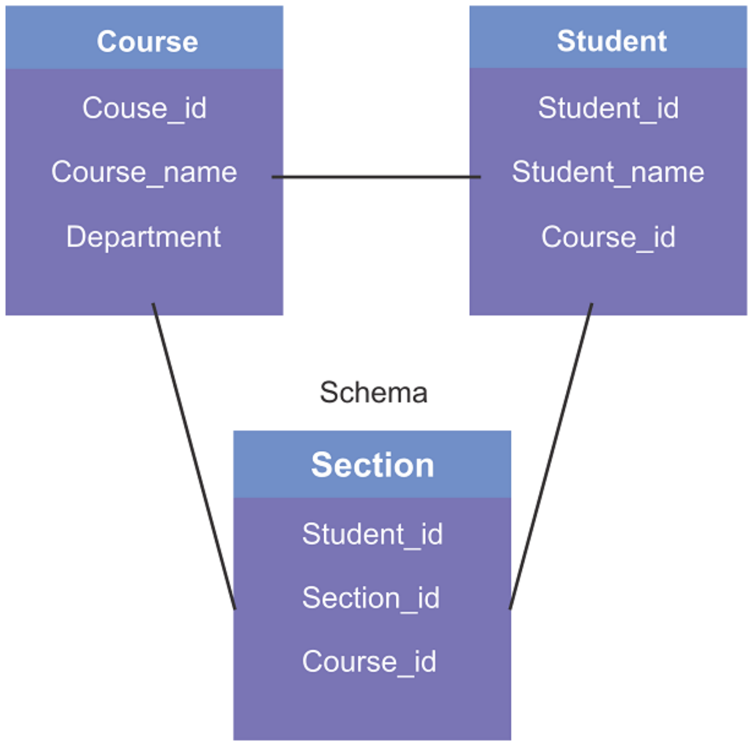
A **data model** is an abstract representation of the structure of a database. It describes the relationships between different data elements and how they are organized. There are several types of data models, including **conceptual**, **logical**, and **physical** models .
A **conceptual data model** is a high-level model that describes the overall structure of the database. It is used to define the entities, attributes, and relationships between them .
A **logical data model** is a more detailed model that describes how data is stored in the database. It includes information about tables, columns, keys, and other database objects .
A **physical data model** is a low-level model that describes how data is stored on disk. It includes information about file formats, storage structures, and access methods .



Schemas and Instances

In database management systems, a **schema** is the overall description of a database, which includes the structure of how data will be stored in the database. A schema is divided into three types: logical schema, physical schema, and view schema. The logical schema describes the database at a logical level, while the physical schema describes the database at a physical level. The view schema defines the design of the database at the view level .

On the other hand, an **instance** is a collection of data stored in a database at a particular moment in time. It is also known as a snapshot of the database. For example, if we have a table named “employee” in our database, then each row or record in that table represents an instance of that table .



Three-Schema Architecture and Data Independence

The **three-schema architecture** is a framework used to describe the structure of a specific database system. It is also called the ANSI/SPARC architecture or three-level architecture . The three-schema architecture is used to separate the user applications and physical database. It contains three levels: external, conceptual, and internal .
The **external schema** describes the part of the database that is relevant to a particular user. It is also called the user schema or view level . The external schema is concerned with how the data appears to the users and how it can be accessed by them .
The **conceptual schema** describes the structure of the whole database for a community of users. It is also called the logical schema or community schema [1]. The conceptual schema describes what data are to be stored in the database and what relationships exist among those data .
The **internal schema** describes the physical storage structure of the database. It is also called the physical schema or storage schema . The internal schema is used to define how data will be stored in a block and is generally concerned with storage space allocations, access paths, data compression and encryption techniques, optimization of internal structures, and representation of stored fields .
Data independence refers to the capacity to change the schema at one level of a database system without having to change the schema at the next higher level . There are two types of data independence: logical data independence and physical data independence .

