GLOBAL EDITION

For these Global Editions, the editorial team at Pearson has collaborated with educators across the world to address a wide range of subjects and requirements, equipping students with the best possible learning tools. This Global Edition preserves the cutting-edge approach and pedagogy of the original, but also features alterations, customization and adaptation from the North American version.

This is a special edition of an established title widely used by colleges and universities throughout the world. Pearson published this exclusive edition for the benefit of students outside the United States and Canada. If you purchased this book within the United States or Canada you should be aware that it has been imported without the approval of the Publisher or Author.

Pearson Global Edition
Chapter Objectives

In this chapter you will learn:

• The importance of database systems.
• Some common uses of database systems.
• The characteristics of file-based systems.
• The problems with the file-based approach.
• The meaning of the term “database.”
• The meaning of the term “database management system” (DBMS).
• The typical functions of a DBMS.
• The major components of the DBMS environment.
• The personnel involved in the DBMS environment.
• The history of the development of DBMSs.
• The advantages and disadvantages of DBMSs.

While barely 50 years old, database research has had a profound impact on the economy and society, creating an industry sector valued at between US$35-US$50 billion annually. The database system, the subject of this book, is arguably the most important development in the field of software engineering, and the database is now the underlying framework of the information system, fundamentally changing the way that many organizations operate. Indeed the importance of the database system has increased in recent years with significant developments in hardware capability, hardware capacity, and communications, including the emergence of the Internet, electronic commerce, business intelligence, mobile communications, and grid computing.

Database technology has been an exciting area to work in and, since its emergence, has been the catalyst for many important developments in software engineering. Database research is not over and there are still many problems that need
to be addressed. Moreover, as the applications of database systems become even more complex we will have to rethink many of the algorithms currently being used, such as the algorithms for file storage, file access, and query optimization. These original algorithms have made significant contributions in software engineering and, without doubt, the development of new algorithms will have similar effects. In this first chapter, we introduce the database system.

“As a result of its importance, many computing and business courses cover the study of database systems. In this book, we explore a range of issues associated with the implementation and applications of database systems. We also focus on how to design a database and present a methodology that should help you design both simply and complex databases.

Structure of this Chapter In Section 1.1, we examine some uses of database systems that we find in everyday life but are not necessarily aware of. In Sections 1.2 and 1.3, we compare the early file-based approach to computerizing the manual file system with the modern, and more usable, database approach. In Section 1.4, we discuss the four types of role that people perform in the database environment, namely: data and database administrators, database designers, application developers, and end-users. In Section 1.5, we provide a brief history of database systems, and follow that in Section 1.6 with a discussion of the advantages and disadvantages of database systems.

Throughout this book, we illustrate concepts using a case study based on a fictitious property management company called DreamHome. We provide a detailed description of this case study in Section 11.4 and Appendix A. In Appendix B, we present further case studies that are intended to provide additional realistic projects for the reader. There will be exercises based on these case studies at the end of many chapters.

1.1 Introduction

The database is now such an integral part of our day-to-day life that often we are not aware that we are using one. To start our discussion of databases, in this section we examine some applications of database systems. For the purposes of this discussion, we consider a database to be a collection of related data and a database management system (DBMS) to be the software that manages and controls access to the database. A database application is simply a program that interacts with the database at some point in its execution. We also use the more inclusive term database system as a collection of application programs that interact with the database along with the DBMS and the database itself. We provide more accurate definitions in Section 1.3.
1.1 Introduction

**Purchases from the supermarket**
When you purchase goods from your local supermarket, it is likely that a database is accessed. The checkout assistant uses a bar code reader to scan each of your purchases. This reader is linked to a database application that uses the bar code to find out the price of the item from a product database. The application then reduces the number of such items in stock and displays the price on the cash register. If the reorder level falls below a specified threshold, the database system may automatically place an order to obtain more of that item. If a customer telephones the supermarket, an assistant can check whether an item is in stock by running an application program that determines availability from the database.

**Purchases using your credit card**
When you purchase goods using your credit card, the assistant normally checks whether you have sufficient credit left to make the purchase. This check may be carried out by telephone or automatically by a card reader linked to a computer system. In either case, there is a database somewhere that contains information about the purchases that you have made using your credit card. To check your credit, there is a database application that uses your credit card number to check that the price of the goods you wish to buy, together with the sum of the purchases that you have already made this month, is within your credit limit. When the purchase is confirmed, the details of the purchase are added to this database. The database application also accesses the database to confirm that the credit card is not on the list of stolen or lost cards before authorizing the purchase. There are other database applications to send out monthly statements to each cardholder and to credit accounts when payment is received.

**Booking a vacation with a travel agent**
When you make inquiries about a vacation, your travel agent may access several databases containing vacation and flight details. When you book your vacation, the database system has to make all the necessary booking arrangements. In this case, the system has to ensure that two different agents do not book the same vacation or overbook the seats on the flight. For example, if there is only one seat left on the flight from New York to London and two agents try to reserve the last seat at the same time, the system has to recognize this situation, allow one booking to proceed, and inform the other agent that there are now no seats available. The travel agent may have another, usually separate, database for invoicing.

**Using the local library**
Your local library probably has a database containing details of the books in the library, details of the readers, reservations, and so on. There will be a computerized index that allows readers to find a book based on its title, authors, or subject area. The database system handles reservations to allow a reader to reserve a book and to be informed by mail or email when the book is available. The system also sends reminders to borrowers who have failed to return books by the due date. Typically, the system will have a bar code reader, similar to that used by the
supermarket described earlier, that is used to keep track of books coming in and going out of the library.

**Taking out insurance**
Whenever you wish to take out insurance—for example personal insurance, property insurance, or auto insurance, your agent may access several databases containing figures for various insurance organizations. The personal details that you supply, such as name, address, age, and whether you drink or smoke, are used by the database system to determine the cost of the insurance. An insurance agent can search several databases to find the organization that gives you the best deal.

**Renting a DVD**
When you wish to rent a DVD from a DVD rental company, you will probably find that the company maintains a database consisting of the DVD titles that it stocks, details on the copies it has for each title, whether the copy is available for rent or is currently on loan, details of its members (the renters), and which DVDs they are currently renting and date they are returned. The database may even store more detailed information on each DVD, such as its director and its actors. The company can use this information to monitor stock usage and predict future buying trends based on historic rental data.

**Using the Internet**
Many of the sites on the Internet are driven by database applications. For example, you may visit an online bookstore that allows you to browse and buy books, such as Amazon.com. The bookstore allows you to browse books in different categories, such as computing or management, or by author name. In either case, there is a database on the organization’s Web server that consists of book details, availability, shipping information, stock levels, and order history. Book details include book titles, ISBNs, authors, prices, sales histories, publishers, reviews, and detailed descriptions. The database allows books to be cross-referenced: for example, a book may be listed under several categories, such as computing, programming languages, bestsellers, and recommended titles. The cross-referencing also allows Amazon to give you information on other books that are typically ordered along with the title you are interested in.

As with an earlier example, you can provide your credit card details to purchase one or more books online. Amazon.com personalizes its service for customers who return to its site by keeping a record of all previous transactions, including items purchased, shipping, and credit card details. When you return to the site, you might be greeted by name and presented with a list of recommended titles based on previous purchases.

**Studying at College**
If you are at college, there will be a database system containing information about yourself, your major and minor fields, the courses you are enrolled in, details about your financial aid, the classes you have taken in previous years or are taking
this year, and details of all your examination results. There may also be a database containing details relating to the next year’s admissions and a database containing details of the staff working at the university, giving personal details and salary-related details for the payroll office.

These are only a few of the applications for database systems, and you will no doubt know of plenty of others. Though we now take such applications for granted, the database system is a highly complex technology that has been developed over the past 40 years. In the next section, we discuss the precursor to the database system: the file-based system.

### 1.2 Traditional File-Based Systems

It is almost a tradition that comprehensive database books introduce the database system with a review of its predecessor, the file-based system. We will not depart from this tradition. Although the file-based approach is largely obsolete, there are good reasons for studying it:

- Understanding the problems inherent in file-based systems may prevent us from repeating these problems in database systems. In other words, we should learn from our earlier mistakes. Actually, using the word “mistakes” is derogatory and does not give any cognizance to the work that served a useful purpose for many years. However, we have learned from this work that there are better ways to handle data.

- If you wish to convert a file-based system to a database system, understanding how the file system works will be extremely useful, if not essential.

#### 1.2.1 File-Based Approach

**File-based system**

A collection of application programs that perform services for the end-users, such as the production of reports. Each program defines and manages its own data.

File-based systems were an early attempt to computerize the manual filing system that we are all familiar with. For example, an organization might have physical files set up to hold all external and internal correspondence relating to a project, product, task, client, or employee. Typically, there are many such files, and for safety they are labeled and stored in one or more cabinets. For security, the cabinets may have locks or may be located in secure areas of the building. In our own home, we probably have some sort of filing system that contains receipts, warranties, invoices, bank statements, and so on. When we need to look something up, we go to the filing system and search through the system, starting at the first entry, until we find what we want. Alternatively, we might have an indexing system that helps locate what we want more quickly. For example, we might have divisions in the filing system or separate folders for different types of items that are in some way *logically related*.

The manual filing system works well as long as the number of items to be stored is small. It even works adequately when there are large numbers of items and we only have to store and retrieve them. However, the manual filing system breaks down when we have to cross-reference or process the information in the files. For
example, a typical real estate agent’s office might have a separate file for each property for sale or rent, each potential buyer and renter, and each member of staff. Consider the effort that would be required to answer the following questions:

- What three-bedroom properties do you have for sale with an acre of land and a garage?
- What apartments do you have for rent within three miles of downtown?
- What is the average rent for a two-bedroom apartment?
- What is the annual total for staff salaries?
- How does last month’s net income compare with the projected figure for this month?
- What is the expected monthly net income for the next financial year?

Increasingly nowadays, clients, senior managers, and staff want more and more information. In some business sectors, there is a legal requirement to produce detailed monthly, quarterly, and annual reports. Clearly, the manual system is inadequate for this type of work. The file-based system was developed in response to the needs of industry for more efficient data access. However, rather than establish a centralized store for the organization’s operational data, a decentralized approach was taken, where each department, with the assistance of Data Processing (DP) staff, stored and controlled its own data. To understand what this means, consider the DreamHome example.

The Sales Department is responsible for the selling and renting of properties. For example, whenever a client who wishes to offer his or her property as a rental approaches the Sales Department, a form similar to the one shown in Figure 1.1(a) is completed. The completed form contains details on the property, such as address, number of rooms, and the owner’s contact information. The Sales Department also handles inquiries from clients, and a form similar to the one shown in Figure 1.1(b) is completed for each one. With the assistance of the DP Department, the Sales Department creates an information system to handle the renting of property. The system consists of three files containing property, owner, and client details, as illustrated in Figure 1.2. For simplicity, we omit details relating to members of staff, branch offices, and business owners.

The Contracts Department is responsible for handling the lease agreements associated with properties for rent. Whenever a client agrees to rent a property, a form with the client and property details is filled in by one of the sales staff, as shown in Figure 1.3. This form is passed to the Contracts Department, which allocates a lease number and completes the payment and rental period details. Again, with the assistance of the DP Department, the Contracts Department creates an information system to handle lease agreements. The system consists of three files that store lease, property, and client details, and that contain similar data to that held by the Sales Department, as illustrated in Figure 1.4.

