Topics to be covered:

- **Characteristics of Database**
- **Database Systems - Advantages and Disadvantages**
- **DBMS users**
- **Three Tier Architecture, Data Abstraction & Data Independence**
- **DBMS Architecture**

Learning Outcome:

This Lecture will provide students with:

- Overview of the role of database administration and best practices and procedures associated with that role.
- Understand the role of all database users and will have clear picture about available jobs in the market associated to each role
- Components of a database system structure and how they work together.
- A broad insight into the architecture of a database management system so that they can understand and appreciate how such a complex piece of software works

---

Characteristics of Database

A database is a collection of related data. A database has the following implicit properties:A database represents some aspect of the real world, sometimes called the mini world or the universe of discourse (UoD). Changes to the mini world are reflected in the database.A database is designed, built, and populated with data for a specific purpose. It has an intended group of users and some preconceived applications in which these users are interested.

A database-management system(DBMS) is a collection of interrelated data and a set of programs to access those data. The collection of data,usually referred to as the database, contains information relevant to an enterprise. The primary goal of a DBMS is to provide a way to store and retrieve database information that is both convenient and efficient.

A database approach is a well-organized collection of data that are related in a meaningful way which can be accessed by different users but stored only once in a system. The various operations performed by the DBMS system are: Insertion, deletion, selection, sorting etc.

Characteristics of database systems:

1. Self-Describing Nature of a Database System:

A fundamental feature of the database approach is that the database systems does not only contain the data but also the complete definition and description of these data. These descriptions are basically details about the extent, the structure, the type and the format of all data and, additionally, the relationship between the data. This kind of stored data is called metadata ("data about data") and it is stored in the DBMS catalog. The catalog is used by the DBMS software and also by database users who need information about the database structure.

2. Insulation between Programs and Data, and Data Abstraction

As described in the feature structured data the structure of a database is described through metadata which is also stored in the database. An application software does not need any knowledge about the physical data storage like encoding, format, storage place, etc. It only communicates with the management system f a database (DBMS) via a standardized interface with the help of a standardized language like SQL. The access to the data and the metadata is entirely done by the DBMS. In this way all the applications can be totally seperated from the data. Therefore database internal reorganizations or improvement of efficiency do not have any influence on the application software.

For example, a file access program may be written in such a way that it can access
only STUDENT records of the structure. If we want to add another piece of data to each STUDENT record, say the Birth_date , such a program will no longer work and must be changed. By contrast, in a DBMS environment, we only need to change the description of STUDENT records in the catalog to reflect the inclusion of the new data item Birth_date ; no programs are changed. The next time a DBMS program refers to the catalog, the new structure of STUDENT
records will be accessed and used.

3. Support of Multiple Views of the Data

A database typically has many users, each of whom may require a different perspective or view of the database. A view may be a subset of the database or it may contain virtual data that is derived from the database files but is not explicitly stored. Some users may not need to be aware of whether the data they refer to is stored or derived.

A multiuser DBMS whose users have a variety of distinct applications must provide facilities for defining multiple views. For example, one user of the database may be interested only in accessing and printing the transcript of each student; a second user, who is interested only in checking that students have taken all the prerequisites of each course for which they register.

4. Sharing of Data and Multiuser Transaction Processing

A database system allows several users to access the database concurrently. Answering different questions from different users with the same (base) data is a central aspect of an information system. Such concurrent use of data increases the economy of a system. An example for concurrent use is the travel database of a bigger travel agency. The employees of different branches can access the database concurrently and book journeys for their clients. Each travel agent sees on his interface if there are still seats available for a specific journey or if it is already fully booked.

A transaction is a bundle of actions which are done within a database to bring it from one consistent state to a new consistent state. In between the data are inevitable inconsistent. A transaction is atomic what means that it cannot be divided up any further. Within a transaction all or none of the actions need to be carried out. Doing only a part of the actions would lead to an inconsistent database state. One example of a transaction is the transfer of an amount of money from one bank account to another. The debit of the money from one account and the credit of it to another account makes together a consistent transaction. This transaction is also atomic. The debit or credit alone would both lead to an inconsistent state. After finishing the transaction (debit and credit) the changes to both accounts become persistent and the one who gave the money has now less money on his account while the receiver has now a higher balance.

