# DBMS DataBase Management System

Author: **Rahul R. Prajapati** Mentor: **Mrs. Sweta Nigam**

**MCA 6th Semester**
Tilak Maharashtra Vidyapeeth Pune

**Abstract**

A database is a system used to manage data on a computer system. There are several database work systems that have their own laws and ways of working. Data is arranged in various levels. Computer data are classified in a hierarchy. A higher level consists of one or more data at a lower level. Example: a folder has several sub folders, some subfolders have several files. Management of database management requires a tool / tool to be able to manage it, so that database management can continue to be managed and continue to improve its performance. With the existence of an information system, an organization will strive to be more competitive and efficient, which in turn adds value to obtaining, changing and distributing information with the aim of increasing decision making, increasing organizational performance in achieving its organizational goals. An effective Information System provides accurate, timely and relevant information to users so that it can be used for decision making. In making decisions, both in daily operations, as well as in strategic planning into the future. The decision-making process must be based on timely and appropriate data and information so that the decisions taken are on target. Information is obtained from data processing, and data processing is carried out by information systems with the support of information technology.

## 1.0 Objectives

At the end of this chapter the reader will be able to:

- Distinguish between data and information and Knowledge
- Distinguish between file processing system and DBMS
- Describe DBMS its advantages and disadvantages
- Describe Database users including data base administrator
- Describe data models, schemas and instances.
- Describe DBMS Architecture & Data Independence
- Describe Data Languages

## 1.1 Introduction

A **database-management system** (DBMS) is a collection of interrelated data and a set of programs to access those data. This is a collection of related data with an implicit meaning and hence is a database. 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*. By **data,** we mean known facts that can be recorded and that have implicit meaning. For example, consider the names, telephone numbers, and addresses of the people you know. You may have recorded this data in an indexed address book, or you may have stored it on a diskette, using a personal computer and software such as DBASE IV or V, Microsoft ACCESS, or EXCEL. A **datum** – a unit of data – is a symbol or a set of symbols which is used to represent something. This relationship between symbols and what they represent is the essence of what we mean by **information**. Hence, information is interpreted data – data supplied with semantics. **Knowledge** refers to the practical use of information. While information can be transported, stored or shared without many difficulties the same can not be said about knowledge. Knowledge necessarily involves a personal experience. Referring back to the scientific experiment, a third person reading the results will have information about it, while the person who conducted the experiment personally will have knowledge about it.

Database systems are designed to manage large bodies of information. Management of data involves both defining structures for storage of information and providing mechanisms for the manipulation of information. In addition, the database system must ensure the safety of the information stored, despite system crashes or attempts at unauthorized access. If data are to be shared among several users, the system must avoid possible anomalous results.

Because information is so important in most organizations, computer scientists have developed a large body of concepts and techniques for managing data. These concepts and techniques form the focus of this book. This chapter briefly introduces the principles of database systems.

### 1.1 Data Processing Vs. Data Management Systems

Although Data Processing and Data Management Systems both refer to functions that take raw data and transform it into usable information, the usage of the terms is very different. **Data Processing** is the term generally used to describe what was done by large mainframe computers from the late 1940's until the early 1980's (and which continues to be done in most large organizations to a greater or lesser extent even today): large volumes of raw transaction data fed into programs that update a master file, with fixed- format reports written to paper.

The term **Data Management Systems** refers to an expansion of this concept, where the raw data, previously copied manually from paper to punched cards, and later into data- entry terminals, is now fed into the system from a variety of sources, including ATMs, EFT, and direct customer entry through the Internet. The master file concept has been largely displaced by database management systems, and static reporting replaced or augmented by ad-hoc reporting and direct inquiry, including downloading of data by customers. The ubiquity of the Internet and the Personal Computer have been the driving force in the transformation of Data Processing to the more global concept of Data Management Systems.

### 1.2 File Oriented Approach

The earliest business computer systems were used to process business records and produce information. They were generally faster and more accurate than equivalent manual systems. These systems stored groups of records in separate files, and so they were called **file processing systems.** In a typical file processing system, each department has its own files, designed specifically for those applications. The department itself, working with the data processing staff, sets policies or standards for the format and maintenance of its files.

Programs are dependent on the files and vice-versa; that is, when the physical format of the file is changed, the program has also to be changed. Although the traditional file oriented approach to information processing is still widely used, it does have some very important disadvantages.

