Chapter 1

Definitions and Introduction to the Subject

The order and connection of ideas is the same as the order and connection of things.

Baruch (Benedict de) Spinoza
Prop. VII from Ethics, Part Two.

When we talk informally about a *d*atabase, we refer to a collection of mutually related data, to the computer hardware that is used to store it, and to the programs used to manipulate it.

By mutually related we mean that the data represents knowledge about a specific enterprise, say a company, a university, or a governmental institution. The data may also be related because it deals with a certain problem area, perhaps about a disease which is of concern to the staff of a number of hospitals. The data should be organized so that it can be processed to yield information.

The organization of the data in a database has to represent the underlying meaning or semantics of the data correctly and efficiently. In conventional programs the structure of data is arranged for the convenience of the program. A database contains data to be used by many and diverse programs. The organization of a database can hence not be solely determined by decisions made while programming specific functions.

This chapter begins by defining the concept of a file, since files are the major physical units into which a database is organized. The second section discusses operations or tasks performed when using a database. In Sec. 1-3 we develop a classification of data management which will provide the structure for all subsequent material. We then take familiar concepts from programming and relate them to the presented classification. This sequence is intended to provide a link between our existing programming experience and the approach taken in this presentation of database-management methodology. In Sec. 1-8 a list of application areas is given as an aid for the selection of a topic for analysis to be followed throughout the text.

You may have learned nothing new after you have read this chapter, but the framework established will allow us to proceed through the book in an orderly fashion.

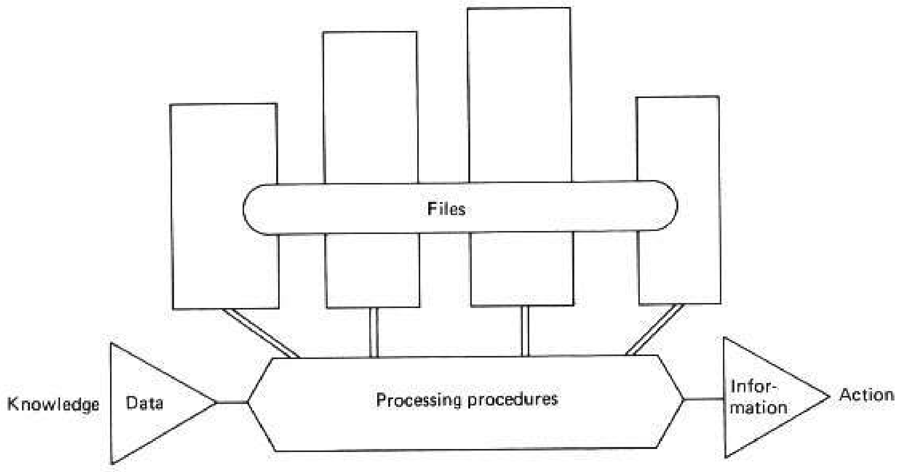

Figure 1-1 A database.

1-1 FILES

A database is a collection of related data. The data storage for a database is accomplished by the use of one or more files. All files of one database are accessible from one computer or from any interconnected computer if the database is distributed over several computers. Procedures in these computers are used to enter data, store data in files, and process the data in response to requests for information. A *file* is defined to be a collection of similar records kept on secondary computer storage devices.

Typical of secondary storage devices are disk drives with magnetic disks, but there are a number of alternatives. A *record* is defined at this point to be a collection of related *fields* containing elemental data items. A more formal and detailed definition will be developed in Chap. 3. A data item is typically represented by a value which is part of a description of an object or an event. Computational processes can manipulate such values.

1-1-1 Size

To warrant the attention and the approaches discussed in this book, the database should be reasonably *large*. We will concentrate on topics that are applicable to large external files. This outlook limits the choice of algorithms that can be applied. Collections of data that can be processed in their entirety in the directly addressable memory of a computer, its primary storage, allow techniques that will not be covered here. The use of the term database also implies that a variety of people are involved. Data entry may be done by people far removed from the users and the data may contain information suitable for several purposes. The quantity of data to be handled may range from moderately large to very large. These measures of size depend on the hardware and on operational constraints which may apply in a given environment.

Large implies a quantity of data which is greater than a single person can handle alone, even when aided by a computer system. The actual quantity will vary depending on the complexity of the data and applications. An example of a large database is the integrated personnel and product data system in a manufacturing company of about 6000 employees, with more than 300,000 records of 21 types [Emerson in Jardine[74]].
*

A very large database is an essential part of an enterprise and will be in continuous use by many people. At the same time it will require many storage devices. A very large database presents particular problems in its management, since it cannot be turned off without affecting the operation and well-being of the enterprise into which it is integrated. A formal definition of a very large database is one where the time required for making a copy is greater than the permissible interval between update transactions performed to keep the database up to date. An example of a very large database is found at a telephone company with 5 million subscribers [Karsner in Kerr[75]]. Much larger yet are databases at the social security administration and other national systems.

To have a copy of the contents of a file frozen at a certain instant in time is important for many purposes, including periodic analysis of data, backup for reliability purposes, and auditing. To avoid problems, it is best not to permit a file to be modified while it is being copied. In a very large database these two considerations conflict. This definition imposes certain design constraints on the systems which we will be discussing.

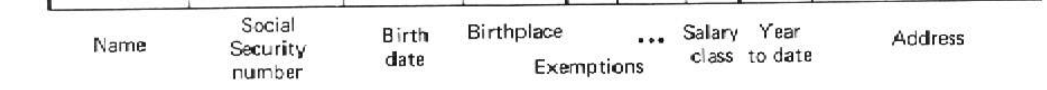

Figure 1-2 A payroll file.

* Superior number next to reference indicates the year of publication.

1-1-2 File Organization

Files not only are characterized by their size but are further distinguished by their organization. Differences in the organizations of files lead to great differences in performance when storing and retrieving records. The evaluation and comparison of file organizations is an important aspect of this book.

Six basic file organization types will be analyzed in detail in Chap. 3, and Chap. 4 will show some of the possible combinations and permutations of these basic file types. A database often requires more than one type of file.

1-1-3 Input-Output

When reading or writing files, data is transferred between storage and memory devices of the computer system. When reading input or writing output, data enters or leaves the computer system. A database is concerned with the data which remains within the scope of the system. Data which is written onto tapes, although the tapes are dismounted, securely stored, and later mounted and read again, can be part of the database. Data which is taken out, made accessible for modification, and reentered has to be considered new input, since its consistency is not assured.

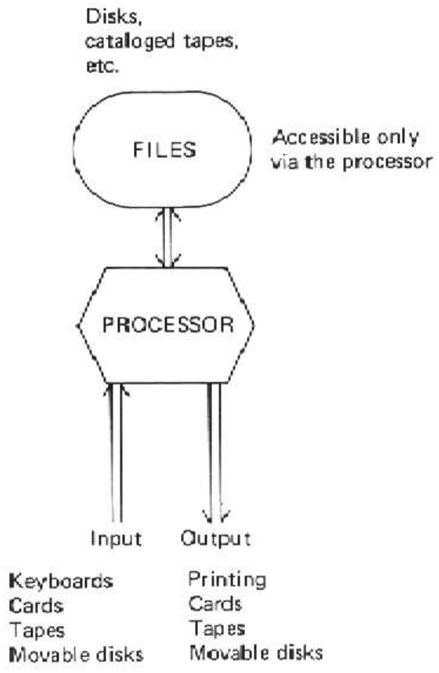

Figure 1-3 Files versus input-output

Examples of devices used for files are nonremovable disks and drums, removable disks that are normally kept mounted, dismountable master tapes or disks kept at the computer site, archival tapes and disks kept in remote vaults for protection, and some times card decks containing master lists that are loaded into the system when needed.

