1. Introductory Concepts of DBMS

Objectives:

1. Introduction and applications of DBMS
2. Purpose of database
3. Types of Database
4. View of Data
5. Data Independence
6. Database System architecture- levels, Mappings
7. Database users and DBA

Introduction and applications of DBMS:

Data: Raw and isolated factors about entity are known as Data.
e.g., text, image, map, video etc.

Information: Processed, meaningful & usable data is known as Information.

Data	Information
What an observer/reader has/read	What an observer/reader understands
Same for all	Depends on observer
e.g., List of buses	e.g., List of buses to Surat for a person going to Surat.

Database: Collection of similar/related data.
e.g., youtube(videos), google maps(maps), gaana(songs) etc.

Database Management System (DBMS): Software used to insert, manipulate & delete databases. e.g., oracle 10g, mongoDB etc.
The primary goal of a DBMS is to provide a way to store and retrieve database information that is both *convenient* and *efficient*.

Applications:

Databases are widely used. Here are some representative applications:

- *Enterprise Information*
  - *Sales*: For customer, product, and purchase information.
  - *Accounting*: For payments, receipts, account balances, assets and other accounting information.
  - *Human resources*: For information about employees, salaries, payroll taxes, and benefits, and for generation of paychecks.
  - *Manufacturing*: For management of the supply chain and for tracking production of items in factories, inventories of items in warehouses and stores, and orders for items.

- *Online retailers*: For sales data noted above plus online order tracking, generation of recommendation lists, and maintenance of online product evaluations.

- *Banking and Finance*
  - *Banking*: For customer information, accounts, loans, and banking transactions.
  - *Credit card transactions*: For purchases on credit cards and generation of monthly statements.
- *Finance*: For storing information about holdings, sales, and purchases of financial instruments such as stocks and bonds; also, for storing real-time market data to enable online trading by customers and automated trading by the firm.
- *Universities*: For student information, course registrations, and grades (in addition to standard enterprise information such as human resources and accounting).
- *Airlines*: For reservations and schedule information. Airlines were among the first to use databases in a geographically distributed manner.
- *Telecommunication*: For keeping records of calls made, generating monthly bills, maintaining balances on prepaid calling cards, and storing information about the communication networks.

Purpose of Database:

Typical **file-processing system** is supported by a conventional operating system. The system stores permanent records in various files, and it needs different application programs to extract records from, and add records to, the appropriate files. Before database management systems (DBMSs) were introduced, organizations usually stored information in such systems.

The following major disadvantages of FPS prompted the development of DBMS:

1. **Data redundancy and inconsistency**. Since different programmers create the files and application programs over a long period, the various files are likely to have different structures and the programs may be written in several programming languages. Moreover, the same information may be duplicated in several places (files). For example, if a student has a double major (say, music and mathematics) the address and telephone number of that student may appear in a file that consists of student records of students in the Music department and in a file that consists of student records of students in the Mathematics department. This redundancy leads to higher storage and access cost. In addition, it may lead to **data inconsistency**; that is, the various copies of the same data may no longer agree. For example, a changed student address may be reflected in the Music department records but not elsewhere in the system.
2. **Difficulty in accessing data.** Suppose that one of the university clerks needs to find out the names of all students who live within a particular postal-code area. The clerk asks the data-processing department to generate such a list. Because the designers of the original system did not anticipate this request, there is no application program on hand to meet it. There is, however, an application program to generate the list of *all* students. The university clerk has now two choices: either obtain the list of all students and extract the needed information manually or ask a programmer to write the necessary application program. Both alternatives are obviously unsatisfactory. Suppose that such a program is written, and that, several days later, the same clerk needs to trim that list to include only those students who have taken at least 60 credit hours. As expected, a program to generate such a list does not exist. Again, the clerk has the preceding two options, neither of which is satisfactory. The point here is that conventional file-

processing environments do not allow needed data to be retrieved in a convenient and efficient manner. More responsive data-retrieval systems are required for general use.