- **Logical data independence** refers to being able to change the conceptual schema without having to change the external schema. Logical data independence separates the external level from the conceptual view. If we make any changes in the conceptual view of the data, then it would not affect the user view of the data .
- **Physical data independence** refers to being able to change the internal schema without having to change the conceptual schema. Physical data independence separates conceptual levels from internal levels. If we make any changes in the storage size of a database system server, then it would not affect the conceptual structure of the database .

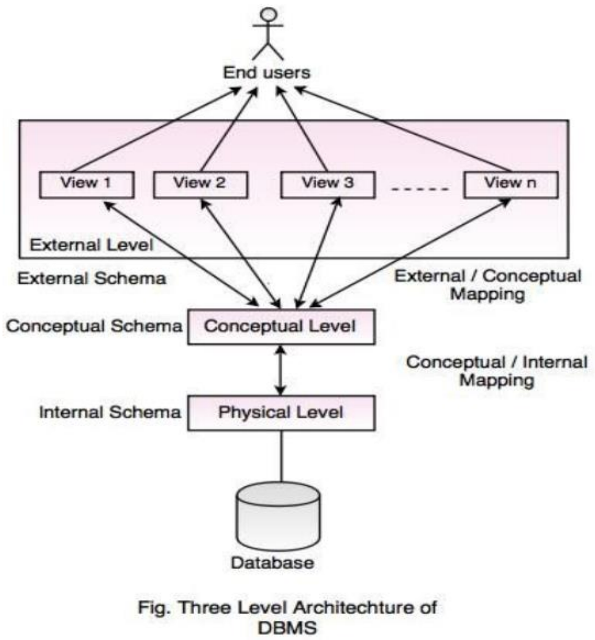
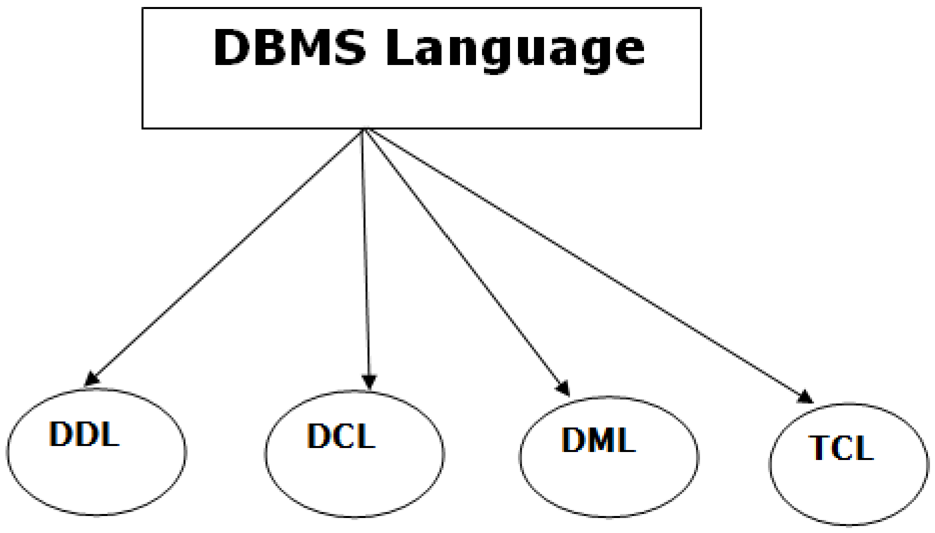


Fig. Three Level Architechture of DBMS

Database Languages in DBMS

- A DBMS has appropriate languages and interfaces to express database queries and updates.
- Database languages can be used to read, store and update the data in the database.

Types of Database Languages



1. **Data Definition Language (DDL)**

- **DDL** stands for **D**ata **D**efinition **L**anguage. It is used to define database structure or pattern.
- It is used to create schema, tables, indexes, constraints, etc. in the database.
- Using the DDL statements, you can create the skeleton of the database.
- Data definition language is used to store the information of metadata like the number of tables and schemas, their names, indexes, columns in each table, constraints, etc.

Here are some tasks that come under DDL:

- **Create:** It is used to create objects in the database.
- **Alter:** It is used to alter the structure of the database.
- **Drop:** It is used to delete objects from the database.
- **Truncate:** It is used to remove all records from a table.

