Unit-2

Database System Concepts & Architecture

Data Models, Schemas, and Instances; Three-Schema Architecture and Data Independence; Database Languages and Interfaces; the Database System Environment; Centralized and Client/Server Architectures for DBMSs; Classification of Database Management Systems

[3 hrs]

Data Models:

In a database system, users interact with database through some application. The user action on the application generates some query in the database. The database systems provide the feature of data abstraction. Data abstraction is the technique by which the implementation details about the data are hidden from the end users. Due to data abstraction, the end users can perform desired operation in the database without concerning on how the data actually are represented in the database.

Data model is the collection of concepts that can be used to describe the structure of the database. The structure of the database means the data types, relationships and constraints that apply to the data. The term data model is not limited to include the description about the database structure, it also includes the basic operations that can be performed on the database. In addition to the basic operations, data models also specify the behavior of the database application. By knowing the data model of a DBMS system, the database designer can specify a set of valid user defined operations on the database.

Categories of data models: Different data models describe the database structure in different ways. The common and more popular data models are as following:

1. **Low level or physical data models:** Low-level physical data models provide concepts that describe the details of how data are stored on the computer storage media, typically magnetic disks. Concepts provided by physical data models are generally meant for computer specialists, not for end users.

2. **High level or conceptual data models:** High-level conceptual data models provide concepts for presenting data in ways that are close to the way people perceive data. A typical example is the entity relationship model, which uses main concepts like entities, attributes and relationships. An entity represents a real-world object such as an employee or a project. The entity has attributes that represent properties such as an employee's name, address and birthdate. A relationship represents an association among entities; for example, an employee works on many projects. A relationship exists between the employee and each project.

3. **Representational or implementation data model:** provide concepts that may be easily understood by end users but that are not too far removed from the way data are organized in computer storage. Representational data models hide many details of data storage on disk but can be implemented on a computer system directly. Most frequently used in traditional commercial DBMSs. Used the concept of relational data model as well as the legacy data model (network and hierarchical models). Representational data models represent data by using record structures and hence are sometimes called record-based data models. We can regard the object data model as an example of a new family of higher-level implementation data models that are closer to conceptual data models.

   Another class of data models is known as self-describing data models. The data storage in systems based on these models combines the description of the data with the data values themselves. In traditional data models, data and their description are separately maintained, whereas in self-describing data models, description of the data are also combined with the data itself. The modern data models use this approach such NoSQL system.

Schemas and Instances:

In a data model, it is important to distinguish between the description of the database and the database itself. The description of a database is called the database schema, which is specified during database design and is not expected to change frequently. Most data models have certain conventions for displaying schemas as diagrams. A displayed schema is called a schema diagram. Following example represents the schema diagram for a database. Clearly, a schema diagram simply contains the database structure or design. It doesn't include data added in the database.

STUDENT (Sid, Name, Address, Email, Contact, DepartID)
DEPARTMENT (DepartID, Name, HOD, email, contact, website)
BOOK (Bid, Title, Author, Price)

Fig: Schema diagram

We call each object in the schema, such as STUDENT or DEPARTMENT, a 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, data type of each data item, the relationships among the various files, different types of constraints etc. are some important aspects of schema but cannot be shown in the schema diagram.

The actual data in a database may change quite frequently. The data in the database at a particular moment of time is called a database instance or snapshot. It is also called the current set of occurrences. Each database instance corresponds to a particular schema. Every time we insert or delete a record or change the value of a data item in a record, we change the database instance from one state to another.

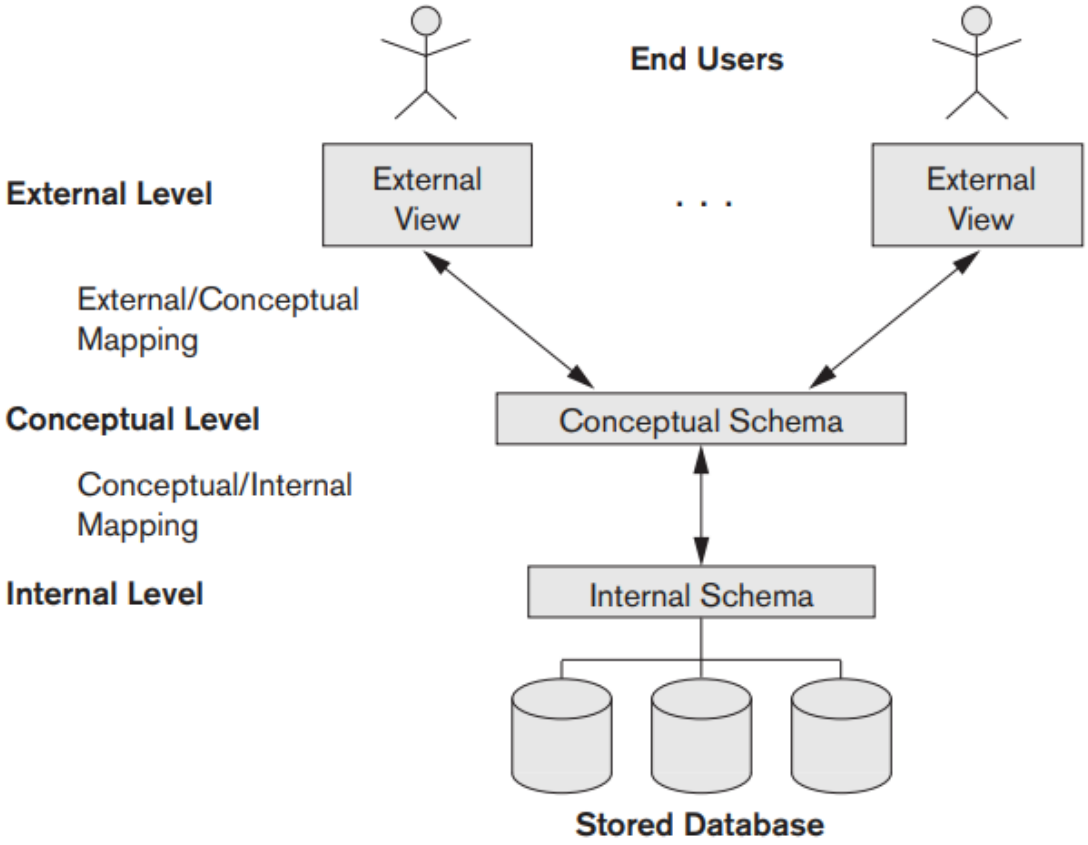
The distinction between database schema and database state is very important. When we define a new database, we specify its database schema only to the DBMS. At this point, the corresponding database state is the empty state with no data. We get the initial state of the database when the database is 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. A database state is a valid state if any database operation has not violated the schema. Hence, specifying a correct schema to the DBMS is extremely important and the schema must be designed with care. 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 needed. The schema is sometimes called the intension, and a database state is called an extension of the schema.

