# Database Management System

**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 cannot 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 technique form the focus of this book. This chapter briefly introduces the principles of database systems.

# 1.2 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 largest 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.

# 1.3 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 systems, 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.

# 1.4 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:

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 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.

1. Difficulty in accessing data.

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.

1. **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.
2. **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.
3. **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.
4. **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. In such an environment, interaction of concurrent updates may result in inconsistent data. Consider bank account *A*, containing $500. If two customers withdraw funds (say $50 and $100 respectively) from account *A* at about the same time, the result of the concurrent executions may leave the account 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 $500, and write back $450 and $400, respectively. Depending on which one writes the value last, the account may contain either $450 or $400, rather than the correct value of $350. 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.
5. **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.

# 1.5 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 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"). **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 seperated 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 reflect 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 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.

## 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 be 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.

# 1.6 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 finetuning 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 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 itselfs 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.

## 1.7 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.Adatabase 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.

## 1.8 Database Management System Architecture

The computers have evolved from big mainframe computers to small desktop personal computers. The advances in computer and its architecture have also resulted in advances in database management system architecture.  
The architecture of databases of three kinds: —

1. Centralized Database management system Architecture
2. Client-Server DBMS Architecture
3. Distributed DBMS Architecture.

### Centralized Database Management System Architecture

Centralized Database Management System Architecture are the traditional database systems where all database functionality, data, the application program, and user interface processing are located on one machine. Access to the database from remote locations is via the communication links. In the early systems, the mainframe computers provided all the functionality required by the users of the computer.

The major features of the centralized database are:

* All processing is performed on the mainframe computer.
* The DBMS is centralized and stored on the mainframes.
* The centralized database is easy to manage.
* higher security as the possibility of attack is decreased to only one central database.

Some of the disadvantages of the centralized database are:

* centralized database leads to a longer time for data access
* failure of one centralized database leads to failure of the whole system
* high traffic on centralized database causes bottlenecks(jam)
* this leads to a decrease in the efficiency of the system

### Client-server DBMS Architecture

Client-Server systems involve a client machine and a server machine. Clients are PCs or workstations that have user interface capability and functionality for local processing. The client is connected to the server via a communication link.  
A client has interfaces to access and utilize the server resources. The client interacts with the server when it requires access to any additional functionality that does not exist in its own machine.

The Client-Server architecture has three components

1. The user interface programs,
2. The application programs that contain the application logic,
3. The DBMS that stores the data.

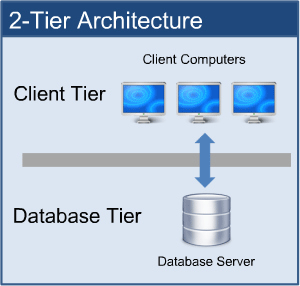
The request made by the user interface program is processed using the application logic which then accesses the database to retrieve the data.

The DBMS architecture on the Client-Server systems are of two kinds: —

* Two-tier Client-Server architecture
* Three-tier Client-Server architecture.

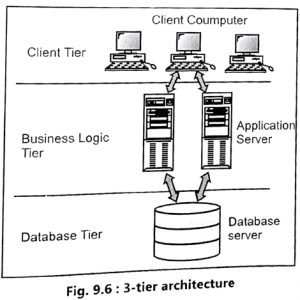
#### Two-tier Client-Server architecture

In the two-tier Client-Server database management system architecture, the user interface programs and the application programs run on the client-side. An Application Program Interface (API) allows client-side programs, to call the DBMS which is at the server-side. The client programs use Open Database Connectivity (ODBC) or Java Database Connectivity (JDBC) interfaces to communicate with the database.



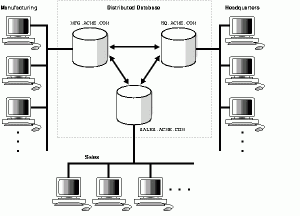
#### Three-tier Client-Server architecture

The three-tier Client-Server database management system architecture is commonly used for web applications. In addition to the client and the database servers, it has an intermediate layer or middleware called Application Server or Web Server. The web server stores the application or business logic part of the application. The DBMS is stored on the server-side.  
The client stores the user interface. The web server interacts with the client at one end, and with the server for the DBMS at the other end. The web server acts like a pipe for receiving the client request, processing it and accessing the data from the DBMS server, and sending it back to the client.



