# Data Base System

## Chapter 1

### Introduction to Database System

Data, Database, Database management system, Characteristics of the database approach, Role of Database administrators, Role of Database Designers, End Users, Advantages of Using a DBMS and When not to use a DBMS

**Data-Database system-Database management system(DBMS):**

**Data**, means 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. Nowadays, this data is typically stored in mobile phones, which have their own simple database software. This data can also be recorded in an indexed address book or stored on a hard drive, using a personal computer and software such as Microsoft Access or Excel.

A **database** is a collection of structured related data. Typically organized as “records” (traditionally, large numbers, on disk) and relationships between records

**Database management system(DBMS)** is the system for creating, manipulating and accessing the Database .**Defining** a database involves specifying the data types, structures, and constraints of the data to be stored in the database. The database definition or descriptive information is also stored by the DBMS in the form of a database catalog or dictionary; it is called meta-data. **Constructing** the database is the process of storing the data on some storage medium that is controlled by the DBMS. **Manipulating** a database includes functions such as querying the database to retrieve specific data, updating the database to reflect changes in the mini world, and generating reports from the data. **Sharing** a database allows multiple users and programs to access the database simultaneously.

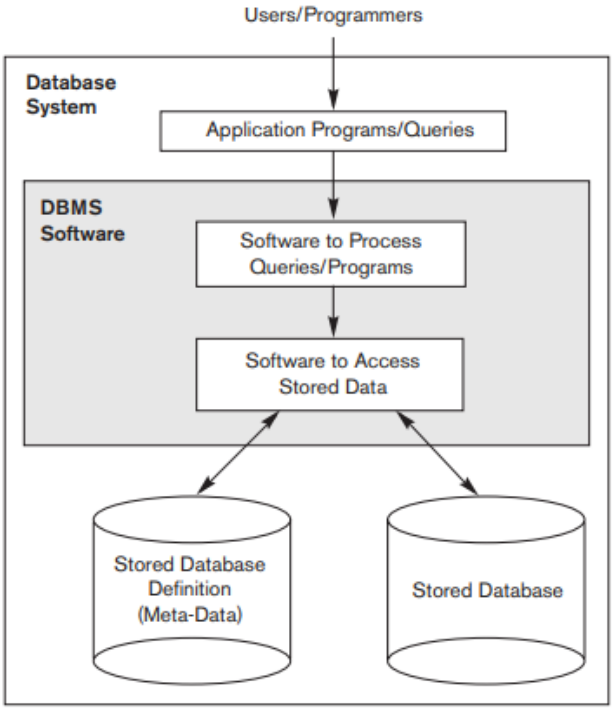


**Figure 1**

Figure 1 shows the database structure and a few sample data records. The database is organized as five files, each of which stores data records of the same type.

**STUDENT**

| Name | Student_number | Class | Major |
|---|---|---|---|
| Smith | 17 | 1 | CS |
| Brown | 8 | 2 | CS |

**COURSE**

| Course_name | Course_number | Credit_hours | Department |
|---|---|---|---|
| Intro to Computer Science | CS1310 | 4 | CS |
| Data Structures | CS3320 | 4 | CS |
| Discrete Mathematics | MATH2410 | 3 | MATH |
| Database | CS3380 | 3 | CS |

**SECTION**

| Section_identifier | Course_number | Semester | Year | Instructor |
|---|---|---|---|---|
| 85 | MATH2410 | Fall | 07 | King |
| 92 | CS1310 | Fall | 07 | Anderson |
| 102 | CS3320 | Spring | 08 | Knuth |
| 112 | MATH2410 | Fall | 08 | Chang |
| 119 | CS1310 | Fall | 08 | Anderson |
| 135 | CS3380 | Fall | 08 | Stone |

**GRADE_REPORT**

| Student_number | Section_identifier | Grade |
|---|---|---|
| 17 | 112 | B |
| 17 | 119 | C |
| 8 | 85 | A |

**Figure 2**

The STUDENT file stores data on each student, the COURSE file stores data on each course, the SECTION file stores data on each section of a course, the GRADE_REPORT file

stores the grades that students receive in the various sections they have completed, and the PREREQUISITE file stores the prerequisites of each course. To define this database, we must specify the structure of the records of each file by specifying the different types of data elements to be stored in each record. In Figure 2, each STUDENT record includes data to represent the student's Name, Student_number, Class (such as freshman or '1', sophomore or '2', and so forth), and Major (such as mathematics or 'MATH' and computer science or 'CS'); each COURSE record includes data to represent the Course_name, Course_number, Credit_hours, and Department (the department that offers the course), and so on. We must also specify a data type for each data element within a record. For example, we can specify that Name of STUDENT is a string of alphabetic characters, Student_number of STUDENT is an integer, and Grade of GRADE_REPORT is a single character from the set {'A', 'B', 'C', 'D', 'F', 'I'}.