3. **Data isolation.** Because data are scattered in various files, and files may be in different formats, writing new application programs to retrieve the appropriate data is difficult.
4. **Integrity problems**. The data values stored in the database must satisfy certain types of **consistency constraints**. Suppose the university maintains an account for each department, and records the balance amount in each account. Suppose also that the university requires that the account balance of a department may never fall below zero. Developers enforce these constraints in the system by adding appropriate code in the various application programs. However, when new constraints are added, it is difficult to change the programs to enforce them. The problem is compounded when constraints involve several data items from different files.
5. **Atomicity problems**. A computer system, like any other device, is subject to failure. In many applications, it is crucial that, if a failure occurs, the data be restored to the consistent state that existed prior to the failure. Consider a program to transfer \$500 from the account balance of department *A* to the account balance of department *B*. If a system failure occurs during the execution of the program, it is possible that the \$500 was removed from the balance of department *A* but was not credited to the balance of department *B*, resulting in an inconsistent database state. Clearly, it is essential to database consistency that either both the credit and debit occur, or that neither occur. That is, the funds transfer must be *atomic*—it must happen in its entirety or not at all. It is difficult to ensure atomicity in a conventional file-processing system.
6. **Concurrent-access anomalies**. For the sake of overall performance of the system and faster response, many systems allow multiple users to update the data simultaneously. Indeed, today, the largest Internet retailers may have millions of accesses per day to their data by shoppers. In such an environment, interaction of concurrent updates is possible and may result in inconsistent data. Consider department *A*, with an account balance of \$10,000. If two department clerks debit the account balance (by say \$500 and \$100, respectively) of department *A* at almost exactly the same time, the result of the concurrent executions may leave the budget in an incorrect (or inconsistent) state. Suppose that the programs executing on behalf of each withdrawal read the old balance, reduce that value by the amount being withdrawn, and write the result back. If the two programs run concurrently, they may both read the value \$10,000, and write back \$9500 and \$9900, respectively. Depending on which one writes the value last, the account balance of department *A* may contain either \$9500 or \$9900, rather than the correct value of \$9400. To guard against this possibility, the system must maintain some form of supervision. But supervision is difficult to provide because data may be accessed by many different application programs that have not been coordinated previously.

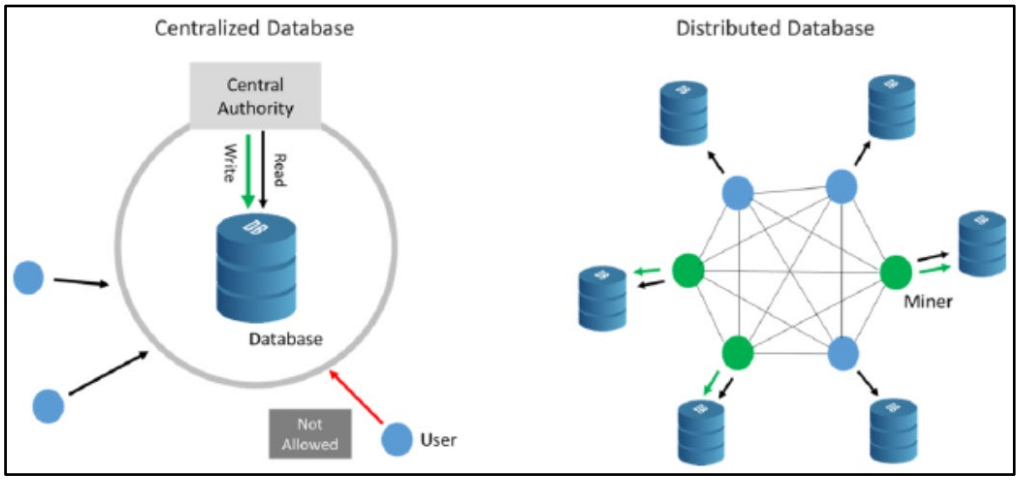
   As another example, suppose a registration program maintains a count of students registered for a course, in order to enforce limits on the number of students registered. When a student registers, the program reads the current count for the courses, verifies that the count is not already at the limit, adds one to the count, and stores the count back in the database. Suppose two students register concurrently, with the count at (say) 39. The two program executions may both read the value 39, and both would then write back 40, leading to an incorrect increase of only 1, even though two students successfully registered for the course and the count should be 41. Furthermore, suppose the course registration limit was 40; in the above case both students would be able to register, leading to a violation of the limit of 40 students.
7. **Security problems**. Not every user of the database system should be able to access all the data. For example, in a university, payroll personnel need to see only that part of

the database that has financial information. They do not need access to information about academic records. But, since application programs are added to the file-processing system in an ad hoc manner, enforcing such security constraints is difficult.