### Distributed DBMS

A distributed database is one with the decentralized functionality of the DBMS. It is distributed among a number of computers. The computers that store the components of the database are physically placed at different geographical locations; however, all the components are logically related. Access to the distributed databases is from remote locations via the communication links.



#### Advantages of Distributed Database

The following are the advantages of distributed databases.

* **Modular Development** − If the system needs to be expanded to new locations or new units, the work simply requires adding new computers and local data to the new site and finally connecting them to the distributed system, with no interruption in current functions.
* **More Reliable** − In case of database failures, when a component fails, the functioning of the system continues may be at a reduced performance. Hence DDBMS is more reliable.
* **Better Response** − If data is distributed in an efficient manner, then user requests can be met from local data itself, thus providing a faster response.
* **Lower Communication Cost** − In distributed database systems, if data is located locally where it is mostly used, then the communication costs for data manipulation can be minimized.

#### Disadvantages of the Distributed database:

* **Need for complex and expensive software** − DDBMS demands complex and often expensive software to provide data transparency and coordination across several sites.
* **Processing overhead** − Even simple operations may require a large number of communications and additional calculations to provide uniformity in data across the sites.
* **Data integrity** − The need for updating data in multiple sites pose problems of data integrity.
* **Overheads for improper data distribution** − Responsiveness of queries is largely dependent upon proper data distribution. Improper data distribution often leads to a very slow response to user requests
  1. 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.

### 1.9.1 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*, *customerstreet* 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.*

### 1.9.2 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 attributes, which are used as basic building blocks in the formation of various expressions that are used to derive a meaningful information. There can be any number of tuples in the table, but all the tuple 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.

### 1.9.3 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 feature of the object-oriented data model and relational data model. Semi-structured 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 semi-structured 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.

## 1.10 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.

### 1.10.1 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.

### 1.10.2 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 are needed *without* specifying how to get those 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 192-837465. **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.

## 1.11 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 purpose.

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 define in all tables of all databases. Specifically the data dictionary stores the name, datatypes, 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 define 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 define for each database tables. 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 data base
* 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 elements are involved: whether the relationship 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 bases 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 command can be used to extract information about the users of the specific table or about the access rights of a particular users.

## 1.12 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.

**1.12.1 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.

### 1.12.2 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.

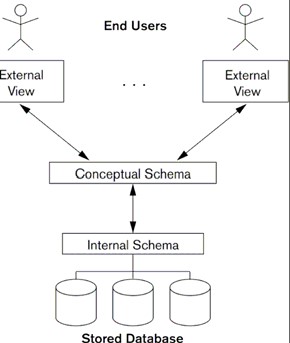
## 1.13 DBMS Architecture and Data Independence

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.

### 1.13.1 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 timeconsuming, 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.

### 1.14 History of Database Systems

Information processing drives the growth of computers, as it has from the earliest days of commercial computers. In fact, automation of data processing tasks predates computers. Punched cards, invented by Herman Hollerith, were used at the very beginning of the twentieth century to record U.S. census data, and mechanical systems were used to process the cards and tabulate results. Punched cards were later widely used as a means of entering data into computers. Techniques for data storage and processing have evolved over the years:

• 1950s and early 1960s: Magnetic tapes were developed for data storage. Dataprocessing tasks such as payroll were automated, with data stored on tapes. Processing of data consisted of reading data from one or more tapes and writing data to a new tape. Data could also be input from punched card decks and output to printers. For example, salary raises were processed by entering the raises on punched cards and reading the punched card deck in synchronization with a tape containing the master salary details. The records had to be in the same sorted order. The salary raises would be added to the salary read from the master tape and written to a new tape; the new tape would become the new master tape. Tapes (and card decks) could be read only sequentially, and data sizes were much larger than main memory; thus, data-processing programs were forced to process data in a particular order by reading and merging data from tapes and card decks.