- **Rename:** It is used to rename an object.
- **Comment:** It is used to comment on the data dictionary.

These commands are used to update the database schema that's why they come under Data definition language.

2. **Data Manipulation Language (DML)**

DML stands for **D**ata **M**anipulation **L**anguage. It is used for accessing and manipulating data in a database. It handles user requests.

Here are some tasks that come under DML:

- **Select:** It is used to retrieve data from a database.
- **Insert:** It is used to insert data into a table.
- **Update:** It is used to update existing data within a table.
- **Delete:** It is used to delete all records from a table.
- **Merge:** It performs UPSERT operation, i.e., insert or update operations.
- **Call:** It is used to call a structured query language or a Java subprogram.
- **Explain Plan:** It has the parameter of explaining data.
- **Lock Table:** It controls concurrency.

3. **Data Control Language (DCL)**

- **DCL** stands for **D**ata **C**ontrol **L**anguage. It is used to retrieve the stored or saved data.
- The DCL execution is transactional. It also has rollback parameters.

  (But in Oracle database, the execution of data control language does not have the feature of rolling back.)

Here are some tasks that come under DCL:

- **Grant:** It is used to give user access privileges to a database.
- **Revoke:** It is used to take back permissions from the user.

There are the following operations which have the authorization of Revoke:

CONNECT, INSERT, USAGE, EXECUTE, DELETE, UPDATE and SELECT.

4. **Transaction Control Language (TCL)**

TCL is used to run the changes made by the DML statement. TCL can be grouped into a logical transaction.

Here are some tasks that come under TCL:

- **Commit:** It is used to save the transaction on the database.
- **Rollback:** It is used to restore the database to original since the last Commit.

Centralized and Client/Server Architectures for DBMSs

1. **Centralized DBMSs 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 system functions, including user application programs and user interface programs, as well as all the DBMS functionality. The reason was that most users accessed such systems via computer terminals that did not have processing power and only provided display capabilities. Therefore, all processing was performed remotely on the computer system, and only display information and controls were sent from the computer to the display terminals, which were connected to the central computer via various types of communications networks.

As prices of hardware declined, most users replaced their terminals with PCs and workstations. At first, database systems used these computers similarly to how 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 inter-face processing were carried out on one machine. Figure 2.4 illustrates the physical components in a centralized architecture. Gradually, DBMS systems started to exploit the available processing power at the user side, which led to client/server DBMS architectures.

2. **Basic Client/Server Architectures**

First, we discuss client/server architecture in general, then we see how it is applied to DBMSs. The **client/server architecture** was developed to deal with computing environments in which a large number of PCs, workstations, file servers, printers, database servers,

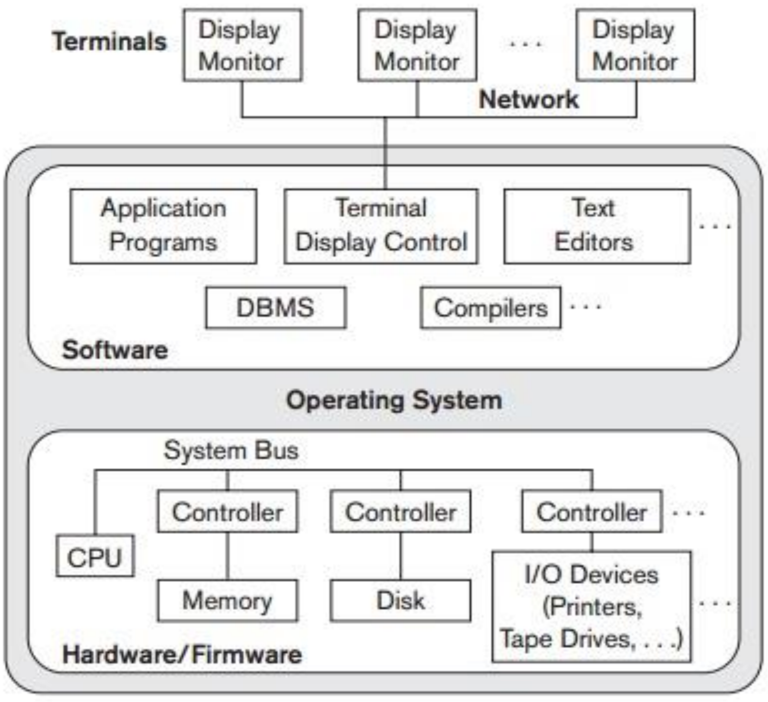


Figure 2.4
A physical centralized architecture.