DBMS	FPS
Redundancy and Inconsistency in data are reduced due to single format. Also, Duplication of data is eliminated.	Both exists.
Data are easily accessed.	Special application program is needed.
Isolation of data is possible due to single format.	Difficult.
Atomicity update is possible.	Not possible.
Security features can be enabled.	Difficult.

Types of Database:

- **Based on number of users:**
  1. Single user
  2. Multiple users
- **Based on location:**
  1. Centralized
  2. Distributed



- **Based on how they used:**
  1. Operational (e.g., Enterprise)
  2. Transactional (e.g., strategic uses)
- **Based on Degree to which data are structured:**
  1. Unstructured: Raw data
  2. Semi-structured: Some data is organised
  3. Structured: Fully organised data

View of Data:

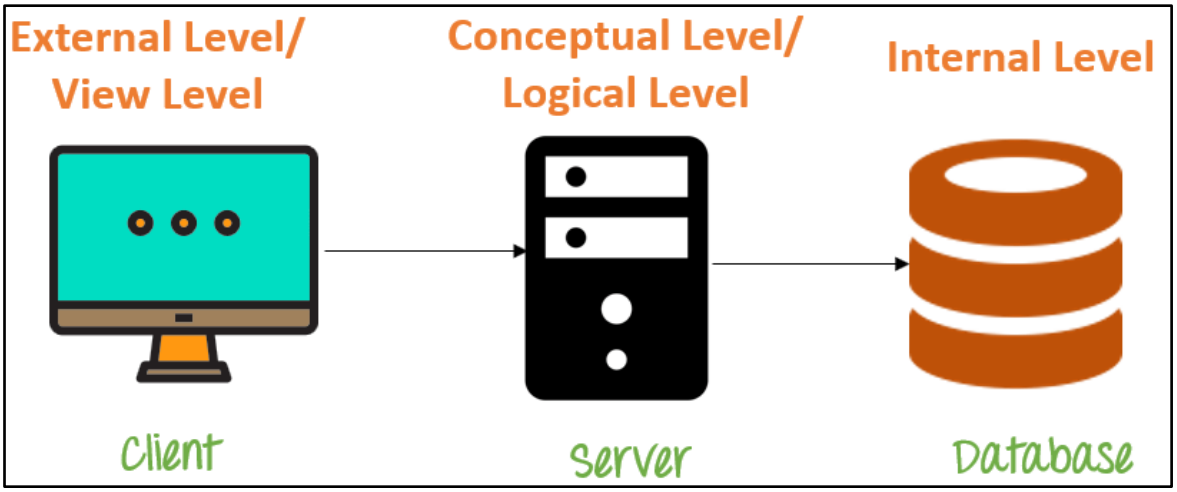
A major purpose of a database system is to provide users with an *abstract* view of the data. That is, the system hides certain details of how the data are stored and maintained.

Reason/Purpose of Abstract view:

1. To hide details from users
2. To hide complexity of Database from users
3. To simplify user interaction with data

These purposes can be achieved by several levels of abstractions as below.

- **Physical/Internal level:** The lowest level of abstraction describes ***how the data are actually stored***. The physical level describes complex low-level data structures in detail.
- **Logical/Conceptual level:** The next-higher level of abstraction describes ***what data are stored in the database, and what relationships exist among those data***. The logical level thus describes the entire database in terms of a small number of relatively simple structures. Although implementation of the simple structures at the logical level may involve complex physical-level structures, the user of the logical level does not need to be aware of this complexity. *Database administrators, who must decide what information to keep in the database, use the logical level of abstraction.*
- **View/External level:** The highest level of abstraction describes only part of the entire database. Even though the logical level uses simpler structures, complexity remains because of the variety of information stored in a large database. Many users of the database system do not need all this information; instead, they need to access only a part of the database. The view level of abstraction exists to simplify their interaction with the system. The system may provide many views for the same database. It is defined by means of external schema, which consists of definition of each of the various external record types in external view.



Data Independence:

Instance: The collection of information stored in the database at a particular moment is called an **instance** of the database.

Schema: The overall design of the database is called the database **schema**.

