WHAT IS A DATABASE

A tutorial
ACM 72
Boston, Ma.

John K. Lyon
Honeywell Information Systems
Waltham, Ma.*

---

* - Author's current address: 200 Smith Street, Waltham, MA 02154

DATABASE ORGANIZATION AND STRUCTURE

I want to emphasize that the database does exist, and that it exists now! It does not depend upon a declaration of database existence by a manager of information systems or by a company president. It exists as a direct consequence of the existence of a business enterprise. It may or may not be in a convenient form but it does exist.

In most enterprises we can expect to find the obvious portions of the database represented in machine-processable form; i.e., tape, disc, punch cards, etc. While it is less obvious, the man-processable data within the enterprise is also a very real part of the database. This includes drawings, reports (including all those stacks of computer output), worksheets, the hidden notebooks in the lower left-hand drawers of our desks, and the little tidbits that we have stashed away in our minds.

The subject of today's talk is limited to machine-processable database elements and how they can be structured and organized in order to make them accessible and useable. We will discuss the definitions of 'database' and 'information'; methods of organizing data; and some of the problems that face us as an industry.

DEFINITIONS

In order to gain some feeling for the meaning of 'database' and 'information', it is convenient to turn to Webster and examine the meaning of data, base, and 'information'. First, base is defined as the foundation; i.e., something to build upon. In our case, it is the processes and procedures which create information, which are built upon the foundation. It is important to note that the foundation comes first, followed by the processes. Secondly, a base is

defined as the fundamental part. The database is the heart of our information processing systems and not simply a by-product of processing. The base is the chief ingredient in a manner similar to that just mentioned; i.e., it is the heart of the system not an adjunct. The last two definitions are extremely important; the base is the start. The base of data that we are discussing is the starting point, the source, for all of our information processes. Finally, the base serves as a stabilizer. A database that has been properly structured and organized, and which actually represents and contains that data of an enterprise, will stabilize the information processing systems. One of the most important factors leading to system obsolescence is the absence of data required to support modifications of the information process. This absence may be real or imagined; it is either too inconvenient or too expensive to get it; or it may never have been collected.

The other component is data which is defined as something assumed or real and used as the basis of reckoning. It does not have to be real. One of our temptations is to restrict the database to only those things which we can observe and measure directly. The most likely reason for this attitude is that we don't know how to use or control the assumed data. It is, however, readily apparent that assumed data plays a significant role in the decision processes of an enterprise and should, therefore, be reflected in the database. Data is something which is used. The point is simply that we do not want to collect data merely for the sake of collection but because it is pertinent and relevant in terms of our enterprise. It is sufficient for me to maintain data representing my employees; I have no desire to collect data covering everybody's employees. Now that I have restricted data to that which can be used, I must determine what I am going to use it for.

The use of data is reckoning, the generation of information which will in turn be used to manage the enterprise. The generation of information is the only reason for having a database.

Finally, Webster defines information as derived knowledge based on observation of unorganized facts or data. The point is, we use data to generate information. The purpose of structure in a database is to facilitate the organization of the data into meaningful groupings which in turn aid in the generation of information.

STRUCTURE

A database has structure which exists in three forms. It has a perceived structure, which is the way that each of us looks at the database. The fact that each of us looks at it in our own way is a reflection of the individuality of our experiences, backgrounds, and the responsibilities which we hold within the enterprise. The accountant may look at the business in terms of his chart of accounts. He knows that certain accounts have a significance in terms of the business and the way in which they are related to other accounts. This is a perceived structure. In a military environment, the organization chart which can be found on almost every wall is a reflection of a perceived structure. The parts-list and where-used relationships are a perceived structure of an enterprise's database as seen by the manufacturing functions.

