Chapter 9

# Data Management Systems

## 0 PREVIEW

Bibliographic information retrieval systems have a good deal in common with other information systems, such as question-answering and data base management systems. Information retrieval and question answering share a common interest in natural language processing. The bond between information retrieval and data base management is provided by the common storage structures—both types of systems use inverted file directories—and the common processing strategies. The design and operations of data base management systems are examined in this chapter.

The main characteristics of the several types of information systems are summarized first, followed by a description of the principal features of data base management systems. The three well-known data base models known as the relational, hierarchical, and network models are introduced. Certain query-processing strategies are then examined that may help in extracting relevant information from the data base in answer to incoming user queries. Finally, the use of data base management systems for the preservation of data quality is described, including the data security and integrity provisions that are often used, and the restart and recovery techniques that can be invoked when system failures occur.

The chapter closes with a brief discussion of the problems arising when several users are allowed to use a data base management system concurrently, and when the stored files are distributed among several cooperating, remotely located sites.

## 1 TYPES OF INFORMATION SYSTEMS

### A Information Retrieval and Question Answering

The previous chapters dealt primarily with document retrieval systems, designed to retrieve bibliographic references in answer to queries submitted by the user population. In that context, the term "information" receives the specialized interpretation of data derived from texts or bibliographic items. Ideally, information retrieval systems can retrieve many different kinds of products for the user, including full bibliographic citations for the retrieved items, abstracts, lists of assigned keywords, contract and project numbers, references to other documents, or even full document texts. The expectation is that many of the retrieved documents will contain elements that directly respond to the user's information needs. Indeed, the success of the existing bibliographic retrieval systems indicates that these systems are used as an important source of information by many users.

Unfortunately the existing systems are restricted in many ways: most importantly, users are not given direct facts in answer to any question, but rather obtain information that may lead to answers; second, the existing systems confine themselves to the retrieval of document citations or document excerpts only. That is, following the document retrieval step, most systems do not provide additional automatic aids to process or interpret the retrieved data. Instead, the users are left entirely to their own devices when it comes to interpreting and utilizing the retrieved information.

In actual fact, many user queries require data from a variety of sources. This may include normal bibliographic references and also nonbibliographic data of many kinds such as product information, marketing data, technology forecasts, business data, information about federal rules and regulations, legal decisions, and industrial or manufacturing summaries. As it happens, data bases covering these and many other classes of information already exist in the field, but the methods needed to coordinate the various data sources and to provide simple and compatible accessing procedures are not available [1].

It is likely that most information system users are interested in facts, rather than in documents that need to be studied before the needed information can be extracted. One possibility for simplifying the user's task in this respect consists in breaking up each document into small pieces—for example, individual chapters, pages, paragraphs, or even sentences—and in retrieving the individual pieces rather than full documents only. The work on *passage retrieval* that was briefly mentioned in Chapter 7 constitutes a step in that direction. If this approach were to be effective, it might become necessary to analyze the document content sufficiently deeply to be able to characterize the various text

portions and to point out differences and relationships between them. Furthermore, the full document texts must also be stored to provide the needed passage retrieval capability.

Passage retrieval is attractive because the basic retrieval process used for full document retrieval, including the file organization, query formulation, and user-system interaction methods, may remain unchanged. On the other hand, the information input and analysis tasks are now vastly more complicated, and the resulting system may be too expensive to use. In the final analysis, passage retrieval may constitute a step in the right direction so far as the user is concerned, but the ability to answer questions by citing specific facts included in the data base is still lacking.

The *question-answering* systems mentioned in Chapter 7 are specifically designed to provide direct answers to questions; however, the earlier discussion indicated that the construction of unrestricted question-answering systems was not a likely prospect for the visible future. Not all relevant facts could possibly be stored in any particular system, nor could all the needed relationships between stored facts be identified in advance. Necessarily then, a viable question-answering system would operate with a restricted data base using procedures for refining the available information as needed by deducing new facts from already available information and supplementing the description of incompletely specified items by extraneous knowledge.

Simple deductive systems can be built that are capable of answering many questions for which the direct answer is not initially contained in the data base. Thus, given the facts that "Joe Smith is an employee" and that "All employees are at least 18 years old," an answer generating system could be built to handle correctly a question such as "Is Joe Smith at least 18 years old?" However, more complicated situations are easy to imagine where even sophisticated systems would soon become stymied. Consider, for example, a knowledge base consisting of the following sentences:

Joe Smith is an employee.
87 percent of all employees are union members.
Joe Smith voted in the last election.

It is obvious that the question "Is Joe Smith a union member?" cannot be answered unequivocally. However, if one knew that Smith's job classification entails compulsory union membership, an answer might be provided. Alternatively, if it were known that a union election had recently been held, the system could answer "yes" with reasonable confidence, assuming that information was available to the system about labor unions and elections and the interactions between them.

In the earlier examples, potential ambiguities—for example, the distinction between political elections and union elections—were deliberately disregarded. Unfortunately, linguistic ambiguities are unavoidable and effective disambiguation or interpretation methods are not readily available. It is not surprising then that practical question-answering systems have been based on

highly simplified world pictures where only a small number of facts are used, the description of all items is unambiguous and complete, and the relationships between the items are prespecified. In such situations, direct answers can be provided to a wide variety of questions. The so-called data base management systems are the most widely used and most successful systems in this class. They are examined in the remainder of this chapter.

### B Data Management Systems

Data management systems exhibit many similarities with standard information retrieval systems: a stored data base is generated and maintained, and information searches are conducted resulting in the retrieval of portions of the stored data in answer to user queries. The distinguishing characteristic in data management systems is the definite structure of the stored information: instead of dealing with natural language texts as in document retrieval, or with arbitrary facts as in question answering, uncertainties and ambiguities are eliminated by using only information items whose structure is severely restricted and completely specified. In particular, data management systems normally process files of data described by a small set of prespecified *attributes*. For example, a file of personnel records may be identified by the names of the people involved, the addresses, the age of each person, the job classifications, and the yearly salaries for each person. Each attribute may be expected to carry only one of a small number of specific *values*—for example, the age attribute may range in value from 18 to 65 for the employees in question. A particular record in the file can then be described by citing the specific values taken on by the attributes for that particular record. Thus the person named Smith might be specified as

(NAME = Smith; ADDRESS = 110 Main St.; AGE = 25;
JOB CLASS = 123; SALARY = 19,500)

It is obvious that in such an environment many of the most difficult retrieval problems can be bypassed, including in particular the choice of the indexing language, and the content analysis and indexing operations themselves. Furthermore, by assuming that each item is completely and unambiguously described by the chosen *attribute values* (the values of the available attributes), the type of processing to be carried out is also limited. In particular, one would want to compare the attribute values specified in a user query with the attribute values characterizing the stored data in the hope of retrieving all records whose attribute values match a given set of query attribute values, or all records whose attribute values fall within a certain range (for example, all persons whose age is between 25 and 35). In addition, data base management systems may be designed to perform simple numeric processing tasks, such as determining counts of the number of records obeying a particular specification—for example, the number of books published in a given year, or by a particular publisher; or obtaining averages, such as the average cost of the books published in a given year.

It is easy to see that systems exhibiting the capabilities outlined above fulfill many immediate needs in a variety of different circumstances: it becomes possible to choose rapidly from a list of employees all those qualified for a given job, or from a set of military personnel all those currently present in a given geographic area and able to take on a particular task. Similarly, one can obtain manufacturing components from a parts inventory obeying particular specifications, or identify mail items that need to be routed in a given direction, or airline flights scheduled for specified routes.

Systems capable of performing tasks of that kind are known as *data base management systems* (DBMS). They are enjoying great popularity and widespread use at the present time [2–5]. Certain data base systems have been adapted for specialized use by particular user classes. *Management information systems* (MIS) are essentially data base management systems that include additional computational features thought to be of interest to managers. Thus, given sales figures covering certain periods of time, a trend analysis could be carried out using an MIS that would project these figures into the future [6–9].

Management information systems extend the processing capabilities of standard data base management systems in a direction useful to management personnel. In the so-called *decision support systems* (DSS) the aim has been to extend the basic file processing routines by adding capabilities from other areas of computer application including, for example, graphic data processing, pattern matching, and artificial intelligence. The aim in this case is to provide in an easily accessible form the large diversity of resources normally thought to be needed for decision making purposes.

Consider as an example the person responsible for the design of school bus routes in a given community. Such a person requires information about the number of school-age children living in particular areas, the number and type of available school buses, the distances between various points on prospective routes, and the schedules followed in various schools. In addition to this standard information that could be provided by conventional data base management systems, other less tangible, but potentially equally important data might also be used to solve the routing problem. This could include information about peak traffic loads, icy road conditions, and traffic light placement. Information might also be useful concerning the composition of voting districts in the community, and the expected opposition from the parents to some of the proposed bus routes. Once a particular routing is tentatively chosen, the ideal decision support system would provide simulation capabilities, complete with graphic output displays, through which the progress of the various buses could be monitored in a dry-run situation in the hope of detecting problem areas before final decisions are actually reached.

Completely flexible decision support systems are not currently used in practice, but beginnings have been made in many application areas including medicine, banking, and airline management [10–15]. Most existing systems lack the ability to use information from many different sources, and in addition provide only limited processing aids. The chart of Fig. 9-1 exhibits relation-

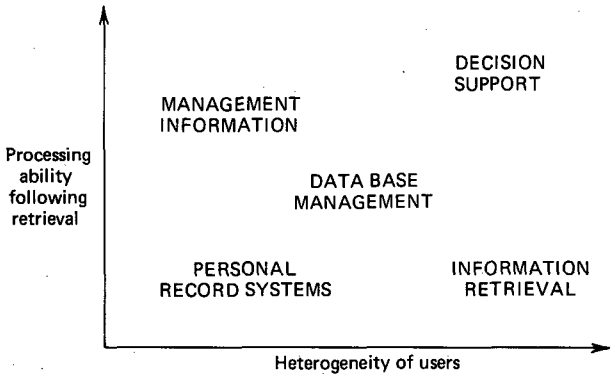



**Figure 9-1** Characteristics of several information systems.

ships between various types of information systems based on the type of user that is likely to be attracted to the system and the processing ability following an initial retrieval action. Whereas normal bibliographic retrieval systems must serve many different user types but provide little processing support beyond retrieval, the reverse is true of management information systems. Decision support systems are demanding on both accounts, which explains why such systems are not yet widely used.

The data base management systems, by constrast, are widely used and have achieved great sophistication with considerable theoretical background. They share the main file structures and search methodologies with bibliographic retrieval, and they provide processing facilities that are also of interest in the bibliographic retrieval environment. The data base management systems will be studied in the remainder of this chapter.

## 2 THE STRUCTURE OF DATA BASE MANAGEMENT SYSTEMS

### A Basic Concepts

It was mentioned earlier that many of the harder problems arising in information retrieval are avoided in data base management because of the simpler, more structured nature of the data being manipulated. On the other hand, the missing content analysis and other text processing operations are replaced in the DBMS by a greater concern for *user friendliness* and *data quality*, and by the inclusion of sophisticated methods for processing and updating the data base. In a typical bibliographic retrieval system, ordinary users are not allowed to change the stored data. This contrasts with a DBMS, where much of the design complexity is associated with the data manipulations and the available methods for introducing changes to the stored information. Furthermore, data base systems are designed to be used by nonexperts, and great care is taken to insulate the users from the details of the processing system, and by extension to ensure that users cannot interfere with each other. This implies that in addition to providing the normal search and retrieval capability, the system is also

charged with various access control tasks and with the preservation of data integrity and correctness.

A data base management system exhibits three main components: first a *data base* consisting of a variety of files broken down into individual records together with the accessing and file maintenance operations performed on the files; second, a *communications* system that provides the interface between the system users and the automatic system, including also the message handling, editing, and output display functions; and finally, a *transaction management* system that is charged with the scheduling of jobs received from the various users, the access control to the files, the handling of concurrent operations for tasks that may be carried out simultaneously, and the implementation of restart and recovery procedures following system failure.

The following features, which are present in most data base management implementations, render such systems attractive to the user populations [16–21]:

**1** The user application programs can normally be written without detailed know-how of the methods used to represent and organize the data in storage, and without regard to the particular procedures used to access the data. In other words the user programs exhibit *physical data independence* because they do not depend on the actual implementation methods used by the data base management system.

**2** Attempts are also made to provide *logical data independence* in the sense that the user programs may effectively be independent of the internal structure of the data base objects and of the various relationships which may be defined between objects.

**3** High-level language facilities are often provided to help the users in submitting queries to the system and in specifying the required file processing operations.

**4** Data quality may be maintained by having the user supply validation statements concerning the characteristics of files and data items—for example, a specification of the range of values that may be acceptable for specific attributes—and letting the system automatically provide error checks designed to verify the validation assertions.

**5** Restart facilities are provided charged with maintaining correct processing sequences when failures occur by having the system record the status of current conditions at appropriate intervals and using backup facilities to resume the programs after corrections based on the recorded status information.

**6** *Security* provisions may be incorporated in the form of privacy transformations and access controls to ensure that the stored information is not misused or misappropriated.

To translate the high-level process specifications into specific machine operations and to provide the data independence mentioned earlier, it is necessary to store a detailed description of the file structures actually used and of the structure of the individual records included in each file. This description of the data and file representations is known as a *schema*. Various schema types may

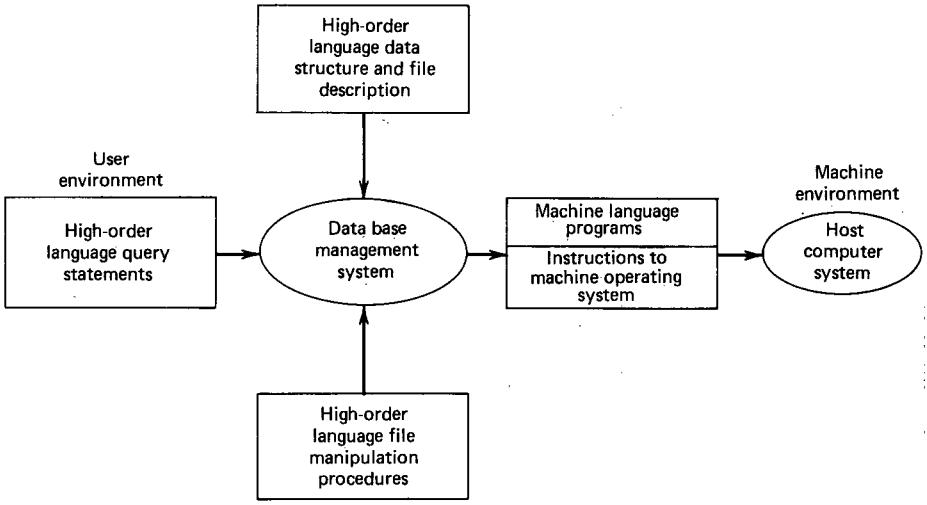



**Figure 9-2** Environment for typical data base management system. (*Adapted from reference 21.*)

be distinguished, including the *external* or *user schema* utilized by individual customers to specify the structure of their own data, a *conceptual schema* which is common for all users and serves to supply the required data independence, and finally a machine or *internal schema* used to represent the actual physical data structures that are needed by the system to carry out the file processing operations.

Normally high-level "data description" languages are provided by the data base management system to help the system users in specifying the individual user schemas. Once a schema is specified by the user, the translation of the external schema into the internal form used by the programs is then completely left to the system. In performing the program setup and translation operations,

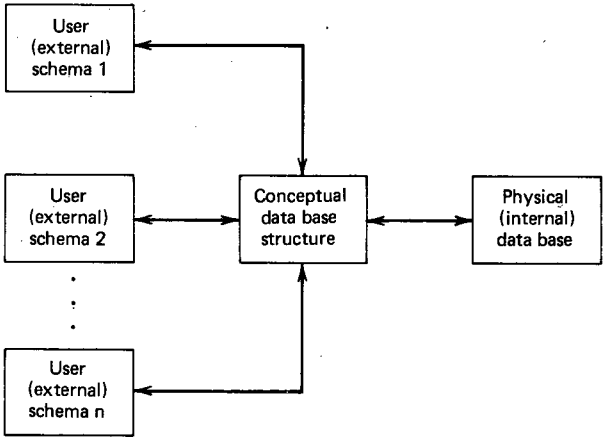



**Figure 9-3** Levels of data base representation.

the system normally uses a stored *data dictionary* which records the description of all objects in the system, including the files and corresponding file schemas, the user terminals and their identifications and characteristics, the individual users and their status and file access authorizations, and the particular transactions or file manipulations that actually need to be carried out.