Advantages and Disadvantages of a DBMS

Using a DBMS to manage data has many advantages:

Data independence: Application programs should be as independent as possible from details of data representation and storage. The DBMS can provide an abstract view of the data to insulate application code from 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 on the data. For example, before inserting salary information for an employee, the DBMS can check that the department budget is not exceeded. Also, the DBMS can enforce access controls that govern what data is visible to different classes of users.

Data administration: When several users share the data, centralizing the administration of data can offer significant improvements. 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 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.

Reduced application development time: Clearly, the DBMS supports many important functions that are common to many applications accessing data stored in the DBMS. This, in conjunction with the high-level interface to the data, facilitates quick development of applications. Such applications are also likely to be more robust than applications developed from scratch because many important tasks are handled by the DBMS instead of being implemented by the application.

Disadvantages of a DBMS

Danger of a Overkill: For small and simple applications for single users a database system is often not advisable.

Complexity: A database system creates additional complexity and requirements. The supply and operation of a database management system with several users and databases is quite costly and demanding.

Qualified Personnel: The professional operation of a database system requires appropriately trained staff. Without a qualified database administrator nothing will work for long.

Costs: Through the use of a database system new costs are generated for the system itself but also for additional hardware and the more complex handling of the system.

Lower Efficiency: A database system is a multi-use software which is often less efficient than specialized software which is produced and optimized exactly for one problem.

Database Users and Administrators

A primary goal of a database system is to retrieve information from and store new information in the database. People who work with a database can be categorized as:

- Actors on the scene
  We identify the people whose jobs involve the day-to-day use of a large database; we call them the *actors on the scene.*
- Worker behind the scene
  Those who work to maintain the database system environment but who are not actively interested in the database contents as part of their daily job.

Actors on the scene

a. Database Administrator

One of the main reasons for using DBMSs is to have central control of both the data and the programs that access those data. A person who has such central control over the system is called a **database administrator** (**DBA**). In any organization where many people use the same resources, there is a need for a chief administrator to oversee and manage these resources. In a database environment, the primary resource is the database itself, and the secondary resource is the DBMS and related software. Administering these resources is the responsibility of the **DBA**. The DBA is responsible for authorizing access to the database, coordinating and monitoring its use, and acquiring software and hardware resources as needed. The DBA is accountable for problems such as security breaches and poor system response time. In large organizations, the DBA is assisted by a staff that carries out these functions.

The functions of a DBA include:

• **Schema definition**. The DBA creates the original database schema by executing a set of data definition statements in the DDL.

• **Storage structure and access-method definition**.

• **Schema and physical-organization modification**.

The DBA carries out changes to the schema and physical organization to reflect the changing needs of the organization, or to alter the physical organization to improve performance.

• **Granting of authorization for data access**. By granting different types of authorization, the database administrator can regulate which parts of the database various users can access. The authorization information is kept in a special system structure that the database system consults whenever someone attempts to access the data in the system.

• **Routine maintenance**. Examples of the database administrator's routine maintenance activities are:

1. Periodically backing up the database, either onto tapes or onto remote servers, to prevent loss of data in case of disasters such as flooding.
2. Ensuring that enough free disk space is available for normal operations, and upgrading disk space as required.
3. Monitoring jobs running on the database and ensuring that performance is not degraded by very expensive tasks submitted by some users.

b. Database Designers

Database designers are responsible for identifying the data to be stored in the database and for choosing appropriate structures to represent and store this data. These tasks are mostly undertaken before the database is actually implemented and populated with data. It is the responsibility of database designers to communicate with all prospective database users in order to understand their requirements and to create a

design that meets these requirements. In many cases, the designers are on the staff of the DBA and may be assigned other staff responsibilities after the database design is completed. Database designers typically interact with each potential group of users and develop **views** of the database that meet the data and processing requirements of these groups. Each view is then analyzed and *integrated* with the views of other user groups. The final database design must be capable of supporting the requirements of all user groups.

c. End Users

End users are the people whose jobs require access to the database for querying, updating, and generating reports; the database primarily exists for their use. There are several categories of end users:

Casual end users occasionally access the database, but they may need different information each time. They use a sophisticated database query language to specify their requests and are typically middle- or high-level managers or other occasional browsers.