The physical structure of the database relates to the manner in which data is physically represented within the database in terms of sort sequence, record and field sizes, and its existence on one or more master files. The danger

with thinking of the database in terms of its physical structure lies in confusing those physical properties (necessitated by devices) with the character of the business. I think that we all have seen instances in which the significance of codes in a particular sort sequence has dominated the design of files and applications to the point that the number of exceptions makes new applications, or modification to current applications, extremely difficult. This is an example in which the physical characteristics of the file (sort sequence) has hidden the important logical characteristics (relationships) of the enterprise.

Thus far we have alluded to the existence of a logical structure of the database. The logical structure describes the way things really exist within the environment. The entities, i.e., the things within the enterprise, really exist; they have properties which distinguish them from all the other things within the business; and the relationships which exist among the entities. Harry just gave us some examples of relationships in his talk.

The function of the designer is to define a logical structure which is sufficient to reflect the perceived structure of its users, and then to map it into a physical structure that will support the information processes. If we can integrate these three structures, we can produce a machine-processable database that adapts to change, enforces system discipline, reduces redundant data, and encourages generation of timely and selective information. The vast majority of change which takes place within the information requirements of an enterprise are those which reflect the changing perceptions of the database. By the representation of a valid logical structure, the database is prepared to act as the stabilizer. In other words, if the database is

structured to represent the business and not a given process, changes in process requirements tend to affect only portion of programs and will not cause a rippling of changes to programs because of alterations in the file format. Because the environment is reflected in the structure of the database, the establishment of valid relationships is not left to the whim and fancy of the programmer to maintain. This is not meant to imply that the programmer intentionally attempts to degrade the system but rather that he may not be aware of the importance or significance of the relationships, and further that he is under strong pressure to produce which in turn encourages "expediency".

Because the database reflects the logical structure of the enterprise, the generation of selective information in a timely fashion is encouraged. If we examine the requirements for selective information, we will note that it is basically a reflection of someone's perceived structure.

FIGURE 1

In order to lecture intelligently in the field of data processing, you must have a pyramid. However, we are going to look at this pyramid in two ways. In the conventional manner, we look at the information requirements of the man at the bottom of the heap and we see a requirement for rather fine detail, but restricted to a very narrow field of interest. As we rise up the organizational levels of the enterprise, the requirements shift to one of broader field emphasis but less detail. Finally as we rise to the top, the level of detail must of necessity diminsih still more. Regardless of level, however, the information is most useful if presented within the perceived structure of the recipient. This is a very important point: information reflects

perceived structure, not data. Too many systems have been doomed to failure from the outset because the organization of the database explicitly followed the perceived structure. When there is personal shift at the top, the need for information is most critical. However this is the time when it is of the least value if it is structured to reflect the predecessor's perceived structure. By the time the database is reoriented, there is another replacement. The only positive aspect of structuring the database along the perceived lines is that programmers are assured of continuing employment.

If we look at the pyramid from the top down, we note that the man at the top sees a very broad spectrum of information. His view of the company must encompass all aspects of the business as opposed to the more provincial views of the lower levels. The database structure must permit the generation of information which satisfies this requirement for the differing views of the business. The point of this discussion is to emphasize that the database does have structure and that if we take advantage of the logical structure, we will be capable of satisfying the information demands (perceived structure) of our users. Because we are attempting to satisfy the needs of more than one level of user, we must be capable of cutting across functional and organizational boundaries. This leads to integration of data and to the necessity for accessing the same data by more than a single method; that is, we must be able to establish more than one relationship between pieces of data.

DATA MANAGEMENT SYSTEM ELEMENTS

With current technology, we can create physical structures to support multiple access techniques. This portion of this tutorial is concerned with the various techniques and their characteristics. The basic purpose of any

data management system, regardless of how complex it may be, is to find the right piece of data at the right time. Most data management systems will also provide other functions, such as: protect data; purge obsolete data; reorganize it; recovery; analysis; etc. However nice and necessary these other functions may be, the fact remains that the basic function is to find one piece of data which exists among all of the other data and to do it correctly.

FIGURE 2

A data management system is built upon one or more of the following techniques: sequential, random, or list. There are systems in existence which are built totally upon random organization. There are many which are totally sequential. However, there are many more which are built upon combinations of these.