Devices used for input and output are terminals and all those machines that read or write media as cards, paper tape, printed reports, microfilm output, and tapes or disks that are shipped between computer systems.

In many computer systems the distinction of files versus input and output is not clearly delineated. The subject of input and output is at best as complex as the database area and will not be covered in this text. We will simply assume that an operating system is available which provides adequate input and output capabilities, including access via on-line terminals where appropriate, when we talk about database systems.

We will not discuss file organizations that are based on input and output facilities. These tend to regard data as a continuous stream of characters. Stream files, as defined by PL/1, and their equivalents in other systems, are based on the reading and writing of continuous lines of text. Continuous text streams are important for communication but are not suitable for data manipulation. The word “file” will also not be used to refer to the hardware used to store the data comprising the files.

1-2 COMPUTATIONS ON A DATABASE

In the first section we considered some of the static aspects of a database, namely, the storage of data as files, records, and fields. We now will look at a database from a dynamic point of view. The dynamic behavior of programming structures has been studied whenever operating systems have been analyzed. The terminology used in this chapter follows Tsichritzis[74]. The term *computation* is used to denote a section of an application which manipulates the database. Most of the computations which are used to manipulate a data collection are conceptually simple. We recognize four kinds of computations related to databases:

1 The building of the data collection
2 Updating of data elements in the data collection
3 Retrieval of data from the data collection
4 Reduction of large quantities of data to usable form

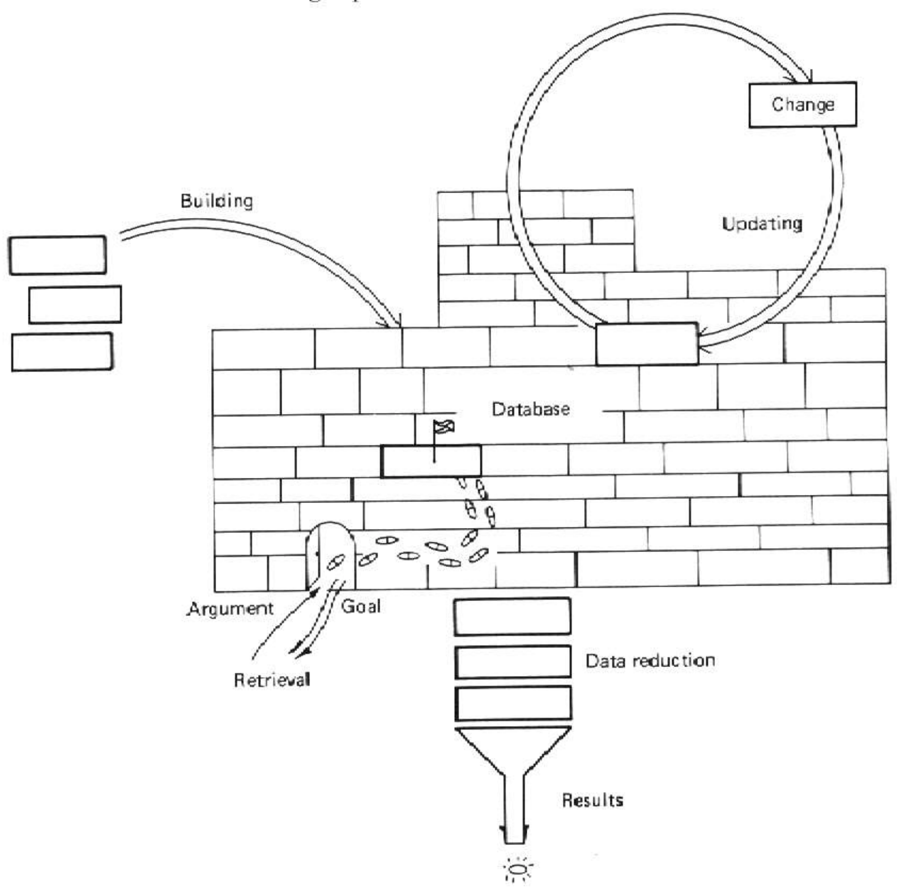

Figure 1-4 Computations with a database.

A database application will use all four kinds of computations, but certain applications will stress one kind more than others. The kinds are sketched in Fig. 1-4. One may note that the only computation which uses the calculating powers of the computer is the reduction of data to yield information.

The building of a database includes data collection, data coding, and data entry. This kind is often the most costly part of a database operation.

The updating of a database includes inserting new data, changing stored data values when necessary, and deleting invalid or obsolete data.

Update activity varies much among types of applications. A static database, one not receiving updates, may be used in a retrospective study where all the data is collected prior to analysis. A dynamic or volatile database is encountered in applications such as reservations systems.

Data retrieval can consist of the fetching of a specific element in order to obtain a stored value or fact, or the collection of a series of related elements to obtain data regarding some relationship which manifests itself when data are joined. To fetch a specific data record the database will be entered using a *s*earch argument, which will be matched with a *k*ey in the database records. The argument of a search is sometimes called the *s*earch key; in general the context clarifies the distinction.

In computing, the usage of the word key is unfortunately the opposite of the concept associated with a key in common household usage: a person who desires something from a house uses a key which will unlock the latch of the proper house; here a person uses the argument to match the key of the proper record.

Data reduction is needed when the relevant data volume exceeds the capability of the requestors. In a static database this kind of activity tends to dominate. In dynamic databases data reduction is used primarily for period summaries and trend analyses. When the desired information is diffused throughout the database, it may be necessary to access most of its contents in order to summarize or abstract the data. Statistical summaries, annual business operating statements, or graphical data presentations are examples of frequently used data reduction techniques. Data reduction is very demanding of database performance.

Conceptual descriptions of computations for a particular problem frequently can be written in a few lines; the procedures to implement them may be diagrammed on a one-page flowchart and programmed in a short time. Yet the effort required to bring a database system into operation frequently involves expenditure of large amounts of time and money. The causes for this discrepancy will be discussed later in this book but may well be obvious before that point.

1-2-1 Processes

The operating system which schedules the execution of a computation requested by a user may decompose the computation into a number of distinct *p*rocesses. Processes are the basic computational units managed by an operating system. The various processes of a single computation may require different units of the computer's hardware and may be executed in parallel. Various types of operating systems will require different properties from the processes they manage. In Sec. 1-7 we will review the types of commonly available operating systems. The scheduling and performance of processes, and hence of the computations that were specified, determine the performance of database systems.

1-2-2 Process Sections

A process itself can be decomposed into a number of sections. A section is the collection of program steps during which the operating system does not have to worry about the process. When a process stops, requests data from a file, requires a new allocation of storage space, and so on, the operating system has to intervene. When, for instance, a record from a file is requested, the operating system has to assure that the record is not currently being modified by another process. A section of a process which presents such interaction potential is known as a critical section. Figure 1-5 sketches a process with its sections.