The process is illustrated in Figure 1.5. It shows each department accessing their own files through application programs written especially for them. Each set of departmental application programs handles data entry, file maintenance, and the generation of a fixed set of specific reports. More important, the physical structure and storage of the data files and records are defined in the application code.
Figure 1.1  Sales Department forms; (a) Property for Rent Details form; (b) Client Details form.
Chapter 1  Introduction to Databases

### PropertyForRent

<table>
<thead>
<tr>
<th>propertyNo</th>
<th>street</th>
<th>city</th>
<th>postcode</th>
<th>type</th>
<th>rooms</th>
<th>rent</th>
<th>ownerNo</th>
</tr>
</thead>
<tbody>
<tr>
<td>PA14</td>
<td>16 Holhead Rd</td>
<td>Aberdeen</td>
<td>AB7 5SU</td>
<td>House</td>
<td>6</td>
<td>650</td>
<td>CO46</td>
</tr>
<tr>
<td>PL94</td>
<td>6 Argyll St</td>
<td>London</td>
<td>NW2</td>
<td>Flat</td>
<td>4</td>
<td>400</td>
<td>CO87</td>
</tr>
<tr>
<td>PG4</td>
<td>6 Lawrence St</td>
<td>Glasgow</td>
<td>G11 9QX</td>
<td>Flat</td>
<td>3</td>
<td>350</td>
<td>CO40</td>
</tr>
<tr>
<td>PG36</td>
<td>2 Manor Rd</td>
<td>Glasgow</td>
<td>G32 4QX</td>
<td>Flat</td>
<td>3</td>
<td>375</td>
<td>CO93</td>
</tr>
<tr>
<td>PG21</td>
<td>18 Dale Rd</td>
<td>Glasgow</td>
<td>G12</td>
<td>House</td>
<td>5</td>
<td>600</td>
<td>CO87</td>
</tr>
<tr>
<td>PG16</td>
<td>5 Novar Dr</td>
<td>Glasgow</td>
<td>G12 9AX</td>
<td>Flat</td>
<td>4</td>
<td>450</td>
<td>CO93</td>
</tr>
</tbody>
</table>

### PrivateOwner

<table>
<thead>
<tr>
<th>ownerNo</th>
<th>fName</th>
<th>lName</th>
<th>address</th>
<th>telNo</th>
</tr>
</thead>
<tbody>
<tr>
<td>CO46</td>
<td>Joe</td>
<td>Keogh</td>
<td>2 Fergus Dr, Aberdeen AB2 7SX</td>
<td>01224-861212</td>
</tr>
<tr>
<td>CO87</td>
<td>Carol</td>
<td>Farrel</td>
<td>6 Elder St, Glasgow G32 9DX</td>
<td>0141-357-7419</td>
</tr>
<tr>
<td>CO40</td>
<td>Tina</td>
<td>Murphy</td>
<td>63 Well St, Glasgow G42</td>
<td>0141-943-1728</td>
</tr>
<tr>
<td>CO93</td>
<td>Tony</td>
<td>Shaw</td>
<td>12 Park Pl, Glasgow G4 0QR</td>
<td>0141-225-7025</td>
</tr>
</tbody>
</table>

### Client

<table>
<thead>
<tr>
<th>clientNo</th>
<th>fName</th>
<th>lName</th>
<th>address</th>
<th>telNo</th>
<th>prefType</th>
<th>maxRent</th>
</tr>
</thead>
<tbody>
<tr>
<td>CR76</td>
<td>John</td>
<td>Kay</td>
<td>56 High St, London SW1 4EH</td>
<td>0207-774-5632</td>
<td>Flat</td>
<td>425</td>
</tr>
<tr>
<td>CR56</td>
<td>Aline</td>
<td>Stewart</td>
<td>64 Fern Dr, Glasgow G42 0BL</td>
<td>0141-848-1825</td>
<td>Flat</td>
<td>350</td>
</tr>
<tr>
<td>CR74</td>
<td>Mike</td>
<td>Ritchie</td>
<td>10 Tain St, PA1G 1YQ</td>
<td>01475-392178</td>
<td>House</td>
<td>750</td>
</tr>
<tr>
<td>CR62</td>
<td>Mary</td>
<td>Tregear</td>
<td>5 Tarbit Rd, Aberdeen AB9 3ST</td>
<td>01224-196720</td>
<td>Flat</td>
<td>600</td>
</tr>
</tbody>
</table>

**Figure 1.2** The PropertyForRent, PrivateOwner, and Client files used by Sales.

**Figure 1.3**

Lease Details form used by Contracts Department.

---

**DreamHome**

**Lease Details**

**Lease Number:** 10012

---

**Payment Details**

**Monthly Rent:** 600

**Payment Method:** Cheque

**Deposit:** 1200 Paid (Y or N) Y

---

**Payment Details**

**Rent Start Date:** 1-Jul-15

**Rent Finish Date:** 30-Jun-14

**Duration:** 1 Year
1.2 Traditional File-Based Systems

We can find similar examples in other departments. For example, the Payroll Department stores details relating to each member of staff’s salary:

\[ \text{StaffSalary(staffNo, fName, lName, sex, salary, branchNo)} \]

The Human Resources (HR) Department also stores staff details:

\[ \text{Staff(staffNo, fName, lName, position, sex, dateOfBirth, salary, branchNo)} \]
It can be seen quite clearly that there is a significant amount of duplication of data in these departments, and this is generally true of file-based systems. Before we discuss the limitations of this approach, it may be useful to understand the terminology used in file-based systems. A file is simply a collection of records, which contains logically related data. For example, the PropertyForRent file in Figure 1.2 contains six records, one for each property. Each record contains a logically connected set of one or more fields, where each field represents some characteristic of the real-world object that is being modeled. In Figure 1.2, the fields of the PropertyForRent file represent characteristics of properties, such as address, property type, and number of rooms.

### 1.2.2 Limitations of the File-Based Approach

This brief description of traditional file-based systems should be sufficient to discuss the limitations of this approach. We list five problems in Table 1.1.

#### Separation and isolation of data

When data is isolated in separate files, it is more difficult to access data that should be available. For example, if we want to produce a list of all houses that match the requirements of clients, we first need to create a temporary file of those clients who have “house” as the preferred type. We then search the PropertyForRent file for those properties where the property type is “house” and the rent is less than the client’s maximum rent. With file systems, such processing is difficult. The application developer must synchronize the processing of two files to ensure that the correct data is extracted. This difficulty is compounded if we require data from more than two files.

#### Duplication of data

Owing to the decentralized approach taken by each department, the file-based approach encouraged, if not necessitated, the uncontrolled duplication of data. For example, in Figure 1.5 we can clearly see that there is duplication of both property and client details in the Sales and Contracts Departments. Uncontrolled duplication of data is undesirable for several reasons, including:

- Duplication is wasteful. It costs time and money to enter the data more than once.
- It takes up additional storage space, again with associated costs. Often, the duplication of data can be avoided by sharing data files.

<table>
<thead>
<tr>
<th>TABLE 1.1</th>
<th>Limitations of file-based systems.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Separation and isolation of data</td>
<td></td>
</tr>
<tr>
<td>Duplication of data</td>
<td></td>
</tr>
<tr>
<td>Data dependence</td>
<td></td>
</tr>
<tr>
<td>Incompatible file formats</td>
<td></td>
</tr>
<tr>
<td>Fixed queries/proliferation of application programs</td>
<td></td>
</tr>
</tbody>
</table>
Perhaps more importantly, duplication can lead to loss of data integrity; in other words, the data is no longer consistent. For example, consider the duplication of data between the Payroll and HR Departments described previously. If a member of staff moves and the change of address is communicated only to HR and not to Payroll, the person’s paycheck will be sent to the wrong address. A more serious problem occurs if an employee is promoted with an associated increase in salary. Again, let’s assume that the change is announced to HR, but the change does not filter through to Payroll. Now, the employee is receiving the wrong salary. When this error is detected, it will take time and effort to resolve. Both these examples illustrate inconsistencies that may result from the duplication of data. As there is no automatic way for HR to update the data in the Payroll files, it is not difficult to foresee such inconsistencies arising. Even if Payroll is notified of the changes, it is possible that the data will be entered incorrectly.

**Data dependence**

As we have already mentioned, the physical structure and storage of the data files and records are defined in the application code. This means that changes to an existing structure are difficult to make. For example, increasing the size of the `PropertyForRent` address field from 40 to 41 characters sounds like a simple change, but it requires the creation of a one-off program (that is, a program that is run only once and can then be discarded) that converts the `PropertyForRent` file to the new format. This program has to:

- Open the original `PropertyForRent` file for reading
- Open a temporary file with the new structure
- Read a record from the original file, convert the data to conform to the new structure, and write it to the temporary file, then repeat this step for all records in the original file
- Delete the original `PropertyForRent` file
- Rename the temporary file as `PropertyForRent`

In addition, all programs that access the `PropertyForRent` file must be modified to conform to the new file structure. There might be many such programs that access the `PropertyForRent` file. Thus, the programmer needs to identify all the affected programs, modify them, and then retest them. Note that a program does not even have to use the address field to be affected: it only has to use the `PropertyForRent` file. Clearly, this process could be very time-consuming and subject to error. This characteristic of file-based systems is known as **program–data dependence**.

**Incompatible file formats**

Because the structure of files is embedded in the application programs, the structures are dependent on the application programming language. For example, the structure of a file generated by a COBOL program may be different from the structure of a file generated by a C program. The direct incompatibility of such files makes them difficult to process jointly.

For example, suppose that the Contracts Department wants to find the names and addresses of all owners whose property is currently rented out. Unfortunately,
Contracts does not hold the details of property owners; only the Sales Department holds these. However, Contracts has the property number (propertyNo), which can be used to find the corresponding property number in the Sales Department’s PropertyForRent file. This file holds the owner number (ownerNo), which can be used to find the owner details in the PrivateOwner file. The Contracts Department programs in COBOL and the Sales Department programs in C. Therefore, matching propertyNo fields in the two PropertyForRent files requires that an application developer write software to convert the files to some common format to facilitate processing. Again, this process can be time-consuming and expensive.

**Fixed queries/proliferation of application programs**

From the end-user’s point of view, file-based systems were a great improvement over manual systems. Consequently, the requirement for new or modified queries grew. However, file-based systems are very dependent upon the application developer, who has to write any queries or reports that are required. As a result, two things happened. In some organizations, the type of query or report that could be produced was fixed. There was no facility for asking unplanned (that is, spur-of-the-moment or *ad hoc*) queries either about the data itself or about which types of data were available.

In other organizations, there was a proliferation of files and application programs. Eventually, this reached a point where the DP Department, with its existing resources, could not handle all the work. This put tremendous pressure on the DP staff, resulting in programs that were inadequate or inefficient in meeting the demands of the users, limited documentation, and difficult maintenance. Often, certain types of functionality were omitted:

- There was no provision for security or integrity.
- Recovery, in the event of a hardware or software failure, was limited or nonexistent.
- Access to the files was restricted to one user at a time—there was no provision for shared access by staff in the same department.

In either case, the outcome was unacceptable. Another solution was required.

### 1.3 Database Approach

All of the previously mentioned limitations of the file-based approach can be attributed to two factors:

1. The definition of the data is embedded in the application programs, rather than being stored separately and independently.
2. There is no control over the access and manipulation of data beyond that imposed by the application programs.

To become more effective, a new approach was required. What emerged were the database and the Database Management System (DBMS). In this section, we provide a more formal definition of these terms, and examine the components that we might expect in a DBMS environment.
1.3 Database Approach

1.3.1 The Database

A shared collection of logically related data and its description, designed to meet the information needs of an organization.

We now examine the definition of a database so that you can understand the concept fully. The database is a single, possibly large repository of data that can be used simultaneously by many departments and users. Instead of disconnected files with redundant data, all data items are integrated with a minimum amount of duplication. The database is no longer owned by one department but is a shared corporate resource. The database holds not only the organization’s operational data, but also a description of this data. For this reason, a database is also defined as a self-describing collection of integrated records. The description of the data is known as the system catalog (or data dictionary or metadata—the “data about data”). It is the self-describing nature of a database that provides program–data independence.

The approach taken with database systems, where the definition of data is separated from the application programs, is similar to the approach taken in modern software development, where an internal definition of an object and a separate external definition are provided. The users of an object see only the external definition and are unaware of how the object is defined and how it functions. One advantage of this approach, known as data abstraction, is that we can change the internal definition of an object without affecting the users of the object, provided that the external definition remains the same. In the same way, the database approach separates the structure of the data from the application programs and stores it in the database. If new data structures are added or existing structures are modified, then the application programs are unaffected, provided that they do not directly depend upon what has been modified. For example, if we add a new field to a record or create a new file, existing applications are unaffected. However, if we remove a field from a file that an application program uses, then that application program is affected by this change and must be modified accordingly.