FIGURE 3

The simplest form of organization, (Figure 3), and probably the most familiar, is the sequential structure. The basic characteristic of a sequential file is a record by record search until I find the desired data or until I decide that it does not exist. To facilitate the decision to quit, most sequential files are ordered in some fashion. Ordering not only helps in deciding that the record does not exist, but it permits me to process many transactions at the same time without starting at the beginning each time. However, if I wish to locate a piece of data based on anything other than the sequence key, I must either reorganize the file or perform an exhaustive search of the file.

If we examine the reasons for using a sequential file we are likely to discover that the files are too big to resequence, that our current hardware

configuration is incapable of supporting any other technique, the cost associated with rewriting systems is considered to be too great, or it's the only way I know how to do it. A sequential process is always appropriate when I can expect a high hit ratio; i.e., when I will need to look at most of the records in the file.

RANDOM ORGANIZATION

There are three basic sub-techniques associated with random files:

- Direct Address
- Dictionary Lookup
- Calculation

FIGURE 4

The direct address (Figure 4) is by far the simplest: I remember where all pieces of data are stored and when I need a particular data element, I give the address to the system. In practice, I need only remember the address of those records which are important to me and the program.

FIGURE 5

Now if I would like to construct an aid to remembering addresses, I might turn to the use of a table (as in Figure 5) in which I stored the key value of a record and the address at which the record is located; that is, a dictionary. The method of organizing the dictionary could then be any one of the three techniques. A dictionary is of value in those situations in which I am trying to remember some, but not all, addresses. If total entry (putting all addresses) in the dictionary is used, a search of the dictionary may take as long as a search of all records.

FIGURE 6

Finally, there is calculation. Very simply and basically, (as shown in Figure 6)

calculation is the application of an algorithm (formula) to the values supplied in the key to compute a number. This number is then mapped to whatever physical space I have available. However, because of the mapping, two different key values may yield the same address. When two or more records want to go to the same address, the first record mapped into the calculated space will contain a pointer with the direct address of its successor(s) in the overflow area. Such pointers serve to link the records together and are known as chains or lists.

FIGURE 7

There are three kinds of lists: simple, inverted, and ring lists. In a simple list (Figure 7) the records will be found in sequence. Each record in the list contains a pointer (address) to the next record. In order to be useful, I have to know how to find the start of the list. This can be by dictionary lookup, by hash coding, or by some other technique. The last pointer in the simple list signals, "End of the list. Stop. Do not go any further."

This is a good time to emphasize that while a list may be in logical sequence, it is not necessarily in physical sequence in terms of physical addressing of the record. That is, the first (logical) record has the lowest key the second record the next highest key, the third record the third highest key, and so on. In a list, one record points to the next record wherever it is physically stored. In a sequential file, such as you normally find on tape, the next record is that physically next in storage.

FIGURE 8

An inverted list is really a search technique, with some sequential file processing. As long as I process a file in sequence, I know when to stop looking. However, when I process a file out of sequence, that is when I process a file

for some other key or value than that on which it is sequenced, I must either search the whole file or sort it (resort it) on the field value(s) in which I am now interested. Inversion is a technique in which I maintain the addresses of all the records that share the same key value or property. As shown in Figure 8, I do this through an index in which 1) I list the properties in which I am interested and 2) associate with each entry the addresses of the records in which those properties exist. This technique quickly identifies records which possess those properties in which I am interested and provides a rapid search technique. In the last several years, we have seen the inverted file technique used, mostly on retrieval systems, in situation where people have to find things by more than one category and in which they want quickly to restrict the area of the file to be searched. Many of the projects for library research by subject are built on inverted files.

FIGURE 9

The ring structures grew out of graphics. To illustrate why rings were necessary in graphics assume that I am at a point on a surface and want to be able to move in any of the three dimensions. In graphics, it is convenient to describe edges using lists; and it become evident that by some method (devious, arcane, or innocent) I can get to the middle of a simple list. With a simple list I can only process to the end. I could continue processing back to the top of the list if the last record of the list pointed to the first, thus the term ring. Because the "bottom" of the list points to the top, I can process from any point of entry through all elements and back to that point of entry (Figure 9).