## Characteristics of the database approach

The main characteristics of the database approach are :

- Self-describing nature of a database system
- Insulation between programs and data, and data abstraction
- Support of multiple views of the data
- Sharing of data and multiuser transaction processing

**Self-Describing Nature of a Database System**

A fundamental characteristic of the database approach is that the database system contains not only the database itself but also a complete definition or description of the database structure and constraints. This definition is stored in the DBMS catalog, which contains information such as the structure of each file, the type and storage format of each data item, and various constraints on the data. The information stored in the catalog is called **meta-data**, and it describes the structure of the primary database.

**RELATIONS**

| Relation_name | No_of_columns |
|---|---|
| STUDENT | 4 |
| COURSE | 4 |
| SECTION | 5 |
| GRADE_REPORT | 3 |
| PREREQUISITE | 2 |

**COLUMNS**

| Column_name | Data_type | Belongs_to_relation |
|---|---|---|
| Name | Character (30) | STUDENT |
| Student_number | Character (4) | STUDENT |
| Class | Integer (1) | STUDENT |
| Major | Major_type | STUDENT |
| Course_name | Character (10) | COURSE |
| Course_number | XXXXNNNN | COURSE |
| .... | .... | ..... |
| .... | .... | ..... |
| .... | .... | ..... |
| Prerequisite_number | XXXXNNNN | PREREQUISITE |

**Figure 3 An example of a database catalog for the database in Figure 2**

## Insulation between programs and data, and data abstraction

In traditional file processing, the structure of data files is embedded in the application programs, so any changes to the structure of a file may require changing all programs that access that file. By contrast, DBMS access programs do not require such changes in most cases. The structure of data files is stored in the DBMS catalog separately from the access programs. We call this property **program-data independence.**

For example, a file access program may be written in such a way that it can access only STUDENT records of the structure shown in Figure 4. If we want to add another piece of data to each STUDENT record, say the Birth_date, such a program will no longer work and must be changed. By contrast, in a DBMS environment, we only need to change the description of STUDENT records in the catalog (Figure 3) to reflect the inclusion of the new data item Birth_date; no programs are changed. The next time a DBMS program refers to the catalog, the new structure of STUDENT records will be accessed and used.

In some types of database systems, such as object-oriented and object-relational systems , users can define operations on data as part of the database definitions. An operation (also called a function or method) is specified in two parts. The interface (or signature) of an operation includes the operation name and the data types of its arguments (or parameters). The implementation (or method) of the operation is specified separately and can be changed without affecting the interface. User application programs can operate on the data by invoking these operations through their names and arguments, regardless of how the operations are implemented. This may be termed **program-operation independence.**

The characteristic that allows **program-data independence** and **program-operation independence** is called **data abstraction**. A DBMS provides users with a conceptual representation of data that does not include many of the details of how the data is stored or how the operations are implemented. Informally, a data model is a type of data abstraction that is used to provide this conceptual representation. The data model uses logical concepts, such as objects, their properties, and their interrelationships, that may be easier for most users to understand than computer storage concepts. Hence, the data model hides storage and implementation details that are not of interest to most database users.

Looking at the example in Figures 2 and 3, the internal implementation of the STUDENT file may be defined by its record length—the number of characters (bytes) in each record—and each data item may be specified by its starting byte within a record and its length in bytes. The STUDENT record would thus be represented as shown in Figure 4. But a typical database user is not concerned with the location of each data item within a record or its length; rather, the user is concerned that when a reference is made to Name of STUDENT, the correct value is returned. A conceptual representation of the STUDENT records is shown in Figure 2. Many other details of file storage organization—such as the access paths specified on a Figure 3.

| Data Item Name | Starting Position in Record | Length in Characters (bytes) |
|---|---|---|
| Name | 1 | 30 |
| Student_number | 31 | 4 |
| Class | 35 | 1 |
| Major | 36 | 4 |

**Figure 4 Internal storage format for a STUDENT record, based on the database catalog in Figure 3**

### Support of Multiple Views of the Data

A database typically has many types of users, each of whom may require a different perspective or view of the database. A view may be a subset of the database or it may contain virtual data that is derived from the database files but is not explicitly stored. Some users may not need to be aware of whether the data they refer to is stored or derived. A multiuser DBMS whose users have a variety of distinct applications must provide facilities for defining multiple views. For example, one user of the database of Figure 2 may be interested only in accessing and printing the transcript of each student; the view for this user is shown in Figure 5(a). A second user, who is interested only in checking that students have taken all the prerequisites of each course for which the student registers, may require the view shown in Figure 5(b).

