



First Edition (April 1970)

Requests for copies of IBM publications should be made to your IBM representative or to the IBM branch office serving your locality. Address comments concerning the contents of this publication to IBM Corporation, DPD Education Development - Publications Services, Education Center, South Road, Poughkeepsie, New York 12602.

© Copyright International Business Machines Corporation 1970

CHAPTER 1: DATA MANAGEMENT

AND INFORMATION	1
Introduction	1
Data Management	1
Information	1
User Information	3

CHAPTER 2: FIELDS AND RECORDS

Fields	5
Types of Data Representation in Fields	5
Encoding and Compression	5
Virtual Data	6
Designing Logical Records	6
Fixed- and Variable-Length Logical Records	8
Logical Record Identification	8
Logical Data Structures	8
Simple Data Structures	8
Hierarchic Data Structures	10
Networks	10
Organizing Field Within Records	13
Locating Fields	13
Delimiting Fields	14
Representing Data Structures	16

CHAPTER 3: DATA STORAGE DEVICES

Introduction	18
Attributes	18
Attaching Devices to Systems	19
Control of Devices	19

CHAPTER 4: DATA ORGANIZATION

Introduction	21
Sequential Organization	21
Direct Organization	23
List Organization	26
Representing Data Structures	27

CHAPTER 5: FUNCTIONS OF DATA

MANAGEMENT	35
Introduction	35
Mapping	35
Information Context and Data	
Independence	38
Data Management Functions	40
Events	43
Data Flow	45

CHAPTER 6: DATA BASE SYSTEMS

CONCEPTS	47
Introduction	47
Information System	47
Design Criteria of a Data Base System	47
Event-Driven Systems	49
Data Independence	49
Binding Time	50
Degree of Independence	50
The System Administrator	51

INDEX	52
-------	----



Chapter 1: Data Management and Information

INTRODUCTION

The purpose of this publication is to introduce the reader to the concepts of data management and information processing. It is intended to address both programmer and non-programmer alike, although not all sections will be of interest to experienced computer personnel.

It is important to distinguish between data management, as described in this publication, and data management described in the Operating System and Disk Operating System publications. Many of the data management concepts expressed in this manual have not been implemented in the Operating System or Disk Operating System. It is the intent of this publication to introduce the reader to data management concepts that will apply to future systems as well as to current implementations.

The terms 'user' and 'information system' will be used in this chapter. An information system provides information on request. A 'user' of an information system can obtain and alter information contained in the system, but cannot alter the system.

DATA MANAGEMENT

Data management is the control, retrieval, and storage of information to be processed by a computer. Each of these three areas of data management is an essential function of any information system.

Control

Control is the authorization and supervision of the data management process. Authorization is the validation of a user's right to access or modify the information in the system. Supervision includes monitoring the location of information, insuring against data loss (data integrity), and insuring that the information in the system is current.

Retrieval

Retrieval is the process of locating, structuring, and ordering the information in the system for the user of the system. Locating information is determining what data is required and where it may be found. If the information is not in a form suitable to the user it must be structured to meet his needs and perhaps ordered in a different sequence.

Storage

Storage is the technique for representing the information both logically and physically on a storage device such as a disk, tape, or punched card. It includes the order in which the information is stored, the way it may be physically accessed or addressed, and the physical method of representing the data itself.

The data management system functions of control, retrieval and storage will be discussed in more detail (with examples) in another section of this manual.

INFORMATION

Before we can talk further about data management, it is necessary to examine information and how it is represented and used in an information processing system.

Information is ideas and facts about things; people, places, machines, etc. We shall refer to these things as entities. We record information about entities. Figure 1 shows some entities and some information about them. The information is in English text and roughly corresponds to how we think about each entity. Much of the information shown in Figure 1 is implicit and a result of our own understanding of the nature of a person or a part.

If we attempt to make the implicit part of the information explicit, the result is shown in Figure 2, Part 1.

We can now take these results and remove the part of the information that applies uniquely to John Jones or Part 573. The result is in Figure 2, Part 2.

We can see that information about entities is composed of at least two parts: the context (Figure 2, Part 2) and data (Figure 2, Part 3). The context is the same for like entities (People) but different for different entities (People, Parts).

In addition to context and data, it is necessary to know how the data is represented. For example, with the data "John Jones" we must know that blanks separate the first name from the last, and that the order of the names is first name first. Information about entities therefore consists of 3 parts:

Information = Context + data + data representation
(see Figure 3).

Entity		
Person		John Jones is an Accountant who works for the ABC Company in Department 5A. He is 40 years old, married, and has 3 children aged 14, 10, and 8. His salary is \$250 per week. He lives at 801 Main Street, Poughkeepsie, N.Y. His Social Security Number is 999-99-9999.
Part		The gear with Part Number A573 is 5 inches in diameter with 40 teeth, and fits a 1/2 inch shaft. It is made of aluminum. It costs \$5.00 to manufacture and sells for \$7.50. There are 100 of these gears in stock.

Figure 1. Information about Entities

Entity	Information (Part 1)	Context (Part 2)	Data (Part 3)
Person	John Jones is the <u>name</u> of an <u>employee</u> whose <u>occupation</u> is Accountant, works for <u>employer</u> ABC Company in Dept. 5A. He has an <u>age</u> of 40 yrs., a <u>marital status</u> of married. His <u>salary</u> is \$250/wk. He lives at <u>address</u> 801 Main Street, Poughkeepsie, N. Y. His <u>Social Security Number</u> is 999-99-9999.	_____ is the <u>name</u> of an <u>employee</u> Whose <u>occupation</u> is _____, works for <u>employer</u> _____ in Dept. ____. He has an <u>age</u> of _____, a <u>marital status</u> of _____. His <u>salary</u> is _____. He lives at <u>address</u> _____. His <u>Social Security Number</u> is _____.	John Jones Accountant ABC Company 5A 40 years married \$250/wk 801 Main St., Pok., N.Y. 999-99-9999
Part	The part with <u>part name</u> gear has a <u>part number</u> of A573. It has a <u>diameter</u> of 5", has <u>number of teeth</u> 40 and has a <u>shaft diameter</u> of 1/2 inch. It is made of <u>material</u> aluminum. It has a <u>manufacturing cost</u> of \$5 and a <u>selling price</u> of \$7.50. There is a <u>quantity on hand</u> of 100.	The part with <u>part name</u> ____ has a <u>part number</u> of _____. It has a <u>diameter</u> of _____, has <u>number of teeth</u> ____ and has a <u>shaft diameter</u> of _____. It is made of <u>material</u> _____. It has a <u>manufacturing cost</u> of ____ and a <u>selling price</u> of _____. There is a <u>quantity on hand</u> of _____.	gear A573 5" 40 1/2" Aluminum \$5.00 \$7.50 100

Figure 2. Information – Context and Data

Information		=	Information				
			Context	+	Data	+	Representation
Entity			Information Attributes		Data Values		Data Attributes
Person	John Jones is an Accountant in Department A26 whose gross pay is \$250 per week. His Social Security Number is 999-99-9999.		NAME		Jones, John		20 Alpha Characters Last Name First
			S. S. #		999-99-9999		11 Alpha Characters
			DEPT. #		A26		3 Alpha Characters
			GROSS PAY		\$250		5 Decimal Digits with Decimal Point between second and third digit
			OCCUPATION		Accountant		10 Alpha Characters
Part	The gear with Part Number A573 is 5" in diameter, has 40 teeth, and is constructed of aluminum. It sells for \$7.50 and costs \$5.00 to manufacture.		PART NUMBER		A 573		4 Alpha Characters
			DIAMETER		5		1 Decimal Digit Inches
			NUMBER TEETH		40		2 Decimal Digits
			MATERIAL		Aluminum		8 Alpha Characters
			SELL PRICE		\$7.50		3 Decimal Digits 2 Decimal Places
			MAKE PRICE		\$5.00		3 Decimal Digits 2 Decimal Places

Figure 3. Information – Context, Data and Data Representation

In conventional data processing the data is stored separately from the context and data representation. The reason is that, as we have shown, the context and representation is common to all like entities. The data is stored on tape, disk, or cards, and the context and representation is part of the computer program that retrieves the data. It is the function of data management to build meaningful information by bringing together the proper context, data, and data representation. The characteristics of an entity that define the context (for the entity "people" they are Name, Address, Employer, Social Security Number, etc.) are called Information Attributes (see Figure 3). A user will define the context of the information he will use by a list of information attributes for which there are recorded corresponding data values.

USER INFORMATION

The user of an information system wishes to know certain facts about certain entities. To retrieve these facts he must define the entities of interest, and for each entity, a list of information attributes. For example, if the manager of the Personnel Department wants to know the weekly salary of all the female employees in the Accounting Department (A45) he would define the entities of interest as:

ENTITIES – Female Employees in Department A45

and the information attributes as:

NAME
WEEKLY SALARY
SEX
DEPARTMENT
MAN NUMBER

The list of information attributes for each entity is a logical record. A logical record has meaning only to the person using the system to obtain information.

The set of entities that is presented to the user is called a file. In the previous example there may be 20 female employees in Department A45. The 20 entities make up the user's file. Figure 4 shows a logical record and file for PARTS.

Up to this point we have not mentioned the relationship between a user's file of logical records and the physical representation and layout of the information. The method by which the logical information is obtained from the physical information is the key to understanding data management. Figure 5 shows the subjects that must be understood before we can continue the discussion of data management. Read the associated chapters in those areas with which you are not familiar.

Entity	Definition of Logical Record (Context)	File (Data)	
Gear	Information Attributes Definition of Logical Records	Part Number A573	
		Diameter 5	
		Number of Teeth 40	
		Material Aluminum	
		Sell Price \$7.50	
		Make Price \$5.00	
		B614	
		6	
		70	
		Steel	
		\$10.00	
		\$ 8.00	
		C720	
		1	
		100	
		Brass	
\$50.00			
\$40.00			

Figure 4. Information – Logical Records and Files

Topic	Chapter
User Concept of Data	Chapter 1 Data Management and Information
Organization of Data Within Logical Records	Chapter 2 Fields and Records
Data Storage Device Characteristics, Volumes, and Physical Records	Chapter 3 Data Storage Devices
Data Organization Concepts and Techniques	Chapter 4 Data Organization
To Continue Discussion of Data Management	Chapter 5 Functions of Data Management

Figure 5. Prerequisite Topics for Understanding Data Management

Chapter 2: Fields and Records

FIELDS

In Chapter 1 we discussed information as the sum of context, data, and data representation. A field is the smallest unit of information of interest. A field has a meaning and a name (context), a value (data), and a representation. Examples of fields are shown in Figure 6. A collection of fields relating to the same entity is a logical record.

Types of Data Representation in Fields

There are many ways of representing data both internal (main storage) and external (disk, tape, cards) to a digital computer. We will discuss those representations that lend themselves to processing by System/360 hardware and language compilers (PL/I, COBOL, ALGOL, Assembler Language, FORTRAN, RPG).

The data representations are:

Arithmetic – represents numeric data

- Decimal Fixed Point
- Decimal Floating Point
- Binary Fixed point
- Binary Floating Point

String – contiguous sequence of characters or binary digits

- Bit String (string of binary digits)
- Character String (string of alphameric characters)

In general, arithmetic data is used for fields that are used in arithmetic calculations and string data is used for all other data types.

Detailed descriptions of the use and processing of these data representations can be found in other IBM publications. Some of these publications are: IBM System/360 Operating System PL/I (F) Language Reference Manual (GC28-8201), IBM System/360 Operating System COBOL Language (GC28-6516), and IBM System/360 Principles of Operation (GA22-6821).

Encoding and Compression

In addition to the different data representations listed above, data can be represented within character and bit strings in encoded or compressed form.

Encoding is translating data from one form to another. Some examples are shown in Figure 7. Encoding is used to conserve either external storage space or to facilitate processing. In example 2, Figure 7, if 'Color' is stored as a fixed-length character string, 6 characters are required in the logical record. If 'Color' is encoded as shown, at least 5 characters will be saved. If many logical tests (and, or, not) are being performed on 'Color', the encoding to bit string may make processing faster and easier. Encoding can also be used for data security by encoding so that the stored data value is meaningless unless the translation scheme is known.

Context		Data	Representation
Meaning	Name	Value	
The Man Number of Entity <u>Employee</u> .	MANNO	778175	6 Alpha Characters (Character String)
Outside Diameter of Entity <u>Pipe</u> .	SIZE	15	2 Decimal Digits Unit of Measure - Inches (Decimal Fixed Point)
Address of Entity <u>Vendor</u> .	ADDRESS	15 Main St., Arlington, Va. 60102	30 Alpha Characters Street, City, State, ZIP (Character String)

Figure 6. Examples of Fields

ENCODING

COMPRESSION

Information Attribute	Possible Field Data Values	Max. Field Size	Translation (→)	Result	Result Length
1	Language GERMAN ENGLISH FRENCH SPANISH	7 Characters	GERMAN → G ENGLISH → E FRENCH → F SPANISH → S	G E F S	1 Character
2	Color RED BLUE ORANGE YELLOW GRAY	6 Characters	RED → 001 BLUE → 010 ORANGE → 011 YELLOW → 100 GRAY → 101	001 010 011 100 101	3 Bits
3	Sex MALE FEMALE	6 Characters	MALE → 1 FEMALE → 0	1 0	1 Bit

Possible Data	Length	Compression Algorithm	Result	Result Length	
4	bbbJOHNbbbbbb bbbJONESbbbb bbFREDbbbbbb bbbbbbbbbb	25 21	Remove Blanks and Replace with a Count of Number of Blanks Removed	3JOHN9JONES4 2FRED15	12 7
5	0015700 0016300 0019560 0187370	7	Remove First and Last Digit Since it Does Not Change	01570 01630 01956 18737	5

Figure 7. Encoding and Compression

Compression is condensing repetitive information (Figure 7). Compression, like encoding, is used to conserve external storage space. In Figure 7, example 4, the data values can contain many blanks. If the blanks are removed and a count is inserted to indicate the number removed, the length of the stored field is greatly reduced. The original data value can be reconstructed using the count field.

Virtual Data

In the fields discussed in the previous sections, data physically exists that corresponds to a field name. For example, if a user defines a field as YEAR-TO-DATE GROSS, a data value exists on a storage media (tape, disks, cards) which represents the Y-T-D Gross. Another type of data that can be useful in the information systems is virtual data. Virtual data does not physically exist on a storage device, but is calculated from other values. For example, a user defines a field in his Logical Record as YRS-OF-SERVICE. The data that is stored is DATE-OF-HIRE. If the system knows that the relationship between YRS-OF-SERVICE and DATE-OF-HIRE is Y-O-S = Today's date - (D-O-H), it can then perform this calculation and present the result to the user. The user need not be aware that YRS-OF-SERVICE is virtual data.

DESIGNING LOGICAL RECORDS

It is important in designing logical records to understand each of these data types and to be able to answer the following questions:

1. How meaningful is the information in this field?

Since we are designing a logical record for a user of the information system, only those fields that have meaning for his application should be included.

2. How will the field be used?

The data representation of the field should be consistent with the predominate usage of that field. For example, if the field is used 80% of the time for arithmetic calculations and 20% of the time for display on a report, it should be stored in arithmetic form (decimal or binary, fixed or float).

3. Are there language restrictions on a particular data representation?

COBOL, FORTRAN, ALGOL, and RPG all have strengths and weaknesses with respect to processing certain data representations. The data representation should be chosen to fit the language that will be used for the particular application.

4. Is it necessary to be machine independent?

Certain data representations (fixed decimal, fixed binary), when implemented for a particular computer, make the data "machine dependent", that is, this data cannot be processed by a computer of different design. Character string data is generally the most compatible data representation.

5. Is the field easy to process?

The data in the field should be as directly usable as possible. For example, a field that contains a code for SEX can be encoded 1 for Male, 2 for Female; however, to display the information on a report would require a translation (1 to MALE, 2 to FEMALE). If the SEX were encoded M for Male, F for Female, a translation may not be required.

Information Attribute	Sample Data Values	Range of Data Values	Data Representation	Maximum Length
Student Name	John Jones	Unlimited	Character String	30
Home Address	801 Main St. Pok, N.Y.	Unlimited	Character String	30
Campus Address	Barker Hall	Barker Hall Jones Hall Brown Hall Davis Hall Off Campus	Character String	11
Class	1972	1960 - 1980	Character String	4
Enrolled Date	09/30/66	1960 - 1970	Character String	8
Tuition	\$500	Unlimited	Character String	4
Room Fee	\$200	Unlimited	Character String	4
Billing Address	801 Main St. Pok, N.Y.	Unlimited	Character String	30
College	Engineering	Unlimited	Character String	15
Major	Elect. Eng.	Unlimited	Character String	10
Grade/Course	Math 101 – B Chem 3 – A Engl 5 – D	Unlimited	Variable-Frequency Character String	10 Each Course
Previous Colleges	None	Unlimited	Variable-Frequency Character String	20 Each College
≈	≈	≈	≈	≈

Figure 8. Available Information about Students

6. How will the data representation of this field affect the overall efficiency of the process?

Certain data types in combination with encoding and compression conserve internal and external storage; others conserve processing time. These tradeoffs should be considered in the design of a field.

7. Will changes in the data values of a field affect the representation?

In any encoding or compression scheme, a prior knowledge of the possible data values is required. If these values change at some later time, extensive program changes may be required.

To illustrate the above principles in designing fields for logical records, let us design a logical record to be used for student billing. The information attributes available for each student are shown in Figure 8 along with sample data values and representation. Please note that we are not interested in how or where the information is currently being stored, we are only interested in designing a record for the purpose of student billing. The information attributes that are required for student billing and the resulting logical records are shown in Figure 9.

The assumptions on which this solution is based are:

- Student Name, Billing Address, and Department are already in satisfactory form.

Information Attribute	Field Size	Data Representation	Encode and Compress	Sample Value
Student Name	30 Characters	Character String	None	JONES, JOHN E.
Billing Address	30 Characters	Character String	None	801 Main St. Pok.
Campus Address	2 Characters	Character String	BARKER HALL → BA JONES HALL → JO BROWN HALL → BR DAVIS HALL → DA OFF CAMPUS → OF	BA
Class	2 Characters	Character String	Remove first two digits	72
Tuition	5 Decimal Digits	Fixed Decimal 2 Decimal Places	None	50000
Room Rent	5 Decimal Digits	Fixed Decimal 2 Decimal Places	None	20000
Dept.	20 Characters	Character String	None	ELECT ENGR

Figure 9. Student Billing Record

- Tuition and Room Rent will be used frequently in arithmetic calculations.
- For the purpose of billing, an Encoded Campus Address and compressed class is adequate.

FIXED- AND VARIABLE-LENGTH LOGICAL RECORDS

In the student billing example, the resulting logical record was fixed length. The fixed-length record is defined as a record in which all fields are of unchanging length and the position of the field within the record is fixed for each record within the file. Although fixed-length records are usually the easiest to process and control, all types of data do not conform to this format.

A record becomes a variable-length record if

1. It contains a variable-length field.
2. It contains a variable number of occurrences of fixed-length fields.

If in our student file (Figure 7) we find that the average length of a student's name is 10 characters, we may want to make STUDENT NAME a variable-length field, instead of a 30-character fixed field (see Figure 10). Grade data in the student file is fixed for each entry (COURSE-GRADE), however, there are a variable number of courses for each student (Figure 10).

Since a variable-length logical record can contain many different variable-length and occurrence fields, techniques have been developed to organize data within these records

to allow retrieval of each field. These techniques will be discussed in a later section of this chapter.

LOGICAL RECORD IDENTIFICATION

Each logical record contains a field that identifies the record and relates it to an entity. This field is called the record key. If the record contains information about an employee, the key is the Man Number; about a part, the Part Number. A file is usually (but not necessarily) in sequence by key. A key can also be a concatenation* of two or more fields within the record, or a truncation of a field. Examples of concatenated and truncated keys are shown in Figure 11. Note that the keys do not have to be unique.

LOGICAL DATA STRUCTURES

Within logical records, relationships can exist between fields as well as between entity and field. These relationships are called data structures. Most information about entities is expressed in one of three data structures; simple, hierarchic, and network. Please remember that we are talking about relationships within a single logical record, not between two or more logical records.

Simple Data Structures

Simple data structures exist when each field within a logical record is dependent only on the key field for its meaning. An example of a simple structure is shown in Figure 12, Part A.

*Concatenation is defined as 'linking together to form a series or chain'.

Variable-Length Fields

Variable Occurrences of Fixed Fields

Student Name

Fixed	Length	Variable	Length
JohnbJonesbbb ——— b	25	JohnbJones	10
lbWUbbb ——— b b	25	lbWU	4
Stanley R. Kolwalski b — b	25	Stanley R. Kolwalski	19

Student	Courses and Grades	Length
Jones	Math 101 B	8
	Psyc 200 C	8
	Musc 500 A	8
	Phil 220 A	8
	Math 102 C	8
	Total	40
Davis	Engl 001 D	8
	Math 010 D	8
	Total	16

Figure 10. Variable-Length Data

Key Type	Entity	Record	Key																								
Simple	Employee	<table> <tr> <td><u>Manno</u></td> <td><u>Name</u></td> <td><u>Gross Pay</u></td> </tr> <tr> <td>778180</td> <td>Jones</td> <td>100</td> </tr> </table> <p>Key</p>	<u>Manno</u>	<u>Name</u>	<u>Gross Pay</u>	778180	Jones	100	778180																		
	<u>Manno</u>	<u>Name</u>	<u>Gross Pay</u>																								
778180	Jones	100																									
Part	<table> <tr> <td><u>Part No.</u></td> <td><u>Part</u></td> <td><u>Description</u></td> </tr> <tr> <td>A622915</td> <td>Wing</td> <td>Brace</td> </tr> </table> <p>Key</p>	<u>Part No.</u>	<u>Part</u>	<u>Description</u>	A622915	Wing	Brace	A622915																			
<u>Part No.</u>	<u>Part</u>	<u>Description</u>																									
A622915	Wing	Brace																									
Concatenated	Component Parts	<table> <tr> <th colspan="3">Key</th> </tr> <tr> <td><u>Parent Part</u></td> <td><u>Component Part</u></td> <td><u>Qty Req</u></td> </tr> <tr> <td>A01</td> <td>XYZ</td> <td>1</td> </tr> <tr> <td>A01</td> <td>XY1</td> <td>2</td> </tr> <tr> <td>C03</td> <td>ZYZ</td> <td>7</td> </tr> <tr> <td>C04</td> <td>ZY5</td> <td>6</td> </tr> <tr> <td>C04</td> <td>125</td> <td>9</td> </tr> <tr> <td>C04</td> <td>173</td> <td>1</td> </tr> </table>	Key			<u>Parent Part</u>	<u>Component Part</u>	<u>Qty Req</u>	A01	XYZ	1	A01	XY1	2	C03	ZYZ	7	C04	ZY5	6	C04	125	9	C04	173	1	<p>A01XYZ</p> <p>A01XY1</p> <p>C03ZYZ</p> <p>C04ZY5</p> <p>C04125</p> <p>C04173</p>
Key																											
<u>Parent Part</u>	<u>Component Part</u>	<u>Qty Req</u>																									
A01	XYZ	1																									
A01	XY1	2																									
C03	ZYZ	7																									
C04	ZY5	6																									
C04	125	9																									
C04	173	1																									
Truncated	Telephone Customer	<table> <tr> <td><u>Name</u></td> <td><u>Phone No.</u></td> <td><u>Address</u></td> </tr> <tr> <td>Robert A. J.</td> <td> </td> <td> </td> </tr> <tr> <td>Roberts C. L.</td> <td> </td> <td> </td> </tr> <tr> <td>Roberts A. N.</td> <td> </td> <td> </td> </tr> <tr> <td>Robertson R. D.</td> <td> </td> <td> </td> </tr> </table> <p>Key</p>	<u>Name</u>	<u>Phone No.</u>	<u>Address</u>	Robert A. J.			Roberts C. L.			Roberts A. N.			Robertson R. D.			<p>Robert</p> <p>Robert</p> <p>Robert</p> <p>Robert</p>									
<u>Name</u>	<u>Phone No.</u>	<u>Address</u>																									
Robert A. J.																											
Roberts C. L.																											
Roberts A. N.																											
Robertson R. D.																											