There are some other terms that come up when we talk about lists: Prior

processable (symmetric) and headed. To be prior processable each entry contains the address of its predecessor in that list; that is, I can go both ways. When we speak of a headed list, we mean that each record contains the address of the record that heads that list. At any time I can get directly from my current position in the list and return to 'home base'.

COMPLEX STRUCTURES

Having come this far, it is time to examine the more complex structures. These are: index sequential, trees, and networks. Rather than being frightened by the word 'complex', remember that all complex structures are built upon one or more of the simple methods of accessing. As long as we keep those simple methods in mind, the complex structures are quite understandable.

FIGURE 10

INDEXED SEQUENTIAL

In an indexed sequential structure (Figure 10) I need to process randomly or in sequence. For example in a payroll or a personnel file, I would like to be able to process the file randomly; that is, given a social security number, I would like to be able to quickly find the record of the person. I would also like to process that file in a sequential fashion, either because there is intelligence in the way the numbers and the keys are assigned and I want to preserve that intelligence, or because I want to look at all the records in sequence to prepare a payroll. How do I retain the sequence and also get the ability for random processing? I have two ways available:

1) I can index every record. That is, I can build a dictionary for every record in my file. Based upon the key, this dictionary will tell me where I put any record.

2) I can index groups of records within the file. To do this, I arbitrarily establish sub-ranges of the key value within the total possible range of that value within the file. Each index entry points to the storage area that contains those records whose key values fall within the sub-range of the index entry. For example, the first index entry might cover a sub-range of 000 through 099, the second the sub-range from 100 through 199, and so on through the last index entry that covers the sub-range 900 through 999.

FIGURE 11

TREES

In a tree structure, (Figure 11) I access records based upon what we may call a hierarchy of keys. That is, the key at the highest level provides one differentiation between the records in the file. The next lower level of keys provides another differentiation, and so through the lowest level of keys until I get the location of no record, one record, or more than one record. For example, say I have a personnel file and I want to access it on the basis of job title and age of employees. The hierarchy of keys I established is "job title" as the highest and "age" as the next lower key. Somewhere in storage are the records I have to access. Say I provide the key values: Job Title = Programmer; Age = 25. The first key value takes me from what can be assumed to be the root of the tree (all employees) to the branch devoted to programmers. The second key leads me down that branch to a sub-branch (twig ?). It is possible that there is no employee who meets the key criteria or I may find more than one employee who is, for example, a programmer and is

twenty-five years old. Note that in my personnel tree structure, I must make the differentiation on the first key before I can make a differentiation on the second key. I cannot directly enter the tree and ask for all employees aged twenty-five regardless of job title, but rather I would have to search all job trees for age twenty-five.

FIGURE 12

NETWORK

A network structure (Figure 12), is essentially composed of two or more tree structures, each organized in a different fashion, tied together so that I can get from one tree to another. For example, using the same personnel file, I construct two trees: One tree is organized by age, call it the age-group tree; the other is organized by title, call it the job-code tree. Now I can ask for the location of the records of those employees who are programmers, those who are twenty-five, those who are programmers or are twenty-five, those who are programmers and twenty-five. This is a network which resulted from cross-indexing the file.

A network structure permits the reduction of redundant data. This is important in reducing storage cost but more importantly as a means of insuring complete maintenance. Incomplete maintenance of redundant data occurs whem some of the duplications are not updated at the same time (or near enough to the same time so that data integrity is more than a wistful hope). If I have correctly cross-indexed no record needs to exist more than once in tne database.