• Late 1960s and early 1970s: Widespread use of hard disks in the late 1960s changed the scenario for data processing greatly, since hard disks allowed direct access to data. The position of data on disk was immaterial, since any location on disk could be accessed in just tens of milliseconds. Data were thus freed from the tyranny of sequentially. With the advent of disks, the network and hierarchical data models were developed, which allowed data structures such as lists and trees to be stored on disk. Programmers could construct and manipulate these data structures. A landmark paper by Edgar Codd in 1970 defined the relational model and nonprocedural ways of querying data in the relational model, and relational databases were born. The simplicity of the relational model and the possibility of hiding implementation details completely from the programmer were enticing indeed. Codd later won the prestigious Association of Computing Machinery Turing Award for his work.

• Late 1970s and 1980s: Although academically interesting, the relational model was not used in practice initially because of its perceived performance disadvantages; relational databases could not match the performance of existing network and hierarchical databases. That changed with System R, a groundbreaking project at IBM Research that developed techniques for the construction of an efficient relational database system. The fully functional System R prototype led to IBM’s first relational database product, SQL/DS. At the same time, the Ingres system was being developed at the University of California at Berkeley. It led to a commercial product of the same name. Also around this time, the first version of Oracle was released. Initial commercial relational database systems, such as IBM DB2, Oracle, Ingres, and DEC Rdb, played a major role in advancing techniques for efficient processing of declarative queries.

By the early 1980s, relational databases had become competitive with network

and hierarchical database systems even in the area of performance. Relational

databases were so easy to use that they eventually replaced network and hierarchical databases. Programmers using those older models were forced to deal with many low-level implementation details, and they had to code their queries in a procedural fashion. Most importantly, they had to keep efficiency in mind when

designing their programs, which involved a lot of effort. In contrast, in a relational

database, almost all these low-level tasks are carried out automatically by the database system, leaving the programmer free to work at a logical level. Since attaining dominance in the 1980s, the relational model has reigned supreme among data models.

The 1980s also saw much research on parallel and distributed databases, as well as initial work on object-oriented databases.

• 1990s: The SQL language was designed primarily for decision support applications, which are query-intensive, yet the mainstay of databases in the 1980s was transaction-processing applications, which are update-intensive.

In the early 1990s, decision support and querying re-emerged as a major application area for databases. Tools for analyzing large amounts of data saw a large growth in usage. Many database vendors introduced parallel database products in this period. Database vendors also began to add object-relational support to their databases.

The major event of the 1990s was the explosive growth of the World Wide Web. Databases were deployed much more extensively than ever before. Database systems now had to support very high transaction-processing rates, as well as very

high reliability and 24 × 7 availability (availability 24 hours a day, 7 days a week, meaning no downtime for scheduled maintenance activities). Database systems also had to support web interfaces to data.

• 2000s: The types of data stored in database systems evolved rapidly during this period. Semi-structured data became increasingly important. XML emerged as a data-exchange standard. JSON, a more compact data-exchange format well suited for storing objects from JavaScript or other programming languages subsequently grew increasingly important. Increasingly, such data were stored in relational database systems as support for the XML and JSON formats was added to the

major commercial systems. Spatial data (that is, data that include geographic information)

saw widespread use in navigation systems and advanced applications. Database systems added support for such data. Open-source database systems, notably PostgreSQL and MySQL saw increased use. “Auto-admin” features were added to database systems in order to allow automatic reconfiguration to adapt to changing workloads. This helped reduce the human workload in administering a database. Social network platforms grew at a rapid pace, creating a need to manage data about connections between people and their posted data, that did not fit well into a tabular row-and-column format. This led to the development of graph databases. In the latter part of the decade, the use of data analytics and data mining in enterprises became ubiquitous. Database systems were developed specifically to serve this market. These systems featured physical data organizations suitable for analytic processing, such as “column-stores,” in which tables are stored by column rather than the traditional row-oriented storage of the major commercial database systems. The huge volumes of data, as well as the fact that much of the data used for analytics was textual or semi-structured, led to the development of programming frameworks, such as *map-reduce*, to facilitate application programmers’ use of parallelism in analyzing data. In time, support for these features migrated into traditional database systems. Even in the late 2010s, debate continued in the database research community over the relative merits of a single database system serving both traditional transaction processing applications and the newer data-analysis applications versus maintaining separate systems for these roles. The variety of new data-intensive applications and the need for rapid development, particularly by startup firms, led to “NoSQL” systems that provide a lightweight form of data management. The name was derived from those systems’ lack of support for the ubiquitous database query language SQL, though the name is now often viewed as meaning “not only SQL.” The lack of a high-level query language based on the relational model gave programmers greater flexibility to work with new types of data. The lack of traditional database systems’ support for strict data consistency provided more flexibility in an application’s use of distributed data stores. The NoSQL model of “eventual consistency” allowed for distributed copies of data to be inconsistent as long they would eventually converge in the

absence of further updates.