Figure 11. Record Keys

Structure Type	Sample Record	Graph																												
(A) Simple	<table border="1"> <thead> <tr> <th>Key</th> <th>Name</th> <th>Home Address</th> <th>Enroll Date</th> <th>Current Courses</th> </tr> </thead> <tbody> <tr> <td>Student No. 7302A</td> <td>Jones, John</td> <td>101 Main St.</td> <td>Sept. 1969</td> <td>Math 501 Engl 001 Psych 700 Phys 317</td> </tr> </tbody> </table>	Key	Name	Home Address	Enroll Date	Current Courses	Student No. 7302A	Jones, John	101 Main St.	Sept. 1969	Math 501 Engl 001 Psych 700 Phys 317																			
Key	Name	Home Address	Enroll Date	Current Courses																										
Student No. 7302A	Jones, John	101 Main St.	Sept. 1969	Math 501 Engl 001 Psych 700 Phys 317																										
(B) Hierarchy	<table border="1"> <thead> <tr> <th>Key</th> <th>Description</th> <th>Warehouse</th> <th>Component Parts</th> </tr> </thead> <tbody> <tr> <td>Part No. AZ110099</td> <td>Base</td> <td>702</td> <td>AX1173321</td> </tr> <tr> <td></td> <td></td> <td>Qty in WH 702 3</td> <td>Qty of Req 1</td> </tr> <tr> <td></td> <td></td> <td>Location in 702 A28</td> <td>AB113227</td> </tr> <tr> <td></td> <td></td> <td>716</td> <td>4</td> </tr> <tr> <td></td> <td></td> <td>100</td> <td>B7336241</td> </tr> <tr> <td></td> <td></td> <td>A27,A01</td> <td>2</td> </tr> </tbody> </table>	Key	Description	Warehouse	Component Parts	Part No. AZ110099	Base	702	AX1173321			Qty in WH 702 3	Qty of Req 1			Location in 702 A28	AB113227			716	4			100	B7336241			A27,A01	2	
Key	Description	Warehouse	Component Parts																											
Part No. AZ110099	Base	702	AX1173321																											
		Qty in WH 702 3	Qty of Req 1																											
		Location in 702 A28	AB113227																											
		716	4																											
		100	B7336241																											
		A27,A01	2																											
(C) Network	<table border="1"> <thead> <tr> <th>Key</th> <th>Job Title</th> <th>Employee</th> <th>Task</th> </tr> </thead> <tbody> <tr> <td>Dept. No. D15</td> <td>① Engineer (↑⑦)</td> <td>⑤ Jones (↑④,↑⑨)</td> <td>⑧ Design CompA</td> </tr> <tr> <td></td> <td>② Secretary (↑⑥)</td> <td>⑥ Smith (↑②,↑⑩)</td> <td>⑨ Build ProtoA</td> </tr> <tr> <td></td> <td>③ E. Tech</td> <td>⑦ Johnson (↑①,↑⑧)</td> <td>⑩ Document</td> </tr> <tr> <td></td> <td>④ M. Tech (↑⑧)</td> <td></td> <td>⑪ Build PartZ</td> </tr> <tr> <td></td> <td></td> <td></td> <td>⑫ Design CompZ</td> </tr> </tbody> </table> <p>(↑ ⊗) Means 'a relationship exists between this field and field ⊗'</p>	Key	Job Title	Employee	Task	Dept. No. D15	① Engineer (↑⑦)	⑤ Jones (↑④,↑⑨)	⑧ Design CompA		② Secretary (↑⑥)	⑥ Smith (↑②,↑⑩)	⑨ Build ProtoA		③ E. Tech	⑦ Johnson (↑①,↑⑧)	⑩ Document		④ M. Tech (↑⑧)		⑪ Build PartZ				⑫ Design CompZ					
Key	Job Title	Employee	Task																											
Dept. No. D15	① Engineer (↑⑦)	⑤ Jones (↑④,↑⑨)	⑧ Design CompA																											
	② Secretary (↑⑥)	⑥ Smith (↑②,↑⑩)	⑨ Build ProtoA																											
	③ E. Tech	⑦ Johnson (↑①,↑⑧)	⑩ Document																											
	④ M. Tech (↑⑧)		⑪ Build PartZ																											
			⑫ Design CompZ																											

Figure 12. Logical Data Structures

All of the fields in the student record are meaningful if we associate each of them with the key of STUDENT NO. This relationship can be shown graphically as in Figure 12. In the graph each field depends on the field above it for its meaning.

Hierarchic Data Structures

Hierarchic relationships exist between fields when a field depends on one or more fields (other than the key) for its meaning.

In the example in Figure 12, Part B, "quantity on hand" is meaningful only if the warehouse in which it is stored is known, and "quantity of a component part" is meaningful only if the part to which it refers is known. The graph in Figure 12, Part B shows this relationship. Graphs of hierarchic structures are called 'trees'. Another characteristic of hierarchies is that each branch of the tree is independent of other branches. In the tree shown in Figure 13 assume that only one field exists at each level. A branch of the tree can then be identified by the concatenation of the names

of all the fields in that branch. All the resulting branches shown in Figure 13 are independent if the data structure is hierarchic.

Fields at the same level of a hierarchy are sometimes formed into groups called segments. It is often convenient to have a key within each segment so that multiple occurrences of that segment can be identified in sequence. Each segment can then be treated as a sub-record and contain fields that are dependent on the key of the segment. In Figure 14 the field hierarchy has been transformed into a segment hierarchy. In this graph, each field in each segment is dependent on the key of that segment and the keys of all higher-level segments for its meaning. None of the meaning has been lost and the structure has been simplified.

Networks

A network exists within a logical record when the branches of a tree representing the structure are not independent of each other. An example of a network is shown in Figure 12.

Tree Representation of Hierarchy

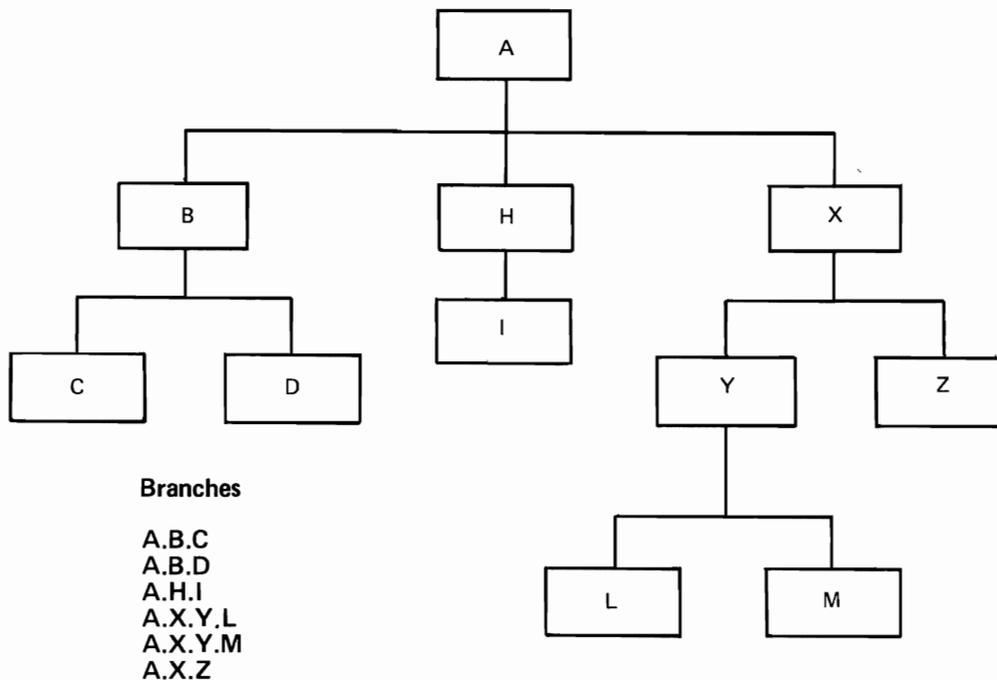


Figure 13. Trees

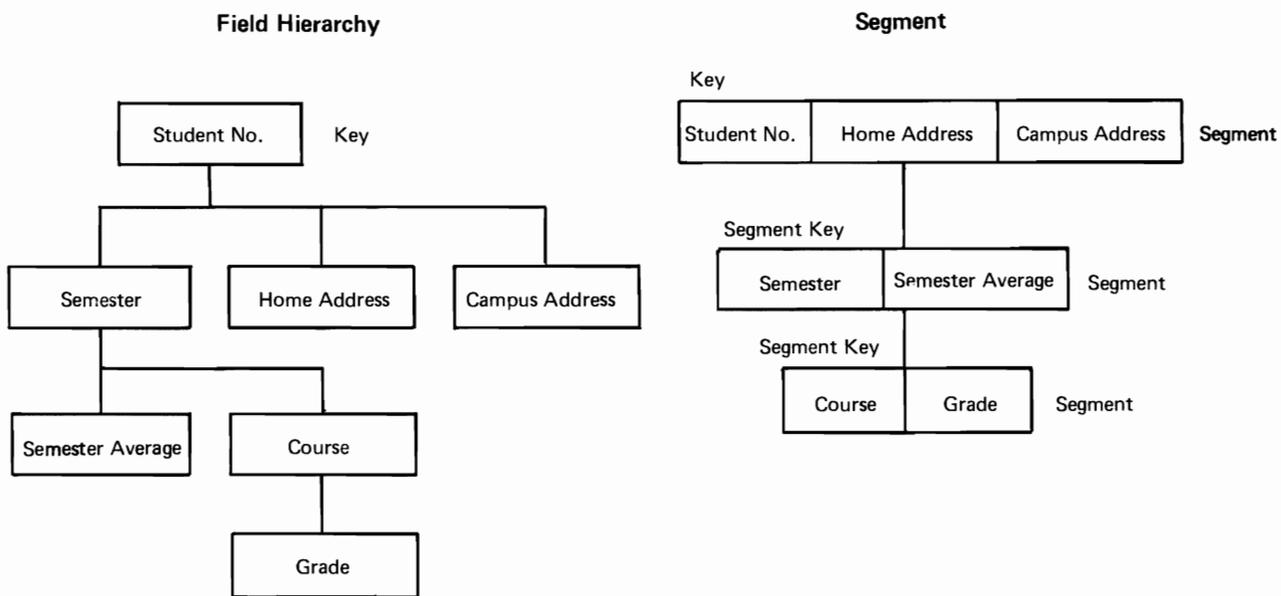


Figure 14. Segment Hierarchy

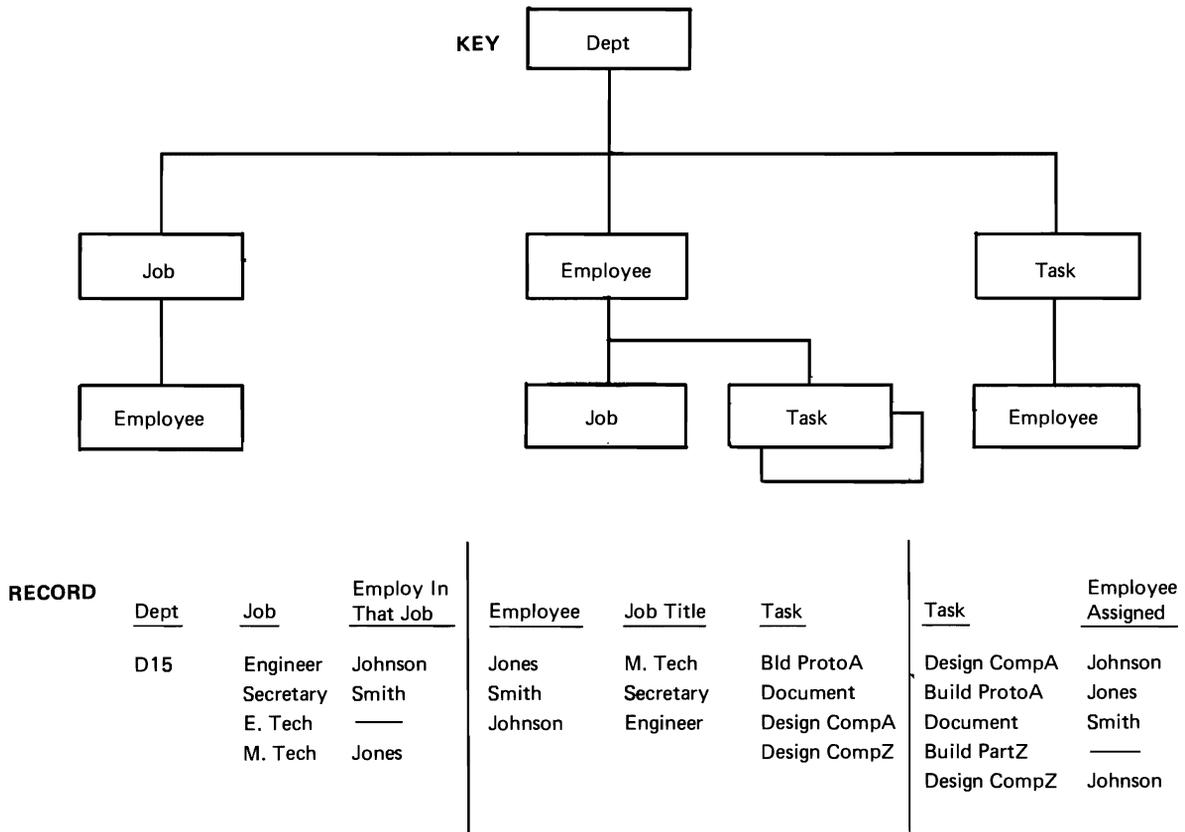


Figure 15. Network as Hierarchy by Duplication

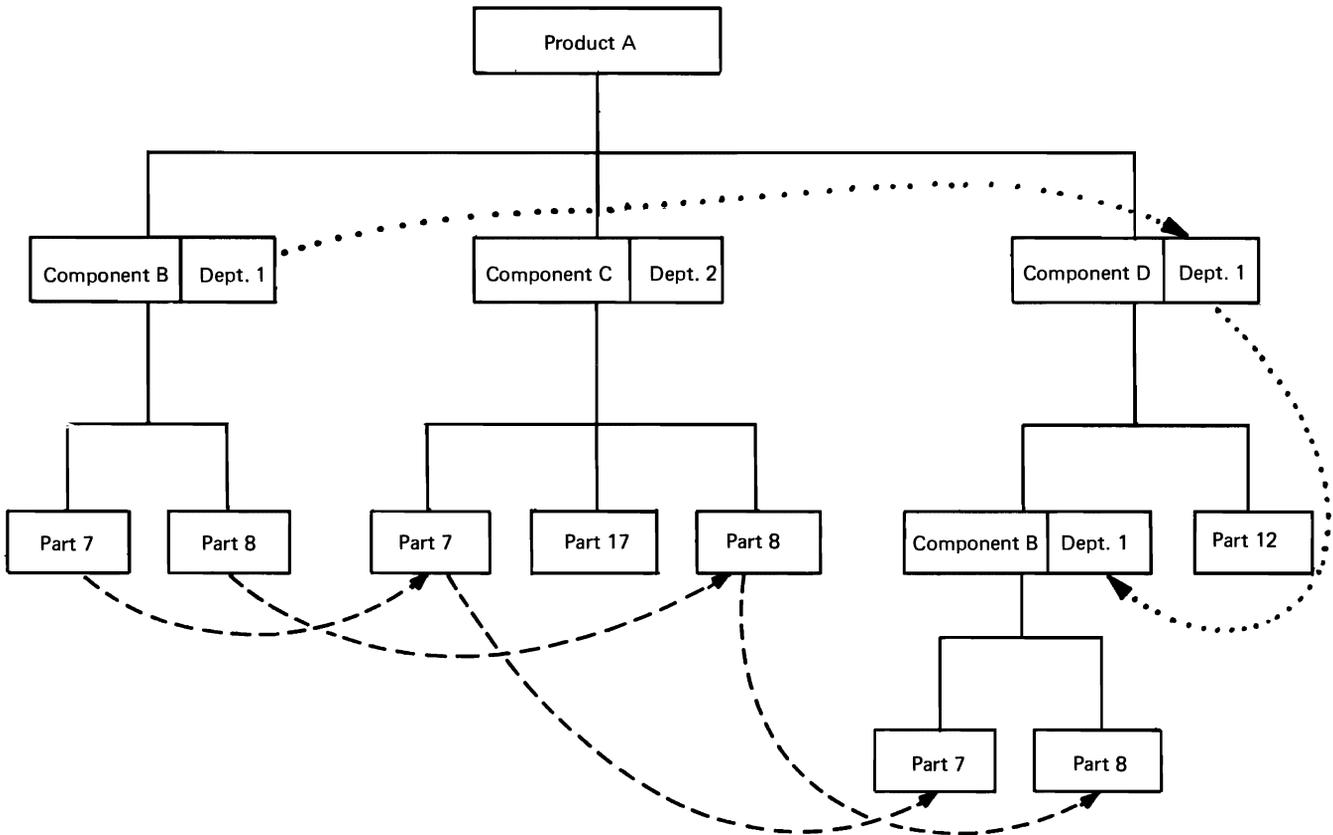


Figure 16. Network

The entity of interest is the department. Within the department are jobs that are represented by job titles (Engineer, Technician, Secretary), people represented by Man Number, and tasks to be or being performed (Design Component A, Build Component B, Document System). An employee both fills a job title and is assigned one or more tasks.

It is interesting to note that by duplicating fields within the department record, the network can be expressed as a hierarchy (see Figure 15).

Another example of a network within a logical record is shown in Figure 16 which represents the structure of a product made up of many parts. If we wish to relate all parts of the same type (shown by the dashed lines) and all components that are made in the same department (shown by dotted lines) a network exists.

ORGANIZING FIELDS WITHIN RECORDS

The data structures described in the previous section represent the logical relationships of the fields within a logical record. Many techniques have been developed to represent these data structures physically within the logical

record as it appears in main storage. These techniques allow the user's problem program to locate and process each field within the record. This section will discuss the most common methods for organizing fields within records.

Locating Fields

There are 3 basic ways in which a field can be located and identified within a logical record:

1. By relative location
2. By embedded identification
3. By pointers and lists

Relative Location

Each field in the logical record is identified by its relative location. For example, in Figure 17, the Name field in the Payroll record is always the second field in the record, and the Address field in the Directory record is always the last field in the record. Relative location is the most common technique used today, and is always used for fixed-length records. It can also be used for variable-length records where no fields are omitted.

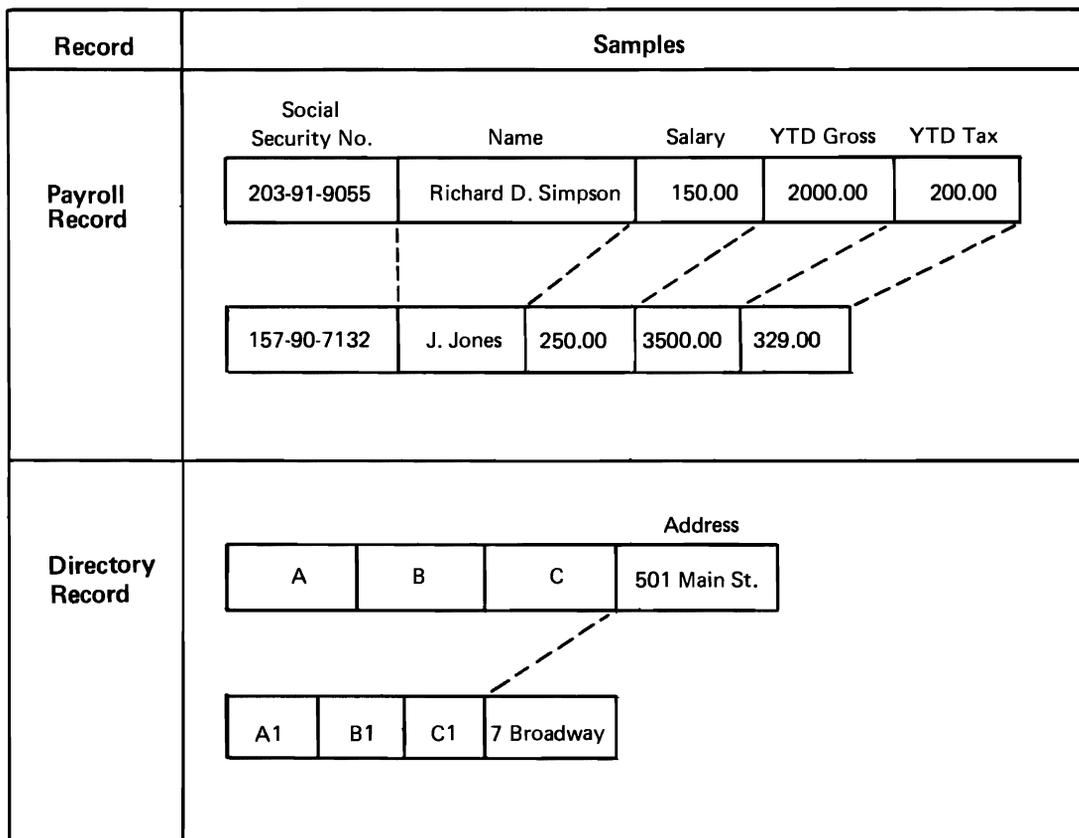


Figure 17. Location by Relative Position

Embedded Identification

This technique requires that additional fields be added to the record. These fields contain no data about an entity, but identify the field or fields that follow. In Figure 18, fields X, Y and Z contain character strings that give the names of the fields which they immediately precede. With this technique, fields can be in any order within the record and still be located.