### 1.3 Database Oriented Approach to Data Management

Consider part of a savings-bank enterprise that keeps information about all customers and savings accounts. One way to keep the information on a computer is to store it in operating system files. To allow users to manipulate the information, the system has a number of application programs that manipulate the files, including.

- A program to debit or credit an account A
  program to add a new account
- A program to find the balance of an account A
  program to generate monthly statements

System programmers wrote these application programs to meet the needs of the bank. New application programs are added to the system as the need arises. For example, suppose that the savings bank decides to offer checking accounts. As a result, the bank creates new permanent files that contain information about all the checking accounts maintained in the bank, and it may have to write new application programs to deal with situations that do not arise in savings accounts, such as overdrafts. Thus, as time goes by, the system acquires more files and more application programs.

This 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) came along, organizations usually stored information in such systems.

Keeping organizational information in a file-processing system has a number of major disadvantages:
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 formats and the programs may be written in several programming languages. Moreover, the same information may be duplicated in several places (files). For example, the address and telephone number of a particular customer may appear in a

file that consists of savings-account records and in a file that consists of checking-account records. 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 customer address may be reflected in savings-account records but not elsewhere in the system.

Suppose that one of the bank officers needs to find out the names of all customers who live within a particular postal-code area. The officer 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* customers. The bank officer has now two choices: either obtain the list of all customers and extract the needed information manually or ask a system 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 officer needs to trim that list to include only those customers who have an account balance of $10,000 or more. As expected, a program to generate such a list does not exist. Again, the officer 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.

**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.

**Integrity problems**: The data values stored in the database must satisfy certain types of **consistency constraints**. For example, the balance of a bank account may never fall below a prescribed amount (say, $25). 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.

**Atomicity problems**: A computer system, like any other mechanical or electrical 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 $50 from account *A* to account *B*. If a system failure occurs during the execution of the program, it is possible that the $50 was removed from account *A* but was not credited to account *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.

**Security problems**: Not every user of the database system should be able to access all the data. For example, in a banking system, payroll personnel need to see only that part of the database that has information about the various bank employees. They do not need access to information about customer accounts. But, since application programs are added to the system in an ad hoc manner, enforcing such security constraints is difficult. These difficulties, among others, prompted the development of database systems. In what follows, we shall see the concepts and algorithms that enable database systems to solve the problems with file-processing systems. In most of this book, we use a bank enterprise as a running example of a typical data-processing application found in a corporation.

**1.4 Characteristics of Database**

The database approach has some very characteristic features which are discussed in detail below:

### 1.5.1 Concurrent Use

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.

### 1.5.2 Structured and Described Data

A fundamental feature of the database approach is that the database system 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").

### 1.5.3 Separation of Data and Applications

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 standardised interface with the help of a standardised 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 separated from the data. Therefore database internal reorganisations or improvement of efficiency do not have any influence on the application software.

### 1.5.4 Data Integrity

Data integrity is a byword for the quality and the reliability of the data of a database system. In a broader sense data integrity includes also the protection of the database from unauthorised access (confidentiality) and unauthorised changes. Data reflects facts of the real world. database.

### 1.5.5 Transactions

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 inevitably inconsistent. A transaction is atomic, which 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.

### 1.5.6 Data Persistence

Data persistence means that in a DBMS all data is maintained as long as it is not deleted explicitly. The life span of data needs to be determined directly or indirectly by the user and must not be dependent on system features. Additionally data once stored in a database must not be lost. Changes of a database which are done by a transaction are persistent. When a transaction is finished even a system crash cannot put the data in danger.

### 1.5 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. Given all these advantages, is there ever a reason *not* to use a DBMS? A DBMS is a complex piece of software, optimized for certain kinds of workloads (e.g., answering complex queries or handling many concurrent requests), and its performance may not be adequate for certain specialized applications. Examples include applications with tight real-time constraints or applications with just a few well-designed critical operations for which efficient custom code must be written. Another reason for not using a DBMS is that an application may need to manipulate the data in ways not supported by the query language. In such a situation, the abstract view of the data presented by the DBMS does not match the application's needs, and actually gets in the way. As an example, relational databases do not support flexible analysis of text data (although vendors are now extending their products in this direction). If specialized performance or data manipulation requirements are central to an application, the application may choose not to use a DBMS, especially if the added benefits of a DBMS (e.g., flexible querying, security, concurrent access, and crash recovery) are not required. In most situations calling for large-scale data management, however, DBMSs have become an indispensable tool.