Database systems have several schemas, partitioned according to the levels of abstraction. The **physical schema** describes the database design at the physical level, while the **logical schema** describes the database design at the logical level. A database may also have several schemas at the view level, sometimes called **subschemas**, that describe different views of the database.

- Data independence can be explained using the three-schema architecture.
- Data independence refers characteristic of being able to modify the schema at one level of the database system without altering the schema at the next higher level.

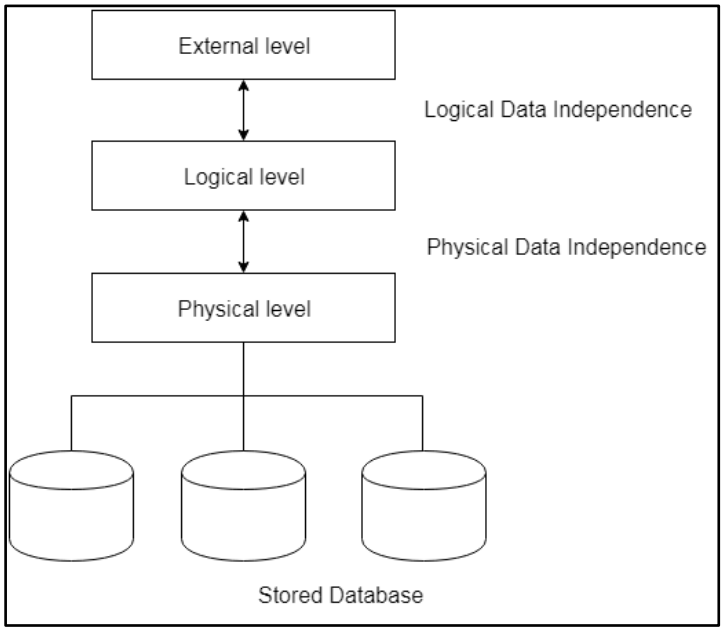
There are two types of data independence:

1. Logical Data Independence

- Logical data independence refers characteristic of being able to change the conceptual schema without having to change the external schema.
- Logical data independence is used to separate the external level from the conceptual view.
- If we do any changes in the conceptual view of the data, then the user view of the data would not be affected.
- Logical data independence occurs at the user interface level.

2. Physical Data Independence

- Physical data independence can be defined as the capacity to change the internal schema without having to change the conceptual schema.
- If we do any changes in the storage size of the database system server, then the Conceptual structure of the database will not be affected.
- Physical data independence is used to separate conceptual levels from the internal levels.
- Physical data independence occurs at the logical interface level.



DBMS Architecture

- The DBMS design depends upon its architecture. The basic client/server architecture is used to deal with a large number of PCs, web servers, database servers and other components that are connected with networks.
- The client/server architecture consists of many PCs and a workstation which are connected via the network.
- DBMS architecture depends upon how users are connected to the database to get their request done.

Types of DBMS Architecture

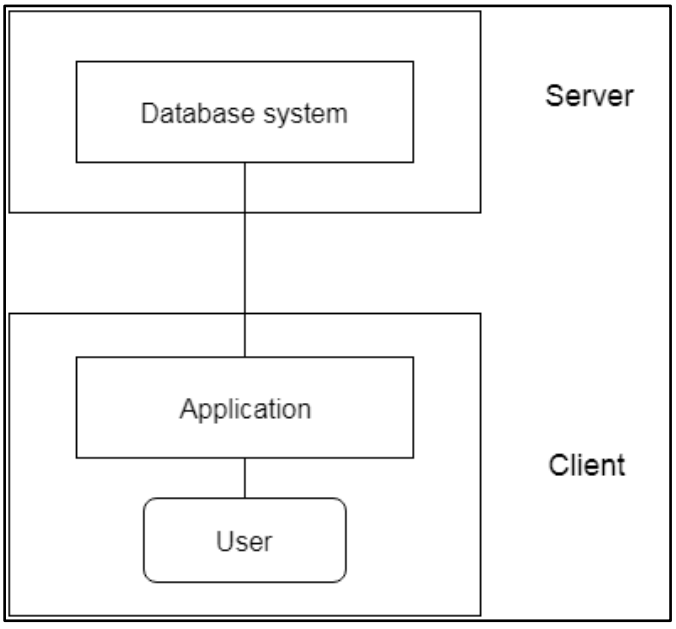
Database architecture can be seen as a single tier or multi-tier. But logically, database architecture is of two types like: **2-tier architecture** and **3-tier architecture**.