Pointers and Lists

Pointers are fields within the record that, like embedded identification, contain no data about the entity. A pointer field contains the location (address) of a field within the record. In Figure 19 the Personnel record contains information on both the employee's skills and his previous positions. The first skill and position are located by relative location. Each skill and position field has associated with it a pointer field that points to the next skill or position. All the skills can be located by finding the first skill then following the pointers until the end is reached (* in pointer field). Fields connected in this manner with pointers are called lists, and the techniques of retrieving and updating lists are called list processing.

Delimiting Fields

The previous three techniques assume that the length of each field is known to the problem programmer. In fixed-length records, or in variable-length records containing a variable number of fixed-length fields, the length does not vary from record to record, and can be specified in the problem programs. If variable-length fields are present, however, the length of the field must be obtained from the record, either implicitly or explicitly.

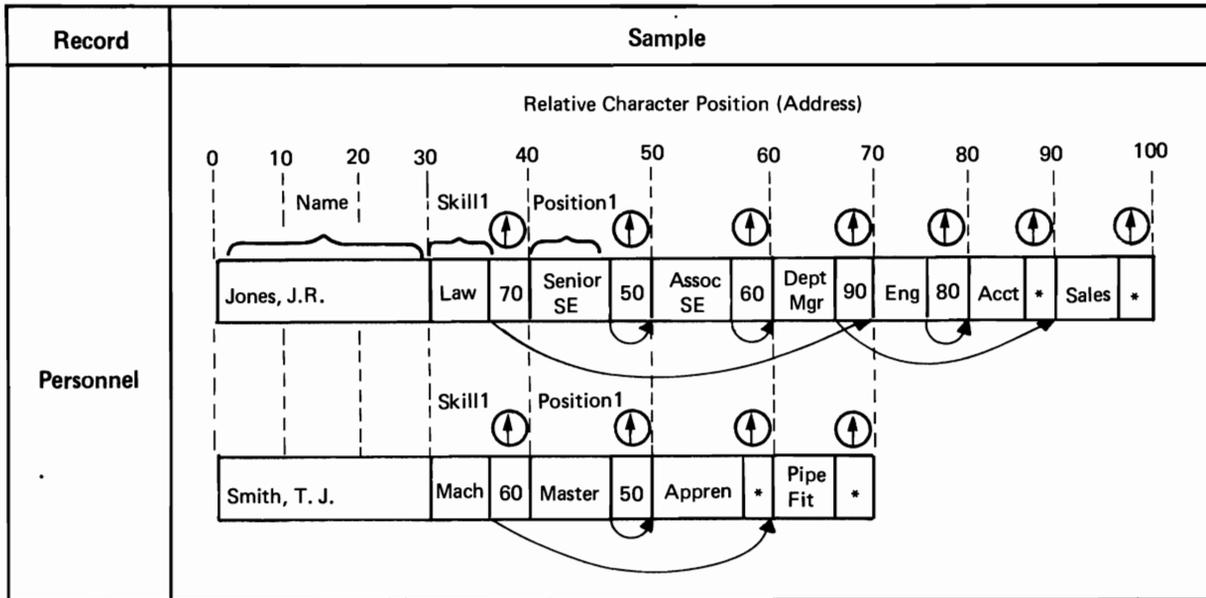
Implicit Length

If pointers are used to locate the beginning of fields, the lengths can be calculated by subtracting the locations of the starting points of adjacent fields. Another method is inserting special characters (delimiters) to mark the beginning or end of a field. Examples of implicit lengths are shown in Figure 20.

Record	Samples							
Payroll	Social Sec. No.	X		Y		Z		
	155-12-8099	Name	Jones, R. J.	Salary	120.00	Address	501 Main	
	173-18-7621	Address	8 Broadway	Name	Smith, T. P	Salary	250.00	
Student	Student Name	X		Y				
		Classes	Math	Engl	Lit	Grades	A	C

Fields X, Y, Z contain character strings that give the name of the fields immediately following.

Figure 18. Location by Embedded Identification



Notes



– Means this is a pointer field.

* – Indicates end of list.

Skills for Jones are Law – Eng – Acct

Positions for Jones are Senior SE – Assoc SE – Dept Mgr – Sales

Skill 1 and Position 1 are located by relative position.

Figure 19. Pointers and Lists

Technique	Sample Record	Length Calculation
Pointers (1)	<p>These pointers point to the start of each field.</p> <p>Student P1 P2 P3 Address Tuition Guardian</p> <p>Roberts, R. K. 20 45 50 501 Main 700.00 Roberts, J. N.</p> <p>Record Address → 20 45 50 End</p>	<p>Length of Address = P2 – P1</p> <p>Length of Tuition = P3 – P2</p> <p>Length of Guardian = End – P3+1</p>
Delimiters (1) (2)	<p>Student Address Tuition Guardian</p> <p>Roberts, R. K. \$ 501 Main \$ 700.00 \$ Robert, J. N.</p> <p>RA1 RA2 RA3 End</p> <p>RA = Record Address of Delimiter \$</p>	<p>Length of Address = RA2 – RA1 – 1</p> <p>Length of Tuition = RA3 – RA2 – 1</p> <p>Length of Guardian = End – RA3</p>

(1) Field Identification is by relative location in both examples.

(2) '\$' Delimiter must not appear as data in this record.

Figure 20. Delimiting Fields – Implicit Length

Explicit Length

The most common technique for specifying the length of variable-length fields is preceding the field with a numeric value that gives the length of the field. In Figure 21 field A contains a value of 25 which is the length of the Name field.

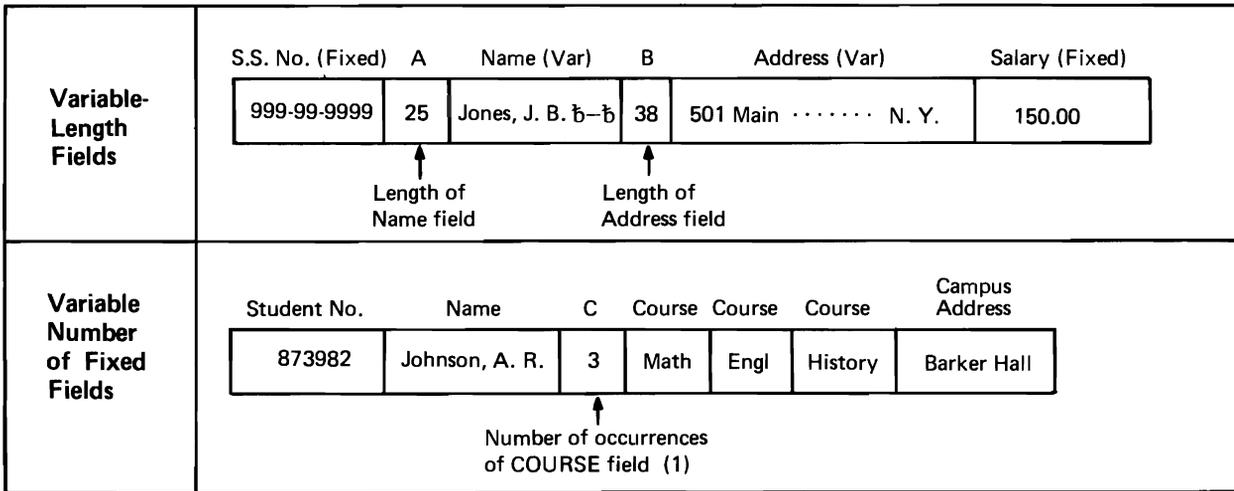
Representing Data Structures

The methods for locating and delimiting fields can be used in combination to physically represent logical data structures. Some examples are:

EXAMPLE 1: Representing a hierarchy using relative position and explicit length. The hierarchy shown in Figure 12(A) can be represented as shown in Figure 22.

The number of repeating fields are given in the fields CNT (X). If fields are absent, the CNT (X) field value is zero, but must still be present if location by relative position is to be retained. This method is frequently used in COBOL programming.

EXAMPLE 2: Representing a network with lists. The network shown in Figure 12 (C) can be represented with lists as shown in Figure 22. Note that because EMPLOYEE NAME is a variable-length field, a length field is included immediately preceding each employee field. Note also that a single field can be on two lists. The first item in each list is located by relative position.



(1) Even though the length of each COURSE field is known, the number of occurrences is required to locate the CAMPUS ADDRESS field.

Figure 21. Delimiting Fields – Explicit Length

Chapter 3: Data Storage Devices

INTRODUCTION

In Chapters 1 and 2 we have looked at data, information, records, and files from the viewpoint of the user of an information system. In this chapter we will look at how the data is stored on and retrieved from the data storage devices attached to a computer system, and the characteristics of these devices.

The ideal complement to a data processing system would be unlimited storage, any part of which could be made available at the instant it was needed by a program. This ideal is only approached by main storage, which certainly is not infinite in size and does require a certain amount of time to access. If technology could provide an infinitely fast, unlimited size storage, there would be no need for data management as we know it today because all data could be directly referenced by the user.

Providing data storage outside the main storage of a computing system is one function provided by Input/Output (I/O) devices. All data (out) to and (in) from devices passes through main storage. Input/Output devices provide external storage and a means of communication between a system and the external world.

I/O devices include card read punches, printers, typewriter-keyboard devices, magnetic tape units, direct access storage (disk and drum), teleprocessing, and process control equipment. This chapter will provide an aid to understanding the characteristics of those devices used in external data storage.

ATTRIBUTES

There are several attributes that classify storage devices. These include data capacity, addressability (how data is located), access time to data, data transfer rate, physical advantages and limitations, and cost or economy. Devices are selected for use in systems by evaluating these attributes against the requirements of the application or system.

Capacity of a storage device is a measure of the amount of data that can be stored on it. A standard measure is bit capacity, usually megabit (10^6 bits). All of the bit capacity is not available for data storage. Some space is required for error checking and synchronizing information, so it is more practical to refer to useful data storage capacity in bytes (a meaningful frame of 8 bits). A data or storage word may also be used to describe a standard format of a number of bits or bytes.

Addressability can be considered in two parts: addressing —

how a data element is located; and resolution — how much data (bits, bytes, or words) is referenced by one data address. Addressing may be simply implied. For example, cards being read from a card reader are read in the order in which they are placed in the hopper. It is not possible to select a particular card for reading in the device. This is known as sequential access, when addressing or locating a given element of data is completely dependent on the data that has already been accessed on the device. The opposite extreme in addressing is the ability to reference data explicitly by location where this access is in no way dependent on the location of the data most recently read or written. This is known as direct access or direct addressing. Another addressing method is access to data based on the value of data or the contents of a storage location. This is called associative addressing. Resolution describes how much data is referenced in an address. For I/O devices, this is usually a physical record or block such as a card in a card reader. Main storage, while not an I/O device, is an example of byte resolution — any given byte may be addressed. Addressability is influenced by physical attributes of the device and the storage media itself. The device can fill both the role of external storage and communication if the storage media is removable and interchangeable. For example, a magnetic tape can serve as additional storage to a system, can be removed from the tape drive and mounted on a drive in another system, and thereby become the source of input to this second system.

Addressability can be limited by the physical structure of the device. A tape unit is primarily suited for sequential access to data. While it is possible to search for a particular element of data, the system must look at every intervening record until the desired one is found.

If you were to take magnetic tape (generally observed on a reel), cut it into short pieces, and add the ability to find any particular piece or strip, you would have the fundamental principle of the data cell. An addressing scheme permits selection of a given strip of tape. Once selected, this strip must still pass by a reading station. Continuing the comparison with a magnetic tape unit, if each strip contains 10 records and we know which strip contains the target record, it is only necessary to search up to 9 records before gaining access to the desired one. Conceptually this could be accomplished by drawing the strip straight across some reading mechanism starting with the beginning of the strip. Another principle can be introduced here — rotating media. Forming this strip of tape into a loop, it is no longer necessary to start reading at the beginning of the tape strip. Reading may start anywhere and, when the end of the strip is reached, simply continue from the beginning of the strip.

The principle of rotating media is the basis for disk and drum devices. If you took many data cell strips, formed them into loops side by side with the recording surfaces facing out, put a reading element on each loop, and made the whole assembly rigid (except the read elements or heads), you would have the basic drum. The same principle holds true for disks except that for economy, there is usually only one read/write head per disk surface. This head is on a slide or arm and is moved to the desired concentric location or track as required. Following the example that we used for the data cell, if we know what track a record is on it is necessary to position a read head on that track, then read up to 10 records to find the one in question. Note that this may be accomplished without moving the unneeded data into main storage.

The media or volume may be removable from the device and interchangeable with other volumes on other devices or it may be permanently mounted. The latter is usually characteristic of drums. Another measure that can be made of the device parameters presented thus far is the factor of time or speed. The time factor includes:

- Access time to the beginning of data.
- Transfer rate of the data to or from main storage.

For magnetic tape, access time may be the time necessary to move the tape to the beginning of the next record (milliseconds) or much longer if it is necessary to search past many records to find the desired data.

Access time for direct access devices consists of “seek time”, which is the time required to position the head to the particular track, plus the “latency” or time necessary to reach the beginning of the desired data on that track. Average latency may be defined as one-half the time interval between two successive occurrences of a particular point or record at a read/write access position (read heads). Once having reached the beginning of the desired data record, transfer of data occurs to or from main storage at some transfer rate.

In System/360, bytes per second is a convenient measure of data transfer rate. This rate of transfer is determined by the device. If tape is moving past a read head at some number of inches per second and each inch of tape contains 800 bytes, then the transfer rate is apparent. This rate may be variable if the device has the ability to “slow down” during possible peak system loads when data cannot be moved into main storage as fast as the device is presenting data. Punched page tape is an example of this. A unique mechanical cycle is taken for each byte of data and the next mechanical cycle is not taken until the preceding data is taken from the device. Magnetic tape and disk units, however, must move data at a uniform rate. If data is not removed from the device, or provided to it fast enough, we say the

device has “overrun” the system.

ATTACHING DEVICES TO SYSTEMS

The preceding discussion of device characteristics indicates that the many different devices cannot be directly attached to a system in the same manner. The channel, oriented to the central processing system and main storage; and the control unit, oriented to a particular device type, are functional units between the device and the system.

The control unit provides the logical capability necessary to operate and control an I/O device, and adapts the characteristics of each I/O device to the standard form of control provided by the channel. The control unit may be housed separately or it may be physically and logically integrated with the I/O device.

The channel directs the flow of information between I/O control units and main storage on a standard I/O interface to the control units. Again, the channel may be housed separately or integrated under the covers of a CPU.

Much of the circuitry necessary to control a device is complex, and is only required for a relatively small portion of the total device operation. Also the normal use of the device itself may create relatively long idle periods with busy intervals. These conditions and the need for economy make it feasible for several devices to share the services of one control unit. Depending on the particular characteristics of the devices, the control unit may permit many devices or only one device to be operated at a time.

The channel is used for relatively brief intervals by a device (control unit) to fetch or store data or control information in main storage. Thus a channel will often have more than one control unit attached to it. Again, concurrent operation of control units may be permitted depending on the design of a particular channel type.

CONTROL OF DEVICES

Now that we have developed device characteristics and shown how devices may be attached to data systems, we may turn to how operations start, end, and are controlled on devices by the system. The method of control used in System/360 is a hierarchy.

Instructions are issued by the program (usually the operating system) to channels. These instructions specify the particular channel, control unit, and device desired and make indirect reference to control words in a predetermined location of main storage. The channel will use these control words to issue commands to the control unit. These commands (such as read, write, control, and test) may be modified and further defined for each device type.

The device executes orders received from the control unit. A DASD seek, for example, requires that a control command be sent to the control unit. This command is modified to indicate that a seek operation is required. The control unit then requests the necessary data bytes from main storage via the channel. Having received this seek address, the control unit looks where the disk device is currently positioned, calculates the direction and amount of arm travel, then gives the device the seek order.

This chapter has taken a very general look at devices, their use, attachment, and operation. Refer to the IBM System/360 Principles of Operation (GA22-6821) for more information on system control of I/O operations and to the Component Description SRL publications for more detailed information about specific devices.

INTRODUCTION

We have seen that devices store data in physical blocks. To make this data useful it is necessary to be able to identify collections of physical blocks that pertain to specific entities, and to be able to retrieve this data in a meaningful sequence. It is also necessary to store the data on the device in such a manner that the device is efficiently utilized. The technique of identifying, retrieving, and storing data in physical blocks on devices is called Data Organization.

We know that data resides on a volume (such as a disk pack or a reel of tape) in the form of a physical block. The size of the block is often determined by device limitations (80-character card) or hardware efficiency considerations (one large block of data is usually more efficient to process than many small blocks). Therefore, data is not normally grouped in terms of physical blocks but in terms of another basic unit of data called a stored record. A stored record is an identifiable collection of related data elements. The relationship of the data elements depends upon the particular data structure. A data element is the data portion of a field and has data attributes such as name, representation, and length as opposed to information attributes pertaining to entities such as employee name, address, or salary. There may be one or more stored records in a block or there may only be part of a stored record in a block with the remainder contained in one or more additional blocks. The correspondence of stored records to logical records will be discussed in the next chapter.

Stored records are grouped on storage volumes as data sets. A data set is a named collection of stored records that may reside on one or more volumes by itself or with other data sets. An example of a data set might be a collection of stored records containing data pertaining to all students in a university. Another data set might be maintained for faculty members, and another for programs used to process student data.

We will now describe and compare three basic forms of data organization: sequential, direct, and list. We will then look at how these data organizations apply to various data structures.

SEQUENTIAL ORGANIZATION

In the past, the most common data organization has been sequential organization. Stored records are physically placed adjacent to one another and to retrieve any record, all preceding records must be retrieved.

The order or sequence is usually chosen according to a common attribute of the stored records such as a particular data element. The data element chosen to order the data set is called the key. The sequence of a data set may be changed by selecting a different data element to be the key and sorting the stored records according to the values of the new key. In Figure 23(A), a data set containing data about the employees of a given company is ordered according to the values associated with the data element EMPLOYEE NUMBER. If the data element NAME were used as the key, the stored records would be physically reordered in the data set as shown in Figure 23(B). In some cases, using one data element as a key is not sufficient to identify a given stored record. In this case, one or more additional data elements would be concatenated to form the key. Figure 23(C) shows the same data set sorted in descending order according to the values of the data element DEPT. NUMBER. Notice that there are two stored records with the value of DEPT. NUMBER equal to 12. To ensure a unique sequence, the data element NAME is concatenated to DEPT. NUMBER and, in this case, the values of NAME are placed in descending order. At the other extreme, stored records might not make use of any key and the order of placement into the data set may be based on their order of arrival into the system.

The advantage of sequential organization is rapid access to the next stored record. Referring to Figure 23(A) again, sequential organization is more suited to processing requests of the form "retrieve all stored records in order by ascending value of EMPLOYEE NUMBER" rather than of the form "retrieve the stored record with EMPLOYEE NUMBER equal to 125". In both requests all of the stored records must be accessed, but the first five stored records accessed in the second request are of no value. Because a data set can have only one physical sequence of stored records, the mode of processing must coincide with the given sequence.

If a group of stored records must be processed using more than one key, the stored records are sorted into different data sets. The stored records in Figure 23 represent three separate data sets. The contents of each data set is the same, but the ordering facilitates processing using one of three different keys. This duplication of data sets obviously wastes storage space and increases maintenance cost because a stored record that is updated in one data set must be updated in all three.

Updating stored records in a sequential organization can be difficult, especially if the updated stored record is longer or shorter than the original. Updating usually involves copying the stored records from one data set to another and

(A)

	Employee Number	Name	Dept. Number
1	008	Smith, J.	14
2	043	Jones, B.	12
3	081	Ritter, K.	20
4	086	Smith, B.	12
5	102	Pearce, D.	17
6	125	Holt, J.	21

┌──────────┐
|
Key

(B)

	Employee Number	Name	Dept. Number
1	125	Holt, J.	21
2	043	Jones, B.	12
3	102	Pearce, D.	17
4	081	Ritter, K.	20
5	086	Smith, B.	12
6	008	Smith, J.	14

┌──────────┐
|
Key

(C)

	Employee Number	Name	Dept. Number
1	125	Holt, J.	21
2	081	Ritter, K.	20
3	102	Pearce, D.	17
4	008	Smith, J.	14
5	086	Smith, B.	12
6	043	Jones, B.	12

┌──────────┐
|
Concatenated Key

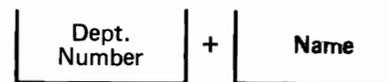


Figure 23. Sequential Organization

updating the affected stored records as they are copied. This becomes expensive when only a small number of stored records require updating in a large data set. The usual practice is to batch stored records to be updated, sort them into the sequence of the data set, and process the data set periodically when a significant number of updates have been accumulated. However, this practice reduces the timeliness of the data in the data sets.

Similar problems are encountered in sequential organizations where stored records are to be added or deleted. Inserting or deleting a stored record means the existing stored records must be shifted apart or pulled together respectively, which means recopying the entire data set. This is not

entirely a disadvantage because retaining a stipulated number of old copies* as a precaution provides a ready method of recreation of data sets in the event of inadvertent destruction or discovery of errors.

Sequential organization offers rapid access to the next stored record in a data set if the basis for retrieval is the same as the basis for the physical ordering of the data set. Sequential organization is inefficient for retrieving stored records not in sequence. Adding, deleting, and updating stored records in a data set presents difficulties in that the data set normally has to be copied in its entirety, but by the same token, back-up procedures and error recovery are simplified.

* These old copies are referred to as 'generations'.

DIRECT ORGANIZATION

Direct data organization ignores the physical sequence of stored records in a data set and accesses stored records on the basis of their physical hardware address in the storage device. Direct organization is applicable only to direct access storage devices such as disks and drums. Stored records are always stored or retrieved using the hardware address of the particular stored record. Variations in the implementation of the direct organization occur in the way that the key of the stored record and the hardware address are related. Three common methods for accessing stored records are: direct address, index search, and calculation or hashing.

In direct address processing, every possible key in the data set corresponds to a unique storage address.* A location must be reserved in the data set for every key in the range. For example, consider a data set that contains stored records with key values ranging from 100 to 499. Because each

key converts to a unique address, 400 locations must be reserved. If only 200 keys are active, half of the space in the data set will be unused. This method of direct addressing allows any stored record to be stored or retrieved with only one movement of the access mechanism. In addition, sequential processing is possible under this organization because the stored records are written in key sequence.