**Disadvantages of a DBMS**

**Danger of an 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 specialised software which is produced and optimised exactly for one problem.

**1.6 Instances and Schemas**

Databases change over time as information is inserted and deleted. The collection of information stored in the database at a particular moment is called an **instance** of the database. The overall design of the database is called the database **schema**. Schemas are changed infrequently, if at all.

The concept of database schemas and instances can be understood by analogy to a program written in a programming language. A database schema corresponds to the variable declarations (along with associated type definitions) in a program. Each variable has a particular value at a given instant. The values of the variables in a program at a point in time correspond to an *instance* of a database schema.

Database systems have several schemas, partitioned according to the levels of abstraction.

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

Of these, the logical schema is by far the most important, in terms of its effect on application programs, since programmers construct applications by using the logical schema. The physical schema is hidden beneath the logical schema, and can usually be changed easily without affecting application programs. Application programs are said to exhibit **physical data independence** if they do not depend on the physical schema, and thus need not be rewritten if the physical schema changes.

We study languages for describing schemas, after introducing the notion of data models in the next section.

**1.7 Data Models**

Underlying the structure of a database is the **data model**: a collection of conceptual tools for describing data, data relationships, data semantics, and consistency constraints.

To illustrate the concept of a data model, we outline two data models in this section: the entity-relationship model and the relational model. Both provide a way to describe the design of a database at the logical level.

**1.8.1 The Entity-Relationship Model**

The entity-relationship (E-R) data model is based on a perception of a real world that consists of a collection of basic objects, called *entities*, and of *relationships* among these objects. An entity is a "thing" or "object" in the real world that is distinguishable from other objects. For example, each person is an entity, and bank accounts can be considered as entities.

Entities are described in a database by a set of **attributes**. For example, the attributes *account-number* and *balance* may describe one particular account in a bank, and they form attributes of the *account* entity set. Similarly, attributes *customer-name*, *customer- street* address and *customer-city* may describe a *customer* entity.

An extra attribute *customer-id* is used to uniquely identify customers (since it may be possible to have two customers with the same name, street address, and city).

A unique customer identifier must be assigned to each customer. In the United States, many enterprises use the social-security number of a person (a unique number the U.S. government assigns to every person in the United States) as a customer identifier.

A **relationship** is an association among several entities. For example, a *depositor*

relationship associates a customer with each account that she has. The set of all entities of the same type and the set of all relationships of the same type are termed an **entity set** and **relationship set**, respectively.

The overall logical structure (schema) of a database can be expressed graphically by an *E-R diagram.*

### 1.8.2 Relational Model

The relational model uses a collection of tables to represent both data and the relationships among those data. Each table has multiple columns, and each column has a unique name.
The data is arranged in a relation which is visually represented in a two dimensional table. The data is inserted into the table in the form of tuples (which are nothing but rows). A tuple is formed by one or more than one attribute, which are used as basic building blocks in the formation of various expressions that are used to derive meaningful information. There can be any number of tuples in the table, but all the tuples contain fixed and same attributes with varying values. The relational model is implemented in database where a relation is represented by a table, a tuple is represented by a row, an attribute is represented by a column of the table, attribute name is the name of the column such as 'identifier', 'name', 'city' etc., attribute value contains the value for column in the row. Constraints are applied to the table and form the logical schema. In order to facilitate the selection of a particular row/tuple from the table, the attributes.

**i.e.** column names are used, and to expedite the selection of the rows some fields are defined uniquely to use them as indexes, this helps in searching the required data as fast as possible. All the relational algebra operations, such as Select, Intersection, Product, Union, Difference, Project, Join, Division, Merge etc. can also be performed on the Relational Database Model. Operations on the Relational Database Model are facilitated with the help of different conditional expressions, various key attributes, pre-defined constraints etc.

### 1.8.3 Other Data Models

The **object-oriented data model** is another data model that has seen increasing attention. The object-oriented model can be seen as extending the E-R model with notions object- oriented data model.