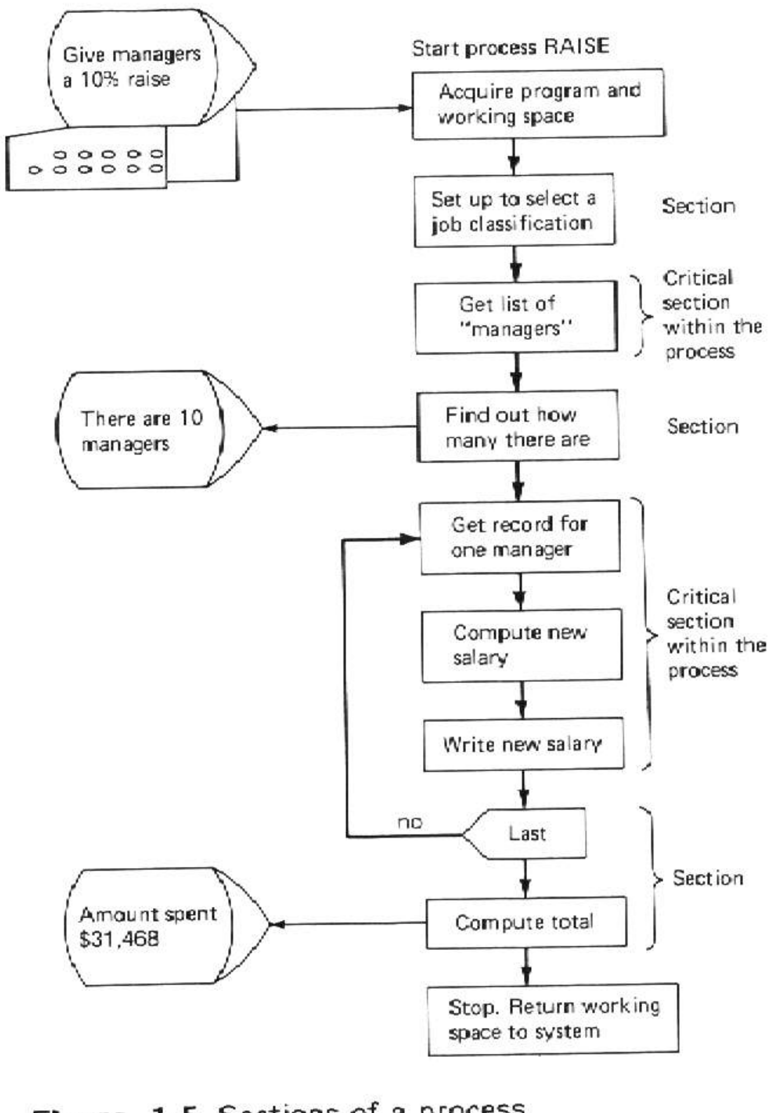
Give managers a 10% raise
Start process RAISE
Acquire program and working space
Set up to select a job classification
Section
Get list of "managers"
Critical section within the process
There are 10 managers
Find out how many there are
Section
Get record for one manager
Compute new salary
Write new salary
Critical section within the process
no
Last
Section
Amount spent $31,468
Compute total
Stop. Return working space to system

Figure 1-5 Sections of a process.

The design and performance of typical sections which manipulate the database are the subjects of Chap. 3. There we will consider that the sections are not affected by activities being carried on simultaneously in the same or adjacent databases. Interactions will be encountered beginning in Chap. 5, and Chap. 13 will be devoted to the problem of resolving interference between parallel critical sections.

1-3 A HIERARCHICAL VIEW OF DATA

Database problems can be initially approached from two points of view. There is the comprehensive view, in which the database is seen as a component of an overall system involving people, organizations, and communication; in contrast there is the microscopic view, in which the database is a collection of the hardware bits and bytes provided by computer technology. In between these two extremes is subject matter of this book. A database is viewed as a structured representation of data which describe a subset of the real world. This representation is implemented using files, records, and references between these records. Figure 1-6 presents the structure in the form of a pyramid to symbolize the breadth of the base required to obtain information.

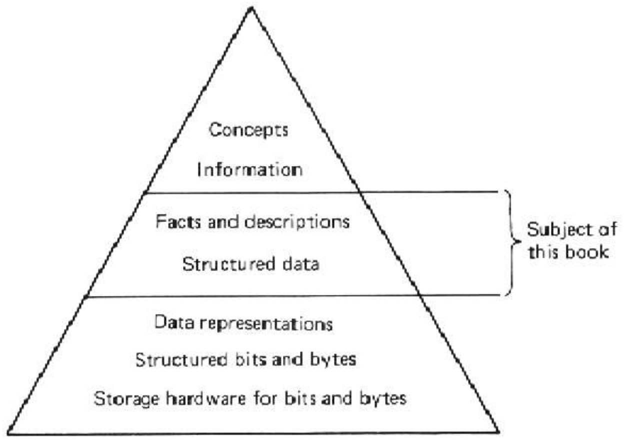

Figure 1-6 Database levels.

The technical aspects of database implementations can be discussed independently of the informational significance of the data value stored. The fact that this book will only casually touch on the value of data is not intended to imply that people who actually implement information processing systems should ignore these considerations in their particular applications. It is clearly irresponsible if computer professionals ignore the ramifications of the task at hand or their contribution to the social and human environment. A number of textbooks deal with the proper use of information for business and governmental operations and decision making [Davis74, DeGreene73, Gotlieb73, Westin71].

This text also avoids subjects on the engineering side of the database domain, the design and construction of data storage devices. Although the architecture and performance of computer hardware is central to the evaluation of database systems, very little discussion of hardware-design alternatives will take place. Obviously, familiarity with the hardware that provides the building blocks of computer systems is beneficial to the system developer. Again, textbooks are available for the computer engineer at many levels of sophistication [Hill^{73}, Siewiorek^{82}]. A sufficiently large body of material is available between these two extremes to keep us busy for some time.

Table 1-1 presents the various areas of discussion and places them in relationship to each other. The table also lists some of the fields of study which provide background or further material appropriate to the levels presented. The reader is not expected to be knowledgeable in all these areas; they are listed merely as navigational aids.

The term *n*aming in the table is used to mean the creation of symbolic references to data with minimal concern as to location and form. The term *a*ddressing is used to denote the exact specification of position and representation. We see in Table 1-1 that as we descend to more primitive levels, naming is replaced more and more by addressing.

A Comparison A comparison can be made here between the levels presented in Table 1-1 and the operation of a railroad.

1 The *c*onceptual level concerns itself with the idea of value added due to the transportation of the goods and their relevance to the economic structure.

2 The *d*escriptive level concerns itself with scheduling, train sizes, and equipment combinations required to move goods at a a reasonable cost and with adequate flexibility.

3 The *o*rganizational level concerns itself with the building of trackage, the operation of marshaling yards, and the operational efficiency of the complex.

4 The *m*aterial level concerns itself with the design of locomotives, freight cars, signaling mechanisms, rails, and ties, and all the myriad of pieces of equipment required to actually run a railroad.

Description and Organization This text considers the two central levels shown in the diagram: the descriptive level and the organizational level.

On the *d*escriptive level, we present the processes carried out to satisfy demands for data according to its meaning or utility relative to some problem. We have to specify the data in terms which aid their selection by such processes. The content of data elements is quite important. Database systems provide tools to relate data elements according to their meaning. Because of this concern we analyze this level using tools that employ logic and formal semantics. There are also issues that required some intuition or common sense for their resolution.

On the *o*rganizational level, we present the processes which locate and move data according to demands of efficiency. Descriptions of data may be manipulated at this level as well as the data themselves. Here it is not the content but the size and the position of the data element which is of concern. File systems provide tools to fetch records containing data according to relative position or name. This level is analyzed using quantitative tools in order to provide good performance estimates.

Table 1-1*a* Levels of Database Structure: conceptual and descriptive

Level 1 At the *conceptual level* we deal with *i*nformation, data with sufficient significance to make us act. Information has the essence of newness and nonredundancy so that its acquisition increases our knowledge.

A: Some processes used to obtain information:
- Cross tabulation
- Exception reporting
- Mathematical and statistical analysis
- Linear and dynamic programming
- Deductive inference

B: Intellectual tools for managing information:
- Intuitive or formal models of the application area
- Formalized conceptual relationships
- Theory of semantics
- Heuristic processes

C: Requirements for the production of information:
- Named and described data
- Means to collect needed data comprehensively or representatively
- Means to analyze such data, intellectually or automatically

Level 2 At the *descriptive level* we deal with *d*ata, a collection of text or numbers describing items or events.