Naive or **parametric end users** make up a sizable portion of database end users. Their main job function revolves around constantly querying and updating the database, using standard types of queries and updates—called **canned transactions—**that have been carefully programmed and tested. The tasks that such users perform are varied:

- Bank tellers check account balances and post withdrawals and deposits.
- Reservation agents for airlines, hotels, and car rental companies check availability for a given request and make reservations.
- Employees at receiving stations for shipping companies enter package identifications via bar codes and descriptive information through buttons to update a central database of received and in-transit packages.

Sophisticated end users include engineers, scientists, business analysts, and others who thoroughly familiarize themselves with the facilities of the DBMS in order to implement their own applications to meet their complex requirements.

Standalone users maintain personal databases by using ready-made program packages that provide easy-to-use menu-based or graphics-based interfaces. An example is the user of a tax package that stores a variety of personal financial data for tax purposes. A typical DBMS provides multiple facilities to access a database. Naive end users need to learn very little about the facilities provided by the DBMS; they simply have to understand the user interfaces of the standard transactions designed and implemented for their use. Casual users learn only a few facilities that they may use repeatedly. Sophisticated users try to learn most of the DBMS facilities in order to achieve their complex requirements. Standalone users typically become very proficient in using a specific software package.

d. System Analysts and Application Programmers (Software Engineers)

System analysts determine the requirements of end users, especially naive and parametric end users, and develop specifications for standard canned transactions that meet these requirements.

Application programmers implement these specifications as programs; then they test, debug, document, and maintain these canned transactions. Such analysts and programmers—commonly referred to as **software developers** or **software engineers**—should be familiar with the full range of capabilities provided by the DBMS to accomplish their tasks.

Worker behind the scene

In addition to those who design, use, and administer a database, others are associated with the design, development, and operation of the DBMS *software and system environment.* These persons are typically not interested in the database content itself. We call them the *workers behind the scene*, and they include the following categories:

DBMS system designers and implementers design and implement the DBMS modules and interfaces as a software package. A DBMS is a very complex software system that consists of many components, or **modules**, including modules for implementing the catalog, query language processing, interface processing, accessing and buffering data, controlling concurrency, and handling data recovery and security. The DBMS must interface with other system software such as the operating system and compilers for various programming languages.

Tool developers design and implement **tools**—the software packages that facilitate database modeling and design, database system design, and improved performance. Tools are optional packages that are often purchased separately. They include packages for database design, performance monitoring, natural language or graphical interfaces, prototyping, simulation, and test data generation. In many cases, independent software vendors develop and market these tools.

Operators and maintenance personnel (system administration personnel) are responsible for the actual running and maintenance of the hardware and software environment for the database system.

View of Data

A database system is a collection of interrelated data and a set of programs that allow users to access and modify these data. A major purpose of a database system is to provide users with an abstract view of the data. That is, the system hides certain details of how the data are stored and maintained.

