3.14 A software development company's employee records contain the following information:

ID (10), Name (25), Position (10), Age (2), Qualifications (9), Projects (10 repeated)

The value in the parentheses is the size of each entry in bytes. An employee can be involved in a number of projects at the same time; thus, this field is repeated. An internal coding mechanism groups qualifications into three types, each requiring 3 bytes to encode. The age of employees is divided into 10 groups. The total number of employees is 500 and, at any given time, up to 100 projects are handled. The file is to be maintained on disk with a physical block size of 4096 bytes. The pointer size for addresses is 4 bytes.

(a) Design file organizations for each of the access methods listed below that at least satisfy the retrieval/query transactions, also specified below, as efficiently as possible. Diagram the organization and discuss how your file organization satisfies the retrieval requirements.

Access methods: Index-sequential, inverted, and B^+-tree.
Retrieval requirements (specified in %):

1. List employees by name in alphabetical order (10%).
2. Print data for employees in some age group and with certain qualifications (50%).
3. Print names and current projects of employees with certain qualifications and holding certain positions (40%).

(b) Compare your design with organizations based on a single type of access method with respect to space and access time. In the derivation of the access time, use the following terms:

Block access time (random): t_R
Block Access time (next in sequence): t_S

Method	*Space*	*Total Access Time*		
		A	*B*	*C*
Your Design				
Index-sequential				
Inverted				
B^+-tree				

(c) Which access method minimizes total access time for all three application types? (Be sure to take transaction frequencies into account.) If accesses for application B also required the changing of age and qualifications, would this method still be the most efficient? Justify your answer.

3.15 The manufacturer's specifications for a disk drive are:

Number of surfaces	20
Number of tracks/surface	800
Number of sectors/track	20
Number of bytes/sector	512
Rotational speed	6000 rpm
Time to move arm to adjacent cylinder	5 msec
Average time to move arm to any cylinder	20 msec

(a) How many cylinders will be required to store 100,000 records each 100 bytes long if no logical record is split across sector boundaries?

(b) If the key plus cylinder-cum-track addresses require 8 bytes, when the above file is created as an indexed-sequential file with cylinder and track indexing, estimate the average time to locate a record. Assume that there are no overflow records and that the search of an index or sector, after having been transferred to main memory, is negligible.

3.16 Consider the cylinder of an index-sequential file as shown below. Each cylinder has six surfaces and a surface has four sectors. Each sector can hold three records. Surface 05 is used for the overflow records. ($\perp$ indicates null pointers)

Cyl.	Surface	Sectors			
41		1	2	3	4
	00	Tr.Index	A_1, A_4	A_5, A_6, A_9	A_{10}, A_{13}
	01	A_{17}, A_{18}	A_{20}	A_{28}, A_{29}	A_{30}, A_{31}, A_{36}
	02	A_{42}, A_{43}	A_{45}, A_{46}, A_{48}	A_{51}, A_{52}, A_{56}	A_{59}, A_{61}
	03	A_{75}, A_{76}, A_{78}	A_{79}, A_{80}	A_{83}	A_{89}, A_{91}
	04	A_{93}, A_{94}	A_{96}, A_{98}	A_{100}	A_{120}, A_{125}
	05	$\perp$	$\perp$	$\perp$	$\perp$

Give a track index that captures the current state of the cylinder. Also give the status of the cylinder and track index after the following operations have been performed:

I A_{34}, I A_{41}, I A_{95}, D A_{83}, I A_3, I A_{82}, I A_{84}, I A_{33}, D A_{36}, I A_2, D A_4, I A_{122}, D A_{125}, I A_{124}, I A_{54}, D A_{61}, I A_{60}

where I represents the insert, and D the delete, operation.