The structure of a typical data base management system from the user's viewpoint is shown in Fig. 9-2, and the relationship between the various file descriptions is summarized in Fig. 9-3.

### B Structure of Information Items

To describe the individual operations of a data base management system, it is necessary to consider in more detail the structure of the information which needs to be manipulated. The utilization of special information structures constitutes in fact a principal characteristic distinguishing conventional bibliographic retrieval from data base management: specifically, the information used in bibliographic information retrieval is normally assumed to be unstructured, and the data elements are often self-describing in the sense that it is easy to distinguish author names from publishers or document titles. In a data base management environment the individual record elements are not self-describing, and a definite record structure is either assumed or implied.

Consider as an example the components of a typical bibliographic record used by the INSPEC retrieval service and represented in Table 9-1. This record includes three types of identifiers, including the objective terms such as author, journal, or page numbers; the content identifiers consisting in Table 9-1 of freely chosen terms and terms extracted from a controlled vocabulary arrangement; and finally the title and abstract of the document. In addition to the structure implied by this list, a great deal of syntactic and semantic structure is inherent in the language of the title and abstract; furthermore, the chronological relationship which exists between the actual article and the bibliographic references attached to the item can be used to define explicit hierarchical relations between articles.

In actual fact, the linguistic structure built into a bibliographic record is not normally used in information retrieval. The terms and keywords are nearly al-

**Table 9-1 Components of Bibliographic Record Used by INSPEC Service**

| | |
|---|---|
| Title of item | Number of references |
| Author name(s) | Classification codes |
| Author affiliation(s) | Controlled index terms |
| Publication identification | Free index terms |
| Volume, issue, or part number | Type of article or treatment (for example, bibliography or literature survey, general or review article) |
| Date of publication | |
| Page numbers | |
| Language of publication | |
| Full text of abstract | |

ways assumed to be independent of each other, and the language analysis methods needed to exhibit the linguistic structure of title and abstract are expensive to carry out and not sufficiently reliable for practical use. In data base management the situation is very different because structure is deliberately built into the records and used in retrieval.

To describe the structure of the information items in a data base system, it is convenient to distinguish the *entities* being manipulated from the *attributes* characterizing the entities. The normal distinction between the two depends on the role they play in a particular retrieval environment: entities are data objects that have an independent life of their own, and as such they constitute the elements of principal interest for the user of the retrieval system; attributes, on the

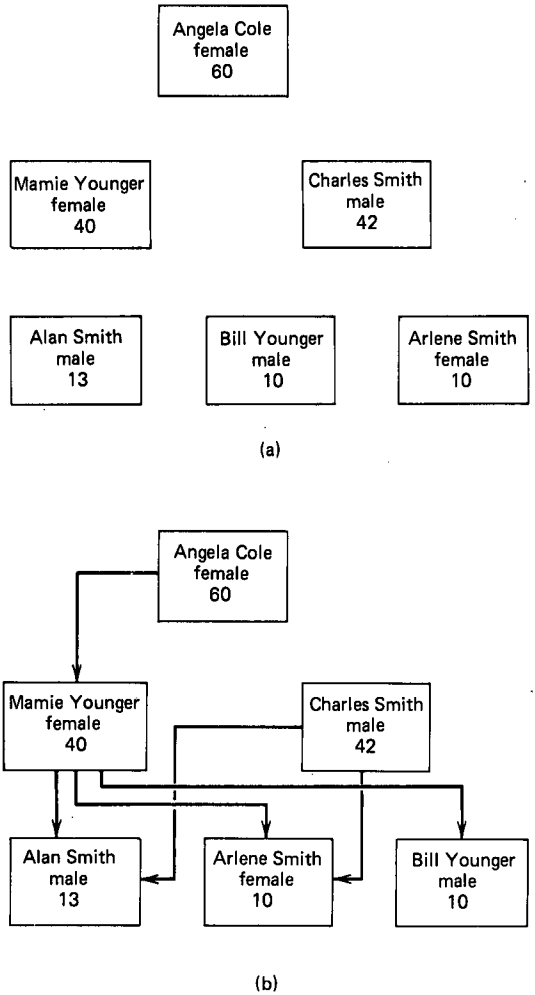



**Figure 9-4** Sample PERSON data base. (a) PERSON entities identified by attributes NAME, SEX, AGE. (b) PERSON entities with specified parent-child relations.

other hand, exist only because they are assigned as identifiers to the defined entities. The choice of appropriate entities and attributes to be used in a given data base system is a matter of substantial difficulty for the data base designer, and data elements used as attributes in one context may well be defined as entities in another. For example, the entity STUDENT may be characterized by the attribute TEACHER who is responsible for a particular class in which the student is registered; on the other hand, a TEACHER may constitute a separate entity identified by separate attributes. A typical entity class consisting of persons identified by attributes NAME, SEX, and AGE is shown in Fig. 9-4a. The actual values of the attributes characterizing the individual persons are included in Fig. 9-4.

It is customary in DBMS environments to define two types of *relationships* between entities, including the generic or *hierarchical inclusion* relationships where some entities are general or governing types of entities and others are narrower and thus dependent on the more general ones. In that case, a hierarchical, tree-type arrangement can be used to represent the relationships such that the governing entities are placed above the corresponding descendants in a two-dimensional graph. Typical hierarchical relationships are whole/part and parent/child relationships. The parent/child relationships defined for the data base of Fig. 9-4a are identified in Fig. 9-4b by vertical pointers (arrows) to the hierarchically inferior child.

The other type of relationship between entities includes the nonhierarchical relationships for which an unambiguous governing-descendant characterization cannot be used. Many different types of nonhierarchical relationships may be used in particular cases, including, for example, cause/effect relationships (as between "poison" and "death") and actor/acted upon relationships (as between "pilot" and "plane"). To represent nonhierarchical relationships, *networks* instead of tree structures must normally be used where the pointers once again represent connections between related entities. A typical example is shown in Fig. 9-5 for the entity types PILOTS, EMPLOYEES, PLANES, and DEPARTURES. The pilots have a flight capability, being certified to fly certain planes; at the same time the pilots are also employees of an airline company. The airline assigns certain employees to particular departures, each departure representing a flight leaving at a certain time and date with a particular routing. Obviously a plane must also be assigned to each departure. Relationships such as those illustrated in Fig. 9-5 cannot be represented by a hierarchical tree arrangement [22,23].

For efficient implementation, relationships between entities are often classified according to how many entities of one type may be associated with the entities of another type. Thus there may be one-to-one (1:1) relationships as between PLANES and DEPARTURES, a particular unique plane being normally assigned to each given departure. Alternatively the relationship may be many-to-one (n:1) as between STUDENTS and CLASSROOM, where many students are assigned to a given class, or many-to-many (n:m) as between PROJECTS and EMPLOYEES, where one project occupies many employees but one employee may be assigned to many projects.

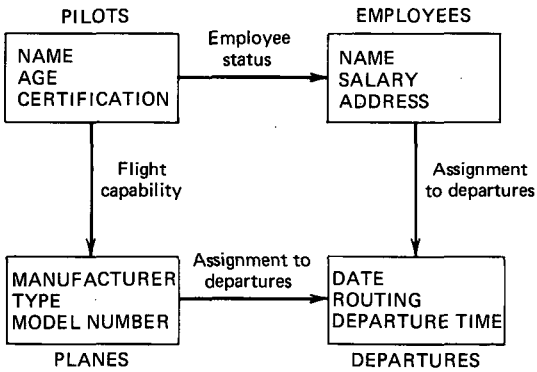



**Figure 9-5** Typical nonhierarchical relationship in data base.

In the data base, the relationships between entities are normally represented by *pointers* included as identifiers for a given entity, and representing the addresses of the related entities. Thus in the example of Fig. 9-4, the record for Mamie Younger includes three pointers, one each to specify the addresses used to store the record for Alan Smith, Arlene Smith, and Bill Younger. The use of physical address pointers simplifies the handling of information requests in which entity relationships play a role, such as, for example, "Give the name and age of all descendants of Mamie Younger" or "Are there any pilots less than 30 years old that earn over $50,000?"

Only relationships between entities have been discussed up to now. However, explicit or implied relationships can also be recognized between attributes of entities. Thus semantic relations of many kinds are definable between attributes, such as, for example, the relationship between a person's age and his occupation (elementary school children are normally less than 15 years old, whereas plumbers or welders are over 15). Another type of relationship between attributes depends on whether the values of certain attributes may or may not be the same for distinct records of a collection. Consider, for example, a personnel file and assume that the social security (SS) number uniquely identifies each person in the file. In these circumstances, it is clear that if two records happen to be identified by the same SS number, then necessarily the age attribute must also be the same, because the two records then represent the same person. Technically speaking, one says that a *functional dependency* exists between the social security number of a person and the age of that person, or the SS number functionally determines the age.

Semantic relationships and functional dependencies between attributes are usable in file design and to verify the correctness of the data, as will be seen later.

### *C The Relational Data Base Model

Depending on the kind of relationships that are explicitly used in the data manipulations, it is customary to distinguish three abstract data base models, known, respectively as the relational, the hierarchical, and the network

models. As the names indicate, the hierarchical and network retrieval models are based, respectively, on hierarchical and network types of relationships between the entities. In the *relational data base model* no explicit relationships are defined between entity types, and no pointers are actually stored. Instead, the relationships between the entities must be derived before they can be used in answering the queries. Because of the absence of directly specified relationships, the relational model is conceptually easy to deal with, but the processing needed to retrieve the records wanted in response to a given query may be quite complex [24,25].

A *relation* is simply a table representing the records in a given file in such a way that each record, also known as a *tuple*, is identified by an ordered set of attribute values. In other words, each record corresponds to a particular row of the table, and all rows are assumed to be distinct and of the same length. The order in which the rows appear in the table is immaterial and so is the order of the columns. Each column corresponds to a particular attribute characterizing the records, the column entries being the values of that attribute for the various records. A sample relation is shown in Table 9-2 representing a collection of students.

Each relation is characterized by its *relation scheme,* which is simply the ordered list of attribute names used to identify the records. The relation scheme for the sample relation of Table 9-2 is RELSCHEME (NAME, NUMBER, CLASS, CREDITS, ADDRESS, CITY). Some attributes or subsets of attributes can be used uniquely to identify the records included in a given relation. These are known as *candidate keys* or simply keys. Normally, one of the candidate keys, known as the *primary key,* is actually selected to represent the records. For the relation of Table 9-2 all the attributes except for CLASS are candidate keys. The student number (which certainly uniquely identifies a given student) could be used as the primary key.

A particular data base normally includes relations corresponding to a number of different relation schemes. The list of relation schemes characterizing a given data base is known as a *relational data base scheme*. A typical example is shown in Fig. 9-6 for a data base dealing with employees, managers, and projects. When a query is processed which requires information from more than one relation, it must be possible to relate the record information included

**Table 9-2 A Sample Relation**

| Name | Number | Class | Credits | Address | City |
|---|---|---|---|---|---|
| Brindle | 01764 | SOPH | 60 | 264 First St. | New York |
| Camino | 25611 | FRESH | 20 | 11 A St. | Kansas City |
| Daniel | 43799 | SENIOR | 100 | 1011 Main St. | Kingston |
| Katzer | 77084 | GRAD | 130 | 2146 Meadowbrook | Philadelphia |
| McGill | 37340 | SOPH | 45 | 1819 Edgemont | Washington |
| Noreault | 19450 | SOPH | 48 | 12 Ackerman | Syracuse |
| Salton | 30981 | GRAD | 110 | 2365 Meadowlark | Cambridge |
| Waldstein | 47592 | JUNIOR | 90 | 12 Woodsy Ave. | Ithaca |

1.

| EMPLOYEE NUMBER | EMPLOYEE NAME | SALARY | ADDRESS | DEPARTMENT NUMBER |
|---|---|---|---|---|

2.

| EMPLOYEE NUMBER | MANAGER |
|---|---|

3.

| MANAGER | PROJECT NUMBER | PROJECT NAME |
|---|---|---|

**Figure 9-6** Sample relational data base scheme.

in one relation with the information contained in some other relation. In the absence of pointers to relate corresponding records, the record relationships are obtainable indirectly if some of the attributes characterizing the records are included in more than one of the relation schemes that make up a given data base. In other words, the pointer structures used in the hierarchical and network data base models are replaced in the relational model by redundancy in the stored attributes.

For the sample relational data base scheme of Fig. 9-6 the EMPLOYEE NUMBER is included in both the first and the second relation schemes, and the attribute MANAGER is included in the second and third relation schemes. Thus by choosing entries in the first and second relations corresponding to a common employee number, one can identify the manager corresponding to a given employee. An additional search in the third relation for the manager record corresponding to a given manager included in the second relation determines the project to which a given employee is assigned.

In addition to interrogating a data base, typical relational database operations include the insertion of records into a relation, as well as the deletion and the modification of stored records. The required operations may be summarized by use of a *relational algebra* that manipulates relations [25]. The following operations are of main interest:

**1** Given two homogeneous relations in which the records are identified by the same number and the same types of attributes, the *union* of the two relations is a new relation containing all the distinct tuples contained in either of the original ones. Given relations R and S shown in Fig. 9-7a and b, respectively, the union R ∪ S is shown in Fig. 9-7c. The record (Adams, 25, New York) appears only once in the union, even though it is contained in both R and S, because duplicated tuples are not allowed in a relation.

**2** The *difference* between two homogeneous relations contains all tuples from the first relation that are not also contained in the second one. The difference R − S appears in Fig. 9-7d.

**3** A *selection* operation can be used to choose certain *rows* of a relation, depending on conditions imposed on the values of certain attributes. The general notation used to choose certain tuples from a relation R is $\text{SELECT}_F(R)$, where F is a formula expressing a condition. Figure 9-7e shows the relation $\text{SELECT}_{(\text{NAME} = \text{ADAMS})}(R)$ consisting of all records in R whose NAME attribute is "Adams."

| | | |
|---|---|---|
| Adams | 22 | New York |
| Adams | 25 | New York |
| Brown | 22 | New York |

(a)

| | | |
|---|---|---|
| Adams | 25 | New York |
| Smith | 25 | Chicago |

(b)

| | | |
|---|---|---|
| Adams | 22 | New York |
| Adams | 25 | New York |
| Brown | 22 | New York |
| Smith | 25 | Chicago |

(c)

| | | |
|---|---|---|
| Adams | 22 | New York |
| Brown | 22 | New York |

(d)

| | | |
|---|---|---|
| Adams | 22 | New York |
| Adams | 25 | New York |

(e)

| | |
|---|---|
| New York | Adams |
| New York | Brown |

(f)

| | | | | | |
|---|---|---|---|---|---|
| Adams | 22 | New York | Adams | 25 | New York |
| Adams | 22 | New York | Smith | 25 | Chicago |
| Adams | 25 | New York | Adams | 25 | New York |
| Adams | 25 | New York | Smith | 25 | Chicago |
| Brown | 22 | New York | Adams | 25 | New York |
| Brown | 22 | New York | Smith | 25 | Chicago |

(g)

**Figure 9-7** Relational algebra. (a) Relation R. (b) Relation S. (c) Relation R ∪ S. (d) Relation R − S. (e) Relation $SEL_{(NAME = ADAMS)}(R)$. (f) Relation $PROJ_{3,1}(R)$. (g) Relation R × S.

**4** The converse of a select operation is known as a *projection*. In a projection, certain columns are chosen from an original relation to form a new relation, care being taken as usual to eliminate any row duplications that may result. The order in which the columns appear in the projected relation may be altered as part of the project operation. Figure 9-7f contains a relation $PROJECT_{3,1}(R)$ obtained by taking column 3 followed by column 1 from the original relation R. The deletion of the second attribute eliminates the distinction between the first two tuples of R; a single tuple (New York, Adams) thus replaces the two original tuples in the projected relation. Project operations are useful to break down large complex relations into several smaller and simpler ones.