(a)

**TRANSCRIPT**

| Student_name | Student_transcript | | | | |
|---|---|---|---|---|---|
| | Course_number | Grade | Semester | Year | Section_id |
| Smith | CS1310 | C | Fall | 08 | 119 |
| | MATH2410 | B | Fall | 08 | 112 |
| Brown | MATH2410 | A | Fall | 07 | 85 |
| | CS1310 | A | Fall | 07 | 92 |
| | CS3320 | B | Spring | 08 | 102 |
| | CS3380 | A | Fall | 08 | 135 |

(b)

**COURSE_PREREQUISITES**

| Course_name | Course_number | Prerequisites |
|---|---|---|
| Database | CS3380 | CS3320 |
| | | MATH2410 |
| Data Structures | CS3320 | CS1310 |

**Figure 5 Two views derived from the database in (a) The TRANSCRIPT view. (b) The COURSE_PREREQUISITES view.**

## Sharing of Data and Multiuser Transaction Processing

A multiuser DBMS, as its name implies, must allow multiple users to access the database at the same time. This is essential if data for multiple applications is to be integrated and maintained in a single database. The DBMS must include concurrency control software to ensure that several users trying to update the same data do so in a controlled manner so that the result of the updates is correct. For example, when several reservation agents try to assign a seat on an airline flight, the DBMS should ensure that each seat can be accessed by only one agent at a time for assignment to a passenger. These types of applications are generally called **online transaction processing (OLTP)** applications. A fundamental role of multiuser DBMS software is to ensure that concurrent transactions operate correctly and efficiently.

The concept of a **transaction** has become central to many database applications. A transaction is an executing program or process that includes one or more database accesses, such as reading or updating of database records. Each transaction is supposed to execute a logically correct database access if executed in its entirety without interference from other transactions. The DBMS must enforce several transaction properties. The **isolation** property ensures that each transaction appears to execute in isolation from other transactions, even though hundreds of transactions may be executing concurrently. The **atomicity** property ensures that either all the database operations in a transaction are executed or none are. The preceding characteristics are important in distinguishing a DBMS from traditional file-processing software.

## Database Administrators

In any organization where many people use the same resources, there is a need for a chief administrator to oversee and manage these resources. In a database environment, the primary resource is the database itself, and the secondary resource is the DBMS and related software. Administering these resources is the responsibility of the database administrator

(DBA). The DBA is responsible for authorizing access to the database, coordinating and monitoring its use, and acquiring software and hardware resources as needed. The DBA is accountable for problems such as security breaches and poor system response time. In large organizations, the DBA is assisted by a staff that carries out these functions.

**Database Designers**

Database designers are responsible for identifying the data to be stored in the database and for choosing appropriate structures to represent and store this data. These tasks are mostly undertaken before the database is actually implemented and populated with data. It is the responsibility of database designers to communicate with all prospective database users in order to understand their requirements and to create a design that meets these requirements. In many cases, the designers are on the staff of the DBA and may be assigned other staff responsibilities after the database design is completed. Database designers typically interact with each potential group of users and develop views of the database that meet the data and processing requirements of these groups. Each view is then analyzed and integrated with the views of other user groups. The final database design must be capable of supporting the requirements of all user groups

**End Users**

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

- **Casual end users** occasionally access the database, but they may need different information each time. They use a sophisticated database query interface to specify their requests and are typically middle- or high-level managers or other occasional browsers.
- **Naive or parametric end users** make up a sizable portion of database end users. Their main job function revolves around constantly querying and updating the database, using standard types of queries and updates— called **scanned transactions**—that have been carefully programmed and tested. Many of these tasks are now available as mobile apps for use with mobile devices. The tasks that such users perform are varied. A few examples are:
    1. Bank customers and tellers check account balances and post withdrawals and deposits.
    2. Reservation agents or customers for airlines, hotels, and car rental companies check availability for a given request and make reservations.
    3. Employees at receiving stations for shipping companies enter package identifications via bar codes and descriptive information through buttons to update a central database of received and in-transit packages.
    4. Social media users post and read items on social media Web sites.
- **Sophisticated end users** include engineers, scientists, business analysts, and others who thoroughly familiarize themselves with the facilities of the DBMS in order to implement their own applications to meet their complex requirements.