The database instance is changing continuously as any database operation takes place. Unlike the instance, the database schema remains fixed. The schema is allowed to change in a controlled manner only when database has to be changed.

Three schema architecture

The goal of the three-schema architecture, illustrated in Figure 2.2, is to separate the user applications from the physical database. In this architecture, schemas can be defined at the following three levels:



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 level, each external schema is typically implemented using a representational data model, possibly based on an external schema design in a high-level conceptual data model.

The three-schema architecture is a convenient tool with which the user can visualize the schema levels in a database system. Most DBMSs do not separate the three levels completely and explicitly, but they support the three-schema architecture to some extent. Some older DBMSs may include physical-level details in the conceptual schema. The three-level ANSI architecture has an important place in database technology development because it clearly separates the users' external level, the database's conceptual level, and the internal storage level for designing a database. It is very much applicable in the design of DBMSs, even today. In most DBMSs that support user views, external schemas are specified in the same data model that describes the conceptual-level information (for example, a relational DBMS like Oracle or SQLServer uses SQL for this). Notice that the three schemas are only descriptions of data; the actual data is stored at the physical level only. In the three-schema architecture, each user group refers 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. Even in such systems, however, it is necessary to transform requests between the conceptual and internal levels.

Data Independence

The three-schema architecture can be used to further explain the concept of data independence, which 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). In the last case, external schemas that refer only to the remaining data should not be affected. Only the view definition and the mappings need to be changed in a DBMS that supports logical data independence. 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 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.

Generally, physical data independence exists in most databases and file environments where physical details, such as the exact location of data on disk, and hardware details of storage encoding, placement, compression, splitting, merging of records, and so on are hidden from the user. Applications remain unaware of these details. On the other hand, logical data independence is harder to achieve because it allows structural and constraint changes without affecting application programs—a much stricter requirement.

The data independence in DBMS is possible due to the use of catalog and mapping logic between the levels. 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 and the catalog is updated.

Databases Languages:

Once the design of a database is completed and a DBMS is chosen to implement the database, the first step is to specify conceptual and internal schemas for the data-base and any mappings between the two. In many DBMSs where no strict separation of levels is maintained, one language, called the data definition language (DDL), is used by the DBA and by database designers to define both schemas. The DBMS will have a DDL compiler whose function is to process DDL statements in order to identify descriptions of the schema constructs and to store the schema description in the DBMS catalog.

In DBMSs where a clear separation is maintained between the conceptual and internal levels, the DDL is used to specify the conceptual schema only. Another language, the storage definition language (SDL), is used to specify the internal schema. The mappings between the two schemas may be specified in either one of these languages. In most relational DBMSs today, there is no specific language that performs the role of SDL. Instead, the internal schema is specified by a combination of functions, parameters, and specifications related to storage of files.

For a true

three-schema architecture, we would need a third language, the view definition language (VDL), to specify user views and their mappings to the conceptual schema, but in most DBMSs the DDL is used to define both conceptual and external schemas. In relational DBMSs, SQL is used in the role of VDL to define user or application views as results of predefined queries.

Once the database schemas are compiled and the database is populated with data, users must have some means to manipulate the database. Typical manipulations include retrieval, insertion, deletion, and modification of the data. The DBMS provides a set of operations or a language called the data manipulation language (DML) for these purposes.

In current DBMSs, the preceding types of languages are usually not considered distinct languages; rather, a comprehensive integrated language is used that includes constructs for

conceptual schema definition, view definition, and data manipulation. Storage definition is typically kept separate, since it is used for defining physical storage structures to fine-tune the performance of the database system, which is usually done by the DBA staff. A typical example of a comprehensive database language is the SQL relational database language which represents a combination of DDL, VDL, and DML, as well as statements for constraint specification, schema evolution, and many other features. The SDL was a component in early versions of SQL but has been removed from the language to keep it at the conceptual and external levels only.

There are two main types of DMLs. They are high-level (non-procedural or declarative) and low-level or procedural. In declarative DML, the DML commands can retrieve the desired data without specifying how to retrieve it. Whereas in procedural DML, programmer has to mention procedural steps to retrieve desired data from the database. Most of the DBMS today support high-level DML. Whenever DML commands, whether high level or low level, are embedded in a general-purpose programming language, that language is called the host language and the DML is called the data sublanguage. On the other hand, a high-level DML used in a standalone interactive manner is called a query language.

Casual end users typically use a high-level query language to specify their requests, whereas programmers use the DML in its embedded form. For naive and parametric users, there usually are user-friendly interfaces for interacting with the database; these can also be used by casual users or others who do not want to learn the details of a high-level query language.

At present, there are mainly four types of database languages:

1. Data Definition Language (DDL)
2. Data Manipulation Language (DML)
3. Data Control Language (DCL)
4. Transaction Control Language (TCL)

1. **Data Definition Language (DDL):** DDL is used to define database structure or schema. It is used to create tables and define constraints. The database skeleton can be created by using DDL. Following are the major tasks that are done by using DDL:
    - Create: To create the objects in database.
    - Alter: To change the schema.
    - Drop: To delete the objects from the database.
    - Truncate: To empty a database object/table.
    - Rename: To rename a database object.