**5** The *Cartesian product* R × S of two relations R and S is a new relation consisting of all possible unique tuples obtained by taking a tuple from R followed by a tuple from S. The length of the rows in R × S is equal to the sum of the length of tuples in R plus the length of tuples in S. Relation R × S is shown in Fig. 9-7g. Unless duplicate tuples are generated, the number of rows in

$R \times S$ is equal to the product of the number of tuples in R and S. The Cartesian product operation can be used to construct a single large relation from two or more smaller ones.

The five basic operations of the relational algebra can be used to generate additional operations that may be useful. Thus the intersection of two relations $R \cap S$ is defined as $R - (R - S)$; that is, it consists of all tuples in R that are not only in R. Of particular interest is the *join* operation in which a single relation is formed from two initially given relations based on certain conditions of equality or inequality among the attribute values included in the relations. Two types of join operators may be distinguished known as the *natural* (or unrestricted) join and the *restricted* (conditional) join. Both of these operations are effectively special cases of the Cartesian product operation, and they are used to construct larger relations from smaller ones.

The restricted join is a product operation where conditions are imposed on the values of attributes from the original relations. The notation T JOIN$_{(B<D)}$ V used in Fig. 9-8f implies that the joined relation consists of the tuples which result by taking a tuple from relation T followed by a tuple from relation V with the added condition that the value of attribute B in relation T be smaller than the value of attribute D in relation V. A restricted join is then equivalent to a Cartesian product followed by a select operation. The restricted join is illustrated in Fig. 9-8f for the relations T and V shown in Figs. 9-8d and e. The first tuple in the joined relation (1,2,3,3,1) is obtained by juxtaposing the first tuple

| A | B | C |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | 9 |
| 1 | 2 | 6 |

(a)

| C | D |
|---|---|
| 3 | 1 |
| 6 | 2 |
| 3 | 3 |

(b)

| A | B | C | D |
|---|---|---|---|
| 1 | 2 | 3 | 1 |
| 1 | 2 | 3 | 3 |
| 4 | 5 | 6 | 2 |
| 1 | 2 | 6 | 2 |

(c)

| A | B | C |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | 9 |
| 1 | 2 | 6 |

(d)

| D | E |
|---|---|
| 3 | 1 |
| 6 | 2 |
| 3 | 3 |

(e)

| A | B | C | D | E |
|---|---|---|---|---|
| 1 | 2 | 3 | 3 | 1 |
| 1 | 2 | 3 | 6 | 2 |
| 1 | 2 | 3 | 3 | 3 |
| 4 | 5 | 6 | 6 | 2 |
| 1 | 2 | 6 | 3 | 1 |
| 1 | 2 | 6 | 6 | 2 |
| 1 | 2 | 6 | 3 | 3 |

(f)

**Figure 9-8** Join operations. (a) Relation T. (b) Relation U. (c) Natural join (unrestricted) T JOIN U. (d) Relation T. (e) Relation V. (f) Relation join T JOIN$_{(B<D)}$ V.

from T and the first tuple from V [(1,2,3) and (3,1), respectively]. In that case, the value of attribute B in relation T (2) is clearly smaller than the value of attribute D in V (3).

For the *unrestricted or natural join,* the assumption is that certain attributes in the relations to be joined cover the same domain of values, that is, certain columns in the two relations represent values for the same attribute. Such a situation exists for relations T and U of Fig. 9-8a and b where attribute C is repeated in both relations. The unrestricted join is then similar to the restricted join except that the join condition is assumed to be the equality of the components for the columns representing identical attributes. For the example of Fig. 9-8 the operation T JOIN U is then equivalent to the restricted join T $\underset{(C = C)}{\text{JOIN}}$ U . In addition one copy of each duplicated column resulting from the join operation is eliminated in the natural join. In the example of Fig. 9-8c only one copy of column C is maintained instead of the two produced by the normal join operation.

The natural join is used when items are wanted for which the components are initially stored in different relations. Thus given the relational data base scheme of Fig. 9-6, a natural join of relations 1 and 2 produces a list of employees together with the corresponding managers. Similarly a natural join of relations 2 and 3 produces a list of managers followed by the corresponding employee numbers, project numbers and project names.

Sample retrieval operations using the relational model are examined later in this chapter.

### *D The Hierarchical Data Base Model

The relational data base model is conceptually very simple, because each file is represented by a table with homogeneous rows consisting of a specified number of attribute values. To obtain answers to information requests, a great deal of work may, however, become necessary to manipulate the relations, including, for example, the previously mentioned join and Cartesian product operations. In addition, large relations may have to be maintained in which certain attribute values are repeated many times. This last solution reduces the number of needed join operations but entails substantially increased storage costs. Thus by maintaining a single relation consisting of the list of managers together with the corresponding employee names one avoids the join between relations 1 and 2 of Fig. 9-6. In the large, single relation, the name of the manager is repeated once for each corresponding employee.

The choice between operational complexity and increased storage cost inherent in the relational model may be avoided by complicating the storage structures through the introduction of pointers connecting different types of records. In the hierarchical model the assumption is that a natural hierarchy is definable where a large number of narrow entities can conveniently be related to a small number of broader entities, that are in turn relatable to still broader entities, and so on, until a single global entity remains at the top of the hierarchy. For example, individual citizens live in cities that are grouped into coun-

ties which in turn are located in states, and so on. A typical hierarchical arrangement of *record types* is shown in Fig. 9-9a. When a many-to-one relationship exists between a given type of record (record type) A and a second record type B, the records of type A are considered hierarchically inferior to those of type B. Hence in the hierarchical arrangement type A would be listed below type B.

In the example of Fig. 9-9, the assumption is made that each postal region contains many post offices each of which in turn serves as a center for many postmen; furthermore each postman services many postal patrons. In the actual file arrangement a hierarchically superior entity can be stored together with all the inferior entities to which it is related. This is suggested in the illustration of Fig. 9-9b. One possible physical implementation for such a system is the pointer chain of Fig. 9-9c, where the record for postman A *points to* (gives the address of) the record of patron a, which in turn points to the next patron ser-

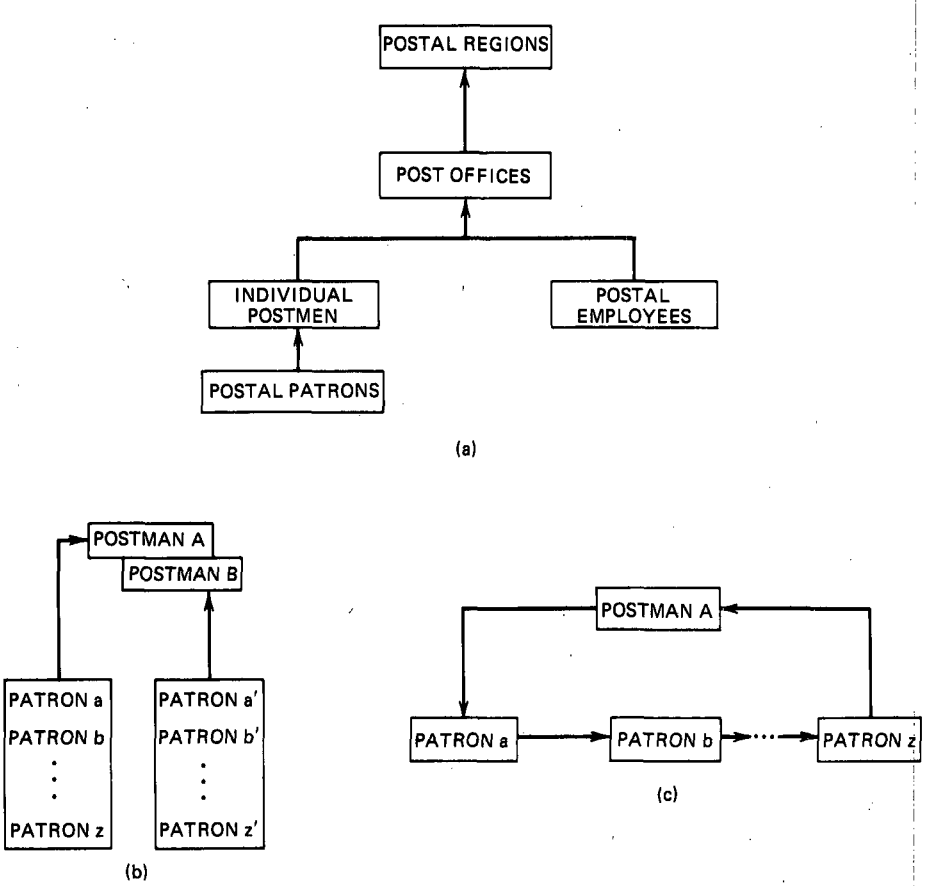



**Figure 9-9** *Hierarchical data base system. (a) Hierarchical arrangement of record types. (b) Many-to-one relationship between record types. (c) Pointer chain arrangement.*

viced by the same postman (patron b), and so on, down to the last patron, whose record then points back to postman A to complete the chain.

Obviously such an arrangement makes it easy to retrieve the set of postal patrons corresponding to a given postman. More generally, in the hierarchical data base system the file structure is accessed in a top-down manner by starting at the top of the hierarchy and proceeding downward in the tree structure until all desired items are found.

When a tree structure is usable to represent the relationships between record types in a natural way, a hierarchical data base system affords an efficient implementation of the search and retrieval problem. Unfortunately, hierarchies appear restrictive in many situations. Consider, for example, the data base of Fig. 9-6 used earlier as an example. If one assumes that each manager handles many projects and many employees, one obtains the structure shown in Fig. 9-10a. The structure of Fig. 9-10a makes it difficult to relate employees and projects. If one were interested in the set of employees assigned to the various projects, one could create a second set of employee records placed below the project information, as shown in Fig. 9-10b. In that case, some of the employee information would be duplicated in the file system.

Suppose now that some employees are assigned to many projects. In that case, a many-to-one relation exists between projects and managers as well as between projects and employees, corresponding to the structure of Fig. 9-10c. Such a structure is not allowed in a hierarchy because the PROJECT records now have two hierarchically superior record types. This can be avoided by creating a second tree to be added to the tree of Fig. 9-10b. Such a tree is shown in Fig. 9-10d. The combination of the two trees of Fig. 9-10b and d is now usable to represent all the necessary relationships for the original data base of Fig. 9-6. For obvious reasons a solution which consists in the use of several partly overlapping hierarchies is not ideal, because a great deal of the stored information must be duplicated and jumps are necessary from one tree to another to gain access to appropriate portions of the data base.

Much duplication in the stored data can be avoided by introducing additional pointer systems to be used together with the normal physical pointers. Consider, for example, the tree structure of Fig. 9-10a, and assume that it becomes necessary to add data indicating that some employees are assigned to many projects. In other words, the projects records are considered to be hierarchically inferior not only to the manager records but also to the employee records. This could be done by adding as a new tree the structure of Fig. 9-10d necessitating the duplication of certain project as well as certain employee information. The duplication can, however, be avoided by using a new set of so-called logical pointers connecting employees to the projects to which they are assigned. The PROJECT file then becomes a logical child (logically inferior) to the logically superior employee file. This is shown in Fig. 9-11a, where the logical pointers are symbolized by dashed connections.

The logical pointer system effectively introduces nonhierarchical relationships into the data base system without abandoning the basic hierarchical struc-

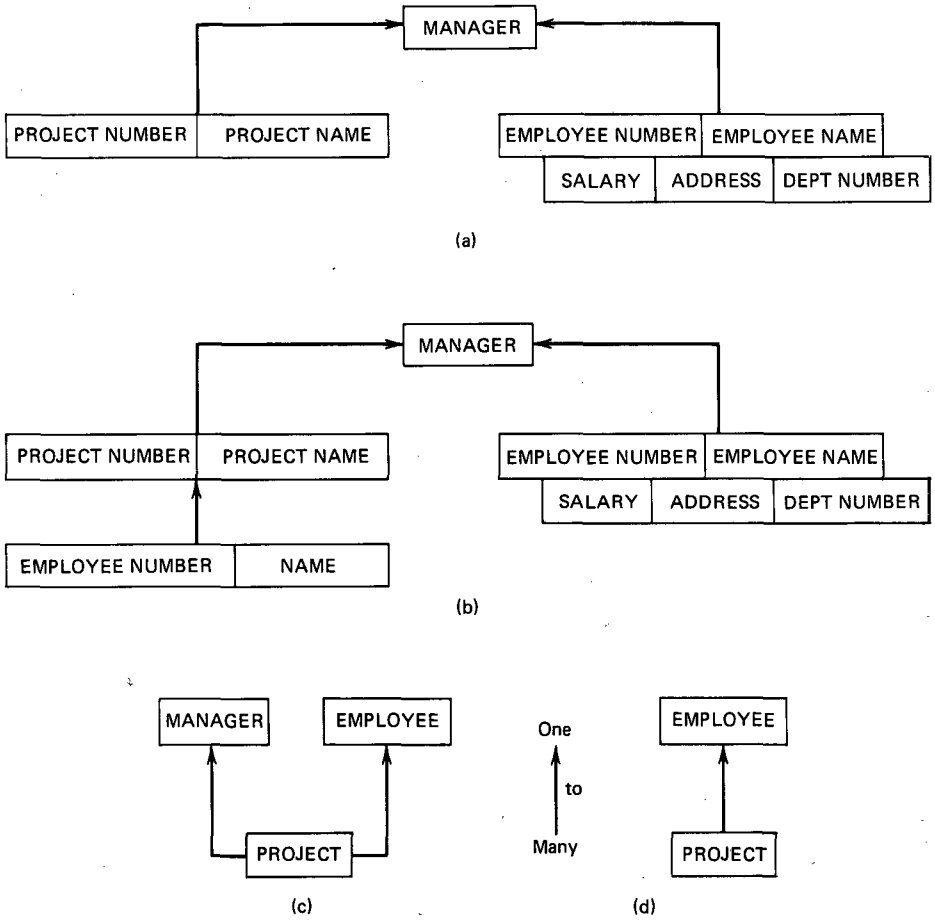



**Figure 9-10** Relationships between record types. (a) Sample hierarchical structure for data base of Fig. 9-6. (b) Altered structure accounting for many-one relation between employees and projects. (c) Hypothetical relationship between record types. (d) Structure reflecting assignment of employees to many projects.

ture. In the example of Fig. 9-11b the dual pointer system provides two kinds of siblings for the PROJECT file: the physical siblings consisting of sets of projects directed by the same manager, and the logical siblings consisting of projects to which a given employee is assigned in common. An example of this situation is shown in Fig. 9-11b, where projects a, b, and c are physical siblings since they are all managed by the same manager, whereas projects a and d are logical siblings because employee 1 is assigned to both projects.