3.17 Create an index-sequential file using three cylinders, each of which has eight tracks. Up to four records can be stored in each track. Make appropriate provisions for overflow. The file is created initially with the following records in the order given:

132, 38, 87, 64, 88, 40, 759, 12, 459, 45, 362, 85, 835, 638, 414, 820, 41, 91, 29, 194, 517, 491, 524, 294, 43, 185, 791, 139, 59, 44, 11, 414, 37, 184, 472, 39, 88, 42, 758, 460, 412, 48, 415

Indicate the reorganization of the file if the following records are subsequently deleted and added. D preceding the key indicates that the record is to be deleted; A indicates that it has to be added:

D91, A92, D44, A43, A47, A46

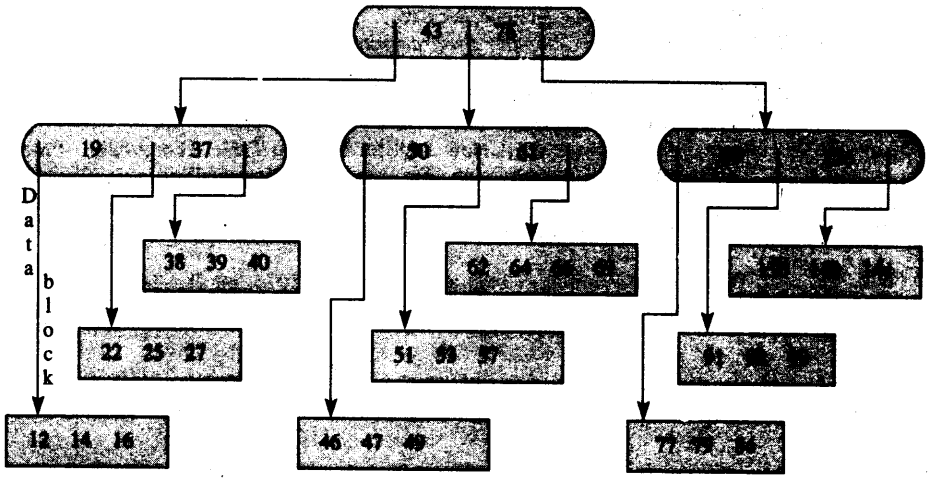
3.18 Comment on the differences between index-sequential files and B^+-tree file organizations. Compare them for use wherever an indexed access may be required.

3.19 Give algorithms for the insertion and deletion of records in a B^+-tree.

3.20 In a B-tree file, pointers to the blocks containing the records exist even in the index level nodes. How does this alter the algorithms for insertion and deletion that you wrote for Exercise 3.19? Comment on the relative advantages and disadvantages of B-trees and B^+-trees.

3.21 The accompanying figure shows the B^+-tree index and the leaf nodes of a B^+-tree of order 3. The blocks containing the leaf nodes hold the actual records. (Only the key values are shown in the figure.) Each block must hold at least three and at most five records. Show the structure of the index after the following records are inserted or deleted. D preceding the key indicates that the record is to be deleted; A indicates that it has to be added:

D91, A98, D44, A43, A47, A46



Bibliographic Notes

Discussion of storage devices and data organization methods is found in computer manufacturers manuals. A discussion on blocking and buffering techniques appears in (Wate 76). The handling of sequential files is the subject of many a textbook on programming; see also (Dwye 81). (Lum 71) presents hashing techniques as does (Litw 80). Index sequential files are the subject of a number of IBM manuals (IBM 1, IBM 2). Extendable hashing is discussed in (Fagi 79). B-tree indexes are presented in (Baye 72). A number of textbooks cover the areas of files and data structures, including (Ghos 86), (Harb 88), (Hans 82), (Horo 82), and (Knut 73). The discussion in this chapter is based to a great extent on (Goya 87).

Bibliography

(Baye 72) R. Bayer, "Symmetric Binary B-trees: Data Structure and Maintenance Algorithms," *Acta Informatica*, 1(4), 1972, pp. 290–306.

(Come 79) D. Comer, "The Ubiquitous B-tree", *ACM Computing Surveys*" 11-(2), June 1979, pp. 121–137.