1-Tier Architecture

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

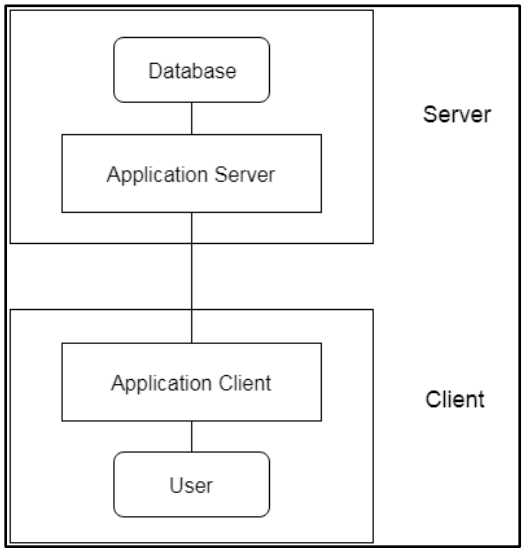
2-Tier Architecture

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

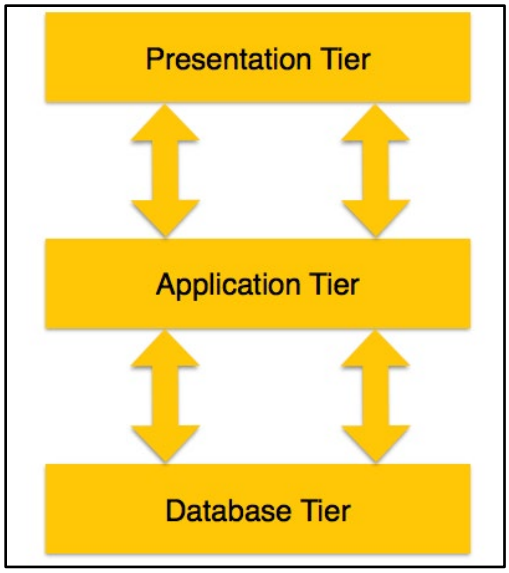


3-Tier Architecture

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



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.



- **Database (Data) Tier** − At this tier, the database resides along with its query processing languages. We also have the relations that define the data and their constraints at this level.
- **Application (Middle) Tier** − At this tier reside the application server and the programs that access the database. For a user, this application tier presents an abstracted view of

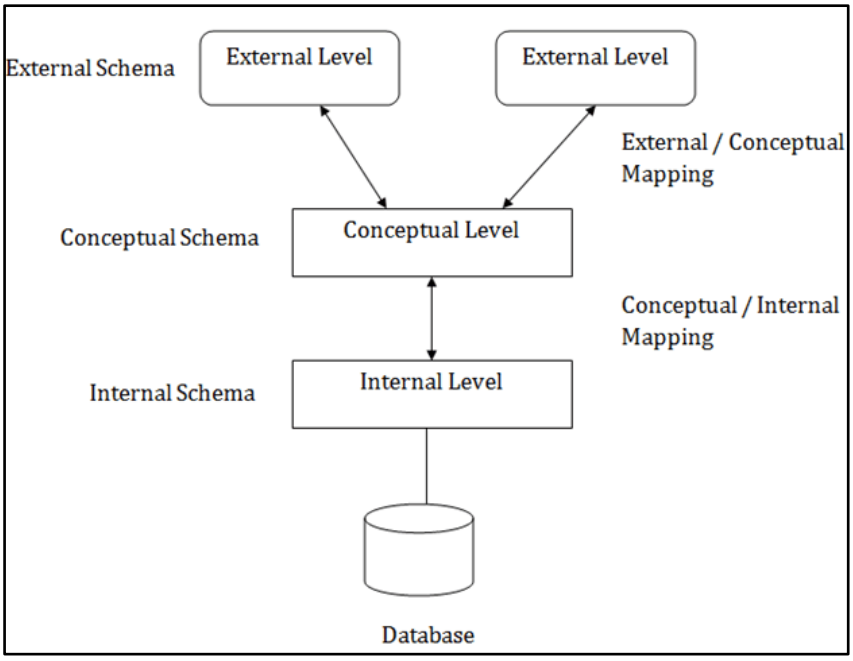
the database. End-users are unaware of any existence of the database beyond the application. At the other end, the database tier is not aware of any other user beyond the application tier. Hence, the application layer sits in the middle and acts as a mediator between the end-user and the database.