One more problem must be mentioned concerning the representation of many-to-many relationships between types of records. It was seen earlier that the normal hierarchical arrangement accommodates only many-to-one and one-to-one relationships. Many-to-many relationships do occur in many situations, and provisions must be made for them in an effective data base system. Con-

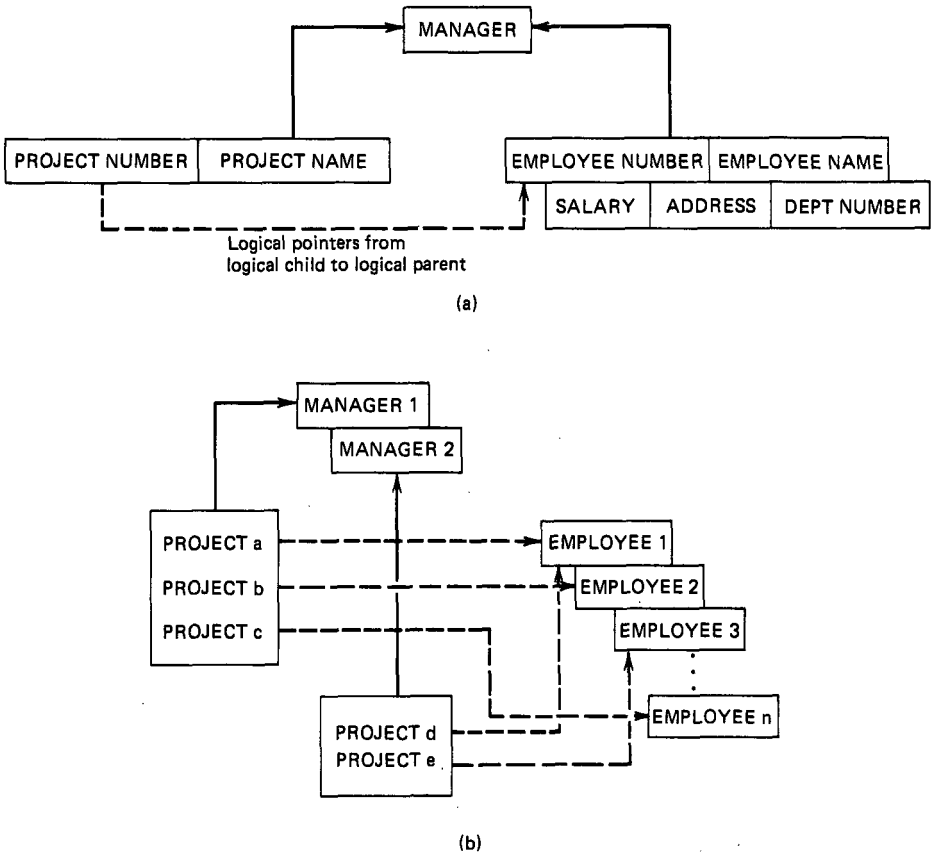



**Figure 9-11** Logical pointer structure. (a) Hierarchical data base with logical pointer arrangement. (b) Pointer arrangement illustrating physical and logical connections.

sider again the sample data base of Fig. 9-10b, where the many-to-one relationship between employees and projects is represented by physical pointers. If the logical pointers already used in Fig. 9-11a to relate the PROJECT and EMPLOYEE records were added to the structure of Fig. 9-10b, a method would be provided to represent the many-to-many relationship between projects and employees. In that case, the many-to-many relationship is replaced by two many-to-one relationships, one of them being implemented by logical pointers as shown in Fig. 9-12a.

An alternative, more direct way for implementing many-to-many relationships between two record types consists in creating a third record type representing the intersection between the two original types of records. The many-to-many relationship can then be represented by conventional many-to-one pointers between each of the original record types and the newly defined intersection records. For the example under discussion, the intersection records could specify the percentage of time each employee spends on a particular project. The logical structure of the resulting data base is shown in Fig. 9-12b

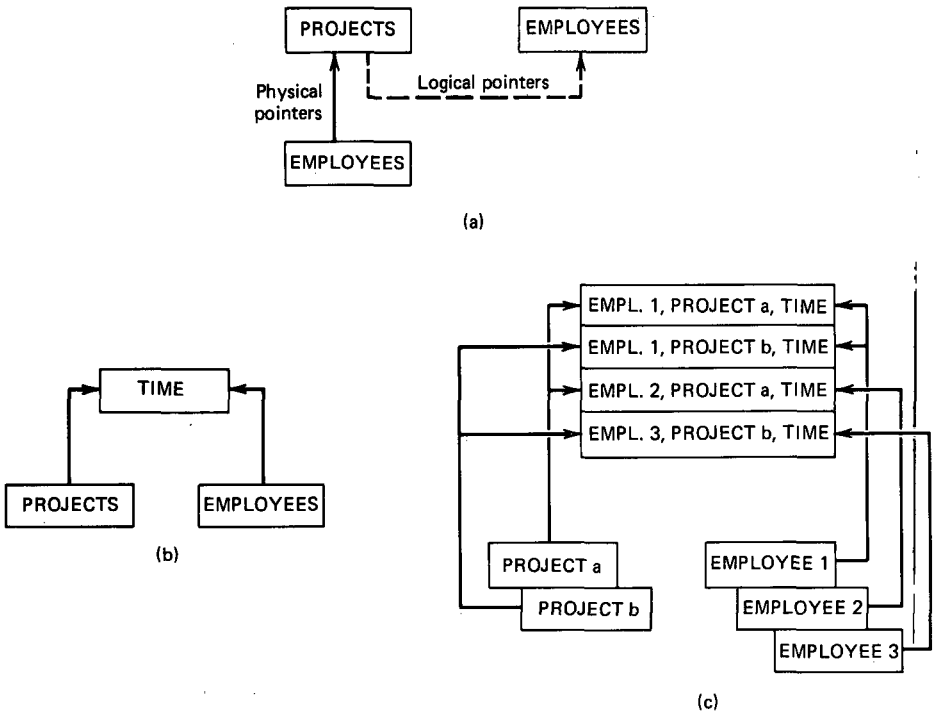



**Figure 9-12** Implementation of many-to-many relationships in hierarchical systems. (a) Implementation of many-to-many relationship using logical pointers. (b) Use of intersection records. (c) Sample data base for structure of part b.

where TIME records are used to represent the intersection. A short example, showing the pointer structure used for two projects and three employees, is included in Fig. 9-12c.

The use of logical in addition to physical (hierarchical) pointers and of intersection record types adapts the hierarchical data base model to most processing requirements. The usefulness of the hierarchical structure is not diminished by the alterations because the hierarchy always controls the record access. Specifically, each record is stored together with its descendants (the other records to which it is related by a many-to-one relationship), and each descendant is stored in turn with its own descendants. Access to the tree structure is obtainable by using the records located at the *root* of the tree, that is, at the top of the tree in the illustrations. For the hierarchy shown in Fig. 9-9a, access must be obtained by first looking at the file of postal regions, which in turn gives access to certain post office records from where one can get to individual postmen or postal employee records. An example of an access implementation is shown in Fig. 9-13.

In the illustration of Fig. 9-13 an index is provided to access the records constituting the root, that is, the records of type A. The descendants of each particular record of type A, say $a_i$, are then stored sequentially after $a_i$ in tree accessing order. The accessing order used in the example of Fig. 9-13 is known

as "preorder," where the root is listed first for each subtree, followed by the leftmost child which is in turn followed by its leftmost child if there is one. When no further children are available, the next sibling to the right of the most recent child is taken up, and so on until the tree is exhausted. For the subtree starting at the root record $a_{10}$, the preorder list contains in order $a_{10}$-$b_1$-$b_2$-$d_1$-$d_2$-$c_1$-$c_2$.

To find a particular record in the tree the index is first consulted to find a particular record among the root records. From there the pointers are followed to the corresponding list of child records. This list is then scanned until the wanted record is found. Since the number of child records may be variable, and the initial ordered arrangement may change as records are added and deleted, it is prudent to provide overflow storage in addition to the primary storage area for the lists of records. Records that do not fit into the primary area are then pushed into the overflow area, which is connected to the primary storage by pointers as shown in Fig. 9-13c.

The access arrangement illustrated in Fig. 9-13 is known as the hierarchical indexed sequential access method (HISAM). It is used in the well-known Information Management System (IMS) implemented by IBM [16,26,27]. In the HISAM access method, immediate access is obtained only to certain records of the root of the tree; the remaining records are then located by a sequential scanning process. A faster (but more expensive) accessing method consists in providing pointers to access the individual descendants of each root record. This is done in the hierarchical direct access method (HIDAM). In the HIDAM organization, individual pointers would thus connect each pair of adjacent records in

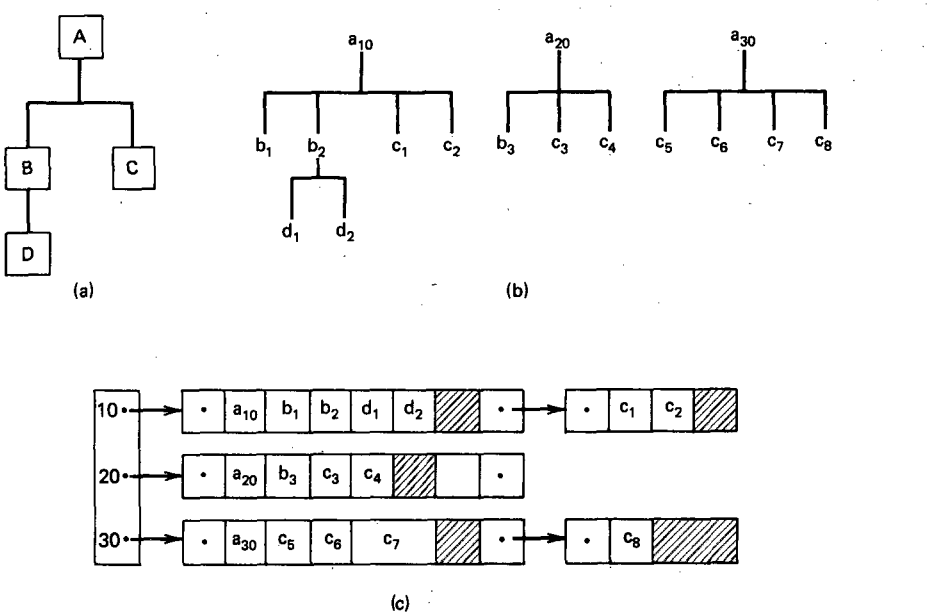



**Figure 9-13** Typical hierarchical indexed sequential access method (HISAM). (a) Basic structure. (b) Sample data base. (c) Indexed sequential storage arrangement.

the access path (for example, in the data base of Fig. 9-13 from $a_{10}$ to $b_1$, from $b_1$ to $b_2$, from $b_2$ to $d_1$, and so on). A hashing method could also be used to access the records located at the root of the tree instead of the special index shown in Fig. 9-13. In that case, one obtains a direct hierarchical access system (HDAM). Because the popular IMS data management system is based on hierarchical data structures, the hierarchical model has been widely accepted as a basis for data base operations.

### *E The Network Data Base Model

The network data base model has been intensively studied over many years and has been sanctioned by official groups charged with the standardization of automated data processing activities [28–30]. More specifically, a special group of experts known as the Data Base Task Group (DBTG) was set up some years ago by the Conference on Data Systems Languages (CODASYL) in order to study and propose standards for data base processing. Since the parent CODASYL committee had earlier been responsible for the development of the well-known Common Business Oriented Language (COBOL), the DBTG eventually proposed a model for data base management, known as the DBTG proposal, which is designed to operate with COBOL. The DBTG proposal uses a network data structure and has served as the basis for the design of several network data base systems; the IDS/II system designed by Honeywell Information Systems is a typical example [31].

The underlying idea behind both the hierarchical and the network data base models is the need for easy manipulation of different types of records. The pointer systems are designed to make it easy to "travel" from a particular record occurrence to the set of related records. It was seen earlier that the principal relationship between different record types used in the hierarchical model was a many-to-one correspondence between a set of records of some particular type and a single record of some other type. The simple many-to-one relationship also forms the basis for the network model, but in that model the allowable patterns of many-to-one relations are more flexible than in the hierarchical case. For example, in a network model, several different many-to-one relationships can be defined from a single record type, say, A, to various other record types B, C, D, etc. The structure of Fig. 9-10c that is disallowed in the hierarchical model is specifically permitted in a network implementation. On the other hand, the direct use of many-to-many relationships is still prohibited because of the resulting complications in the pointer structures. Intersection record types may again be used for the substitution of many-to-one relationships for the original many-to-many relationship, as explained earlier.

The greater complexity of the allowable relationships in the network model simplifies the navigation in the data base, that is, the movement from one record type to another as required to generate answers to certain queries. On the other hand, the file access becomes more complicated. Consider, as an example, a set of suppliers of certain goods, and a set of clients that order these goods. A many-to-many relationship is definable between the supplier records

and the goods records because one supplier may furnish many different items, while a single item may be obtained from several suppliers. Analogously, a many-to-many relationship also exists between the client records and the goods records because once again a client buys many items while one item may be bought by many clients. This situation is represented schematically in Fig. 9-14a.

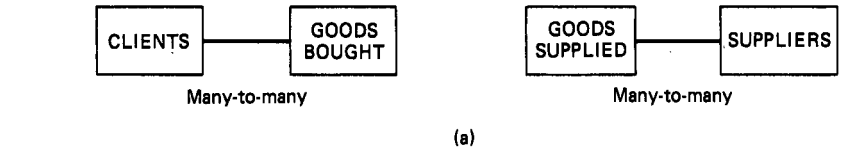
Since the many-to-many relationships are not directly representable, two new record types can be introduced: the first one, labeled "quantity," provides the quantity of a certain item ordered by a particular client; and the second, known as "price," can state the price for an item obtained from a particular supplier. Using the three basic record types and the two intersection record types, the two many-to-many relationships of Fig. 9-14a are now replaceable by four many-to-one relationships as shown in Fig. 9-14b.

The structure of Fig. 9-14b is obviously not a hierarchy and could not therefore be directly used in a hierarchical system. In a hierarchical system it would be necessary to break up the structure into several trees following duplication of the intersection records. In a network system, a direct implementation is possible by using, for example, pointer chains of the kind shown in Fig. 9-9c to represent the many-to-one relationships. In the terminology used by the Data Base Task Group, the set of records included in a particular pointer chain, consisting of a particular single record of type A and of the number of related records of type B, is known as a "data base set." (The term "set" as used here is not to be confused with the conventional mathematical meaning of that term.) The single record of type A to which the other records in a data base set are related is a member of the *owner* type of records, the other records of type B are included in the *member* record type. In Fig. 9-14b, the intersection record types (that is, quantity and price) are "member" types, whereas clients, goods, and suppliers are "owner" types.

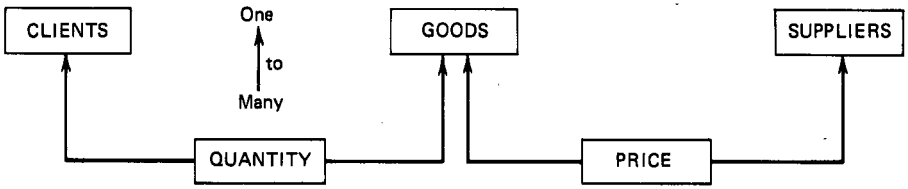
A typical implementation using pointer chains to represent the data base sets is shown in Fig. 9-14c for the earlier example of Fig. 9-14b. It is easy to see that the pointer chains can be followed from any particular record to any other related record. However, the structure is substantially more complicated than a comparable hierarchical implementation. To avoid errors in a search, it is necessary to control the traversal process when using the overlapping pointer structures. In the DBTG proposal this is done by means of so-called currency pointers which identify the last record of each record type that was accessed at each particular point, as well as the last accessed record in each data base set.

A detailed evaluation of the effectiveness and efficiency of the various data base models is not possible at present. Some general remarks may suffice. The relational system appears superior from the viewpoint of the casual user. Only a single construct must be understood, and that construct consists of simple tables. Furthermore, high-level languages are available to manipulate the tables. For small data base problems, the relational model appears obviously preferable.

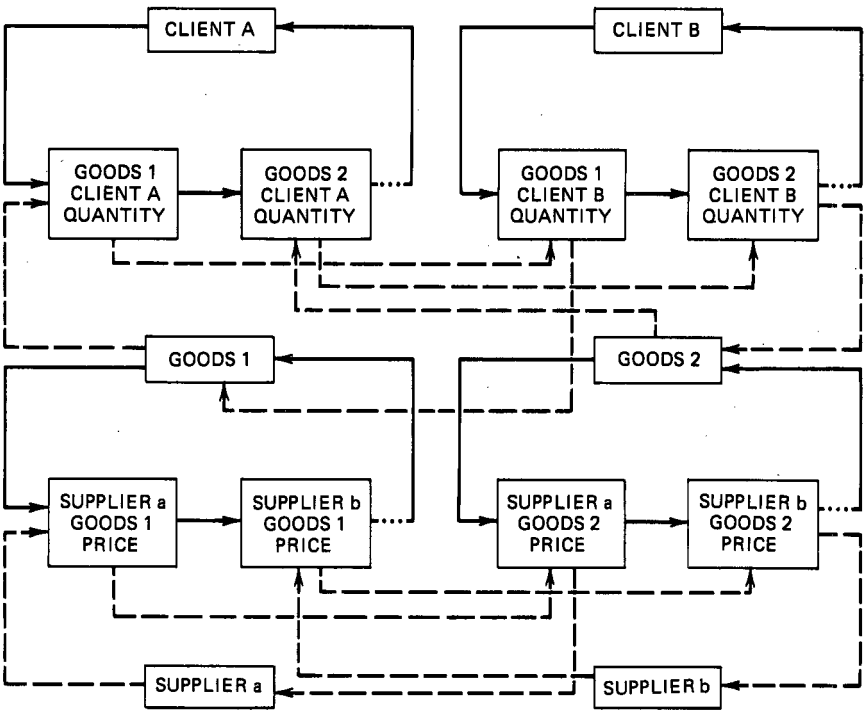
On the other hand, as the data base grows in size, the efficiency of the pro-




(a)




(b)




(c)

**Figure 9-14** Network data base implementation. (a) Two many-to-many relationships. (b) Four many-to-one relationships for data base of part a. (c) Pointer chain implementation for relationships of part b.

cessing operations becomes important. In that case, the explicit relationship indicators (pointers) or the use of separate indexes affording access to various groups of records may become essential. In particular, in the hierarchical and network systems, the pointers allow direct access to sets of related records, whereas in the relational system these relationships and the corresponding access paths may first have to be established before they can be used. Unfortunately, the pointer structures and the implementation of complex many-to-many relationships between record types are more difficult to deal with than a simple table; as a result a substantial burden is placed on the programming staff charged with setting up the data base system.