(Desa 89) B. C. Desai, "Performance of a Composite Attribute and Join Index," *IEEE Trans. on Software Engineering* 15-(2), February 1989, pp. 142–152.

(Dwye 81) B. Dwyer, "One More Time—How to Update a Master File," *Communications of the ACM* 24 (1), 1981, pp. 3–8.

(Fagi 79) R. Fagin, J. Nievergelt, N. Pippenger, & H. R. Strong, "Extendible Hashing—A Fast Access Method for Dynamic Files," *ACM Trans. on Database Systems*, 4-(3), September 1979, pp.315–344.

(Ghos 86) S. P. Ghosh, *Data Base Organization for Data Management*, 2nd ed. Orlando, FL: Academic Press, 1986.

(Goya 87) P. Goyal, "File Organization," *Computer Science Report*, Concordia University, Montreal, 1987.

(Hans 82) O. Hanson, *Design of Computer Data Files*, Rockville, MD: Computer Science Press, 1982.

(Harb 88) T. R. Harbon, *File Systems Structures and Algorithms*. Englewood Cliffs, NJ: Prentice-Hall, 1988.

(Horo 82) E. Horowitz & S. Sahni, *Fundamentals of Data Structures*, 2nd ed. Rockville, MD: Computer Science Press, 1982.

(IBM 1) "Introduction to IBM Direct Access Storage Devices and Organization Methods," IBM Manual GC 20164910.

The data models introduced in Chapter 2 differed only in the manner in which relationships among data are represented. In this chapter we concentrate on the relational data model (RDM), which was formally introduced in 1970. Since that time, it has undergone extensive study. The relational model frees the user from details of storage structures and access methods. It is also conceptually simple and, more importantly, based on sound theoretical principles that provide formal tools to tackle problems arising in database design and maintenance.

Numerous different formulations of the RDM have been presented and recently interest has been shown in its formalization. We shall, however, take a semiformal approach.

4.1 Introduction

In practice we can distinguish between entities and the relationships that exist between them. In modeling, we represent an entity set in the database by a set of its properties. However, only those properties of the entity type of interest to the application are used in the model. A data model allows the capturing of these properties using its data structures. Note that the association between the properties is only implicitly captured, i.e., we do not state what kind of association exists between the properties.

Furthermore, we may wish to retrieve or update the stored data and for this purpose a data model supports certain operations. The data may also need to conform to certain consistency and integrity rules, as in the case of a bank's rule that a customer's account balance remain nonnegative (i.e., ≥ 0). These constraints are specified as integrity rules.

The relational data model, like all data models, consists of three basic components:

- a set of domains and a set of relations
- operations on relations
- integrity rules

Each of these components is illustrated in the following examples.

Example 4.1 In this simple example we model certain properties of a number of database management systems (DBMSs). Let us assume that we want to maintain a database of these DBMSs. This database will register their names, the particular data models employed, and the company that developed and markets the DBMSs. Some of these DBMSs are shown in the table SOME_DBMS in Figure A. ■

Figure A Sample relation SOME_DBMS *(Name, Data_Model, Company)*

SOME_DBMS

Name	*Data_Model*	*Company*
Data	Network	WXY Inc.
Data-R	Relational	WXY Inc.
ISS	Hierarchical	BCD Systems
ISS/R	Relational	BCD Systems
ISS/R-PC	Relational	BCD Systems
Tables	Relational	ABC Relational Systems Inc.