A: Processes used to obtain and manipulate data:
- Data collection
- Data organization
- Data selection and retrieval

B: Intellectual tools for data manipulation:
- Logic
- Data modeling
- Computer science concepts of syntax and semantics
- Knowledge of formal languages and meta-languages
- Propositional and relational calculus

C: Requirements for large-scale data manipulation:
- Named files
- Named or addressable records
- Addressable data elements
- Catalogs and descriptions of available facilities
- Facilities to obtain resources from the system
- Procedure libraries

The two central levels are highly dependent on each other. The descriptive level can easily specify requirements that the organizational level cannot reasonably supply using any given set of hardware. On the other hand, a file organization that expands the basic hardware capabilities only minimally may impose restrictions that prevent the descriptive level from being effective. The separation of the two inner levels assures that logical requirements are kept distinct from the performance issues.

Table 1-1*b* Levels of Database Structure: organizational and material

Level 3 At the *organizational level* we deal with *d*ata representation, data removed from context, but named and processable.

A: Processes used to encode data and manipulate the encoded data:
Encoding and decoding
Storage layout definition
File management
Space management and cataloging
Moving and comparing of fields containing coded data
Control of errors

B: Intellectual tools for the design of data representation and manipulation:
Information theory
Arithmetic processes
Programming languages
Knowledge of computer architecture and component performance

C: Required for the representation of data:
Storage devices for encoded data
Hardware facilities to move addressed fields between memory and storage
Capability for comparison of encoded data

Level 4 At the *material level* finally we find *h*ardware, the equipment used to provide unformatted addressable storage.

A: Processes used to obtain hardware:
Engineering design
Production

B: Tools required for the engineer who is designing hardware:
Knowledge of electronics, physics, and logic
Experience in production practice

C: What the engineer in turn has to rely on:
Availability of components, devices, and measurement tools
Availability of production facilities

The applications of a database will be a reflection of the decision maker's concept of the information problems and will vary over time as the needs, the analysis tools, and the insight of the users of the database mature. During this time the operating systems and computers which support the database may evolve and change. We therefore want a clean separation or interface between the levels.

In order to ensure that the separation of levels is not evaded, systems which attempt to support the concept of distinct levels may hide the structural detail of lower levels from users at the higher levels. This idea of *i*nformation hiding can remain valid only when lower level functions are reliable, complete, and reasonably efficient. The independence of the descriptive and organizational levels is also referred to as *d*ata independence.

1-4 CURRENT PRACTICE

The previous section discussed database concepts in an abstract fashion. Experience has to be placed into the framework which has been developed. Some examples will help relate programming practice to database concepts.

1-4-1 A File Program

To illustrate the use of a file, we will use a simple program to carry out the actions that were presented in Fig. 1-5. The program is not intended to be especially good or bad, but rather typical.

Example 1-1 A transaction program for a database

```
/* Transaction program to give a 'raise' to a class of employees */
raise:PROCEDURE(message);
DECLARE message CHAR(200);
s1 :DECLARE emp_list(5000) INITIAL((5000)0);
s2 :DECLARE workspace(20);
DECLARE class CHAR(8), pct INTEGER;
class = SUBSTR(message,6,8);  pct = SUBSTR(message,17,2);
/* get record listing employees of this class */
s3 :OPEN FILE(job_classification) DIRECT;
s4 :READ FILE(job_classification) KEY(class) INTO(emp_list);
CLOSE FILE(job_classification);
/* count how many employees are eligible for the raise */
DO i = 1 TO 5000 WHILE emp_list(i) ¬= 0;  END;
no_employees = i - 1;
s5 :PUT SKIP EDIT(no_employees, class) ('There are ', I5 / A8);
/* prepare to give raises */
OPEN FILE(payroll) SEQUENTIAL UPDATE;
total_increase = 0;
DO j = 1 TO no_employees;
s6 :  READ FILE(payroll) KEY(emp_list(j) ) INTO(workspace);
/*   increase salary field of record */
increase = workspace(18) * pct / 100;
total_increase = total_increase + increase;
workspace(18) = workspace(18) + increase;
REWRITE FILE(payroll) FROM(workspace);
END;
/* report total cost of all raises */
PUT SKIP EDIT(total_increase) ('Amount spent $', F(8,2));
END raise;
```

The DECLARE statements, s1 and s2, define working areas for a copy of each of the two types of records used within the program. All items of one record are related by being kept in adjoining cells on the file and in the declared arrays. The OPEN statements name the files and specify their usage. When they are executed the name of the requested file is found by the operating system (OS) in a directory of file names. The programmer who wrote the program expects that the file, from this point on, is under exclusive control of this program until the file is released by a CLOSE statement. The execution of the OPEN statements can involve a number of sections within the OS, and may take a relatively long time.

The READ statement s4 fetches the particular record named by the title class = "Managers", as specified by the input message of Fig. 1-5, to obtain a list of employee numbers. The print statement s5 reports the number of employees found for this class. The READ statement s6 names a particular record of the file identified with the employee and moves a copy of this record into the workspace. The system will have to look up and compute the proper record address in the files corresponding to the identification. It will use primary or core storage for the workspace. The address for the workspace is determined during program compilation and loading.

The program now can extract information, such as the salary, by addressing fields in the copy of the record. The copy of the record is modified to reflect the employee's raise. The comment provides some help in understanding the content of the addressed fields. The final REWRITE completes the update of a record in the file. For its proper operation, it depends on the record address that was determined by the file system during the execution of the READ statement. The file system must have saved this address at the time of the READ for use in the REWRITE.

Naming Versus Addressing In this example the records on the file have been specified using a *name*, namely, the title or the employee_number. The file system performs some computational procedure to determine an actual address for the corresponding record and then fetches the record. It remembers this address for any file which is to be updated in order to perform a later REWRITE properly. Within the record fetched, the program located data by providing addresses; for instance, the salary field was addressed as position 18 of the workspace. When an address is used, the programmer controls position and size of the data unit.

1-4-2 Operating System Control

Facilities of an operating system were used above to handle the major units of data, the files. This has provided independence of the program and the specification of the storage hardware. The writer of the transaction processing program expects that there exists somewhere an expert who controls the actual placement of the records of the file in storage. The expert directs the file system, which is generally a part of a computer operating system. The expert is restricted to select among facilities provided within the computer system by the vendor. Operating system alternatives are summarized in Sec. 1-7.

Statements outside the programming language are used to provide information to the operating system regarding the program. These statements form the *c*ontrol language for the operating system. The lack of communication between the programming language and the control language is often a barrier to effective utilization of computer systems.

A control language statement for the above example may read as shown in Example 1-2.

Example 1-2 Control language for "raise" transaction

```
FILE(data):(NAME('payroll'), DEVICE(diskdrive), LOCATION(disk5),
ORGANIZATION(fixed records(80 bytes), indexed, sequential)
BUFFERS(2), SIZE(60 tracks), etc.
```

The statement above has been made prettier than most control languages permit in order to aid comprehension. Once this statement is processed, the payroll file is available to the program throughout the entire computation.

It is not clear if the payroll file belongs exclusively to the program during the entire computation, during the period between `OPEN` and `CLOSE` statements, or only during the interval between `READ` and `REWRITE` statements. The answer depends on the operating system in use, and the alternatives are discussed in Chap. 13. The question of exclusive ownership is a concern when multiple programs share access to data. For instance, another transaction, to add up all the salaries in order to prepare a budget, should wait until the entire `raise` transaction is completed if inconsistent results are to be avoided.