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-system users are not computer trained, developers hide the complexity from users through several levels of abstraction, to simplify users' interactions with the system:

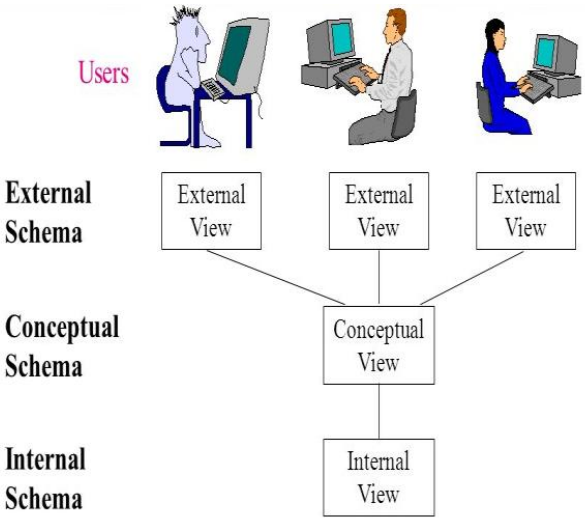

Figure: Levels of Abstraction in a DBMS

• Physical level (or Internal View / Schema): 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 (or Conceptual View / Schema): 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. This is referred to as physical data independence. Database administrators, who must decide what information to keep in the database, use the logical level of abstraction.

• View level (or External View / Schema): 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. Above Figure shows the relationship among the three levels of abstraction.

An analogy to the concept of data types in programming languages may clarify the distinction among levels of abstraction. Many high-level programming languages support the notion of a structured type. For example, we may describe a record as follows:

```
type instructor = record
ID : char (5);
name : char (20);
dept name : char (20);
salary : numeric (8,2);
end;
```

This code defines a new record type called instructor with four fields. Each field has a name and a type associated with it. A university organization may have several such record types, including

- department, with fields dept_name, building, and budget
- course, with fields course_id, title, dept_name, and credits
- student, with fields ID, name, dept_name, and tot_cred

At the physical level, an instructor, department, or student record can be described as a block of consecutive storage locations. The compiler hides this level of detail from programmers. Similarly, the database system hides many of the lowest-level storage details from database programmers. Database administrators, on the other hand, may be aware of certain details of the physical organization of the data.

At the logical level, each such record is described by a type definition, as in the previous code segment, and the interrelationship of these record types is defined as well. Programmers using a programming language work at this level of abstraction. Similarly, database administrators usually work at this level of abstraction.

Finally, at the view level, computer users see a set of application programs that hide details of the data types. At the view level, several views of the database are defined, and a database user sees some or all of these views.

In addition to hiding details of the logical level of the database, the views also provide a security mechanism to prevent users from accessing certain parts of the database. For example, clerks in the university registrar office can see only that part of the database that has information about students; they cannot access information about salaries of instructors.

Data Independence

The three-schema architecture can be used to 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 schema or application programs. We may change the conceptual schema to expand the database (by adding a record type or data item), or to reduce the database (by removing a record type or data item). In the latter case, external schema that refer only to the remaining data should not be affected. Only the view definition and the mappings need be changed in a DBMS that supports logical data independence. Application programs that reference the external schema constructs must work as before, after the conceptual schema undergoes a logical reorganization. Changes to constraints can be applied also to the conceptual schema without affecting the external schema or application programs.

2. Physical data independence is the capacity to change the internal schema without having to change the conceptual (or external) schema. 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.

Instances and Schema

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. Schema are changed infrequently, if at all. The concept of database schema 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 schema, 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 schema at the view level, sometimes called subschemas, which 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.

Database System Structure

A database system is partitioned into modules that deal with each of the responsibilities of the overall system. The functional components of a database system can be broadly divided into the storage manager and the query processor components. The storage manager is important because databases typically require a large amount of storage space. Corporate databases range in size from hundreds of gigabytes to, for the largest databases, terabytes of data. A gigabyte is 1000 megabytes (1 billion bytes), and a terabyte is 1 million megabytes (1 trillion bytes). Since the main memory of computers cannot store this much information, the information is stored on disks. Data are moved between disk storage and main memory as needed. Since the movement of data to and from disk is slow relative to the speed of the central

processing unit, it is imperative that the database system structure the data so as to minimize the need to move data between disk and main memory. The query processor is important because it helps the database system simplify and facilitate access to data. High-level views help to achieve this goal; with them, users of the system are not be burdened unnecessarily with the physical details of the implementation of the system. However, quick processing of updates and queries is important. It is the job of the database system to translate updates and queries written in a nonprocedural language, at the logical level, into an efficient sequence of operations at the physical level.