- **Standalone users** maintain personal databases by using ready-made program packages that provide easy-to-use menu-based or graphics-based interfaces. An example is the user of a financial software package that stores a variety of personal financial data.

## Advantages of Using the DBMS Approach

Some of the advantage are discussed as follows

### Controlling Redundancy

Redundancy in storing the same data multiple times leads to several problems. First, there is the need to perform a single logical update—such as entering data on a new student—multiple times: once for each file where student data is recorded. This leads to duplication of effort. Second, storage space is wasted when the same data is stored repeatedly, and this problem may be serious for large databases. Third, files that represent the same data may become inconsistent. This may happen because an update is applied to some of the files but not to others. Even if an update—such as adding a new student—is applied to all the appropriate files, the data concerning the student may still be inconsistent because the updates are applied independently by each user group. For example, one user group may enter a student's birth date erroneously as 'JAN-19-1988', whereas the other user groups may enter the correct value of 'JAN-29-1988'.

In the database approach, the views of different user groups are integrated during database design. Ideally, we should have a database design that stores each logical data item —such as a student's name or birth date—in only one place in the database. This is known as **data normalization,** and it ensures consistency and saves storage space . It is sometimes necessary to use **controlled redundancy** to improve the performance of queries. For example, we may store Student_name and Course_number redundantly in a GRADE_REPORT because whenever we retrieve a GRADE_REPORT record, we want to retrieve the student name and course number along with the grade, student number, and section identifier. By placing all the data together, we do not have to search multiple files to collect this data. This is known as **denormalization**

### Restricting Unauthorized Access

When multiple users share a large database, it is likely that most users will not be authorized to access all information in the database. For example, financial data such as salaries and bonuses is often considered confidential, and only authorized persons are allowed to access such data. In addition, some users may only be permitted to retrieve data, whereas others are allowed to retrieve and update. Hence, the type of access operation—retrieval or update—must also be controlled. Typically, users or user groups are given account numbers protected by passwords, which they can use to gain access to the database. A DBMS should provide a security and authorization subsystem, which the DBA uses to create

accounts and to specify account restrictions. Then, the DBMS should enforce these restrictions automatically. Notice that we can apply similar controls to the DBMS software. For example, only the DBA's staff may be allowed to use certain privileged software, such as the software for creating new accounts. Similarly, parametric users may be allowed to access the database only through the predefined apps or canned transactions developed for their use.

**Providing Persistent Storage for Program Objects**

Databases can be used to provide persistent storage for program objects and data structures. This is one of the main reasons for object-oriented database systems . Programming languages typically have complex data structures, such as structs or class definitions in C++ or Java. The values of program variables or objects are discarded once a program terminates, unless the programmer explicitly stores them in permanent files, which often involves converting these complex structures into a format suitable for file storage. When the need arises to read this data once more, the programmer must convert from the file format to the program variable or object structure. Object-oriented database systems are compatible with programming languages such as C++ and Java, and the DBMS software automatically performs any necessary conversions. Hence, a complex object in C++ can be stored permanently in an object-oriented DBMS. Such an object is said to be persistent, since it survives the termination of program execution and can later be directly retrieved by another program. The persistent storage of program objects and data structures is an important function of database systems. Traditional database systems often suffered from the so called impedance mismatch problem, since the data structures provided by the DBMS were incompatible with the programming language's data structures. Object-oriented database systems typically offer data structure compatibility with one or more object-oriented programming languages.

**Providing Storage Structures and Search Techniques for Efficient Query Processing**

Database systems must provide capabilities for efficiently executing queries and updates. Because the database is typically stored on disk, the DBMS must provide specialized data structures and search techniques to speed up disk search for the desired records. Auxiliary files called indexes are often used for this purpose. Indexes are typically based on tree data structures or hash data structures that are suitably modified for disk search. In order to process the database records needed by a particular query, those records must be copied from disk to main memory. Therefore, the DBMS often has a buffering or caching module that maintains parts of the database in main memory buffers. In general, the operating system is responsible for disk-to-memory buffering. However, because data buffering is crucial to the DBMS performance, most DBMSs do their own data buffering.

The query processing and optimization module of the DBMS is responsible for choosing an efficient query execution plan for each query based on the existing storage structures. The

choice of which indexes to create and maintain is part of physical database design and tuning, which is one of the responsibilities of the DBA staff.