In conclusion, let us get back to a basic point that is too often overlooked. No matter what the data technique; no matter how complex that technique; and no matter how sophisticated or devious we are in implementing that technique;

all that we are attempting to do is to determine the address of a record. Each increment of complexity moves the user, or the programmer, away from that basic task. As I move from a sequential file, through a dictionary, and on through the other techniques, into the network structure, I am presented with processing that seems more and more automatic. Regardless of how automated the processing, I am still trying to determine the address of a record. The question of the degree of automatic processing in data management systems is one of the most crucial issues of the database community.

FIGURE 1. Breadth and depth of information varies by position in the corporate hierarchy.

FIGURE 2. Data Management Systems are built upon simple techniques.

FIGURE 3. A sequential organization requires a record by record search.

FIGURE 4. Direct access method retrieves a record based on a supplied address.

FIGURE 5. A dictionary is used to determine direct addresses.

FIGURE 6. Calculation is used to convert a key to an address.

FIGURE 7. Within a list, a logical sequence may be maintained without regard to physical sequence.

FIGURE 8. Organization of an inverted file.

FIGURE 9. The ring permits unlimited list processing.

FIGURE 10. Indices are used to shorten sequential searches.

FIGURE 11. Tree structures require a level by level search.

FIGURE 12. Networks result from cross-indexing.