The **object-relational data model** combines features of the object-oriented data model and relational data model. Semistructured data models permit the specification of data where individual data items of the same type may have different sets of attributes. This is in contrast with the data models mentioned earlier, where every data item of a particular type must have the same set of attributes. The **extensible markup language (XML)** is widely used to represent semistructured data.

Historically, two other data models, the **network data model** and the **hierarchical data model**, preceded the relational data model. These models were tied closely to the underlying implementation, and complicated the task of modeling data. As a result they are little used now, except in old database code that is still in service in some places. They are outlined in Appendices A and B, for interested readers.

**1.8 Database Languages**

A database system provides a **data definition language** to specify the database schema and a **data manipulation language** to express database queries and updates. In practice, the data definition and data manipulation languages are not two separate languages; instead they simply form parts of a single database language, such as the widely used SQL language.

**1.9.1 Data-Definition Language**

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

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

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

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

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

We specify the storage structure and access methods used by the database system by a set of statements in a special type of DDL called a **data storage and definition** language. These statements define the implementation details of the database schemas, which are usually hidden from the users.

The data values stored in the database must satisfy certain **consistency constraints**. For example, suppose the balance on an account should not fall below $100. The DDL provides facilities to specify such constraints. The database systems check these constraints every time the database is updated.

### 1.9.2 Data-Manipulation Language

***Data manipulation is***

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

The modification of information stored in the database.

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

***There are basically two types:***

**Procedural DMLs** require a user to specify *what* data are needed and *how* to get those data.

**Declarative DMLs** (also referred to as **nonprocedural** DMLs) require a user to specify *what* data is needed *without* specifying how to get that data.

Declarative DMLs are usually easier to learn and use than are procedural DMLs. However, since a user does not have to specify how to get the data, the database system has to figure out an efficient means of accessing data. The DML component of the SQL language is nonprocedural.

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

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

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

The query specifies that those rows *from* the table *customer where* the *customer-id* is 192-83-7465 must be retrieved, and the *customer-name* attribute of these rows must be displayed.

Queries may involve information from more than one table. For instance, the following query finds the balance of all accounts owned by the customer with customerid
192-83- 7465.

> **select** *account.balance*
> **from** *depositor*, *account*
> **where** *depositor.customer-id* = 192-83-7465 **and**
> *depositor.account-number* = *account.account-number*

There are a number of database query languages in use, either commercially or experimentally.

The levels of abstraction apply not only to defining or structuring data, but also to manipulating data. At the physical level, we must define algorithms that allow efficient access to data. At higher levels of abstraction, we emphasize ease of use. The goal is to allow humans to interact efficiently with the system. The query processor component of the database system translates DML queries into sequences of actions at the physical level of the database system.

### 1.9 Data Dictionary

We can define a data dictionary as a DBMS component that stores the definition of data characteristics and relationships. You may recall that such "data about data" were labeled metadata. The DBMS data dictionary provides the DBMS with its self describing characteristic. In effect, the data dictionary resembles and X-ray of the company's entire data set, and is a crucial element in the data administration function.

The two main types of data dictionary exist, integrated and stand alone. An integrated data dictionary is included with the DBMS. For example, all relational DBMSs include a built in data dictionary or system catalog that is frequently accessed and updated by the RDBMS. Other DBMSs especially older types, do not have a built in data dictionary instead the DBA may use third party stand alone data dictionary systems.

Data dictionaries can also be classified as active or passive. An active data dictionary is automatically updated by the DBMS with every database access, thereby keeping its access information up-to-date. A passive data dictionary is not updated automatically and usually requires a batch process to be run. Data dictionary access information is normally used by the DBMS for query optimization purposes.

The data dictionary's main function is to store the description of all objects that interact with the database. Integrated data dictionaries tend to limit their metadata to the data managed by the DBMS. Stand alone data dictionary systems are more usually more flexible and allow the DBA to describe and manage all the organization's data, whether or not they are computerized. Whatever the data dictionary's format, its existence provides database designers and end users with a much improved ability to communicate. In addition, the data dictionary is the tool that helps the DBA to resolve data conflicts.