Web servers, e-mail servers, and other software and equipment are connected via a network. The idea is to define **specialized servers** with specific functionalities. For example, it is possible to connect a number of PCs or small workstations as clients to a **file server** that maintains the files of the client machines. Another machine can be designated as a **printer server** by being connected to various printers; all print requests by the clients are forwarded to this machine. **Web servers** or **e-mail servers** also fall into the specialized server category. 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 other software packages, with specialized programs—such as a CAD (computer-aided design) package—being stored on specific server machines and being made accessible to multiple clients. Figure 2.5 illustrates client/server architecture at the logical level; Figure 2.6 is a simplified diagram that shows the physical architecture. Some machines would be client sites only (for example, diskless work-stations or workstations/PCs with disks that have only client software installed).

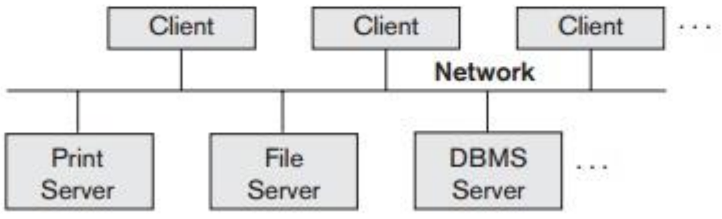


Figure 2.5
Logical two-tier client/server architecture.

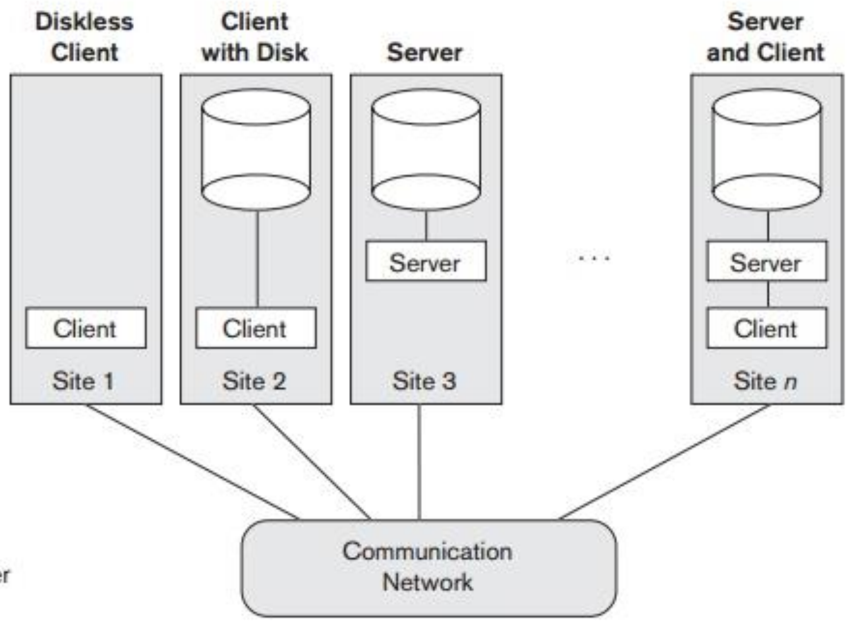


Figure 2.6
Physical two-tier client/server architecture.

Other machines would be dedicated servers, and others would have both client and server functionality.

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 LANs 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 system containing both hard-ware and software that can provide services to the client machines, such as file access, printing, archiving, or database access. In general, some machines install only client software, others only server software, and still others may include both client and server software, as illustrated in Figure 2.6. However, it is more common that client and server software usually run on separate machines. Two main types of basic DBMS architectures were created on this underlying client/server framework: **two-tier** and **three-tier**.