The index search variation of direct organization alleviates some of the shortcomings of direct addressing. In this method, an index is maintained consisting of keys and assigned hardware addresses. The assigned addresses are all the available storage locations within a data set. When a stored record is to be placed into the data set, as in Figure 24(A), the index is searched to locate an available storage address, the key of the stored record is equated to that address, and the stored record is written into the data set. When a stored record is to be retrieved, as in Figure 24(B), the index is searched to locate the required key value, the associated storage address is obtained, and the address is

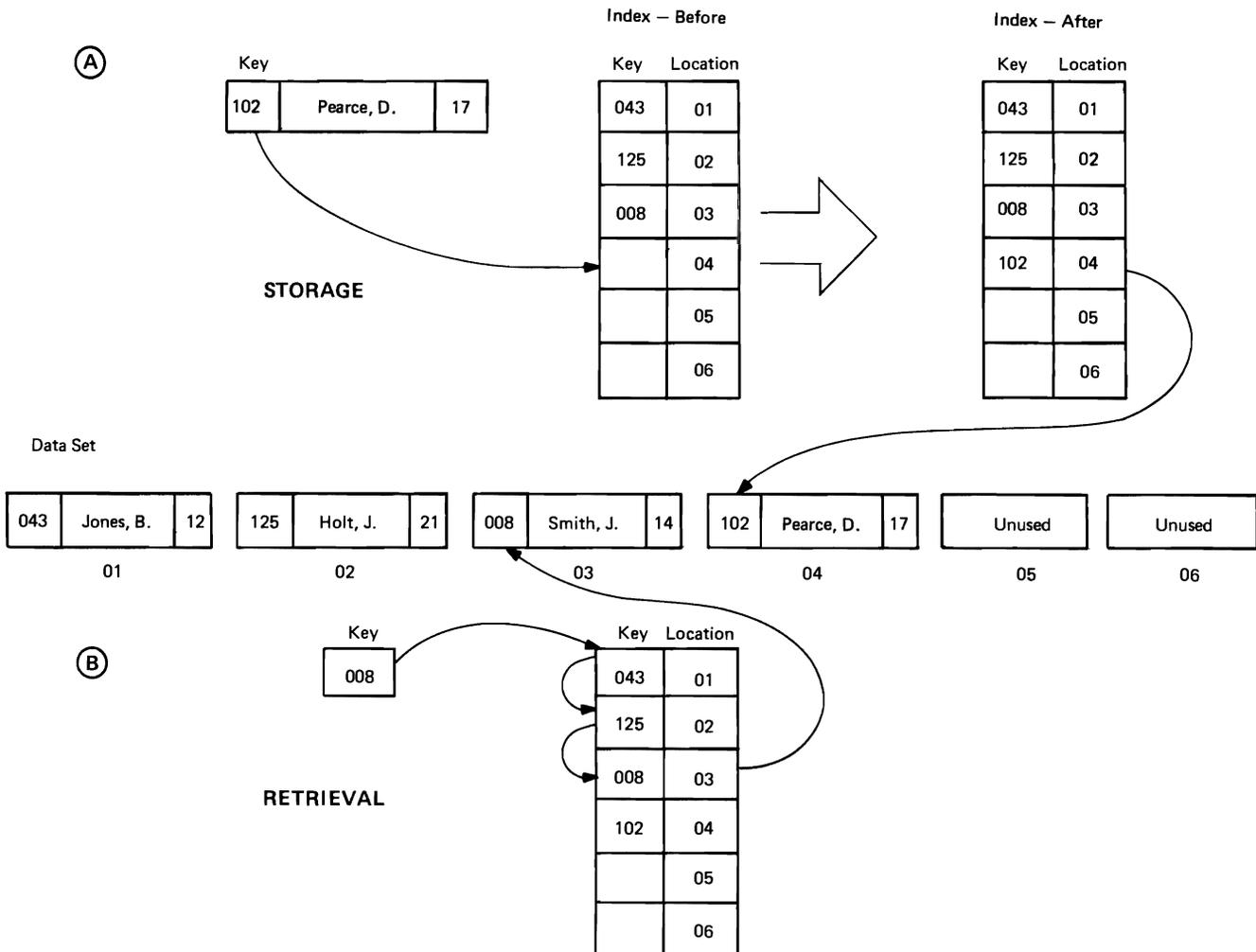


Figure 24. Index Search

* It is restricted to the use of numeric keys only.

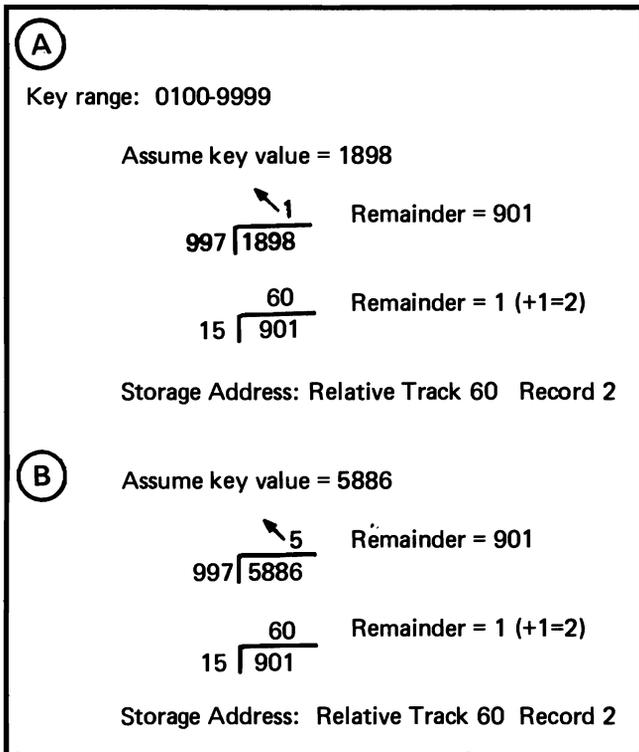


Figure 25. Example of Calculation Technique

used to access the given stored record. The index search method allows data sets to be allocated space based only on the actual space requirements of the stored records and not on all possible key values within a range. Also, keys do not have to be numeric values as in direct address, but can be any representation. Unique addresses are assured and any record can be accessed with only one seek once the address is obtained from the index. However, time is required to look up the address in the index, which may be a serious disadvantage when it contains many entries. The index also requires additional storage space in excess of that required by the stored records in the data set. For sequential retrieval the index can be sorted into key sequence.

The calculation or hashing method of direct organization is used for non-numeric keys and when the range of keys for a data set contains so many unused values that direct addressing would not be practical. The purpose of the calculation method is to manipulate the keys in a data set by some algorithm to compress the range of key values to a smaller range of stored addresses. Many techniques have been used to calculate valid storage addresses from a given set of key values. An example of one technique is shown in Figure 25(A). Consider a data set that will contain 1000 stored records on a device that will hold 15 stored records per track. A prime number is chosen close to 1000; in this case 997. The key is divided by this prime number, the quotient discarded, and the remainder divided by the number of records per track. The quotient from this

calculation is the relative track address from the beginning of the data set and the remainder plus one represents the stored record location on the track. In Figure 25(A), for key value of 1898, the result of this calculation gives an address of relative track 60 and the second stored record location on that track. This is a valid address because the data set contains 67 tracks (1000 locations divided by 15 locations per track).

An inevitable result of calculation techniques is the occurrence of synonyms — two or more stored records that result in the same storage address. In Figure 25(B) the same calculation is performed for key value 5886 as was performed for key value 1898 in Figure 25(A). Notice the storage addresses derived are identical. Various techniques are used to find a location for a synonym to a stored record already located in a data set. One technique is to read stored records sequentially until the nearest empty location is found and place the synonym there. Another is to provide a separate storage area for synonyms. Stored records placed in such locations can later be retrieved simply by searching until they are found or by using pointers to locate the synonym more rapidly. *

For example, when a stored record cannot be placed in the intended or “home” location because another stored record is already there, a pointer from which the address of the later stored record can be obtained is maintained as an additional data element in the first stored record. If additional synonyms are added later, pointers can be maintained from the next-to-last to the last synonym. Thus a “chain” of addresses is set up; the stored record at the home location will contain a pointer to the first synonym, the first synonym will point to the next, and so on until the last synonym either points back to the home address or contains an arbitrary value recognized to mean “end of the chain” (Figure 26). This technique reduces retrieval time at the expense of maintaining an additional data element in each stored record and at the expense of increased complexity in updating the data set because the previous stored record’s pointer must be changed whenever a new synonym is added or relocated.

The basic objectives of a calculation technique are to derive a valid address for every key in the data set and to distribute the addresses as evenly as possible across the key range to minimize synonyms. A sought-after goal is to have no more than 20% synonyms.

Direct organization is most suited to rapidly accessing a particular stored record with a known key value. It is less suited to rapidly accessing a large number of stored records following a given relationship as in the sequential organization. This is because the time required for the hardware access mechanism to locate a given record will normally be much greater if the next stored record is not physically adjacent to the current one.

* A pointer in this case is the direct address of the stored record in question.

Stored Record

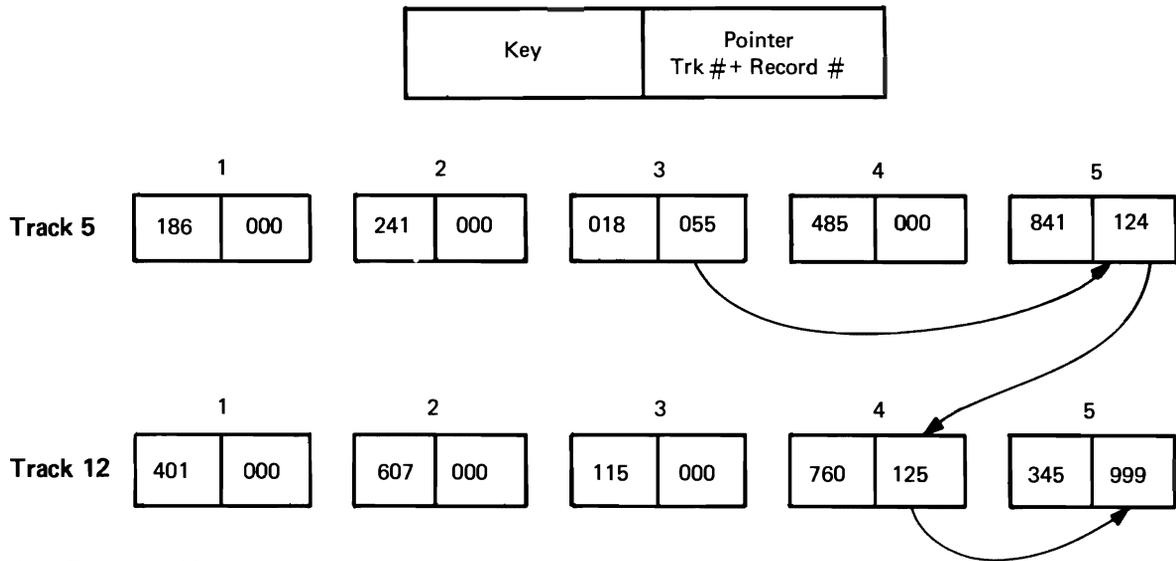


Figure 26. Chaining of Synonyms

'999' Signifies End-of-Chain

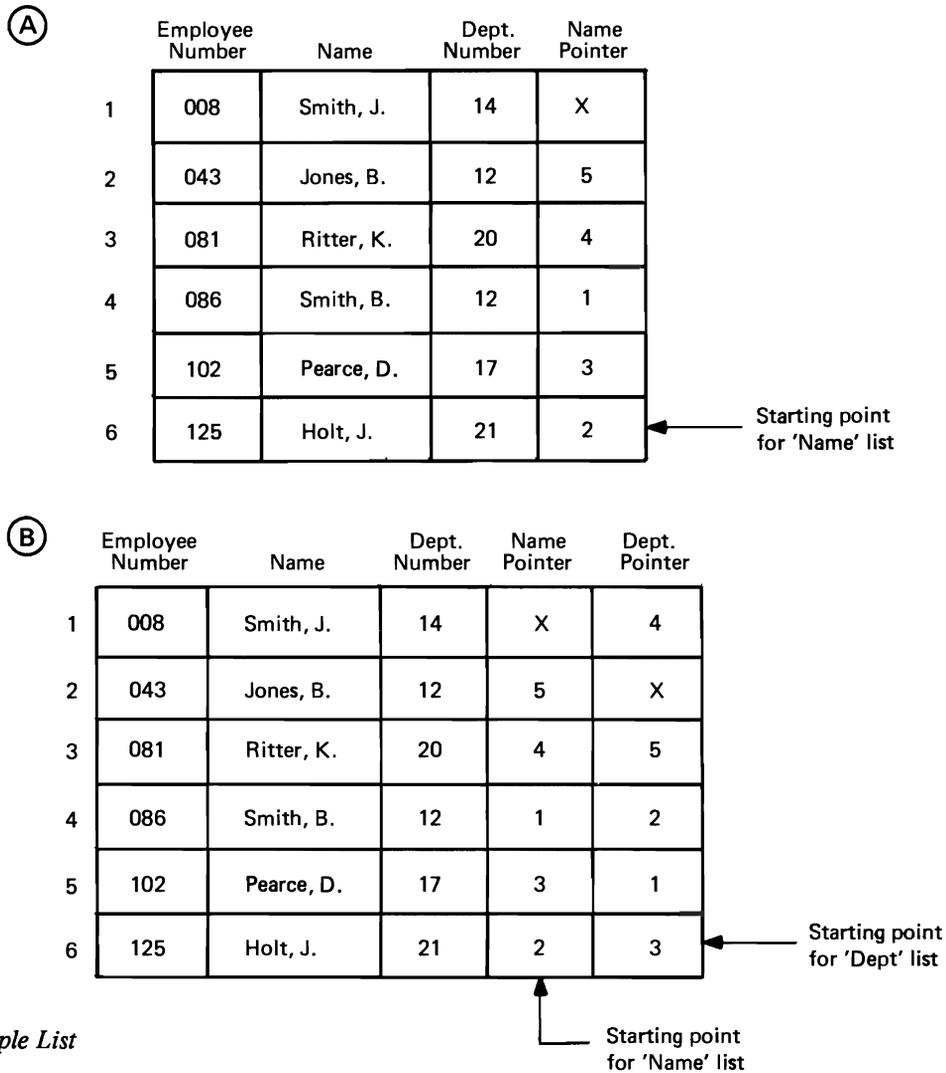


Figure 27. Simple List

Most methods used in direct organization result in considerable unused storage or many groups of synonyms.

LIST ORGANIZATION

The basic concept of a list organization is the use of pointers to define the relationship between stored records. A pointer is a data element within a stored record that contains as a value the address of another stored record. By including a pointer within each stored record, relationships between stored records can be established that are independent of key values and physical location. By using more than one pointer in each stored record, multiple relationships can be established with one physical sequence or a data set. Figure 27 gives an example of a simple list organization. It is the data set of Figure 23(A) with a pointer added to each stored record. We wish to establish a relationship between the stored records based on the alphabetical ordering of the data element NAME. We find that the stored record in location 6 is the starting point of the list. This stored record contains a pointer to the stored record in location 2 which in turn contains a pointer to the stored record in location 5, etc. The end of the list is signified by the letter 'X' in the example. We see that by adding one data element to each stored record we can process the data set in employee number sequence by means of the physical ordering of the data set and also in alphabetical order by employee name by using the list pointers. We add another data element as a pointer in each stored record in Figure 27(B) and we have established all the relationships between the stored records in one data set that required three separate data sets in Figure 23. Therefore, one obvious advantage of list organization is the possibility of multiple relationships between stored records using only one copy of the actual data. Updating is simplified under simple list organization if data elements not used as keys are changed because a change to a non-key data element applies automatically to all lists passing through the stored record. Updating data elements used as keys requires changing the list pointers for that particular relationship.

Inserting new stored records into a list is accomplished by simply changing the pointer in the preceding stored record to point to the one being inserted. The pointer that was originally in the preceding stored record is placed in the new stored record. If the new stored record is to be in more than one list, each list must be searched from the beginning in order to place the stored record correctly within the given relationships. Deleting stored records is simple if only one list passes through the stored records. The list must be searched to locate the stored record and, when it is located, the pointer from the stored record to be deleted replaces the pointer in the preceding one. Deleting a stored record through which multiple lists pass is difficult

unless pointers are maintained to the preceding stored record as well as to the one following.

Insertion and deletion of stored records in a data set can be made slightly easier by using a ring organization. Rings are an extension of the simple list organization and are created by having the last pointer in a list point back to the first stored record in the list rather than indicate "end of list". The first stored record in the list will have some type of identifying symbol to denote that it is the beginning of the list. When multiple lists pass through a given stored record that is to be inserted or deleted, the starting point record or the record that precedes or follows the desired record can be found by searching the ring. Unless backward pointers are maintained, it is not possible in a simple list to find the preceding stored record without locating the list starting point in some manner and searching from there. Ring organization allows the preceding stored record to be found without locating the starting point or using backward pointers. However, this would be a time-consuming process with large lists.

Because locating a specified stored record in both simple list and ring organizations can be a relatively slow process for long lists, a way to reduce the list length is advantageous. This can be accomplished with an inverted list organization. In a simple list, pointers are maintained with the stored records to relate these stored records according to the values of specific data elements used as keys. In an inverted list, the data element values are placed in an index that relates the value of the particular data element to the locations of stored records that correspond to the given value. For example, look at the data set in Figure 23(A). If the data element DEPT.NUMBER were placed in an index with pointers to the appropriate stored records, the result would be as shown in Figure 28(A). Notice that no pointers are included in the data set and that the relationship between the stored records based on DEPT.NUMBER is contained in five short lists in the index rather than in one longer list in the data set. To retrieve the stored records that contain data about employees who work in Department 12, the index would be scanned for the key value 12 and the data set accessed for stored records in locations 4 and 2. As a further example, suppose the data set of Figure 23(A) had a data element added to each stored record to indicate the sex of each employee as shown in Figure 28(B). This example points out the ability of an inverted list organization to handle classification requests such as "how many female employees work in Dept. 12?". To satisfy this request, the index is searched to find the lists corresponding to 12 and FEMALE. Only the stored record in location 4 contains both attributes, so that record is retrieved. The request was satisfied with only one access to the data set with the majority of the processing being done in the index. Another point to be made from the example in Figure 28(B) is that keys used in an inverted list organization are more effective if they contain classes of values such as MALE or FEMALE; MARRIED, SINGLE,

(A)

Index

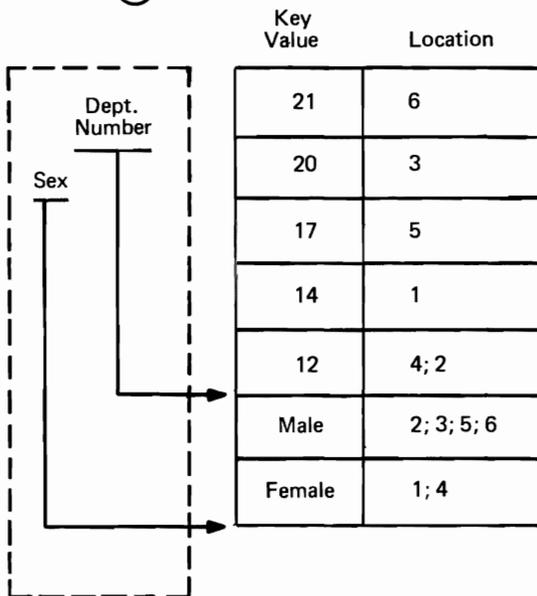
Key Value	Location
21	6
20	3
17	5
14	1
12	4; 2

Data Set

	Employee Number	Name	Dept. Number
1	008	Smith, J.	14
2	043	Jones, B.	12
3	081	Ritter, K.	20
4	086	Smith, B.	12
5	102	Pearce, D.	17
6	125	Holt, J.	21

(B)

Index



Data Set

	Employee Number	Name	Dept. Number	Sex
1	008	Smith, J.	14	Female
2	043	Jones, B.	12	Male
3	081	Ritter, K.	20	Male
4	086	Smith, B.	12	Female
5	102	Pearce, D.	17	Male
6	125	Holt, J.	21	Male

Figure 28. Inverted Lists

or DIVORCED; rather than keys with inherently unique values such as EMPLOYEE NUMBER. Unique keys generate a large number of short lists and greatly increase the length of the index. This can be alleviated somewhat by having a higher level index to delineate key ranges as is shown within the dotted lines in Figure 28(B).

In the general case, the inverted list organization can use every data element as a key so that all data can be accessed equally well. This would be very advantageous for unpredictable data retrieval environments. However, the indexes would take a great deal of storage space and the problem of updating becomes very serious. A common compromise approach is to organize the data in a sequential or direct

manner and use inverted lists only on selected keys. These inverted key lists are usually referred to as secondary indexes.

REPRESENTING DATA STRUCTURES

In Chapter 2, three types of logical data structures were defined — simple, hierarchy, and network. These three categories of data structures are also applicable to relationship between data elements within and between stored records. Therefore, let us look at some examples of the basic data organizations in the context of these three types of data structures. No attempt will be made to show all possibilities. We will simply demonstrate how a particular data structure might be represented by an appropriate data organization.

Stored Record

Employee Number	Name	Dept. Number	Hire Date
10342	Johnson, R.	14	12-4-69

Graph

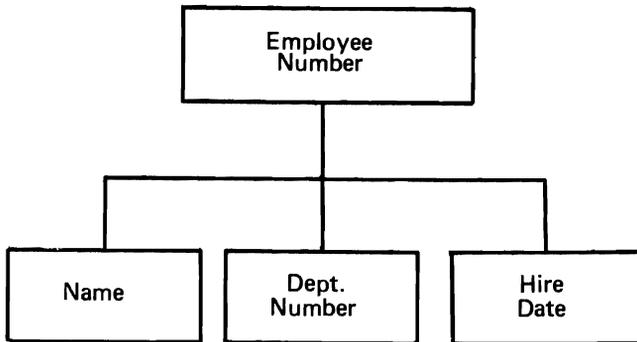


Figure 29. Simple Data Structure

The example of a simple data structure shown in Figure 29 is similar to the stored record format we used in previous examples. It is a simple structure because the data elements 'NAME', 'DEPT NUMBER', and 'HIRE DATE' are dependent on the value of the key 'EMPLOYEE NUMBER' to derive meaning. Let us assume that we have a data set of 10,000 stored records of the format in Figure 29 and the requirement is to be able to process these stored records sequentially by the value of 'EMPLOYEE NUMBER'. We must also be able to directly access any stored record pertaining to a given employee. If we create the data set using the sequential data organization and place the stored records physically adjacent to each other in sequence by employee number, we can satisfy the first requirement. However, because sequential organization does not lend itself to rapidly accessing a stored record other than the one that is physically located next on the storage device, we must have something more than simple sequential organization.

Remember that the index search type of direct data organization allowed both sequential and direct processing of a data set in some cases. Therefore, if we physically place the stored records in the data set in sequence by the values of the key 'EMPLOYEE NUMBER' and also place the key values in sequence in the index, we can achieve both sequential and direct processing.