2. **Data Manipulation Language (DML):** DML is used to access and manipulate data in a database. It is used over a pre-built database schema. DML compiler of DBMS converts the DML commands to equivalent low level statements. Following are the major tasks that are done by using DML:
    - Select: To retrieve data from table.
    - Insert: To insert data into table.
    - Update: To change data in a table.
    - Delete: To delete records from a table.

3. **Data Control Language (DCL):** DCL is used to control access to data stored in database. DCL commands are operated by DBA to define the access rights to different users in the database. Following are the major tasks that are done by using DML:
    - Grant: To give user access privileges to a database.
    - Revoke: To take back user access privileges to a database.

4. **Transaction Control Language (TCL):** TCL is used to manage transactions in the database. TCL commands are normally not used by the DBMS users, rather they are typically used by DBMS designers and developers. These commands are used to enhance database performance, security, concurrency and consistency. TCL commands are used in a database transaction. Some major tasks that are done by using TCL are:
    - Commit: To save the changes in data made by a transaction into physical disk.
    - Rollback: To revert the database to previous state from the last commit.

Database Interfaces:

User-friendly interfaces provided by a DBMS may include the following:

1. **Menu-based Interfaces for Web Clients or Browsing:**
   These interfaces present the user with lists of options (called menus) that lead the user through the formulation of a request. Menus do away with the need to memorize the specific commands and syntax of a query language; rather, the query is composed step-by step by picking options from a menu that is displayed by the system. Pull-down menus are a very popular technique in Web-based user interfaces. They are also often used in browsing interfaces, which allow a user to look through the contents of a database in an exploratory and unstructured manner.

2. **Apps for Mobile Devices:**
   These interfaces present mobile users with access to their data. For example, banking, reservations, and insurance companies, among many others, provide apps that allow users to access their data through a mobile phone or mobile device. The apps have built-in programmed interfaces that typically allow users to login using their account name and password; the apps then provide a limited menu of options for mobile access to the user data, as well as options such as paying bills (for banks) or making reservations (for reservation Web sites).

3. **Forms-based Interfaces**
   A forms-based interface displays a form to each user. Users can fill out all of the form entries to insert new data, or they can fill out only certain entries, in which case the DBMS will retrieve matching data for the remaining entries. Forms are usually designed and programmed for naive users as interfaces to canned transactions. Many DBMSs have forms specification languages, which are special languages that help programmers specify such forms. SQL*Forms is a form-based language that specifies queries using a form designed in conjunction with the relational database schema. Oracle Forms is a component of the Oracle product suite that provides an extensive set of features to design and build applications using forms. Some systems have utilities that define a form by letting the end user interactively construct a sample form on the screen.

4. **Graphical User Interfaces:**
   A GUI typically displays a schema to the user in diagrammatic form. The user then can specify a query by manipulating the diagram. In many cases, GUIs utilize both menus and forms.

5. **Natural Language Interfaces:**
   These interfaces accept requests written in English or some other language and attempt to understand them. A natural language interface usually has its own schema, which is similar to the database conceptual schema, as well as a dictionary of important words. The natural language interface refers to the words in its schema, as well as to the set of standard words in its dictionary, that are used to interpret the request. If the interpretation is successful, the interface generates a high-level query corresponding to the natural language request and submits it to the DBMS for processing; otherwise, a dialogue is started with the user to clarify the request.

6. **Keyword-based Database Search:**
   These are somewhat similar to Web search engines, which accept strings of natural language (like English or Spanish) words and match them with documents at specific sites (for local search engines) or Web pages on the Web at large (for engines like Google or Ask). They use predefined indexes on words and use ranking functions to retrieve and present resulting documents in a decreasing degree of match. Such “free form” textual query interfaces are not yet common in structured relational databases, although a research area called keyword-based querying has emerged recently for relational databases.

7. **Speech Input and Output:**
   Limited use of speech as an input query and speech as an answer to a question or result of a request is becoming commonplace. Applications with limited vocabularies, such as inquiries for telephone directory, flight arrival/departure, and credit card account information, are allowing speech for input and output to enable customers to access this information. The speech input is detected using a library of predefined words and used to set up the parameters that are supplied to the queries. For output, a similar conversion from text or numbers into speech takes place.

8. **Interfaces for Parametric Users:**
   Parametric users, such as bank tellers, often have a small set of operations that they must perform repeatedly. For example, a teller is able to use single function keys to invoke routine and repetitive transactions such as account deposits or withdrawals, or balance inquiries. Systems analysts and programmers design and implement a special interface for each known class of naive users. Usually a small set of abbreviated commands is included, with the goal of minimizing the number of keystrokes required for each request.

9. **Interfaces for the DBA:**
   Most database systems contain privileged commands that can be used only by the DBA staff. These include commands for creating accounts, setting system parameters,

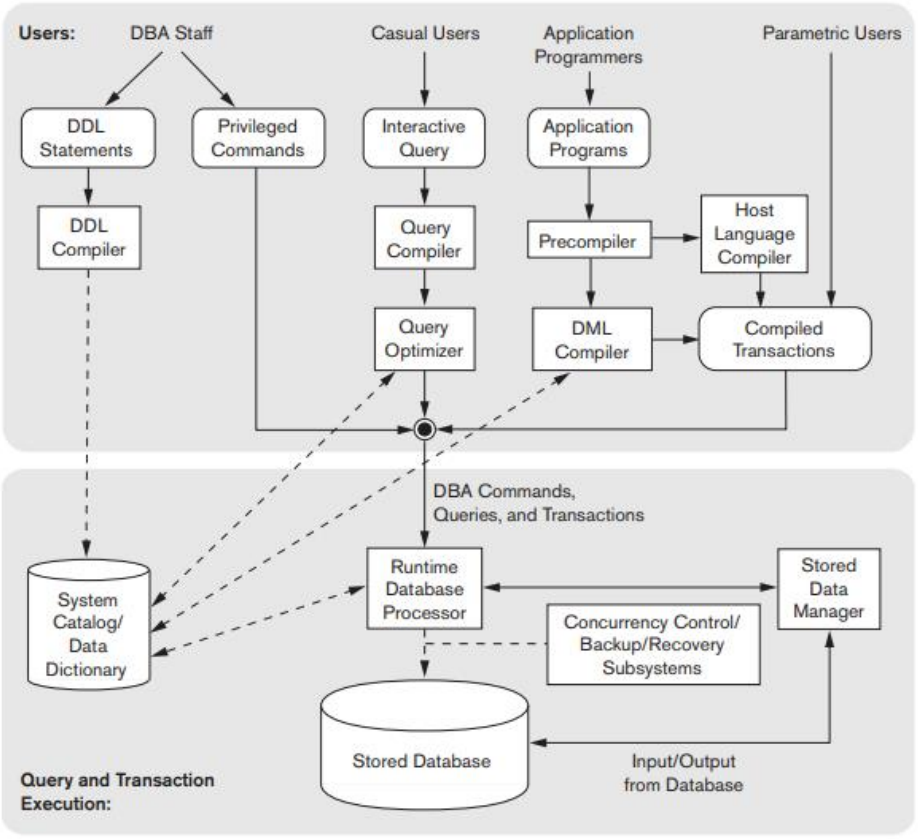
granting account authorization, changing a schema, and reorganizing the storage structures of a database.