Although there is no standard format for the information stored in the data dictionary several features are common. For example, the data dictionary typically stores descriptions of all:

- Data elements that are defined in all tables of all databases. Specifically the data dictionary stores the name, data types, display formats, internal storage formats, and validation rules. The data dictionary tells where an element is used, by whom it is used and so on.
- Tables defined in all databases. For example, the data dictionary is likely to store the name of the table creator, the date of creation access authorizations, the number of columns, and so on.
- Indexes defined for each database table. For each index the DBMS stores at least the index name, the attributes used, the location, specific index characteristics and the creation date.
- Define databases: who created each database, the date of creation where the database is located, who the DBA is and so on.
- End users and The Administrators of the database
- Programs that access the database including screen formats, report formats, application formats, SQL queries and so on.
- Access authorization for all users of all databases.
- Relationships among data elements which are involved: whether the relationships are mandatory or optional, the connectivity and cardinality and so on.

If the data dictionary can be organized to include data external to the DBMS itself, it becomes an especially flexible to for more general corporate resource management. The management of such an extensive data dictionary, thus, makes it possible to manage the use and allocation of all of the organization information regardless whether it has its roots in the database data. This is why some managers consider the data dictionary to be the key element of the information resource management function. And this is also why the data dictionary might be described as the information resource dictionary.

The metadata stored in the data dictionary is often the basis for monitoring the database use and assignment of access rights to the database users. The information stored in the database is usually based on the relational table format, thus , enabling the DBA to query the database with SQL command. For example, SQL commands can be used to extract information about the users of the specific table or about the access rights of a particular user.

**1.10 Database Administrators and Database Users**

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 database users or database administrators.

**1.11.1 Database Users and User Interfaces**

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.

**Naive users** are unsophisticated users who interact with the system by invoking one of the application programs that have been written previously. For example, a bank teller who needs to transfer $50 from account *A* to account *B* invokes a program called *transfer*. This program asks the teller for the amount of money to be transferred, the account from which the money is to be transferred, and the account to which the money is to be transferred.

As another example, consider a user who wishes to find her account balance over the World Wide Web. Such a user may access a form, where she enters her account number. An application program at the Web server then retrieves the account balance, using the given account number, and passes this information back to the user. The typical user interface for naive users is a forms interface, where the user can fill in appropriate fields of the form. Naive users may also simply read *reports* generated from the database.

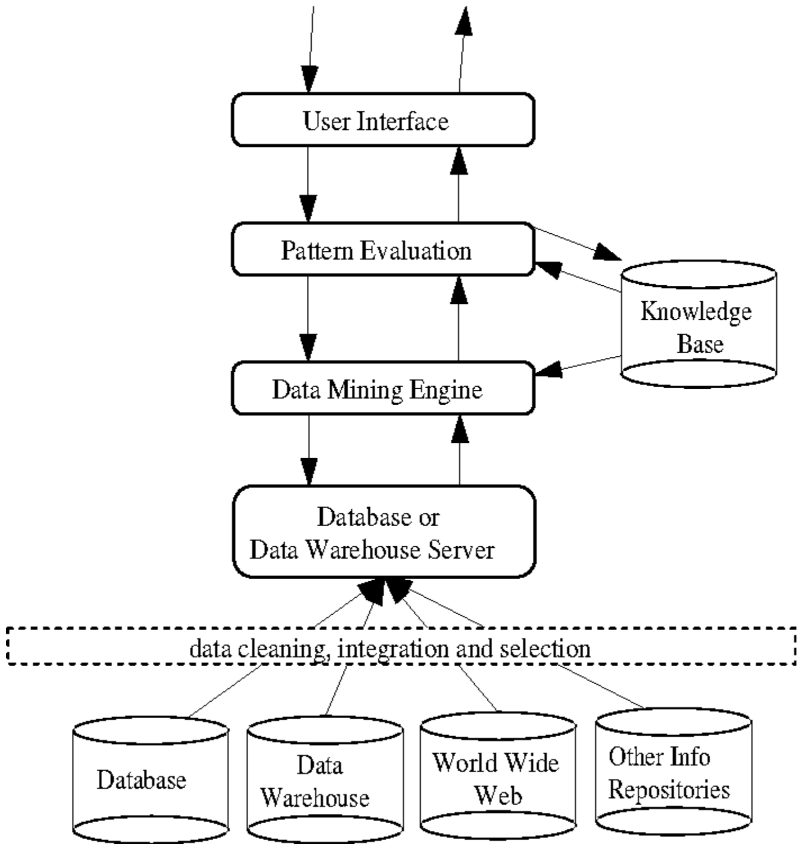
**Application programmers** are computer professionals who write application programs. Application programmers can choose from many tools to develop user interfaces. **Rapid application development (RAD)** tools are tools that enable an application programmer to construct forms and reports without writing a program. There are also special types of programming languages that combine imperative control structures (for example, for loops, while loops and if-then-else statements) with statements of the data manipulation language. These languages, sometimes called *fourth-generation languages*, often include special features to facilitate the generation of forms and the display of data on the screen. Most major commercial database systems include a fourth generation language. **Sophisticated users** interact with the system without writing programs. Instead, they form their requests in a database query language. They submit each such query to a **query processor**, whose function is to break down DML statements into instructions that the storage manager understands. Analysts who submit queries to explore data in the database fall in this category.