We have met the processing requirements for the data set but there are several weaknesses. The index used to directly access the stored records contains 10,000 entries. This might lead to lengthy search times to locate the particular key value. We could reduce this time by creating a higher level index to locate a more appropriate starting point for the search but this approach increases the total size of the indexes used. Another approach might be to place only the key and storage location of every 10th stored record into the index, thereby reducing the index size in this example to 1,000 entries. To retrieve a given stored record, the index is sequentially scanned until the desired stored record is located. This approach is sometimes known as "sparse indexing".

This approach would be satisfactory except that we cannot add stored records and keep the sequence without rewriting the entire data set and index. We might overcome this by placing stored records to be added in a separate area of the data set called an "overflow area" and use pointers to maintain the sequence. A common approach is to use a second index entry for each group of stored records (in this example, every group of 10). The index entry indicates the existence of overflow entries and, if any are present, where the overflow stored records are located. If more than one overflow stored record is present for a given group, pointers might be used within them to maintain the sequence. Figure 30(A) shows how a portion of the index and data set might look before the addition of stored records with key values of 14, 15, and 29. Figure 30(B) shows the index and data after the additions. The additions required only rewriting two groups of stored records and two sets of index entries and writing three stored records in the overflow area instead of re-copying the entire data set and index. The approach just described is the basis for a common data organization known as Indexed Sequential.

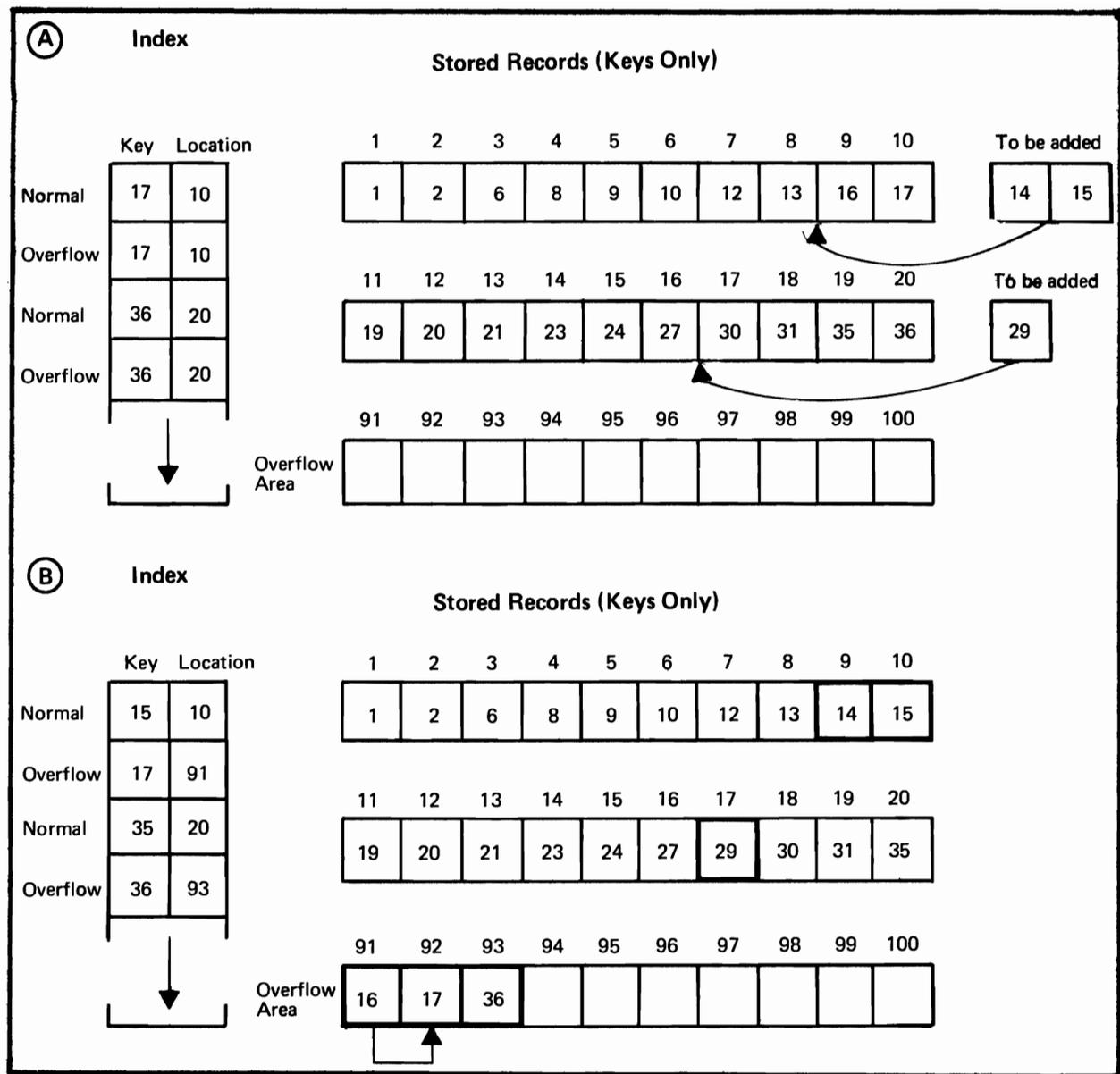
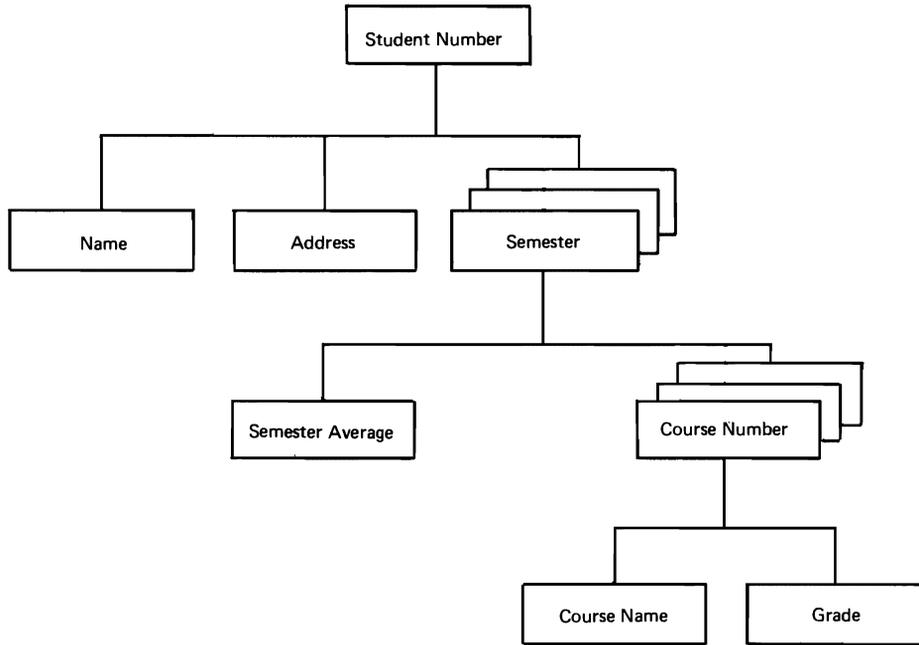


Figure 30. Addition of Stored Records

(A)



(B)

Student Number	Name	Addr	Semester 1	Avg	Course Number 1	Course Name	Grade	Course Number 2	Course Name	Grade	...	Semester 2	Avg	Course Number 1	Course Name
----------------	------	------	------------	-----	-----------------	-------------	-------	-----------------	-------------	-------	-----	------------	-----	-----------------	-------------

Figure 31. Hierarchical Data Structure

Figure 31(A) shows a graph of a hierarchical data structure. It is hierarchical because certain data elements such as COURSE NUMBER and GRADE depend on the values of data elements other than the key for meaning. We could combine all occurrences of the data elements into one stored record as in Figure 31(B). The stored record would be of variable length depending on the number of semesters the student was in school, the number of courses per semester, etc. A data set containing variable length stored records presents unique updating problems when the length of a stored record is changed. In addition, the length of the stored record may become rather long, resulting in the unnecessary transfer of data for a particular request because only a limited number of data elements might be needed to satisfy the request. Therefore, it would be to our advantage to break the stored record in Figure 31(B) into smaller, fixed-length stored records and try to take advantage of the hierarchical structure.

Looking at the graph in Figure 31(A) we see that the variable nature of the data stems from multiple occurrences of SEMESTER for a given student and multiple occurrences of COURSE NUMBER for a given semester. Therefore, we might define three types of stored records for a given student and use the simple list data organization to relate the various types as shown in Figure 32. The fixed data about a student such as NAME and ADDRESS are placed in one stored record per student and arranged in sequence by STUDENT NUMBER. A pointer is appended to this stored record to point to the first stored record containing semester data that could be contained in a different data set. For each student, there will be one stored record for each semester he was in school containing fixed data for a given semester (such as SEMESTER AVG) and a pointer to the next semester's stored record. In addition, for each semester there is a pointer to the first stored record in a list containing data pertaining to courses in which the student was enrolled during that semester.

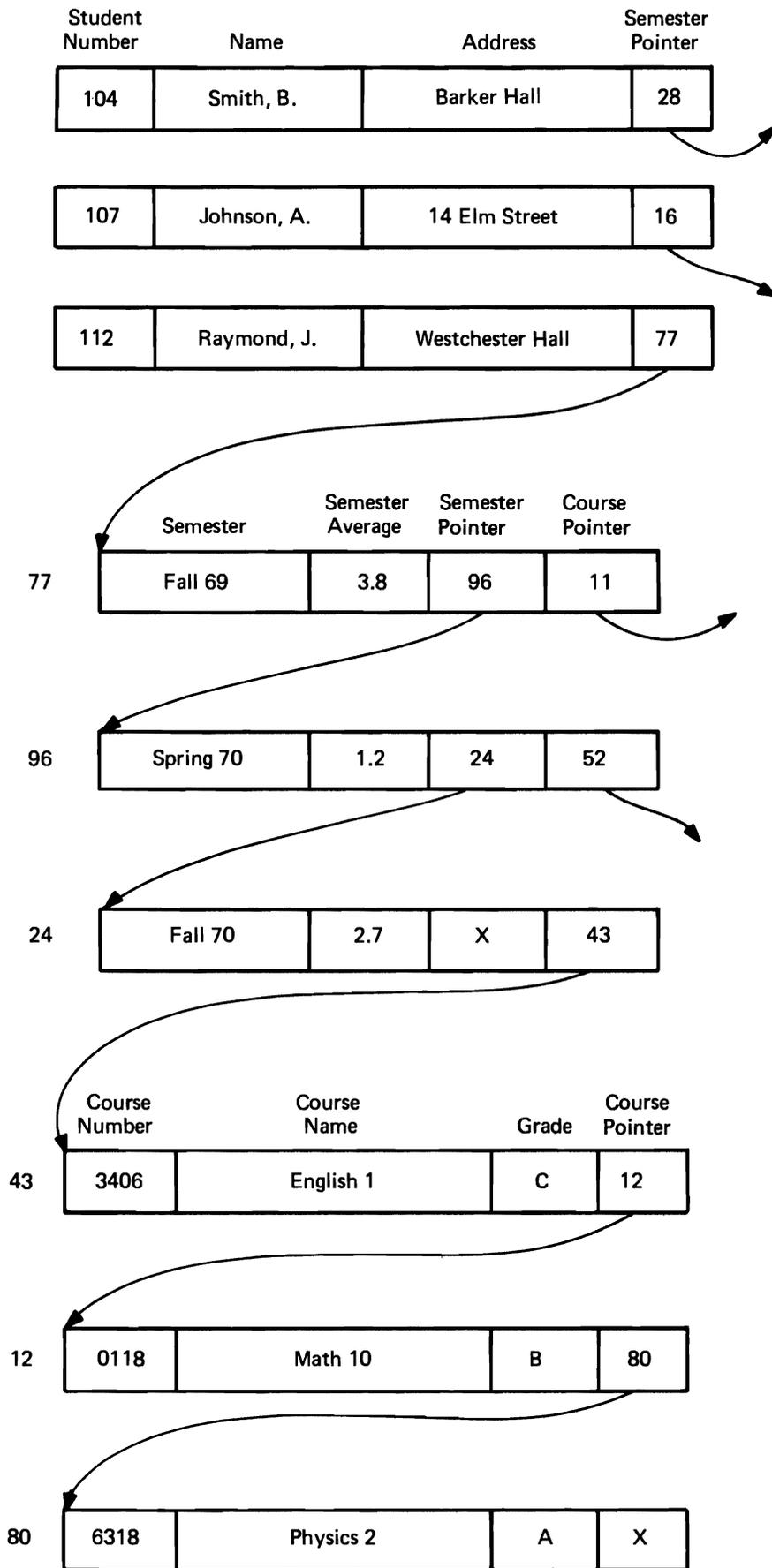


Figure 32. Use of Simple List Organization

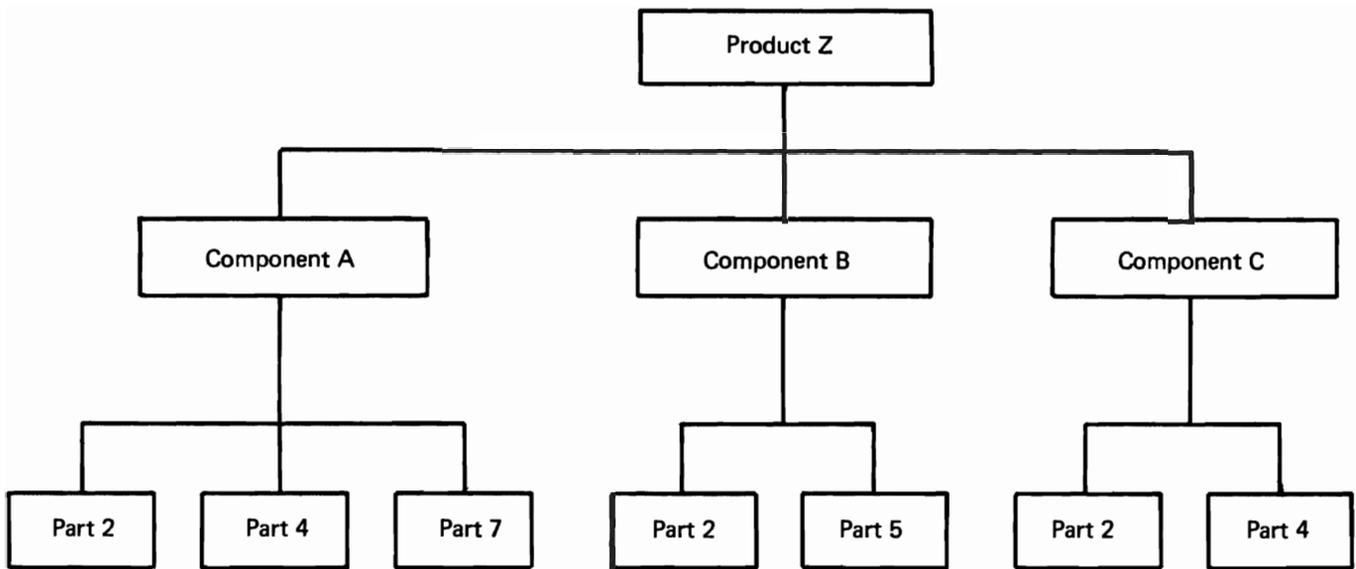


Figure 33. Network Data Structure

Updating the data about a particular student is eased somewhat under this data organization. Suppose we wish to add an additional course for student number 112 for the FALL 70 semester. We locate the stored record with student number 112. This stored record contains a pointer that tells us that the semester list begins in location 77. We retrieve each semester's stored record by means of the SEMESTER POINTER data element until we encounter the one for the FALL 70 semester. This stored record contains a pointer that tells us that the course list begins in location 43. This list is searched until the last stored record is retrieved (denoted by an X in the COURSE POINTER data element) and the new course stored record is added by changing the X to the location of the new stored record. If we had organized the data as in Figure 31(B), it would have been necessary to rewrite the entire stored record due to the increase in its length. In addition, no course data was retrieved other than that pertaining to the semester in question.

The last data structure we will consider is a network data structure as depicted as a hierarchy by duplication in Figure 33. It shows a product made up of components that in turn are composed of specific parts. Each block in the graph represents a stored record. We wish to be able to find all the components making up a product, all the parts in a given component, and to identify all components that contain a given part. We might be able to accomplish these objectives by using the ring variation of list data organization. Looking at Figure 34(A), we can accomplish the first objective by maintaining a pointer from the product stored record to the first component stored record, which in turn will contain a pointer to the next component, etc. When the last component stored record is retrieved, it will point back to the product data. Each component will have another data element used as a pointer to locate the first stored record containing part data. The part stored records will be chained together until the last one is reached. It will point back to the first as shown in Figure 34(B) for part numbers 2 and 4. Similarly, if a pointer were placed into each component stored record to point to other stored records under different products that contain the same component number, we could locate all products that contain a given component. The stored records described might be as shown in Figure 35.

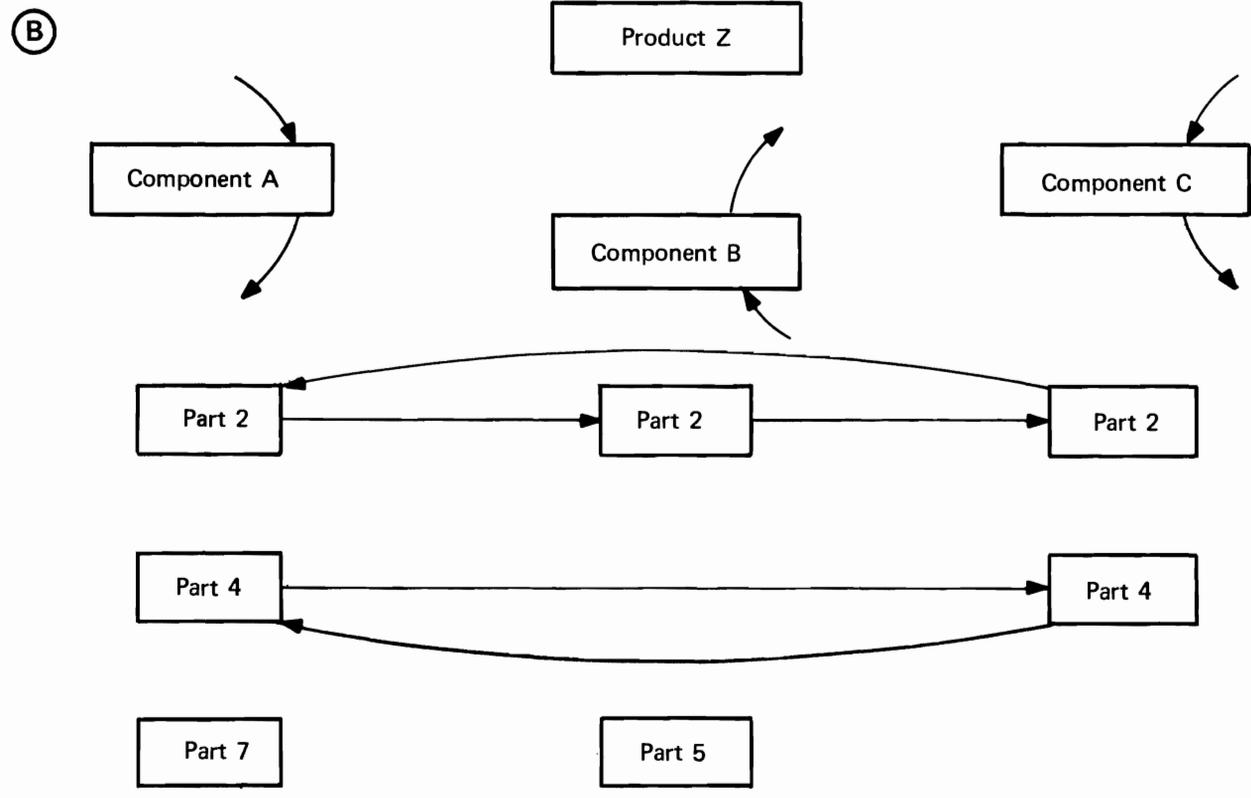
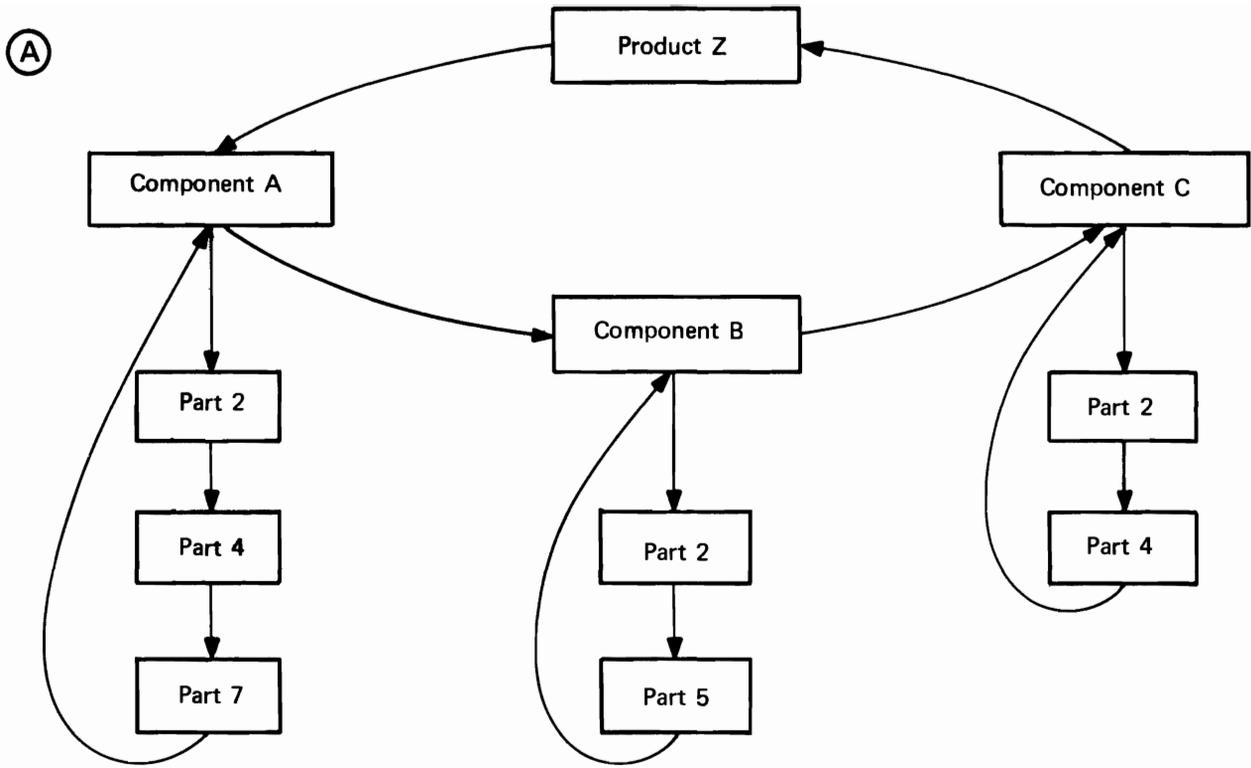