The Database System Environment:

DBMS is a complex software system. It consists different modules, utilities and tools for various purpose. The typical components of database system environment are:

- DBMS Component Modules
- Database System Utilities
- Tools, Application Environment and Communication Facilities

a. DBMS Component Modules: Following figure illustrates, in a simplified form, the typical DBMS components. The figure is divided into two parts. The top part of the figure refers to the various users of the database environment and their interfaces. The lower part shows the internal modules of the DBMS responsible for storage of data and processing of transactions.



The database and the DBMS catalog are usually stored on disk. Access to the disk is controlled primarily by the operating system (OS), which schedules disk read/write. Many DBMSs have their own buffer management module to schedule disk read/write, because management of buffer storage has a considerable effect on performance. Reducing disk read/write improves performance considerably. A higher-level stored data manager module of the DBMS controls access to DBMS information that is stored on disk, whether it is part of the database or the catalog.

Let us consider the top part of the above figure first. It shows interfaces for the DBA staff, casual users who work with interactive interfaces to formulate queries, application programmers who create programs using some host programming languages, and parametric users who do data entry work by supplying parameters to predefined transactions. The DBA staff works on defining the database and tuning it by making changes to its definition using the DDL and other privileged commands.

The DDL compiler processes schema definitions, specified in the DDL, and stores descriptions of the schemas (meta-data) in the DBMS catalog. The catalog includes information such as the names and sizes of files, names and data types of data items, storage details of each file, mapping information among schemas, and constraints. In addition, the catalog stores many other types of information that are needed by the DBMS modules, which can then look up the catalog information as needed.

Casual users and persons with occasional need for information from the database interact using the interactive query interface. These queries are parsed and validated for correctness of the query syntax, the names of files and data elements, and so on by a query compiler that compiles them into an internal form. This internal query is subjected to query optimization, among other things, the query optimizer is concerned with the rearrangement and possible reordering of operations, elimination of redundancies, and use of efficient search algorithms during execution. It consults the system catalog for statistical and other physical information about the stored data and generates executable code.

Application programmers write programs in host languages such as PHP, Java, C, or C++ that are submitted to a pre-compiler. The pre-compiler extracts DML commands from an application program written in a host programming language. These commands are sent to the DML compiler for compilation for database access. The rest of the program is sent to the host language compiler.

The compiled DML commands and the host language programs are stored as compiled transactions to be used by parametric users. The parametric users simply supply the parameters to the compiled transactions through the application programs.

In the lower part of the diagram, the runtime database processor processes the different commands, queries and transactions coming from different types of users. It manages the system catalog as well as performs desired database operation with the help of another module called stored data manager. The basic operations in the physical database is performed by the stored data manager module.

The runtime database processor module performs all its operations in coordination with another module called concurrency control/backup/recovery subsystem.

b. Database System Utilities: DBMSs also provide the database utilities that help the DBA manage the database system. Common utilities have the following types of functions:

- **Data Loading:**
  A loading utility is used to load existing data files—such as text files or sequential files—into the database. Usually, the current (source) format of the data file and the desired (target) database file structure are specified to the utility, which then automatically reformats the data and stores it in the database. With the proliferation of DBMSs, transferring data from one DBMS to another is becoming common in many

organizations. Some vendors offer conversion tools that generate the appropriate loading programs, given the existing source and target database storage descriptions (internal schemas).

- **Backup:**
  A backup utility creates a backup copy of the database, usually by dumping the entire database onto tape or other mass storage medium. The backup copy can be used to restore the database in case of catastrophic disk failure. Incremental backups are also often used, where only changes since the previous backup are recorded. Incremental backup is more complex, but saves storage space.
- **Database storage organization:**
  This utility can be used to reorganize a set of database files into different file organizations and create new access paths to improve performance.
- **Performance monitoring:**
  Such a utility monitors database usage and provides statistics to the DBA. The DBA uses the statistics in making decisions such as whether or not to reorganize files or whether to add or drop indexes to improve performance.

Other utilities may be available for sorting files, handling data compression, monitoring access by users, interfacing with the network, and performing other functions.

c. Tools, Application Environment and Communication Facilities: Some advanced and modern DBMSs also provide various tools, environment and facilities. Such tools are often available to database designers, users, and the DBMS.

CASE tools (including design tools, project management tools, analysis tools, documentation tools, version control tools, maintenance tools etc) are used in the design phase of database systems.

Another tool that can be quite useful in large organizations is an expanded **data dictionary** (or data repository) system. In addition to storing catalog information about schemas and constraints, the data dictionary stores other information, such as design decisions, usage standards, application program descriptions, and user information. Such a system is also called an information repository. This information can be accessed directly by users or the DBA when needed. A data dictionary utility is similar to the DBMS catalog, but it includes a wider variety of information and is accessed mainly by users rather than by the DBMS software.

Application development environments, such as PowerBuilder (Sybase) or JBuilder (Borland), have been quite popular. These systems provide an environment for developing database applications and include facilities that help in many facets of database systems, including database design, GUI development, querying and updating, and application program development.