Another expression in the definition of a database that we should explain is “logically related.” When we analyze the information needs of an organization, we attempt to identify entities, attributes, and relationships. An entity is a distinct object (a person, place, thing, concept, or event) in the organization that is to be represented in the database. An attribute is a property that describes some aspect of the object that we wish to record, and a relationship is an association between entities. For example, Figure 1.6 shows an Entity–Relationship (ER) diagram for part of the DreamHome case study. It consists of:

- six entities (the rectangles): Branch, Staff, PropertyForRent, Client, PrivateOwner, and Lease;
- seven relationships (the names adjacent to the lines): Has, Offers, Oversees, Views, Owns, LeasedBy, and Holds;
- six attributes, one for each entity: branchNo, staffNo, propertyNo, clientNo, ownerNo, and leaseNo.

The database represents the entities, the attributes, and the logical relationships between the entities. In other words, the database holds data that is logically related. We discuss the ER model in detail in Chapters 12 and 13.
Chapter 1  Introduction to Databases

1.3.2 The Database Management System (DBMS)

The DBMS is the software that interacts with the users’ application programs and the database. Typically, a DBMS provides the following facilities:

- It allows users to define the database, usually through a Data Definition Language (DDL). The DDL allows users to specify the data types and structures and the constraints on the data to be stored in the database.

- It allows users to insert, update, delete, and retrieve data from the database, usually through a Data Manipulation Language (DML). Having a central repository for all data and data descriptions allows the DML to provide a general inquiry facility to this data, called a query language. The provision of a query language alleviates the problems with file-based systems where the user has to work with a fixed set of queries or there is a proliferation of programs, causing major software management problems. The most common query language is the Structured Query Language (SQL, pronounced “S-Q-L”, or sometimes “See-Quel”), which is now both the formal and de facto standard language for relational DBMSs. To emphasize the importance of SQL, we devote Chapters 6–9 and Appendix I to a comprehensive study of this language.

- It provides controlled access to the database. For example, it may provide:
  - a security system, which prevents unauthorized users accessing the database;
  - an integrity system, which maintains the consistency of stored data;
  - a concurrency control system, which allows shared access of the database;
  - a recovery control system, which restores the database to a previous consistent state following a hardware or software failure;
  - a user-accessible catalog, which contains descriptions of the data in the database.

![Example Entity–Relationship diagram](image)

**Figure 1.6** Example Entity–Relationship diagram.
1.3.3 (Database) Application Programs

A computer program that interacts with the database by issuing an appropriate request (typically an SQL statement) to the DBMS.

Users interact with the database through a number of application programs that are used to create and maintain the database and to generate information. These programs can be conventional batch applications or, more typically nowadays, online applications. The application programs may be written in a programming language or in higher-level fourth-generation language.

The database approach is illustrated in Figure 1.7, based on the file approach of Figure 1.5. It shows the Sales and Contracts Departments using their application programs to access the database through the DBMS. Each set of departmental application programs handles data entry, data maintenance, and the generation of reports. However, as opposed to the file-based approach, the physical structure and storage of the data are now managed by the DBMS.

Views

With this functionality, the DBMS is an extremely powerful and useful tool. However, as the end-users are not too interested in how complex or easy a task is for the system, it could be argued that the DBMS has made things more complex, because they now see more data than they actually need or want. For example, the details that the Contracts Department wants to see for a rental property, as shown in Figure 1.5, have changed in the database approach, shown in Figure 1.7. Now the database also holds the property type, the number of rooms, and the owner details. In recognition of this problem, a DBMS provides another facility known as a view mechanism, which allows each user to have his or her own view of the database.

![Figure 1.7 Database processing.](image-url)
database (a view is, in essence, some subset of the database). For example, we could set up a view that allows the Contracts Department to see only the data that they want to see for rental properties.

As well as reducing complexity by letting users see the data in the way they want to see it, views have several other benefits:

- **Views provide a level of security.** Views can be set up to exclude data that some users should not see. For example, we could create a view that allows a branch manager and the Payroll Department to see all staff data, including salary details, and we could create a second view that other staff would use that excludes salary details.

- **Views provide a mechanism to customize the appearance of the database.** For example, the Contracts Department may wish to call the monthly rent field (rent) by the more obvious name, Monthly Rent.

- **A view can present a consistent, unchanging picture of the structure of the database,** even if the underlying database is changed (for example, fields added or removed, relationships changed, files split, restructured, or renamed). If fields are added or removed from a file, and these fields are not required by the view, the view is not affected by this change. Thus, a view helps provide the program–data independence we mentioned previously.

The previous discussion is general and the actual level of functionality offered by a DBMS differs from product to product. For example, a DBMS for a personal computer may not support concurrent shared access, and may provide only limited security, integrity, and recovery control. However, modern, large multi-user DBMS products offer all the functions mentioned and much more. Modern systems are extremely complex pieces of software consisting of millions of lines of code, with documentation comprising many volumes. This complexity is a result of having to provide software that handles requirements of a more general nature. Furthermore, the use of DBMSs nowadays requires a system that provides almost total reliability and 24/7 availability (24 hours a day, 7 days a week), even in the presence of hardware or software failure. The DBMS is continually evolving and expanding to cope with new user requirements. For example, some applications now require the storage of graphic images, video, sound, and so on. To reach this market, the DBMS must change. It is likely that new functionality will always be required, so the functionality of the DBMS will never become static. We discuss the basic functions provided by a DBMS in later chapters.

### 1.3.4 Components of the DBMS Environment

We can identify five major components in the DBMS environment: hardware, software, data, procedures, and people, as illustrated in Figure 1.8.

---

**Figure 1.8** The DBMS environment.
Hardware
The DBMS and the applications require hardware to run. The hardware can range from a single personal computer to a single mainframe or a network of computers. The particular hardware depends on the organization’s requirements and the DBMS used. Some DBMSs run only on particular hardware or operating systems, while others run on a wide variety of hardware and operating systems. A DBMS requires a minimum amount of main memory and disk space to run, but this minimum configuration may not necessarily give acceptable performance. A simplified hardware configuration for *DreamHome* is illustrated in Figure 1.9. It consists of a network of small servers, with a central server located in London running the **backend** of the DBMS, that is, the part of the DBMS that manages and controls access to the database. It also shows several computers at various locations running the **frontend** of the DBMS, that is, the part of the DBMS that interfaces with the user. This is called a **client–server** architecture: the backend is the server and the frontends are the clients. We discuss this type of architecture in Section 3.1.

![Diagram of DreamHome hardware configuration](image-url)
**Software**

The software component comprises the DBMS software itself and the application programs, together with the operating system, including network software if the DBMS is being used over a network. Typically, application programs are written in a third-generation programming language (3GL), such as C, C++, C#, Java, Visual Basic, COBOL, Fortran, Ada, or Pascal, or a fourth-generation language (4GL), such as SQL, embedded in a third-generation language. The target DBMS may have its own fourth-generation tools that allow rapid development of applications through the provision of nonprocedural query languages, reports generators, forms generators, graphics generators, and application generators. The use of fourth-generation tools can improve productivity significantly and produce programs that are easier to maintain. We discuss fourth-generation tools in Section 2.2.3.

**Data**

Perhaps the most important component of the DBMS environment—certainly from the end-users’ point of view—is the data. In Figure 1.8, we observe that the data acts as a bridge between the machine components and the human components. The database contains both the operational data and the metadata, the “data about data.” The structure of the database is called the *schema*. In Figure 1.7, the schema consists of four files, or *tables*, namely: `PropertyForRent`, `PrivateOwner`, `Client`, and `Lease`. The `PropertyForRent` table has eight fields, or *attributes*, namely: `propertyNo`, `street`, `city`, `zipCode`, `type` (the property type), `rooms` (the number of rooms), `rent` (the monthly rent), and `ownerNo`. The `ownerNo` attribute models the relationship between `PropertyForRent` and `PrivateOwner`: that is, an owner *Owns* a property for rent, as depicted in the ER diagram of Figure 1.6. For example, in Figure 1.2 we observe that owner CO46, Joe Keogh, owns property PA14.

The data also incorporates the system catalog, which we discuss in detail in Section 2.4.

**Procedures**

Procedures refer to the instructions and rules that govern the design and use of the database. The users of the system and the staff who manage the database require documented procedures on how to use or run the system. These may consist of instructions on how to:

- Log on to the DBMS.
- Use a particular DBMS facility or application program.
- Start and stop the DBMS.
- Make backup copies of the database.
- Handle hardware or software failures. This may include procedures on how to identify the failed component, how to fix the failed component (for example, telephone the appropriate hardware engineer), and, following the repair of the fault, how to recover the database.
- Change the structure of a table, reorganize the database across multiple disks, improve performance, or archive data to secondary storage.
People
The final component is the people involved with the system. We discuss this component in Section 1.4.

1.3.5 Database Design: The Paradigm Shift
Until now, we have taken it for granted that there is a structure to the data in the database. For example, we have identified four tables in Figure 1.7: PropertyForRent, PrivateOwner, Client, and Lease. But how did we get this structure? The answer is quite simple: the structure of the database is determined during database design. However, carrying out database design can be extremely complex. To produce a system that will satisfy the organization's information needs requires a different approach from that of file-based systems, where the work was driven by the application needs of individual departments. For the database approach to succeed, the organization now has to think of the data first and the application second. This change in approach is sometimes referred to as a paradigm shift. For the system to be acceptable to the end-users, the database design activity is crucial. A poorly designed database will generate errors that may lead to bad decisions, which may have serious repercussions for the organization. On the other hand, a well-designed database produces a system that provides the correct information for the decision-making process to succeed in an efficient way.

The objective of this book is to help effect this paradigm shift. We devote several chapters to the presentation of a complete methodology for database design (see Chapters 16–19). It is presented as a series of simple-to-follow steps, with guidelines provided throughout. For example, in the ER diagram of Figure 1.6, we have identified six entities, seven relationships, and six attributes. We provide guidelines to help identify the entities, attributes, and relationships that have to be represented in the database.

Unfortunately, database design methodologies are not very popular. Many organizations and individual designers rely very little on methodologies for conducting the design of databases, and this is commonly considered a major cause of failure in the development of database systems. Owing to the lack of structured approaches to database design, the time or resources required for a database project are typically underestimated, the databases developed are inadequate or inefficient in meeting the demands of applications, documentation is limited, and maintenance is difficult.

1.4 Roles in the Database Environment
In this section, we examine what we listed in the previous section as the fifth component of the DBMS environment: the people. We can identify four distinct types of people who participate in the DBMS environment: data and database administrators, database designers, application developers, and end-users.

1.4.1 Data and Database Administrators
The database and the DBMS are corporate resources that must be managed like any other resource. Data and database administration are the roles generally associated with the management and control of a DBMS and its data. The
**Data Administrator (DA)** is responsible for the management of the data resource, including database planning; development and maintenance of standards, policies and procedures; and conceptual/logical database design. The DA consults with and advises senior managers, ensuring that the direction of database development will ultimately support corporate objectives.

The **Database Administrator (DBA)** is responsible for the physical realization of the database, including physical database design and implementation, security and integrity control, maintenance of the operational system, and ensuring satisfactory performance of the applications for users. The role of the DBA is more technically oriented than the role of the DA, requiring detailed knowledge of the target DBMS and the system environment. In some organizations there is no distinction between these two roles; in others, the importance of the corporate resources is reflected in the allocation of teams of staff dedicated to each of these roles. We discuss data and database administration in more detail in Section 20.15.

### 1.4.2 Database Designers

In large database design projects, we can distinguish between two types of designer: logical database designers and physical database designers. The **logical database designer** is concerned with identifying the data (that is, the entities and attributes), the relationships between the data, and the constraints on the data that is to be stored in the database. The logical database designer must have a thorough and complete understanding of the organization’s data and any constraints on this data (the constraints are sometimes called **business rules**). These constraints describe the main characteristics of the data as viewed by the organization. Examples of constraints for *DreamHome* are:

- a member of staff cannot manage more than 100 properties for rent or sale at the same time;
- a member of staff cannot handle the sale or rent of his or her own property;
- a solicitor cannot act for both the buyer and seller of a property.