Figure 34. Use of Rings

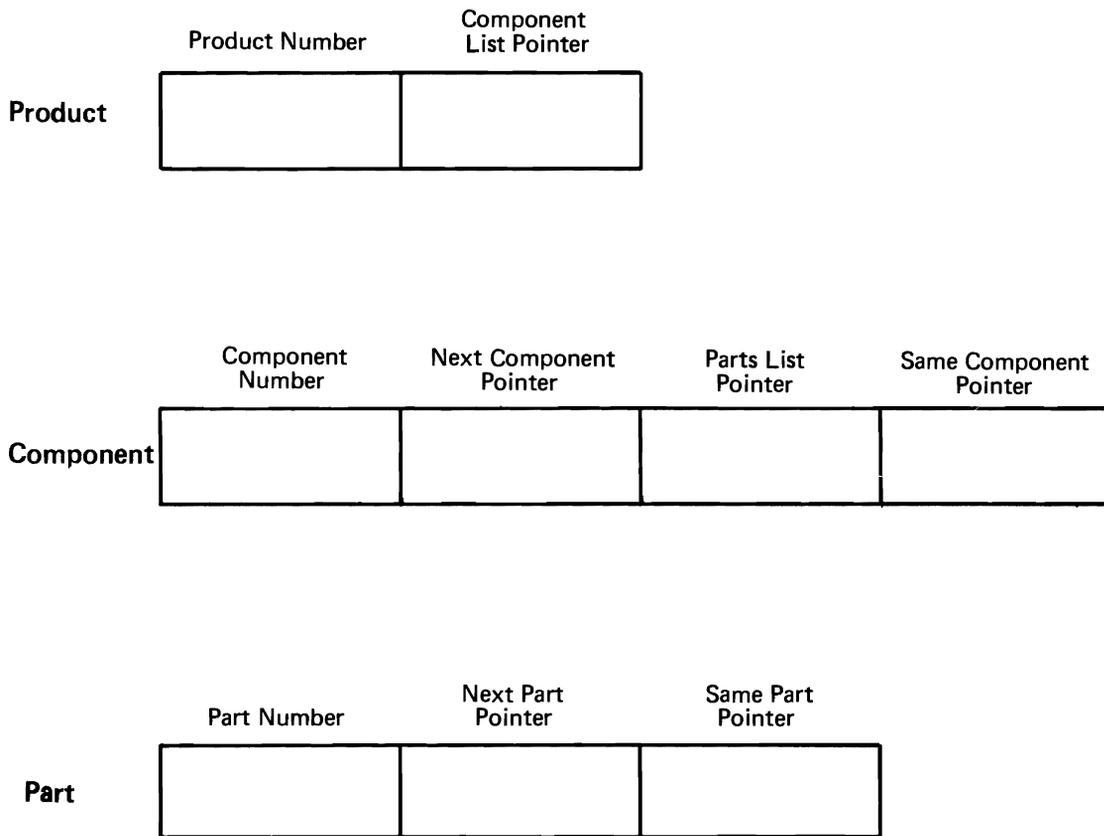


Figure 35. Stored Record Descriptions

We can now get an idea of the power of rings in network data structures. For example, not only can we locate all components making up a product, but we can locate all products that contain a given component by retrieving each stored record using the SAME COMPONENT pointer until the starting component stored record is reached. For any component we can retrieve all parts data pertaining to that component. For any given part within a product, we can find all components that contain this part by searching the SAME PART list. For example, this would be helpful if a particular part could be made at a lower cost. We would want to determine the impact on the price of the finished product. It is easy to see that the ring data organization allows us to get from any stored record to any other by following the appropriate list pointers. However, if the rings are long or the stored records widely dispersed throughout the data set, retrieval could be rather time consuming.

Chapter 5: Functions of Data Management

INTRODUCTION

Up to this point we have looked at three aspects of information storage:

- The user's concept of information consisting of fields, logical records, and files (Chapters 1 and 2).
- The device concept of information consisting of physical blocks recorded on a storage medium (Chapter 3).
- The system concept of information consisting of stored records and data sets (Data Organization, Chapter 4).

MAPPING

It is the function of data management to map the information from physical blocks into stored records into logical records. The organization of stored records and their mapping into physical blocks is called DATA ORGANIZATION. The organization and mapping of stored records into logical records is called FILE ORGANIZATION (see Figure 36).

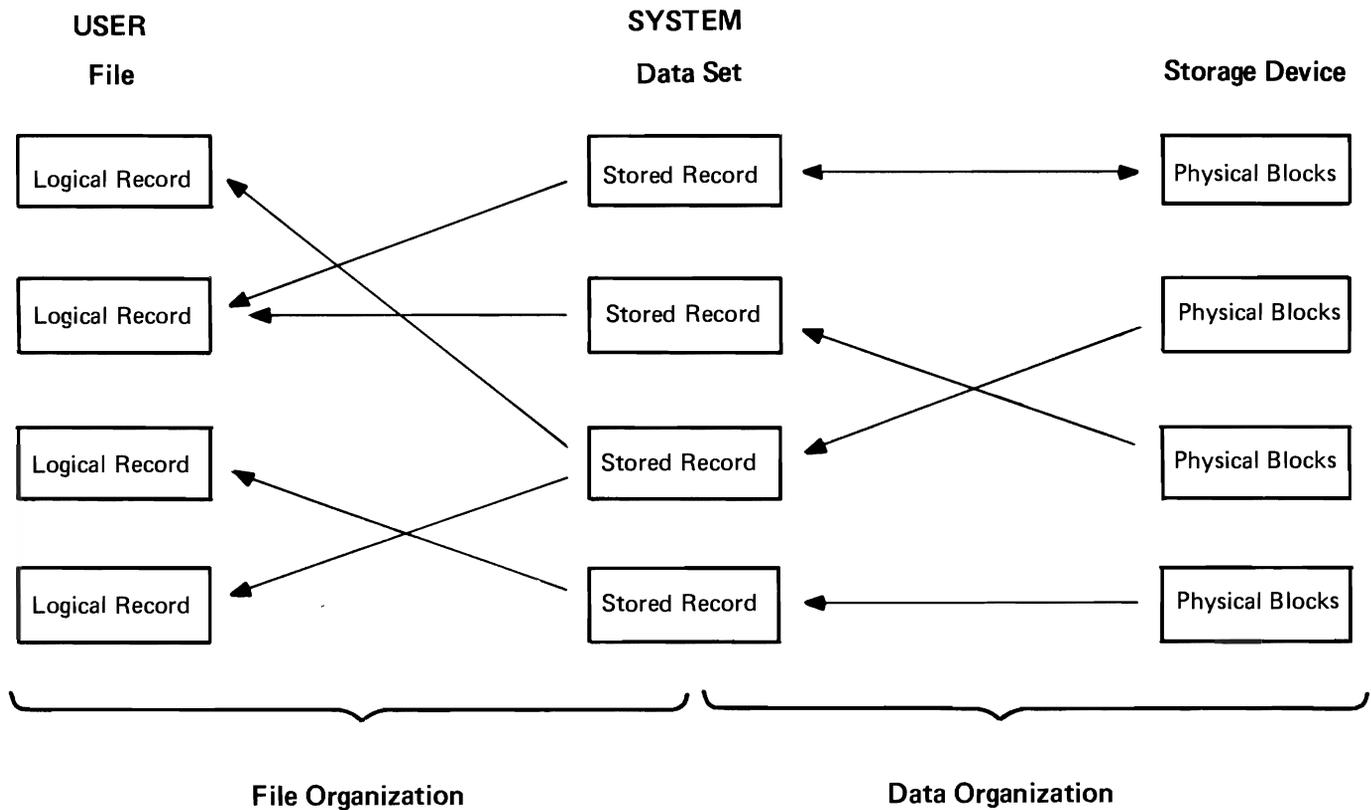


Figure 36. File and Data Organization

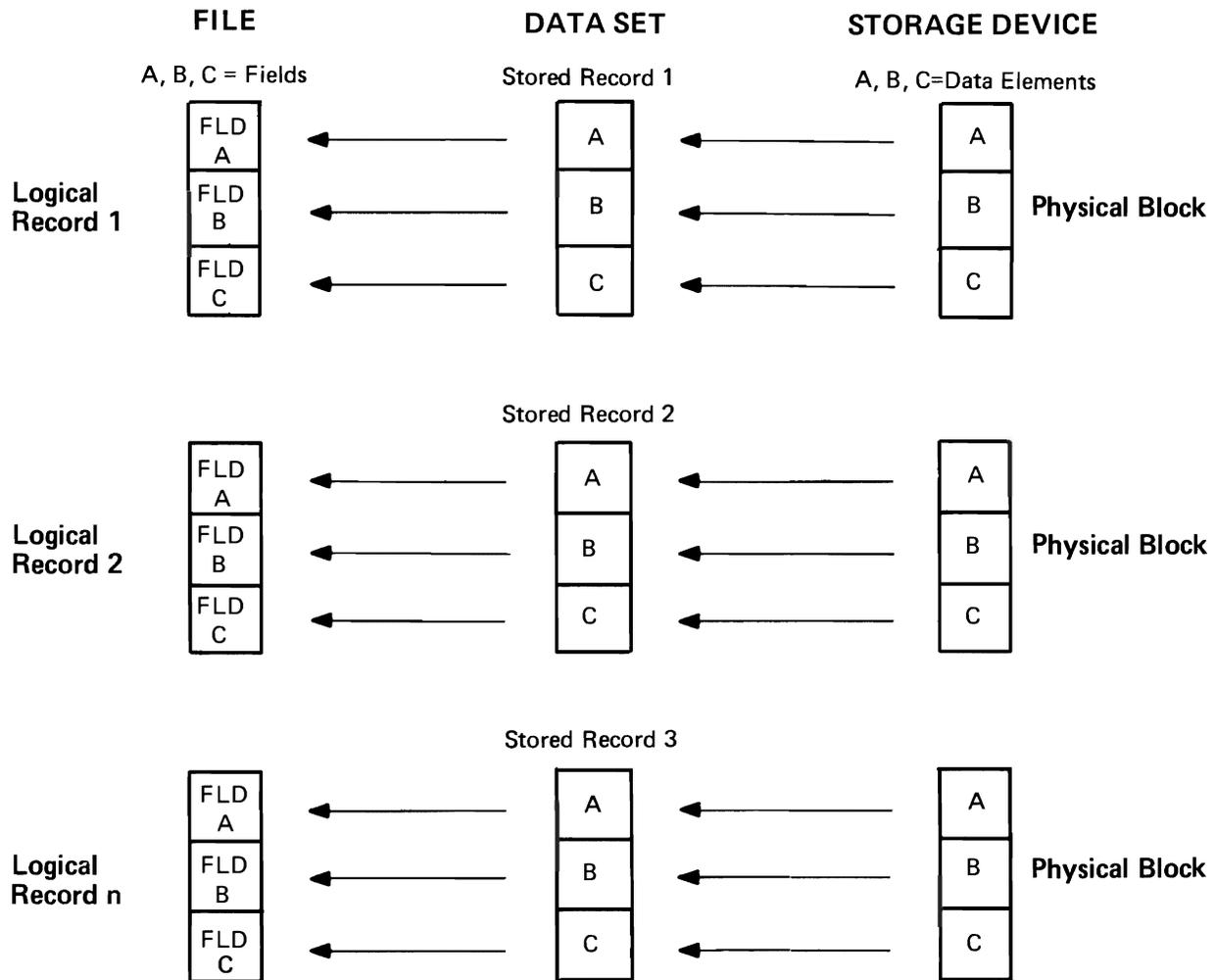


Figure 37. Field and Sequence Mapping (Simplest Case)

The simplest mapping possible is shown in Figure 37. The logical, stored, and physical records are identical, as are the file, data set, and sequence of the physical blocks. In Figure 38, a more complex mapping is shown. The logical sequence of the stored records is different from the physical sequence of the blocks, and the field position in the logical records is the reverse of the stored record data elements. This example suggests two distinct types of mapping: field, and record sequence. Figure 39 shows complex field and record sequence mapping for both system and logical file organization.

In addition to field and record sequence mapping, transformations can also be performed between data set and file. Transformations include changes in data representation, encoding and truncation (see Chapter 2).

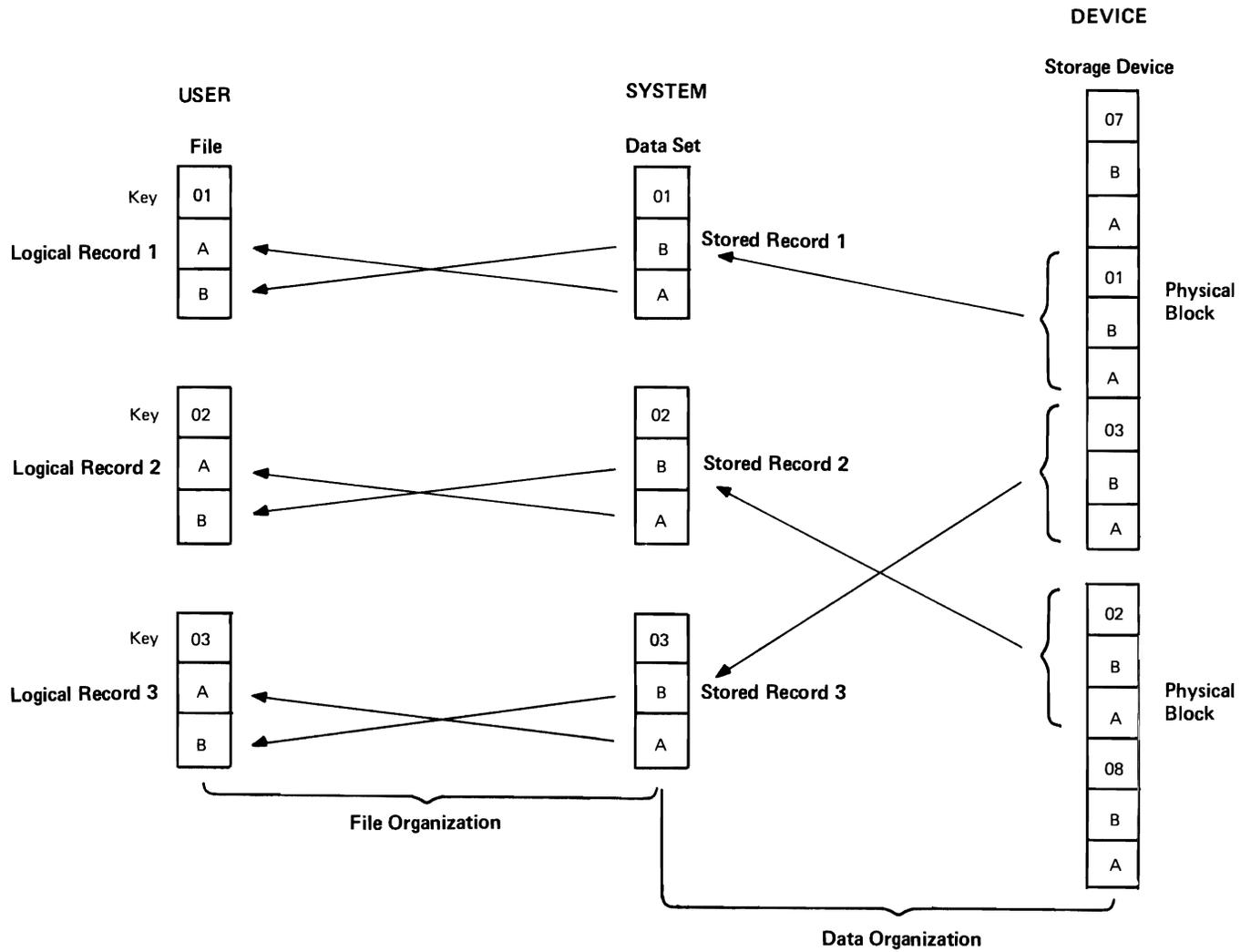


Figure 38. Field and Sequence Mapping

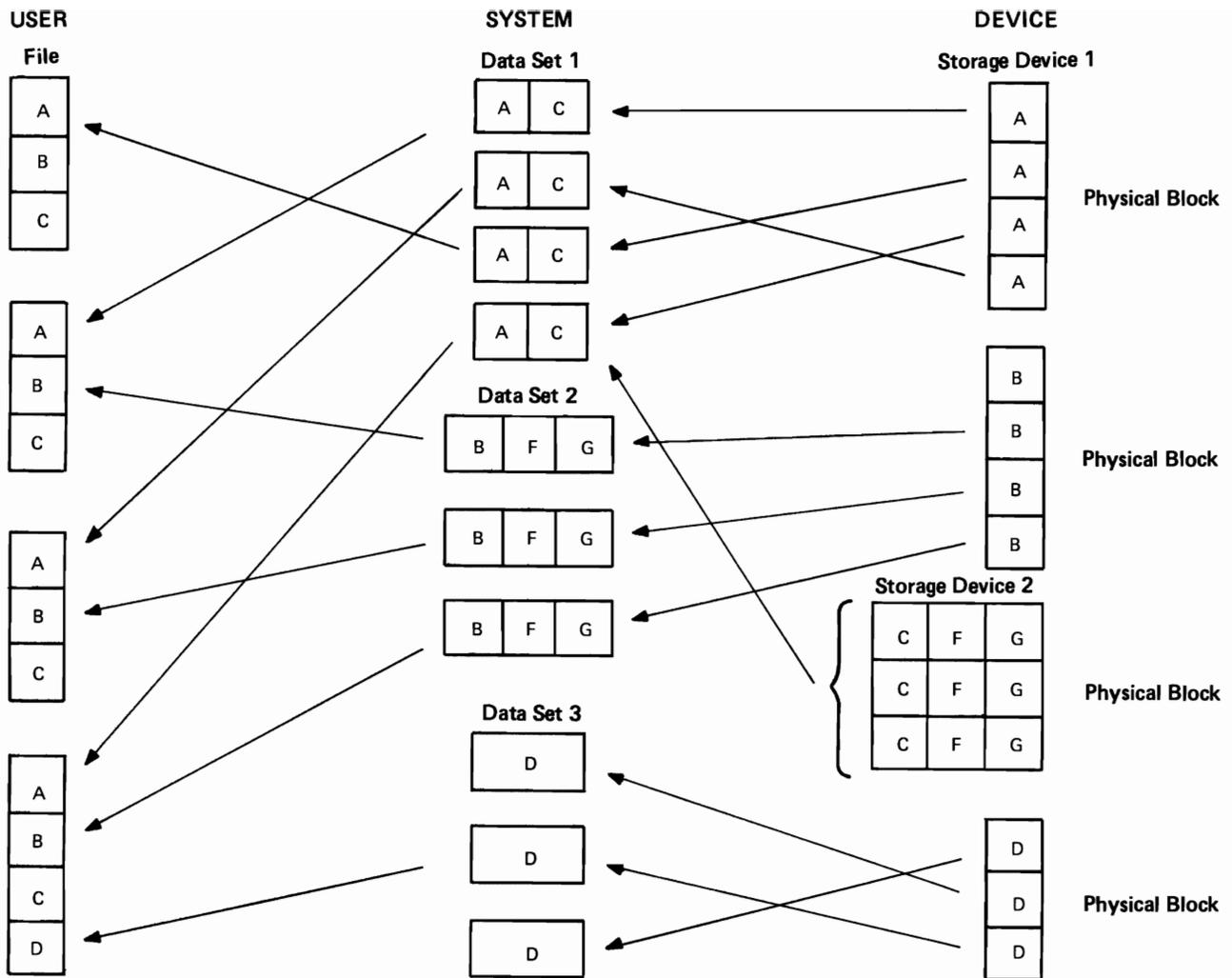


Figure 39. Complex Field and Sequence Mapping

Information Context and Data Independence

To perform sequence mapping the system must know the record keys. Field mapping and field transformations require that the context and data representation be known to the system. System knowledge of context is the most important design criteria of an information system.

In the past, knowledge of the context and data representation was contained in the user's problem program (see Figure 40). The system therefore could perform no field mapping or transformation at either the File or Data Organization levels. If the order of the fields within a stored record was changed or a data field was represented in a different fashion (character string changed to fixed decimal), then all users of the data set had to change their processing programs. When the system knows the context and can perform field mapping and transformation, the data set content and data organization can be changed without affecting the user's programs (see Figure 41). The user's program becomes data independent. Data independence will be discussed in more detail in Chapter 6.