The DBMS also needs to interface with **communications software**, whose function is to allow users at locations remote from the database system site to access the database through computer terminals, workstations, or personal computers. These are connected to the database site through data communications hardware such as Internet routers, phone lines, long-haul networks, local networks, or satellite communication devices. Many commercial database systems have communication

packages that work with the DBMS. The integrated DBMS and data communications system is called a DB/DC system. In addition, some distributed DBMSs are physically distributed over multiple machines. In this case, communications networks are needed to connect the machines. These are often local area networks (LANs), but they can also be other types of networks.

Centralized and Client/Server Architectures for DBMS:

a. **Centralized DBMS Architecture:** Architectures for DBMSs have followed trends similar to those for general computer system architectures. Older 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 in older systems, most users accessed the DBMS via computer terminals that did not have processing power and only provided display capabilities. Therefore, all processing was performed remotely on the computer system housing the DBMS, 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.

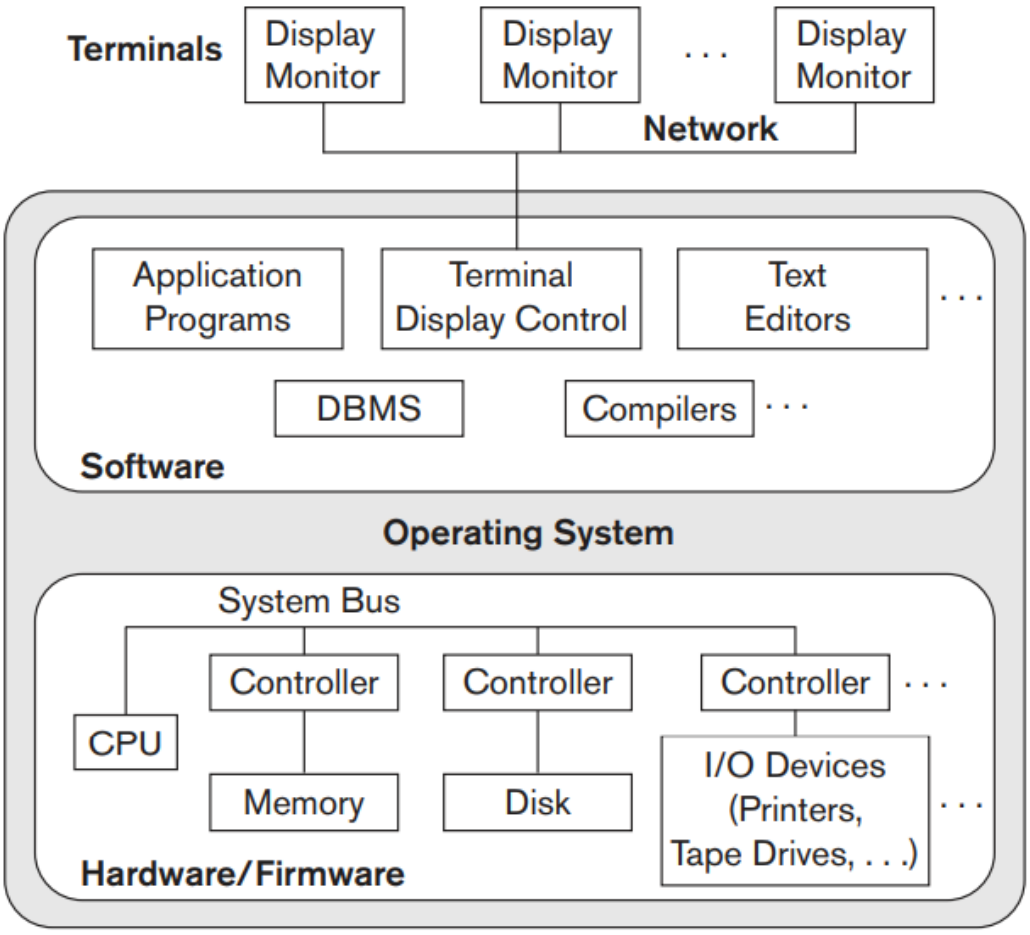


Fig: Centralized Architecture

As prices of hardware declined, most users replaced their terminals with PCs and workstations, and more recently with mobile devices. 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 interface processing were carried out on one machine. Above diagram 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.

b. 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, 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. Following figure illustrates client/server architecture at the logical level; another diagram shown below is a simplified diagram that shows the physical architecture. Some machines would be client sites only (for example, mobile devices or workstations/PCs that have only client software installed). Other machines would be dedicated servers, and others would have both client and server functionality.