At the present time, most commercial applications are based on the hierarchical or network data base models. The conceptually simpler relational model has, however, been studied intensively in recent years, and various theoretical advantages have been claimed for that model [32]. Furthermore the slow sequential scan of all the records included in a given relation may be avoided by providing auxiliary indexes that offer access to groups of related records. Because of its greater conceptual simplicity and ease of use, one may expect that the relational model will become solidly established as a tool for data base manipulations in the visible future.

## 3 QUERY PROCESSING

### *A Query Language Types

A great deal of work has been devoted to the construction of data base query languages. Since data base systems are designed for use by business-oriented persons rather than by computer experts, a main consideration is user friendliness and transparency of the operations. In particular, great importance is attached to the use of high-level query languages which reflect the overall user's intent rather than the computer operations that may be required to obtain any particular result. Unhappily, the use of high-level query languages entails a good deal of extra work in transforming the user statements into programs acceptable by the computer. Furthermore, the efficiency of the programs resulting from such a transformation may leave something to be desired. A tradeoff thus exists between the use of low-level programming-type query statements that may be hard to formulate but lead to efficient search and retrieval specifications, and the use of high-level user-friendly query formulations that may lead to slower, more expensive retrieval processes.

The query languages used in data base environments must provide for more processing alternatives that those commonly used in bibliographic retrieval. Whereas the latter need only specify how to retrieve documents, a DBMS query language also permits the user to ask for calculations on the stored numerical data and for the retrieval of computed information derived from the originally stored data. In principle, it might be useful to provide the full scope of the unrestricted natural language for the formulation of data base queries. For reasons outlined earlier, a natural language analysis system is still

too complex and expensive for use in most commercial applications. However, restricted natural language features are being included in some data base query languages, as will be seen [33–39].

In some situations, the problem of whether a high-level (user-friendly) or a low-level (programming-type) query language is to be used never arises. Only a very high-level querying capability can, for example, serve in a department store to provide information to masses of untrained customers about the location of various kinds of merchandise. In principle, a game of "twenty questions" might be played by confronting the customers with question sequences in the hope of eventually leading them to the desired information. Specifically, as each question is proposed, the user provides a yes/no answer which then triggers the next query. A typical query sequence of this kind is shown in Table 9-3.

Since it is difficult to design efficient sequences of simple yes/no queries that will lead to correct final responses in a small number of steps, a faster process is the so-called menu approach, where the user is confronted with an enumeration of different available alternatives. The choice of a particular alternative will then lead to a further display containing a more refined list of possibilities. Eventually, the user obtains the desired information without further question. The menu approach serves as a basis for the operation of certain electronic news services where news items are displayed at individual user stations to specific customers upon demand [40].

A typical menu display suitable for a department store information system is illustrated in Fig. 9-15. In that case, the user first chooses item 3 (women's wear) from the number of alternatives offered, followed in the next display frame by item 4 (shoes), eventually leading to a response locating the shoe department on floor 2 section C. It should be pointed out that while the menu approach may rapidly lead to a final answer, the system is not foolproof. In fact, users may find it difficult to make a proper choice among the offered alternatives; and once the wrong path is followed, all subsequent displays in the current sequence will be unhelpful. Consider, for example, a user interested in fluids used to wash automobiles. This user might decide to follow the sequence

**Table 9-3 Sample Query Sequence**

| Query | User response |
|---|---|
| Purchase item? | Yes/no ✓ |
| Clothing? | Yes/no ✓ |
| Appliances? | Yes/no ✓ |
| Kitchen appliances? | Yes/no ✓ |
| Major kitchen appliances such as stoves, dishwashers? | Yes/no ✓ |
| Answer: Floor 5 Section B | |

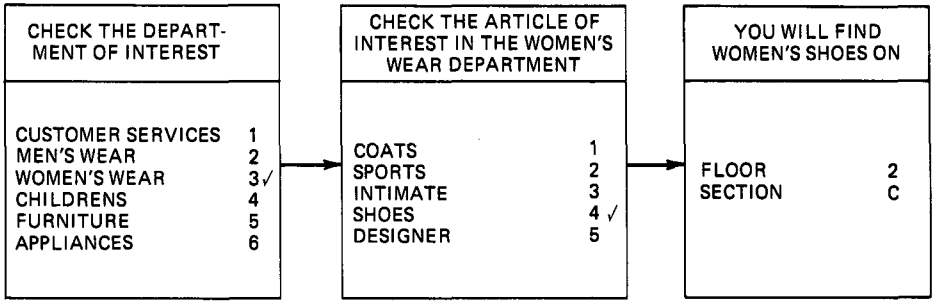



**Figure 9-15** Menu display suitable for department store inquiry system.

starting with automobile supplies only to find later that cleaning agents were actually classified under household rather than automobile supplies. The design of transparent classification systems leading to unambiguous menu sequences is not a task that is currently well understood.

The menu approach is most successful when it is used by people who may be expert in a particular subject area without, however, being knowledgeable in computer processing. In that case the kind of erroneous choice previously illustrated can be avoided and proper responses may be obtained without training the users in the formulation of queries or the search methodologies. Typical examples that come to mind are populations of medical doctors where the retrieval system might furnish diagnostic aids based on menu displays of symptom lists. Alternatively, the menu system could be used by lawyers for statute or case law searching, by chemists dealing with the properties of chemical compounds, and by other professional groups operating in restricted topic areas with a well-defined terminology.

The use of *tabular querying* methods is related to menu querying but provides somewhat greater flexibility in the permissible query formulations and in the specification of processing sequences. In that case an electronic questionnaire properly filled in and submitted through the automatic display equipment replaces the normal query formulation process. A sample tabular query is contained in Table 9-4 for a data base dealing with hazardous chemicals. The user in this case is interested in the chemical names and the manufacturers of toxic

**Table 9-4 Sample Tabular Query**

| Property or specification | Relation | Value | Output | Operation number |
|---|---|---|---|---|
| CHEMICAL FORM | = | GAS | | 1 |
| LETHAL DOSE FOR 50% | ≥ | 10 PARTS/MILLION | | 2 |
| PRODUCTION | ≥ | 1000 KG/YEAR | | 3 |
| 1, 2 | AND | | | 4 |
| 4, 3 | AND | | | 5 |
| CHEMICAL NAME | | | PRINT | |
| COMPANY NAME | | | PRINT | |
| COMPANY ADDRESS | | | PRINT | |

gases for which the lethal dose to 50 percent of the exposed population is at least 10 parts per million and the production is at least 1,000 kilograms per year. In the formulation of Table 9-4 the three basic requirements (that is, the chemical form, the lethal dose, and the production amount) are effectively "anded" together to produce the final query.

The well-known Query-by-Example (QBE) system is a commercial tabular querying system specially adapted to the relational data base model [41]. In QBE, a relation (file) to be operated upon during the search process must first be displayed in raw form on the user console by drawing a table showing the relation name at the head of column 1 and the attribute names identifying that relation as heads of subsequent columns. A typical frame for the display of a given relation is shown in Table 9-5. In a real case, the first row of the table would be automatically filled in with the correct relation and attribute names.

To operate on a displayed relation, the user fills in the various columns of the table with appropriate instructions and values of attributes. In QBE, two kinds of values are distinguished: those taken literally, where the value shown is directly used as an operand, and those used as samples to designate unknown values of corresponding attributes. In the examples that follow, sample values are underlined, whereas actual values are written without underline.

To retrieve the name and employee numbers for all records contained in the EMPLOYEES file, one first requests a display of the EMPLOYEES relation and then fills in the attribute columns corresponding to employee name and number with the command PRINT (P.) followed by a dummy value. The result is shown in Table 9-6a. When the query formulation involves more than one relation, the frames for all required relations must be displayed, and the relationships between the values of corresponding attributes must be appropriately specified. Consider as an example, the query formulation shown in Table 9-6b. This display involves the EMPLOYEES and the MANAGER relations. The first part of Table 9-6b is identical to Table 9-6a. The second part qualifies the names of the employees actually wanted to those whose manager is "Smith."

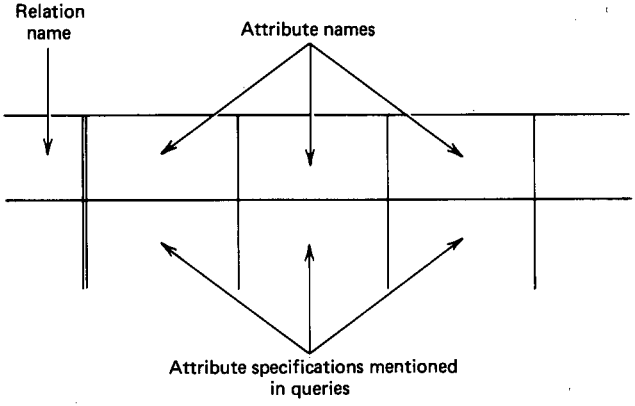



**Table 9-5** Display of relation framework in the query-by-example system.

| EMPLOYEES | EMP. NUMBER | EMP. NAME | SALARY | ADDRESS | DEPT. NUMBER |
|---|---|---|---|---|---|
| | P. 123 | P. BROOKS | | | |

(a)

| EMPLOYEES | EMP. NUMBER | EMP. NAME | SALARY | ADDRESS | DEPT. NUMBER |
|---|---|---|---|---|---|
| | P. 123 | P. BROOKS | | | |

| MANAGER | EMP. NUMBER | MANAGER NAME | |
|---|---|---|---|
| | 123 | SMITH | |

(b)

| EMPLOYEES | EMP. NUMBER | EMP. NAME | SALARY | ADDRESS | DEPT. NUMBER |
|---|---|---|---|---|---|
| | | SMITH | X | | |
| | P. 123 | P. BROOKS | > X | | |

(c)

**Table 9-6** Query-by-example query formulations. (a) Print the names and numbers of all employees. (b) Print the names and numbers of all employees whose manager is Smith. (c) Print the names and numbers of employees whose salary exceeds Smith's.

Note that the name "Smith" is to be taken literally which accounts for the lack of underlining in the table. In Table 9-6b, the *same* dummy employee number (namely, 123) is used in the EMPLOYEES and the MANAGER relations to indicate that the employee names to be retrieved are precisely those managed by "Smith." Relationships between attribute values other than equality can be specified by using appropriate quantifiers as shown in the example of Table 9-6c, where the request covers the names and numbers of all employees whose salary is at least as large as some unknown value x, x being specified as Smith's salary. Even though the QBE system can be used to formulate queries of substantial complexity, the basic features can be mastered rapidly by untrained users.

In addition to the table languages, several higher-level query languages have been proposed in which English keywords are used to specify the role of files and attributes. The SEQUEL language is a typical example which is also designed to operate in a relational framework [42,43]. The basic construct in SEQUEL is the SELECT-FROM-WHERE specification, where the SELECT

part specifies the attributes actually wanted, the FROM portion identifies the relation in which these attributes occur, and the WHERE clause introduces an expression characterizing the attribute values previously named. In many cases, a SEQUEL formulation can be immediately understood without further clarification or instruction. Thus the formulation

```
SELECT  EMP. NAME, EMP. NUMBER
FROM    EMPLOYEES
WHERE   SALARY ≥ 10,000
AND     DEPT = 123
```

requests names and numbers of employees included in the EMPLOYEE relation such that the value of the SALARY attribute equals at least 10,000 and the DEPARTMENT is specified as department 123.

Attributes from several relations can be used by nesting the SELECT-FROM-WHERE clauses. In particular, the expression following the WHERE statement can itself introduce a new relation. Thus, the names of employees whose manager is SMITH may be requested in the following way:

```
SELECT  EMP. NAME, EMP. NUMBER
FROM    EMPLOYEES
WHERE   EMP. NUMBER IS IN
        SELECT EMP. NUMBER
        FROM   MANAGER
        WHERE  MANAGER NAME = 'SMITH'
```

The keywords "IS IN" introduce a second relation, which restricts the values of the employee numbers to only those managed by Smith.

A language such as SEQUEL uses linguistic constructs for query specification. The English words are, however, used in a specialized sense with unambiguous syntactic and semantic properties. Higher-level query languages also exist in which the English-like specifications are replaced by mathematical formulas. The so-called relational calculus is a case in point, where the wanted records are identified by a given variable name, and a mathematical expression involving the variable is then used to characterize the records actually wanted [44]. Thus, if the variable t is used to identify the records under consideration, and R and S are names of relations, the specification $\{t \mid R(t) \vee S(t)\}$ requests the retrieval of all records that are contained either in relation R or in relation S.

In addition to the higher-level nonprocedural languages, it is of course also possible to use lower-level programming-type languages where the actual procedures needed to carry out the retrieval operations are directly specified. The *relational algebra* previously introduced in Fig. 9-7 is an example where join, project, and select operations correspond directly to particular relational operations. For example, given two relations containing attributes A and B and attributes C and D, respectively, the following expression in the relational alge-

bra is used to obtain the values of attribute A for all the records for which the value of attribute B in the first relation equals the value of attribute C in the second, and the value of attribute D is "Smith."

$$\text{PROJECT}_{A}(\text{SELECT}_{B=C \text{ AND } D='SMITH'}(AB \times CD)) \quad (1)$$

Assuming that the two relations are the previously used EMPLOYEES and MANAGER relations, and the four attributes A, B, C, D correspond, respectively, to employee names, employee numbers, employee numbers, and manager names, the query of expression (1) once again requests the names of all employees whose manager is "Smith." The difference between the procedural formulation of expression (1) and the previous formulations is the direct operational specification of expression (1), consisting in this case of a Cartesian product joining the two relations, followed by a selection, which is followed in turn by a projection.

In some operational environments, several different query facilities are provided to serve different user classes. This is the case, for example, in the Laboratory Animal Data Bank available through the National Library of Medicine which is searchable by experimental scientists with a menu approach or by information professionals with programming knowledge using a programming-type query language [45,46]. The idea is that the information professionals charged with operating the system will find it worthwhile to study the file structure and the relationships between attributes in order to be able to propose efficient query formulations. On the other hand, the file search itself constitutes only an incidental part of the activities for the experimental scientist. In that case there is no immediate need to learn the command structure of the retrieval system or the file organization actually used to store the characteristics of laboratory animals.

The use of a high-level nonprocedural query manipulation language is particularly important in management information systems and decision support systems where few users have programming knowledge [47–49]. In such situations English-like query languages are often considered to be essential because the user is then removed from the details of the processing system to the greatest possible extent. The same motivation lies behind the work in natural language query formulations in some question-answering environments.

### *B Processing Strategies

All the data base query languages mentioned earlier contain features for processing, summarizing, maintaining, and updating the stored information files in addition to the formulation of actual user queries. The following types of operations are particularly important:

1 File modification commands such as insert, delete, and modify
2 Arithmetic capabilities such as addition and subtraction
3 Assignment and print commands used to assign new or computed

values to the attributes of certain records and to print out information contained in the records

4 Aggregate operations used to obtain global values for the records stored in a given file, including average, sum, maximum, and minimum values

When the files to be processed are small and the contents remain static over long periods of time, the search and updating processes are not of much consequence. In that case, a sequential scan through the whole file might be used to process each stored record sequentially in order to carry out the necessary operations. Unfortunately, in most real situations the files are neither static nor small. As the number of records becomes larger, the number of terms needed to identify these records will also grow, and a sequential scan that processes all records sequentially one at a time becomes too slow and too costly for practical use. Typically, indexes are then created which can be used to isolate certain portions of the file without touching the remainder of the stored information. An inverted index similar to that introduced earlier for bibliographic records can then be used to identify all the records containing particular values of certain attributes. The corresponding records can also be retrieved rapidly if they are clustered, that is, stored contiguously in the main file.