The knowledge required for applications programming may be separate from the knowledge to write control language statements; but in order to write effective database systems, both aspects have to be understood. In order to design the entire database application, considerably broader knowledge is required. The analysis of retrieved data requires a different type of expertise than that required for the design, organization, or collection and updating of the data. In order to manage an entire database, a specialist in database administration may be needed, as shown in Chap. 15.

1-4-3 Modularity

As database applications become more complex and better understood, formal separation of functions takes place. We use the term *m*odule for a small but integrated set of programs which deals with some definable subtask and has well-defined interfaces to the other modules of a larger programming system. The user programs which operate on the database will often be *t*ransaction modules. These will use *f*ile-access program modules, and they will themselves be used by higher-level modules, which carry out the user's information-generation requirements. The `raise` program, for example, is a module which implements one transaction type of a larger payroll system and is called upon when the `message` entered says `raise`. The `OPEN`, `READ`, `REWRITE`, etc., statements are calls invoking modules of the file-access system. The `OPEN` is passed directly to the operating system. The data moving statements will also generate some operating system commands. A high degree of modularity is needed in database processing because:

1 Projects are undertaken that are of a size where many people have to work together, and individuals take on more specialized functions.

2 Many projects may make use of a single database or of some part of it, and the specification of the database becomes the sum of many requirements.

3 The information in a database has a value that makes unlimited access by everyone involved unreasonable. In order to provide protection of access to private data, the access processes are controlled by the operating system.

In the example above, all of the program was written by someone who was aware of the application. The control-language statement may have been generated with the assistance of a specialist. The system (invoked by the file access statements `OPEN`, `READ`, and `REWRITE`) was written by someone not aware of the application, at an earlier time and at a different place. The operating system was designed in concert with the computer hardware.

The example is not intended as an indictment or blessing of PL/1 as a programming language. No computer language exists that forces structuring of database programming according to the levels we discussed, although there are a number of languages which make structuring truly difficult.

1-5 DESCRIPTIONS

In order to permit implementation and use of a system by a number of people some documentation will have to exist to describe both the static aspects, the *d*atabase structure, and the procedural aspects, the *c*omputations, which carry out the data transformations.

A picture of the data in terms of files, records, fields, and the relation between items of data contained in these elements is appropriate for the static aspect. Concepts to guide the development of such descriptions are presented in Chap. 7. Such a data-structure definition may take the form of a document to be used by the people who program the file operations on this database. It is even better to materialize the structure definitions in the form of a collection of computer-readable codes, a *s*chema which guides file processes automatically. Much of the content of this book will be concerned with the design and use of such schemas. Examples of schemas are presented in Chap. 8.

The procedural description may be given as a formula, a description of program sections to be executed, or a flowchart. In many commercial programming groups much effort is put into *s*ystems analysis which prepares process descriptions in minute detail for subsequent implementation by coders. It is easy for errors to develop because of communication problems between systems analysts and coders. The analyst may assume that the coders are aware of conditions which are beyond their knowledge. Coders, on the other hand, may program for cases which cannot occur. The determination of critical sections is often a source of difficulty. No formal methods are yet adequate to fully verify correctness for interacting processes. We therefore prefer detailed static data descriptions for the development of databases.

Static versus Procedural Descriptions The static and procedural aspects often occur together when we describe files. For many simple data-processing problems, however, no process descriptions are needed. In those cases the computation is indirectly described by providing two pictures of a file, specifying the form and contents before and after the computation. This method is also applicable to the movement of data between two separate files of a database. The programmer uses the before and after layout charts to write the code. Dispensing with the coding entirely is the intent of *r*eport generators, which accept the description of the source files and a layout of the output report which is desired. Most processing steps are implied and created by the report generator system. The same idea is the basis for many high-level query systems where an information-retrieval request is stated as a formula and its resolution is the task of a retrieval system. Aspects of these approaches will be presented in Chap. 9.

At other times one may find that the data organization is implicitly defined by a listing of the steps of the process which puts the data into the database. Much analysis is needed when new programs want access to data that have only been defined procedurally. The two choices available are implied in the existence of two terms for the description of storage structures: *f*ile organization versus *a*ccess methods.

Special data elements, described and kept in the database, can be used to control program dynamics. A data element of a database could state that a particular file has been completely updated by some computation. Such information can be used to assure a correct execution sequence of programs that share data. Data elements kept with each record may specify that certain records have not been updated yet, maybe because of lack of input data, such as hours worked by some employees. Some computations could be programmed to decide not to proceed at all in that case; other processes would not be impeded unless they require one of the records which have not yet been brought up to date. We refer to data which describe the state of the database for the control of computations as *c*ontrol data.

We see that there is a relationship between the quality or completeness of our description of the data and our ability to specify the program sections which are combined to form data-processing computations.

Flowcharts In order to describe complex processes and data flow, we sometimes have to resort to a graphic description of the dynamics of an algorithm, a *f*lowchart as shown in Fig. 1-7. In these we will distinguish three types of activity:

*C*ontrol flow, the sequence of instruction executed by the processing unit under various conditions. This sequence is described when flowcharts are used to document a programming algorithm.

*C*ontrol-data flow, the flow of data elements whose effect is on the control flow of the computations. Typical examples are counts, references, control switches that indicate whether an operation is permitted, whether an area is full or empty, and so forth. These may be set by one program and used by another program.

*D*ata flow, the flow of the bulk of the coded data as it is moved, merged, and processed to generate information for the benefit of the users of the system. This flow is typically between files, memory, and input or output devices.

The last two activities are sometimes difficult to distinguish, since often some control data are extracted during data processing. In many cases, however, the distinction is helpful. Figure 1-7 indicates the three types of flow in the process of using a library.

With modern techniques of programming, including the use of higher-level languages, well-defined modules, well-structured processes, and documented standard algorithms, the need for description of the program flow has lessened. Detailed descriptions of file and program state are much preferable unless process interactions are unusually complex.

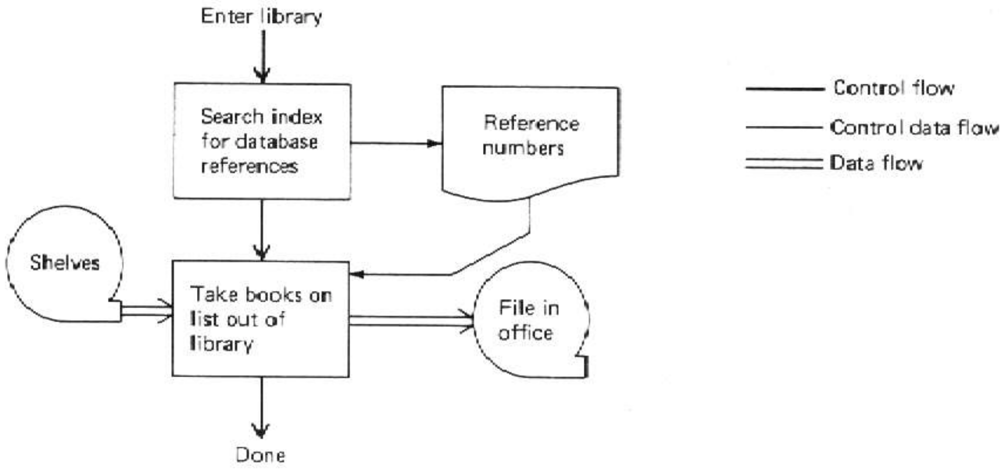

Figure 1-7 Flowchart notation.