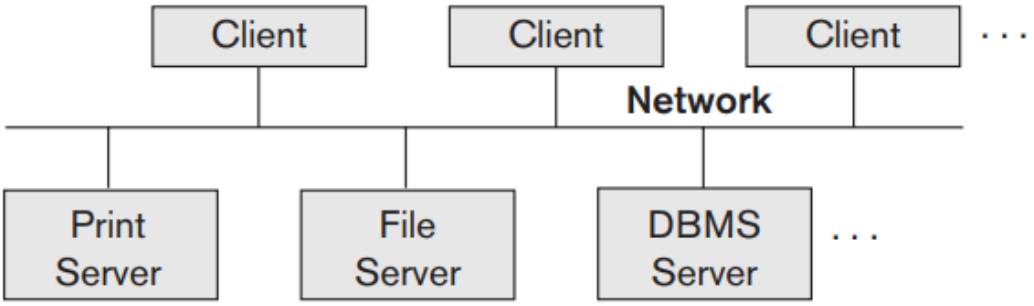


Fig: Logical two-tier client server architecture

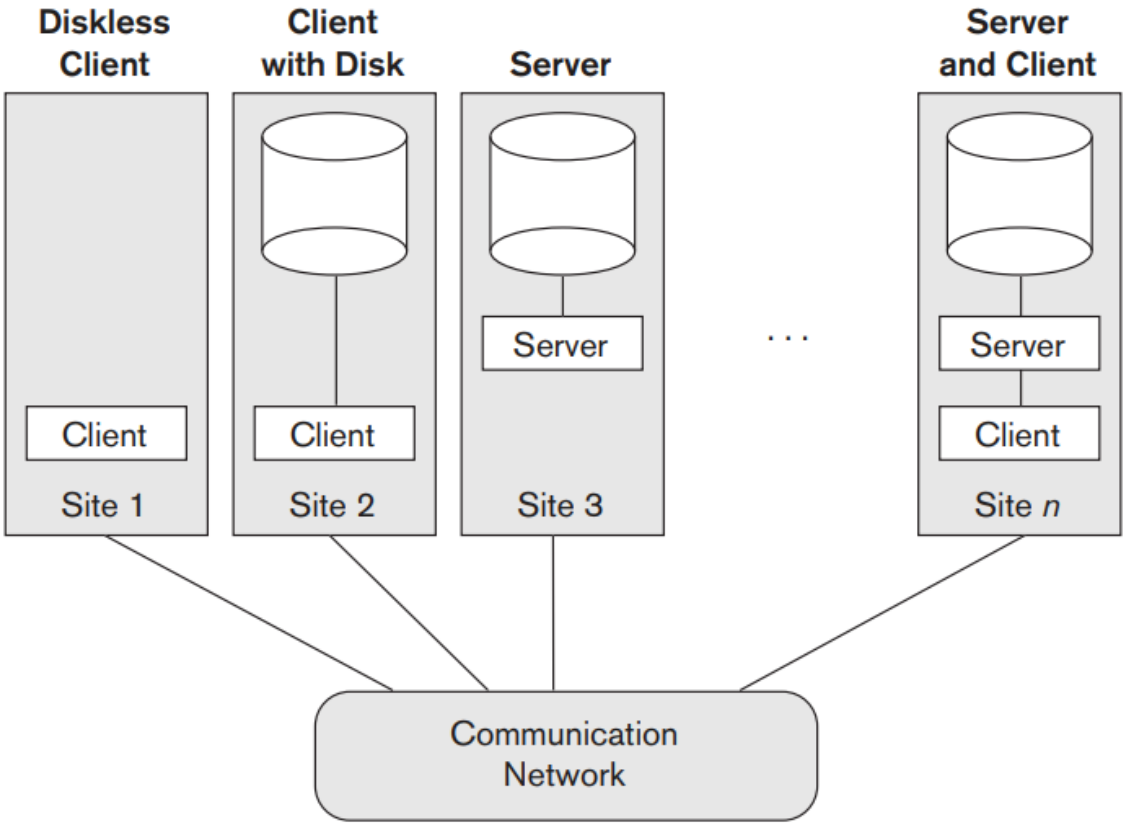


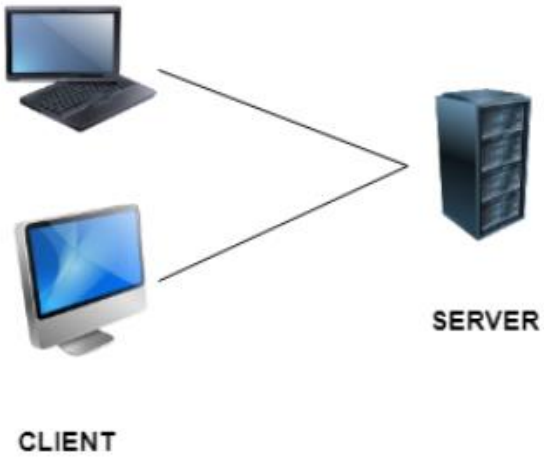
Fig: Physical two-tier client server architecture

The concept of client/server architecture assumes an underlying framework that consists of many PCs/workstations and mobile devices as well as a smaller number of server machines, connected via wireless networks or 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 the client, it connects to a server that provides the needed functionality. A server is a system containing both hardware 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 second figure above. 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.

i. **Two-Tier Client/Server Architecture for DBMS:** In relational database management systems (RDBMSs), many of which started as centralized systems, the system components that were first moved to the client side were the user interface and application programs. Because SQL provided a standard language for RDBMSs, this created a logical dividing point between client and server. Hence, the query and transaction functionality related to SQL processing remained on the server side. In such an architecture, the server is often called a query server or transaction server because it provides these two functionalities. In an RDBMS, the server is also often called an SQL server. 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. 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 and display the results as needed.

The architectures described here are called two-tier architectures because the software components are distributed over two systems: client and server.



The major advantages of the two-tier client/server structure are:

- This structure is quite easy to maintain and modify.
- The communication between the client and server in the form of request response messages is quite fast.

A major disadvantage of the two-tier client/server structure is:

- If the client nodes are increased beyond capacity in the structure, then the server is not able to handle the request overflow and performance of the system degrades.
- It is comparatively less secure.

ii. **Three-Tier and n-Tier Architectures for Web Applications:** 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.

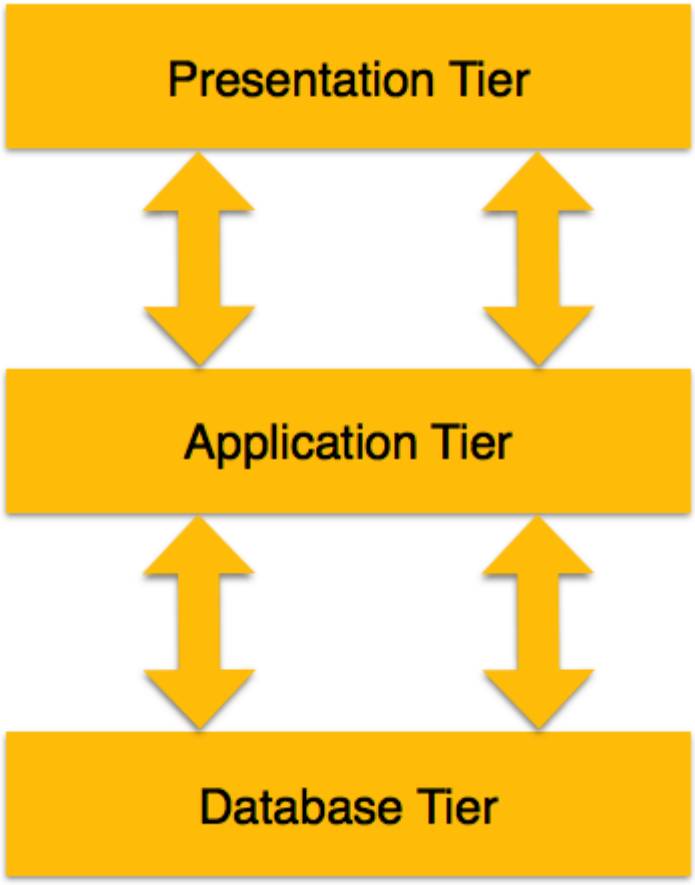


Fig: Three-tier Client/Server Architecture

- **User (Presentation) 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.
- **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.
- **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.