As the number of terms used to identify the stored records or the number of clusters used to partition the collection grows, the single index can in turn be broken down into a hierarchy of indexes including successively fewer entries. The highest-level index is then used to provide access to a more detailed lower-level index which gives access in turn to still more detailed indexes, until in time some of the records stored in the main data file are eventually identified. This is the method used in the cluster tree search where successively smaller groupings of records are determined until at the end a few records included in a common low-level centroid are obtained.

The hierarchy of indexes and the main records file must be searched and maintained efficiently under addition, deletion, and modification of the contents. When related records are connected by pointers, additions and deletions can be handled by simple changes in the pointer arrangements as shown in Fig. 9-16a and b respectively. Eventually as more and more local changes are made a complete reorganization of the storage area becomes necessary. Such a reorganization will "compact" the space by storing in adjacent positions items that actually belong together and by plugging the holes left by deleted records.

File reorganizations may involve alterations at the conceptual level that require changes in the file schema. This is the case when the hierarchical or network arrangement is altered or when changes are made to the attribute lists that identify certain record types. Alternatively changes may be made at the physical level, for example, by creating a new level of indexing, or by changing the accessing method used to retrieve the records from storage. In either case, the file reorganization can be carried out by copying the data base to auxiliary storage and performing the necessary modifications off-line. Eventually the corrected version is then reloaded into the main store following the updating oper-

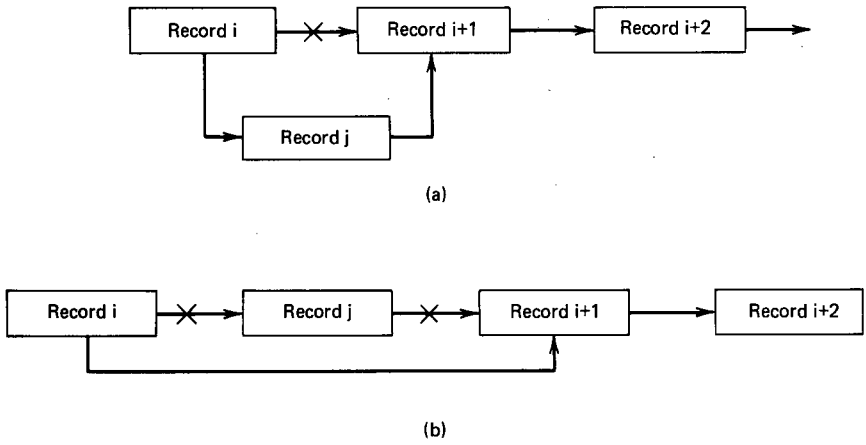



**Figure 9-16** Pointer changes used for record addition and deletion. (a) Addition of record j between record i and record (i + l). (b) Deletion of record j from chain.

ation. Alternatively, the corrections can be performed in place, while other operations proceed normally. In that case, it becomes necessary to block access to portions of the data base while the revisions are carried out. While special methods have been devised that minimize the difficulties inherent in data base updating and maintenance, any reorganization may be expected to require substantial resources especially when the size of the data base is large [50,51].

In addition to the file organization and file modification processes, attention must also be paid to the efficiency of the search and retrieval operations. A good deal is known about the tradeoffs involved, for example, in creating and maintaining an auxiliary index compared with the larger search cost incurred when an index is not available. Unfortunately, it is not possible in the present context to examine these efficiency problems in detail. Some observations relating to the search process used for the relational data base model will suffice for present purposes [52–54].

The basic problem in query optimization consists in determining a reasonable order for executing the operations necessary to retrieve the stored records in response to a particular query formulation. The term "reasonable" implies that the grossest inefficiencies should be eliminated whenever possible. The optimization problem is of special importance for relational data base queries because the easy access paths provided by the pointer structures between records are not always available in that system. In fact, unless special precautions are taken, typical query language expressions may produce very inefficient processing sequences in a relational environment.

Consider, as an example, the query formulation of expression (1) involving two relations consisting of employee name and employee number (attributes A and B) on the one hand, and employee number and manager name (attributes C and D) on the other. The first operation specified in expression (1) is the Cartesian product (AB × CD). This involves a single scan of the first relation; how-

ever, for *each* record in relation 1, all records in the second relation must be scanned to generate the single relation involving the four attributes ABCD. Assuming that the number of records in the two relations is $n_{AB}$ and $n_{CD}$, respectively, the number of records that need to be scanned is thus equal to $n_{AB}(1 + n_{CD})$. Obviously the generation of a Cartesian product such as the one specified in expression (1) is expensive for large relations.

More generally, it is obvious that relational operations designed to generate large relations from smaller ones should be avoided, whereas on the contrary the transformation of large relations into small ones should be preferred. This implies that Cartesian products as well as joins and relational union operations should be replaced whenever possible by select, project, or relational intersection; in any case, the expensive operations should be delayed in the hope of reducing the scope of these operations.

It turns out that for the query of expression (1), the product operation is totally unnecessary. Indeed only the attribute values corresponding to certain employee numbers are to be retrieved in that case, rather than the complete set of records involving all attributes. More specifically, the following transformations are possible for the query formulation of expression (1):

**1** Since the selection operation $[\text{SELECT}_{D='\text{SMITH}'}(AB \times CD)]$ involves only the records in the second relation, the SELECT operation can be migrated inside the product, thereby obtaining a reduced query specification as follows:

$$\text{PROJECT}_A(\text{SELECT}_{B=C}(AB \times \text{SELECT}_{D='\text{SMITH}'}CD)) \tag{2}$$

The Cartesian product is now much less costly, since only a few of the CD records (those corresponding to manager Smith) are involved.

**2** The condition $\text{SELECT}_{B=C}$ which follows the Cartesian product converts that product into a natural join operation between AB and the selected records from CD. The new query statement thus becomes

$$\text{PROJECT}_A\left(AB \underset{B=C}{\text{JOIN}} (\text{SELECT}_{D='\text{SMITH}'}CD)\right) \tag{3}$$

Various methods can be used to execute the join operation. A multiple scan of the relations is always possible: typically for each element (record) of relation 1 it is necessary to scan the second relation in order to find all possible elements that meet the join conditions. This requires as many complete scans of relation 2 as there are elements in relation 1. The multiple scanning can be avoided by constructing an index for the attributes of the second relation that are needed in the join operation. When an index is available, the elements of relation 2 corresponding to each element in relation 1 are immediately identified without further scanning. Another possibility consists in sorting the two relations according to the values of the attributes needed in the join prior to the actual joining operation. This makes it possible to carry out the join operation

by performing a single scan of both relations. Assuming that relation 1 has n entries and relation 1 has m entries, the multiple scan requires n × m file access operations. If the two relations were sorted and then merged, the number of operations will be n log n and m log m for the sorting, and m + n for the final merging. A comparison of n × m with n log n + m log m + n + m shows that the latter is preferable except when the file sizes are trivially small.

Whether it is worthwhile to construct an index on the join attributes or to sort the relations prior to joining depends on the size of the relations, the difficulty of the indexing or sorting operations, the size of the internal memory used in the system under consideration, and the frequency with which a given join operation must be carried out.

The following general optimization rules are useful under most operational conditions:

1 Perform all selection operations as early as possible since the effect is a reduction in the size of the relations.

2 Preprocess the files before performing a join by creating an index on the join attributes, or by sorting where indicated.

3 Assemble sequences of selections or sequences of projection operations into a single selection or a single projection; a sequence of these operations can be executed in a single scan of the relation.

4 Combine projection operations with other binary operations involving several relations to decrease the number of attributes that need to be processed.

5 Combine selection operations with a prior Cartesian product to generate a join, a natural join being generally less expensive to perform than a Cartesian product.

A substantial effort has been devoted to the optimization of the query processing function in some of the modern relational data base systems [32,52–54]. The alternative hierarchical and network systems which provide fast access paths for the most important record combinations may nevertheless be preferred to an optimized relational system in many circumstances.

## 4 DATA QUALITY

### A Integrity and Security

The preservation of data correctness is considered of great important in many data base environments. Several sources of errors are of interest: the accidental introduction of incorrect data resulting from clerical input errors or from common programming errors, and the malicious modification of information. The problem of data base protection takes on two main aspects: the *security* or *access control* mechanism which is designed to ensure that users will access only that portion of the data base which they are actually entitled to see, and the preservation of *data integrity*, which is designed to protect the data from nonmalicious errors such as the introduction of attribute values that are seman-

tically inconsistent (such as EMPLOYEE AGE = 139), or the use of duplicate key values for different records.

The security problem is normally attacked by assigning to each user a *password* designed to identify all legitimate users for access purposes. The passwords are presumed to be known only to individual users and to the system itself. In addition to the compulsory use of passwords before obtaining system access, each user is expected to identify those portions of the data base that are actually needed for each particular application. This may be done by introducing a separate *subschema* for each user in which all system objects required by that particular user are properly defined. The assignment of particular system objects to particular users implies that users receive their own *view* of the data base. Furthermore by restricting the individual users to their own user views—for example, managers to the records covering their own department only—the data objects may be protected from involuntary misuse to some extent. In particular, programming errors which force users to access portions of the data base for which they have no use can then be easily detected.

Certain *physical* protection mechanisms also exist, including especially *data encryption* methods designed to transform the data into a form which cannot be recognized by unauthorized persons. Encryption methods can be used prior to storing the data to ensure that the contents of a data base will not be revealed to unauthorized users. On the other hand, encrypted files may be difficult to sort and to access by means of standard index files.

Methods may also be available which provide *authorization* to individual users to perform certain operations. In the query-by-example system, users may be individually authorized to perform specific operations on particular files (relations), including insertion (I.), deletion (D.), updating (U.), and printing (P.). Tabular displays similar to those shown in Table 9-6 can be used to specify the necessary authorizations. The first sample authorization of Table 9-7 allows any user to read records for employees whose salary does not reach 10,000. In Table 9-7b, a specific person (J. Doe) is authorized to update the salary of a specific record.

The available security mechanisms may be expected to prevent many erroneous alterations of the data base. On the other hand, no currently available security device will permanently foil a persistent transgressor with sufficient imagination and know-how [55–57].

An important aspect of data base preservation is the preservation of data *integrity*. In many situations restrictions can be placed on the characteristics or values of certain attributes that must be observed if the data environment is to appear "sensible." For example, the value of currently used postal zip codes in the United States is less than 100,000 and greater than 00600; the length of employee names is less than 30 characters; and the age of active employees is less than 100. Integrity constraints of many kinds can be declared in advance to be enforced by the system at convenient points in time. A given constraint may, for example, be verified each time an affected record is updated, or each time the record is accessed.

| EMPLOYEES | EMP. NUMBER | EMP. NAME | SALARY |
|---|---|---|---|
| AUTR(P.) BROOKS | 123 | SMITH | < 10,000 |

(a)

| EMPLOYEES | EMP. NUMBER | EMP. NAME | SALARY |
|---|---|---|---|
| AUTR(U.) J. DOE | 123 | SMITH | 10,000 |

(b)

**Table 9-7** Authorization mechanism in query-by-example. (a) Authorization for anyone to read employee names and numbers provided salary is less than 10,000. (b) Authorization for J. Doe to update the salary of employee Smith.

In the QBE system, integrity constraints are declared by means of tabular displays similar to those used for authorizations. An example is shown in Table 9-8, which specifies that during record insertion and updating, the salary figure for any employee must necessarily amount to at least $5,000.

In addition to using direct integrity constraints declared for specific record attributes, the integrity of the data may also depend on preserving certain implied relationships between attributes. A good deal of attention has been devoted in recent years to the development of a data base design theory based on the use of attribute relationships. This design theory is most developed for the relational data base model, and depends in large measure on the specification and use of *dependencies* of various kinds between attributes. The most important dependency type is the *functional* dependency, which guarantees that the values of certain attributes are identical for different records under the assumption that the values of certain other attributes are also the same. Formally one says that attribute B *functionally depends* on attribute A (or A functionally determines B) if no two records can have a different value for attribute B, assuming that the value is identical for attribute A. Functional dependency may be denoted by drawing an arrow from the determining toward the functionally dependent element, that is, A → B. It is clear that if an attribute is a candidate key for a given relation, then necessarily the other attributes will be functionally dependent on that key.

Consider, as an example, a STUDENT relation whose relational scheme is (STUDENT NAME, STUDENT ADDRESS, COURSE, GRADE). If the student names are all distinct, then the names may function as a key. It is then clear that

STUDENT NAME → STUDENT ADDRESS

since the same student cannot have two different addresses. Similarly the combination of the student name and the course number in which the student is

| EMPLOYEES | EMP. NUMBER | EMP. NAME | SALARY |
|---|---|---|---|
| CONSTR(I.U.) | | | >= 5,000 |

**Table 9-8** Integrity declaration in query-by-example.

enrolled will determine a unique grade

STUDENT NAME, COURSE → GRADE

The functional dependencies, once determined, can be used directly for integrity checking by verifying that all dependencies are valid at all times. In addition, the dependencies are useful for the generation and use of relations in *normalized* form. The normalization process is generally carried out by decomposing each relation into component parts that are conceptually simpler than the original. The simpler normalized form is attractive because certain difficulties that may arise for nonnormalized relations as a result of data insertion or deletion are absent for the normalized case [58–60].

Consider, as an example, the STUDENT file introduced earlier. Because the values of all attributes must be specified for each record in the relational model, it is impossible to enter into the file the names and addresses of students who are not enrolled in a course. Alternatively, the names of students who drop their only course are also lost. If the basic STUDENT relation were replaced by two smaller relations called NAME FILE and GRADE RECORD, respectively, the update problems mentioned earlier would not exist. Specifically if the two new relations are defined as

NAME FILE(STUDENT NAME, STUDENT ADDRESS)
GRADE RECORD(STUDENT NAME, COURSE, GRADE)

the original relation can be reconstructed and the names of unenrolled students are preserved.

Three principal normal forms are described in the literature, known, respectively, as the *first, second,* and *third normal forms* (1NF, 2NF, and 3 NF). The first normal form characterizes a relation in which each component of each record is not further decomposable; that is, each attribute represents a nondecomposable entity. The relation used as an example in Table 9-2 is certainly in 1NF, assuming that the components of the address attribute (for example, the street number) are not meaningful by themselves.

A relation is in 2NF if it is in 1NF and if every attribute that is not a key is *fully* dependent on each key attribute. That is, the so-called partial dependencies are disallowed where a given attribute is dependent on a portion of a key attribute. In the relation R identified by the scheme R(CITY, STREET, ZIP) the combination of (STREET, ZIP) is a key, but one actually observes that the

zip code by itself determines the city (although not vice versa). Hence the functional dependency

$$\text{ZIP} \rightarrow \text{CITY}$$

represents a partial dependency on a key attribute.

A relation is in 3NF if it is in 2NF and no nontrivial *transitive* dependencies are present. A transitive dependency between attributes A, B, and C implies that $A \rightarrow B$ and $B \rightarrow C$, but $B \not\rightarrow A$. (The latter condition is needed because if $A \rightarrow B$ and $B \rightarrow A$, then attributes A and B would be equivalent and the transitivity property would become trivial.) For the previously used STUDENT relation one observes

$$\begin{aligned} &\text{STUDENT, COURSE} \rightarrow \text{STUDENT} \\ &\text{STUDENT} \rightarrow \text{ADDRESS} \\ &\text{STUDENT} \not\rightarrow \text{STUDENT, COURSE} \end{aligned} \tag{4}$$

Hence the relation is certainly not in 3NF. Assuming that the combination (STUDENT, COURSE) is used as a key, the relation is also not in 2NF because of the partial dependency STUDENT → ADDRESS. Update anomalies can occur for nonnormalized relations, as previously illustrated for the STUDENT relation. This accounts for the interest in normalization and relational decomposition.