**Providing Backup and Recovery**

A DBMS must provide facilities for recovering from hardware or software failures. The backup and recovery subsystem of the DBMS is responsible for recovery. For example, if the computer system fails in the middle of a complex update transaction, the recovery subsystem is responsible for making sure that the database is restored to the state it was in before the transaction started executing. Disk backup is also necessary in case of a catastrophic disk failure.

**Providing Multiple User Interfaces**

Because many types of users with varying levels of technical knowledge use a database, a DBMS should provide a variety of user interfaces. These include apps for mobile users, query languages for casual users, programming language interfaces for application programmers, forms and command codes for parametric users, and menu-driven interfaces and natural language interfaces for standalone users. Both forms-style interfaces and menu-driven interfaces are commonly known as **graphical user interfaces (GUIs).** Many specialized languages and environments exist for specifying GUIs. Capabilities for providing Web GUI interfaces to a database—or Web-enabling a database—are also quite common.

**Representing Complex Relationships among Data**

A database may include numerous varieties of data that are interrelated in many ways. Consider the example shown in Figure 2. The record for 'Brown' in the STUDENT file is related to four records in the GRADE_REPORT file. Similarly, each section record is related to one course record and to a number of GRADE_REPORT records—one for each student who completed that section. A DBMS must have the capability to represent a variety of complex relationships among the data, to define new relationships as they arise, and to retrieve and update related data easily and efficiently.

**Enforcing Integrity Constraints**

Most database applications have certain integrity constraints that must hold for the data. A DBMS should provide capabilities for defining and enforcing these constraints. The simplest type of **integrity constraint** involves specifying a data type for each data item. For example, in Figure 3, we specified that the value of the Class data item within each STUDENT record must be a one-digit integer and that the value of Name must be a string of no more than 30 alphabetic characters. To restrict the value of Class between 1 and 5 would be an additional constraint that is not shown in the current catalog. A more complex type of constraint that frequently occurs involves specifying that a record in one file must be related to records in other files. For example, Figure 2 can specify that every section record must be related to a course record. This is known as a **referential integrity constraint.** Another type of

constraint specifies uniqueness on data item values, such as every course record must have a unique value for Course_number. This is known as a **key or uniqueness constraint.** These constraints are derived from the meaning or semantics of the data and of the mini world it represents. It is the responsibility of the database designers to identify integrity constraints during database design.

**Permitting Inferencing and Actions Using Rules and Triggers**

Some database systems provide capabilities for defining deduction rules for inferencing new information from the stored database facts. Such systems are called deductive database systems. For example, there may be complex rules in the mini world application for determining when a student is on probation. These can be specified declaratively as rules, which when compiled and maintained by the DBMS can determine all students on probation. In a traditional DBMS, an explicit procedural program code would have to be written to support such applications. But if the mini world rules change, it is generally more convenient to change the declared deduction rules than to recode procedural programs. In today's relational database systems, it is possible to associate **triggers** with tables. A trigger is a form of a rule activated by updates to the table, which results in performing some additional operations to some other tables, sending messages, and so on. More involved procedures to enforce rules are popularly called stored procedures; they become a part of the overall database definition and are invoked appropriately when certain conditions are met. More powerful functionality is provided by active database systems, which provide active rules that can automatically initiate actions when certain events and conditions occur.

**Additional Implications of Using the Database Approach**

This section discusses a few additional implication

- Potential for Enforcing Standards
- Reduced Application Development Time.
- Flexibility
- Availability of Up-to-Date Information
- Economies of Scale

**When Not to Use a DBMS**

In spite of the advantages of using a DBMS, there are a few situations in which a DBMS may involve unnecessary overhead costs that would not be incurred in traditional file processing. The overhead costs of using a DBMS are due to the following:

- High initial investment in hardware, software, and training
- The generality that a DBMS provides for defining and processing data
- Overhead for providing security, concurrency control, recovery, and integrity functions

Therefore, it may be more desirable to develop customized database applications under the following circumstances:

- Simple, well-defined database applications that are not expected to change at all
- Stringent, real-time requirements for some application programs that may not be met because of DBMS overhead
- Embedded systems with limited storage capacity, where a general-purpose DBMS would not fit
- No multiple-user access to data

Certain industries and applications have elected not to use general-purpose DBMSs. For example, many computer-aided design (CAD) tools used by mechanical and civil engineers have proprietary file and data management software that is geared for the internal manipulations of drawings and 3D objects. Similarly, communication and switching systems designed by companies like AT&T were early manifestations of database software that was made to run very fast with hierarchically organized data for quick access and routing of calls. GIS implementations often implement their own data organization schemes for efficiently implementing functions related to processing maps, physical contours, lines, polygons, and so on.