To be effective, the logical database designer must involve all prospective database users in the development of the data model, and this involvement should begin as early in the process as possible. In this book, we split the work of the logical database designer into two stages:

- conceptual database design, which is independent of implementation details, such as the target DBMS, application programs, programming languages, or any other physical considerations;
- logical database design, which targets a specific data model, such as relational, network, hierarchical, or object-oriented.

The **physical database designer** decides how the logical database design is to be physically realized. This involves:

- mapping the logical database design into a set of tables and integrity constraints;
- selecting specific storage structures and access methods for the data to achieve good performance;
- designing any security measures required on the data.
Many parts of physical database design are highly dependent on the target DBMS, and there may be more than one way of implementing a mechanism. Consequently, the physical database designer must be fully aware of the functionality of the target DBMS and must understand the advantages and disadvantages of each alternative implementation. The physical database designer must be capable of selecting a suitable storage strategy that takes account of usage. Whereas conceptual and logical database design are concerned with the what, physical database design is concerned with the how. It requires different skills, which are often found in different people. We present a methodology for conceptual database design in Chapter 16, for logical database design in Chapter 17, and for physical database design in Chapters 18 and 19.

### 1.4.3 Application Developers

Once the database has been implemented, the application programs that provide the required functionality for the end-users must be implemented. This is the responsibility of the application developers. Typically, the application developers work from a specification produced by systems analysts. Each program contains statements that request the DBMS to perform some operation on the database, which includes retrieving data, inserting, updating, and deleting data. The programs may be written in a third-generation or fourth-generation programming language, as discussed previously.

### 1.4.4 End-Users

The end-users are the “clients” of the database, which has been designed and implemented and is being maintained to serve their information needs. End-users can be classified according to the way they use the system:

- **Naïve users** are typically unaware of the DBMS. They access the database through specially written application programs that attempt to make the operations as simple as possible. They invoke database operations by entering simple commands or choosing options from a menu. This means that they do not need to know anything about the database or the DBMS. For example, the checkout assistant at the local supermarket uses a bar code reader to find out the price of the item. However, there is an application program present that reads the bar code, looks up the price of the item in the database, reduces the database field containing the number of such items in stock, and displays the price on the till.

- **Sophisticated users.** At the other end of the spectrum, the sophisticated end-user is familiar with the structure of the database and the facilities offered by the DBMS. Sophisticated end-users may use a high-level query language such as SQL to perform the required operations. Some sophisticated end-users may even write application programs for their own use.

### 1.5 History of Database Management Systems

We have already seen that the predecessor to the DBMS was the file-based system. However, there was never a time when the database approach began and the file-based system ceased. In fact, the file-based system still exists in specific areas. It has been suggested that the DBMS has its roots in the 1960s Apollo
moon-landing project, which was initiated in response to President Kennedy’s objective of landing a man on the moon by the end of that decade. At that time, there was no system available that would be able to handle and manage the vast amounts of information that the project would generate.

As a result, North American Aviation (NAA, now Rockwell International), the prime contractor for the project, developed software known as GUAM (for Generalized Update Access Method). GUAM was based on the concept that smaller components come together as parts of larger components, and so on, until the final product is assembled. This structure, which conforms to an upside-down tree, is also known as a hierarchical structure. In the mid-1960s, IBM joined NAA to develop GUAM into what is now known as IMS (for Information Management System). The reason that IBM restricted IMS to the management of hierarchies of records was to allow the use of serial storage devices; most notably magnetic tape, which was a market requirement at that time. This restriction was subsequently dropped. Although one of the earliest commercial DBMSs, IMS is still the main hierarchical DBMS used by most large mainframe installations.

In the mid-1960s, another significant development was the emergence of IDS (or Integrated Data Store) from General Electric. This work was headed by one of the early pioneers of database systems, Charles Bachmann. This development led to a new type of database system known as the network DBMS, which had a profound effect on the information systems of that generation. The network database was developed partly to address the need to represent more complex data relationships than could be modeled with hierarchical structures, and partly to impose a database standard. To help establish such standards, the Conference on Data Systems Languages (CODASYL), comprising representatives of the U.S. government and the world of business and commerce, formed a List Processing Task Force in 1965, subsequently renamed the Data Base Task Group (DBTG) in 1967. The terms of reference for the DBTG were to define standard specifications for an environment that would allow database creation and data manipulation. A draft report was issued in 1969, and the first definitive report was issued in 1971. The DBTG proposal identified three components:

- the network schema—the logical organization of the entire database as seen by the DBA—which includes a definition of the database name, the type of each record, and the components of each record type;
- the subschema—the part of the database as seen by the user or application program;
- a data management language to define the data characteristics and the data structure, and to manipulate the data.

For standardization, the DBTG specified three distinct languages:

- a schema DDL, which enables the DBA to define the schema;
- a subschema DDL, which allows the application programs to define the parts of the database they require;
- a DML, to manipulate the data.

Although the report was not formally adopted by the American National Standards Institute (ANSI), a number of systems were subsequently developed following the DBTG proposal. These systems are now known as CODASYL or DBTG systems. The CODASYL and hierarchical approaches represented the first generation of DBMSs.
We look more closely at these systems on the Web site for this book (see the Preface for the URL). However, these two models have some fundamental disadvantages:

- complex programs have to be written to answer even simple queries based on navigational record-oriented access;
- there is minimal data independence;
- there is no widely accepted theoretical foundation.

In 1970, E. F. Codd of the IBM Research Laboratory produced his highly influential paper on the relational data model (“A relational model of data for large shared data banks,” Codd, 1970). This paper was very timely and addressed the disadvantages of the former approaches. Many experimental relational DBMSs were implemented thereafter, with the first commercial products appearing in the late 1970s and early 1980s. Of particular note is the System R project at IBM’s San José Research Laboratory in California, which was developed during the late 1970s (Astrahan et al., 1976). This project was designed to prove the practicality of the relational model by providing an implementation of its data structures and operations, and led to two major developments:

- the development of a structured query language called SQL, which has since become the standard language for relational DBMSs;
- the production of various commercial relational DBMS products during the 1980s, for example DB2 and SQL/DS from IBM and Oracle from Oracle Corporation.

Now there are several hundred relational DBMSs for both mainframe and PC environments, though many are stretching the definition of the relational model. Other examples of multi-user relational DBMSs are MySQL from Oracle Corporation, Ingres from Actian Corporation, SQL Server from Microsoft, and Informix from IBM. Examples of PC-based relational DBMSs are Office Access and Visual FoxPro from Microsoft, Interbase from Embarcadero Technologies, and R:Base from R:Base Technologies. Relational DBMSs are referred to as second-generation DBMSs. We discuss the relational data model in Chapter 4.

The relational model is not without its failings; in particular, its limited modeling capabilities. There has been much research since then attempting to address this problem. In 1976, Chen presented the Entity–Relationship model, which is now a widely accepted technique for database design and the basis for the methodology presented in Chapters 16 and 17 of this book. In 1979, Codd himself attempted to address some of the failings in his original work with an extended version of the relational model called RM/T (1979) and subsequently RM/V2 (1990). The attempts to provide a data model that represents the “real world” more closely have been loosely classified as semantic data modeling.

In response to the increasing complexity of database applications, two “new” systems have emerged: the object-oriented DBMS (OODBMS) and the object-relational DBMS (ORDBMS). However, unlike previous models, the actual composition of these models is not clear. This evolution represents third-generation DBMSs, which we discuss in Chapters 9 and 27–28.

The 1990s also saw the rise of the Internet, the three-tier client-server architecture, and the demand to allow corporate databases to be integrated with Web applications. The late 1990s saw the development of XML (eXtensible Markup Language), which has had a profound effect on many aspects of IT, including
database integration, graphical interfaces, embedded systems, distributed systems, and database systems. We will discuss Web–database integration and XML in Chapters 29–30.

Specialized DBMSs have also been created, such as data warehouses, which can store data drawn from several data sources, possibly maintained by different operating units of an organization. Such systems provide comprehensive data analysis facilities to allow strategic decisions to be made based on, for example, historical trends. All the major DBMS vendors provide data warehousing solutions. We discuss data warehouses in Chapters 31–32. Another example is the enterprise resource planning (ERP) system, an application layer built on top of a DBMS that integrates all the business functions of an organization, such as manufacturing, sales, finance, marketing, shipping, invoicing, and human resources. Popular ERP systems are SAP R/3 from SAP and PeopleSoft from Oracle. Figure 1.10 provides a summary of historical development of database systems.

<table>
<thead>
<tr>
<th>TIMEFRAME</th>
<th>DEVELOPMENT</th>
<th>COMMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1960s (onwards)</td>
<td>File-based systems</td>
<td>Precursor to the database system. Decentralized approach: each department stored and controlled its own data.</td>
</tr>
<tr>
<td>Mid-1960s</td>
<td>Hierarchical and network data models</td>
<td>Represents first-generation DBMSs. Main hierarchical system is IMS from IBM and the main network system is IDMS/R from Computer Associates. Lacked data independence and required complex programs to be developed to process the data.</td>
</tr>
<tr>
<td>1970</td>
<td>Relational model proposed</td>
<td>Publication of E. F. Codd’s seminal paper “A relational model of data for large shared data banks,” which addresses the weaknesses of first-generation systems.</td>
</tr>
<tr>
<td>1970s</td>
<td>Prototype RDBMSs developed</td>
<td>During this period, two main prototypes emerged: the Ingres project at the University of California at Berkeley (started in 1970) and the System R project at IBM’s San José Research Laboratory in California (started in 1974), which led to the development of SQL.</td>
</tr>
<tr>
<td>1976</td>
<td>ER model proposed</td>
<td>Publication of Chen’s paper “The Entity-Relationship model—Toward a unified view of data.” ER modeling becomes a significant component in methodologies for database design.</td>
</tr>
<tr>
<td>1979</td>
<td>Commercial RDBMSs appear</td>
<td>Commercial RDBMSs like Oracle, Ingres, and DB2 appear. These represent the second generation of DBMSs.</td>
</tr>
<tr>
<td>1990s</td>
<td>OODBMS and ORDBMSs appear</td>
<td>This period initially sees the emergence of OODBMSs and later ORDBMSs (Oracle 8, with object features released in 1997).</td>
</tr>
<tr>
<td>1990s</td>
<td>Data warehousing systems appear</td>
<td>This period also see releases from the major DBMS vendors of data warehousing systems and thereafter data mining products.</td>
</tr>
<tr>
<td>Mid-1990s</td>
<td>Web–database integration</td>
<td>The first Internet database applications appear; DBMS vendors and third-party vendors recognize the significance of the Internet and support web–database integration.</td>
</tr>
<tr>
<td>1998</td>
<td>XML</td>
<td>XML 1.0 ratified by the W3C. XML becomes integrated with DBMS products and native XML databases are developed.</td>
</tr>
</tbody>
</table>

Figure 1.10  Historical development of database systems.
1.6 Advantages and Disadvantages of DBMSs

The database management system has promising potential advantages. Unfortunately, there are also disadvantages. In this section, we examine these advantages and disadvantages.

Advantages

The advantages of database management systems are listed in Table 1.2.

Control of data redundancy  As we discussed in Section 1.2, traditional file-based systems waste space by storing the same information in more than one file. For example, in Figure 1.5, we stored similar data for properties for rent and clients in both the Sales and Contracts Departments. In contrast, the database approach attempts to eliminate the redundancy by integrating the files so that multiple copies of the same data are not stored. However, the database approach does not eliminate redundancy entirely, but controls the amount of redundancy inherent in the database. Sometimes it is necessary to duplicate key data items to model relationships; at other times, it is desirable to duplicate some data items to improve performance. The reasons for controlled duplication will become clearer as you read the next few chapters.

Data consistency  By eliminating or controlling redundancy, we reduce the risk of inconsistencies occurring. If a data item is stored only once in the database, any update to its value has to be performed only once and the new value is available immediately to all users. If a data item is stored more than once and the system is aware of this, the system can ensure that all copies of the item are kept consistent. Unfortunately, many of today’s DBMSs do not automatically ensure this type of consistency.