The decomposition of relations into component parts would not be useful if methods were not available for reconstructing the original by rejoining the components without loss of information. A so-called lossless join decomposition of relations is possible under well-defined conditions. In particular, it is known that any relation has a lossless join decomposition into third normal form preserving all the original dependencies between attributes [20]. For example, the relation R(STUDENT, ADDRESS, COURSE, GRADE) has a lossless pair decomposition into $R_1$(STUDENT, ADDRESS) and $R_2$(STUDENT, COURSE, GRADE) because R may be recovered by the unrestricted join $R_1$ JOIN $R_2$ using the equivalence of the student names as a join condition.

Substantial advances are continually being made in the development of data base theories, for example through the introduction of new normal forms such as the fourth or Boyce-Codd normal form (BCNF) and the use of new types of dependencies between attributes such as the so-called multivalued dependencies [61]. One may expect that these theoretical activities will substantially affect the design characteristics of the data management systems in the future.

### **B Concurrent Data Base Operations

It has been implicit in the foregoing exposition that the programs used to access the data base operate independently of each other and that provisions are made for preventing interference between activities that may take place concur-

rently. In most data base environments the concurrent execution of many transactions (processes) is the rule rather than the exception. For example, an automated airline reservations system simultaneously receives hundreds of different inquiries all of which must be answered by consulting the same data base. The main interference problem does not of course arise from reading the data because in that case the integrity of the data is not normally compromised. Problems do, however, arise when modifications are made to the stored information in a multiuser environment. In the airline reservations system, methods must thus be available to prevent the simultaneous sale by different agents of the same airplane seat to different prospective passengers.

The basic methodology used to prevent interference between different transactions consists in applying a *locking mechanism* to a specific portion of the data base before a given transaction is to introduce modifications to the data. The lock is used to prevent other transactions from accessing the same data until the lock is removed. A transaction involving a lock mechanism thus consists of five basic steps:

Locking a portion of the data base
Reading the original data
Modifying the data
Writing the modified data
Unlocking the data base

A special *lock manager* serves in most data base systems to assign and record all existing locks, and to prevent access to locked portions of the data base [62–64].

When a locking mechanism is used to control access, special care must be taken to avoid undesirable side effects. Assuming, for example, that several transactions are waiting to access a given locked portion of the data base, the next transaction to be given access following release of the lock must not be chosen arbitrarily because some transactions could then be made to wait forever. Such a condition, known as "livelock," can be avoided, for example, by providing access on a first-come first-served basis.

A possibly more serious situation arises when two or more transactions are *deadlocked* because they each demand access to data locked by other transactions. The following order of operations leads to a deadlock situation:

1 Transaction 1 : lock data base A
2 Transaction 2 : lock data base B
3 Transaction 1 : request lock on data base B
4 Transaction 2 : request lock on data base A

In this example, the first two steps are carried out normally because different portions of the data base are affected by the locking requests for the two transactions. At step 3, transaction 1 is asked to wait because transaction 2 currently

holds a lock on data base B. At step 4, transaction 2 will similarly come to rest because another lock has already been placed on data base A. From this point on, both transactions will obviously wait forever.

Various solutions are available for preventing deadlock. For example, each transaction may be asked to request all the needed locks at the same time, at which point either the locks are all granted simultaneously or they are all rejected. Alternatively, the locks affecting a particular set of data bases may all have to be placed in some particular preestablished order. For example, if data base A necessarily precedes data base B in the established order, then the lock requests for transaction 2 are not admissible. Instead of preventing deadlock, the system can simply detect deadlock, and later *roll back* some of the transactions involved and cancel their effect on the data base. Rollback and recovery methods are mentioned later in this chapter. In general, rollback is less satisfactory than deadlock prevention because some users may then be forced to reenter a transaction that had already been started.

Even when livelock and deadlock operations are successfully avoided, the concurrent execution of different processes may still lead to unwanted interference problems. In effect the concurrent transactions must be interleaved in such a way that the end effect is the same as if the transactions were executed separately and independently of each other. The latter condition is known as "serializability." In situations where several data base transactions can be executed concurrently, a *scheduler* included in the data base system is charged with the resolution of potential conflicts between transactions. More specifically, the scheduler imposes a *protocol* on each transaction restricting the sequences of steps that a transaction may execute. Assuming that the established protocols are followed by all transactions, the scheduler will then produce a serializable schedule.

One kind of protocol which is known to guarantee serializability is the *two-phase locking protocol* which specifies that for each transaction all locking operations must precede all unlocking operations. The two-phase protocol may lead to deadlock; however, assuming that deadlock prevention methods are also used, the two-phase requirement will allow the scheduler to produce a processing order that is equivalent to that of a serial schedule. In a serial schedule, the transactions are completely independent of each other.

An example of a serial schedule is shown in Table 9-9a, where all the operations for transaction 2 precede those for transaction 1. A serializable, but not serial, schedule is shown for the same operations in Table 9-9b. It may be seen that for each transaction all locking operations precede all unlocking operations. However, the schedule of Table 9-9b is equivalent to the serial schedule of Table 9-9a where all of transaction 2 precedes all of transaction 1. A nonserializable schedule for the transactions is given in Table 9-9c. In the example of Table 9-9c, the locks on data base B cannot be placed in such a way that operation 6 of transaction 1 may precede operation 7 of transaction 2. The effect of transaction 2 on data base B is in fact lost in the schedule of Table 9-9c, because the old values for data base B are read at step 6 in Table 9-9c, whereas

**Table 9-9 Illustration of Two-Phase Locking Protocol**

| Transaction 1 | Transaction 2 |
|---|---|
| **a Serial schedule for two transactions** | |
| | 1. Read data base A |
| | 2. Perform operations on A |
| | 3. Write modified A |
| | 4. Read data base B |
| | 5. Perform operations on B |
| | 6. Write modified B |
| 7. Read data base B | |
| 8. Perform operations on B | |
| 9. Write modified B | |
| 10. Read data base C | |
| 11. Modify C | |
| 12. Write modified C | |
| **b Serializable schedule for two transactions** | |
| 1. Lock C | |
| | 2. Lock A,B |
| 3. Read data base C | |
| | 4. Read data base B |
| 5. Operate on C | |
| | 6. Operate on B |
| 7. Write modified C | |
| | 8. Write modified B |
| | 9. Unlock B |
| 10. Lock B | |
| 11. Read data base B | |
| | 12. Read data base A |
| 13. Operate on B | |
| | 14. Operate on A |
| 15. Write modified B | |
| | 16. Write modified A |
| 17. Unlock B,C | |
| | 18. Unlock A |
| **c Nonserializable schedule for two transactions** | |
| 1. Read data base C | |
| | 2. Read data base B |
| 3. Modify C | |
| | 4. Modify data base B |
| 5. Write data base C | |
| 6. Read data base B | |
| | 7. Write data base B |
| 8. Modify B | |
| | 9. Read data base A |
| 10. Write modified B | |
| | 11. Modify A |
| | 12. Write data base A |

the new modified values are read at that point by transaction 1 in Table 9-9a and b.

### **C Restart and Recovery Methods

It was seen earlier that deadlock conditions can be eliminated by "rolling back" or undoing certain transactions and letting others proceed normally. To be able to go back in time and reverse the effect of certain transactions, it is necessary to store information regarding the effect on the data base of the various operations, and the order in which these operations are executed. The normal method used for this purpose consists in maintaining a *journal* or *log* which records all the operations that may have to be undone or repeated in case of trouble [65]. In particular, whenever an item in the data base is updated, inserted, or deleted, a record describing the particular modification is added to the log. When the system malfunctions, or crashes, for one reason or another, the log can be consulted and the data base can be restored by rolling back the system to a consistent state existing at some point prior to the malfunction. From that point on the various operations can then be repeated.

In general, the information in the log is used to undo the effect of transactions that had affected the state of the data base prior to the crash. On the other hand, incomplete transactions whose results had not yet been recorded in the data base can be ignored during rollback and may simply be repeated when a restart of operations is ordered.

To limit the amount of information in the log that needs to be consulted during the recovery process, it is customary to keep periodic summaries of the state of the data base in the form of *backup copies* and *journal checkpoints*. A backup copy of the data base is a complete copy produced at infrequent intervals representing a consistent data base state. Typically a backup copy might be produced once a day. When all else fails, the backup copy can be read in and all operations may be resumed at that point. A journal checkpoint represents a summary of the state of the transactions at a given point in time. The checkpoint information consists of lists of active transactions together with appropriate status information for each transaction. Checkpoint information is recorded in the log with reasonable frequency. When a crash occurs, each entry in the log is analyzed back to the last previous checkpoint to determine its effect. In addition the checkpoint information identifies transactions that need to be undone and/or repeated.

To simplify the restart and recovery operations it is customary to recognize a *commit* point for each transaction when the transaction must be regarded as being *completed*. This implies that all operations included in the transaction are terminated and the results have been entered in the log. If a crash occurs following commitment, the effects will survive the crash, even though the newly altered information may not yet have been entered into the data base. The log specifically distinguishes committed from uncommitted transactions. Commitment of a transaction should be reflected to users who must know when

they can safely assume that the transaction need not be reentered in case of later trouble. Until the commit point is reached, a transaction can always be aborted by canceling or backing out of the transaction.

A so-called two-phase commit policy, not to be confused with the two-phase locking protocol is often followed which minimizes the problems inherent in crash recovery. Two main rules are followed:

1 No transaction can actually write into a data base before it has committed.

2 No transaction can commit before all its effects on the items in the data base have been entered into the log.

Assuming that the two-phase locking protocol is used in addition to this commit policy, and that unlocking operations occur only following commitment, it becomes impossible for any transaction to read values in the data base provided by uncommitted transactions.

When the two-phase commit policy is followed, uncommitted transactions may simply be disregarded in a crash situation because these transactions could certainly not have affected the data. On the other hand, committed transactions must be redone following the rollback operation. When the two-phase commit or some equivalent policy is not used, the recovery process is much more difficult because uncommitted transactions might then have changed the data base and other transactions might have read values written by these uncommitted transactions. The original changes to the data base introduced by the uncommitted transactions must then be undone, and so must the transactions that had read values from these transactions. This effect can propagate indefinitely and render the recovery process impossible to carry out in an effective manner. Even when an appropriate commit policy is used to distinguish transactions that must participate in the recovery process from others that can be ignored, any crash recovery procedure is costly to implement and requires substantial system resources.

### **D Distributed Data Bases

The assumption has been made up to this point that the data base under consideration is centrally managed, although of course the users might be geographically dispersed and various transactions might be executed concurrently. In actual fact many systems exist involving several different computers and a number of different data bases located in a variety of different places. In such a circumstance one speaks of a *distributed* data base system. Possibly the best-known operating distributed data base system is the worldwide airline reservations system: each participating airline uses its own data base located near its particular headquarters location; common protocols are, however, used which control the operations when information involving more than one data base is needed to answer a given query [66–70].

The usefulness of distributed data base operations has become increasingly obvious because of the popularity of the many small minicomputer systems. These systems can obviously be used to control local data bases and to perform operations of interest in local environments. Furthermore, when data are needed that are not locally available, the local computers can address requests to a network of other machines and other data bases located in various remote places.

A distributed data base environment complicates the systems organization and the resulting data base operations. A decision must first be made about the allocation of the files to the various locations, or *nodes*, of the data base network. A particular file could be kept in some unique, central place; alternatively it could be *partitioned* by allocating the various file portions to several different nodes. Finally, the file or certain file portions could be *replicated* by storing copies in several places. The use of partitioned files may reduce the number of messages and requests circulating from node to node assuming that the file portions of interest at a particular site are locally stored. When the data files are replicated, the message traffic between nodes may be substantially reduced. In fact, a tradeoff exists between the extra storage used by the data replication and the increased speed of operations resulting from the reduced communications load between the nodes.

In a distributed data base system the need for the basic physical and logical data independence is expanded to include also *location* and *replica transparency*. Location transparency implies that the user programs are independent of the particular location of the files, while replica transparency extends the transparency to the use of an arbitrary number of copies.

Once a particular file environment is created, procedures must be available for executing the various transactions and furnishing results to the requesting parties. A given transaction might be run locally; alternatively, various remote points might be asked to carry out the operations followed by the routing of responses to the originating points. The latter strategy involves a good deal of overhead in handling the message queues that may be formed at various points in the network.

It goes without saying that all operations must be carried out in a distributed environment in such a way that data integrity and consistency are maintained. This implies that special locking and update strategies must be used to ensure that all copies of a given data base are properly updated. Specifically, all files must be locked before updating, the locks must be held until the end of a given transaction, and file alterations must be broadcast through the network to all replicas before the end of the transaction. Special transaction commit policies have been invented for this purpose in distributed systems. Specifically, a transaction coordinator is named for each transaction, and the coordinator alone is empowered to commit a given transaction after querying the participating sites concerning their individual readiness to commit.

It is obvious from what has been said that the distributed data base environment creates a host of complications. Because of the current widespread

use of computer networks and the increasing availability of on-line access to computational facilities by a wide range of users, the organization and operations of distributed data base systems have become popular areas of investigation for researchers in the computer field.

## 5 SUMMARY

The questions of interest in data base management coincide to a large extent with those arising in all information retrieval environments: they involve the management of interactive multiuser systems including efficient accessing, search, and retrieval procedures, and user-friendly query formulation and output display methods. By restricting the information to be processed in data base management to well-defined, homogeneous entities, characterized by a small number of unambiguous attribute values, the indexing and content-analysis problems met in other retrieval environments are avoided. The emphasis is placed instead on the processing of mixed queries often involving partly numeric data whose values need to be compared or manipulated arithmetically before answers can be generated to the available queries. High-level procedures and languages may then be available in a data base management system for defining, organizing, processing and accessing the structured data. In addition, a data base system normally maintains data quality and security, and ensures that the user programs are insulated from the details of the logical and physical system implementation.

The strength of the data management area is the wide applicability in many different user environments, and the large number of utilities and system resources that are often furnished, including crash protection, deadlock detection, automatic construction of secondary indexes and hashing systems, integrity checks, access authorization management, and transaction concurrency and distributed data facilities. The weaknesses of the existing systems arise from the small number of well-defined operations that are implemented efficiently often at the expense of complicating operations that do not fit the standard model. Efficient access paths are normally provided for programs that use the available data structures, pointer systems, and record grouping properties of the files. When the existing facilities are not suited to the given processing requirements, the execution of the operations becomes cumbersome and time-consuming. In the relational system tradeoffs exist between the use of decomposed relations in normalized form where the integrity and correctness of the data are easily ensured, and the extra work involved in rejoining the normalized relations when necessary.

One may expect that increasing efforts will be made in the immediate future to eliminate some of the restrictions inherent in existing data management implementations. Eventually, the existing data base operations may be converted into truly flexible decision support and question-answering environments.