■

From our knowledge of the relational model gained in Chapter 2, we can identify SOME_DBMS as a relation with the attributes *Name*, *Data_Model*, and *Company*. The fact that a relation has certain attributes is specified by its scheme, usually written as **RELATION_SCHEME_NAME***(Attribute_Name*$_1$, *Attribute_Name*$_2$, . . .*)*. Each attribute is defined over a set of values known as its domain. For our sample relation, the scheme can be specified as **SOME_DBMS***(Name, Data_Model, Company)*. The relation SOME_DBMS shown in Figure A in Example 4.1 consists of six tuples, i.e., the **cardinality** of the relation SOME_DBMS is six. The number of attributes in the relation scheme is called its **degree** or **arity**. The degree of the scheme **SOME_DBMS** is three. Each tuple captures the association among the properties name, data model, and company of a DBMS package. Here the attribute *Name* can be used to uniquely identify a given DBMS and the corresponding tuple in the relation.

Just as we are able to model an entity and its properties by a relation, we can model relationships between entities using a relation. This is illustrated in Example 4.2. In Section 4.2 we shall study the relational database structures in a more formal manner.

Example 4.2 Certain DBMSs of Example 4.1 are used in particular applications. The application can be modeled using the budget code of the application as an identifying attribute or key and the name of the application. Some tuples for the relation APPLICATION*(App_Name, Budget_Code)* are shown in part i of Figure B. The E-R diagram of the relationship between APPLICATION and SOME_DBMS, named *WHERE_USED*, is shown in part ii of the figure. We can record the information about this relationship in the relation WHERE_USED by pairing the keys from the entities SOME_DBMS and APPLICATION. This relationship can be expressed as a relation, some tuples of which are shown in part iii of Figure B. ■

required rows of the table of Figure D are then identified to respond to the above query. Note that in joining the rows of the two tables, we only join those rows or tuples that have the same value for the attribute *Name* that is common to both these relations. As we will see in Section 4.3.2, the relations shown in Figures D and E, are the result of the so-called equi-join operations. ■

The number of tuples in the join of SOME_DBMS and VERSION is the same as those in VERSION because a tuple in SOME_DBMS has the same value of the *Name* attribute as a tuple in VERSION. Note that the two occurrences of the attribute *Name* in the join can be distinguished by preceding each with the corresponding relation name. The first attribute is labeled VERSION.*Name* and the second similarly named attribute is called SOME_DBMS.*Name*.

Figure D in Example 4.3 demonstrates that many of the tuples in the resulting tables are not required for answering the query. We could have approached the selection on the table of Figure A in Example 4.1, choosing only relational DBMSs and thereby giving a joined table as shown in Figure Ei in Example 4.3. But if we had selected only those rows or tuples from Figure B in Example 4.2, released after 1984, and joined this reduced set of tuples with the table SOME_DBMS, we would get a smaller table as illustrated in Figure Eii in Example 4.3. The response to the query is obtained by selecting only those tuples from one of the tables in Figure E that satisfy the two conditions of the query (in other words, taking a "horizontal subset" of the tables of Figure E). The resulting tuples are given in Figure 4.1a. The names of the companies are obtained by taking a "vertical subset" of the table on the column *Company* (in other words, **projecting** the table of Figure 4.1a on the column *Company*). The result is shown in Figure 4.1b. The method of determining which operation to perform first is the topic of query optimization, which we discuss in Chapter 10.

The **join** is just one way in which data in a relational database can be manipulated. Several kinds of data manipulation languages have been defined for the relational model. Most relational data manipulation languages are more assertional than procedural. In a purely assertional data manipulation language the target data are specified by stating their properties instead of describing how they can be retrieved. The majority of languages are based on a combination of relational algebra and re-

Figure 4.1 (a) Selecting only some tuples from the join of relation SOME_DBMS relation VERSION and (b) projecting on the column *Company*.

VERSION. *Name*	*Release*	*Year*	SOME_DBMS. *Name*	*Data_Model*	*Company*
ISS/R-PC	1.0	1985	ISS/R-PC	Relational	BCD Systems
Data-R	3.0	1986	Data-R	Relational	WXY Inc.
Tables	1.0	1987	Tables	Relational	ABC

(a)

Company
BCD Systems
WXY Inc.
ABC

(b)