More information from the same amount of data  With the integration of the operational data, it may be possible for the organization to derive additional information from the same data. For example, in the file-based system illustrated in Figure 1.5, the Contracts Department does not know who owns a leased property. Similarly, the Sales Department has no knowledge of lease details. When we

<table>
<thead>
<tr>
<th>TABLE 1.2 Advantages of DBMSs.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control of data redundancy</td>
</tr>
<tr>
<td>Data consistency</td>
</tr>
<tr>
<td>More information from the same amount of data</td>
</tr>
<tr>
<td>Sharing of data</td>
</tr>
<tr>
<td>Improved data integrity</td>
</tr>
<tr>
<td>Improved security</td>
</tr>
<tr>
<td>Enforcement of standards</td>
</tr>
</tbody>
</table>
integrate these files, the Contracts Department has access to owner details and the Sales Department has access to lease details. We may now be able to derive more information from the same amount of data.

Sharing of data Typically, files are owned by the people or departments that use them. On the other hand, the database belongs to the entire organization and can be shared by all authorized users. In this way, more users share more of the data. Furthermore, new applications can build on the existing data in the database and add only data that is not currently stored, rather than having to define all data requirements again. The new applications can also rely on the functions provided by the DBMS, such as data definition and manipulation, and concurrency and recovery control, rather than having to provide these functions themselves.

Improved data integrity Database integrity refers to the validity and consistency of stored data. Integrity is usually expressed in terms of constraints, which are consistency rules that the database is not permitted to violate. Constraints may apply to data items within a single record or to relationships between records. For example, an integrity constraint could state that a member of staff’s salary cannot be greater than $40,000 or that the branch number contained in a staff record, representing the branch where the member of staff works, must correspond to an existing branch office. Again, integration allows the DBA to define integrity constraints, and the DBMS to enforce them.

Improved security Database security is the protection of the database from unauthorized users. Without suitable security measures, integration makes the data more vulnerable than file-based systems. However, integration allows the DBA to define database security, and the DBMS to enforce it. This security may take the form of user names and passwords to identify people authorized to use the database. The access that an authorized user is allowed on the data may be restricted by the operation type (retrieval, insert, update, delete). For example, the DBA has access to all the data in the database; a branch manager may have access to all data that relates to his or her branch office; and a sales assistant may have access to all data relating to properties but no access to sensitive data such as staff salary details.

Enforcement of standards Again, integration allows the DBA to define and the DBMS to enforce the necessary standards. These may include departmental, organizational, national, or international standards for such things as data formats to facilitate exchange of data between systems, naming conventions, documentation standards, update procedures, and access rules.

Economy of scale Combining all the organization’s operational data into one database and creating a set of applications that work on this one source of data can result in cost savings. In this case, the budget that would normally be allocated to each department for the development and maintenance of its file-based system can be combined, possibly resulting in a lower total cost, leading to an economy of scale. The combined budget can be used to buy a system configuration that is more suited to the organization’s needs. This may consist of one large, powerful computer or a network of smaller computers.
1.6 Advantages and Disadvantages of DBMSs

**Balance of conflicting requirements** Each user or department has needs that may be in conflict with the needs of other users. Because the database is under the control of the DBA, the DBA can make decisions about the design and operational use of the database that provide the best use of resources for the organization as a whole. These decisions will provide optimal performance for important applications, possibly at the expense of less-critical ones.

**Improved data accessibility and responsiveness** Again, as a result of integration, data that crosses departmental boundaries is directly accessible to the end-users. This provides a system with potentially much more functionality that can, for example, be used to provide better services to the end-user or the organization’s clients. Many DBMSs provide query languages or report writers that allow users to ask *ad hoc* questions and to obtain the required information almost immediately at their terminal, without requiring a programmer to write some software to extract this information from the database. For example, a branch manager could list all flats with a monthly rent greater than £400 by entering the following SQL command at a terminal:

```sql
SELECT * FROM PropertyForRent WHERE type = 'Flat' AND rent > 400;
```

**Increased productivity** As mentioned previously, the DBMS provides many of the standard functions that the programmer would normally have to write in a file-based application. At a basic level, the DBMS provides all the low-level file-handling routines that are typical in application programs. The provision of these functions allows the programmer to concentrate on the specific functionality required by the users without having to worry about low-level implementation details. Many DBMSs also provide a fourth-generation environment, consisting of tools to simplify the development of database applications. This results in increased programmer productivity and reduced development time (with associated cost savings).

**Improved maintenance through data independence** In file-based systems, the descriptions of the data and the logic for accessing the data are built into each application program, making the programs dependent on the data. A change to the structure of the data—such as making an address 41 characters instead of 40 characters, or a change to the way the data is stored on disk—can require substantial alterations to the programs that are affected by the change. In contrast, a DBMS separates the data descriptions from the applications, thereby making applications immune to changes in the data descriptions. This is known as **data independence** and is discussed further in Section 2.1.5. The provision of data independence simplifies database application maintenance.

**Increased concurrency** In some file-based systems, if two or more users are allowed to access the same file simultaneously, it is possible that the accesses will interfere with each other, resulting in loss of information or even loss of integrity. Many DBMSs manage concurrent database access and ensure that such problems cannot occur. We discuss concurrency control in Chapter 22.
Improved backup and recovery services  Many file-based systems place the responsibility on the user to provide measures to protect the data from failures to the computer system or application program. This may involve performing a nightly backup of the data. In the event of a failure during the next day, the backup is restored and the work that has taken place since this backup is lost and has to be re-entered. In contrast, modern DBMSs provide facilities to minimize the amount of processing that is lost following a failure. We discuss database recovery in Section 22.3.

Disadvantages
The disadvantages of the database approach are summarized in Table 1.3.

Complexity  The provision of the functionality that we expect of a good DBMS makes the DBMS an extremely complex piece of software. Database designers and developers, data and database administrators, and end-users must understand this functionality to take full advantage of it. Failure to understand the system can lead to bad design decisions, which can have serious consequences for an organization.

Size  The complexity and breadth of functionality makes the DBMS an extremely large piece of software, occupying many megabytes of disk space and requiring substantial amounts of memory to run efficiently.

Cost of DBMSs  The cost of DBMSs varies significantly, depending on the environment and functionality provided. For example, a single-user DBMS for a personal computer may only cost $100. However, a large mainframe multi-user DBMS servicing hundreds of users can be extremely expensive, perhaps $100,000 or even $1,000,000. There is also the recurrent annual maintenance cost, which is typically a percentage of the list price.

Additional hardware costs  The disk storage requirements for the DBMS and the database may necessitate the purchase of additional storage space. Furthermore, to achieve the required performance, it may be necessary to purchase a larger machine, perhaps even a machine dedicated to running the DBMS. The procurement of additional hardware results in further expenditure.

TABLE 1.3  Disadvantages of DBMSs.

<table>
<thead>
<tr>
<th>Disadvantage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complexity</td>
</tr>
<tr>
<td>Size</td>
</tr>
<tr>
<td>Cost of DBMSs</td>
</tr>
<tr>
<td>Additional hardware costs</td>
</tr>
<tr>
<td>Cost of conversion</td>
</tr>
<tr>
<td>Performance</td>
</tr>
<tr>
<td>Greater impact of a failure</td>
</tr>
</tbody>
</table>
Cost of conversion  In some situations, the cost of the DBMS and extra hardware may be relatively small compared with the cost of converting existing applications to run on the new DBMS and hardware. This cost also includes the cost of training staff to use these new systems, and possibly the employment of specialist staff to help with the conversion and running of the systems. This cost is one of the main reasons why some organizations feel tied to their current systems and cannot switch to more modern database technology. The term **legacy system** is sometimes used to refer to an older, and usually inferior, system.

Performance  Typically, a file-based system is written for a specific application, such as invoicing. As a result, performance is generally very good. However, the DBMS is written to be more general, to cater for many applications rather than just one. The result is that some applications may not run as fast as they used to.

Greater impact of a failure  The centralization of resources increases the vulnerability of the system. Because all users and applications rely on the availability of the DBMS, the failure of certain components can bring operations to a halt.
network or CODASYL model by IDS (Integrated Data Store), both developed in the mid-1960s. The relational model, proposed by E. F. Codd in 1970, represents the second generation of DBMSs. It has had a fundamental effect on the DBMS community and there are now over one hundred relational DBMSs. The third generation of DBMSs are represented by the Object-Relational DBMS and the Object-Oriented DBMS.

- Some advantages of the database approach include control of data redundancy, data consistency, sharing of data, and improved security and integrity. Some disadvantages include complexity, cost, reduced performance, and higher impact of a failure.

Review Questions

1.1 List four government sectors in your country that use database systems.

1.2 Discuss each of the following terms:
   (a) data
   (b) database
   (c) database management system
   (d) database application program
   (e) data independence
   (f) security
   (g) integrity
   (h) views

1.3 Describe the role of database management systems (DBMS) in the database approach. Discuss why knowledge of DBMS is important for database administrators.

1.4 Describe the main characteristics of the database approach and contrast it with the file-based approach.

1.5 Describe the five components of the DBMS environment and discuss how they relate to each other.

1.6 Discuss the roles of the following personnel in the database environment:
   (a) data administrator
   (b) database administrator
   (c) logical database designer
   (d) physical database designer
   (e) application developer
   (f) end-users

1.7 Discuss the three generations of DBMSs.

1.8 Why are views an important aspect of database management systems?

Exercises

1.9 Interview some users of database systems. Which DBMS features do they find most useful and why? Which DBMS facilities do they find least useful and why? What do these users perceive to be the advantages and disadvantages of the DBMS?

1.10 A database approach addresses several of the problems and challenges associated with the traditional file-based approach. Using a DBMS to control how data is shared with different applications and users, through applications such as views, has a number of advantages. However, the implementation of a database approach has its own
challenges, such as expense. Discuss the various costs associated with the implementation of a database approach.

1.11 Study the DreamHome case study presented in Section 11.4 and Appendix A.
(a) In what ways would a DBMS help this organization?
(b) What do you think are the main objects that need to be represented in the database?
(c) What relationships do you think exist between these main objects?
(d) For each of the objects, what details do you think need to be stored in the database?
(e) What queries do you think are required?

1.12 Study the Wellmeadows Hospital case study presented in Appendix B.3.
(a) In what ways would a DBMS help this organization?
(b) What do you think are the main objects that need to be represented in the database?
(c) What relationships do you think exist between these main objects?
(d) For each of the objects, what details do you think need to be stored in the database?
(e) What queries do you think are required?

1.13 Discuss what you consider to be the three most important advantages for the use of a DBMS for a company like DreamHome and provide a justification for your selection. Discuss what you consider to be the three most important disadvantages for the use of a DBMS for a company like DreamHome and provide a justification for your selection.

1.14 Organizations have a vital need for quality information. Discuss how the following database roles relate to each other.
(a) Data Administrator
(b) Database Administrator
(c) Database Designer
(d) Application Developer
(e) End-Users
Chapter Objectives

In this chapter you will learn:

• The purpose and origin of the three-level database architecture.
• The contents of the external, conceptual, and internal levels.
• The purpose of the external/conceptual and the conceptual/internal mappings.
• The meaning of logical and physical data independence.
• The distinction between a Data Definition Language (DDL) and a Data Manipulation Language (DML).
• A classification of data models.
• The purpose and importance of conceptual modeling.
• The typical functions and services that a DBMS should provide.
• The function and importance of the system catalog.

A major aim of a database system is to provide users with an abstract view of data, hiding certain details of how data is stored and manipulated. Therefore, the starting point for the design of a database must be an abstract and general description of the information requirements of the organization that is to be represented in the database. In this chapter, and throughout this book, we use the term “organization” loosely to mean the whole organization or part of the organization. For example, in the DreamHome case study, we may be interested in modeling:

• the “real-world” entities Staff, PropertyforRent, PrivateOwner, and Client;
• attributes describing properties or qualities of each entity (for example, each Staff entry has a name, position, and salary);
• relationships between these entities (for example, Staff Manages PropertyforRent).