- **Storage Manager**

A *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 for the interaction with the file manager. The raw data are stored on the disk using the file system, which is usually provided by a conventional operating system. The storage manager translates the various DML statements into low-level file-system commands. Thus, the storage manager is responsible for storing, retrieving, and updating data in the database.

The storage manager components include:
• **Authorization and integrity manager**, which tests for the satisfaction of integrity constraints and checks the authority of users to access data.

• **Transaction manager**, which ensures that the database remains in a consistent (correct) state despite system failures, and that concurrent transaction executions proceed without conflicting.

• **File manager**, which manages the allocation of space on disk storage and the data structures used to represent information stored on disk.

• **Buffer manager**, which is responsible for fetching data from disk storage into main memory, and deciding what data to cache in main memory. The buffer manager is a critical part of the database system, since it enables the database to handle data sizes that are much larger than the size of main memory. The storage manager implements several data structures as part of the physical system implementation:
• **Data files**, which store the database itself.
• **Data dictionary**, which stores metadata about the structure of the database, in particular the schema of the database.
• **Indices**, which provide fast access to data items that hold particular values.

- **The Query Processor**

The query processor components include
• **DDL interpreter**, which interprets DDL statements and records the definitions in the data dictionary.
• **DML** compiler, which translates DML statements in a query language into an evaluation plan consisting of low-level instructions that the query evaluation engine understands. A query can usually be translated into any of a number of alternative evaluation plans that all give the same result. The DML compiler also performs **query optimization**, that is, it picks the lowest cost evaluation plan from among the alternatives.
• **Query evaluation engine**, which executes low-level instructions generated by the DML compiler.
Figure 1.4 shows these components and the connections among them.