Some of the advantages of the three-tier client/server structure are:

- The three-tier structure provides much better service and is more reliable.
- The structure can be scaled according to requirements without any problem.
- Data security is much improved in the three-tier structure.

A major disadvantage of the three-tier client/server structure are:

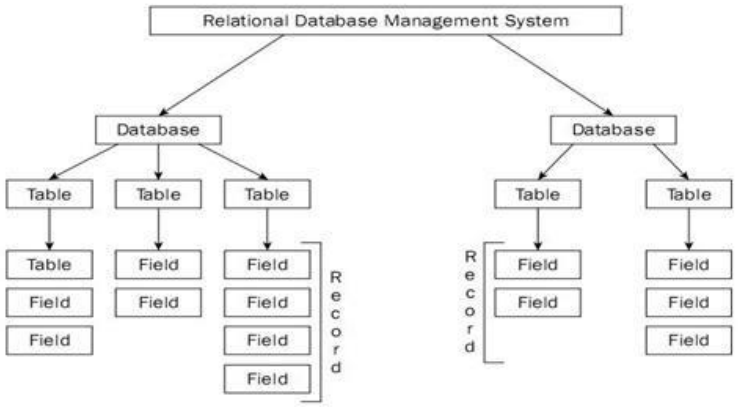
- Three - tier client/server structure is quite complex due to advanced features.
- It is relatively slower in comparison to the two-tier architecture.

It is possible to divide the layers between the user and the stored data further into finer components, thereby giving rise to n-tier architectures, where n may be four or five tiers. Typically, the business logic layer is divided into multiple layers.

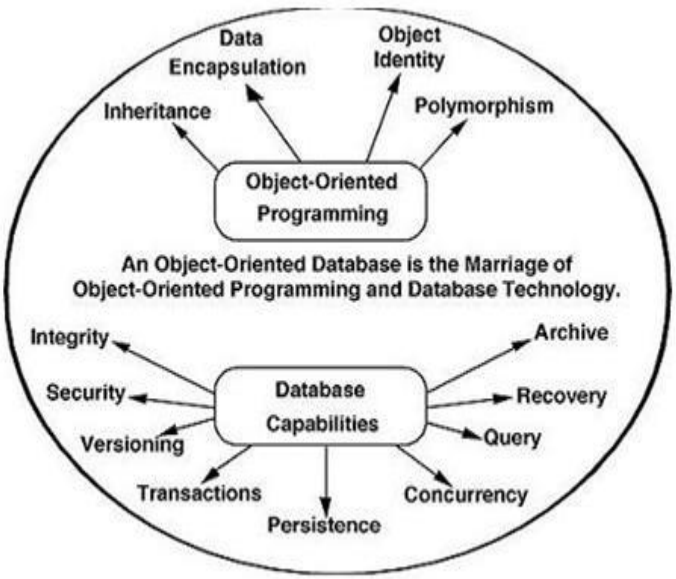
Classification of Database Management Systems:

1. Based on the data model:

a. Relational database – This is the most popular data model used in industries. It is based on the SQL. They are table oriented which means data is stored in different access control tables, each has the key field whose task is to identify each row. The tables or the files with the data are called as relations that help in designating the row or record, and columns are referred to attributes or fields. Few examples are MYSQL(Oracle, open source), Oracle database (Oracle), Microsoft SQL server(Microsoft) and DB2(IBM).

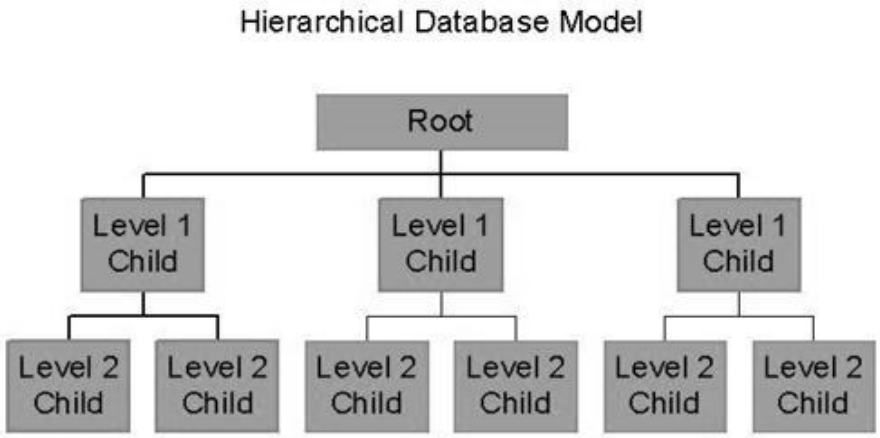


b. Object oriented database – The information here is in the form of the object as used in object-oriented programming. It adds the database functionality to object programming languages. It requires less code, use more natural data and also code bases are easy to maintain. Examples are ObjectDB (ObjectDB software).