• 2010s: The limitations of NoSQL systems, such as lack of support for consistency, and lack of support for declarative querying, were found acceptable by many applications (e.g., social networks), in return for the benefits they provided such as scalability and availability. However, by the early 2010s it was clear that the limitations made life significantly more complicated for programmers and database administrators. As a result, these systems evolved to provide features to support stricter notions of consistency, while continuing to support high scalability and availability. Additionally, these systems increasingly support higher levels of abstraction to avoid the need for programmers to have to reimplement features that are standard in a traditional database system.

Enterprises are increasingly outsourcing the storage and management of their data. Rather than maintaining in-house systems and expertise, enterprises may store their data in “cloud” services that host data for various clients in multiple, widely distributed server farms. Data are delivered to users via web-based services.

Other enterprises are outsourcing not only the storage of their data but also whole applications. In such cases, termed “software as a service,” the vendor not only stores the data for an enterprise but also runs (and maintains) the application software. These trends result in significant savings in costs, but they create new issues not only in responsibility for security breaches, but also in data ownership, particularly in cases where a government requests access to data.

The huge influence of data and data analytics in daily life has made the management of data a frequent aspect of the news. There is an unresolved tradeoff between an individual’s right of privacy and society’s need to know. Various national governments have put regulations on privacy in place. High-profile security breaches have created a public awareness of the challenges in cybersecurity and the risks of storing data.

### 1.15 Summary

• A database-management system (DBMS) consists of a collection of interrelated data and a collection of programs to access those data. The data describe one particular enterprise.

• The primary goal of a DBMS is to provide an environment that is both convenient and efficient for people to use in retrieving and storing information.

• Database systems are ubiquitous today, and most people interact, either directly or indirectly, with databases many times every day.

• Database systems are designed to store large bodies of information. The management of data involves both the definition of structures for the storage of information and the provision of mechanisms for the manipulation of information. In addition, the database system must provide for the safety of the information stored in the face of system crashes or attempts at unauthorized access. If data are to be shared among several users, the system must avoid possible anomalous results.

• 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.

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

• The relational data model is the most widely deployed model for storing data in databases. Other data models are the object-oriented model, the object-relational model, and semi-structured data models.

• A data-manipulation language (DML) is a language that enables users to access or manipulate data. Nonprocedural DMLs, which require a user to specify only what data are needed, without specifying exactly how to get those data, are widely used today.

• A data-definition language (DDL) is a language for specifying the database schema

and other properties of the data.

• Database design mainly involves the design of the database schema. The entity relationship (E-R) data model is a widely used model for database design. It provides a convenient graphical representation to view data, relationships, and constraints.

• A database system has several subsystems.

The storage manager subsystem provides the interface between the low-level data stored in the database and the application programs and queries submitted to the system.

The query processor subsystem compiles and executes DDL and DML statements.

• Transaction management ensures that the database remains in a consistent (correct) state despite system failures. The transaction manager ensures that concurrent transaction executions proceed without conflicts.

• The architecture of a database system is greatly influenced by the underlying computer

system on which the database system runs. Database systems can be centralized, or parallel, involving multiple machines. Distributed databases span multiple geographically separated machines.

• Database applications are typically broken up into a front-end part that runs at client machines and a part that runs at the backend. In two-tier architectures, the front end directly communicates with a database running at the back end. In three tier architectures, the back end part is itself broken up into an application server and a database server.

• 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.

• Data-analysis techniques attempt to automatically discover rules and patterns from

data. The field of data mining combines knowledge-discovery techniques invented

by artificial intelligence researchers and statistical analysts with efficient implementation

techniques that enable them to be used on extremely large databases.

### 1.16 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. This chapter has described several major advantages of a database system.What are two disadvantage?
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.
8. List four significant differences between a file-processing system and a DBMS?
9. Explain the concept of physical data independence and its importance in database systems.