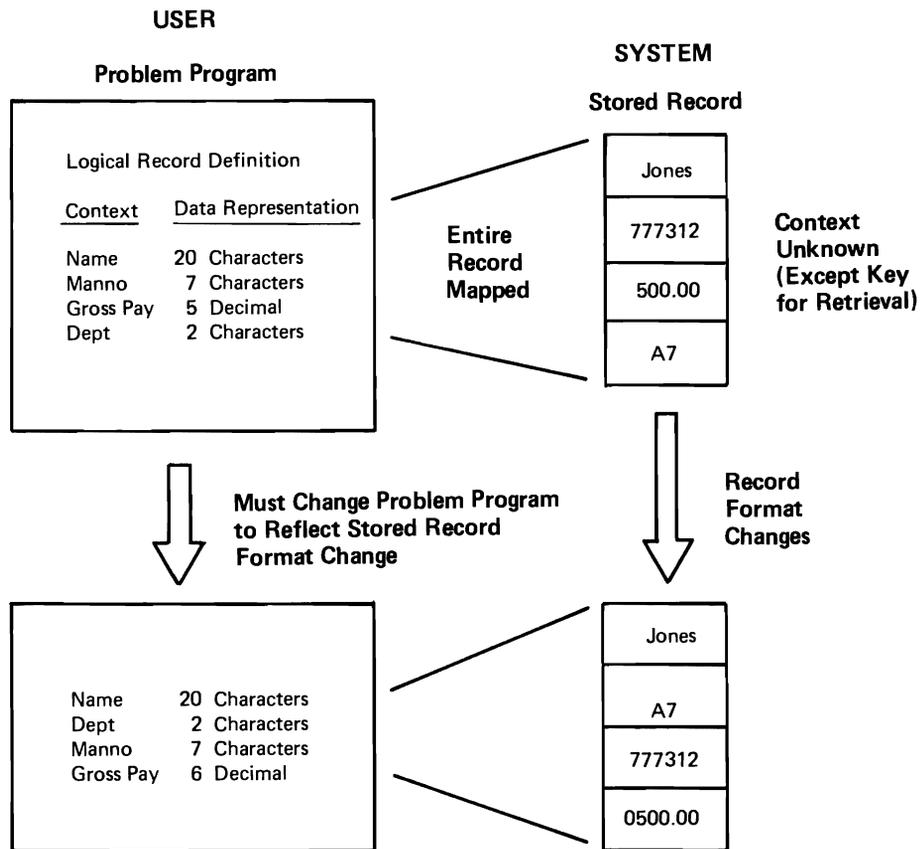


Figure 40. No Data Independence

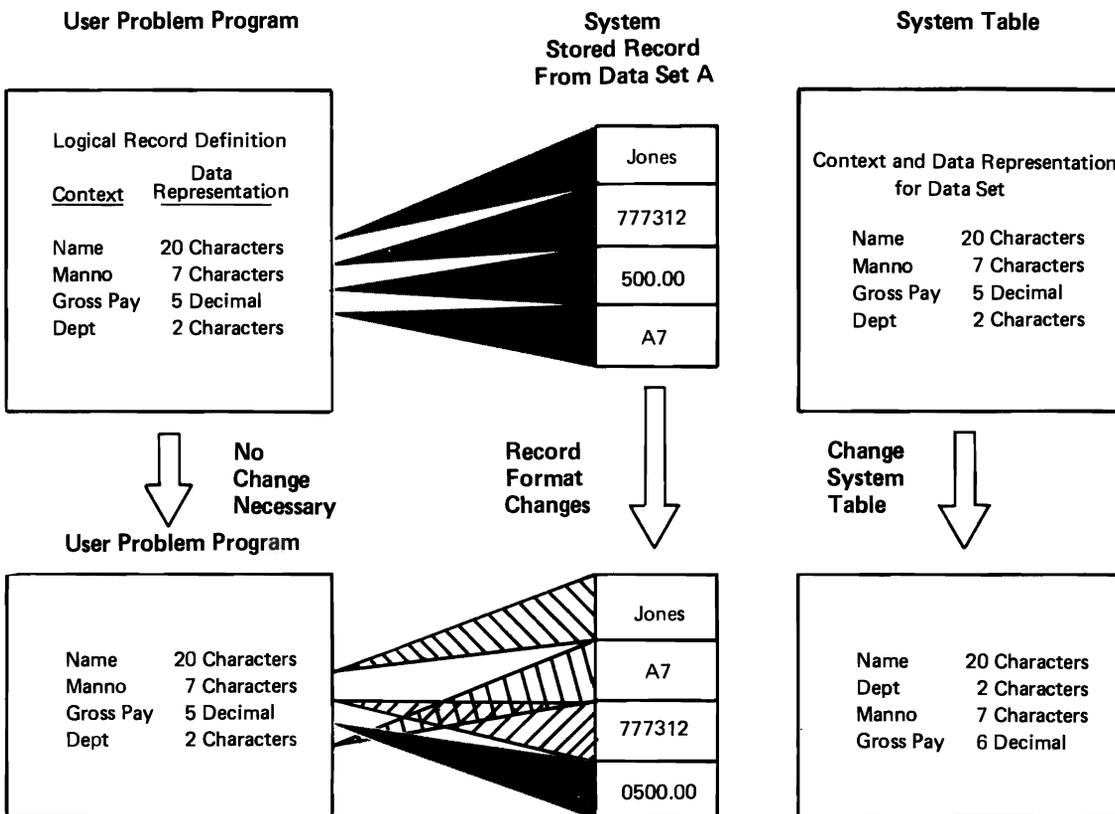


Figure 41. Data Independence

DATA MANAGEMENT FUNCTIONS

In Chapter 1 we described the basic functions of data management as:

- Control
- Retrieval
- Storage

To illustrate these functions we will discuss what might be involved in processing a request by a user for information (Figure 42). The system described is a hypothetical one and is used to illustrate the necessary functions.

Figure 43(A) shows the information (file and logical record) requested from the system by the user. The user wishes to process a file that is in sequence by department and, within department, by man number. He specifies in his definition of a logical record the names of the fields he requires, their data representation, and how he intends to process those fields (Read only, Update). Figure 43(B) also shows the two data sets that contain the data that the user wishes to retrieve. These data sets reside on removable disk packs. We will assume that our system allows the user to be totally ignorant of where or how the data he requests is stored (Data Independence). He need only know the names of the fields he requires. In order to be able to provide the user data independence and ensure the validity and privacy of the data, the system must have information about both users and data. This information is stored in system directories (see Figure 44).

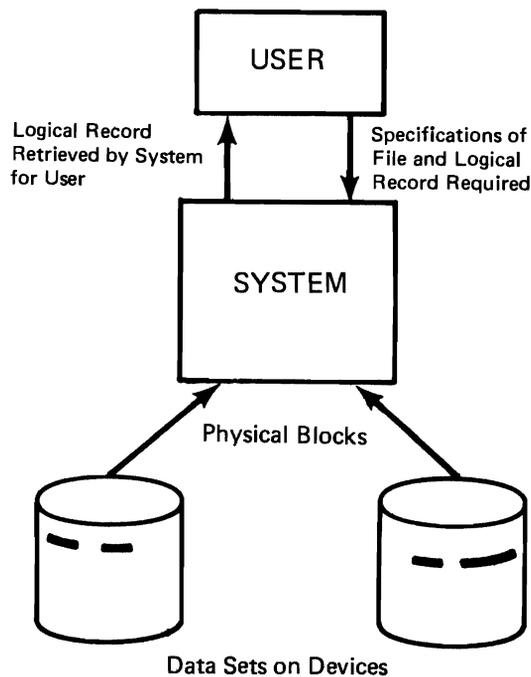


Figure 42. Information Flow to Satisfy User Request

Problem Program

User ID B7601	
Retrieve File Sequentially in Sequence by DEPT/MANNO	
<u>Logical Record Definition</u>	
Dept Manno Salary Name Manager Yrs of Service	} Read Only
Job Title	Update
Last Appraisal	Read Only

Figure 43(A). User Request

<u>Payroll Data Set</u>		<u>Personnel Data Set</u>
Stored Record Definition		Stored Record Definition
.Manno (Key) Name Salary YTD Gross Deductions	} Primary Index of both data sets is on Manno	{ Manno (Key) Name Dept Job Title Hire Date Last Appraisal Manager

Figure 43(B). Data Sets Available

User Directory

ID	Name	Accounting Information	Facilities Allowed
A273	_____	_____	_____ _____ _____
B7601	J. Jones	Z72-A26-01	70K Core 2 Tapes 3 DASD Service Prog 1
B7999	_____ _____	_____	_____

Data Set Directory

D. S. Name	Volume	Key	Organization	Fields
Inventory	_____	_____	_____	_____ _____ _____
Payroll	123456	Manno	Indexed	Manno Name Salary YTDGR Etc.
Personnel	789102	Manno	Indexed	Manno Dept Job Title Etc.

Figure 44. System Directories

User Directory

This directory contains the identification of users who are allowed to use the system. It might also include the user's name, and accounting information to be used for billing. Because each user may be limited to certain services, the facilities he is allowed to use are contained in this directory.

Data Set Directory

This directory contains the names of the data sets, the volumes on which they reside, the key, organization method, and the users authorized to use them.

Field Directory

Name	Data Set	Data Representation	Key ^①	Index ^②	Authorized Users ^③	
					ID	R/W/U
Education	_____	_____	_____	_____	_____	_____
Department	Personnel	2 Characters	No	Yes	B7601 Z2929	R R/U
Hire Date	Personnel	8 Characters	No	Yes	A2771 B7601	R R
Job Title	Personnel	10 Characters	No	Yes	B7601	R/U
Last Appraisal	_____	_____	_____	_____	_____	_____
Manager	Personnel	10 Characters	No	Yes	B7601	R/U
Manno	Payroll	7 Characters	Yes	Yes	A7731	R
	Personnel	7 Characters	Yes	Yes	B7601	R
	Project	7 Characters	No	Yes	D8612	R
Name	Payroll	20 Characters	No	Yes	A003	R
	Personnel	20 Characters	No	Yes	B7601	R
Salary	Payroll	6 Decimal	No	No	B7601	R
YTD Gross	_____	_____	_____	_____	_____	_____

Notes

- ① Key Yes – An index exists for this field for this Data Set.
 No – No index exists for this field for this Data Set.
- ② Index Yes – A temporary index may be built for this field.
 No – No temporary index may be built for this field.
- ③ R/W/U R – The user may read this field.
 W – The user may create a new occurrence.
 U – The user may update this field.

Figure 44. System Directories (continued)

Field Directory

This directory contains the names of the fields available and their location. Location consists of the data sets in which these fields can be found, and the location of the field within the stored record. The data representation of the data in the stored record, and authorized users are also included. Information indicating which fields can be used for secondary indexes, and the status of these indexes is required.

Exclusive Control Directory

Resource	Use	Owner
Field – Job Title In Stored Record ____ ① In Data Set Personnel	Update	B7601
Data Set – Project	Create	Z9991

Note ① ID of stored record will vary with each record retrieved.

Facilities Directory

Facility	Availability	User
10K Core	Yes	_____
70K Core	No	B7601
180K Core	No	Z9991
2314 Disk Unit	No	
2314 Disk Unit	Yes	B7601
2314 Disk Unit	No	
2314 Disk Unit	Yes	B7601
Service Program A	Yes	Z9991
Service Program B	No	_____

Figure 44. System Directories (continued)

Exclusive Control Directory

It may be necessary during the processing of a user's request for information to limit access of stored records, fields, indexes, data sets, and system services to a single user. This is normally required when information is being updated or altered in any way. If a user requires exclusive control of any facilities, the facility is entered into this directory along with the user's identification. (Note: This differs from the authorization facilities that are contained in the User Directory. The Exclusive Control Directory is used to control interaction between users.)

Facilities Directory

This directory contains the facilities available to the system (main storage, I/O devices, programs, services, etc.), their status, and the user of the facility.

These directories may be wholly or partly contained in either DASD or core storage.

The data management system will then make use of the directories, the user's description of file and logical record, and the data sets to provide the requested information (Figure 45). Two aspects of this system will be examined:

- The events that occur in the system.
- The data flow through the system from data set to user's problem program.

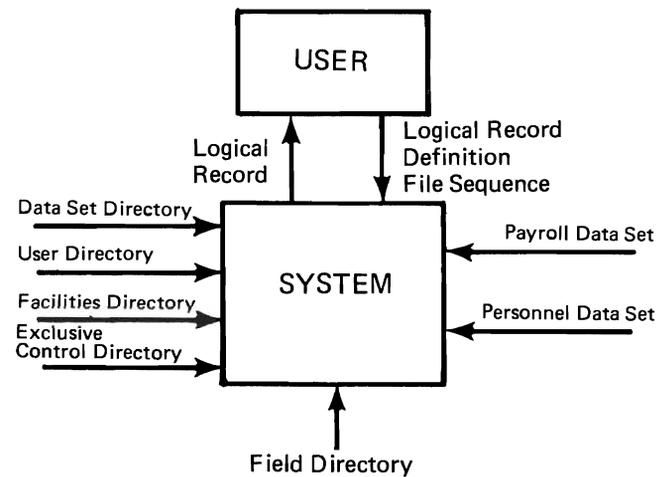


Figure 45. Control Information

Events

The events that occur in the system are made up of the user's request for information, and the system's response to these requests. Figure 46 shows the events that take place and the systems directories that are referenced. The example in Figure 46 represents only a part of the many occurrences in a data system. By examining these events we can isolate certain functional subsystems that compose a Data Management System. Refer to Figure 47. These subsystems are:

- Allocation
- Library System
- Access Method
- I/O Systems
- Field Conversion and Mapping
- Authorization
- Logging and Statistics
- Exclusive Control

The User	The System	Directory Used
Identify himself by submitting his identification code (ID).	<ol style="list-style-type: none"> 1. Verify identification and initiate accounting. 2. Schedule the user for processing 	User
Submit problem program that contains request for a file of logical records and method of processing those records	<ol style="list-style-type: none"> 1. Validate field names <ul style="list-style-type: none"> – Are they valid? – Is the user authorized to use them? 2. Determine data set required 3. Validate user's authorization to use these data sets. 4. Determine facilities required. 5. Validate user's authorization for use of the facility. 6. Allocate facilities. 7. Locate data set. 8. Mount data set 9. Build secondary index of Dept./Man No. for personnel file. 	<p>Field</p> <p>Data Set</p> <p>User problem program and Data Sets User</p> <p>Facility Data Set</p> <p>Field</p>
Get logical record (for update)	<ol style="list-style-type: none"> 1. Locate next entry in secondary index. 2. Look up man no. in Man No. Index for personnel data set. 3. Look up man no. in Man No. Index for salary data set. 4. Obtain exclusive control of the record from personnel file for update. 5. Initiate read of physical records from both data sets. 6. Extract stored records from physical records. 7. Extract fields from stored records. 8. Convert fields to user's data representation. 9. Encode if necessary. 10. Calculate YEARS OF SERVICE from HIRE DATE. 11. Build user record. 12. Return to user. 	<p>Exclusive Control</p> <p>Field User problem program and field User problem program and field</p> <p>Field</p>
Process user record and alter job title		
Put logical record (updated record)	<ol style="list-style-type: none"> 1. Move and convert JOB TITLE to stored record. 2. Rewrite physical block. 3. If JOB TITLE was secondary index, update or note. 4. Log the change for recovery purposes. 5. Drop exclusive control of record. 	<p>Field</p> <p>Exclusive Control</p>
Loop back to get and repeat until end-of-file.		
End program	<ol style="list-style-type: none"> 1. Deallocate facilities. 2. Scratch secondary index. 3. Remove data sets. 4. Calculate account data. 5. Output run statistics to user and log. 	<p>Facilities Field Data Set User</p>

Figure 46. Events that Take Place and Directories Referenced

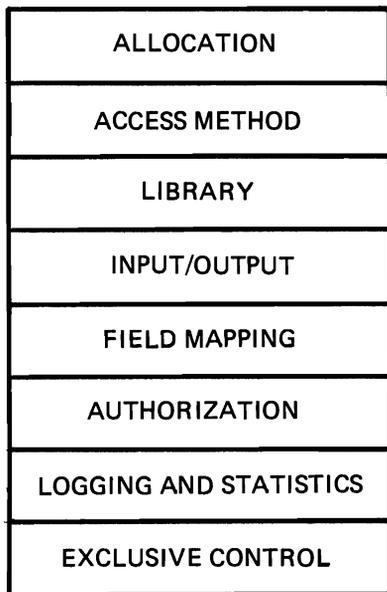


Figure 47. Data Management Subsystems

ALLOCATION: Schedules, allocates, deallocates system facilities (I/O units, DASD Space, Core Storage, Data Sets, Buffers, Program Services, etc.).

LIBRARY SYSTEM: Maintains and accesses system catalogs and directories.

ACCESS METHOD: Performs data organization, which includes:

- Maintaining and creating data set indexes
- Locating physical records and stored records
- Error checking and correction (logical errors)
- Organizing data for output or retrieval

FIELD CONVERSION AND MAPPING: Performs the logical file organizational functions of:

- Extracting and building fields from and into stored records
- Converting the stored record's data representation to the user's data representation
- Encoding and truncation
- Developing virtual data
- Validating field requests and formats

INPUT/OUTPUT SYSTEM: Responsible for the physical transfer of data from I/O device to CPU, channel and device scheduling, and physical device error analysis and recovery.

AUTHORIZATION: Validates user's authority to use system facilities, alter access fields, and to establish accounting and priorities.

LOGGING AND STATISTICS: Logs transactions and updates for recovery and evaluation. Records statistics of system operations and accounting information.

EXCLUSIVE CONTROL: Responsible for preventing data loss or inaccuracy due to simultaneous use of fields, stored records, or data sets by multiple users.

Data Flow

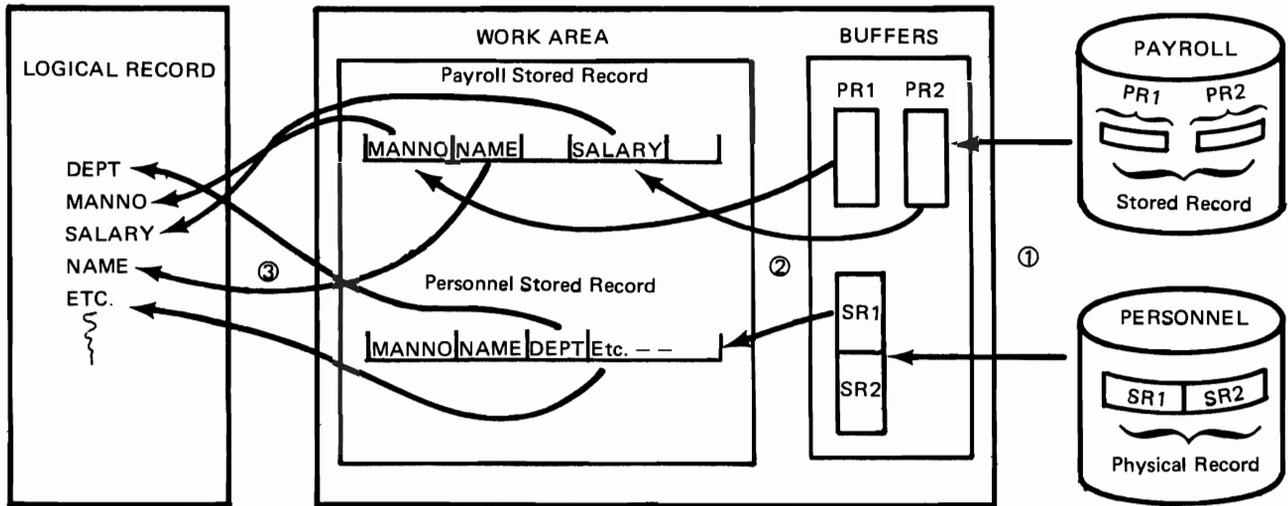
Figure 48 illustrates the data flow from the data set to logical record. It is assumed that the physical records that are required to build the next logical record have been located in the access method. The physical records are read into buffers. The stored records are then extracted from the buffers and moved to a work area. The fields within the stored records that the user requires are extracted, data representations are changed, encoding is performed, and then the fields are moved into the user's logical record.

The interaction of the data management subsystems that accomplish the above data flow is shown in Figure 49. (Note: Not all possible interactions are shown. Allocation and authorization are assumed complete.) It should be restated at this point that the above subsystems are not meant to describe an implementation of a data management system, but rather to illustrate functions of such a system. It is also important to note that these functions are required in any data management system regardless of whether they are provided by the operating system, or the user's problem program.

As the system provides more of these functions, the user can:

- Become less sophisticated in data processing
- Concentrate on application development
- Reduce maintenance costs of problem programs
- Utilize the power of complex data processing systems more efficiently

In the preceding section we have discussed how a data management system can provide information for the user. A system that can provide the functions just mentioned for many users simultaneously is called a Data Base (D/B) System. The next chapter will discuss the design requirements of D/B systems.



- ① Physical blocks are read into system buffers.
- ② Stored records are built or extracted from buffers.
- ③ Fields are extracted, converted (if necessary), and moved to user area.

Figure 48. Data Flow

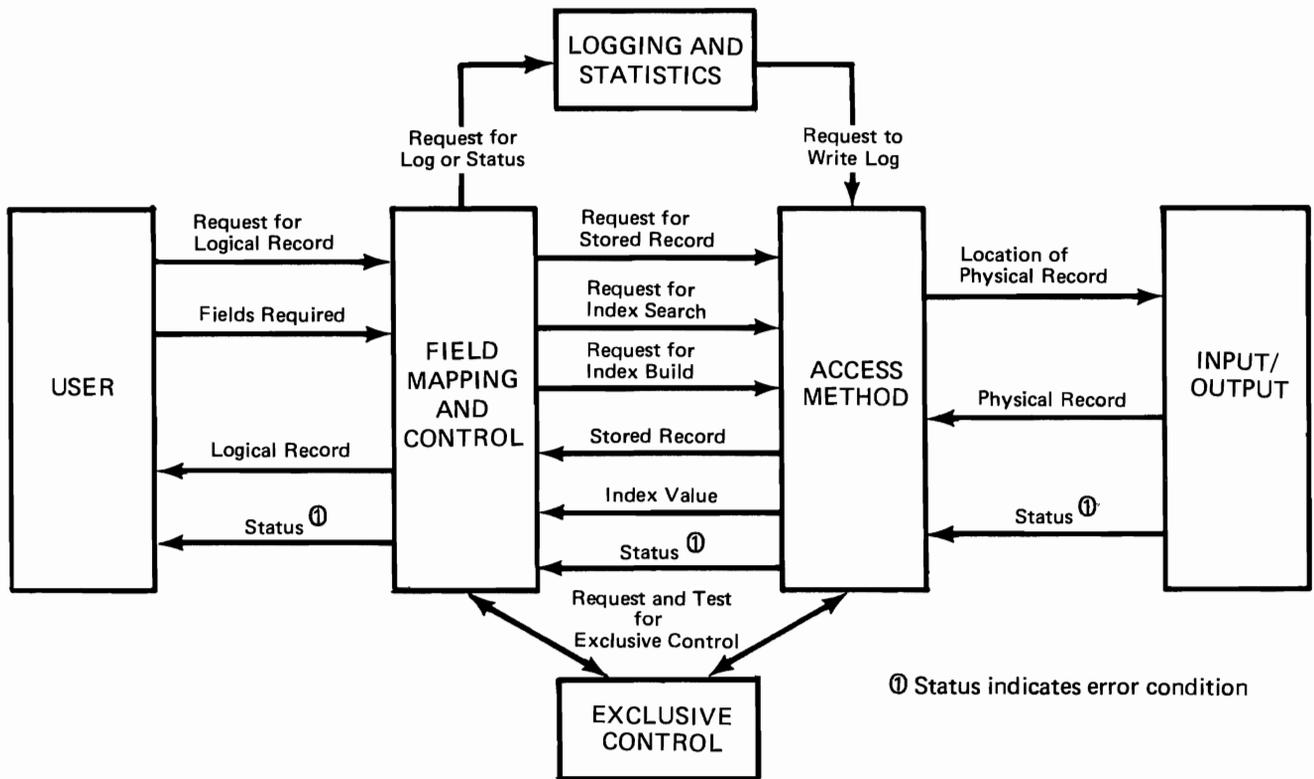


Figure 49. Interaction of Data Management Subsystems

Chapter 6: Data Base System Concepts

INTRODUCTION

In the previous chapter the functions of a data management system were described. If we add to data management a control program to schedule and monitor facilities and a communications control program to handle telecommunications, the result is a data base or information system (see Figure 50).