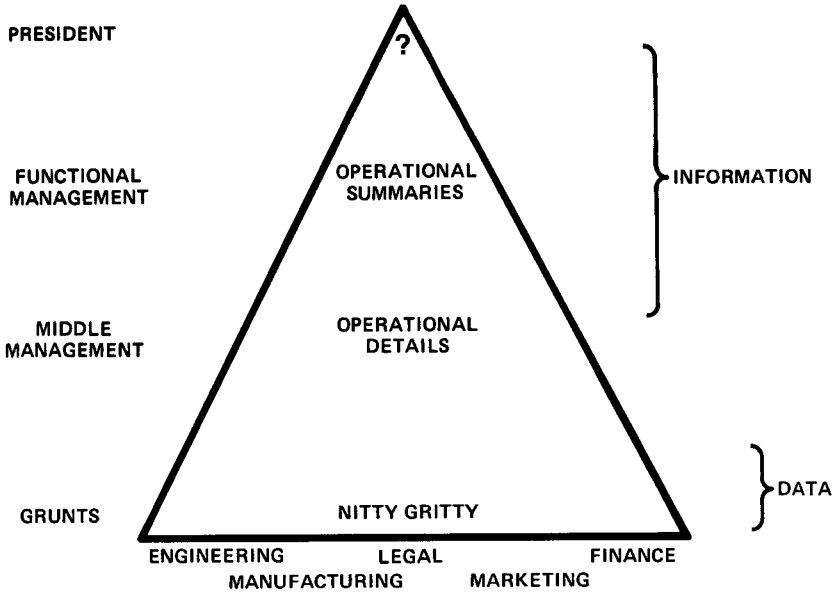

Figure 1

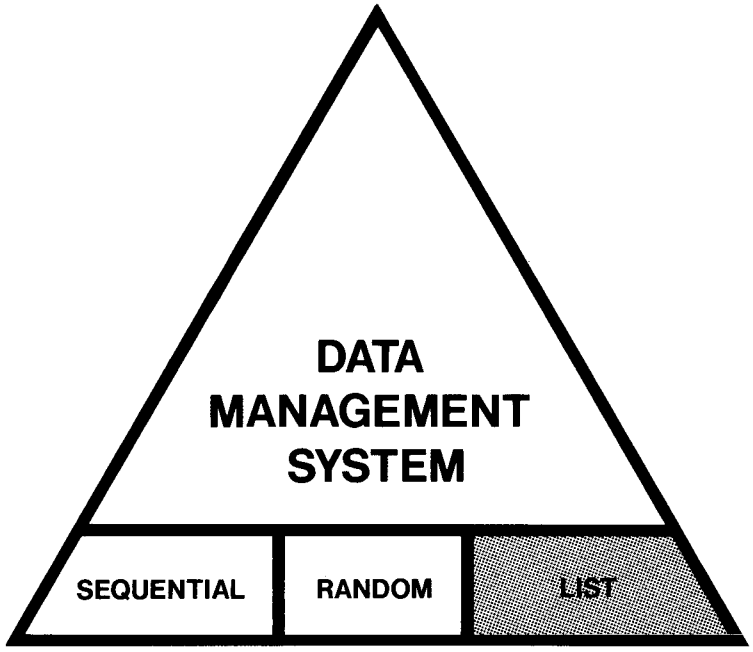

Figure 2

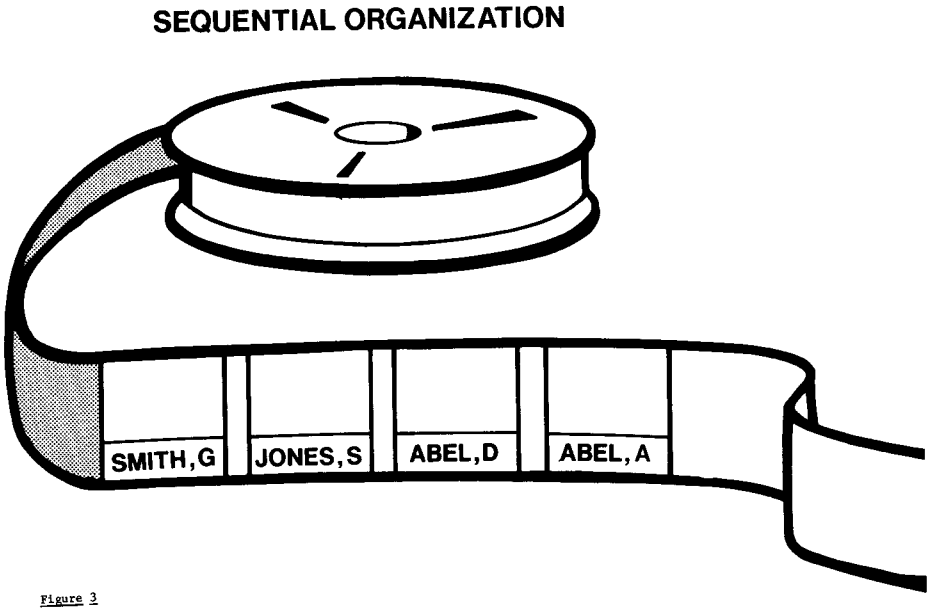

Figure 3

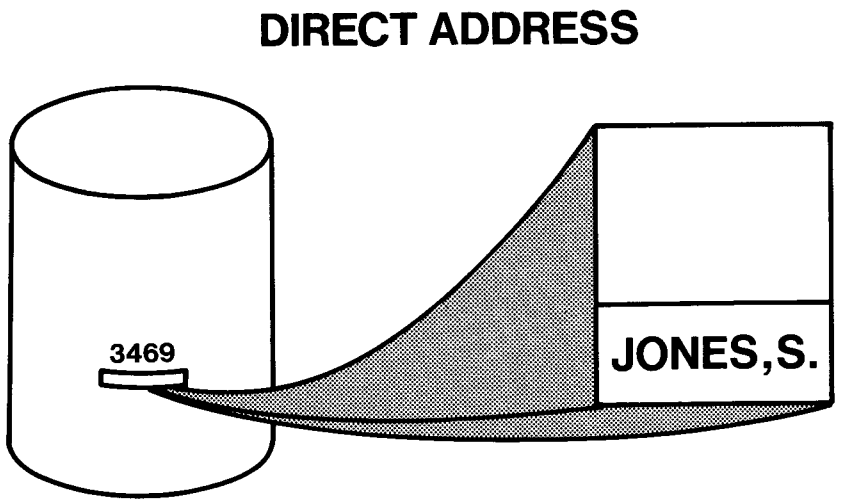

Figure 4

DICTIONARY LOOKUP

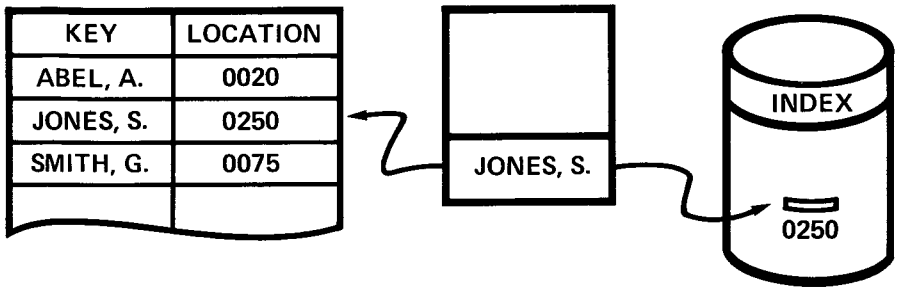