1-6 BINDING

Many people in data processing still accept that the contents and the structure of the database have to be fully determined before procedures that operate on it can be written. This assumption defeats one of the major advantages that can be accomplished when we consign processes to computer technology. When designing and implementing a database system, it is desirable that methods be employed that allow development and change, so that we are not wholly dependent on perfect foresight. The fact that all our data and procedures can be manipulated by computation should make it possible to add data and relationships to the database at a later time. The operations involved in such a change may not be trivial, but a system that takes advantage of such growth possibilities can avoid obsolescence.

Whenever a decision is implemented which defines a fixed relationship between data elements, the number of choices available from that point on is reduced. Some knowledge has been used to make processing feasible. The process of fixing data or processing relationships is referred to as *b*inding. At the final stage of data processing all relationships needed to generate information have to be bound. Methods that delay the binding of data definition and structure through the use of schemas are discussed in Chap. 8.

Binding Time The concept of binding is in common use to distinguish compiling, *e*arly binding, versus interpretive, *l*ate binding, processes. The range of binding choices in the database area is considerably greater than in the area of programming. Binding frequently takes place years before a new database system produces its first results.

Binding choices can be illustrated by expanding Example 1-1. Let us assume that an employees classification is not found in a single file (`job_classification`), but that there is a file for each department, and the names of these files are in an array `dep_class_filename`. Statements `s3` to `s5` then become a loop which appends the lists together.

Example 1-3 Using departmental files for the "raise" transaction

```
...
no_employees = 0;
DECLARE dep_emp_list(1000) INTEGER, dep INTEGER,
dep_class_filename(100) CHAR(8) EXTERNAL, filename CHAR(8);
s3:  DO dep = 1 TO no_depts;
dep_emp_list = 0;
filename = dep_class_filename(dep);
OPEN FILE(job_classification) TITLE(filename);
s4:     READ FILE(job_classification) KEY(class) INTO(dep_emp_list);
CLOSE FILE(job_classification);
s4a:    DO i = 1 TO 1000 WHILE dep_emp_list(i) ¬= 0;
no_employees = no_employees + 1;
emp_list(no_employees) = dep_emp_list(i);
s4b:    END;
END;
s5:  PUT SKIP ...
```

This program is bound to the assumption that an employee, including a manager, is listed in only one department file. This constraint will have to be enforced when the departmental employee lists are created. Otherwise, if one manager becomes responsible for two departments, `raise` will give this manager a 21 percent raise instead of the 10 percent raise. Inserting after `s4a` the statements

```
/* check for duplicated employees */
DECLARE id INTEGER;
DO id = 1 TO no_employees;
IF dep_emp_list(i) = emp_list(id) THEN GO TO s4b;
END;
```

releases the binding and resolves the problem at the time that `raise` is executed. An analysis of the program's execution will show that now retrieval processing takes much more computational effort. Late binding can be especially costly because it is done at a critical time, when the answer is needed, and because it has to be done on an element-by-element basis, rather than once and for all when the structure is defined.

Early binding may be chosen because it can provide considerable initial efficiencies. In a database a significant amount of binding occurs when the designer specifies the position of data elements within the files. Later modifications of bound structures will be extremely awkward and often cause inefficient processing when the opposite was intended. When new data types or relationships are to be added, a major reorganization of the file structure and modification of the associated programs may be required.

To avoid the cost of a major change a programmer may decide to put additional data elements into positions originally intended for other purposes. This situation is analogous to the making of machine-language changes in the output of compilers. Such situations are no longer acceptable in programming because of the low cost of recompilation; in the area of file management, the patching of file structures is still common. An example of such a patch is given in Example 1-4.

Example 1-4 A database patch

Given that the records of a personnel file have fields as follows:

```
name,address,sex,salary,military_service,date_employed;
```

and a field for `maternity_leave` is required. The records are used by many programs, even though only a few of them use the `military_service` field. The patch adopted is to use this field for `maternity_leave` whenever the `sex` field indicates a female employee.

```
IF sex='F' THEN maternity_leave = military_service;
```

In this example the decisions made at an earlier binding time determine the implementation of later system features. The patch will cause further problems as progress in sexual equality invalidates the assumptions made.

The lack of proper programming methodology and the pressures to patch rather than to do changes properly are a cause of frustration in many computer shops. The consequences are often cynicism and low productivity of professional personnel. Much of the design guidance presented in this book has as objective that decisions in regard to binding will be made in a knowledgeable manner. Both the users and the system maintainers have to be satisfied. The choice of binding time made by a database designer reflects a compromise between efficiency and flexibility.

1-7 CLASSIFICATION OF OPERATING SYSTEMS

Databases can be established using any of the large variety of operating systems that are available. Operating systems provide a variety of services, of which the major ones can be classified as:

Process scheduling
File-system support
Input and output support
Usage measurement

File systems include programs to maintain directories of files and assure their long-term security and programs invoked by the users to access, that is, read and write, the files during computational tasks. This book will be concerned with access to file systems, but as little as possible with the specific operating system aspects. Some aspects of usage measurement are discussed in Chapter 15, and the other two areas are briefly summarized here. This review is not intended to explain the material but is needed only to establish the vocabulary used throughout this book.

Scheduling is a system process which allocates the resources available to user processes or to jobs containing a sequence of processes that are submitted for execution on a computer. The resources consist of the computing capability provided by the Central Processing Unit (CPU), the core memory which contains the process, the file storage, and the input and output capability including the control of user terminals. The scheduling system will of necessity consume computing resources themselves in direct proportion to their complexity.

1-7-1 Batch Processing

Sequential scheduling of computations, or *b*atch processing, assigns all available resources of the computer to one single process while it is running. Figure 1-8 provides diagrams of this scheduling method and other methods discussed in the remainder of this section. Only a few jobs will be able to use all the available resources effectively when we use the single job-stream batch method.