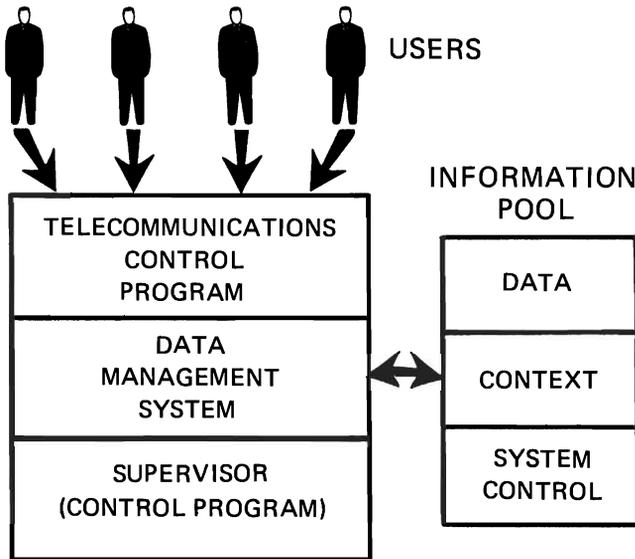


Figure 50. Information System

INFORMATION SYSTEM

An information system is a system that controls, maintains, and provides concurrent access to a pool of information for an identifiable set of users. This pool of information is often called a 'data bank'. The components are:

1. Data base, which contains all the data that can be accessed by users.*
2. Context and data representation for the data in the data base.
3. System control information, such as users authorized to access data, and available facilities.**

An information system possesses many advantages:

- Data Independence
- Data Availability
- Data Control
- Data Consistency

*User, as defined in Chapter 1, can retrieve and alter information but cannot change the system.

**Items 2 and 3 could be contained in system directories as discussed in Chapter 5.

Data Independence

As described in Chapter 5, data independence allows changes in location and data representations of fields, without users being aware of these changes. This can result in significant savings in program maintenance costs as well as allowing application development to proceed without reference to impacting current applications.

Data Availability

Because the system knows the location of all fields and can retrieve them, all the information in the system is available in any combination (if the user is authorized to retrieve it).

Data Control

Another by-product of data independence is centralized control of the location and representation of fields. The user has no power to alter the operation of the system or to change and retrieve data unless authorized to do so.

Data Consistency

Access to data can be limited to those users capable of using it correctly. Because the system processes each field, it can also check to see if the value of the field is valid and reasonable.*

We will now discuss the design criteria for an information system that can offer the above advantages.

DESIGN CRITERIA OF A DATA BASE SYSTEM

In Chapter 5, we defined the requirements of the data management system that are necessary to retrieve information for a single user. Many additional features are necessary if the information system is to prove useful to an organization consisting of many users who simultaneously require access to the information. In addition, these users may require many different levels of service, from simple inquiries taking seconds, to complex file creations and searches taking hours. Users can include programmers, accounting department clerks, executives, department heads, etc., each requiring unique types of service. This environment imposes severe design requirements on an information system. These criteria are in addition to those given in Chapter 5. They are:

- Security and integrity of data
- System availability and recovery

*While the system can provide reasonableness checks, it cannot be responsible for the absolute data value.

- Concurrent access to data
- Expandability
- Dynamic reconfiguration and control
- Test/Debug facility
- Multi-level function
- Controllable response time

Security and Integrity of Data

It is extremely important that sensitive data be protected against inadvertent destruction (data integrity), and unauthorized access (data security). Payroll data about an individual should be retrieved only by a select set of users (Payroll Department, Individual's Manager). Bill of Material (parts structures) data, while it may be available for anyone to see, should be protected against accidental loss or inaccuracy. Equally as important as prevention, is the detection and correction of events that lead to violations of security and integrity.

System Availability and Recovery

Because an information system contains critical information necessary to run an organization, it should be available for use a large percentage of the time. A system that handles customer inquiries or reservations may require 100% availability, while an in-house inventory status system may tolerate short periods of unavailability. Another aspect of availability is operation with a degradation of function or performance. In a customer information system for example, failure of some part of the system may cause answers to inquiries to be delayed by several seconds, or allow only certain types of inquiries to be made.

When some part of an information system fails it is necessary for the system to have recovery procedures. When an error or failure is detected, this procedure will determine the corrective action required, and initiate this action. For example, if a device that contains part of the data base fails, the recovery system should make another device available and ask the operator to move the volume. Recovery should be undertaken if possible without reducing system availability, or, at worst, with a reduction of function.

Concurrent Access to Data

Many users will require concurrent access to the same data sets and fields. This implies that the system will need to protect a user from the actions of all other users, without seriously affecting performance. This capability is most important when concurrent updating is to be provided.

Expandability

An information system should allow functions and applications to be added to the system without affecting

current applications. For example, an executive query language or a new set of inventory control programs can be added to the system without affecting the already running payroll and accounts receivable systems. The data can also be expanded to include more information.

Dynamic Reconfiguration and Control

The load on an information system will vary from hour to hour, and day to day. This load can involve both number of users and type of functions, or data required. This situation, coupled with the requirement for high availability, dictates a capability of dynamic reconfiguration. Dynamic reconfiguration is the ability of the system to alter itself to meet current loads or changes in configuration. The system does not necessarily have to alter itself, but it must at least provide statistics that indicate when a change is necessary and allow this change to be made without influencing system availability.

Test/Debug Facilities

An information system can offer powerful testing facilities due to its ability to build logical records of any form. However, it must guard against exposure of sensitive data to violations of security or integrity due to program failure or malicious programmers. It should also allow test data to be entered in the system and to be used as if it were real data when a program being tested is encountered.

Multi-level Functions

An information system must be able to provide response time and functions consistent with the nature of the user. An unanticipated, unstructured request from an executive for statistical information does not normally require fast response, but may tie up many system facilities. On the other hand, an anticipated* request from a phone operator answering a customer inquiry requires very fast response (1-2 seconds).

Controllable Response Time

Response time can be controlled by restructuring the data base, altering the ways in which users compete for facilities, or inserting time delays. All these methods are required to effectively regulate the use of the information system, and to provide satisfactory service. For example, the response to an inquiry may vary from 1-6 seconds, depending on the load. However, to mask the load or to provide consistent response, a time delay is added to 1-, 2-, or 3-second responses so that the range of response times appears to the user as 4-6 seconds. Changing response times could also be used to encourage or discourage use of certain facilities.

*Anticipated in this context does not mean "when", but rather that this type of transaction occurs frequently and requires predetermined facilities.

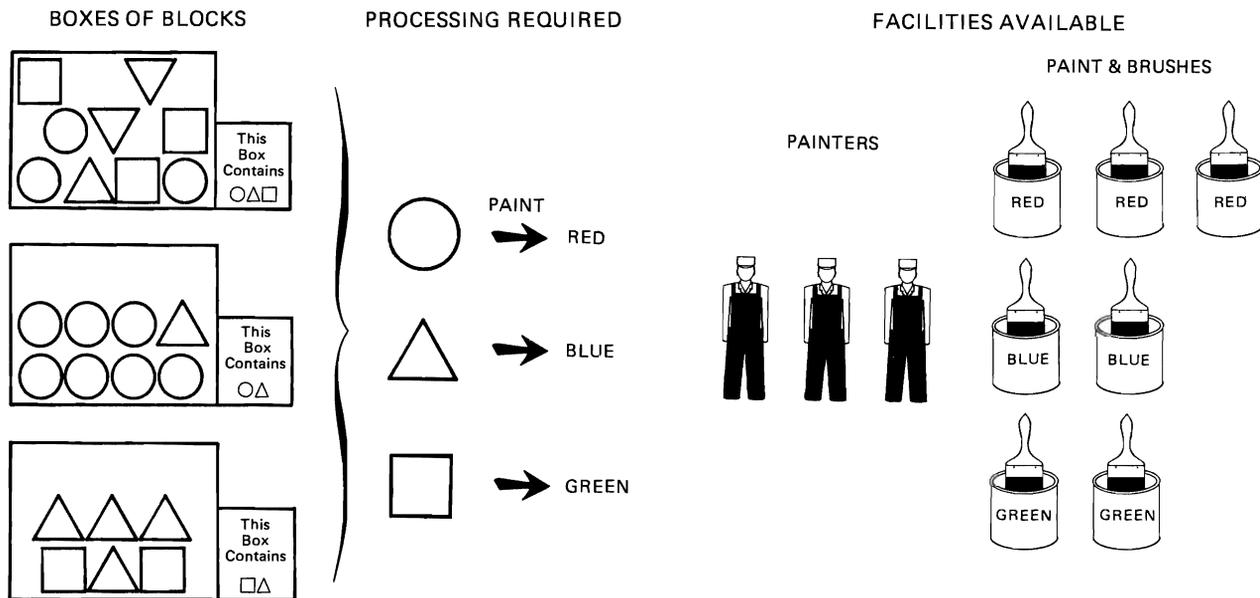


Figure 51. Job Step System

Techniques for implementing these design criteria are complex and require careful system design and are therefore beyond the scope of this publication.

EVENT-DRIVEN SYSTEMS

In current operating system design (System/360 Operating System; System/360 Disk Operating System) it is assumed that:

1. Each job will require a large portion of the system facilities.
2. The required facilities will be dedicated to the job while that job exists.
3. Each job will run for a long time (minutes).
4. The facilities of the system will be shared with only a few other jobs.

The unit of work in such a system is a job step.

In an information system, the unit of work is the event or transaction. Events:

1. Require only a small portion of system facilities.
2. Are of short duration (seconds).
3. Can share system facilities concurrently (as many as 1000).
4. Do not have facilities dedicated to them.

To better illustrate the difference between a job step system and an event system, let's look at an example. Figure 51 shows a set of boxes, each containing a number of wooden blocks of different shapes (circular, triangular, square). These

shapes are to be painted by a painter (circular-red, square-blue, triangular-green). Several cans of paint for each color are available. The boxes are marked indicating what shapes are in the box. The painter, however, selects the blocks at random. The flow of this process is shown in Figure 52. In this example each box represents a job step, which is composed of a number of different tasks (events). The events that will be performed are known, but the order in which they will be done is not. The facilities (painter and paint) are allocated to job step (boxes) and dedicated to that step until all events (blocks) are complete. Figures 51 and 52 therefore illustrate a conventional job-step system.

Figure 53 represents an event-oriented system. The blocks (events) are independent and the facilities are not dedicated, but shared. A conventional job step can then be viewed as a collection of events for which all resources are allocated for all events within the step prior to execution. Because the order in which the events are to be executed are not known to the system, the resources must be held for the duration of the step. The event system treats events as separate entities and allocates and deallocates resources as required.

DATA INDEPENDENCE

Data independence is a feature of data base systems that allows the user's definition of his records, files, and processing to be unaffected by changes to data organization, data representation, devices, etc. There are two aspects of data independence that require definition:

- Binding time
- Degree of independence

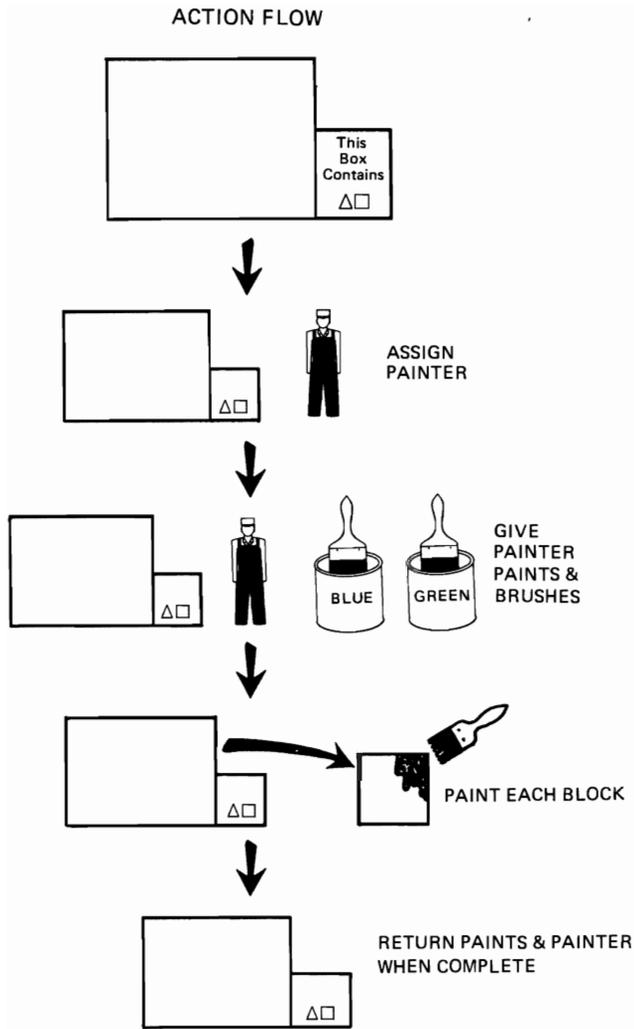


Figure 52. Job Step System

Binding Time

As described in Chapter 5, one function of data management is to map the data element into the user's field in his logical record. The time that this mapping occurs is called the binding time. When binding has occurred, the user is no longer independent of changes in the data base.

Binding can be performed at different times for different data attributes. For example, the name and location of the data set containing the data may be bound at job initiation time, while the location of the data element and the data representation may be bound when the users issue a request to data management to retrieve a logical record. In most conventional batch data processing systems, the location of a data element within a stored record and the representation of that data element must be mapped when the user-program is written. Changes in format or representation of a stored record must be accompanied by a rewrite and recompilation of all programs using those stored records. If binding is not

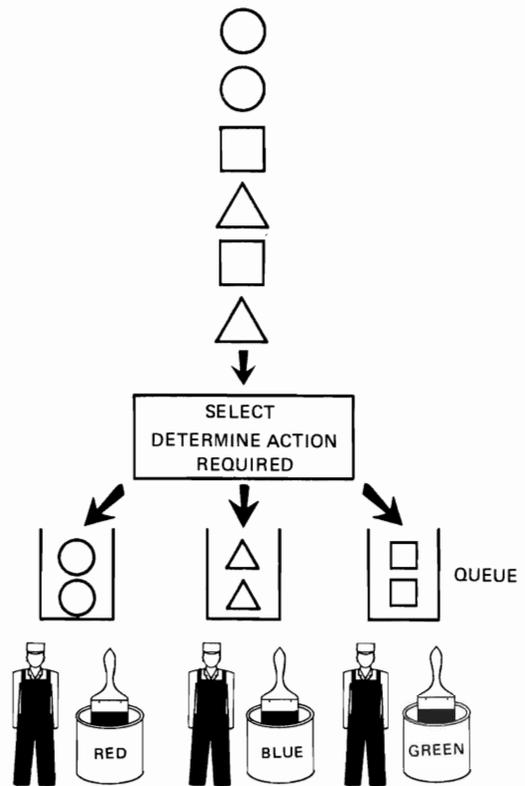


Figure 53. Event System

performed until the user issues a request for information, changes in the data base can occur between requests without influencing the user's program. The latter method is described in Chapter 5.

Degree of Independence

The other aspect of data independence is the amount of information the user must know about the data elements he requests for his fields. Some different degrees of independence (from high to low) are:

1. User must know only the name of the data element (see chapter 5).
2. User must know name and data representation.
3. User must know name, data representation, and data set.
4. User must know level 3 plus data organization technique.
5. User must know level 4 plus physical blocking and device characteristics.

Binding time and degree can be used to evaluate the type of data independence a data base system possesses.

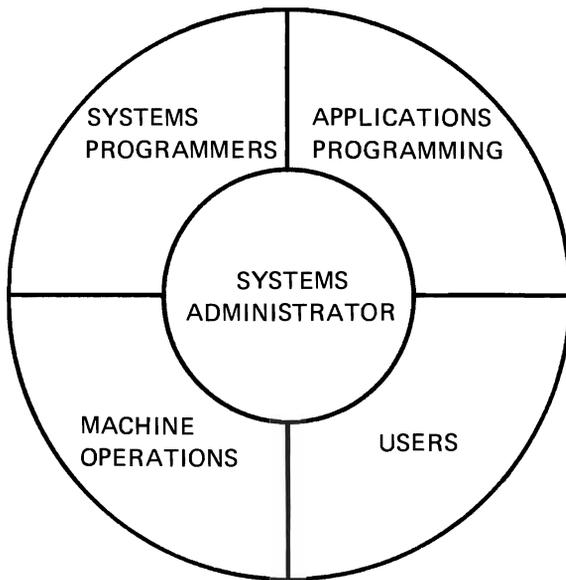


Figure 54. System Administrator: Relationships with System Users

THE SYSTEM ADMINISTRATOR

Any system with the power and potential benefits of a data base requires careful control in planning and day-to-day operation. This control function is provided by the system administrator. The system administrator is an individual or organization responsible for:

1. The economics of the system.
2. Levels of service provided to the users.
3. Custody of the data base and resources of the system.

All use of the system is coordinated and audited by the system administrator (see Figure 54). It is essential that this position be created and filled with competent individuals whenever a data base system is installed.

Index

- Access method 43, 45
- Access time 18, 19
- Addressability 18
 - addressing 18
 - resolution 18
- Allocation 43, 45
- Arithmetic data 5
- Associative addressing 18
- Attaching devices to systems 19
- Authorization 43, 45

- Binding time 49, 50
- Buffers 45

- Calculation 23, 24
- Capacity 18
- Chain 24
- Channel 19
- Character string 14
- Commands 19
- Compression 5, 6, 7
- Concatenated key 8, 9
- Concurrent access to data 48
- Context 1, 2, 38, 47
- Control 1, 40
- Controllable response time 48
- Control of devices 19
- Control unit 19, 20

- Data 1, 2, 3
- Data availability 47
- Data bank 47
- Data base 47
- Data base system 49, 51
- Data cell 18, 19
- Data consistency 47
- Data control 47
- Data elements 21
- Data independence 38, 40, 47, 49, 50
- Data management
 - definition of 1
 - functions of 3, 35, 40, 50
- Data organization 21, 35, 38, 49
- Data representation 1, 5, 6, 38, 47, 49
- Data set 21
- Data set directory 43
- Data storage devices 18
- Data structures 8, 27
 - simple 8, 10, 27
 - hierarchy 10, 27
 - network 10, 27
- Data transfer rate 19

- Direct access 18
- Direct address 23
- Directories 40, 41, 42, 43
- Direct organization 23, 24, 28
- Disk and drum devices 19
- Degree of independence 50
- Delimiters 14
- Delimiting fields 14
- Dynamic reconfiguration and control 48

- Encoding 5, 6, 7, 36, 45
- Entities 1, 21
- Event 43, 44, 49
- Event-driven systems 49
- Event-oriented system 49
- Exclusive control directory 43
- Expandability 48
- Explicit length 16
- External storage 18

- Facilities directory 43
- Field 5, 42
- Field conversion and mapping 47, 49
- Field directory 42
- Field mapping 36
- File 3, 38
- File organization 35
- Fixed-length record 8, 13, 14, 46

- Hardware address 23
- Hashing 23, 24
- Hierarchical data structures 30
- Hierarchy 11, 13, 16
- Higher level index 27

- Index 23
- Indexed sequential organization 28
- Index search 23
- Information 1, 5
- Information attributes 3
- Information system 1, 38, 47
- Input/Output devices 18
- Input/Output system 47, 49
- Instructions 19
- Inverted list 26
- Inverted list organization 26, 27
- Implicit length 14

- Job step 49

- Key 8, 10, 21, 41
 - concatenated 8, 9
 - truncated 8, 9

- Latency 19
- List 14, 16
- List organization 26
- List processing 14
- Locating fields 13
 - by relative location 13
 - by embedded identification 13, 14
 - by pointers and lists 13, 14, 15
- Logical data structures 8, 16
- Logical record 3, 5, 21, 35
- Logical records, designing 6, 7
- Logical record identification 8

- Mapping 35
 - field 36
 - record sequence 36
- Multi-level functions 48

- Network 10, 13, 16
- Network data structure 32

- Orders 20
- Organizing fields within records 13
- Overflow area 28
- Overrun 19

- Physical block 21, 35
- Pointer 14, 26, 30
- Pointer field 14

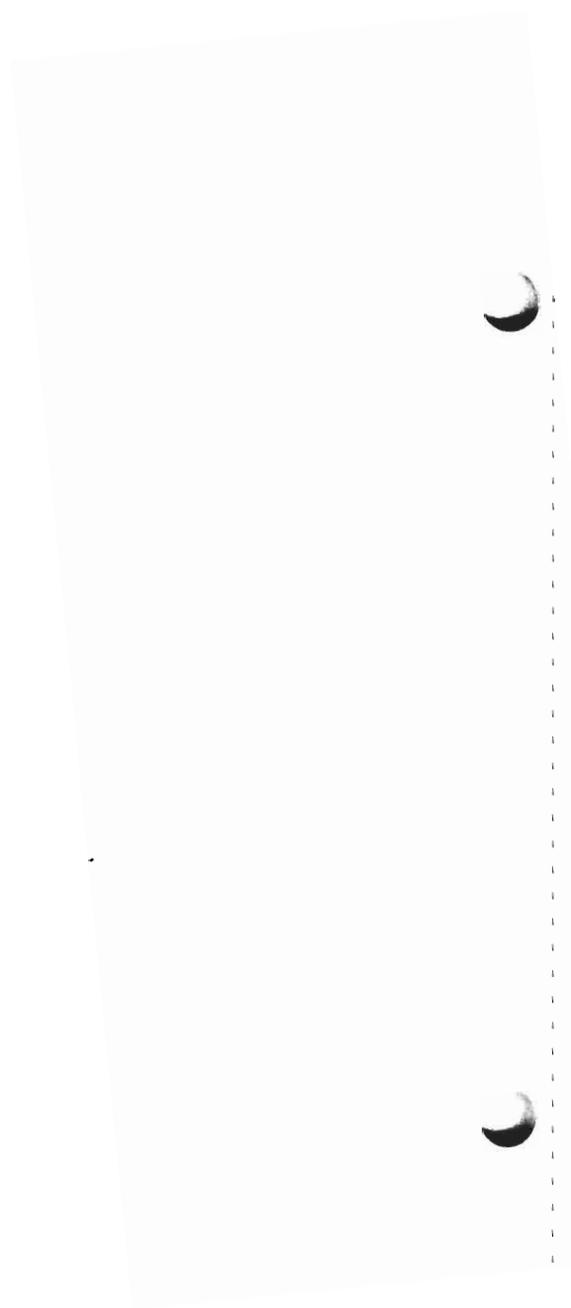
- Record sequence mapping 36
- Relative position 16
- Representing data structures 16
- Retrieval 1, 40
- Ring 26
- Ring organization 26, 32, 34

- Secondary index 27
- Security of data 47, 48
- Segments 10, 11
- Sequential access 18
- Sequential organization 21, 22, 24, 28
- Simple data structures 8, 10, 27
- Simple list organization 26
- Sparse indexing 28
- Storage 1, 40
- Storage device attributes 18
 - capacity 18
 - addressability 18
 - access time 18
- Stored record 21, 23, 28, 35
 - updating in sequential organization 21, 26
- String data 5
- Synonym 24
- System administrator 51
- System availability and recovery 47, 48

- Test/Debug facilities 48
- Transaction 51
- Trees 10
- Truncated key 8, 9
- Truncation 36, 45

- User directory 41, 43
- User information 3
- User of information system 1

- Variable-length record 8, 13, 14, 30
- Virtual data 6, 45
- Volume 19, 21





International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
(U.S.A. only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)

GC20-8096-1