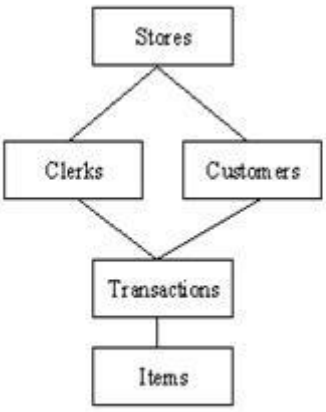


c. **Object relational database** – Relational DBMS are evolving continuously and they have been incorporating many concepts developed in object database leading to a new class called extended relational database or object relational database.

d. **Hierarchical database** – In this, the information about the groups of parent or child relationships is present in the records which is similar to the structure of a tree. Here the data follows a series of records, set of values attached to it. They are used in industry on mainframe platforms. Examples are IMS(IBM), Windows registry (Microsoft).



e. **Network database** – Mainly used on a large digital computer. If there are more connections, then this database is efficient. They are extension to hierarchical database, they look like a cobweb or interconnected network of records. Examples are CA-IDMS (COMPUTER associates), IMAGE(HP). In this model, it is not always required to access top to bottom fashion. Some child can be accessed without going through root. In fact, there is no any specific node called as root. A child can have multiple parent nodes. Hierarchical data model stores data in tree structure, whereas, network data model stores data in the form of graph.



2. Based on the number of users:

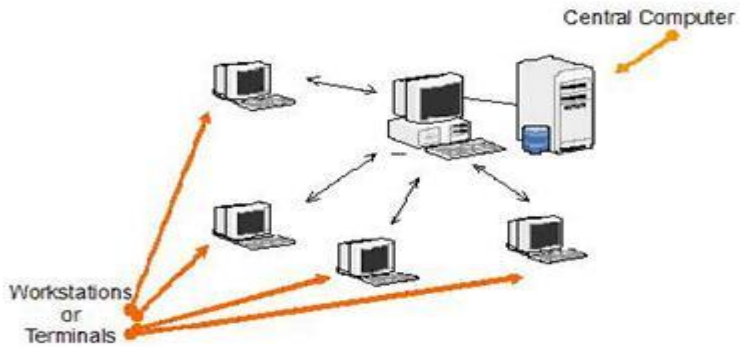
a. **Single user** – As the name itself indicates it can support only one user at a time. It is mostly used with the personal computer on which the data resides accessible to a single person. The user may design, maintain and write the database programs.

b. **Multiple users** – It supports multiple users concurrently. Data can be both integrated and shared, a database should be integrated when the same information is not need

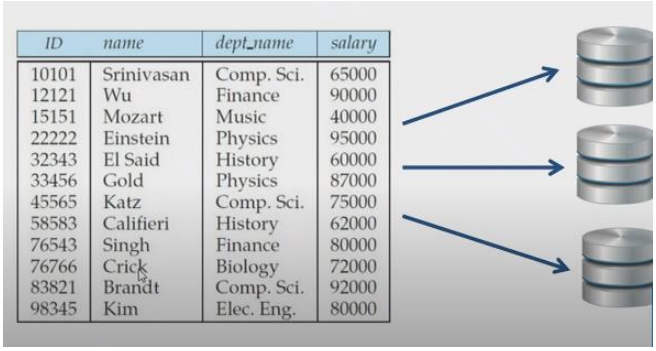
be recorded in two places. For example, a student in the college should have the database containing his information. It must be accessible to all the departments related to him. For example, the library department and the fee section department should have information about student's database. So, in such case, we can integrate and even though database resides in only one place both the departments will have the access to it.

3. Based on the sites over which DBMS is distributed:

a. **Centralized database system** – The DBMS and database are stored at the single site that is used by several other systems too. We can simply say that data here is maintained on the centralized server.

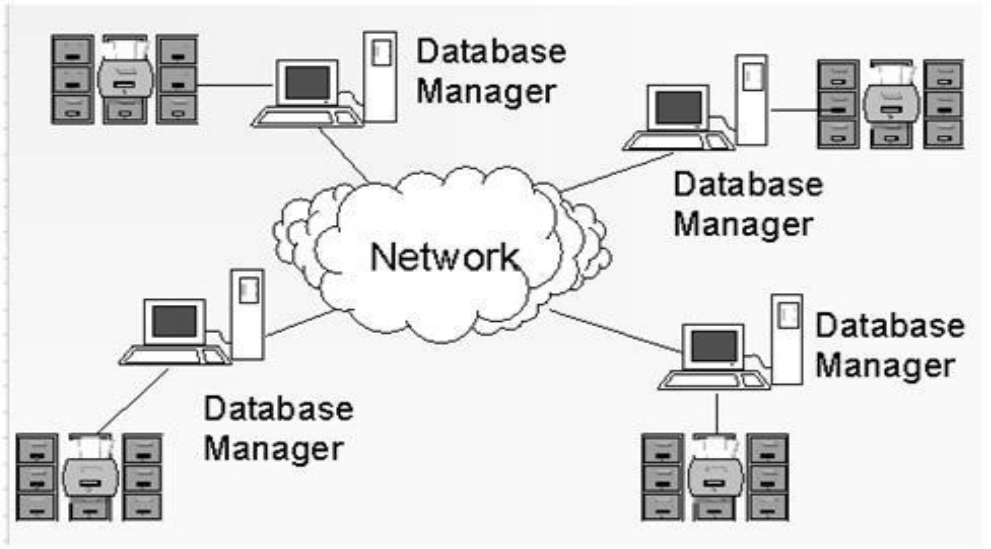


b. **Parallel network database system** – This system has the advantage of improving processing input and output speeds. Majorly used in the applications that have query to larger database. It holds the multiple central processing units and data storage disks in parallel. Database can be partitioned at different disks and processed by assigned processors. There will be common DBMS to manage the entire database partitioned over multiple disks and processors.



ID	name	dept_name	salary
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

c. **Distributed database system** – In this data and the DBMS software are distributed over several sites but connected to the single computer. In this system, data is partitioned in different disks and each disk is accessed by assigned processor just like parallel database system. The main difference here with parallel database system is that; each disk is managed by separate DBMS which in turn connected to central DBMS system.



Distributed DBMS can be of two types:

- **Homogeneous DBMS** – They use same software but from the multiple sites. Data exchange between the sites can be handled easily. For example, library information systems by the same vendor, such as Geac Computer corporation, use the same DBMS software that allows the exchanges between various Geac library sites.

- **Heterogeneous DBMS** – They use different DBMS software for different sites but there is an additional software that helps the exchange of the data between the sites.