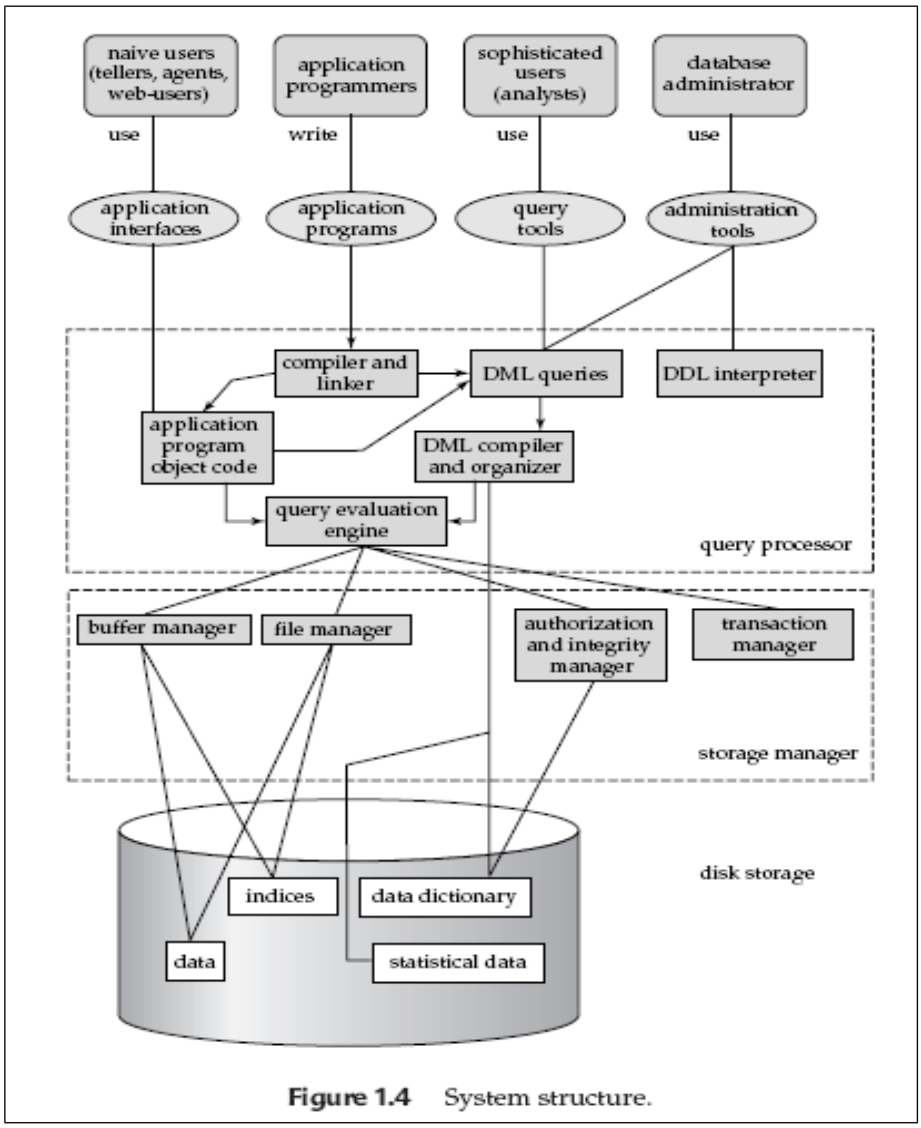

Figure 1.4 System structure.

Application Architectures

Most users of a database system today are not present at the site of the database system, but connect to it through a network. We can therefore differentiate between **client** machines, on which remote database users work, and **server** machines, on which the database system runs. Database applications are usually partitioned into two or three parts, as in Figure 1.5. In a **two-tier architecture**, the application is partitioned into a component that resides at the client machine, which invokes database system functionality at the server machine through query language statements. Application program interface standards like ODBC and JDBC are used for interaction between the client and the server.

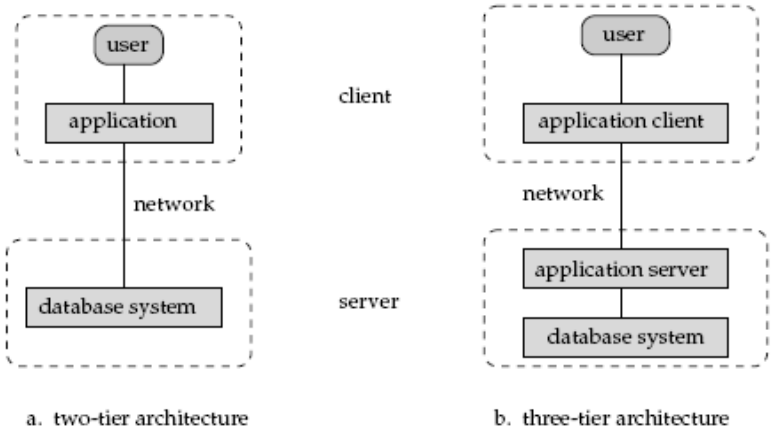

Figure 1.5 Two-tier and three-tier architectures.

In contrast, in a **three-tier architecture**, the client machine acts as merely a front end and does not contain any direct database calls. Instead, the client end communicates with an **application server**, usually through a forms interface. The application server in turn communicates with a database system to access data. The **business logic** of the application, which says what actions to carry out under what conditions, is embedded in the application server, instead of being distributed across multiple clients. Three-tier applications are more appropriate for large applications, and for applications that run on theWorld WideWeb.

Source:

Silberschatz−Korth−Sudarshan:
Database SystemConcepts,
Fourth Edition

Ramez Elmasri, Shamkant B. Navathe
Fundamentals of Database System
Fourth Edition

Long Answer Type Questions:

1. Explain the structure of DBMS.
2. Describe the overall Architecture of DBMS with the diagram.
3. Write a short notes on – DBA
4. Define the following terms:
5. Data independence, user view, DBA, end user, canned transaction.
6. What are the responsibilities of the DBA and the database designers?
7. What are the different types of database end users? Discuss the main activities of each.
8. Explain Users of Database system in detail.
9. What are the characteristics of the database systems?