**Online analytical processing (OLAP)** tools simplify analysts' tasks by letting them view summaries of data in different ways. For instance, an analyst can see total sales by region (for example, North, South, East, and West), or by product, or by a combination of region and product (that is, total sales of each product in each region). The tools also permit the analyst to select specific regions, look at data in more detail (for example, sales by city within a region) or look at the data in less detail (for example, aggregate products together by category).

Another class of tools for analysts is **data mining** tools, which help them find certain kinds of patterns in data.

**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, knowledge base and expert systems, systems that store data with complex data types (for example, graphics data and audio data), and environment-modeling systems.


User Interface
Pattern Evaluation
Knowledge Base
Data Mining Engine
Database or Data Warehouse Server
data cleaning, integration and selection
Database
Data Warehouse
World Wide Web
Other Info Repositories

**1.11.2 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.

**1.11 DBMS Architecture and Data Independence**

Three important characteristics of the database approach are (1) insulation of programs and data (program-data and program-operation independence); (2) support of multiple user views; and (3) use of a catalog to store the database description (schema). In this section we specify an architecture for database systems, called the **three-schema architecture**, which was proposed to help achieve and visualize these characteristics. We then discuss the concept of data independence.

### 1.12.1 The Three-Schema Architecture

The goal of the three-schema architecture, illustrated in Figure 1.1, is to separate the user applications and 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. A high-level data model or an implementation data model can be used at this level.
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. A high-level data model or an implementation data model can be used at this level.

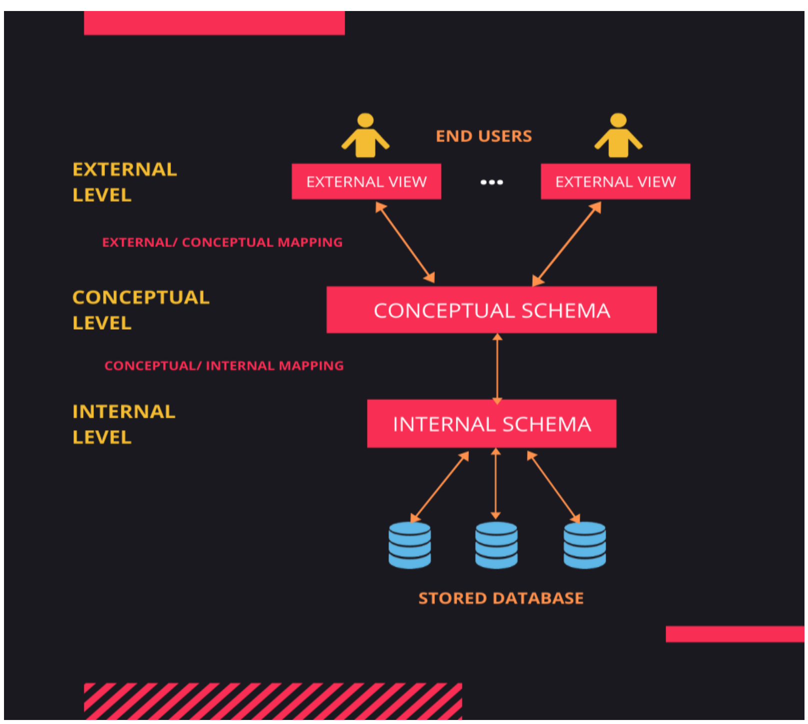



**Figure 1.1 The Three Schema Architecture**

The three-schema architecture is a convenient tool for the user to visualize the schema levels in a database system. Most DBMSs do not separate the three levels completely, but support the three-schema architecture to some extent. Some DBMSs may include physical-level details in the conceptual schema. In most DBMSs that support user views,
external schemas are specified in the same data model that describes the conceptual-level information. Some DBMSs allow different data models to be used at the conceptual and external levels.