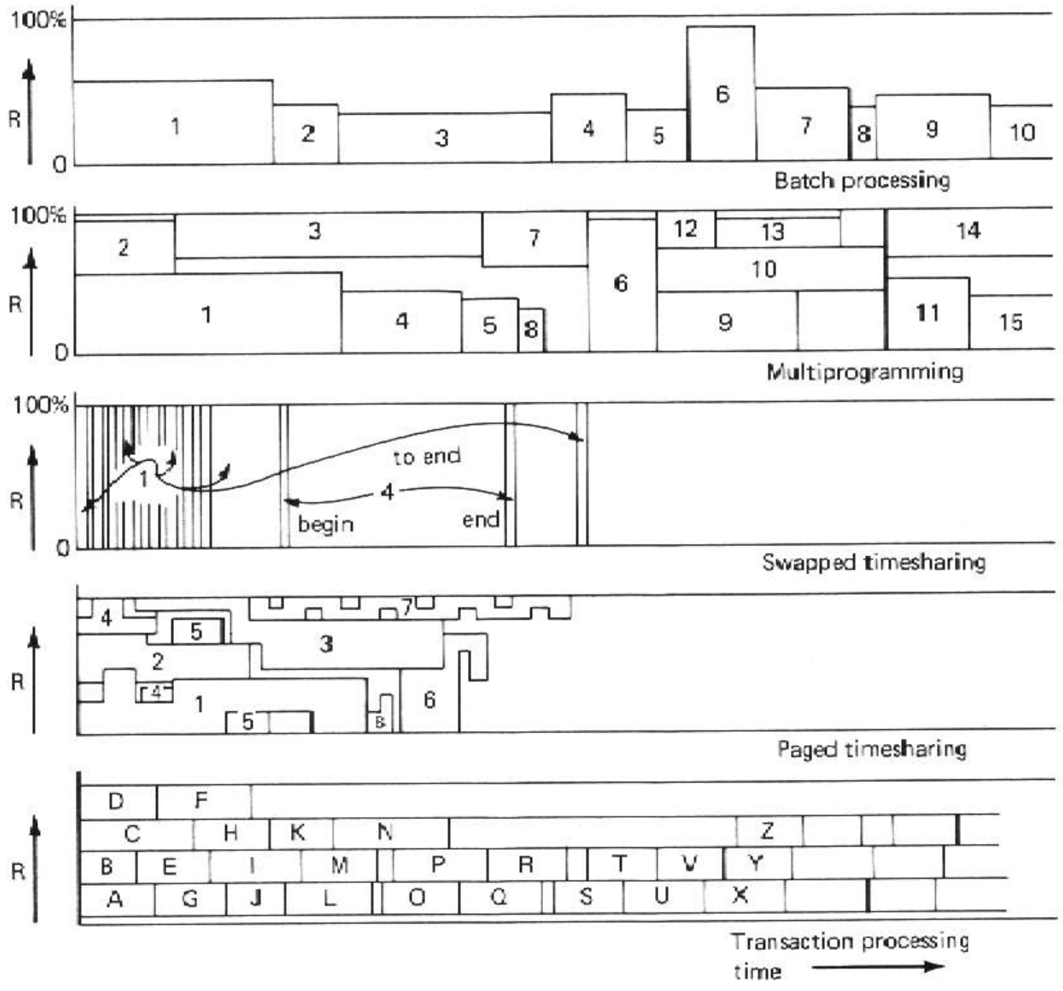

Figure 1-8 Resource allocation for five modes of system operation.

The sketches present the allocation of one resource. Actually all the resources mentioned above participate in the execution of a computation. When use of one resource reaches its limit, the full utilization of the other resources by the computations is prevented.

A typical example of unequal utilization occurs when reading data which are to be processed. Reading may proceed at the maximum speed of the input device, but the central processor (CPU) uses only a few percent of its capacity to perform the immediately required computations on the data. Subsequently a large summary calculation takes place, and the reading device waits for a new set of instructions from the CPU.

1-7-2 Multiprogramming

*M*ultiprogramming attempts to utilize resources more effectively by allowing sharing of the resources among a number of computations according to their current requirements. If a computation cannot use a resource, it will give up its claim so that the operating system can start a process which has been waiting to use the resource. Individual computations will take longer, but if the resources can be distributed better among the computations, the total productivity will be larger. Some amount of processor capability will be taken from the total available in order to effect the switching between computations.

1-7-3 Timesharing

*T*imesharing splits the computations into slices in an attempt to strike a balance between the productivity of users and the system. Note that the users which interact on-line, at a terminal, are also a critical resource. They perform unfortunately very slowly but are quite demanding when they require computation. Fast response for on-line users is provided by limiting the competing computational slices to a small fraction of a second by means of a timer-driven interrupt.

New processes that users want to initiate are put in a queue with the processes that have been interrupted, and are executed in turn. The time between successive turns depends on the number of active users, the size of the slices, and the time it takes to switch from one slice to the next. A very critical resource in timesharing systems with many simultaneous users is the core memory, so that during inactive periods the users' processes may be "swapped" out of memory and kept on disk or drum storage. This storage space is generally not part of the storage space available to data files.

Timeshared use of CPU and core provides effective resource utilization when the computations are short. Many database operations deal with users at terminals, and timesharing permits the process to be inactive while the user responds. A computation which is longer than one slice will take longer to complete than it would have in batch mode, since all other active users will get their slices interposed. This computation hence ties up resources for a longer time. A database user often has files assigned exclusively to the computation, and these are now not available to the other users. The active user is well served, but the system incurs costs of slicing and swapping, and the periods of file assignment are longer, which may hold up others.

1-7-4 Paging

Since a single slice of a timeshared computation rarely makes use of all the available core memory a further division of the memory into pages improves sharing of this resource and reduces swapping volume. Pages for programs used by multiple users, such as database systems, may be shared. The number of pages actually required will vary from slice to slice. The pages are generally brought in on demand. Pages which show heavy recent usage are kept in core memory if possible. *P*aging capability can also aid the allocation process when multiprogramming in general.

1-7-5 Transaction Processing

The most important operating system approach for interactive databases is called *t*ransaction processing. This method assumes that the computations are naturally small, in terms of both space and time. The database system software is assumed to be already available, so that the users' computations consist mainly of calls to the database for reading and writing, and a limited amount of numeric and logical computing. Such a computation is called a *t*ransaction. Transactions are started as soon as possible after the user requests them and are permitted to run as long as necessary to fulfill their computational requirements. When a transaction comes to a point where it has to wait for a response from terminals or for data from files, it may yield to the system to enable another transaction process to be started or resumed. At times a transaction may avoid yielding to prevent release of a file which is not in a well-defined state. A transaction still ties up resources like files that have been exclusively assigned to it, but its total computation time will not have been lengthened by slicing and swapping. When a transaction is finished, it will inform the system, which can then free all allocated resources. Example 1-1 was written to show a typical transaction program.

A transaction is also formally defined as a computation which, after completion, leaves the database in a well-defined state, or at least not in a worse state than before. A failure during the execution of a transaction requires that the database be restored to its original state, Sec. 11-3 presents the required mechanisms. If only well-behaved transactions operate on a database, the integrity of the database will always be maintained. For example, if a transaction is to update a `salary` and increment the `budget` amount in a database, it will do either both or neither, so that the sum of the `salary` amounts will always match the `budget`.

A transaction-oriented environment often exercises close control over the computations to prevent one faulty computation from locking up the entire system. Limits are applied to program size, execution time, number of file accesses, and locking actions for exclusive use of files. This control is applied when the computation is specified or bound. At execution time little checking is performed and hence the operating system can be considerably less complex. A computation which grossly exceeds reasonable limits of resource consumption can be abruptly terminated. Typical design constraints are 25 calls to the database system, 2 seconds total execution time, and 25 pages of memory [Inmon^{79}]. Many database actions can be handled most effectively in a transaction processing environment.

1-7-6 Sharing and Distribution

Any form of resource sharing is associated with a considerable overhead for operation and management. If there is no significant benefit to the use of shared resources, a dedicated system that is of just adequate size is preferable. Such a system will, of course, not be able to cope well with irregular high-intensity demands. Mixed systems, having local and distributed capability as well as access to central shared resources, are of great interest to system designers. A distributed system will have data and processing tasks that are of mainly local importance executed at a local computer. Requirements for data and processing that cannot be carried out locally will generate requests to remote computers. A logically remote computer may be another local computer or a specialized device attached to a central computer. Design issues of distributed computation will be covered specifically in Secs. 2-4-4, 5-3-3, 5-4-4, 7, 9-7-1, 11-3-1, 11-3-4, 11-5-4, 13-2-6.

1-8 APPLICATIONS

Application areas that employ database systems currently include:

Manufacturing with inventory management, Bills-of-Material processing, and production equipment scheduling

Government at all levels with records on taxpaying individuals and property

Financial institutions with lists of individual accounts, assets, and convertibility of funds

Service industries with lists of service capabilities and allocation schedules

Medical services with patient records, disease histories, problem classification, treatment effectiveness data

Economic models with production and consumption data for allocation and planning

Scientific research with collections of previously gathered data used to determine future research directions

Offices which are automating their document management

Libraries cataloging abstracts and indexes of their holdings

The background and references section cites some examples from each category. It will be useful to gain some understanding of the uses to which these systems are put, in order to obtain a perspective of the problems discussed. We note that information systems based on data in text form, often used for automated library support, tend not to use structured data but depend greatly on searches for keywords in titles, abstracts, and author lists. This book deals mainly with structured data, so that it is less relevant to applications that are based on content analysis.