Figure 5

CALCULATION

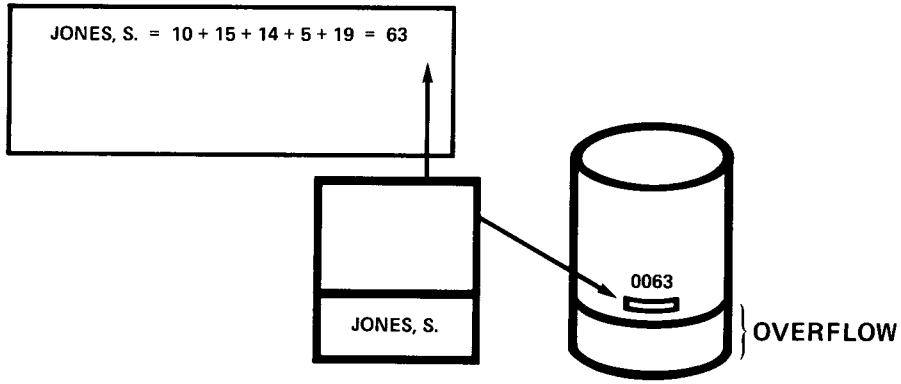

Figure 6

SIMPLE LIST ORGANIZATION

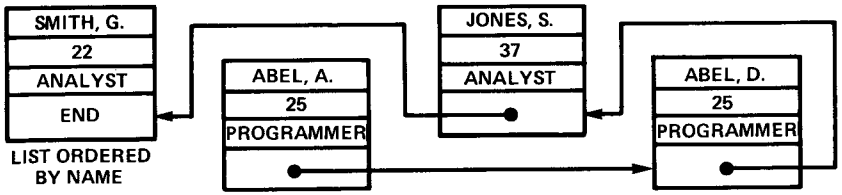

Figure 7

INVERTED TECHNIQUE

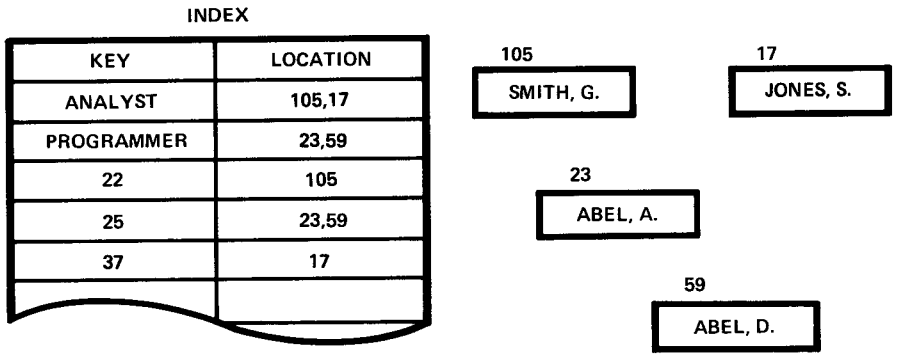

Figure 8

RING TECHNIQUE