Notice that the three schemas are only *descriptions* of data; the only data that *actually* exists is at the physical level. In a DBMS based on the three-schema architecture, each user group refers only 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, a certain amount of mapping is necessary to transform requests between the conceptual and internal levels.

### 1.12.2 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 schemas 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 schemas 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 schemas or application programs.

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

Whenever we have a multiple-level DBMS, its catalog must be expanded to include information on how to map requests and data among the various levels. The DBMS uses additional software to accomplish these mappings by referring to the mapping information in the catalog. Data independence is accomplished 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. Hence, application programs referring to the higher-level schema need not be changed.
The three-schema architecture can make it easier to achieve true data independence, both physical and logical. However, the two levels of mappings create an overhead during compilation or execution of a query or program, leading to inefficiencies in the DBMS. Because of this, few DBMSs have implemented the full three-schema architecture.

**1.12 Types of Database System**

Several criteria are normally used to classify DBMSs. The *first* is the data model on which the DBMS is based. The main data model used in many current commercial DBMSs is the relational data model. The object data model was implemented in some commercial systems but has not had widespread use. Many legacy (older) applications still run on database systems based on the hierarchical and network data models. The relational DBMSs are evolving continuously, and, in particular, have been incorporating many of the concepts that were developed in object databases. This has led to a new class of DBMSs called object-relational DBMSs. We can hence categorize DBMSs based on the *data model*: **relational, object, object-relational, hierarchical, network, and other**. The *second* criterion used to classify DBMSs is the number of users supported by the system. **Single-user systems** support only one user at a time and are mostly used with personal computers. **Multiuser systems**, which include the majority of DBMSs, support multiple users concurrently. A *third* criterion is the number of sites over which the database is distributed. A DBMS is centralized if the data is stored at a single computer site. A **centralized DBMS** can support multiple users, but the DBMS and the database themselves reside totally at a single computer site. A **distributed DBMS** (DDBMS) can have the actual database and DBMS software distributed over many sites, connected by a computer network. Homogeneous DDBMSs use the same DBMS software at multiple sites. A recent trend is to develop software to access several autonomous pre existing databases stored under heterogeneous llBMSs. This leads to a federated DBMS (or multidatabase system), in which the participating DBMSs are loosely coupled and have a degree of local autonomy. Many DBMSs use a client-server architecture.

### 1.13 Summary

In this chapter we have discussed in a relatively informal manner the major components of a database system. We summarise the discussion below:

A **database-management system** (DBMS) is a collection of interrelated data and a set of programs to access those data. This is a collection of related data with an implicit meaning and hence is a database.

A **datum** – a unit of data – is a symbol or a set of symbols which is used to represent something. This relationship between symbols and what they represent is the essence of what we mean by **information**.

**Knowledge** refers to the practical use of information.

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**.

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

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.

Underlying the structure of a database is the **data model**: a collection of conceptual tools for describing data, data relationships, data semantics, and consistency constraints.

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

**1.14 Key Words**

DBMS, Data Integrity, Data Persistence, Instances, Schemas, Physical Schema, Logical Schema, Data Model, DDL, DML, Data Dictionary

**1.15 Self Assessment Questions**

1. Why would you choose a database system instead of simply storing data in operating system files? When would it make sense *not* to use a database system?
2. What is logical data independence and why is it important?
3. Explain the difference between logical and physical data independence.
4. Explain the difference between external, internal, and conceptual schemas. How are these different schema layers related to the concepts of logical and physical data independence?
5. What are the responsibilities of a DBA?
6. Distinguish between logical and physical database design.
7. Describe and define the key properties of a database system. Give an organizational example of the benefits of each property.

**1.16 References/Suggested Readings**

1 http://www.microsoft-accesssolutions.co.uk

2 The Relational Database Dictionary, Extended Edition C. J. Date,Apress

3 Database Management Systems By **Raghu Ramakrishnan**