Furthermore, because a database is a shared resource, each user may require a different view of the data held in the database. To satisfy these needs, the architecture of most commercial DBMSs available today is based to some extent on the so-called ANSI-SPARC architecture. In this chapter, we discuss various architectural and functional characteristics of DBMSs.
Structure of this Chapter In Section 2.1, we examine the three-level ANSI-SPARC architecture and its associated benefits. In Section 2.2, we consider the types of language that are used by DBMSs, and in Section 2.3, we introduce the concepts of data models and conceptual modeling, which we expand on in later parts of the book. In Section 2.4, we discuss the functions that we would expect a DBMS to provide. The examples in this chapter are drawn from the DreamHome case study, which we discuss more fully in Section 11.4 and Appendix A.

Much of the material in this chapter provides important background information on DBMSs. However, the reader who is new to the area of database systems may find some of the material difficult to comprehend fully on first reading. Do not be too concerned about this, but be prepared to revisit parts of this chapter at a later date when you have read subsequent chapters of the book.

2.1 The Three-Level ANSI-SPARC Architecture

An early proposal for a standard terminology and general architecture for database systems was produced in 1971 by the DBTG appointed by CODASYL in 1971. The DBTG recognized the need for a two-level approach with a system view called the schema and user views called subschemas. The American National Standards Institute (ANSI) Standards Planning and Requirements Committee (SPARC), or ANSI/X3/SPARC, produced a similar terminology and architecture in 1975 (ANSI, 1975). The ANSI-SPARC architecture recognized the need for a three-level approach with a system catalog. These proposals reflected those published by the IBM user organizations Guide and Share some years previously, and concentrated on the need for an implementation-independent layer to isolate programs from underlying representational issues (Guide/Share, 1970). Although the ANSI-SPARC model did not become a standard, it still provides a basis for understanding some of the functionality of a DBMS.

For our purposes, the fundamental point of these and later reports is the identification of three levels of abstraction, that is, three distinct levels at which data items can be described. The levels form a three-level architecture comprising an external, a conceptual, and an internal level, as depicted in Figure 2.1. The way users perceive the data is called the external level. The way the DBMS and the operating system perceive the data is the internal level, where the data is actually stored using the data structures and file organizations described in Appendix F. The conceptual level provides both the mapping and the desired independence between the external and internal levels.

The objective of the three-level architecture is to separate each user’s view of the database from the way the database is physically represented. There are several reasons why this separation is desirable:

• Each user should be able to access the same data, but have a different customized view of the data. Each user should be able to change the way he or she views the data, and this change should not affect other users.
• Users should not have to deal directly with physical database storage details, such as indexing or hashing (see Appendix F). In other words, a user’s interaction with the database should be independent of storage considerations.
• The DBA should be able to change the database storage structures without affecting the users’ views.
• The internal structure of the database should be unaffected by changes to the physical aspects of storage, such as the changeover to a new storage device.
• The DBA should be able to change the conceptual structure of the database without affecting all users.

2.1.1 External Level

The users’ view of the database. This level describes that part of the database that is relevant to each user.

The external level consists of a number of different external views of the database. Each user has a view of the “real world” represented in a form that is familiar for that user. The external view includes only those entities, attributes, and relationships in the “real world” that the user is interested in. Other entities, attributes, or relationships that are not of interest may be represented in the database, but the user will be unaware of them.

In addition, different views may have different representations of the same data. For example, one user may view dates in the form (day, month, year), while another may view dates as (year, month, day). Some views might include derived or calculated data: data not actually stored in the database as such, but created when needed. For example, in the DreamHome case study, we may wish to view the age of a member of staff. However, it is unlikely that ages would be stored, as this data would have to be updated daily. Instead, the member of staff’s date of birth would be stored and age would be calculated by the DBMS when it is referenced.
Views may even include data combined or derived from several entities. We discuss views in more detail in Sections 4.4 and 7.4.

### 2.1.2 Conceptual Level

The community view of the database. This level describes what data is stored in the database and the relationships among the data.

The middle level in the three-level architecture is the conceptual level. This level contains the logical structure of the entire database as seen by the DBA. It is a complete view of the data requirements of the organization that is independent of any storage considerations. The conceptual level represents:

- all entities, their attributes, and their relationships;
- the constraints on the data;
- semantic information about the data;
- security and integrity information.

The conceptual level supports each external view, in that any data available to a user must be contained in, or derivable from, the conceptual level. However, this level must not contain any storage-dependent details. For instance, the description of an entity should contain only data types of attributes (for example, integer, real, character) and their length (such as the maximum number of digits or characters), but not any storage considerations, such as the number of bytes occupied.

### 2.1.3 Internal Level

The physical representation of the database on the computer. This level describes how the data is stored in the database.

The internal level covers the physical implementation of the database to achieve optimal runtime performance and storage space utilization. It covers the data structures and file organizations used to store data on storage devices. It interfaces with the operating system access methods (file management techniques for storing and retrieving data records) to place the data on the storage devices, build the indexes, retrieve the data, and so on. The internal level is concerned with such things as:

- storage space allocation for data and indexes;
- record descriptions for storage (with stored sizes for data items);
- record placement;
- data compression and data encryption techniques.

Below the internal level there is a physical level that may be managed by the operating system under the direction of the DBMS. However, the functions of the DBMS and the operating system at the physical level are not clear-cut and vary from system to system. Some DBMSs take advantage of many of the operating
system access methods, and others use only the most basic ones and create their own file organizations. The physical level below the DBMS consists of items that only the operating system knows, such as exactly how the sequencing is implemented and whether the fields of internal records are stored as contiguous bytes on the disk.

### 2.1.4 Schemas, Mappings, and Instances

The overall description of the database is called the database schema. There are three different types of schema in the database and these are defined according to the levels of abstraction of the three-level architecture illustrated in Figure 2.1. At the highest level, we have multiple external schemas (also called subschemas) that correspond to different views of the data. At the conceptual level, we have the conceptual schema, which describes all the entities, attributes, and relationships together with integrity constraints. At the lowest level of abstraction we have the internal schema, which is a complete description of the internal model, containing the definitions of stored records, the methods of representation, the data fields, and the indexes and storage structures used. There is only one conceptual schema and one internal schema per database.

The DBMS is responsible for mapping between these three types of schema. It must also check the schemas for consistency; in other words, the DBMS must confirm that each external schema is derivable from the conceptual schema, and it must use the information in the conceptual schema to map between each external schema and the internal schema. The conceptual schema is related to the internal schema through a conceptual/internal mapping. This mapping enables the DBMS to find the actual record or combination of records in physical storage that constitute a logical record in the conceptual schema, together with any constraints to be enforced on the operations for that logical record. It also allows any differences in entity names, attribute names, attribute order, data types, and so on to be resolved. Finally, each external schema is related to the conceptual schema by the external/conceptual mapping. This mapping enables the DBMS to map names in the user’s view to the relevant part of the conceptual schema.

An example of the different levels is shown in Figure 2.2. Two different external views of staff details exist: one consisting of a staff number (sNo), first name (fName), last name (lName), age, and salary; a second consisting of a staff number (staffNo), last name (lName), and the number of the branch the member of staff works at (branchNo). These external views are merged into one conceptual view. In this merging process, the major difference is that the age field has been changed into a date of birth field, DOB. The DBMS maintains the external/conceptual mapping; for example, it maps the sNo field of the first external view to the staffNo field of the conceptual record. The conceptual level is then mapped to the internal level, which contains a physical description of the structure for the conceptual record. At this level, we see a definition of the structure in a high-level language. The structure contains a pointer, next, which allows the list of staff records to be physically linked together to form a chain. Note that the order of fields at the internal level is different from that at the conceptual level. Again, the DBMS maintains the conceptual/internal mapping.
Chapter 2  Database Environment

It is important to distinguish between the description of the database and the database itself. The description of the database is the **database schema**. The schema is specified during the database design process and is not expected to change frequently. However, the actual data in the database may change frequently; for example, it changes every time we insert details of a new member of staff or a new property. The data in the database at any particular point in time is called a **database instance**. Therefore, many database instances can correspond to the same database schema. The schema is sometimes called the **intension** of the database; an instance is called an **extension** (or **state**) of the database.

2.1.5 Data Independence

A major objective for the three-level architecture is to provide **data independence**, which means that upper levels are unaffected by changes to lower levels. There are two kinds of data independence: **logical** and **physical**.

**Logical data independence**

The immunity of the external schemas to changes in the conceptual schema.

Changes to the conceptual schema, such as the addition or removal of new entities, attributes, or relationships, should be possible without having to change existing external schemas or having to rewrite application programs. Clearly, the users for whom the changes have been made need to be aware of them, but what is important is that other users should not be.

**Physical data independence**

The immunity of the conceptual schema to changes in the internal schema.
Changes to the internal schema, such as using different file organizations or storage structures, using different storage devices, modifying indexes or hashing algorithms, should be possible without having to change the conceptual or external schemas. From the users’ point of view, the only effect that may be noticed is a change in performance. In fact, deterioration in performance is the most common reason for internal schema changes. Figure 2.3 illustrates where each type of data independence occurs in relation to the three-level architecture.

The two-stage mapping in the ANSI-SPARC architecture may be inefficient, but it also provides greater data independence. However, for more efficient mapping, the ANSI-SPARC model allows the direct mapping of external schemas onto the internal schema, thus by-passing the conceptual schema. This mapping of course reduces data independence, so that every time the internal schema changes, the external schema and any dependent application programs may also have to change.

### 2.2 Database Languages

A data sublanguage consists of two parts: a Data Definition Language (DDL) and a Data Manipulation Language (DML). The DDL is used to specify the database schema and the DML is used to both read and update the database. These languages are called *data sublanguages* because they do not include constructs for all computing needs, such as conditional or iterative statements, which are provided by the high-level programming languages. Many DBMSs have a facility for *embedding* the sublanguage in a high-level programming language such as COBOL, Fortran, Pascal, Ada, C, C++, C#, Java, or Visual Basic. In this case, the high-level language is sometimes referred to as the *host language*. To compile the embedded file, the commands in the data sublanguage are first removed from the host-language program and replaced by function calls. The preprocessed file is then compiled, placed in an object module, linked with a DBMS-specific library containing the replaced functions, and executed when required. Most data sublanguages also provide *nonembedded* or *interactive* commands that can be input directly from a terminal.
Chapter 2  Database Environment

2.2.1 The Data Definition Language (DDL)

**DDL**

A language that allows the DBA or user to describe and name the entities, attributes, and relationships required for the application, together with any associated integrity and security constraints.

The database schema is specified by a set of definitions expressed by means of a special language called a Data Definition Language. The DDL is used to define a schema or to modify an existing one. It cannot be used to manipulate data.

The result of the compilation of the DDL statements is a set of tables stored in special files collectively called the system catalog. The system catalog integrates the metadata, which is data that describes the objects in the database and makes it easier for those objects to be accessed or manipulated. The metadata contains definitions of records, data items, and other objects that are of interest to users or are required by the DBMS. The DBMS normally consults the system catalog before the actual data is accessed in the database. The terms data dictionary and data directory are also used to describe the system catalog, although the term “data dictionary” usually refers to a more general software system than a catalog for a DBMS. We discuss the system catalog further in Section 2.4.

At a theoretical level, we could identify different DDLs for each schema in the three-level architecture: namely, a DDL for the external schemas, a DDL for the conceptual schema, and a DDL for the internal schema. However, in practice, there is one comprehensive DDL that allows specification of at least the external and conceptual schemas.

2.2.2 The Data Manipulation Language (DML)

**DML**

A language that provides a set of operations to support the basic data manipulation operations on the data held in the database.

Data manipulation operations usually include the following:

- insertion of new data into the database;
- modification of data stored in the database;
- retrieval of data contained in the database;
- deletion of data from the database.

Therefore, one of the main functions of the DBMS is to support a Data Manipulation Language in which the user can construct statements that will cause such data manipulation to occur. Data manipulation applies to the external, conceptual, and internal levels. However, at the internal level we must define rather complex low-level procedures that allow efficient data access. In contrast, at higher levels, emphasis is placed on ease of use and effort is directed at providing efficient user interaction with the system.