The management of inventories in manufacturing, particularly where products consist of many parts and subassemblies, has been one of the earliest and most productive areas of database systems. A particular objective of these *B*ills-of-Material systems is the proper accounting and order scheduling of parts that are components of diverse end products and hence particularly critical. We will draw many examples from personnel files, since their contents and function are frequently self-evident. More interesting and potentially more significant applications are in food and energy resource planning.

Definition of an Application of a Database In any application of computers, the purpose of the effort should be spelled out in detail. Only then can one identify and judge the relevance of the data elements contained in the files, the processes required to manipulate the data, and the file structures which will enable the processes to be effective. Knowledge of the environment is needed to determine the value of the produced information. We also have to know the response time intervals required to ensure timely delivery of the system outputs. All these items provide the data for system analysis and the subsequent selection of the hardware. The dynamics of the application environment will provide useful information to decide the degree of formalization of the data organization and the corresponding binding times.

A statement of objectives and constraints, which specifies the goal and environment of the application, is an essential part of any project carried out in practice. Such a statement should also be presented with any project carried out as an exercise within the scope of this book. A frequent error in the formulation of objectives is the inclusion of the method or technology to be used as part of the objective. A proper place for such an account is the section which describes the results of the system analysis.

Objective statements which imply that the desire to *computerize* was the reason for development of a database should send a shiver down the spine of any computing professional. If the choice of methods or technology has been intentionally limited, a separate section of the objective statement should clarify such conditions.

1-9 REVIEW

We will treat files and databases which use files as an extension of the storage capability of a computer system. Their data contents remain under the control of the system. Since the user will never directly see the stored data, they can be organized and formatted by the system. This allows optimal use of resources. New information which enters the system or results that leave the system are handled via a separate input or output mechanism. Within the data storage organization, we distinguish two levels: the file system and the database system. The file system operates without knowledge of the contents of the data, whereas the database system will select data on the basis of the expected contents.

1-9-1 Overview of the Material to Be Presented

The chapters following will discuss the level of hardware and file systems, and we will continue with database structures only in Chap. 7. Chapter 2 will describe available hardware and reduce its complexities to a small number of quantifiable parameters. In Chapter 3 we will develop evaluation formulas based on these parameters which can be used to analyze appropriate basic file organizations. Chapter 4 will introduce more complex file designs. The design process, beginning from user requirements, is presented in Chap. 5. A central measure of goodness of a design is the overall economic utility of the services provided by the file system. Some primary elements of positive performance are economy of storage, speed of access to information, and speed and ease of update of data.

The database topics begin with database models, in Chap. 7. The description of models by schemas, the structure of database management systems, and their information-retrieval methods follow through Chap. 10. Related factors required for a successful database system include flexibility, as well as reliability, privacy, and integrity. These are discussed in Chaps. 11, 12, and 13.

This breakdown of the database problem into manageable subjects makes an analytic approach possible. A synthesis of preceding material is attempted in Chaps. 5 and 15. Chapters 6 and 14 discuss areas that are not of primary concern in this book but which contribute material important to database system analysis.

BACKGROUND AND REFERENCES

The structure of information systems has emerged slowly. A prophetic paper, published originally in 1945 (Vannevar Bush, reprinted in Kochen[67]), predicted machines for information retrieval, and in 1948 Shannon[62] published a paper defining the concept of information in a formal sense. Storage devices capable for databases became available in the early sixties, and some early systems such as MEDLARS for medical documentation (Katter[75]), SABRE for airline reservations (Desmonde[64]), and other systems for governmental planning (Clark[67]) were initiated. Expectations of effective use were tempered by disappointments and reassessments (Aron[69], Lucas[75], and in Riley[81]).

In parallel a scientific approach to database problems began to be concerned with the concepts of data and data structure (Steel[64], Mealy[67], Bachman[72], Senko[75,77], Kent[78], Lucas[81], and Tsichritzis[82]), and this, influenced by the developing ideas of structure in programming (Dykstra[71], Dahl[72], Parnas[72]), has led to an approximate agreement of structural levels; a taxonomy is given in Booth[73]. Lindgreen[74] has suggested a primary stress on the information-management level in database design.

The SHARE and GUIDE organizations of IBM computer users collaborated on a study of database requirements; a SHARE database conference (Jardine[74]) summarized effectively the state of the art in applications and systems. Another product of these user groups is a projection of growth and problems in the eighties (Dolotta[76]). Textbooks are now available which describe (Sanders[74], Kroenke[78], Atre[80]) or analyze (Davis[74], Everest[82]) management-oriented applications.

Applications The popular computing literature (DATAMATION, etc.) as well as specialized journals in every field of science carry regularly articles on automation which often involve databases. In the management area the HARVARD BUSINESS REVIEW carries regularly relevant material. Murdock[75] has collected a number of relevant early papers.

Databases are found in local (Mantey[75]) and national (Mendelsohn[71], Sibley[74], and Ghosh in Furtado[79]) government. The directions in office automation are surveyed in Ellis[80]. The use of computer databases in banks (Cannellys[67]), insurance (Allen[68]), utilities (Liu[68]), for sales support (Lucas[75]), in manufacturing of aircraft (Fulton[80]), computers (Mallmann[80]), and for the general Bill-of-Materials problems (Bachman[74]), as well as fashion design (Kunii[75]) has been described.

Databases have been applied in agriculture (Kendrik^{80}), the social sciences (Bisco^{70}), environmental control (Ouelette^{75}), and to the study of history (Schneider^{75}). The use of computers in information systems oriented toward bibliographic data is described in books by Salton^{75} and Lancaster^{79}. Chemical data (Anzelmo^{71}) have long been cataloged, and applications in clinical medicine have a long history (Allen^{66}, Davis^{70}, Yourdon^{72}, Weyl^{75}, Barnett^{79}, Wiederhold^{81}). Motivation for scientific databases is provided by Lide^{81}. This list can be extended further by use of references in the works cited.

Operating Systems Many of the system problems faced in the database area were first encountered in the design of operating systems. Textbooks in this area are Brinch Hansen^{73A}, Tsichritzsis^{74}, and Haberman^{76}. The ACM COMPUTING SURVEYS is an excellent source for analytic material and further references. Transaction-processing systems have received less attention until recently; they are described in Martin^{67}, Yourdon^{72}, Davis^{74}, and Holt^{75}. Clark^{66} describes the data management in IBM-OS, using a large variety of arrows to distinguish control, the various types of control data, and data flow. Distributed systems are covered by Tanenbaum^{81} and Davies^{82}.

Basic algorithms, such as sorting, table look-ups, and the manipulation of data structures as lists and trees in core memory will have been encountered by most readers. An introductory reference is Stone^{72} and a thorough analysis of most techniques is found in Knuth^{73F,S}. COMPUTING SURVEYS is also a useful resource here.

EXERCISES

1 Write a statement of objective for one of the applications mentioned in Sec. 1-8. Select a subject area with which you are familiar and in which you are interested. Many other exercises in this book can be based on the same application.

2 Provide a list of data elements which you think would be included in the application presented above, and justify their value with a short sentence for each.

3 Identify which elements are likely to have a dynamic relationship or function and hence will cause definitional problems.

4 Inspect a file-oriented program that you have access to and select statements which typify some of the classifications given in Table 1-1.

5 Rewrite the control-language statement presented in Example 1-2 in the language of the computer system you have available.

6 Write a query to the database with the patch shown in Example 1-4 which would produce an error.

7 Discuss the desired binding time for elements of a specific file-oriented program.

8 Classify the operating system you have available according to the categories of Sec. 1-7. List aspects of your system which you consider especially good or poor for database-system implementation.

9 Give an example of a very large database as defined in Sec. 1-1-1.