## REFERENCES

[1] R. N. Landau, J. Wanger, and M. C. Berger, Directory of Online Databases, Vol. 1, No. 3, Cuadra Associates, Santa Monica, California, Spring 1980, p. 72.
[2] C. Mader and R. Hagin, Information Systems: Technology, Economics, Applications, Science Research Associates, Chicago, Illinois, 1974.
[3] I. Benbasat and R. Goldstein, Data Base Systems for Small Business : Miracle or Mirage, Data Base, Vol. 9, No. 1, Summer 1977, pp. 5–8.
[4] J. P. Fry and E. H. Sibley, Evolution of Data-Base Management Systems, Computing Surveys, Vol. 8, No. 1, March 1976, pp. 7–42.
[5] R. Ashany and M. Adamowicz, Data Base Systems, IBM Systems Journal, Vol. 15, No. 3, 1976, pp. 253–263.
[6] W.M. Zani, Blueprint for MIS, Harvard Business Review, Vol. 48, November–December 1970, pp. 95–100.
[7] G. B. Davis, Management Information Systems: Conceptual Foundations, Structure and Development, McGraw-Hill Book Company, New York, 1974.
[8] J. C. Emery, An Overview of Management Information Systems, Data Base, Vol. 5, No. 2–4, December 1973, pp. 1–11.
[9] S. L. Alter, How Effective Managers Use Information Systems, Harvard Business Review, Vol. 54, November–December 1976, pp. 97–104.
[10] E. D. Carlson, J. L. Bennett, G. M. Giddings, and P. E. Mantey, The Design and Evaluation of an Interactive Geo-Data Analysis and Display System, Information Processing '74, North Holland Publishing Company, Amsterdam, 1974, pp. 1057–1061.
[11] P. Berger and F. Edelman, IRIS: A Transaction Based DSS for Human Resources Management, Data Base, Vol. 8, No. 3, Winter 1977, pp. 22–29.
[12] R. Davis, A DSS for Diagnosis and Therapy, Data Base, Vol. 8, No. 3, Winter 1977, pp. 58–72.
[13] C. F. Starmer and R. A. Rosati, A DSS for Managing Patients with a Choice Illness, Data Base, Vol. 8, No. 3, Winter 1977, pp. 51–57.
[14] E. R. McLean and T. Riesing, MAPP: A DSS for Financial Planning, Data Base, Vol. 8, No. 3, Winter 1977, pp. 9–14.
[15] R. L. Klaas, A DSS for Airline Management, Data Base, Vol. 8, No. 3, Winter 1977, pp. 3–8.
[16] C. J. Date, An Introduction to Data Base Systems, 3rd Edition, Addison Wesley Publishing Company, Reading, Massachusetts, 1981.
[17] D. C. Tsichritzis and F. H. Lochovsky, Data Base Management Systems, Academic Press, New York, 1977.
[18] G. Wiederhold, Database Design, McGraw-Hill Book Company, New York, 1977.
[19] D. Kroenke, Database Processing: Fundamentals, Modeling, Applications, Science Research Associates Inc., Chicago, Illinois, 1977.
[20] J. D. Ullman, Principles of Data Base Systems, Computer Science Press, Potomac, Maryland, 1980.
[21] C. T. Meadow, Analysis of Information Systems, 2nd Edition, John Wiley and Sons, New York, 1973.
[22] M. E. Senko, Data Structures and Data Accessing in Data Base Systems: Past, Present, Future, IBM Systems Journal, Vol. 16, No. 3, 1977, pp. 208–257.
[23] M. E. Senko, Information Systems: Records, Relations, Sets, Entities and Things, Information Systems, Vol. 1, 1975, pp. 3–13.

[24] D. D. Chamberlin, Relational Data-Base Management Systems, Computing Surveys, Vol. 8, No. 1, March 1976, pp. 43–66.
[25] E. F. Codd, A Relational Model of Data for Large Shared Data Banks, ACM Communications, Vol. 13, No. 6, June 1970, pp. 377–387.
[26] IBM Corporation, Information Management System—General Information Manual, IBM Report GH 20-1260, White Plains, New York, 1975.
[27] W. C. McGee, The Information Management System IMS/VS, IBM Systems Journal, Vol. 16, No. 2, 1977, pp. 84–168.
[28] R. W. Taylor and R. L. Frank, CODASYL Data-Base Management Systems, ACM Computing Surveys, Vol. 8, No. 1, March 1976, pp. 67–103.
[29] CODASYL, Database Task Group Report, Association for Computing Machinery, New York, April 1971.
[30] T. W. Olle, The Codasyl Approach to Data Base Management, John Wiley and Sons, New York, 1978.
[31] C. W. Bachman, On a Generalized Language for File Organization and Manipulation, Communications of the ACM, Vol. 9, No. 3, March 1966, pp. 225–226.
[32] M. M. Astrahan, M. W. Blasgen, D. D. Chamberlin, K. P. Eswaran, J. N. Gray, P. P. Griffiths, W. F. King, R. A. Lorie, P. R. McJones, J. W. Mehl, G. R. Putzolu, I. L. Traiger, B. W. Wade, and V. Watson, System R: A Relational Approach to Database Management, ACM Transactions on Database Systems, Vol. 1, No. 2, June 1976, pp. 97–137.
[33] L. R. Harris, User Oriented Database Query with the ROBOT Natural Language Query System, International Journal of Man-Machine Studies, Vol. 9, 1977, pp. 697–713.
[34] D. L. Waltz, An English Language Question Answering System for a Large Relational Database, Communications of the ACM, Vol. 21, No. 7, July 1978, pp. 526–539.
[35] E. F. Codd, R. S. Arnold, J. M. Cadiou, C. L. Chang, and N. Roussopoulos, Rendezvous Version 1: An Experimental English-Language Query Formulation System for Casual Users of Relational Data Bases, IBM Research Report RJ 2144, IBM Research Laboratory, San Jose, California, January 1978.
[36] G. G. Hendrix, E. D. Sacerdoti, D. Sagalowicz, and J. Slocum, Developing a Natural Language Interface to Complex Data, ACM Transactions on Database Systems, Vol. 3, No. 2, June 1978, pp. 105–147.
[37] P. Dell 'Orco, V. N. Spadavecchia, and M. King, Using Knowledge of a Data Base World in Interpreting Natural Language Queries, Information Processing 77, North Holland Publishing Company, Amsterdam, 1977, pp. 139–144.
[38] J. Mylopoulos, P. A. Bernstein, and H. K. T. Wong, A Language Facility for Designing Database-Intensive Applications, ACM Transactions on Database Systems, Vol. 5, No. 2, June 1980, pp. 185–207.
[39] S. C. Shapiro and S. C. Kwasny, Interactive Consulting via Natural Language, Communications of the ACM, Vol. 18, No. 8, August 1975, pp. 459–462.
[40] J. Martyn, Prestel and Public Libraries: An LA/Aslib Experiment, Aslib Proceedings, Vol. 31, No. 5, May 1979, pp. 216–236.
[41] M. M. Zloof, Query-by-Example: A Database Language, IBM Systems Journal, Vol. 16, No. 4, 1977, pp. 324–343.
[42] D. D. Chamberlin, M. M. Astrahan, K. P. Eswaran, P. P. Griffiths, R. A. Lorie, J. W. Mehl, T. Reimer, and B. W. Wade, Sequel 2: A Unified Approach to Data Definition, Manipulation and Control, IBM Journal of Research and Development, Vol. 20, No. 6, November 1976, pp. 560–575.

[43] D. D. Chamberlin and R. F. Boyce, SEQUEL: A Structured English Query Language, Proceedings of ACM SIGFIDET Workshop, Ann Arbor, Michigan, May 1974, pp. 249–264.

[44] E. F. Codd, Relational Completeness of Database Sublanguages, in Proceedings of ACM/SIGMOD Conference on Data Models, R. Rustin, editor, Association for Computing Machinery, New York, 1974, pp. 64–98.

[45] National Library of Medicine, Instructions to the Laboratory Animal Data Bank, Memorandum to LADB Users, Bethesda, Maryland, January 14, 1980.

[46] National Library of Medicine, Laboratory Animal Data Bank—Fact Sheet, Bethesda, Maryland, August 1980.

[47] K. D. Eason, Understanding the Naive Computer User, The Computer Journal, Vol. 19, No. 1, February 1976, pp. 3–7.

[48] T. P. Gerrity, Jr., Design of Man-Machine Decision Support Systems: An Application to Portfolio Management, Sloan Management Review, Vol. 12, No. 3, Winter 1971, pp. 59–75.

[49] H. T. Gibson, Determining User Involvement, Journal of Systems Management, August 1977, pp. 20–22.

[50] N. C. Shu, B. C. Housel, R. W. Taylor, S. P. Ghosh, and V. Y. Lum, Express: A Data Extraction, Processing and Restructuring System, ACM Transactions on Database Systems, Vol. 2, No. 2, June 1977, pp. 134–174.

[51] G. H. Sockut and R. P. Goldberg, Database Reorganization—Principles and Practice, ACM Computing Surveys, Vol. 11, No. 4, December 1979, pp. 371–395.

[52] P. A. V. Hall, Optimization of Single Expressions in a Relational Data Base System, IBM Journal of Research and Development, Vol. 20, No. 3, May 1976, pp. 244–257.

[53] J. M. Smith and P. Y. Chang, Optimizing the Performance of a Relational Algebra Database Interface, Communications of the ACM, Vol. 18, No. 10, October 1975, pp. 568–579.

[54] S. B. Yao, Optimization of Query Evaluation Algorithms, ACM Transactions on Database Systems, Vol. 4, No. 2, June 1979, pp. 133–155.

[55] C. Wood, E. B. Fernandez, and R. C. Summers, Data Base Security: Requirements Policies and Models, IBM Systems Journal, Vol. 19, No. 2, 1980, pp. 229–252.

[56] R. Fagin, On an Authorization Mechanism, ACM Transactions on Data Base Systems, Vol. 3, No. 3, September 1978, pp. 310–319.

[57] D. D. Chamberlin, J. N. Gray, and I. L. Traiger, Views, Authorization and Locking in a Relational Data Base System, Proceedings National Computer Conference 1975, AFIPS Press, Montvale, New Jersey, 1975, pp. 425–430.

[58] E. F. Codd, Further Normalization of the Database Relational Model, in Proceedings ACM/SIGMOD Conference on Data Models, R. Rustin, editor, Association for Computing Machinery, New York, 1974, pp. 33–63.

[59] C. Delobel and R. O. Casey, Decomposition of a Data Base and the Theory of Boolean Switching Functions, IBM Journal of Research and Development, Vol. 17, No. 5, September 1973, pp. 374–386.

[60] C. Delobel, Normalization and Hierarchical Dependencies in the Relational Data Model, ACM Transactions on Data Base Systems, Vol. 3, No. 3, September 1978, pp. 201–222.

[61] R. Fagin, Multivalued Dependencies and a New Normal Form for Relational Databases, IBM Transactions on Data Base Systems, Vol. 2, No. 3, September 1977, pp. 262–278.

[62] D. D. Chamberlin, R. F. Boyce, and I. L. Traiger, A Deadlock Free Scheme for Resource Locking in a Database Environment, Information Processing 74, North Holland Publishing Company, Amsterdam, 1974, pp. 340–343.
[63] K. P. Eswaran, J. N. Gray, R. A. Lorie, and I. L. Traiger, The Notions of Consistency and Predicate Locks in a Database System, Communications of the ACM, Vol. 19, No. 11, November 1976, pp. 624–633.
[64] R. H. Thomas, A Majority Consensus Approach to Concurrency Control for Multiple Copy Databases, ACM Transactions on Database Systems, Vol. 4, No. 2, June 1979, pp. 180–209.
[65] J. S. M. Verhofstad, Recovery Techniques for Database Systems, Computing Surveys, Vol. 10, No. 2, June 1978, pp. 167–195.
[66] H. Lorin, Distributed Processing: An Assessment, IBM Systems Journal, Vol. 18, No. 4, 1979, pp. 582–603.
[67] A. L. Scherr, Distributed Data Processing, IBM Systems Journal, Vol. 17, No. 4, 1978, pp. 324–343.
[68] J. N. Gray, Notes on Database Operating Systems, Operating Systems—An Advanced Course, R. Bayer, R. M. Graham, and G. Segmuller, editors, Springer Verlag, New York, 1978, pp. 393–481.
[69] J. B. Rothnie Jr., P. A. Bernstein, S. Fox, N. Goodman, M. Hammer, T. A. Landers, C. Reeve, D. W. Shipman, and E. Wong, Introduction to a System for Distributed Databases (SDD-1), ACM Transactions on Database Systems, Vol. 5, No. 1, March 1980, pp. 1–17.
[70] P. A. Bernstein, D. W. Shipman and J. B. Rothnie, Jr., Concurrency Control in a System for Distributed Databases (SDD-1), ACM Transactions on Database Systems, Vol. 5, No. 1, March 1980, pp. 18–51.

**BIBLIOGRAPHIC REMARKS:**

At the present time the data base area is very active. New texts appear every year, new journals are created regularly, and the literature is rapidly expanding. The following texts cover the theory and practice of data base management systems:

D. Kroenke, Database Processing: Fundamentals, Modeling, Applications, Science Research Associates, Inc., Chicago, Illinois, 1977.
C. J. Date, An Introduction to Data Base Systems, 3rd Edition, Addison Wesley Publishing Co., Reading, Massachusetts, 1981.
J. D. Ullman, Principles of Data Base Systems, Computer Science Press, Potomac, Maryland, 1980.

The text by Date emphasizes the hierarchical database systems, whereas Ullman examines mainly the relational systems.

The following texts deal largely with data structures and file access methods:

E. Horowitz and S. Sahni, Fundamentals of Data Structures, Computer Science Press, Woodland Hills, California, 1976.
G. Wiederhold, Database Design, McGraw-Hill Book Company, New York, 1977.

Several review articles cover the history and the general context of data base management systems including:

J. P. Fry and E. H. Sibley, Evolution of Data-Base Management Systems, Computing Surveys, Vol. 8, No. 1, March 1976, pp. 7–42.
M. E. Senko, Data Structures and Data Accessing in Data Base Systems: Past, Present, Future, IBM Systems Journal, Vol. 16, No. 3, 1977, pp. 208–257.
I. Benbasat and R. Goldstein, Data Base Systems for Small Business: Miracle or Mirage, Data Base, Vol. 9, No. 1, Summer 1977, pp. 5–8.

The latter is a nontechnical general review addressed to business managers.

The following articles provide an introduction to the conceptual data base models:

E. F. Codd, A Relational Model of Data for Large Shared Data Banks, ACM Communications, Vol. 13, No. 6, June 1970, pp. 377–387.
R. W. Taylor and R. L. Frank, CODASYL Data Base Management Systems, ACM Computing Surveys, Vol. 8, No. 1, March 1976, pp. 67–103.
T. W. Olle, The CODASYL Approach to Data Base Management, John Wiley and Sons, New York, 1978.
D. D. Chamberlin, Relational Data-Base Management Systems, Computing Surveys, Vol. 8, No. 1, March 1976, pp. 43–66.

Many journals contain material in the data base management area, including periodicals covering the practical aspects addressed to relatively nontechnical audiences such as:

Computerworld
Data Base
Datamation
Harvard Business Review

Additional more technical and theoretical material is contained in the following journals:

ACM Transactions on Database Systems
IBM Systems Journal
Information Systems
Information Technology: Research and Development
Journal of Systems Management

## EXERCISES

**9-1** Use the chart of Fig. 9-1 to place the following types of information systems in appropriate positions. Explain your choice in each case.

**a** Question-answering systems
**b** Public libraries
**c** Private industrial or special libraries

**d** Poison control centers
**e** Automated inventory control systems

**9-2** Consider a data base management system designed to maintain the student-course records needed by the faculty and administration in a particular school. Specify the types of records that must be included in the system and design a user (external) data base schema as well as the conceptual and the internal data base schemas.

**9-3** Given the two relations M and N, carry out the following operations in the relational algebra:

| A | B | C |
|---|---|---|
| One | Sine | Plus |
| Two | Cosine | Plus |
| Three | Tangent | Minus |
| Seven | Cotangent | Minus |

Relation M

| C | D | E |
|---|---|---|
| Plus | Sine | Seven |
| Minus | Sine | Two |
| Plus | Tangent | Three |
| Minus | Cotangent | Two |

Relation N

**a** The Cartesian product $N \times M$
**b** The natural join M JOIN N
**c** The restricted join $N \underset{B \neq D}{\text{JOIN}} M$
**d** The projection $\text{PROJ}_{3,1}\ N$

**9-4** List the advantages and disadvantages of each of the three main data base models including the relational, hierarchical, and network models. What mechanisms are available to obtain file access to data bases organized according to the various model specifications? How does one search the stored data in each model?

**9-5** Why is it useful to make special provisions to implement many-to-many relationships in some data base systems? How are many-to-many relationships implementated in relational, hierarchical, and network systems?

**9-6** Design relational, hierarchical, and network data bases that include the following relationships between record types
**a** Managers responsible for many projects
**b** Managers responsible for many employees
**c** Employees assigned to many projects
**d** Projects carried out by many employees
Describe how the three data base organizations would be used to retrieve the names of all employees that report to several managers, and the number of managers responsible for more than one project.

**9-7** List the various circumstances that render necessary the protection of data bases against unauthorized access. What methods can be used to provide the required protection, and how do these methods operate?

**9-8** Why is it necessary to protect the integrity of the stored data in a data base management system? What methods are available for integrity preservation? In which way are integrity considerations used in specifying the structure of relational data base systems?