The part of a DML that involves data retrieval is called a query language. A query language can be defined as a high-level special-purpose language used to satisfy diverse requests for the retrieval of data held in the database. The term
“query” is therefore reserved to denote a retrieval statement expressed in a query language. The terms “query language” and “DML” are commonly used interchangeably, although this is technically incorrect.

DMLs are distinguished by their underlying retrieval constructs. We can distinguish between two types of DML: **procedural** and **nonprocedural**. The prime difference between these two data manipulation languages is that procedural languages specify how the output of a DML statement is to be obtained, while nonprocedural DMLs describe only what output is to be obtained. Typically, procedural languages treat records individually, whereas nonprocedural languages operate on sets of records.

**Procedural DMLs**

**Procedural DML**
A language that allows the user to tell the system what data is needed and exactly how to retrieve the data.

With a procedural DML, the user, or more often the programmer, specifies what data is needed and how to obtain it. This means that the user must express all the data access operations that are to be used by calling appropriate procedures to obtain the information required. Typically, such a procedural DML retrieves a record, processes it and, based on the results obtained by this processing, retrieves another record that would be processed similarly, and so on. This process of retrievals continues until the data requested from the retrieval has been gathered. Typically, procedural DMLs are embedded in a high-level programming language that contains constructs to facilitate iteration and handle navigational logic. Network and hierarchical DMLs are normally procedural (see Section 2.3).

**Nonprocedural DMLs**

**Nonprocedural DML**
A language that allows the user to state what data is needed rather than how it is to be retrieved.

Nonprocedural DMLs allow the required data to be specified in a single retrieval or update statement. With nonprocedural DMLs, the user specifies what data is required without specifying how it is to be obtained. The DBMS translates a DML statement into one or more procedures that manipulate the required sets of records, which frees the user from having to know how data structures are internally implemented and what algorithms are required to retrieve and possibly transform the data, thus providing users with a considerable degree of data independence. Nonprocedural languages are also called **declarative languages**. Relational DBMSs usually include some form of nonprocedural language for data manipulation, typically SQL or QBE (Query-By-Example). Nonprocedural DMLs are normally easier to learn and use than procedural DMLs, as less work is done by the user and more by the DBMS. We examine SQL in detail in Chapters 6–9 and Appendix I, and QBE in Appendix M.
2.2.3 Fourth-Generation Languages (4GLs)

There is no consensus about what constitutes a **fourth-generation language**; in essence, it is a shorthand programming language. An operation that requires hundreds of lines of code in a third-generation language (3GL), such as COBOL, generally requires significantly fewer lines in a 4GL.

Compared with a 3GL, which is procedural, a 4GL is nonprocedural: the user defines *what* is to be done, not *how*. A 4GL is expected to rely largely on much higher-level components known as fourth-generation tools. The user does not define the steps that a program needs to perform a task, but instead defines parameters for the tools that use them to generate an application program. It is claimed that 4GLs can improve productivity by a factor of ten, at the cost of limiting the types of problem that can be tackled. Fourth-generation languages encompass:

- presentation languages, such as query languages and report generators;
- speciality languages, such as spreadsheets and database languages;
- application generators that define, insert, update, and retrieve data from the database to build applications;
- very high-level languages that are used to generate application code.

SQL and QBE, mentioned previously, are examples of 4GLs. We now briefly discuss some of the other types of 4GL.

**Forms generators**

A forms generator is an interactive facility for rapidly creating data input and display layouts for screen forms. The forms generator allows the user to define what the screen is to look like, what information is to be displayed, and where on the screen it is to be displayed. It may also allow the definition of colors for screen elements and other characteristics, such as bold, underline, blinking, reverse video, and so on. The better forms generators allow the creation of derived attributes, perhaps using arithmetic operators or aggregates, and the specification of validation checks for data input.

**Report generators**

A report generator is a facility for creating reports from data stored in the database. It is similar to a query language in that it allows the user to ask questions of the database and retrieve information from it for a report. However, in the case of a report generator, we have much greater control over what the output looks like. We can let the report generator automatically determine how the output should look or we can create our own customized output reports using special report-generator command instructions.

There are two main types of report generator: language-oriented and visually oriented. In the first case, we enter a command in a sublanguage to define what data is to be included in the report and how the report is to be laid out. In the second case, we use a facility similar to a forms generator to define the same information.
2.3 Data Models and Conceptual Modeling

Graphics generators
A graphics generator is a facility to retrieve data from the database and display the data as a graph showing trends and relationships in the data. Typically, it allows the user to create bar charts, pie charts, line charts, scatter charts, and so on.

Application generators
An application generator is a facility for producing a program that interfaces with the database. The use of an application generator can reduce the time it takes to design an entire software application. Application generators typically consist of prewritten modules that comprise fundamental functions that most programs use. These modules, usually written in a high-level language, constitute a library of functions to choose from. The user specifies what the program is supposed to do; the application generator determines how to perform the tasks.

2.3 Data Models and Conceptual Modeling

We mentioned earlier that a schema is written using a DDL. In fact, it is written in the DDL of a particular DBMS. Unfortunately, this type of language is too low level to describe the data requirements of an organization in a way that is readily understandable by a variety of users. What we require is a higher-level description of the schema: that is, a data model.

A model is a representation of real-world objects and events, and their associations. It is an abstraction that concentrates on the essential, inherent aspects of an organization and ignores the accidental properties. A data model represents the organization itself. It should provide the basic concepts and notations that will allow database designers and end-users to communicate unambiguously and accurately their understanding of the organizational data. A data model can be thought of as comprising three components:

1. a structural part, consisting of a set of rules according to which databases can be constructed;
2. a manipulative part, defining the types of operation that are allowed on the data (this includes the operations that are used for updating or retrieving data from the database and for changing the structure of the database);
3. a set of integrity constraints, which ensures that the data is accurate.

The purpose of a data model is to represent data and to make the data understandable. If it does this, then it can be easily used to design a database. To reflect the ANSI-SPARC architecture introduced in Section 2.1, we can identify three related data models:

1. an external data model, to represent each user’s view of the organization, sometimes called the Universe of Discourse (UoD);
(2) a conceptual data model, to represent the logical (or community) view that is DBMS-independent;

(3) an internal data model, to represent the conceptual schema in such a way that it can be understood by the DBMS.

There have been many data models proposed in the literature. They fall into three broad categories: object-based, record-based, and physical data models. The first two are used to describe data at the conceptual and external levels, the third is used to describe data at the internal level.

2.3.1 Object-Based Data Models

Object-based data models use concepts such as entities, attributes, and relationships. An entity is a distinct object (a person, place, thing, concept, event) in the organization that is to be represented in the database. An attribute is a property that describes some aspect of the object that we wish to record, and a relationship is an association between entities. Some of the more common types of object-based data model are:

- Entity-Relationship (ER)
- Semantic
- Functional
- Object-oriented

The ER model has emerged as one of the main techniques for database design and forms the basis for the database design methodology used in this book. The object-oriented data model extends the definition of an entity to include not only the attributes that describe the state of the object but also the actions that are associated with the object, that is, its behavior. The object is said to encapsulate both state and behavior. We look at the ER model in depth in Chapters 12 and 13 and the object-oriented model in Chapters 27–28. We also examine the functional data model in Section 27.5.2.

2.3.2 Record-Based Data Models

In a record-based model, the database consists of a number of fixed-format records, possibly of differing types. Each record type defines a fixed number of fields, typically of a fixed length. There are three principal types of record-based logical data model: the relational data model, the network data model, and the hierarchical data model. The hierarchical and network data models were developed almost a decade before the relational data model, so their links to traditional file processing concepts are more evident.

Relational data model

The relational data model is based on the concept of mathematical relations. In the relational model, data and relationships are represented as tables, each of which has a number of columns with a unique name. Figure 2.4 is a sample instance of a relational schema for part of the DreamHome case study, showing branch and staff
details. For example, it shows that employee John White is a manager with a salary of £30,000, who works at branch (branchNo) B005, which, from the first table, is at 22 Deer Rd in London. It is important to note that there is a relationship between Staff and Branch: a branch office has staff. However, there is no explicit link between these two tables; it is only by knowing that the attribute branchNo in the Staff relation is the same as the branchNo of the Branch relation that we can establish that a relationship exists.

Note that the relational data model requires only that the database be perceived by the user as tables. However, this perception applies only to the logical structure of the database, that is, the external and conceptual levels of the ANSI-SPARC architecture. It does not apply to the physical structure of the database, which can be implemented using a variety of storage structures. We discuss the relational data model in Chapter 4.

**Network data model**

In the network model, data is represented as collections of records, and relationships are represented by sets. Compared with the relational model, relationships are explicitly modeled by the sets, which become pointers in the implementation. The records are organized as generalized graph structures with records appearing as nodes (also called segments) and sets as edges in the graph. Figure 2.5 illustrates an instance of a network schema for the same data set presented in Figure 2.4. The most popular network DBMS is Computer Associates’ IDMS/R. We discuss the network data model in more detail on the Web site for this book (see the Preface for the URL).

**Hierarchical data model**

The hierarchical model is a restricted type of network model. Again, data is represented as collections of records and relationships are represented by sets.
However, the hierarchical model allows a node to have only one parent. A hierarchical model can be represented as a tree graph, with records appearing as nodes (also called segments) and sets as edges. Figure 2.6 illustrates an instance of a hierarchical schema for the same data set presented in Figure 2.4. The main hierarchical DBMS is IBM’s IMS, although IMS also provides nonhierarchical features. We discuss the hierarchical data model in more detail on the Web site for this book (see the Preface for the URL).

Record-based (logical) data models are used to specify the overall structure of the database and a higher-level description of the implementation. Their main drawback is that they do not provide adequate facilities for explicitly specifying constraints on the data, whereas the object-based data models lack the means of logical structure specification but provide more semantic substance by allowing the user to specify constraints on the data.

Figure 2.5 A sample instance of a network schema.

Figure 2.6 A sample instance of a hierarchical schema.
The majority of modern commercial systems are based on the relational paradigm, whereas the early database systems were based on either the network or hierarchical data models. The latter two models require the user to have knowledge of the physical database being accessed, whereas the former provides a substantial amount of data independence. Hence, relational systems adopt a declarative approach to database processing (that is, they specify what data is to be retrieved), but network and hierarchical systems adopt a navigational approach (that is, they specify how the data is to be retrieved).

2.3.3 Physical Data Models

Physical data models describe how data is stored in the computer, representing information such as record structures, record orderings, and access paths. There are not as many physical data models as logical data models; the most common ones are the unifying model and the frame memory.

2.3.4 Conceptual Modeling

From an examination of the three-level architecture, we see that the conceptual schema is the heart of the database. It supports all the external views and is, in turn, supported by the internal schema. However, the internal schema is merely the physical implementation of the conceptual schema. The conceptual schema should be a complete and accurate representation of the data requirements of the enterprise.† If this is not the case, some information about the enterprise will be missing or incorrectly represented and we will have difficulty fully implementing one or more of the external views.

Conceptual modeling or conceptual database design is the process of constructing a model of the information use in an enterprise that is independent of implementation details, such as the target DBMS, application programs, programming languages, or any other physical considerations. This model is called a conceptual data model. Conceptual models are also referred to as “logical models” in the literature. However, in this book we make a distinction between conceptual and logical data models. The conceptual model is independent of all implementation details, whereas the logical model assumes knowledge of the underlying data model of the target DBMS. In Chapters 16 and 17 we present a methodology for database design that begins by producing a conceptual data model, which is then refined into a logical model based on the relational data model. We discuss database design in more detail in Section 10.6.

2.4 Functions of a DBMS

In this section, we look at the types of function and service that we would expect a DBMS to provide. Codd (1982) lists eight services that should be provided by any full-scale DBMS, and we have added two more that might reasonably be expected to be available.

†When we are discussing the organization in the context of database design we normally refer to the business or organization as the enterprise.
(1) **Data storage, retrieval, and update**

A DBMS must furnish users with the ability to store, retrieve, and update data in the database.