- **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.

Three schema Architecture

- The three-schema architecture is also called ANSI/SPARC architecture or three-level architecture.
- This framework is used to describe the structure of a specific database system.
- The three-schema architecture is also used to separate the user applications and physical database.
- The three-schema architecture contains three-levels. It breaks the database down into three different categories.

The three-schema architecture is as follows:



In the above diagram:

- It shows the DBMS architecture.
- Mapping is used to transform the request and response between various database levels of architecture.
- Mapping is not good for small DBMS because it takes more time.
- In **External / Conceptual mapping**, it is necessary to transform the request from external level to conceptual schema. It defines the correspondence between external & conceptual view.

- In **Conceptual / Internal mapping**, DBMS transform the request from the conceptual to internal level. It defines the correspondence between conceptual & stored DB. It specifies how conceptual records and fields are represented at internal level.

1. Internal (/Physical/Storage) Level:

- The internal level has an internal schema which describes the physical storage structure of the database.
- It is defined by means of internal schema.
- It consists of many occurrences of many types of different internal records.
- The internal schema is also known as a physical schema.
- It uses the physical data model. It is used to define that how the data will be stored in a block.
- The physical level is used to describe complex low-level data structures in detail.

2. Conceptual (/Logical/ Community logical) Level:

- The conceptual schema describes the design of a database at the conceptual level. Conceptual level is also known as logical level.
- It is the level of Indirection between other 2 levels.
- It is defined by means of conceptual schema.
- The conceptual schema describes the structure of the whole database.
- The conceptual level describes what data are to be stored in the database and also describes what relationship exists among those data.
- In the conceptual level, internal details such as an implementation of the data structure are hidden.
- Programmers and database administrators work at this level.

3. External (/View/User logical) Level:

- External level is individual user level.
- At the external level, a database contains several schemas that sometimes called as subschema. The subschema is used to describe the different view of the database.
- An external schema is also known as view schema.
- Each view schema describes the database part that a particular user group is interested and hides the remaining database from that user group.
- The view schema describes the end user interaction with database systems.

The external & conceptual levels will be relational, but internal level will not.

Database Users:

There are four different types of database-system users, differentiated by the way they expect to interact with the system. Different types of user interfaces have been designed for the different types of users.

- **Naïve users (Parametric end users)** don't have any kind of knowledge regarding DBMS. They are unsophisticated users who interact with the system by invoking one of the application programs that have been written previously. For example, a clerk in the university who needs to add a new instructor to department *A* invokes a program called *new hire*. This program asks the clerk for the name of the new instructor, her new *ID*, the name of the department (that is, *A*), and the salary. The typical user interface for naïve users is a forms interface, where the user can fill in appropriate fields of the form. Naïve users may also simply read *reports* generated from the database.

As another example, consider a student, who during class registration period, wishes to register for a class by using a Web interface. Such a user connects to a Web application program that runs at a Web server. The application first verifies the identity of the user, and allows her to access a form where she enters the desired information. The form information is sent back to the Web application at the server, which then determines if there is room in the class (by retrieving information from the database) and if so adds the student information to the class roster in the database.

- **Application programmers** are computer professionals who write application programs. Application programmers can choose from many tools to develop user interfaces. **Rapid application development (RAD)** tools are tools that enable an application programmer to construct forms and reports with minimal programming effort.
- **System Analyst** is a user who analyses the requirements of parametric end users. They check whether all the requirements of end users are satisfied.
- **Sophisticated users (End users)** interact with the system without writing programs. Instead, they form their requests either using a database query language or by using tools such as data analysis software. Analysts who submit queries to explore data in the database fall in this category.
- **Specialized users** are sophisticated users who write specialized database applications that do not fit into the traditional data-processing framework. Among these applications are computer-aided design systems, knowledgebase and expert systems, systems that store data with complex data types (for example, graphics data and audio data), and environment-modelling systems.

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