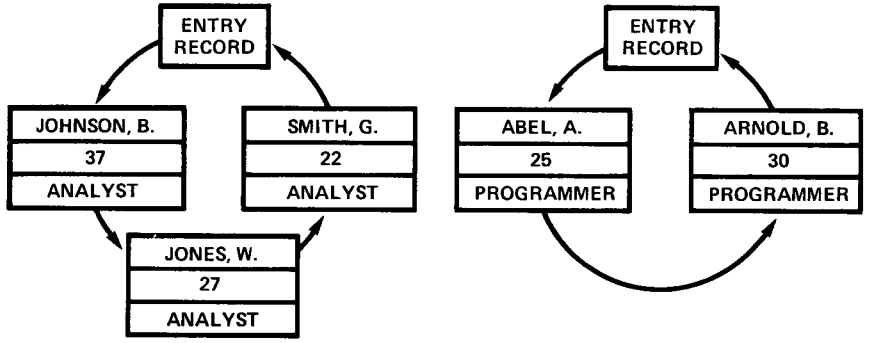

Figure 9

INDEXED SEQUENTIAL

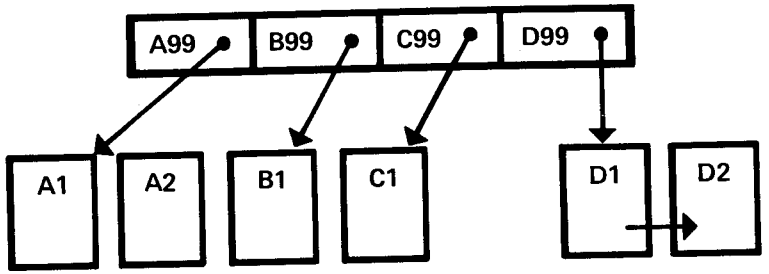

Figure 10

TREE STRUCTURE

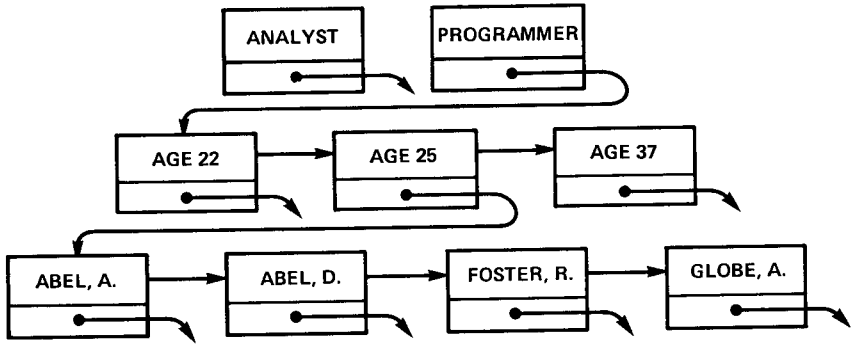

Figure 11

NETWORK STRUCTURE

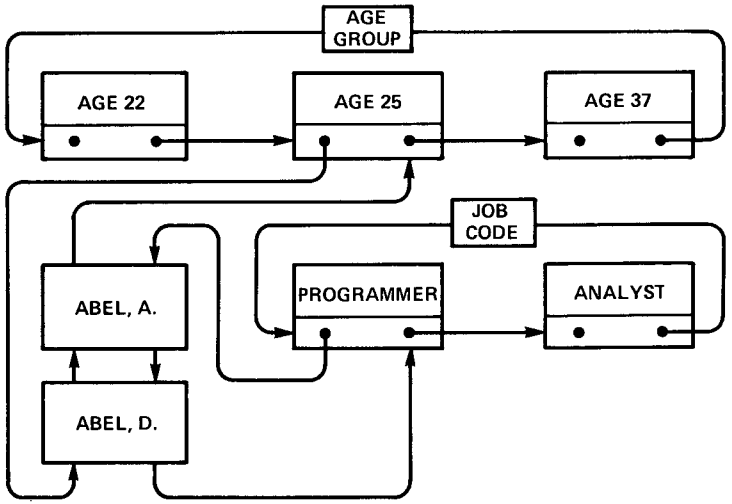

Figure 12