This is the fundamental function of a DBMS. From the discussion in Section 2.1, clearly in providing this functionality, clearly the DBMS should hide the internal physical implementation details (such as file organization and storage structures) from the user.

(2) **A user-accessible catalog**

A DBMS must furnish a catalog in which descriptions of data items are stored and which is accessible to users.

A key feature of the ANSI-SPARC architecture is the recognition of an integrated **system catalog** to hold data about the schemas, users, applications, and so on. The catalog is expected to be accessible to users as well as to the DBMS. A system catalog, or data dictionary, is a repository of information describing the data in the database; it is the “data about the data” or the **metadata**. The amount of information and the way the information is used vary with the DBMS. Typically, the system catalog stores:

- names, types, and sizes of data items;
- names of relationships;
- integrity constraints on the data;
- names of authorized users who have access to the data;
- the data items that each user can access and the types of access allowed; for example, insert, update, delete, or read access;
- external, conceptual, and internal schemas and the mappings between the schemas, as described in 2.1.4;
- usage statistics, such as the frequencies of transactions and counts on the number of accesses made to objects in the database.

The DBMS system catalog is one of the fundamental components of the system. Many of the software components that we describe in the next section rely on the system catalog for information. Some benefits of a system catalog are:

- Information about data can be collected and stored centrally. This helps to maintain control over the data as a resource.
- The meaning of data can be defined, which will help other users understand the purpose of the data.
- Communication is simplified, because exact meanings are stored. The system catalog may also identify the user or users who own or access the data.
- Redundancy and inconsistencies can be identified more easily as the data is centralized.
- Changes to the database can be recorded.
• The impact of a change can be determined before it is implemented because the
system catalog records each data item, all its relationships, and all its users.
• Security can be enforced.
• Integrity can be ensured.
• Audit information can be provided.

Some authors make a distinction between system catalog and data directory, in
that a data directory holds information relating to where data is stored and how it
is stored. The ISO has adopted a standard for data dictionaries called Information
Resource Dictionary System (IRDS) (ISO 1990, 1993). IRDS is a software tool that
can be used to control and document an organization’s information sources. It pro-
vides a definition for the tables that comprise the data dictionary and the operations
that can be used to access these tables. We use the term “system catalog” in this book to
refer to all repository information. We discuss other types of statistical information
stored in the system catalog to assist with query optimization in Section 23.4.1.

(3) Transaction support

A DBMS must furnish a mechanism that will ensure either that all the updates
corresponding to a given transaction are made or that none of them is made.

A transaction is a series of actions, carried out by a single user or application pro-
gram, which accesses or changes the contents of the database. For example, some
simple transactions for the DreamHome case study might be to add a new member of
staff to the database, to update the salary of a member of staff, or to delete a prop-
erty from the register. A more complicated example might be to delete a member of
staff from the database and to reassign the properties that he or she managed to
another member of staff. In this case, there is more than one change to be made to
the database. If the transaction fails during execution, perhaps because of a com-
puter crash, the database will be in an inconsistent state: some changes will have
been made and others will not. Consequently, the changes that have been made
will have to be undone to return the database to a consistent state again. We discuss
transaction support in Section 22.1.

(4) Concurrency control services

A DBMS must furnish a mechanism to ensure that the database is updated correctly
when multiple users are updating the database concurrently.

One major objective in using a DBMS is to enable many users to access shared data
concurrently. Concurrent access is relatively easy if all users are only reading data,
as there is no way that they can interfere with one another. However, when two
or more users are accessing the database simultaneously and at least one of them
is updating data, there may be interference that can result in inconsistencies. For
example, consider two transactions $T_1$ and $T_2$, which are executing concurrently, as
illustrated in Figure 2.7.
T₁ is withdrawing £10 from an account (with balance balₓ) and T₂ is depositing £100 into the same account. If these transactions were executed serially, one after the other with no interleaving of operations, the final balance would be £190, regardless of which was performed first. However, in this example transactions T₁ and T₂ start at nearly the same time and both read the balance as £100. T₂ then increases balₓ by £100 to £200 and stores the update in the database. Meanwhile, transaction T₁ decrements its copy of balₓ by £10 to £90 and stores this value in the database, overwriting the previous update and thereby “losing” £100.

The DBMS must ensure that when multiple users are accessing the database, interference cannot occur. We discuss this issue fully in Section 22.2.

### (5) Recovery services

A DBMS must furnish a mechanism for recovering the database in the event that the database is damaged in any way.

When discussing transaction support, we mentioned that if the transaction fails, then the database has to be returned to a consistent state. This failure may be the result of a system crash, media failure, a hardware or software error causing the DBMS to stop, or it may be the result of the user detecting an error during the transaction and aborting the transaction before it completes. In all these cases, the DBMS must provide a mechanism to restore the database to a consistent state. We discuss database recovery in Section 22.3.

### (6) Authorization services

A DBMS must furnish a mechanism to ensure that only authorized users can access the database.

It is not difficult to envision instances where we would want to prevent some of the data stored in the database from being seen by all users. For example, we may want only branch managers to see salary-related information for staff and to prevent all other users from seeing this data. Additionally, we may want to protect the database from unauthorized access. The term “security” refers to the protection of the database against unauthorized access, either intentional or accidental. We expect the DBMS to provide mechanisms to ensure that the data is secure. We discuss security in Chapter 20.
(7) **Support for data communication**

A DBMS must be capable of integrating with communication software.

Most users access the database from workstations. Sometimes these workstations are connected directly to the computer hosting the DBMS. In other cases, the workstations are at remote locations and communicate with the computer hosting the DBMS over a network. In either case, the DBMS receives requests as *communications messages* and responds in a similar way. All such transmissions are handled by a data communication manager (DCM). Although the DCM is not part of the DBMS, it is necessary for the DBMS to be capable of being integrated with a variety of DCMs if the system is to be commercially viable. Even DBMSs for personal computers should be capable of being run on a local area network so that one centralized database can be established for users to share, rather than having a series of disparate databases, one for each user. This does not imply that the database has to be distributed across the network, but rather that users should be able to access a centralized database from remote locations. We refer to this type of topology as *distributed processing* (see Section 24.1.1).

(8) **Integrity services**

A DBMS must furnish a means to ensure that both the data in the database and changes to the data follow certain rules.

“Database integrity” refers to the correctness and consistency of stored data: it can be considered as another type of database protection. Although integrity is related to security, it has wider implications: integrity is concerned with the quality of data itself. Integrity is usually expressed in terms of *constraints*, which are consistency rules that the database is not permitted to violate. For example, we may want to specify a constraint that no member of staff can manage more than 100 properties at any one time. Here, we would want the DBMS to check when we assign a property to a member of staff whether this limit would be exceeded and to prevent the assignment from occurring if the limit has been reached.

In addition to these eight services, we could also reasonably expect the following two services to be provided by a DBMS.

(9) **Services to promote data independence**

A DBMS must include facilities to support the independence of programs from the actual structure of the database.

We discussed the concept of data independence in Section 2.1.5. Data independence is normally achieved through a view or subschema mechanism. Physical data independence is easier to achieve: there are usually several types of change that
can be made to the physical characteristics of the database without affecting the views. However, complete logical data independence is more difficult to achieve. The addition of a new entity, attribute, or relationship can usually be accommodated, but not their removal. In some systems, any type of change to an existing component in the logical structure is prohibited.

(10) Utility services

A DBMS should provide a set of utility services.

Utility programs help the DBA administer the database effectively. Some utilities work at the external level, and consequently can be produced by the DBA. Other utilities work at the internal level and can be provided only by the DBMS vendor. Examples of utilities of the latter kind are:

- import facilities, to load the database from flat files, and export facilities, to unload the database to flat files;
- monitoring facilities, to monitor database usage and operation;
- statistical analysis programs, to examine performance or usage statistics;
- index reorganization facilities, to reorganize indexes and their overflows;
- garbage collection and reallocation, to remove deleted records physically from the storage devices, to consolidate the space released, and to reallocate it where it is needed.

Chapter Summary

- The ANSI-SPARC database architecture uses three levels of abstraction: external, conceptual, and internal. The external level consists of the users’ views of the database. The conceptual level is the community view of the database: it specifies the information content of the entire database, independent of storage considerations. The conceptual level represents all entities, their attributes, and their relationships, as well as the constraints on the data, and security and integrity information. The internal level is the computer’s view of the database: it specifies how data is represented, how records are sequenced, what indexes and pointers exist, and so on.

- The external/conceptual mapping transforms requests and results between the external and conceptual levels. The conceptual/internal mapping transforms requests and results between the conceptual and internal levels.

- A database schema is a description of the database structure. There are three different types of schema in the database; these are defined according to the three levels of the ANSI-SPARC architecture. Data independence makes each level immune to changes to lower levels. Logical data independence refers to the immunity of the external schemas to changes in the conceptual schema. Physical data independence refers to the immunity of the conceptual schema to changes in the internal schema.

- A data sublanguage consists of two parts: a Data Definition Language (DDL) and a Data Manipulation Language (DML). The DDL is used to specify the database schema and the DML is used to both read and update the database. The part of a DML that involves data retrieval is called a query language.
A data model is a collection of concepts that can be used to describe a set of data, the operations to manipulate the data, and a set of integrity constraints for the data. They fall into three broad categories: object-based data models, record-based data models, and physical data models. The first two are used to describe data at the conceptual and external levels; the latter is used to describe data at the internal level.

Object-based data models include the Entity–Relationship, semantic, functional, and object-oriented models. Record-based data models include the relational, network, and hierarchical models.

Conceptual modeling is the process of constructing a detailed architecture for a database that is independent of implementation details, such as the target DBMS, application programs, programming languages, or any other physical considerations. The design of the conceptual schema is critical to the overall success of the system. It is worth spending the time and effort necessary to produce the best possible conceptual design.

Functions and services of a multi-user DBMS include data storage, retrieval, and update; a user-accessible catalog; transaction support; concurrency control and recovery services; authorization services; support for data communication; integrity services; services to promote data independence; and utility services.

The system catalog is one of the fundamental components of a DBMS. It contains “data about the data,” or metadata. The catalog should be accessible to users. The Information Resource Dictionary System is an ISO standard that defines a set of access methods for a data dictionary. This standard allows dictionaries to be shared and transferred from one system to another.

**Review Questions**

2.1 Explain the concept of database schema and discuss the three types of schema in a database.
2.2 What are data sublanguages? Why are they important?
2.3 What is a data model? Discuss the main types of data model.
2.4 Discuss the function and importance of conceptual modeling.
2.5 Describe the types of facility that you would expect to be provided in a multi-user DBMS.
2.6 Of the facilities described in your answer to Question 2.5, which ones do you think would not be needed in a standalone PC DBMS? Provide justification for your answer.
2.7 Discuss the function and importance of the system catalog.
2.8 Discuss the differences between DDL and DML. What operations would you typically expect to be available in each language?
2.9 Discuss the differences between procedural DMLs and nonprocedural DMLs.
2.10 Name four object-based data models.
2.11 Name three record-based data models. Discuss the main differences between these data models.
2.12 What is a transaction? Give an example of a transaction.
2.13 What is concurrency control and why does a DBMS need a concurrency control facility?
2.14 Define the term “database integrity”. How does database integrity differ from database security?
Exercises

2.15 Analyze the DBMSs that you are currently using. Determine each system’s compliance with the functions that we would expect to be provided by a DBMS. What type of language does each system provide? What type of architecture does each DBMS use? Check the accessibility and extensibility of the system catalog. Is it possible to export the system catalog to another system?

2.16 Write a program that stores names and telephone numbers in a database. Write another program that stores names and addresses in a database. Modify the programs to use external, conceptual, and internal schemas. What are the advantages and disadvantages of this modification?

2.17 Write a program that stores names and dates of birth in a database. Extend the program so that it stores the format of the data in the database: in other words, create a system catalog. Provide an interface that makes this system catalog accessible to external users.

2.18 A database approach uses different data models. Common database models include the relational model, the network model and the hierarchical model. Which data model should be chosen under which circumstances and why?